ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE LA
MAÎTRISE EN GÉNIE ÉLECTRIQUE
M.Ing.

PAR
PIERRE-SAMUEL DUBÉ

SYSTÈME DE REPRÉSENTATION D'INTERFACES CENTRÉ SUR L'USAGER

MONTRÉAL, LE 13 DÉCEMBRE 2005

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

M. Éric Fimbel, directeur de mémoire
Département de génie électrique à l'École de technologie supérieure

M. Pierre Bourque, président du jury
Département de génie logiciel et des TI à l'École de technologie supérieure

M. Roger Champagne, membre du jury
Département de génie logiciel et des TI à l'École de technologie supérieure

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 11 NOVEMBRE 2005

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

SYSTÈME DE REPRÉSENTATION D'INTERFACES CENTRÉ SUR L'USAGER

Pierre-Samuel Dubé

SOMMAIRE

Le système *Activity-oriented Interface Representation (AIR)* est un outil d'aide à la conception et à l'évaluation des interfaces usager. Du côté de la conception, il permet de spécifier le détail des interactions entre un usager et une interface à l'aide d'un langage. Pour ce qui est de l'évaluation, un outil ou modèle permet de prédire le temps d'exécution d'un usager hautement familier avec les interfaces graphiques avec une précision d'au plus 26 %, ce qui est acceptable en modélisation analytique. Ce modèle ne tient compte que des opérations explicites de l'usager, soit celles qui on un effet direct sur l'interface. Le modèle fût mis au point et sa précision validée par le biais de tests avec sujets humains dans des conditions contrôlées. Le système AIR comprend des outils logiciels pour faciliter l'utilisation du langage et du modèle.

SYSTÈME DE REPRÉSENTATION D'INTERFACES CENTRÉ SUR L'USAGER

Pierre-Samuel Dubé

SOMMAIRE

Le présent projet de recherche introduit le système *Activity-oriented Interface Representation (AIR)* qui sert à spécifier les interactions entre un usager et une interface. Ce système comporte deux volets. Dans un premier temps, on a développé le langage AIR permettant de spécifier le déroulement des tâches au niveau de détail. Chaque tâche est décrite par un script dont chaque étape se compose des éléments suivants: 1) Action de l'usager. 2) État interne du système. 3) Rétroaction de l'interface. 4) Durée. Chaque étape d'un script peut être spécifiée à différents niveaux allant d'une description abstraite (ex. déplacer une icône, ouvrir une application) à une description au niveau des opérations élémentaires (ex. frappe d'une touche du clavier, mouvement de pointage). On a également développé un outil logiciel qui permet d'écrire des scripts sous forme de fichiers texte ou HTML (*« Hypertext Markup Language »*), et de vérifier leur syntaxe.

Dans un second temps, on a réalisé un outil permettant de prédire les temps d'exécution des usagers (réalisant des tâches sur interfaces graphiques) à partir des scripts écrits en langage AIR. La prédiction s'effectue en comptant les *opérations explicites*, c'est-à-dire, les actions qui agissent sur l'interface par opposition aux *opérations implicites* comme les recherches visuelles ou les activités cognitives. Ces opérations sont les suivantes : 1) Frappe d'une touche du clavier (K). 2) Mouvement de pointage (P). 3) Clic d'un bouton de la souris (C). Dans un premier temps, on transforme le script AIR en une séquence d'opérations explicites, sans tenir compte des changements d'état interne ou des rétroactions de l'interface. Une telle séquence est appelée un modèle KPC. Ensuite, on multiplie le nombre d'occurrences de chaque opération dans la séquence par un estimé de son temps moyen d'exécution. Le modèle KPC assume que les usagers sont *habiles*, c'est à dire hautement familiers avec les interfaces graphiques. C'est le cas de la plupart des usagers actuels d'ordinateurs. Notre hypothèse est que même s'ils ne connaissent pas une interface donnée (état naïf), les usagers habiles apprendront rapidement à l'utiliser grâce aux automatismes acquis précédemment.

Le modèle KPC a été mis au point et validé par le biais d'une étude expérimentale impliquant 20 sujets habiles avec les interfaces graphiques. Avec les dix premiers sujets, on a estimé les temps moyens d'exécution des opérations K, P et C. Avec les dix autres sujets, on a vérifié la précision des prédictions. Selon les résultats obtenus, il est possible de prédire le temps d'exécution d'un usager habile, qu'il soit naïf ou entraîné, à partir du modèle KPC. L'erreur de prédiction pour des usagers habiles et naïfs ne dépasse pas 26 %, ce qui est acceptable selon les standards en modélisation analytique. De plus, les

résultats montrent que les usagers habiles (naïfs ou entraînés) exécutent une même tâche en utilisant plusieurs stratégies efficientes. En d'autre termes, contrairement à ce qui est couramment admis, il n'existe pas une façon optimale de réaliser une tâche. Nous proposons que, dans le futur, il sera possible d'enregistrer les séquences réalisées par des usagers habiles pour construire des scripts AIR de façon semi-automatique.

ACTIVITY-ORIENTED INTERFACE REPRESENTATION SYSTEM

Pierre-Samuel Dubé

ABSTRACT

This research project introduces the *Activity-oriented Interface Representation (AIR)* system which aims at specifying interactions between a user and an interface. This system covers two points. First of all, we developed the AIR language allowing task execution specification at a detailed level. In order to describe a given task, a script has to be written with this language. For each step of a script, the description includes four elements: 1) User action. 2) System internal state. 3) Interface feedback. 4) Duration. Each step can be specified at different levels from an abstract description (e.g. move an icon, open an application) to a description at the elementary operation level (e.g. typing of a keyboard key, pointing movement). During the course of the project, we also developed a software tool to write AIR scripts in text and HTML (*« Hypertext Markup Language »*) formats and to verify their respective syntax.

Secondly, we developed a tool to predict user execution time (for tasks on graphical interfaces) from AIR scripts. The prediction is done on three overt operations: 1) Typing of a keyboard key (K). 2) Pointing movement (P). 3) Mouse button click (C). We first transform an AIR script into a sequence of overt operations without considering system state change and interface feedback. Such a sequence is called a KPC model. For each operation, we then multiply its number of occurrences in the sequence by an estimate of its mean execution time. Thus, we are able to predict time taken by a user in his execution of an AIR script. The KPC model assumes that users are *skilled*. Skilled users are very familiar with graphical interface usage which is the case for most computer users these days. Our hypothesis states that even when users are not really familiar (novice state) with a given interface, they will learn fast how to use it because of the automatisms that they developed with time. The KPC model was tuned and validated via an experimental study on 20 skilled subjects. The first half of the subjects was used to determine the estimates of the K, P and C mean execution times. The other half helped in determining the prediction accuracy. From the obtained results, we can affirm that it is possible to predict skilled user (novice or trained) execution time from a KPC model. For skilled novice users, we obtained an error in prediction not exceeding 26 % which is acceptable according to the standards in analytical modeling. According to the same results, it has been shown that skilled users, novice or trained, can execute a task using multiple efficient strategies. In other terms, there is no optimal strategy in executing a given task. We propose that in the near future, it will be possible to record sequences actually executed by skilled users in order to build AIR scripts in a semi-automatic fashion.

REMERCIEMENTS

Je souhaite tout d'abord remercier l'École de technologie supérieure (ÉTS) pour avoir subventionné le présent projet de recherche via le Programme de Support Institutionnel à la Recherche et à l'Enseignement (PSIRE). Je veux également exprimer ma reconnaissance scientifique au professeur Éric Fimbel du Département de génie électrique de l'ÉTS pour avoir supervisé l'exécution des travaux de recherche ainsi que la rédaction de ce mémoire. Son expertise scientifique très vaste fût une source de motivation constante. Finalement, je désire remercier mes parents pour leur soutien moral et financier tout au long de mes études graduées ainsi que leurs conseils précieux quant au déroulement de ma carrière professionnelle. Un gros merci également à ma copine Mireille pour son soutien au quotidien depuis quelques années déjà.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ABRÉVIATIONS ET SIGLES

ACT-R	Adaptive Control of Thought
AIR	Activity-oriented Interface Representation
ANOVA	Analysis of variance
ASCII	American Standard Code for Information Interchange
CATHCI	Cognitive Analysis Tool for Human Computer Interfaces
CCT	Cognitive Complexity Theory
c-dKPC	context-dependent KPC
CMN-GOMS	Card, Moran, Newell GOMS
COGNET	COGnition as a NETwork of tasks
CPM-GOMS	Cognitive-Perceptual-Motor GOMS
CRITIQUE	Convenient, Rapid, Interactive Tool for Integrating Quick Usability Evaluations
CTA	Cognitive Task Analysis
EPIC	Executive-Process Interactive Control
ÉTS	École de technologie supérieure
GOMS	Goals, Operators, Methods and Selection rules
GUI	Graphical User Interface
HOS	Human Operator Simulator
HTML	Hypertext Markup Language
ICS	Interactive Cognitive Subsystems
ISO	International Organization for Standardization
IJHCS	International Journal of Human-Computer Studies
KLM	Keystroke-Level Model
LCD	Liquid crystal display
MAC	Macintosh
MHP	Model Human Processor
MS	Microsoft

MUMMS	Measuring Usability of MultiMedia Systems
NGOMSL	Natural GOMS Language
PC	Personal Computer
PSIRE	Programme de Support Institutionnel à la Recherche et à l'Enseignement
PUM	Programmable User Model
QGOMS	Quick and dirty GOMS
RAM	Random access memory
SUMI	Software Usability Measurement Inventory
SUN	Sun Microsystems
TE	Temps d'exécution
UAN	User Action Notation
UIDE	User Interface Development Environment
URL	Universal (uniform) Resource Locator
USAGE	UIDE System for semi-Automated GOMS Evaluation
WAMMI	Website Analysis and MeasureMent Inventory
Web	World wide web
WIMP	Windows, Icons, Menus and Pointers
XUAN	Extended UAN
μ	Moyenne d'un échantillon
σ	Écart-type d'un échantillon
n	Nombre d'observations
F(a,b)	Distribution de Fisher-Snedecor d'une analyse ANOVA
p	Coefficient de confiance d'une analyse ANOVA
Δ	Différence entre deux quantités

INTRODUCTION

Il existe différentes méthodes pour établir les spécifications d'une interface (Carr, 1996) et différents modèles pour évaluer son *usabilité* (Ivory & Hearst, 2001). Le terme usabilité est un néologisme qui sera employé ici comme équivalent du terme anglais *« usability »*. Ce terme s'apparent à usager, usage et il a des connotations d'habitude, de coutume et de pratique qui sont absentes des termes utilisabilité (traduction officielle du terme *« usability »*), utilisateur et utilisation. L'usabilité est définie (ISO 9241-11) comme étant le degré selon lequel un produit peut être utilisé par des usagers identifiés, pour atteindre des buts définis avec efficacité, efficience et satisfaction, dans un contexte d'utilisation spécifié. Une application logicielle est efficace si elle permet à l'usager de réaliser une tâche donnée. Elle est efficiente si elle lui permet de réaliser une tâche avec un minimum de ressources (ex. temps d'exécution, effort requis). La satisfaction caractérise le confort et l'acceptabilité fournis à un usager dans son usage d'une application (ex. degré de satisfaction par rapport à la rétroaction observée).

La norme ISO 9126, quant à elle, définit la facilité d'utilisation *(« usability »)* d'un produit logiciel comme étant un ensemble d'attributs portant sur l'effort nécessaire pour l'utilisation et sur l'évaluation individuelle de cette utilisation par un ensemble défini ou implicite d'utilisateurs. Cependant, cette norme ne considère pas l'efficacité et le rendement (efficience) comme étant des composants de la facilité d'emploi.

Peu de systèmes permettent actuellement de combiner la spécification et l'évaluation (Hudson, John, Knudsen & Byrne, 1999, John, Prevas, Salvucci & Koedinger, 2004). Le but du présent travail consiste à concevoir un système de représentation permettant de: 1) spécifier les interactions entre l'usager et l'interface pour une tâche donnée et de 2) prédire le temps d'exécution.

Dans l'optique de spécification, les objectifs sont les suivants :

1) Concevoir un langage permettant la spécification des interactions entre l'usager et l'interface au niveau de détail, et ce de façon intuitive.

2) Développer un outil permettant de vérifier la syntaxe d'une spécification.

En ce qui a trait à la prédiction du temps d'exécution, les objectifs sont les suivants :

1) Mettre au point un modèle prédictif du temps d'exécution d'une tâche quelconque exécutée par un usager entraîné sur une interface graphique *(Graphical User Interface, GUI)*.

2) Développer un outil permettant de prédire le temps d'exécution des usagers à partir de la spécification des interactions, et ce dès les premières étapes du développement de l'interface.

Le système *Activity-oriented Interface Representation (AIR)* a été conçu dans le but de rencontrer ces objectifs. Son développement initial a été subventionné par le Programme de Support Institutionnel à la Recherche et à l'Enseignement (PSIRE) de l'École de technologie supérieure (ÉTS). Le système AIR comprend un langage de spécification, un modèle de prédiction du temps d'exécution et des outils logiciels. Le langage permet de spécifier le déroulement des tâches au niveau de détail. Chaque tâche est décrite par un script dont chaque étape se compose des éléments suivants : 1) Action de l'usager. 2) État interne du système. 3) Rétroaction de l'interface. 4) Durée. Chaque étape d'un script peut être spécifiée à différents niveaux allant d'une description abstraite (ex. déplacer une icône, ouvrir une application) à une description au niveau des *opérations explicites*. Les opérations explicites ont un effet direct sur l'interface (ex. frappe d'une touche du clavier, mouvement de pointage), par opposition aux *opérations implicites*, comme les recherches visuelles de composants, ou les activités cognitives. Dans le cadre de ce projet, on a également développé un outil logiciel qui permet d'écrire des scripts

sous forme de fichiers texte (ASCII, *« American Standard Code for Information Interchange »*) ou HTML (*« Hypertext Markup Language »*), et de vérifier leur syntaxe. Le langage AIR a été validé par le biais d'une enquête réalisée auprès de 78 étudiants en génie logiciel. Cette dernière a permis de comparer le langage AIR à un autre langage de spécification, soit l'UAN (*User Action Notation*). Les participants ont évalué chacun de ces langages selon trois critères : i) Facilité d'apprentissage. ii) Facilité d'écriture. iii) Facilité de lecture. Cette enquête est présentée aux annexes 9 et 10 de ce document. Selon les résultats, le niveau d'effort requis pour apprendre et utiliser le langage AIR est plus faible que pour l'UAN. Selon ces mêmes résultats, l'AIR est préféré à l'UAN.

Dans le cadre du projet, on a également réalisé un outil permettant de prédire les temps d'exécution des usagers (réalisant des tâches sur interfaces GUI) à partir des scripts écrits en langage AIR. La prédiction s'effectue sur les trois opérations explicites suivantes: 1) Frappe d'une touche du clavier (K). 2) Mouvement de pointage (P). 3) Clic d'un bouton de la souris (C). Dans un premier temps, on transforme le script AIR en une séquence d'opérations explicites, sans tenir compte des changements d'état interne ou des rétroactions de l'interface. Une telle séquence est appelée un modèle KPC. Ensuite, pour chaque opération, on multiplie son nombre d'occurrences dans la séquence par un estimé de son temps moyen d'exécution. Ceci permet d'estimer le temps mis par un usager qui exécute le script AIR.

Le modèle KPC assume que les usagers sont *habiles*, c'est à dire hautement familiers avec les interfaces GUI. C'est le cas de la plupart des usagers actuels d'ordinateurs. Notre hypothèse est que même s'ils ne connaissent pas une interface donnée (état naïf), les usagers habiles apprendront rapidement à l'utiliser grâce aux automatismes acquis. Le modèle KPC a été mis au point et validé par le biais d'une étude expérimentale impliquant 20 sujets habiles avec les interfaces GUI. Avec les dix premiers sujets, on a estimé les temps moyens d'exécution des opérations K, P et C. Avec les dix autres sujets, on a vérifié la précision des prédictions. Un article scientifique sur le modèle KPC a été

soumis en août 2005 au *International Journal of Human-Computer Studies (IJHCS)*. Cet article est fourni à l'annexe 12 de ce document.

Le schéma de la figure 1 illustre les interrelations entre les différents composants du système AIR.

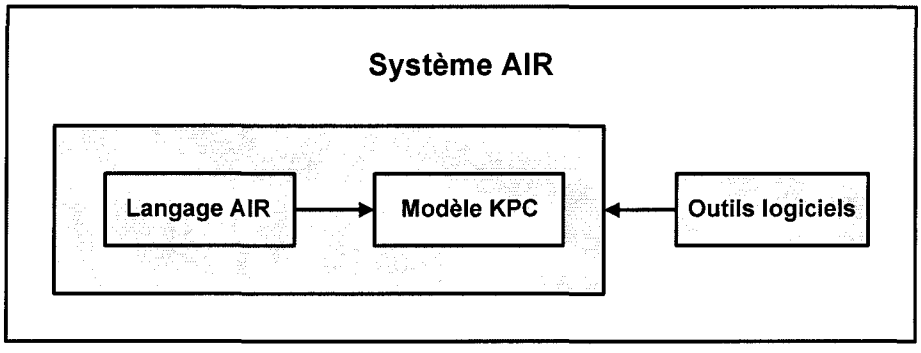


Figure 1 Système AIR

Le système AIR est utilisé pour résoudre des problèmes d'efficacité et d'efficience dans les interfaces GUI. Un exemple d'un tel problème réside dans l'enregistrement d'un usager (*« login »*) dans une fenêtre réservée à cette fin. Il arrive que l'usager tente de confirmer son enregistrement en faisant un retour de chariot dans le dernier champ de texte rempli. Si l'interface ne lui permet pas cela, l'usager devra, par exemple, utiliser la souris pour aller cliquer sur le bouton de confirmation situé plus bas. Il perdra alors du temps sans compter la frustration et l'inconfort causés par cette situation. Le coût cumulatif de ces petits problèmes devient vite considérable lorsque l'interface est utilisée fréquemment. Par exemple, une perte de temps de 5 secondes répétée 4 fois par jour par mille usagers correspond à plus de 1300 heures par an.

Les interfaces Web présentent trop souvent des problèmes d'efficience ponctuels. Par exemple, l'usager doit employer trop d'hyperliens pour atteindre son objectif. Un autre exemple est l'absence d'informations importantes et/ou fréquemment consultées sur la

page d'accueil d'un site Web. Le cas échéant, l'usager aura de la difficulté à obtenir ces informations ou ne les obtiendra jamais.

Le système AIR est utilisé pour aider à minimiser ces problèmes dès la conception. Pour ce faire, on choisira un ensemble de tâches simples et fréquentes. On utilisera le langage AIR afin de spécifier les séquences d'opérations correspondantes. On utilisera ensuite le modèle KPC afin d'obtenir un estimé du temps d'exécution et du nombre d'opérations exécutées pour chacune des séquences. En comparant, pour chaque tâche, les temps obtenus dans différentes implémentations possibles de l'interface, on sera capable de choisir l'interface la plus efficiente ou même d'estimer l'écart d'efficience entre l'implémentation actuelle et une bonne implémentation.

Le corps du présent document comporte six chapitres. Le contenu de chacun d'eux est décrit brièvement ci-dessous.

Chapitre 1 : ce chapitre présente une revue de la littérature portant sur la spécification des interfaces et l'évaluation de l'usabilité. La problématique de la recherche ainsi que les objectifs sont énoncés dans ce chapitre. Une introduction au système AIR est également présentée.

Chapitre 2 : ce chapitre décrit le langage AIR. Les fonctionnalités du langage sont décrites. La structure et la syntaxe (éléments et règles syntaxiques) sont également présentées.

Chapitre 3 : ce chapitre présente les outils logiciels du système AIR. Les fonctionnalités (spécification des interactions, prédiction de temps) sont décrites. La méthodologie de conception utilisée et l'architecture des outils sont également présentées.

Chapitre 4 : ce chapitre présente le modèle KPC. Dans un premier temps, la façon d'utiliser le modèle pour représenter l'exécution d'une tâche et calculer son temps d'exécution est décrite. Ensuite, on explique comment il est possible d'extraire un modèle KPC à partir d'un script écrit en langage AIR. La méthodologie de construction expérimentale du modèle est également incluse.

Chapitre 5 : ce chapitre décrit la méthodologie de mise au point du modèle KPC et la validation expérimentale de ce dernier. Les résultats obtenus sont présentés et discutés.

Chapitre 6 : ce chapitre discute des forces et faiblesses du système AIR et de ses composants. On discute également de sa capacité à être diffusé dans les milieux universitaire et industriel.

CHAPITRE 1

REVUE DE LA LITTÉRATURE

Ce premier chapitre présente un sommaire de ce qui existe dans les domaines de la spécification d'interfaces et l'évaluation de l'usabilité. L'accent est mis principalement sur les interfaces de type *Windows, Icons, Menus and Pointers (WIMP)* qui sont les interfaces concernées par ce projet. Ensuite, la validité de ces différentes approches dans le contexte technologique actuel est discutée. Troisièmement, la problématique et les objectifs spécifiques sont décrits. Enfin, le système AIR est introduit.

1.1 Spécification d'interfaces

La spécification d'interfaces doit prendre en compte simultanément les aspects logiciels et matériels ainsi que l'activité de l'usager. Il est donc nécessaire d'employer des méthodes spécifiques, différentes de celles de la spécification de systèmes logiciels ou matériels. Plusieurs méthodes peuvent être utilisées pour spécifier une interface usager (Carr, 1996). Ces méthodes se divisent en cinq catégories :

1) Spécification algébrique : spécification axiomatique[1] des types de données abstraits. Selon Carr (1996): *« this method permits formally proving properties of the user interface»*. Avec cette méthode, il est difficile de spécifier la nature séquentielle du dialogue entre un usager et une interface. De plus, la notation n'est pas très intuitive (difficile à lire et à écrire).

2) Diagrammes de transition (Shneiderman & Plaisant, 2004) : diagrammes constitués de boîtes et de liens uni- ou bidirectionnels. Les boîtes représentent les états (écrans) de l'interface et les liens représentent les transitions (déclenchées

[1] Système axiomatique: système basé sur un ensemble d'énoncés (axiomes) ainsi que des règles d'inférence afin de dériver des théorèmes.

par des actions de l'usager) entre les états. Les liens peuvent être accompagnés de leur fréquence d'exécution. Ces diagrammes permettent d'avoir une vue d'ensemble de l'interface. Il est également possible de détecter des problèmes quant à l'usabilité globale de l'interface. Cependant, ceci est limité puisque le détail des écrans et des interactions est absent de ces diagrammes.

3) Spécification *« rule-based »* : spécification des interactions qui dépend de conditions spécifiques. Les actions de l'usager dépendent de conditions prédéterminées tandis que leur exécution impose des conditions a posteriori (Rosenbloom, Laird, Newell & McCarl, 1991, Laird & Rosenbloom, 1996). Cependant, cette approche est exigeante puisque qu'elle dépend d'une analyse exhaustive de l'usager qui n'est peut-être pas nécessaire à la conception.

4) Spécification par démonstration (Hudson et al., 1999, John et al., 2004) : avec cette approche, le concepteur ou l'analyste effectue des actions sur l'interface (prototype) et démontre les changements d'état et la rétroaction de l'interface. Cependant, certaines séquences d'opérations ne peuvent être simulées puisqu'elles sont difficiles à prévoir (beaucoup d'interfaces sont sans contrainte, ex. recherche Web à la place d'utiliser les *hyperliens*).

5) Systèmes de grammaire : systèmes permettant de spécifier les interactions entre un usager et une interface à l'aide d'une grammaire (Moran, 1981, Payne & Green, 1989) ou d'un langage (Hartson, Siochi & Hix 1990; voir aussi Brock, Hix, Dievendorf & Trafton 1995, Hartson & Mayo, 1994, Shneiderman & Plaisant, 2004).

Parmi toutes ces approches, l'UAN est celle qui attire le plus notre attention car elle permet de spécifier facilement les actions, les changements d'état du système et la rétroaction de l'interface. Sa syntaxe permet une spécification à un niveau détaillé afin

de ne pas oublier certains éléments d'interaction qui peuvent échapper au concepteur ou à l'analyste (beaucoup d'interfaces sont sans contrainte).

1.2 UAN

L'UAN (Hartson et al., 1990; voir aussi Brock et al., 1995, Carr, 1996, Hartson & Mayo, 1994, Shneiderman & Plaisant, 2004) permet de spécifier les interactions sous forme d'un tableau à trois colonnes. Elles représentent les actions de l'usager, la rétroaction de l'interface ainsi que l'état interne du système. Parfois, une quatrième colonne est rajoutée pour spécifier la connexion au système à des fins de calcul (c'est-à-dire l'utilisation de fonctionnalités du système). La taille (nombre de lignes) du tableau dépend du nombre d'actions spécifiées.

L'UAN est une notation ouverte : " ... *we encourage extension of UAN symbols, columns, or any other feature, so that UAN technique more completely supports interface representation* " (Hartson et al., 1990, p. 199). Par exemple, pour adapter la notation aux interfaces pour quadriplégiques (ex. Steriadis, 2003), il serait nécessaire d'ajouter de nouveaux symboles afin de représenter le mieux possible le fonctionnement des dispositifs d'interaction associés.

L'UAN permet également ce qui suit :

- Contrôle du flot d'exécution en imposant des conditions d'opération.
- Gestion des relations temporelles (exécution séquentielle, exécution en ordre quelconque, interruption, entrelacement et concurrence).
- Représentation des tâches à plusieurs niveaux d'abstraction : fonctionnalité utile pour la spécification d'une tâche fréquente qui ne change pas d'une exécution à l'autre (ex. copier et coller).

L'*Extended User Action Notation (XUAN)* (Brock et al., 1995) est une variante de l'UAN permettant de spécifier la technique d'interaction utilisée. La technique d'interaction est simplement la façon dont l'usager utilise un dispositif pour interagir avec l'interface (ex. usage de la souris vs raccourcis du clavier). Certains changements ont été apportés à la structure du tableau. Ils ont pris place majoritairement dans la colonne des actions de l'usager divisant cette dernière en sous–colonnes en fonction du nombre de dispositifs. Ainsi, l'XUAN permet de comparer les stratégies employées avec des dispositifs différents.

1.3 Usabilité

Définition

Nous adoptons ici une définition de l'usabilité tirée de la norme ISO 9241-11. Rappelons que nous employons ici ce terme à la place du terme utilisabilité pour ses connotations d'habitude, de coutume et de pratique.

Évaluation de l'usabilité

Selon Ivory & Hearst (2001), l'évaluation de l'usabilité peut s'effectuer de cinq manières différentes : 1) Test d'usabilité. 2) Enquête. 3) Inspection. 4) Modélisation analytique. 5) Simulation. Lors d'un test d'usabilité, un expérimentateur observe un usager dans l'exécution de ses tâches afin de détecter des problèmes. Lors d'une enquête, il est demandé au sujet d'émettre ses commentaires (verbalement ou en remplissant un questionnaire) sur l'usage d'une application. L'inspection est effectuée par un évaluateur qui, suivant une liste de contrôles déterminée a priori, inspectera l'application afin d'y détecter des problèmes d'usabilité. Dans le cadre d'une modélisation analytique, un évaluateur utilise un modèle de l'usager et/ou de l'interface afin de produire des données prédictives relatives à l'efficience, notamment le temps

d'exécution. En simulation, des modèles de l'usager et de l'interface sont utilisés afin de produire un comportement interactif et simuler des résultats provenant de ces interactions. La modélisation analytique et la simulation sont inspirées des techniques utilisées afin d'évaluer les performances des systèmes informatiques (Ivory & Hearst, 2001).

Automatisation de l'évaluation de l'usabilité

Pour chacune des cinq techniques énoncées précédemment, une ou plusieurs parmi les activités suivantes peuvent êtres automatisées : 1) La saisie des interactions. 2) L'analyse (Hammontre, Hendrickson & Hensley, 1992, Ivory & Hearst, 2001). 3) La critique (Ivory & Hearst, 2001). Dans la première activité, les événements déclenchés par les interactions entre l'usager et le système sont enregistrés dans un fichier journal (journal d'événements) appelé *« log file »*. L'enregistrement est effectué par un programme renifleur appelé *« sniffer program »* (ou simplement *« sniffer »*). L'analyse a pour but de détecter des problèmes d'usabilité de nature quantitative (ex. temps d'exécution, taux d'erreur) et qualitative. Dans l'activité de critique, des améliorations possibles relatives aux problèmes détectés sont suggérées.

1.3.1 Test d'usabilité

Définition et déroulement

Lors d'un test d'usabilité typique (Nilsen, 1993; voir aussi Rubin, 1994, Shneiderman & Plaisant, 2004), un expérimentateur observe un sujet dans l'exécution de ses tâches afin de détecter des problèmes d'efficacité et d'efficience. Les biais possiblement causés par la subjectivité de l'évaluateur sont alors évités. Les problèmes sont donc détectés de façon objective, indépendamment des préférences de l'expérimentateur (ce qui n'est pas le cas des inspections, section 1.3.3). On peut également demander au sujet d'émettre ses

commentaires à voix haute durant le test. Ceci aide l'expérimentateur à détecter des problèmes qui lui échappent lors de l'observation et à recueillir des indices sur la satisfaction du sujet. À la suite du test, une enquête (entrevue ou questionnaire) peut être effectuée dans le but de recueillir d'autres informations (sur le déroulement du test et/ou la satisfaction) de la part du sujet.

Saisie de données

Lors d'un test d'usabilité, la saisie peut être automatisée. Un programme renifleur recueille les interactions entre l'usager et l'interface en temps réel et les enregistre dans un fichier journal. Ce journal contient chaque événement et le temps correspondant. Un enregistrement vidéo en temps réel peut aussi être utilisé. Ceci permettra de visualiser en détail l'activité de l'usager correspondant aux événements enregistrés. Le tableau I présente le contenu potentiel d'un fichier journal pour un déplacement d'une icône à l'écran. Dans cet exemple, on donne une description des événements à des fins de compréhension. Notons cependant que cette descrption est absente des journaux réels ce qui les rend difficiles à interpréter.

Tableau I

Exemple de contenu d'un fichier journal provenant d'un test d'usabilité

Instant (ms)	Événement	Description
1000	MOUSE_MOVED	Place la souris sur l'icône
1245	MOUSE_PRESSED	Appuie sur le bouton de gauche
2000	MOUSE_MOVED	Déplace la souris
2225	MOUSE_RELEASED	Relâche le bouton de gauche

En ce qui a trait aux interfaces Web, les données peuvent être enregistrées dans un journal par un renifleur qui est connecté au serveur Web. Le journal, de type serveur

(« *server log file* », ex. Tec-Ed, 1999), contient les événements relatifs à la navigation (ex. clics de boutons, hyperliens empruntés) et le temps correspondant.

Test à distance (« remote testing »)

Les experts en test d'usabilité ne sont pas nombreux et il n'est pas avantageux pour les développeurs logiciels de se déplacer pour faire évaluer leurs interfaces (Hartson et al., 1996). Dans le cadre d'un test à distance, l'expérimentateur et le sujet ne sont pas colocalisés. Cependant, l'expérimentateur peut observer le déroulement du test par le biais du réseau reliant sa machine et celle du sujet.

Il existe deux catégories de test à distance : 1) En même temps et à des endroits différents. 2) En différé et à des endroits différents. Pour la première, le réseau permet à l'expérimentateur d'observer l'écran du sujet en temps réel. La communication audio s'effectue par téléphone ou microphone. La communication peut également se faire par écrit via le logiciel. Ceci permet au sujet d'émettre des commentaires ou de répondre aux questions de l'expérimentateur.

Pour ce qui est de la deuxième catégorie, l'expérimentateur observe l'écran du sujet en différé. Dans ce type de test, une application guide le sujet et peut même lui permettre d'émettre des commentaires via des boîtes de dialogue. Cette méthode peut être utilisée pour : 1) obtenir la rétroaction du sujet pour une application finie ou en développement (temps d'exécution, taux de réussite des tâches) ou 2) obtenir des données statistiques pour une application finie (ex. fréquence et type d'usage).

Les deux méthodes de test à distance permettent d'élargir l'usage des tests d'usabilité. Cependant, l'expérimentateur peut rencontrer des problèmes techniques au niveau de la machine, de l'application et/ou du réseau.

Peu importe la méthode de saisie de données utilisée, les données brutes sont souvent volumineuses et contiennent beaucoup d'informations qui n'aident pas nécessairement à la spécification des interactions. Ainsi leur interprétation n'est pas évidente. La section qui suit survole les différentes techniques utilisées afin d'analyser ces données.

Analyse de données

À la suite d'un test d'usabilité, il est possible d'analyser les données contenues dans le journal de façon automatique. Selon Ivory & Hearst (2001), cette analyse peut être effectuée de quatre façons différentes: 1) Mesures quantitatives. 2) Reconnaissance de patrons. 3) Analyse de la tâche. 4) Analyse inférentielle. Pour les interfaces WIMP, les mesures quantifient les performances des usagers (ex. temps d'apprentissage, temps d'exécution, taux d'erreurs). Pour les interfaces Web, les mesures quantifient les performances du serveur ou du réseau (ex. temps de réponse) et la navigation du site Web (ex. temps de navigation, hyperliens empruntés). Cependant, elles ne donnent pas beaucoup d'informations quant à l'usabilité du site en soi (Ivory & Hearst, 2001) ni sur les détails de la tâche, notamment le but de l'usager (Byrne, John, Wehrle & Crow, 1999). En fait, les fichiers journaux de type serveur ne capturent pas toujours l'activité réelle de l'usager (Tec-Ed, 1999). Voici quelques exemples : i) L'usager entre une URL directement sans naviguer. ii) Le fureteur est programmé pour visualiser les pages Web à partir de la mémoire cache. iii) Les pages Web ne sont pas rechargées par le serveur.

Dans la reconnaissance de patrons, une association entre les différents événements du journal et la tâche exécutée est effectuée. Les problèmes d'usabilité sont déterminés en identifiant les commandes causant des erreurs qui sont répétitives. Dans l'analyse de la tâche, les problèmes d'usabilité sont identifiés en comparant les résultats obtenus à ceux prédits par un modèle et en identifiant les écarts. Finalement, l'analyse inférentielle utilise des techniques de visualisation et des données statistiques afin d'identifier les problèmes. Elle est utilisée majoritairement avec les interfaces Web.

1.3.2 Enquête

Les commentaires des usagers peuvent être recueillis par enquête soit en les interrogeant en personne ou en leur faisant remplir un questionnaire (papier ou électronique). Des exemples de questionnaires interactifs pour les interfaces WIMP et Web sont *Software Usability Measurement Inventory (SUMI)* (Kirakowski & Corbett, 1993, Van Veenendaal, 1998) et *Website Analysis and MeasureMent Inventory (WAMMI)* (Kirakowski & Claridge, 1998, Gallego, Rodriguez, Salàn & Fernandez, 2002), respectivement. SUMI et WAMMI permettent de récolter de l'information objective sur l'usabilité de l'application et du site Web, respectivement. Dans les deux cas, une série de questions et de choix de réponses préétablis est présentée à l'usager (typiquement, 50 énoncés pour SUMI et 20 énoncés pour WAMMI). Ces questionnaires permettent de produire un rapport sur l'usabilité de l'application. Un questionnaire pour les applications multimédia a également été développé, soit *Measuring Usability of MultiMedia Systems (MUMMS)* (Van Veenendaal, 1998).

Selon Ivory & Hearst (2001), la saisie de données est la seule activité automatisable pour les enquêtes de type questionnaires à remplir. Ils présentent des méthodes visant à automatiser la saisie pour les interfaces WIMP (ex. boîtes de dialogues incluses dans l'interface) et Web (ex. formulaires interactifs sur Internet). Pour les interfaces Web, il est possible de recueillir un grand ensemble de données rapidement grâce au réseau Internet. Dans les deux cas (WIMP et Web), les informations recueillies sont cependant à caractère subjectif (excepté SUMI et WAMMI). De plus, les informations acquises en présence de l'usager (ex. réponses à des questions de l'expérimentateur qui sont de nature objective) sont perdues (Hartson, Castillo, Kelso & Neale, 1996).

1.3.3 Inspection

Lors d'une inspection, un évaluateur compare certains aspects de l'usabilité d'une application à un ensemble de règles de conception préétablies (*« design guidelines »*). Les incohérences entre les observations et les règles représentent des problèmes d'usabilité. Cependant, le niveau d'incohérence dépend de l'évaluateur (Ivory & Hearst, 2001). Des exemples de méthodes d'inspection sont *« heuristic evaluation »* et *« cognitive walkthrough »*.

Puisqu'il n'est pas toujours facile pour les développeurs de respecter les règles de conception, des outils logiciels peuvent être utilisés afin de détecter automatiquement certaines violations. Ces outils aident l'évaluateur en lui spécifiant les violations de certaines règles et lui fournissant même des façons de remédier aux problèmes. La saisie de données, l'analyse de données et la critique peuvent être automatisées dans le but de détecter le plus de violations possibles (Ivory & Hearst, 2001). Parmi ces activités, la critique est celle qui fournit le plus de données permettant de se conformer aux règles d'usabilité.

1.3.4 Modélisation analytique

L'utilisation de modèles de l'usager et/ou de l'interface permet de prédire certains aspects de l'usabilité, notamment le temps d'apprentissage, le temps d'exécution et le taux d'erreurs. Selon Ivory & Hearst (2001), ces modèles se divisent en quatre catégories :

1) Modèles de l'interface usager : ils permettent de représenter le design à plusieurs niveaux d'abstraction. Les notations UAN et XUAN utilisent de tels modèles.

2) Modèles pour analyser l'environnement de la tâche : ils permettent à l'expérimentateur de connaître la façon avec laquelle le sujet utilise l'interface pour atteindre ses objectifs.

3) Modèles cognitifs: ils permettent à l'expérimentateur de représenter les connaissances et les processus cognitifs (ex. raisonnements, appels à la mémoire) d'un usager lors de l'usage d'une interface. Ainsi, l'expérimentateur pourra comparer plusieurs interfaces sur la base de l'apprentissage.

4) Modèles de performances de l'usager (ex. KLM, GOMS; voir plus loin) : ils permettent de prédire le comportement de l'usager, notamment son temps total d'exécution pour une tâche donnée (Ivory & Hearst, 2001). Une fonctionnalité intéressante de ces modèles est qu'ils permettent d'évaluer facilement et rapidement certains aspects de l'usabilité d'une interface avant même qu'elle soit conçue (Young et al., 1989). Typiquement, l'erreur moyenne de précision ne dépasse pas 20 % (Card, Moran & Newell, 1980; 1983; voir aussi Kieras, 1993, Kieras, Wood, Abotel & Hornof, 1995, John & Kieras, 1996a; 1996b, Hudson et al., 1999, Baumeister, John & Byrne, 2000, Ivory & Hearst, 2001, John et al., 2004).

En modélisation analytique, seulement l'analyse de données est automatisée. De plus, il en est ainsi seulement pour les modèles de performance. En fait, la prédiction de performance est basée sur une analyse quantitative qui est facile à automatiser (Ivory & Hearst, 2001). Les autres types de modèles sont basés principalement sur une analyse qualitative de l'usager et/ou de l'interface.

Dans les prochaines sections, on met l'accent sur certains modèles qui, comme le modèle KPC (voir chapitre 4) du système AIR, visent à prédire les performances des usagers. Les modèles présentés sont les suivants :

- *Goals, Operators, Methods and Selection Rules (GOMS)*
- *Keystroke-Level Model (KLM)*
- *Cognitive Task Analysis (CTA)*
- *Programmable User Model (PUM)*

1.3.4.1 GOMS

Les modèles GOMS décrivent l'exécution d'une tâche (à l'aide d'un programme ou d'une séquence d'opérations) comme une hiérarchie de buts (et sous-buts), d'opérateurs simples, de méthodes utilisées afin d'atteindre les buts et de règles de sélection servant à choisir parmi les méthodes (Card et al., 1983). En fait: *«A GOMS model is a description of the knowledge that a user must have in order to carry out tasks on a device or system»* (Kieras et al., 1995). Les modèles GOMS sont basés sur le *Model Human Processor (MHP)* (Card et al., 1983). Ce dernier postule que : 1) l'usager exécute ses tâches comme s'il résolvait un problème, c'est à dire en se déplaçant dans un *espace de problème* (un graphe dont les noeuds sont les situations possibles) jusqu'à ce qu'il atteigne son objectif et 2) l'usager est rationnel, c'est-à-dire qu'il développe des stratégies efficientes afin d'atteindre son but.

Il existe plusieurs formes de GOMS. Selon John & Kieras (1996b), les plus répandues sont les suivantes :

1) *Card, Moran, Newell GOMS (CMN-GOMS)* (Card et al., 1983) : ce modèle est la forme d'origine. Il se construit sous forme d'un programme informel constitué de méthodes et de conditions. En utilisant ce modèle, il est possible de prédire la séquence d'opérations ainsi que le temps d'exécution.

2) *Natural GOMS Language (NGOMSL)* (Kieras, 1988, 1996) : ce modèle permet de représenter un modèle GOMS en langage naturel en décomposant les buts en

méthodes puis en opérations élémentaires. Il se construit sous forme d'un programme. En utilisant ce modèle, il est possible de prédire la séquence d'opérations ainsi que les temps d'apprentissage (des méthodes) et d'exécution.

3) *Cognitive-Perceptual-Motor GOMS (CPM-GOMS)* (John, 1990) : ce modèle considère les relations temporelles entre les différents processeurs (cognitifs, perceptuels et moteurs) du MHP et les opérations exécutées par ces derniers afin de prédire le temps d'exécution. Les opérations ne sont pas considérées comme étant exécutées de façon séquentielle. Ainsi, la séquence ne peut être prédite.

Malgré les résultats obtenus, l'utilisation de modèles GOMS dans l'industrie reste marginale, probablement à cause de la difficulté à construire ces modèles. En effet, comme la structure des tâches doit être complètement définie, la construction de modèles GOMS peut être exigeante lorsque l'interface sous évaluation comporte plusieurs tâches complexes (John & Kieras, 1996a, Ivory & Hearst, 2001).

Il existe différents outils pour faciliter la construction de modèles GOMS (Ivory & Hearst, 2001; voir aussi Baumeister, John & Byrne, 2000). Voici quelques uns d'entre eux :

1) *Cognitive Analysis Tool for Human Computer Interfaces (CATHCI)* (Williams, 1993) : CATHCI est un outil de type *« wizard »* qui guide le concepteur. Ce dernier construit un GOMS en créant des diagrammes de boîtes (*« block diagrams »*). Les buts et opérateurs sont identifiés par des boîtes de couleurs différentes. Pour spécifier les méthodes possibles pour un but, on doit cliquer sur la boîte associée au but. Les méthodes sont alors spécifiées dans une autre fenêtre par une séquence de boîtes (une séquence par méthode). Lorsqu'une condition est remplie, l'usager doit cliquer un bouton (ex. si la main n'est pas sur le clavier, alors mettre la main sur le clavier (bouton cliquable); la condition et le bouton sont dans une même boîte). Le temps

d'exécution est obtenu en exécutant le modèle. Le temps va dépendre des réponses aux différentes conditions.

2) *UIDE System for semi-Automated GOMS Evaluation (USAGE)* (Byrne, Wood, Sukaviriya, Foley & Kieras, 1994) : à l'aide d'un UIDE (*User Interface Development Environment*), l'analyste construit des modèles en spécifiant les tâches à trois niveaux : 1) Application (ex. créer un fichier). 2) Interface (ex. sélectionner un menu). 3) Interactions (ex. bouger la souris et cliquer le bouton de gauche).

   L'UIDE reçoit en entrée les modèles et produit pour chaque action de l'application une séquence hiérarchique d'actions de l'interface. La séquence est ensuite traduite à l'aide d'un outil traducteur en un programme NGOMSL. Le temps d'apprentissage peut être obtenu directement à partir de ce programme. Un outil d'interprétation reçoit en entrée le programme NGOMSL et une description des tâches à effectuer. Il fournit ensuite un estimé du temps d'exécution pour chaque tâche.

3) GLEAN (Kieras et al., 1995) : le concepteur construit un GOMS en écrivant un programme avec le langage NGOMSL. L'outil exécute ensuite le programme (processus dynamique) et fournit deux choses dans un fichier texte : 1) Séquence d'opérations. 2) Temps d'exécution. Un processus statique permet de fournir le temps d'apprentissage (à l'écran ou dans un fichier) en comptant le nombre d'énoncés du programme nécessitant un apprentissage.

4) *Quick and dirty GOMS (QGOMS)* (Beard, Smith & Denelsbeck, 1996) : via l'interface (une seule fenêtre), le concepteur (ou analyste) construit un GOMS en créant des arbres graphiques. Dans un arbre, les buts sont représentés par les nœuds et les opérateurs par les feuilles. Un arbre représente une méthode. Le concepteur associe une probabilité d'exécution à chaque arbre afin d'éviter l'utilisation de règles de sélection. Un temps d'apprentissage fixe est associé à chaque nœud. Le temps

d'apprentissage total dépend du nombre de nœuds. Le temps d'exécution varie en fonction du temps de chaque nœud (spécifié par l'analyste) et des probabilités des arbres. Ces temps (apprentissage et exécution) sont calculés de façon automatique.

Le niveau d'automatisation d'USAGE est supérieur à celui de CATHCI, GLEAN et QGOMS car il effectue des traitements sur la séquence qui est fournie en entrée. En automatisant la création de modèles qui sont fournis à l'UIDE, USAGE deviendrait un outil pleinement automatique.

Ces outils réduisent le niveau d'effort requis lors la création de GOMS. Cependant, la construction (en partie ou complète) du modèle n'est pas évitée. Selon Baumeister, John & Byrne (2000), l'outil idéal pour la construction de GOMS devrait posséder des fonctionnalités de prototypage d'interfaces.

1.3.4.2 KLM

Le *Keystroke-Level Model (KLM)* (Card et al., 1980) est l'un des modèles les plus simples. Il est souvent présenté comme une variante des modèles GOMS (Card et al., 1983). Cependant, d'un point de vue chronologique, il serait plutôt leur ancêtre. Le KLM se construit sous forme d'une séquence d'opérations, et ne nécessite ni méthode ni règle de sélection. La séquence est constituée d'opérateurs au niveau *keystroke* (ex: entrée au clavier, mouvement de souris, clic de souris, etc.). Le KLM permet d'insérer des opérations cachées ou implicites dans la séquence. Une opération cachée ou implicite est une action n'ayant aucun effet direct sur l'interface (ex: activité cognitive, déplacement du bras entre le clavier et la souris). L'insertion s'effectue en utilisant des règles heuristiques (ex. insérer une activité mentale avant chaque opération nécessitant un choix ou ayant un effet sur l'interface). Dans la forme d'origine (Card et al., 1980) du KLM, une séquence est une combinaison des opérations suivantes :

- K : entrée au clavier
- P : mouvement de souris
- D(n,l) : dessin; n = nombre de segments, l = longueur totale des segments
- M : activité mentale
- H : « *homing* » (déplacement du bras dominant du clavier vers la souris et vice et versa ou du clavier principal vers le clavier numérique et vice et versa)
- R(t) : temps de réponse du système

Des estimés des temps moyens pour ces opérations ont été déterminés de façon empirique dans l'étude de Card et al. (1980). Le temps moyen total d'exécution d'une séquence est calculé en utilisant la formule (1.1). Dans cette formule, $\hat{E}_{séquence}$ = temps total estimé, N_X = nombre d'opérations de type X dans la séquence et $\hat{E}_X$ = temps estimé d'une opération de type X.

$$\hat{E}_{séquence} = N_K * \hat{E}_K + N_P * \hat{E}_P + N_D * \hat{E}_D + N_M * \hat{E}_M + N_H * \hat{E}_H + N_R * \hat{E}_R \quad (1.1)$$

Dans sa version originale, le KLM est limité aux usagers experts (connaissant intimement un système et l'utilisant fréquemment) exécutant les tâches de façon optimale (en terme du temps d'exécution). Il a aussi été proposé d'utiliser le KLM pour modéliser des *séquences significatives*, qu'elles soient optimales ou simplement fréquemment utilisées (Kieras, 1993).

Convenient, Rapid, Interactive Tool for Integrating Quick Usability Evaluations (CRITIQUE, Hudson et al., 1999) est un environnement de modélisation permettant de créer des modèles de performances à partir de démonstrations sur prototypes d'interfaces. Par exemple, il permet de construire automatiquement un KLM (prédire la séquence d'opérations) et de calculer (prédire) le temps d'exécution.

CRITIQUE est accompagné d'un UIDE et d'un outil de développement de prototypes d'interfaces. Dans un premier temps, le prototype de l'interface est développé. Ensuite, la démonstration est effectuée par l'analyste (ou concepteur) en interagissant avec l'interface (exécution de la tâche). En d'autres mots, la séquence d'opérations est fournie par l'analyste. Les données d'utilisation sont enregistrées dans un journal durant l'exécution de la tâche. Les événements enregistrés sont ensuite traités automatiquement dans le but de créer le modèle associé à la tâche. Ainsi, la séquence d'opérations et le temps d'exécution sont prédits. Ces prédictions sont possibles pour tout prototype développé avec l'outil.

Hudson et al. (1999) ont comparé les prédictions obtenues avec un KLM construit à la main et celui construit automatiquement par CRITIQUE pour deux tâches de complexités différentes. Ils reportent une différence inférieure ou égale à 2 % dans les deux cas. L'utilisation de CRITIQUE pour la création d'autres modèles GOMS est moins évidente, vu la nécessité de créer automatiquement une hiérarchie de buts, de méthodes et de règles de sélection. En fait, Hudson spécifie: « *Producing good goal hierarchies in a fully automatic fashion will be difficult. As a result, we will also consider semi-automatic approaches which automate some parts of the task and facilitate human performance of the remainder* » (Hudson et al., 1999, p. 101).

1.3.4.3 CTA et PUM

Les modèles CTA et PUM considèrent les activités cognitives de l'usager. Le modèle CTA permet d'effectuer un design qui permettra à l'usager de focaliser son attention sur la tâche qu'il a à faire et non l'usage de l'interface. Pour ce faire, le concepteur étudie le système du point de vue de l'usager et identifie les aspects du design de l'interface qui lui demandent beaucoup de ressources cognitives (mémoire, attention, etc.) (May & Barnard, 1995). Le modèle PUM permet au concepteur de programmer certains aspects

du comportement de l'usager avant d'effectuer la prédiction de temps (Young et al., 1989).

1.3.5 Simulation

Contrairement aux approches de modélisation analytique vues à la section 1.3.4, la simulation permet, en plus de prédire les temps, de détecter des problèmes d'usabilité (erreurs, échecs, etc.) à partir de journaux d'événements produits artificiellement. Il s'agit de la combinaison des deux activités suivantes qui peuvent toutefois être effectuées séparément :

1) Simulation des journaux d'événements : à partir d'un modèle de l'usager, les données d'utilisation sont générées automatiquement. Il s'agit d'un processus plus empirique que les méthodes de modélisation analytique. Cependant, les données sont obtenues beaucoup plus rapidement et facilement que lorsque des tests avec sujets humains sont effectués.

2) Simulation des interactions : des modèles de l'usager et/ou de l'interface sont utilisés afin de simuler les interactions entre un usager et une interface et d'en obtenir les prédictions (ex. séquences d'opérations, mesures de performances, etc.).

Simulation des données d'utilisation

À l'aide de paramètres d'entrée, les données d'utilisation sont générées automatiquement et enregistrées dans un journal. Il s'agit d'une façon rapide de recueillir un grand nombre de données sans nécessiter un usage de l'interface. Ivory & Hearst (2001) ont cité deux applications permettant cela; une pour les interfaces WIMP et l'autre pour les interfaces Web.

La première (Kasik & George, 1996), permet d'utiliser les données (exécution sur une interface WIMP) d'un usager expert afin de produire celles d'un usager ayant un niveau d'entraînement différent (ex. usager naïf). Les interactions entre un usager expert et l'interface sont spécifiées dans un script qui est ensuite exécuté (résultat = données d'utilisation de l'usager expert). En altérant le contenu des données de l'usager expert (ex. remplacer l'usage de la touche TAB par celui de la souris pour changer de champ de texte) et en utilisant un algorithme génétique pour émuler l'apprentissage, on peut obtenir les données pour un autre usager (avec un niveau d'expertise moindre).

La deuxième application (Chi, Pirolli & Pitkow, 2000), permet de produire des journaux contenant les chemins de navigation empruntés (ou hyperliens utilisés) dans les interfaces Web. Cette application requiert un effort de modélisation (modèle de l'interface) de la part de l'analyste.

Simulation des interactions

La seule méthode permettant de simuler les interactions directement à partir des données d'utilisation est AMME (Rauterberg & Aeppli, 1995). Elle permet de simuler l'apprentissage, la prise de décision, les erreurs et mesure même la complexité du comportement de l'usager. Un désavantage d'AMME est qu'elle ne prend pas comme entrée des données d'utilisation générées artificiellement.

De façon similaire au MHP (pas nécessairement comme le MHP), plusieurs méthodes de simulation pour interfaces WIMP sont basées sur des architectures cognitives complexes et des modèles du comportement de l'usager. Leur but est de simuler de la façon la plus réaliste possible le comportement de l'usager (Ivory & Hearst, 2001). De plus, certaines de ces méthodes fournissent des prédictions de la même façon que les modèles prédictifs (statistiquement, sans simulation). Ces méthodes de simulation opèrent à plusieurs niveaux :

- Modélisation de la tâche : les méthodes *Adaptive Control of Thought (ACT-R)* (Anderson 1990,1993), *COGnition as a NETwork of tasks (COGNET)* (Zachary, Mentec & Ryder, 1996), *Interactive Cognitive Subsystems (ICS)* (Barnard, 1987, Barnard & Teasdale, 1991) et *Soar* (Polk & Rosenbloom, 1994, Laird & Rosenbloom, 1996) permettent de simuler les activités cognitives d'un usager. Avec *Human Operator Simulator (HOS)* (Glenn, Schwartz & Ross, 1992), on peut simuler l'exécution d'une tâche avec un système interactif. Les méthodes *Cognitive Complexity Theory (CCT)* (Kieras & Polson, 1985), *Executive-Process Interactive Control (EPIC)* (Kieras, Wood & Meyer, 1997) et GLEAN (Kieras et al., 1995) permettent de simuler une tâche exécutée sur une interface usager.

- Modélisation de composants : les méthodes ACT-R et COGNET simulent seulement les activités cognitives de l'usager tandis que CCT, EPIC, GLEAN, HOS, ICS et Soar permettent de simuler les activités cognitives, motrices et perceptuelles.

- Traitement de composants : ACT-R, CCT et GLEAN sont des méthodes qui modélisent l'exécution d'une tâche comme un processus séquentiel. Les méthodes ICS, EPIC et Soar modélisent la tâche comme un processus parallèle tandis que COGNET et HOS la considèrent comme un processus semi-parallèle.

- Représentation du modèle : les modélisations de l'usager et du système s'effectuent en utilisant soit une hiérarchie de tâches (CCT et HOS), des règles de production (ACT-R, CCT, EPIC, ICS et Soar) ou un programme (COGNET et GLEAN).

- Prédictions : les méthodes ACT-R, CCT, COGNET, EPIC, GLEAN, HOS et Soar fournissent des prédictions de performances. CCT et ICS prédisent la quantité d'information que l'usager a dans sa mémoire à court terme. Des

mesures prédictives sur l'apprentissage sont fournies par les méthodes ACT-R, CCT, GLEAN, ICS et Soar. Finalement, ACT-R, COGNET et EPIC permettent de prédire le comportement de l'usager.

Selon Ivory & Hearst (2001), une méthode efficace pour identifier des problèmes d'usabilité devrait permettre : 1) la modélisation d'une interface usager, 2) la modélisation des activités cognitives, motrices et perceptuelles en parallèle, 3) l'utilisation de règles de production et 4) la génération de prédictions (performances, quantité d'information en mémoire, apprentissage et comportement). Parmi les méthodes présentées précédemment (ACT-R, CCT, COGNET, EPIC, GLEAN, HOS, ICS et Soar), EPIC est celle qui possède le plus de caractéristiques parmi ces dernières. Cependant, EPIC ne permet pas de prédire la quantité d'information en mémoire ni l'apprentissage.

1.4 Validité des différentes approches dans le contexte technologique actuel

Outils de spécification

Les langages de spécification sont utilisés depuis longtemps déjà. Cependant, beaucoup d'entre eux n'ont pas survécu, c'est-à-dire qu'ils n'ont jamais excédé une diffusion marginale, contrairement aux environnements de développement moins sophistiqués. Selon Myers et al. (2000), cet échec pourrait être dû, entre autres, aux facteurs suivants :

- Le développeur est confronté à de nouveaux langages (programmation, modélisation) difficiles à apprendre et à utiliser.

- L'utilisation de certains outils oblige le développeur à penser différemment de ce qu'il fait d'habitude.

- Le degré d'effort requis pour apprendre et utiliser un outil est trop élevé (*« high threshold »*) pour des résultats trop peu élevés (*« low ceiling »*).

- L'outil ne guide pas le développeur vers ce qu'il doit faire et ne l'éloigne pas de ce qu'il ne doit pas faire.

- L'outil utilise des techniques automatiques et produit des résultats imprévisibles.

- Dû à l'évolution rapide du développement et des techniques d'interfaces, l'outil devient désuet avant même que les développeurs puissent l'utiliser convenablement.

Outils de modélisation

Les facteurs précédents pourraient également être une des causes de la faible diffusion de certains outils de modélisation (voir Ivory & Hearst, 2001). De plus, une autre limitation de ces derniers serait leur faible degré d'automatisation. En fait, seulement ceux réduisant leur représentation de la tâche au niveau *keystroke* et leur utilisation à une catégorie d'usagers spécifique (ex. usagers experts) sont complètement automatisables (ex. KLM). Les modèles plus complexes sont difficiles à construire et ne sont automatisés que partiellement.

Les nouveaux usagers

Une problématique supplémentaire réside dans le fait que plusieurs de ces outils de modélisation sont anciens. Même si les interfaces graphiques n'ont pas connu d'évolution majeure depuis au moins une décennie, leurs usagers, quant à eux, ont probablement changé. Autrement dit, leur population d'usagers s'est agrandie (de plus en plus de gens sont initiés aux ordinateurs tôt dans leur vie) et diversifiée (les ordinateurs ne sont plus utilisés que par les informaticiens).

En fait, les ordinateurs (ex. PC (« *Personal Computer* »), MAC (« *Macintosh* »), SUN (« *Sun Microsystems* »)) sont devenus avec les années une partie intégrante de notre environnement (travail ou loisirs). Le fonctionnement et l'apparence (« *look-and-feel* ») des interfaces restent pratiquement les mêmes d'un ordinateur à un autre et même d'une application à une autre. Ainsi, il semble que les usagers aient développé des automatismes (ex. Shiffrin & Schneider, 1977) dus à un usage répété d'interfaces similaires. De nos jours, même un usager non familier (naïf) avec une application peut être considéré comme étant habile avec son interface.

On peut donc se demander dans quelle mesure les vieux outils de modélisation tirent adéquatement partie des caractéristiques (ex. usage d'automatismes, apprentissage plus rapide) de ces nouveaux usagers.

1.5 Problématique et objectifs spécifiques

Problématique

De ce qui précède, il appert que l'application des outils de spécification et de modélisation se trouve restreinte par des coûts d'apprentissage et d'utilisation qui sont élevés par rapport aux bénéfices attendus. Les problèmes abordés dans le présent projet sont donc les suivants :

- On ne dispose pas d'un outil automatique permettant de spécifier les tâches au niveau de détail qui soit intuitif. Le survol des méthodes de spécification effectuée par Carr (1996) révèle que ces dernières sont anciennes (ex. grammaires, langages). Un certain nombre d'outils facilitant la création d'interfaces n'ont jamais été diffusés pour les raisons majeures suivantes :

- Effort d'apprentissage et d'utilisation requis de la part du développeur trop élevé (*« high threshold »*).

- Trop complexes pour permettre une automatisation complète (*« low ceiling »*).

- On ne dispose pas d'un outil qui soit utile à la fois pour le développement (au niveau de la spécification) et la modélisation. En fait, les outils de spécification ne permettent pas la prédiction alors que les spécifications fournies par les modèles prédictifs ne sont pas assez précises pour aider les développeurs.

Objectifs spécifiques

Aux vues des problèmes précédents, les objectifs poursuivis dans le présent projet sont les suivants :

1) Concevoir un langage de spécification :

- destiné aux futurs concepteurs d'interfaces (ex. étudiants de programmes de génie logiciel et des TI). Ces derniers seront les usagers du langage AIR;

- ressemblant étroitement aux langages de programmation et de scriptage habituellement utilisés par les étudiants et professionnels du domaine de l'informatique afin de minimiser les efforts d'apprentissage et d'utilisation;

- permettant d'écrire des scripts d'interactions au niveau de détail et la réutilisation de sous-scripts à un niveau d'abstraction plus élevé.

2) Concevoir un modèle prédictif :

- qui peut être extrait directement d'un script AIR;

- qui soit aussi simple que le KLM tout en fournissant une précision de prédiction comparable à celle des standards en modélisation analytique, soit environ 20 %;

- reflétant le profil des usagers actuels (habiles) des GUI et leurs modes d'apprentissage (qui sont influencés par leurs automatismes).

3) Concevoir des outils logiciels :

- permettant de vérifier la syntaxe de scripts d'interactions écrits en langage AIR;

- permettant la prédiction automatique du temps d'exécution à partir d'un script AIR;

- intégrés à une interface usager qui permet la vérification et/ou la prédiction avec un minimum d'effort (apprentissage et utilisation) de la part de l'usager (concepteur).

1.6 Introduction au langage AIR

Le système AIR comprend un langage qui permet de spécifier les interactions entre un usager et une interface (au cours d'une tâche) et ce au niveau de détail. Chaque tâche est décrite par un script dont chaque étape se compose des éléments suivants : 1) Action de l'usager. 2) État interne du système. 3) Rétroaction de l'interface. 4) Durée de l'étape.

Chaque étape d'un script peut être spécifiée à différents niveaux allant d'une description abstraite (ex. déplacer une icône, ouvrir une application) à une description au niveau des opérations explicites (niveau *keystroke*). Les opérations explicites ont un effet direct sur l'interface (ex. frappe d'une touche du clavier, mouvement de pointage). Le système AIR comprend également un outil logiciel qui permet d'écrire des scripts sous forme de fichiers texte ou HTML et de vérifier leur syntaxe.

1.7 Introduction au modèle KPC

Le système AIR comprend également un outil permettant de prédire les temps d'exécution des usagers (réalisant des tâches sur interfaces GUI) à partir des scripts écrits en langage AIR. La prédiction s'effectue sur les trois opérations explicites suivantes : 1) Frappe d'une touche du clavier (K). 2) Mouvement de pointage (P). 3) Clic d'un bouton de la souris (C).

Dans un premier temps, on transforme le script AIR en une séquence d'opérations explicites sans tenir compte des changements d'état interne ou des rétroactions de l'interface. Une telle séquence est appelée un *modèle KPC*. Ensuite, pour chaque opération, on multiplie son nombre d'occurrences dans la séquence par un estimé de son temps moyen d'exécution. Ceci permet d'estimer le temps mis par un usager qui exécute le script AIR.

Le modèle KPC assume que les usagers sont *habiles*, c'est-à-dire hautement familiers avec les interfaces GUI. C'est le cas de la plupart des usagers actuels d'ordinateurs. Notre hypothèse est que même s'ils ne connaissent pas une interface donnée (état naïf), les usagers habiles apprendront rapidement à l'utiliser grâce aux automatismes acquis avec des interfaces similaires.

CHAPITRE 2

LANGAGE DE SPÉCIFICATION AIR

Ce chapitre présente le langage de spécification d'interfaces AIR. Dans un premier temps, on présente le cahier des charges relatant les spécifications techniques du langage. Deuxièmement, la façon de structurer une spécification est présentée. Ensuite, on aborde la méthodologie de développement de la grammaire du langage. Les sections suivantes présentent les généralités de la syntaxe puis les différents éléments et règles syntaxiques, respectivement.

2.1 Cahier des charges

Cette section présente les spécifications techniques du langage AIR. Elles sont les suivantes :

- Doit permettre au concepteur de décrire les interactions usager-interface à un niveau détaillé.

- Doit permettre l'utilisation d'une notation ouverte (c'est-à-dire, une notation qui peut être augmentée par le concepteur) afin de supporter les changements apportés aux interfaces (ex. nouveaux dispositifs d'interaction et/ou composants logiciels).

- Doit permettre au concepteur de décrire les interactions à n'importe quel niveau d'agrégation ou d'abstraction.

- Doit permettre au concepteur de spécifier les interactions de façon concise en réutilisant des scripts déjà écrits (librairies de sous-scripts).

- Ses éléments et règles syntaxiques doivent être similaires à ceux utilisés en programmation structurée (ex. C++, Java).

- Doit être facile à apprendre et à écrire par le concepteur (*« low threshold »*). C'est-à-dire qu'on doit pouvoir commencer à écrire des spécifications en connaissant le moins d'éléments de syntaxe possible.

- Doit être facile à lire, comprendre et interpréter par un programmeur (qui n'est pas le concepteur). Ce dernier utilisera un script AIR afin de programmer une interface.

- Doit être accompagné d'un outil automatisé permettant aux concepteurs de respecter les règles syntaxiques. L'outil doit pouvoir fonctionner sur plusieurs plateformes (ex. Windows et Linux).

2.2 Structure de représentation

La spécification AIR d'une tâche donnée s'effectue sous forme d'un tableau (script) tout comme l'UAN (chapitre 1). Un script se compose du nombre d'étapes (lignes) nécessaires à la réalisation de la tâche. Chaque étape se compose de 5 colonnes. Elles sont les suivantes : i) Numéro de l'étape. ii) Action de l'usager. iii) État (ou changement d'état) interne du système. iv) Rétroaction de l'interface (entre autres, les effets appliqués sur les composants graphiques). v) Durée (temps d'exécution) de l'étape.

Un script AIR s'exécute une étape à la fois (c'est-à-dire de haut en bas). Chaque étape est exécutée dans l'ordre suivant (de gauche à droite) : 1) Action. 2) Changement d'état interne, s'il y a lieu. 3) Rétroaction de l'interface, s'il y a lieu. Le tableau II présente un script AIR pour la tâche de sélection d'un fichier.

Tableau II

Exemple d'exécution d'un script AIR

Step	Action	State	Feedback	Duration
1	repeat * times			
1.1	mouse.goto(x,y)		mouse.arrow	700 ms
1.2	endrepeat			
2	mouse.goto(fileIcon)			700 ms
3	mouse.press	selected = fileIcon	fileIcon.activate	
4	mouse.release			235 ms
	Description			
1 – 1.2	L'usager déplace la souris. L'icône de la souris (flèche) est visible à l'écran.			
2	L'usager positionne la souris sur l'icône du fichier.			
3	L'usager appuie sur le bouton de gauche de la souris. Du côté système, l'icône du fichier est sélectionnée. Sur l'interface, l'icône est activée (apparence grisée).			
4	L'usager relâche le bouton de gauche de la souris.			

2.3 Développement de la grammaire du langage

Après avoir déterminé les spécifications du langage, nous avons procédé au développement de sa grammaire. Pour chaque colonne (action, état interne, rétroaction et durée) d'un script AIR, les différents cas de spécification possibles ont été déterminés. Pour chaque cas, l'expression générique a été déterminée. Ensuite, nous avons regroupé chaque expression dans un schéma représentant la structure grammaticale de la colonne en question. La structure grammaticale regroupe syntaxe et règles syntaxiques (structures de contrôles, opérateurs, etc.). Tous les schémas construits sont présentés à l'annexe 1.

Après avoir construit les différents schémas de la structure grammaticale, les différents éléments de syntaxe ont été développés. Dans un premier temps, les concepts généraux ont été déterminés. Ensuite, notre attention a été portée sur les détails syntaxiques de chaque type d'expression. Tout au long de ce processus, nous avons comparé notre conception avec les éléments de syntaxe utilisés dans les langages de programmation structurés comme C++ et Java. Les prochaines sections présentent un survol des différents éléments et règles syntaxiques. Un document de spécification plus détaillé du langage et de sa syntaxe est présenté à l'annexe 2.

2.4 Généralités sur la syntaxe du langage

Expressions et identificateurs

Un script AIR se remplit en écrivant des expressions dans les cellules (du tableau) appropriées. Il est possible d'écrire plusieurs expressions par cellule. Les identificateurs apparaissant dans les expressions suivent la même convention que les noms de variables en langage Java[1] (ex. *myApplication*, *okButton*, *loginWindow*). Les expressions peuvent également être composées d'opérateurs et de structures de contrôle (du flot d'exécution) de façon similaire à ce qu'on retrouve en programmation structurée. En particulier, les actions de l'usager ou les rétroactions de l'interface se présentent généralement sous la forme d'invocation de méthodes (ex. *mouse.goto(textArea)*, *keyboard.enter(« allo »)*, *button.display*).

Mots-clés (ou réservés)

La syntaxe contient des mots (identificateurs) réservés, entre autres, pour : i) les fonctions associées aux dispositifs d'interaction et actions de l'usager (ex. *mouse.**press***,

[1] La convention de *Sun Microsystems* est la suivante: les noms de variables sont écrits en lettres minuscules. Si un nom de variable est constitué de plus d'un mot, les mots sont joints ensembles, et tous, sauf le premier mot, commencent par une lettre majuscule.

*user.**seek***), ii) les variables d'état (ex. ***selected*** = *icon*), iii) les effets associés aux composants graphiques (ex. *button.**display***, *window.**activate***) et iv) le type de la durée (ex. ***specified***, ***estimated***).

2.5 Actions

Les expressions de la colonne *Action* servent à spécifier les activités que l'usager peut effectuer avec un dispositif d'interaction (pointeur, dispositif d'entrée de texte, interrupteur) ou par lui-même (activités physiques, activités cognitives). Une telle expression débute par l'identificateur du dispositif ou de l'usager et se termine par l'identificateur de la fonction exécutée (ex. *mouse.click*, *user.gaze*).

Il existe trois types de dispositifs d'interaction: i) Dispositifs de pointage ou pointeurs. ii) Dispositifs d'entrée de texte. iii) Dispositifs d'interruption ou interrupteurs. Dans le cas où une action peut être effectuée avec n'importe quel dispositif, le préfixe *any* est ajouté au début de l'identificateur du dispositif (ex. *anyPointer.click*, *anyTextEntry.enter*, *anySwitch.press*). Consulter les annexes 1 et 2 afin d'obtenir plus de détails sur la syntaxe et la structure grammaticale.

Dispositifs de pointage (pointeurs)

Les dispositifs de pointage permettent de se déplacer dans l'écran et de déclencher des actions via leurs boutons. La syntaxe AIR permet de représenter les dispositifs suivants : i) Le pointeur générique *anyPointer* qui signifie que n'importe quel pointeur peut être utilisé. ii) La souris (*mouse*). iii) Le manche à balai (*joystick*).

Une expression impliquant un pointeur est constituée en ordre de son identificateur, d'un séparateur (« . »), et de l'action (ou fonction) exécutée. Les différentes actions possibles sont : i) *goto* : mouvement du pointeur, ii) *press* : presse d'un bouton, iii) *release* : relâchement d'un bouton et iv) *click* : presse et relâchement successifs d'un bouton. Un

exemple de pointage à une position absolue (x_{abs}=100, y_{abs}=200) de l'écran avec la souris est le suivant : *mouse.goto (100,200)*. Le click du bouton de gauche du joystick se traduit par : *joystick.leftButton.click*. L'annexe 2 fournit plus de détails sur ce type de dispositifs et leurs actions.

Dispositifs d'entrée de texte

Ces dispositifs permettent d'entrer du texte dans un champ ou une zone de texte. La syntaxe AIR permet de représenter les dispositifs suivants : i) Le dispositif générique *anyTextEntry* qui signifie que l'entrée de texte peut s'effectuer avec le dispositif de notre choix. ii) Le clavier (*keyboard*). Dans ce cas, les touches de commandes (ex. *Enter*, *Tab*) qui ont un effet de déplacement ne font pas partie de la saisie de texte. Elles sont plutôt considérées comme des interrupteurs (voir sous-section suivante). iii) Un dispositif de saisie d'écriture (*writingPad*) soit manuscrite (« *Palm Pilots* ») ou avec reconnaissance de caractères ou de mots (« *Tablet PCs* »). iv) Le microphone (*voice*).

Une expression impliquant un tel dispositif est constituée en ordre de son identificateur, du séparateur et de la seule fonction possible, soit *enter*. La chaîne de caractères saisie peut être soit forcée (ex. *voice.enter (« login »)*) ou non forcée (ex. *keyboard.enter()*). Dans le dernier cas, l'usager peut entrer ce qu'il veut. L'annexe 2 fournit plus de détails sur ce type de dispositifs.

Dispositifs d'interruption (interrupteurs)

Ces dispositifs permettent de manipuler des boutons à deux ou plusieurs états. La syntaxe AIR actuelle ne possède qu'un identificateur, soit celui de l'interrupteur générique *anySwitch*.

Une expression impliquant un interrupteur s'écrit de la même façon que pour les pointeurs et les dispositifs d'entrée de texte. Les différentes actions possibles sont : i) *press(level)* : presse d'un bouton au niveau spécifié, ii) *release* : relâchement d'un

bouton et iii) *click(level)* : presse et relâchement successifs d'un bouton au niveau spécifié. Par exemple, l'activation d'un interrupteur au niveau 1 suivie de sa désactivation se traduit par *anySwitch.press(1)* suivi de *anySwitch.release.* L'annexe 2 fournit plus de détails sur ce type de dispositifs.

L'usager

Les actions de l'usager peuvent être soit implicites (ex. mouvement de la main du clavier vers la souris, hésitation, réflexion, recherche visuelle à l'écran, perception d'un changement à l'écran) ou explicites (ex. mouvement du bras vers l'écran dans le cas d'un écran tactile).

Une expression impliquant l'usager est constituée en ordre de l'identificateur *user*, du séparateur et de l'action exécutée. Les actions implicites possibles sont les suivantes : i) *home* : déplacement de la main dominante du clavier vers la souris et vice et versa ou du clavier principal vers le clavier numérique et vice et versa. ii) *think* : pause mentale (réflexion, hésitation, etc.). iii) *gaze* : poser le regard sur une position de l'écran. iv) *seek* : recherche visuelle à l'écran. La seule action explicite possible est un mouvement de la main ou du doigt vers un écran tactile et se traduit par *goto*.

Par exemple, le déplacement de la main droite vers la position absolue (x_{abs}=100, y_{abs}=200) d'un écran tactile est représenté par *user.rightHand.goto(100,200).* Le déplacement de la main dominante de l'usager entre le clavier et la souris, quant à lui, se traduit en AIR par *user.home.* Pour plus de détails sur les actions de l'usager, consulter l'annexe 2. L'exemple du tableau III présente les différentes actions effectuées par l'usager pour s'identifier sur un ordinateur. Dans cet exemple, les colonnes *State* et *Duration* sont omises.

Tableau III

Langage AIR – exemple d'utilisation de la colonne *Action*

Step	Action	Feedback	Description
1	mouse.goto(userField)	mouse.arrow	Déplace la souris sur un champ de texte. L'icône de la souris est visible à l'écran.
2	mouse.click	cursor.blink	Clique dans le champ. Le curseur texte clignote.
3	user.home		Déplace la main de la souris vers le clavier.
4	keyboard. enter(« my_user »)	userField.text	Entre le nom d'usager. Le texte apparaît dans le champ.
5	Tab.click		Tape la touche de commande *Tab*.
6	keyboard. enter(« my_password »)	passwordField.text	Tape le mot de passe.
7	user.home		Déplace la main de la souris vers le clavier.
8	mouse.goto(okButton)	mouse.arrow	Déplace la souris sur le bouton de confirmation.
9	user.think		Réfléchit à savoir si tout est correct.
10	mouse.click	newWindow	Clique sur le bouton de confirmation. Une nouvelle fenêtre apparaît.

2.6 État interne du système

Les expressions de la colonne *State* servent à spécifier l'état (ou changement d'état) interne du système (matériel ou logiciel) en réponse à une action de l'usager. Par exemple, une icône est sélectionnée lorsque l'usager appuie un bouton de la souris sur cette dernière (*selected = fileIcon*). Une expression de la colonne *State* est constituée dans l'ordre d'une variable d'état matériel (ex. *inputMouse*, *inputKeyboard*) ou logiciel (ex. *selected*, *stored*, *frozen*), d'un opérateur d'affectation et de la valeur assignée. Le langage ne pose pas de limite sur les noms de variables, et on peut aussi leur assigner des valeurs numériques ou des chaînes de caractères. Bien que ceci permet d'étendre la notion d'état interne bien au delà de l'interface proprement dite, il est recommandé de se limiter aux variables qui affectent directement l'aspect et le fonctionnement de l'interface. L'annexe 2 présente l'ensemble des variables habituellement gérées par le système. L'exemple du tableau IV montre ce qui se passe en mémoire lors d'une entrée de texte. Dans cet exemple, la colonne *Duration* est omise.

Tableau IV

Langage AIR – exemple d'utilisation de la colonne *State*

Step	Action	State	Feedback	Description
1	mouse.click(textField)		cursor.blink	Clique dans un champ.
2	keyboard. enter(«information»)	stored= textField.content	textField.text	Tape du texte. Le contenu du champ est emmagasiné en mémoire.

2.7 Rétroaction de l'interface

Les expressions de la colonne *Feedback* servent à spécifier ce que l'usager perçoit à l'écran en réponse à une action posée. Il s'agit, en majeure partie, des effets appliqués sur les composants graphiques. Le cas échéant, une telle expression débute par

l'identificateur du composant graphique et se termine par l'identificateur de l'effet appliqué (ex. *button.activate*, *frame.display*). Tous les effets applicables peuvent être représentés. L'annexe 2 présente l'ensemble des effets habituellement observés. Dans le cas où un effet doit être appliqué à tous les composants d'un type spécifique, le préfixe *any* est ajouté au début de l'identificateur du composant (ex. *anyWindow.close*, *anyButton.activate*). Ce cas est différent de celui pour les dispositifs d'interaction (ex. *anyPointer.goto(x,y)*) où *any* signifie que l'action est effectuée par n'importe quel dispositif parmi un groupe spécifique. L'exemple du tableau V illustre la rétroaction de l'interface pour un extrait de navigation Web. Dans cet exemple, les colonnes *State* et *Duration* sont omises.

Tableau V

Langage AIR – exemple d'utilisation de la colonne *Feedback*

Step	Action	Feedback	Description
1	repeat * times		Déplace la souris à l'écran
2	mouse.goto(x,y)	mouse.arrow	Affichage du composant graphique de la souris (flèche).
3	endrepeat		
4	mouse.goto(infoLink)	mouse.hand infoLink.activate	Affichage du composant graphique de la souris (main). Le lien *infoLink* devient plus foncé.
5	mouse.click	infoWindow.display	Une nouvelle fenêtre (*infoWindow*) apparaît.

2.8 Durée d'exécution

La durée d'une étape est soit spécifiée, estimée ou mesurée. Lorsque le concepteur connaît la durée d'une étape a priori, il la spécifie directement dans la colonne *Duration*. La modélisation analytique permet au concepteur d'obtenir rapidement un estimé de la durée d'une étape. Il utilise un modèle (ex. KLM, KPC) afin de représenter la durée par

un symbole qu'il inscrit dans la colonne *Duration*. Pour ce qui est de la durée mesurée, elle est obtenue par enregistrement des actions de l'usager sur l'interface. Un programme renifleur effectue l'enregistrement puis filtre les données pour obtenir une séquence d'actions et leur durée. Ceci est utilisé sur un prototype de l'interface afin de mesurer l'efficience du design.

La représentation de la durée est constituée, dans l'ordre, du temps, de l'unité et du mot-clé approprié parmi les suivants: *specified*, *estimated* et *measured*. Les trois composants sont séparés par des espaces. Un exemple pour la durée estimée d'un pointage de la souris serait : ***700 ms estimated***. Consulter les annexes 1 et 2 pour plus de détails.

2.9 Structures de contrôle

La syntaxe AIR permet l'utilisation des structures de contrôle du flot d'exécution comme en programmation structurée. Il y a trois types permis : i) Exécutions conditionnelles (*if, switch*). ii) Boucles (*for, while, repeat*). iii) Sauts directs (*step, break, continue, return*). Ces structures sont employées dans les colonnes *Action*, *State* et *Feedback* et leur effet débute dans la colonne en question (exécution de gauche à droite). Un exemple d'utilisation de la structure conditionnelle *if (boolean) ... endif* est présenté au tableau VI. Dans cet exemple, on spécifie l'effet qui est visible à l'écran lorsque la souris est à une position spécifique. Dans cet exemple, la colonne *Duration* est omise.

Tableau VI

Langage AIR – exemple d'utilisation d'une structure de contrôle

Step	Action	State	Feedback	Description
1	mouse.goto(area)		mouse.arrow	Déplace la souris sur une région graphique.
2	if (mouse ON area)			

Tableau VI (suite)

Step	Action	State	Feedback	Description
2.1		selected=area	area.activate	Lorsque l'icône de souris est sur la région, cette dernière change d'apparence.
2.2	endif			

2.10 Encapsulation

Les appels de scripts secondaires (sous-scripts) sont utiles pour : i) simplifier la lecture des scripts, ii) éviter d'écrire un script plus d'une fois et iii) faciliter la mise à jour des scripts en cas de modification de l'interface. Ils sont similaires aux appels de fonctions en programmation. Les sous-scripts doivent porter un nom explicite (ex. *cutAndPaste*, *copyAll*). Leur appel est spécifié en indiquant leur nom et si nécessaire, leur(s) paramètre(s) d'entrée entre parenthèses. L'exécution du script appelé s'effectue de haut en bas et de gauche à droite. Ensuite, l'exécution du script appelant reprend à la cellule suivant celle de l'appel. Il faut souligner que même si un sous-script est appelé dans la colonne *Action*, il y a des changements d'état et des rétroactions durant son exécution.

Un exemple est le cas d'une action de déplacement d'une icône (*« drag-and-drop »*) à l'écran qui est présenté aux tableaux VII et VIII. Dans cet exemple, les colonnes *State* (tableau VII) et *Duration* (tableaux VII et VIII) sont omises. À l'étape 1 du script principal (tableau VII), l'usager positionne la souris sur une icône à l'écran. À l'étape 2, le sous-script *dragAndDrop* est appelé afin de spécifier le déplacement de l'icône. À la fin du sous-script, l'exécution reprend dans la colonne *State* de l'étape 2 du script principal.

Tableau VII

Langage AIR – exemple d'appel d'un sous-script

Step	Action	Feedback	Description
1	mouse.goto(icon)	mouse.arrow	
2	dragAndDrop(icon)	icon	Appel du sous-script
3	mouse.goto(newWindow)	mouse.arrow	

Tableau VIII

Langage AIR – sous-script *dragAndDrop(icon)*

Step	Action	State	Feedback
1	mouse.press	selected=icon	icon.activate
2	mouse.goto(x,y)		mouse.arrow && icon.outline
3	mouse.release		icon

2.11 Séquences de sous-scripts

L'appel de plusieurs sous-scripts lors d'une même étape (même cellule) est utilisé, si nécessaire, pour gérer les relations temporelles entre ceux-ci. Cela permet également d'alléger le script. Dans le cas général, les sous-scripts sont exécutés dans un ordre aléatoire (ou quelconque). Les autres cas sont les suivants :

- Concurrence : les sous-scripts sont exécutés simultanément. L'appel est suivi de l'identificateur *concurrent*.

- Interruption : chaque sous-script peut être interrompu par n'importe quel autre sous-script de la liste et ce n'importe quand. L'appel est suivi de l'identificateur *interrupted*.

- Entrelacement : des sections de chaque sous-script sont exécutées tour à tour. Il s'agit d'exécutions entrelacées. L'appel est suivi de l'identificateur *interleaved.*

Le tableau IX illustre deux types de séquences de sous-scripts. La première tâche spécifie l'entrée du nom d'usager et du mot de passe dans un ordre aléatoire. La deuxième spécifie une édition de texte et une tâche Web entrelacées. Dans les deux cas, seulement la colonne *Action* est présentée.

Tableau IX

Langage AIR – exemples de séquences de sous-scripts

Action	Description
Tâche 1	
enterFirstName() enterLastName()	**Cas général**. Les deux scripts sont exécutés de façon aléatoire.
Tâche 2	
editText() webBrowse() interleaved	Les exécutions de ces deux scripts sont entrelacées.

2.12 Notation ouverte

Il est permis d'augmenter la syntaxe AIR actuelle pour représenter d'autres types d'interfaces. Ces interfaces seront le résultat de l'évolution des dispositifs d'interactions et/ou de l'apparition de nouveaux composants logiciels. Un exemple d'augmentation de la syntaxe pour représenter une impulsion envoyée à l'ordinateur par une personne quadriplégique via son dispositif d'interaction est le suivant : *quadPad.sendImpulse.*

Dans ce chapitre, la méthodologie de conception du langage AIR et ses éléments syntaxiques de base ont été présentés. Il a été montré que l'AIR permet de spécifier les détails des interactions entre un usager et une interface sous la forme d'un script. On a également vu qu'il est possible d'alléger un script en utilisant l'abstraction via des appels de sous-scripts. Le langage permet de faire plus que ce qui est présenté dans ce

chapitre. Les détails du langage ont donc été regroupés à l'annexe 2 de ce document. Afin d'aider le concepteur à respecter les règles syntaxiques de l'AIR, un outil logiciel a été développé. Ce dernier permet de vérifier la syntaxe d'un script de façon automatique. Cet outil est présenté au chapitre 3.

CHAPITRE 3

OUTILS LOGICIELS AIR

Ce chapitre présente les outils logiciels du système AIR. Dans un premier temps, on présente l'outil de vérification syntaxique. Deuxièmement, le module d'estimation du temps d'exécution est présenté. Finalement, on présente l'interface usager. Les outils permettent, de façon automatique, de vérifier la syntaxe d'un script AIR (permettant ainsi de respecter les règles de conception AIR) et d'estimer son temps d'exécution.

3.1 Outil de vérification syntaxique

Méthodologie de conception de l'outil de vérification syntaxique

Cet outil fût développé en collaboration avec trois étudiants de premier cycle de l'ÉTS dans le cadre du cours *Projet synthèse en génie logiciel (LOG790)*. La conception a été effectuée en suivant les étapes suivantes :

1) Une structure de donnée (arbre) a été choisie afin de représenter une expression d'un script AIR en mémoire.

2) Pour chaque type d'expression (action, état, rétroaction et durée), le jeu d'essai (annexe 7) fût construit à partir du schéma de la structure grammaticale (annexe 1).

3) Un programme logiciel permettant de construire les arbres pour chaque expression d'un script a été développé. Cela s'est fait en suivant intimement les schémas des différentes structures grammaticales.

4) Une fois le logiciel réalisé, les différents jeux d'essai ont été utilisés afin d'en valider son fonctionnement.

5) Les étapes 1 à 4 ont été répétées jusqu'à ce que le logiciel produise le résultat attendu pour tous les jeux d'essai.

Fonctionnement général de l'outil de vérification syntaxique

La figure 2 illustre le fonctionnement général de l'outil de vérification.

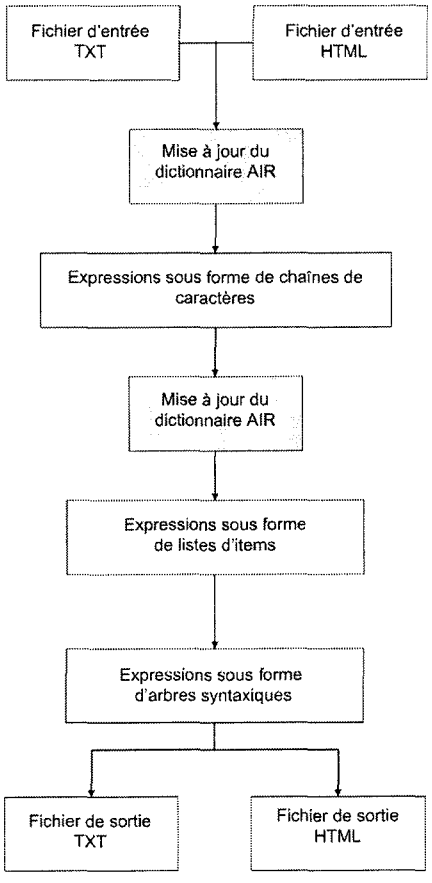


Figure 2 Fonctionnement général de l'outil de vérification syntaxique

Un fichier contenant le script AIR à vérifier constitue le point d'entrée de l'outil. Ce fichier doit être écrit soit en format TXT (texte) ou HTML. Le contenu (expressions) du script est ensuite vérifié cellule par cellule de haut en bas et de gauche à droite. Chaque expression ou chaîne de caractères est transformée en une liste d'items (mots de la chaîne). Les listes d'items sont ensuite utilisées pour construire les arbres syntaxiques. Un arbre n'est construit que si l'expression ne contient pas d'erreur de syntaxe. Si toutes les expressions sont syntaxiquement valides, un fichier de sortie est créé automatiquement à partir du contenu de chacun des arbres. Dans le cas contraire, un message d'erreur indiquant le type d'erreur et le lieu (cellule) est généré.

Dictionnaire d'identificateurs

Le dictionnaire AIR (annexe 3) contient les différents identificateurs du langage et leur type. Les types possibles sont présentés à l'annexe 4. Ces types sont utilisés pour vérifier si une expression est syntaxiquement valide. Le concepteur peut déclarer ses propres identificateurs si leur type le permet (voir annexe 4 pour les types d'identificateurs que l'on peut déclarer). Ceci s'effectue à la création du script en en-tête du fichier. L'outil mettra alors à jour le dictionnaire, qui est stocké en mémoire au démarrage de l'outil. Si certains identificateurs ne font pas partie du dictionnaire et ne sont pas déclarés, alors un type leur est associé automatiquement en fonction du contenu de la cellule. Ici également, l'outil mettra à jour le dictionnaire. Les prochaines sections présentent plus en détail les différentes étapes du fonctionnement de l'outil.

Édition d'un script AIR (fichier d'entrée)

Le concepteur écrit un script AIR en utilisant un des deux formats suivants :

- TXT : les colonnes sont séparées par une ou plusieurs tabulations (*Tab*). La déclaration des identificateurs est effectuée en en-tête de fichier dans la section

délimitée par <declare> et </declare>. Les commentaires doivent être précédés du caractère « ; ». Un patron de fichier de ce format est présenté à l'annexe 5. *Restriction* : une expression par cellule maximum.

- HTML : la déclaration des identificateurs est effectuée dans un tableau réservé à cette fin en haut du tableau utilisé pour le script principal. Les commentaires sont rajoutés dans le code source HTML en utilisant les balises de commentaires. Un patron de fichier de ce format est présenté à l'annexe 5. *Restriction* : une expression par cellule maximum.

Mise à jour du dictionnaire

Le dictionnaire AIR doit être mis à jour si certains identificateurs du script n'en font pas partie initialement. Ceci est équivalent à la déclaration dynamique de variables en programmation. La mise à jour est effectuée à deux moments différents dans le processus de vérification (figure 2) : i) Lorsque les déclarations d'identificateurs sont vérifiées. Les identificateurs déclarés et leur type sont alors ajoutés au dictionnaire. ii) Lors de la vérification d'une expression avant de générer la liste d'items. Si un identificateur n'est pas dans le dictionnaire ni déclaré, un type lui est associé en fonction du contenu de l'expression. Cela est effectué en vérifiant les types des autres identificateurs. Ensuite, l'identificateur et son type sont ajoutés au dictionnaire. Ce processus est illustré à l'annexe 4.

Représentation interne d'un script AIR

Chaque expression est représentée en mémoire par un arbre syntaxique suivant deux étapes :

1) L'expression est transformée en une liste d'items (ou mots). Nous avons modifié la classe *StringTokenizer*[1] de Java afin de gérer efficacement les tabulations et espaces présents à l'intérieur de l'expression AIR ainsi que pour utiliser des délimiteurs ayant une longueur supérieure à un caractère.

2) L'arbre syntaxique est créé à partir de la liste d'items. La création de l'arbre se divise en deux étapes : i) déterminer la racine et ii) ajouter les feuilles. Dans l'annexe 6, on peut voir, pour chaque type d'expression, quels types d'identificateurs peuvent représenter la racine et ses feuilles. Si la création de l'arbre est impossible, alors un message d'erreur syntaxique (spécifiant le type d'erreur et la cellule touchée) est généré.

Script AIR vérifié syntaxiquement (fichier de sortie)

Un fichier de sortie du même format que celui d'entrée est généré dans lequel apparaît le script validé. Chaque expression est reconstruite à partir du contenu de son arbre en mémoire. Tous les identificateurs utilisés et leur type apparaissent en en-tête de fichier. Ce fichier pourra être réutilisé si l'on veut vérifier la syntaxe à nouveau suivant des modifications. Ceci évitera de déclarer à nouveau les identificateurs. En cas d'erreur, le fichier n'est pas généré et le ou les messages apparaissent à la console.

---

[1] *StringTokenizer* est une classe Java permettant de diviser une chaîne de caractères en items (*tokens* en Java) à l'aide de délimiteurs d'une longueur d'un caractère (ex. espace, point).

Limitations de l'outil de vérification syntaxique

Les limitations techniques sont les suivantes : i) La vérification est intracellulaire. Cela signifie que la cohérence entre les différentes cellules n'est pas vérifiée. ii) La sémantique n'est pas testée.

Structure du programme de vérification syntaxique

Le code du programme a été écrit entièrement en langage Java en utilisant l'environnement de développement *Eclipse*. Il est composé de six paquetages (*packages* en Java) qui sont décrits dans le tableau X.

Tableau X

Outil de vérification syntaxique – structure du programme

Paquetage	Contenu
air.memoryscript	Classes pour la représentation interne (chaînes de caractères, listes d'items et arbres syntaxiques): *TextScript*, *SymbolicScript* et *AirTreeNode*.
air.tokenizer	Classes servant à transformer les chaînes de caractères en listes d'items: *StringTokenizer*, *StringToTokens*, *TokenList* et *TokenScript*.
air.parser	Classe nécessaire à la construction d'un arbre à partir d'une liste d'items : *Parser*.
air.dictionary	Classes de gestion du dictionnaire : *StaticDictionary*, *DynamicDictionary*, *DictionaryEntry* et *EntryTypes*.
air.analyzer	Classe unique contenant la fonction *main()* : *Analyzer*.
air.exception	Classes de gestion des exceptions : *InvalidAsciScriptException*, *InvalidHtmlScriptException*, *InvalidTypeException*, *InvalidDurationException*, *MissingDictionaryEntryException* et *InvalidSyntaxException*.

Des classes de type conteneur (*« container »*) ont été utilisées afin d'alléger le code et de limiter le nombre d'endroits où les types des éléments sont forcés (*« casting »*). Elles sont les suivantes: *TokenList*, *TokenScript*, *TextScript* et *SymbolicScript*. Ces classes contiennent une collection de données et des méthodes (fonctions) pour y accéder. Dans la classe *Parser*, la récursivité a été utilisée pour la construction des arbres syntaxiques. Cela rend le code plus léger et élimine les répétitions inutiles. Enfin, le patron *Singleton* a été utilisé pour les classes de gestion du dictionnaire, soit *StaticDictionary et DynamicDictionary*. Ainsi, l'application ne pourra jamais contenir plus d'une instance de chacune de ces classes; la méthode *getInstance()* étant alors utilisée pour récupérer l'instance.

3.2 Module d'estimation du temps d'exécution

Fonctionnalités du module d'estimation du temps d'exécution

Ce module a été greffé au programme de vérification syntaxique pour pouvoir calculer le temps d'exécution d'un script AIR. Il a été développé uniquement dans le cadre de ce projet de maîtrise contrairement à l'outil de vérification (LOG790). Le temps d'exécution est calculé de deux façons différentes :

- Mode de spécification (*specified*) : les temps spécifiés par le concepteur dans la colonne *Duration* sont additionnés afin de calculer le temps total d'exécution.

- Mode d'estimation (*estimated*) : l'estimation du temps total d'exécution se fait en utilisant le modèle KPC (chapitre 4). Les deux étapes à suivre sont les suivantes : 1) Extraire la séquence KPC à partir des actions spécifiées (colonne *Action*). Pour ce faire, une des trois opérations du KPC est associée pour chaque action explicite (ayant un effet sur l'interface) en ajoutant le symbole (K, P ou C) dans la colonne *Duration* du script puisque ces symboles correspondent à des temps

d'exécution. 2) Calculer le temps total d'exécution du script. Pour ce faire, la séquence de symboles de la colonne *Duration* est observée (ex. PCKKKKPC) et le temps total estimé à partir des estimés des opérations K, P et C (330 ms, 700 ms et 235 ms).

Limitation du module d'estimation du temps d'exécution

La principale limitation technique réside dans l'estimation du temps d'une boucle. En fait, la flexibilité de la syntaxe rend difficile la détermination automatique du nombre d'itérations spécifié. Pour surmonter cette limitation, le temps total estimé de la boucle est obtenu sous la forme d'une expression du type *X ms * loopCounter* où X représente le temps estimé pour une itération seulement.

Structure du programme d'estimation du temps d'exécution

Deux paquetages (3 classes) ont été ajoutés à la structure du programme de vérification syntaxique. Ils sont décrits dans le tableau XI.

Tableau XI

Module d'estimation du temps d'exécution - structure du programme

Paquetage	Contenu
air.heuristics	Classes contenant les règles de calcul du temps d'exécution en fonction du mode (spécification, estimation): *SpecifiedEt* et *EstimatedEt*.
air.etComputer	Classe unique permettant le calcul du temps d'exécution : *EtComputer*.

3.3 Interface usager et exemple d'utilisation

Interface usager

Les outils (outil de vérification syntaxique et module d'estimation du temps d'exécution) permettent de vérifier la syntaxe d'un script AIR et d'en estimer le temps d'exécution avant même d'implanter l'interface. Le diagramme de transition de l'interface usager est présenté à la figure 3. Dans cette figure, le texte associé à une flèche représente une action ou commande exécutée. Les différents écrans de l'interface sont présentés à l'annexe 8.

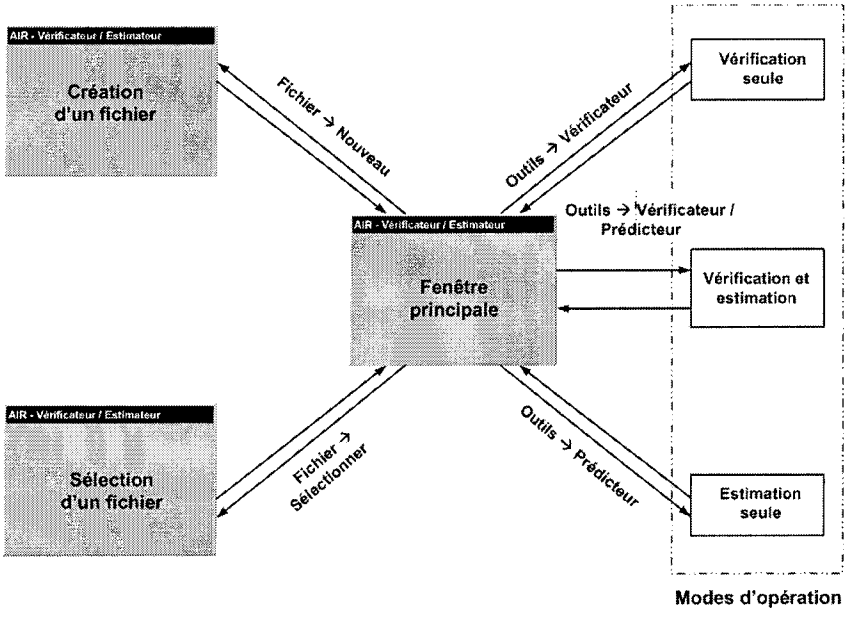


Figure 3 Outils logiciels AIR – diagramme de transition de l'interface usager

L'interface usager permet trois modes d'opération :

- Vérification syntaxique et estimation du temps d'exécution : on utilise les fonctionnalités des programmes de vérification (section 3.1) et d'estimation (section 3.2).

- Vérification syntaxique seulement : on utilise les fonctionnalités de l'outil de vérification.

- Estimation du temps d'exécution seulement : on utilise les fonctionnalités du module d'estimation. Cela permet d'estimer le temps total d'exécution d'une séquence KPC (fichier TXT) fournie en entrée.

Exemple d'utilisation

Cet exemple démontre le fonctionnement des outils de vérification et d'estimation de temps (mode de calcul *estimated*). La tâche spécifiée (sans erreur) est l'action de déplacement d'une icône à l'écran. La spécification contient les étapes suivantes:

1) L'usager déplace la souris sur une icône nommée *icon*.
2) L'usager appuie sur le bouton de gauche de la souris. L'icône devient active (apparence grisée).
3) L'usager déplace la souris à la position absolue de l'écran : x_{abs}=100, y_{abs}=300. Pendant ce temps, le contour (*outline*) de l'icône est visible à l'écran.
4) L'usager relâche le bouton de la souris. À ce moment, l'icône apparaît à la nouvelle position (x_{abs}=100, y_{abs}=300). Elle est encore d'apparence grisée.
5) L'usager déplace la souris sur le bureau (*desktop*) à côté de l'icône et clique. À ce moment, l'icône reprend son apparence d'origine (elle est désactivée).

La figure 4a présente le script AIR (fichier TXT) fourni en entrée. Le script vérifié (fichier de sortie) est présenté à la figure 4b. La représentation de la tâche en terme de K, P et C est visible dans la colonne *Duration*. La séquence KPC obtenue est : P **CPC** PC où les trois opérations en gras représentent le déplacement de l'icône. L'affichage de la console est présenté à la figure 4c. Ce dernier indique le temps d'exécution sous la forme de l'expression suivante : Delay (0.0) + 0 * K (330 ms) + 3 * P (700 ms) + 3 * C (235 ms) pour un temps d'exécution estimé de 2.805 secondes.

Ce chapitre a présenté les outils logicels développés dans ce projet ainsi que la méthodologie de conception employée. Premièrement, on a présenté un outil aidant le concepteur à respecter les règles syntaxiques du langage AIR. On a également présenté un outil permettant d'extraire automatiquement un modèle KPC d'un script AIR et de fournir un estimé du temps moyen d'exécution. Le KPC est présenté en détail au chapitre suivant.

a) fichier TXT d'entrée

```
Step    Action                  State   Feedback                        Duration
1       mouse.goto(icon)        ---     mouse.arrow                     ---
2       mouse.press             ---     icon.activate                   ---
3       mouse.goto(100,300)     ---     mouse.arrow && icon.outline     ---
4       mouse.release           ---     icon.display                    ---
5       mouse.goto(200,300)     ---     mouse.arrow                     ---
6       mouse.click             ---     icon.deactivate                 ---
```

b) fichier TXT de sortie

```
Step    Action                  State   Feedback                        Duration
1       mouse.goto(icon)        ---     mouse.arrow                     P
2       mouse.press             ---     icon.activate                   C
3       mouse.goto(100,300)     ---     mouse.arrow && icon.outline     P
4       mouse.release           ---     icon.display                    C
5       mouse.goto(200,300)     ---     mouse.arrow                     P
6       mouse.click             ---     icon.deactivate                 C
```

c) affichage de la console

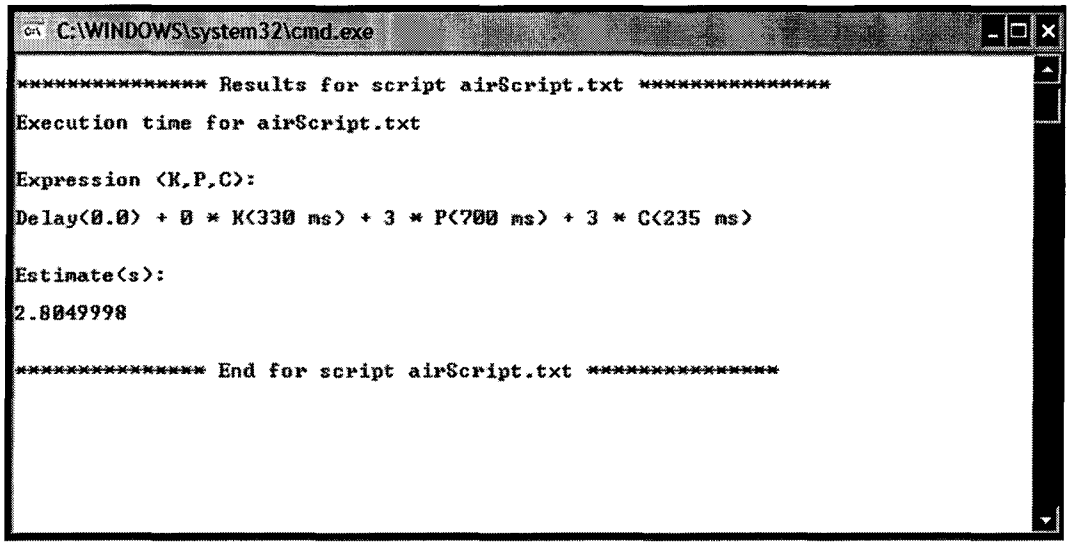


Figure 4 Outils logiciels AIR - exemple de fonctionnement

CHAPITRE 4

MODÈLE KPC

Ce chapitre présente un modèle permettant de prédire les performances des usagers habiles (voir chapitre 1) dans l'exécution de tâches sur interfaces GUI. Ce modèle porte le nom de KPC. Il permet de transformer un script écrit en langage AIR en une séquence simplifiée d'opérations explicites (K, P et C) sans tenir compte des changements d'état interne ou des rétroactions de l'interface pour enfin estimer le temps d'exécution. Dans un premier temps, on présente les fonctionnalités de base du KPC. Ensuite, il est démontré que le KPC peut être appliqué à un script AIR permettant de joindre l'évaluation à la conception. Finalement, un processus empirique de construction du KPC à partir de la tâche réellement exécutée par l'usager est proposé.

4.1 Représentation de la tâche et prédiction de temps

Le modèle KPC a deux fonctionnalités :

1) Représenter une tâche exécutée sur une interface GUI;

2) Prédire le temps d'exécution d'une tâche.

Le KPC sert à représenter une tâche uniquement en terme d'actions explicites au niveau des opérations élémentaires. Il s'agit d'une version simplifiée du KLM. Les trois opérations élémentaires qui sont représentées sont une entrée au clavier (K), un mouvement de pointage (P) et un clic de souris (C). Aucun composant n'est associé à une opération implicite (pause mentale, recherche visuelle, homing, etc.). Le résultat de cette représentation est une séquence de K, P et C (ex. PCKKKKPC).

Pour une séquence *S* donnée, le temps d'exécution prédit $\hat{E}_{KPC}(S)$ dépend du nombre de K, P et C dans la séquence, soit $N_K(S)$, $N_P(S)$ et $N_C(S)$, respectivement. Les temps moyens estimés des K, P et C sont $\hat{E}_K$, $\hat{E}_P$ et $\hat{E}_C$, respectivement. Ces temps ont été déterminés expérimentalement (voir chapitre 5). Ils sont présentés au tableau XII.

Tableau XII

Temps moyens estimés du modèle KPC

Composant	Description	Temps moyen d'exécution (ms)
K	Entrée au clavier	331 ($\hat{E}_K$)
P	Mouvement de pointage	701 ($\hat{E}_P$)
C	Clic de souris	235 ($\hat{E}_C$)

$\hat{E}_{KPC}(S)$ est calculé en utilisant l'équation suivante :

$$\hat{E}_{KPC}(S) = N_K(S)*\hat{E}_K + N_P(S)*\hat{E}_P + N_C(S)*\hat{E}_C \quad (5.1)$$

Le tableau XIII présente un exemple d'application du KPC. Dans cet exemple, l'usager tape son nom d'usager (*« user »*) et son mot de passe (*« passwd »*) dans deux champs de texte séparés. Il clique ensuite sur un bouton de confirmation. La séquence de K, P et C et le temps total prédit sont présentés. Pour une question de simplification, les estimés des temps d'exécution ont été arrondis aux valeurs suivantes : $\hat{E}_K$ = 330 ms, $\hat{E}_P$ = 700 ms et $\hat{E}_C$ = 235 ms.

Tableau XIII

Exemple de modèle KPC

Séquence	Temps moyen d'exécution (ms)	Description
P	700	Place la souris sur le premier champ
C	235	Clique dans le premier champ

Tableau XIII (suite)

Séquence	Temps moyen d'exécution (ms)	Description
4K	4 * 330 = 1320	Tape la chaîne suivante: "user"
K	330	Change de champ en appuyant sur la touche *TAB*
6K	6 * 330 = 1980	Tape la chaîne suivante: "passwd"
P	700	Place la souris sur le bouton de confirmation
C	235	Clique sur le bouton de confirmation
$\hat{E}_{KPC}(S)$	**5500**	

4.2 Application du KPC à un script AIR

Le modèle KPC peut être utilisé afin de prédire le temps d'exécution d'un script AIR. La méthodologie est la suivante : 1) Représenter chaque action explicite (ayant un effet direct sur l'interface) du script par l'opération K, P ou C correspondante sans tenir compte des changements d'état interne ni de la rétroaction. Le résultat est la séquence KPC. 2) Calculer le temps d'exécution en utilisant les estimés arrondis et l'équation 5.1. Un exemple d'application de cette approche est illustré au tableau XIV. Dans cet exemple, les colonnes *State* et *Feedback* sont omises. L'exemple se traduit par le déplacement d'une icône de fichier à l'écran.

Tableau XIV

Exemple d'utilisation du KPC avec un script AIR

Step	Action	Duration	Description
1	mouse.goto(icon)	P	Positionne la souris sur l'icône nommée *icon*.
2	mouse.press	C	Appuie sur le bouton de gauche de la souris. (Début de la tâche de déplacement).

Tableau XIV (suite)

Step	Action	Duration	Description
3	mouse.goto(100,300)	P	Déplace la souris à la position absolue suivante : x_{abs}=100, y_{abs}=300.
4	mouse.release	C	Relâche le bouton de gauche de la souris. (Fin de la tâche de déplacement).
	Temps total (ms)	**1870**	

Actions implicites

Les actions implicites (ex. déplacement de la main, recherche visuelle, réflexion, hésitation) ne sont pas modélisées par le KPC. Il sera vu dans l'étude du chapitre 5 que leur temps d'exécution est inclus dans les estimés de ceux des actions explicites (K, P et C). Si une spécification AIR contient des actions implicites, elles n'apparaissent pas dans la séquence et ne sont donc pas utilisées de façon concrète dans le calcul du temps d'exécution.

Boucles et appels de sous-scripts

Dans le cas d'une boucle (*for, while, repeat*), la séquence comprise entre les délimiteurs de début et de fin de la boucle est multipliée par le nombre d'itérations (ex. nbIterations*(PCKKKKPC)). Pour l'appel d'un sous-script, la séquence du script appelé est additionnée à la séquence du script appelant.

4.3 Processus empirique de construction du KPC

Dans les approches existantes en modélisation analytique, les séquences d'opérations sont définies par l'analyste, soit directement (ex. Kieras, 1993) ou par l'entremise d'un

prototype de l'interface (Hudson et al., 1999, John et al., 2004). Selon l'approche présentée ici, les séquences sont obtenues directement de l'usager.

Les séquences réelles (tel qu'exécutées) sont obtenues suivant un processus empirique de récolte des données (une séquence par journal d'événements). Bien que ce processus requière plus de ressources que certaines autres approches, il permet d'enregistrer toutes les séquences réellement utilisées par l'usager. Ainsi, les séquences qui ont échappé au concepteur et/ou à l'analyste (ex. recherche Web au lieu de l'emprunt des *hyperliens*, Augustine & Greene, 2002) seront tout de même enregistrées.

Parmi les séquences enregistrées, on ne tient compte que des séquences optimales (journaux contenant les séquences optimales) en terme du temps d'exécution (arrondi à la seconde près, par exemple) et/ou du nombre d'opérations. Ainsi, on élimine automatiquement les séquences les moins efficientes et/ou celles contenant des erreurs sans effectuer une analyse d'erreur qui est souvent complexe et qui ne peut être automatisée complètement (Ivory & Hearst, 2001).

L'approche est résumée en trois étapes :

1) Développer un prototype de l'interface qui est équipé d'un programme renifleur.

2) Enregistrer l'usager (à l'aide du renifleur) pendant qu'il effectue une tâche sur le prototype. L'enregistrement des activités se retrouve dans un journal d'événements.

3) Filtrer le journal afin d'extraire le modèle (séquence) KPC de la tâche telle qu'exécutée.

Dans l'approche de construction d'un KLM (Card et al., 1980), la séquence est obtenue par modélisation analytique à partir de la connaissance à priori de la tâche. La figure 5 illustre la différence entre les processus de construction du KPC et du KLM.

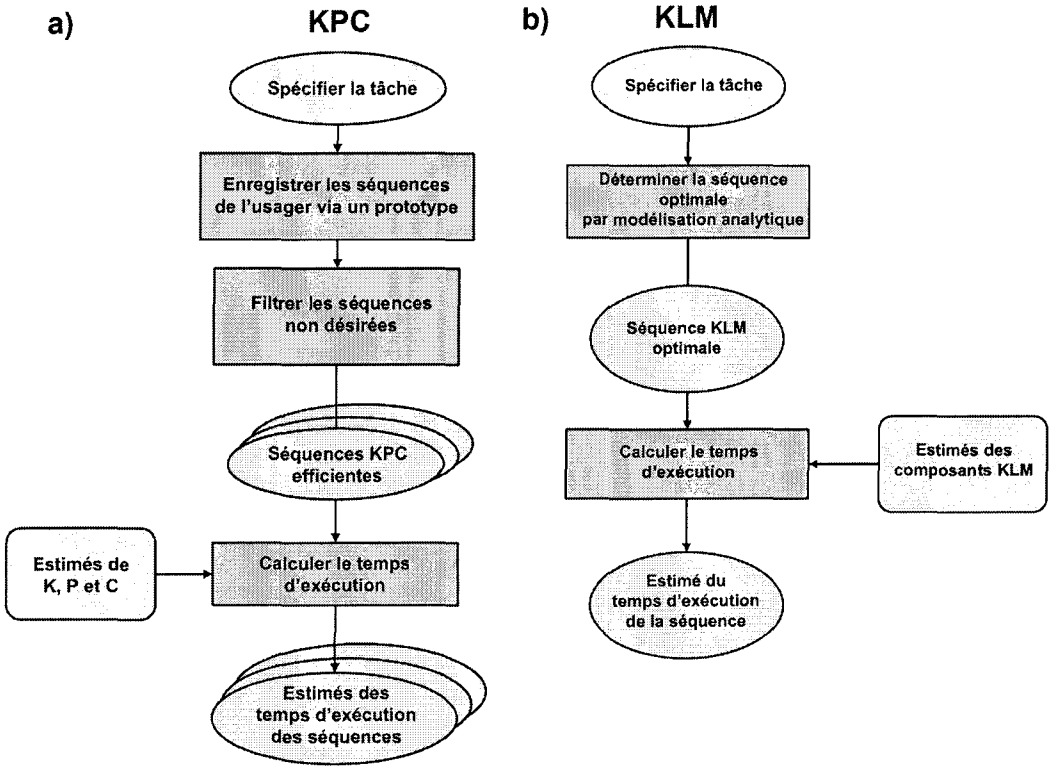


Figure 5 Processus de construction des modèles KPC et KLM

Dans ce chapitre, nous avons présenté un modèle du même style que le KLM, soit le KPC. Ce dernier permet de représenter une tâche exécutée par un usager habile sur une interface GUI à l'aide de trois opérations, soit K, P et C. Il permet également d'obtenir une prédiction du temps d'exécution à partir des estimés de chaque opération. On a également montré qu'il est possible d'extraire un modèle KPC à partir d'un script AIR. Les outils logiciels qui ont été présentés au chapitre 3 permettent l'extraction de façon automatique.

Finalement, on a montré que le KPC s'applique sur la séquence d'opérations réellement exécutées par l'usager. Ainsi, on a introduit un processus empirique de construction du KPC qui est for utile lorsque plusieurs séquences optimales différentes sont exécutées par l'usager. Les estimés des temps d'exécution des opérations du KPC ont été déterminés et la précision de prédiction du modèle validée dans le cadre d'une étude expérimentale. Cette étude fait l'objet du prochain chapitre.

CHAPITRE 5

MISE AU POINT ET VALIDATION EXPÉRIMENTALE DU KPC

Ce chapitre présente la mise au point et la validation expérimentale du modèle KPC. Dans un premier temps, on présente le processus de mise au point du KPC. La méthodologie de mise au point, les conditions expérimentales, les analyses et les résultats sont présentés. Ensuite, on présente la validation du modèle. Les hypothèses de validation, les analyses ainsi que les résultats obtenus sont présentés. Finalement, le modèle KPC est discuté.

5.1 Mise au point du KPC

Une étude expérimentale sur 20 sujets en conditions contrôlées a été effectuée. Les sujets ont été enregistrés durant 40 exécutions de 6 différentes tâches sur interfaces GUI. Les données des dix premiers sujets ont été utilisées pour quantifier les composants du modèle. Les prochaines sections présentent la méthodologie utilisée pour la mise au point, les conditions expérimentales, les analyses effectuées et les résultats obtenus.

5.2 Méthodologie de mise au point

Le KPC a été mis au point en suivant quatre étapes : 1) Développer un prototype d'interface pour 6 différentes tâches. 2) Enregistrer les sujets (1-10) dans leurs dix dernières exécutions (31-40) des six tâches et ce en conditions contrôlées. On prend ces exécutions car c'est à ce moment que les sujets sont à leur mieux (entraînés). 3) Extraire la séquence de chaque exécution de la tâche. Contrairement au processus de construction du KPC présenté au chapitre 4, aucun filtrage des séquences non efficientes n'est utilisé lors de la mise au point. 4) Moyenner les temps d'exécution des opérations K, P et C.

Ces moyennes représentent les estimés (non arrondis) du KPC (voir chapitre 4). La figure 6 illustre cette méthodologie.

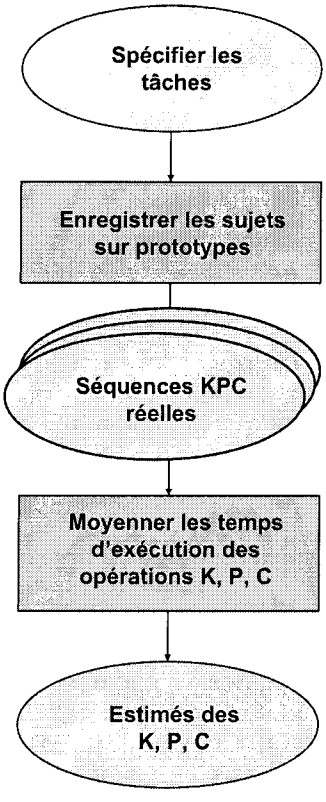


Figure 6 Mise au point du modèle KPC

5.3 Conditions expérimentales

Vingt sujets (âge moyen: 25 ans (écart-type σ = 3.7), tous droitiers, 4 femmes, niveau d'études complétées : collégial ou plus) ont participé à l'étude. Tous les sujets ont rempli un questionnaire (fiche de renseignements personnels; voir annexe 11) en rapport avec leur expérience avec les ordinateurs. Afin d'être considérés comme habiles (*« skilled »*), ils devaient rencontrer les critères d'inclusion suivants :

- L'usage hebdomadaire d'un ordinateur doit avoir été supérieure à 10 fois durant la dernière année.
- L'auto qualification quant à la maîtrise du système d'exploitation MS Windows® doit être supérieure à la moyenne.

Les critères d'exclusion utilisés furent les suivants :

- Problème auditif non corrigé.
- Problème visuel non corrigé.
- Problème moteur.
- Autres problèmes d'ordre neurologique.

Afin de respecter les règles d'éthique de l'ÉTS, les sujets ont rempli un formulaire de consentement avant de participer à l'étude. Le patron de ce formulaire est présenté à l'annexe 11.

Toutes les séances de test ont été effectuées sur un ordinateur fixe (Pentium 4 (2 GHz), 512 MB de mémoire RAM (*« Random access memory »*), Système d'exploitation : *Windows XP Professional®*, écran LCD (*«Liquid crystal display »*) de 17'', clavier standard, souris optique à 3 boutons). Les sujets étaient assis sur une chaise ergonomique conçue pour ce type de test. Chaque sujet devait exécuter six tâches simples (tableau XV). Une série de pratique (tâches 1 à 6 dans l'ordre) était à la disponibilité des sujets. Par la suite, ils devaient effectuer la même série à 40 reprises. Le temps d'exécution moyen de la tâche la plus courte était d'environ 19 secondes tandis que celui de la plus longue était d'environ 35 secondes (tableau XV).

Chacune des 6 tâches était réalisable via une interface de type GUI possédant le *« look-and-feel »* de *Java Swing* (ensemble de paquetages Java permettant la conception d'interfaces GUI). Toutes les tâches sans exception débutaient avec la presse du bouton

de départ situé sur le premier écran de l'application et se terminaient avec la presse du bouton de confirmation situé sur le dernier écran. Ceci était exigé afin de fixer les points de départ et de fin de tâche pour tous les sujets. De cette façon, on favorisait l'obtention d'un temps d'exécution précis. Les informations à saisir étaient prédéfinies ou spécifiées dynamiquement au sujet en cours de tâche. Advenant une erreur de l'usager en cours d'exécution, un message d'erreur générique était affiché ne spécifiant pas la nature ni la localisation de l'erreur. Dans le cas d'une erreur non corrigée, il était impossible pour le sujet de poursuivre à l'écran suivant ou de terminer l'application s'il était au dernier écran au moment de l'erreur.

Les sujets disposaient de consignes écrites sur papier pour toutes les tâches et celles-ci étaient placées sur le bureau de test afin qu'il soit facile pour l'expérimentateur d'identifier les consultations. Il était permis aux sujets de consulter ces consignes avant ou durant l'exécution, si nécessaire. Dans le but de ne pas faciliter l'apprentissage des consignes, celles-ci étaient placées dans des fiches papier qui devaient être refermées après chaque consultation. Les consignes pour les différentes tâches ainsi que les diagrammes de transition et les captures d'écrans des interfaces de test sont présentés à l'annexe 11.

Tableau XV

Description sommaire des tâches de test

Tâche	Description	Temps moyen d'exécution (secondes)	Nombre moyen d'opérations KPC
1	Remplir un formulaire de paiement en ligne.	35	88
2	Copier et coller une zone de texte. Ensuite, modifier le texte copié.	25	54

Tableau XV (suite)

Tâche	Description	Temps moyen d'exécution (secondes)	Nombre moyen d'opérations KPC
3	Installer une application sur un ordinateur.	19	49
4	Sauvegarder un document, l'imprimer et quitter l'application.	22	58
5	S'enregistrer sur une base de données et effectuer une recherche bibliographique.	20	45
6	Consulter une liste d'étudiants travaillant pour un professeur particulier.	19	49

5.4 Saisie et traitement des données

Saisie et filtrage des données d'utilisation

Les actions élémentaires des usagers ont été enregistrées dans des journaux d'événements. Afin de faciliter la lecture des séquences, ces journaux furent filtrés afin de ne conserver que les événements correspondant aux actions élémentaires suivantes: presse d'une clé du clavier (*« key typed »*), mouvement de la souris (*« mouse moved »*) ainsi qu'une presse (*« mouse pressed »*) et un relâchement (*« mouse released »*) d'un bouton de souris. La précision d'enregistrement était d'environ 15 ms.

Détermination des séquences d'opérations K, P et C

Une fois la saisie et le filtrage des données d'utilisation effectués, un traitement approprié a été réalisé afin de convertir les données brutes en données prêtes pour analyse. Le traitement choisi consistait à extraire, pour chaque journal, la séquence KPC correspondant à l'exécution de la tâche. Cette séquence était ajoutée au journal dans une colonne additionnelle. Les règles d'extraction qui furent utilisées sont les suivantes :

- K : le symbole K fût ajouté dans la colonne vis-à-vis chaque événement de type *key_typed*;

- P : les mouvements de pointage de la souris étaient identifiés par des blocs d'événements de type *mouse_moved* consécutifs. Pour chaque bloc, un symbole P était ajouté vis-à-vis le dernier événement du bloc. Pour ce qui est des pointages de longueur inférieure ou égale à 1 pixel, aucune association dans la séquence n'était effectuée;

- C : dans chaque cas où un événement de type *mouse_pressed* était immédiatement suivi d'un événement de type *mouse_released*, un symbole C était ajouté vis-à-vis le *mouse_released*. Il s'agit d'un clic de la souris. Dans le cas où un autre événement (*mouse_moved* ou *key_typed*) était présent entre les événements *mouse_pressed* et *mouse_released*, un C était ajouté pour chacun d'eux et le symbole de l'événement intermédiaire également. Un bon exemple de ceci est le déplacement d'une icône connu dans le domaine sous le nom de *« drag-and-drop »*.

Pour ce qui est de la durée d'une opération, elle était déterminée en soustrayant l'instant (*« time stamp »*) de l'opération précédente de celui de l'opération courante. Le tableau XVI présente un exemple de journal à la suite du traitement. La description a été

rajoutée pour fin de compréhension. Dans cet exemple, l'usager tape son nom d'usager, son mot de passe et clique sur le bouton de confirmation.

Tableau XVI

Exemple d'une séquence KPC extraite d'un journal d'événements

Opération	Argument	Instant (ms)	Durée (ms)	Description
P	100,200	10210	655	Pointage sur loginChamp (position=100,200)
C	loginChamp	10450	240	Clic sur loginChamp
K	«l»	10751	301	Saisie de caractère
K	«o»	11061	310	Saisie de caractère
K	«g»	11339	278	Saisie de caractère
K	«i»	11628	289	Saisie de caractère
K	«n»	11887	259	Saisie de caractère
K	Tab	12576	689	Presse sur Tab
K	«p»	12976	400	Saisie de caractère
K	«a»	13284	308	Saisie de caractère
K	«s»	13739	455	Saisie de caractère
K	«s»	14607	868	Saisie de caractère
K	«w»	15108	501	Saisie de caractère
K	«d»	15456	348	Saisie de caractère
P	250,300	16326	870	Pointage sur okBouton (position=250,300)
C	okBouton	16545	219	Clic sur okBouton

5.5 Détermination des valeurs du KPC

Les estimés des temps d'exécution des opérations K, P et C furent déterminés avec des sujets habiles et entraînés avec l'interface. Ces estimés sont notés $\hat{E}_K$, $\hat{E}_P$ et $\hat{E}_C$, respectivement. Ils ont été déterminés en moyennant les temps d'exécution des opérations (K, P et C) de chaque séquence sur le groupe suivant : sujets 1-10, tâches 1-6 et essais 31-40. La stabilité de ces estimés a été étudiée en comparant les temps d'exécution moyens et les écarts-types obtenus pour les groupes de 5, 10, 15 et 20 sujets.

5.6 Résultats – estimés des temps d'exécution des opérations du KPC

Les estimés du modèle KPC qui ont été obtenus sont les suivants : $\hat{E}_K = 331$ ms, $\hat{E}_P$=701 ms et $\hat{E}_C$=235 ms. Le tableau XVII présente les distributions complètes (écart type et nombre d'observations). Les temps moyens d'exécution ne varient pas de façon significative en fonction du nombre de sujets utilisés pour calculer les valeurs. De plus, l'écart-type n'a pas augmenté en fonction d'un nombre de sujets plus grand. Ceci vient valider le choix de prendre 10 sujets pour établir les valeurs du modèle KPC. Les résultats de cette étude sur la stabilité sont présentés au tableau XVIII où les moyennes sont établies sur les tâches 1-6 et les essais 31-40. Dans ce tableau, le symbole Δ représente la différence relative avec les valeurs du tableau XVII.

On peut constater que l'écart-type est considérable pour les trois temps d'exécution, et parfois supérieur au temps moyen. On verra plus loin que ces écarts-types sont dus à la variabilité entre les sujets, entre les tâches et celle causée par la pratique et le contexte d'exécution. Par contre, la variabilité se cancelle lorsque l'on additionne les estimés pour prédire le temps d'exécution moyen d'une séquence. Malgré cette variabilité considérable sur les opérations élémentaires K, P et C, l'erreur de prédiction ne dépasse pas 26 % et ce même dans le cas des sujets naïfs avec la tâche (essais 1-10).

Tableau XVII

Temps moyens d'exécution du modèle KPC

Opération	Temps d'exécution moyen (ms)
K	331 écart-type (σ) = 348 nombre d'observations (n) = 22147
P	701 σ = 677, n = 5433
C	235 σ = 301, n = 5159

Tableau XVIII

Temps moyens d'exécution des opérations K, P et C obtenus avec 5, 10, 15 et 20 sujets

	Sujets 1-5	Sujets 1-10 (référence pour Δ)	Sujets 1-15	Sujets 1-20
K	301 , Δ=9% σ=307, n=11339	331 , ----- σ=348, n=22147	334 , Δ=1% σ=356, n=33571	325 , Δ=2% σ=357, n=44829
P	660 , Δ=6% σ=503, n=660	701 , ----- σ=677, n=5433	703 , Δ=0% σ=749, n=8046	689 , Δ=2% σ=750, n=11648
C	232 , Δ=1% σ=314, n=2338	235 , ----- σ=301, n=5159	229 , Δ=3% σ=306, n=7611	208 , Δ=11% σ=292, n=10924

5.7 Validation du KPC

La même expérience que la mise au point a permis de valider le modèle KPC. Les données des dix derniers sujets (11-20) ont été utilisées. Les prochaines sections présentent les hypothèses de validation, les analyses effectuées et les résultats obtenus.

5.8 Hypothèses de validation

La validité du modèle KPC dépend de la confirmation de trois hypothèses. Elles sont les suivantes :

1) Un usager habile utilise une panoplie de séquences considérées comme optimales dans l'exécution de la tâche prescrite. Dans le cas où seulement une séquence optimale est exécutée, cette dernière est déterminée analytiquement évitant ainsi un processus empirique visant à mesurer les séquences réelles. La figure 5 (chapitre 4) illustre bien cette différence.

2) La variabilité entre les sujets, les tâches et les niveaux d'entraînement (ou de pratique) ne doit pas avoir un effet significatif sur les temps d'exécution moyens des opérations K, P et C. Le cas échéant, les valeurs du modèle représenteront un estimé significatif quel que soit l'usager, la tâche et le niveau de pratique.

3) Le contexte et la présence d'opérations implicites ne doivent pas avoir un effet significatif sur les temps moyens d'exécution. Par exemple, il se peut que le temps d'exécution du premier caractère d'une chaîne saisie soit un peu plus long que celui des autres caractères dû au fait qu'il est possible que ce caractère soit précédé d'une pause mentale.

5.9 Analyse de données sur les opérations K, P et C

5.9.1 Détermination des séquences optimales

Les séquences optimales furent déterminées pour chaque tâche selon deux critères : 1) Le temps minimal d'exécution. 2) Le nombre minimal d'opérations. En ce qui a trait au temps d'exécution, les essais optimaux furent déterminés comme ceux ayant un temps d'exécution minimal. Les temps d'exécution furent arrondis à la seconde près. Les différentes séquences furent déterminées comme les séquences optimales. Pour le temps d'exécution d'une séquence KPC, l'ordre des opérations n'est pas important. En fait, les séquences ayant le même nombre d'opérations K, P et C sont considérées identiques. Ainsi, de nouvelles séquences appelées séquences optimales génériques furent déterminées en fonction de leur composition en terme du nombre de K, P et C. Pour ce qui est du nombre d'opérations, les essais optimaux furent déterminés comme ceux ayant un nombre minimal d'opérations. Ensuite, les séquences optimales génériques furent déterminées de la même façon que dans le cas du temps d'exécution. Les résultats sont présentés aux tableaux XX et XXI et illustrés à la figure 7 de la section 5.11.

5.9.2 Effet de la tâche, du sujet et de la pratique sur les temps d'exécution du KPC

Le but ici était de déterminer l'exactitude des estimés $\hat{E}_K$, $\hat{E}_P$ et $\hat{E}_C$ lorsqu'ils sont déterminés à partir de tâches et de sujets différents. Ainsi, les effets de la tâche, du sujet et de l'entraînement sur les estimés du temps d'exécution des opérations K, P et C ont été déterminés sur tous les sujets (1-20) pour toutes les tâches (1-6) et tous les essais (1-40). À cette fin, des analyses de variances (ANOVA, *« analysis of variance »*) ont été effectuées avec le logiciel SPSS 12.0. De telles analyses servent à mesurer la différence entre les moyennes de deux ou plusieurs échantillons (ou groupes de données). Les résultats de ces analyses sont présentés aux tableaux XXII et XXIII de la section 5.12. L'amplitude de chaque effet fût déterminée en moyennant la différence entre les moyennes marginales de chaque paire de valeurs. Pour ce qui est de l'entraînement, il a été catégorisé en quatre groupes ou niveaux, soit les groupes d'essais 1-10, 11-20, 21-30 et 31-40. Les différences de temps d'exécution pour les différents niveaux furent étudiées pour vérifier quand l'entraînement avait lieu.

5.9.3 Effet du contexte sur les temps d'exécution du KPC

Dû à la présence de certaines opérations implicites (homing, pauses mentales, etc.), il est possible que les temps moyens d'exécution des opérations K, P et C ne suivent pas une distribution normale, même pour des sujets habiles et entraînés. En fait, les opérations implicites ont tendance à se produire que dans des contextes particuliers. Voici deux exemples : 1) Une activité mentale qui se produit avant la saisie du premier caractère d'une chaîne rendant le temps d'exécution de ce caractère probablement plus élevé que celui des autres caractères saisis. 2) À la suite d'un déplacement de la main dominante du clavier vers la souris (*« homing »*) afin de pointer, le temps d'exécution du *« homing »* pourrait être inclus dans celui du mouvement de la souris.

Un contexte est défini ici comme étant l'opération précédant l'opération courante. Puisque le modèle KPC comprend trois opérations, chacune d'elle peut se retrouver dans trois contextes différents, sauf exception (voir tableau XIX). Il y a donc neuf combinaisons possibles. Afin de pouvoir observer l'effet de ces actions implicites, les distributions des temps d'exécution des opérations K, P et C ont été obtenues (sujets 1-10, tâches 1-6 et essais 31-40) dans tous les contextes possibles. Ces distributions possèdent chacune un estimé du temps d'exécution moyen ($\hat{E}_{X,Y}$: temps d'exécution de l'opération Y lorsque précédée par l'opération X). Ces estimés constituent les composants du modèle *context-dependent* KPC. Les différents contextes sont expliqués dans le tableau XIX qui suit. Dans ce tableau, une rangée représente le contexte et une colonne représente l'opération courante. Chaque cellule contient un exemple potentiel pour une interface GUI et les opérations implicites pouvant avoir lieu. Les résultats de cette analyse sont présentés à la figure 8 et au tableau XXIV de la section 5.13.

Tableau XIX

Description des différents contextes du *context-dependent* KPC

	K	P	C
K	Transcription de texte dans un champ ou saisie de commandes à la console	Déplacement de la souris après une action au clavier. Opération(s) implicite(s) possible(s): Homing et activité mentale	Clic de la souris après une action au clavier sans bouger la souris. Opération(s) implicite(s) possible(s): Homing et activité mentale

Tableau XIX (suite)

	K	P	C
P	Relâchement de la souris et transcription de texte ou saisie de commandes à la console. Opération(s) implicite(s) possible(s): Homing et activité mentale	Impossible	Déplacement de la souris et clic sur un composant. Opération(s) implicite(s) possible (s): Activité mentale
C	Clic de la souris sur une zone de texte. Ensuite, il y a une transcription de texte. Opération(s) implicite(s) possible(s): Homing et activité mentale	Clic de la souris sur un composant et déplacement de la souris. Opération(s) implicite(s) possible(s): Activité mentale	Clic double

5.10 Analyse de données sur les performances du KPC

5.10.1 Précision de prédiction du KPC

Dans le processus de prédiction, les séquences d'opérations réelles des journaux d'événements furent utilisées. Afin d'obtenir le temps d'exécution $\hat{E}_{KPC}$ prédit pour une séquence donnée, les temps d'exécution de toutes les opérations de la séquence furent additionnés. Pour ce faire, $\hat{E}_K$, $\hat{E}_P$ et $\hat{E}_C$ furent utilisés. Deux types de prédictions ont été identifiés afin de pouvoir établir les limites du modèle. Dans un premier temps, on prédit les temps d'exécution pour des usagers habiles et naïfs (dix premiers essais). Ensuite, on fait de même pour des usagers habiles et entraînés (dix derniers essais).

Le groupe constitué des 10 premiers sujets était le groupe de référence tandis que les sujets 11 à 20 constituaient le groupe sous prédiction. Les essais 1 à 10 ont été utilisés pour la prédiction des usagers naïfs tandis que pour les usagers entraînés, les essais 31 à 40 furent utilisés. Afin de déterminer la précision de la prédiction, la différence entre le temps d'exécution prédit $\hat{E}_{KPC}$ et le temps d'exécution réel (TE) fût calculée de deux façons. Dans un premier temps, l'erreur de prédiction sur un essai unique, $abs(TE-\hat{E}_{KPC})$, a été calculée. Cette différence fût moyennée pour les sujets habiles naïfs et entraînés, séparément. Ensuite, le biais (différence entre les moyennes) fût calculé pour les sujets habiles naïfs et entraînés. Le biais indique si la prédiction est pessimiste (biais positif) ou optimiste (biais négatif). Les figures 9 et 10 ainsi que le tableau XXV de la section 5.14 présentent les résultats obtenus pour la précision de prédiction du modèle KPC.

5.10.2 Comparaison avec le *context-dependent* KPC

Le but de cette comparaison était de déterminer si les estimés du modèle *context-dependent KPC* ($\hat{E}_{X,Y}$) améliorent la précision de la prédiction. Ainsi, le $\hat{E}_{c\text{-}dKPC}$ d'une séquence donnée est obtenu en additionnant les estimés des temps d'exécution de tous les contextes ($\hat{E}_{X,Y}$) de la séquence. Lorsqu'une opération n'avait pas lieu dans un des trois contextes (ex. première opération de la séquence), l'estimé du temps d'exécution de cette opération ($\hat{E}_K$, $\hat{E}_P$ ou $\hat{E}_C$) était utilisé. Afin de mesurer la précision du modèle, la méthode utilisée fût la même que dans le cas du KPC. Les résultats sont également présentés aux figures 9 et 10 et au tableau XXV.

5.10.3 Comparaison avec le KLM

Afin de comparer le KPC avec le KLM, la précision de ce dernier fût étudiée de la même façon que pour le KPC. Les données prédictives utilisées ainsi que les règles d'insertion des opérations implicites (M et H) sont les mêmes que celles présentées dans Hudson et al. (1999). Puisque nous avons mesuré les séquences de façon expérimentale, la prédiction du KLM est effectuée sur la tâche réelle et non la tâche optimale. Pour une

séquence donnée, $\hat{E}_{KLM}$ est obtenu en additionnant les temps d'exécution de toutes les opérations M, H, P, C et K (voir Hudson et al., 1999) constituant la séquence. Pour ce qui est de la précision, la méthode utilisée fût la même que dans les cas du KPC et du *context-dependent KPC*. Les résultats sont également présentés aux figures 9 et 10 et au tableau XXV.

5.10.4 Effet de la tâche, du sujet et de la pratique sur la précision de prédiction du KPC

Les effets de la tâche, du sujet et de l'entraînement sur la précision de prédiction ont été déterminés en utilisant les données des sujets 11 à 20 pour toutes les tâches (1-6) et les essais 1 à 10 pour les sujets habiles et naïfs et les essais 31 à 40 pour les sujets habiles et entraînés. Des ANOVA ont été effectuées et l'amplitude de chaque effet fût déterminée en moyennant la différence entre les moyennes marginales de chaque paire de valeurs. Ceci fût également effectué avec SPSS 12.0. Il ne s'agit pas ici du même type d'analyse qu'à la section 5.9.2. Les résultats sont présentés aux tableaux XXVI et XXVII de la section 5.15.

5.11 Résultats – séquences optimales

On présente ici les résultats provenant de l'analyse présentée à la section 5.9.1. On rappelle que les séquences optimales ont été déterminées selon le temps d'exécution et le nombre d'opérations présentes dans une séquence. Les séquences optimales en terme de temps d'exécution ont été déterminées pour l'ensemble des données recueillies (sujets 1-20, tâches 1-6 et essais 1-40). Le tableau XX présente ces résultats pour les six tâches séparément. Le nombre de séquences optimales dites génériques (voir section 5.9.1) est entre 15 et 29. Ceci démontre que les sujets ont utilisé plusieurs séquences optimales et ce pour chacune des tâches exécutées. Cependant, les essais optimaux en temps ne représentent qu'entre 3.4 % et 7.5 % des essais effectués.

Tableau XX

Nombres de séquences optimales en terme du temps d'exécution pour chaque tâche

Tâche	Séquences optimales génériques en temps	Séquences optimales en temps	Essais optimaux en temps	% d'essais optimaux
1	21	21	27	3.4%
2	19	23	42	5.3%
3	22	38	60	7.5%
4	29	33	37	4.6%
5	20	24	36	4.5%
6	15	26	50	6.3%

Pour ce qui est du nombre d'opérations, les séquences optimales génériques (déterminées à partir des mêmes données) sont entre 8 et 16. Le tableau XXI présente ces résultats pour les six tâches séparément. Comme pour le cas du temps d'exécution, on constate que les sujets ont utilisé plusieurs séquences optimales et ce pour chacune des tâches. Cependant, le nombre de ces séquences est inférieur à celui des séquences optimales en temps. Ainsi, il est possible que certains essais soient optimaux en terme de temps d'exécution tout en ne l'étant pas en terme du nombre d'opérations. Le contraire est également possible. On remarque également que les essais optimaux en terme du nombre d'opérations représentent une plus grande proportion des essais exécutés que ceux optimaux en terme du temps d'exécution.

Tableau XXI

Nombres de séquences optimales en terme du nombre d'opérations pour chaque tâche

Tâche	Séquences optimales génériques en nombre d'opérations	Séquences optimales en nombre d'opérations	Essais optimaux en nombre d'opérations	% d'essais optimaux
1	14	14	62	7.8%
2	8	9	127	15.9%
3	10	15	59	7.4%

Tableau XXI (suite)

Tâche	Séquences optimales génériques en nombre d'opérations	Séquences optimales en nombre d'opérations	Essais optimaux en nombre d'opérations	% d'essais optimaux
4	16	16	40	5.0%
5	9	16	118	14.8%
6	5	8	71	8.9%

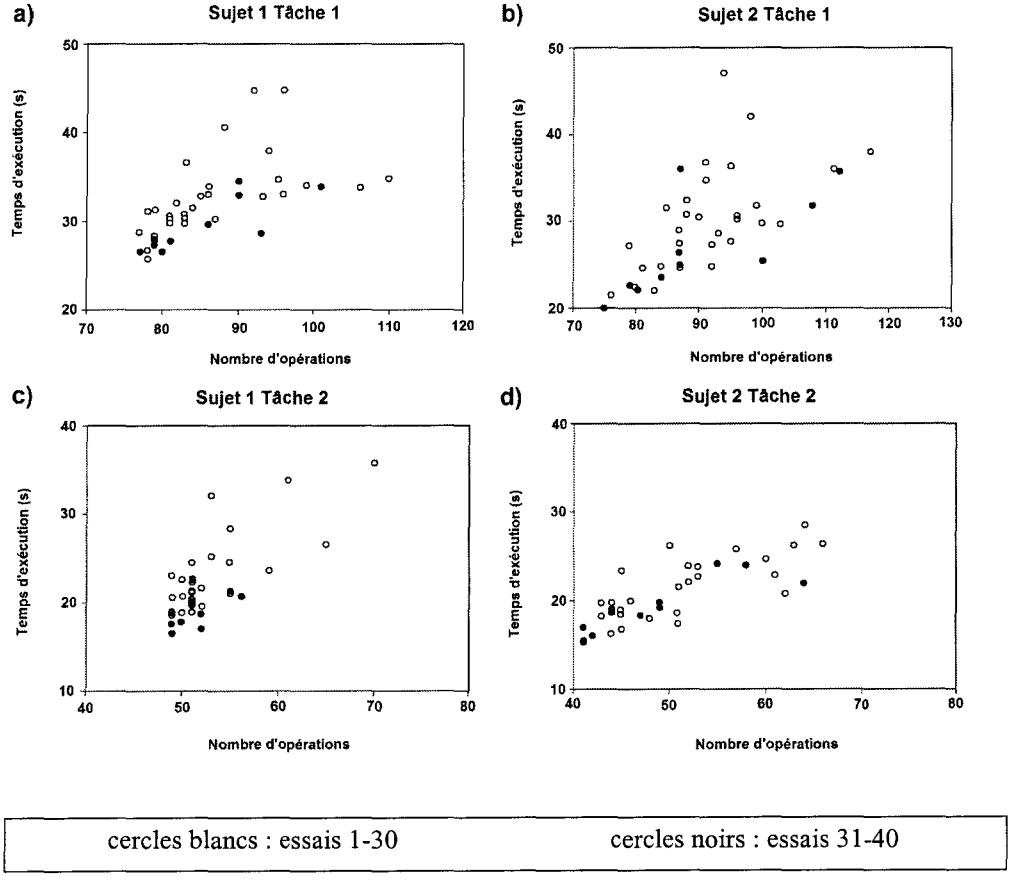


Figure 7 Distribution des essais pour les deux premiers sujets et les deux premières tâches

Les résultats des tableaux XX et XXI sont illustrés à la figure 7 pour les deux premières tâches des deux premiers sujets. Dans cette figure, chaque cercle représente un essai. On

constate que quelques essais sont optimaux selon les deux critères et que certains des dix derniers essais ne sont pas optimaux. Le dernier point peut être justifiée par la fatigue des sujets à ce stade du test.

5.12 Résultats – effets de la tâche, du sujet et de l'entraînement sur les temps d'exécution du KPC

Les résultats présentés ici sont directement reliés à l'analyse expliquée à la section 5.9.2. On rappelle que le but de cette analyse était d'étudier la variation des estimés du KPC en fonction de la tâche, du sujet et de la pratique. Les effets de la tâche, du sujet et de l'entraînement sur les temps d'exécution du modèle KPC ont tous été significatifs ($p<0.0005$). Le facteur p (coefficient de confiance) d'une ANOVA spécifie si la différence entre les moyennes de deux groupes ou plus est significative. Les effets croisés (interactions de ces facteurs) ont également été significatifs excepté pour celui de l'interaction entre la tâche et l'entraînement pour l'opération P. Les effets sont présentés au tableau XXII (**: significatif avec $p<0.0005$. *ns*: non significatif avec $p=0.1$). Ces effets sont présentés sous la forme du facteur F d'une ANOVA, c'est-à-dire la distribution de Fisher-Snedecor qui donne la variance d'un échantillon de taille t qui suit une loi $N(\mu,\sigma^2)$ où μ et σ^2 sont la moyenne et la variance de l'échantillon, respectivement. La présence d'astérix (**) signifie que le F est significatif, c'est-à-dire que la différence des moyennes est significative. Les amplitudes de ces effets se sont étalées de 9 % à 36 % du temps d'exécution. Elles sont présentées au tableau XXIII (gras: moyenne des différences entre moyennes marginales pour les paires de valeurs divisée par le temps moyen d'exécution obtenu sur les sujets 1-20, tâches 1-6 et essais 1-40).

Une étude post-hoc (de type *Dunnett T3*) sur l'entraînement a été effectuée avec SPSS 12.0. Ce genre d'étude spécifie, en plus de la présence d'une différence significative (s'il y a lieu), quels échantillons (ou groupes de données) sont différents les uns des autres et lesquels ne le sont pas. L'entraînement a été étudié en divisant les essais en quatre

groupes, soit les groupes comprenant les essais 1-10, 11-20, 21-30 et 31-40. Les amplitudes de la différence de prédiction entre le premier groupe et les trois autres groupes ainsi qu'entre les trois autres groupes ensembles ont été déterminées. Pour l'opération P, elles ont été de 240 ms et 80 ms, respectivement. Elles ont été de 45 ms et 19 ms pour l'opération K ainsi que de 53 ms et 14 ms pour l'opération C. Ainsi, on constate que l'entraînement s'est majoritairement fait sentir durant les dix premiers essais.

Tableau XXII

Effets de la tâche, du sujet et de l'entraînement sur les temps d'exécution du KPC

	Tâche	Sujet	Entraînement	Tâche x Sujet	Tâche x Entraînement	Sujet x Entraînement
C	F(5, 44559) = 244.78 **	F(19, 44559) = 41.20 **	F(3, 44559) = 50.34 **	F(95, 44559) = 6.47 **	F(15, 44559) = 3.49 **	F(57, 44559) = 2.26 **
K	F(5, 180591) = 336.40 **	F(19, 180591) = 174.89 **	F(3, 180591) = 167.54 **	F(95, 180591) =12.18 **	F(15, 180591) = 4.81 **	F(3, 180591) = 5.13 **
P	F(5, 47727) = 19.90 **	F(19, 47727) = 30.61 **	F(3, 47727) = 197.74 **	F(95, 47727) = 2.84 **	**F(15, 47727) = 1.02 ns**	F(57, 47727) = 3.09 **

Tableau XXIII

Amplitudes des effets de la tâche, du sujet et de l'entraînement sur les temps d'exécution du KPC

	Tâche	Sujet	Entraînement
C	**36%** (91 / 251 ms)	**20%** (51 / 251 ms)	**11%** (28 / 251 ms)
K	**20%** (66 / 340 ms)	**22%** (74 / 340 ms)	**10%** (33 / 340 ms)
P	**9%** 70 / 807 ms	**18%** 144 / 807 ms	**21%** 167 / 807 ms

5.13 Résultats – estimés des temps d'exécution du modèle *context-dependent* KPC

Les résultats présentés ici sont directement reliés à l'analyse expliquée à la section 5.9.3. Dans cette section, on étudie l'effet du contexte sur le temps d'exécution de chaque opération du KPC. La figure 8 illustre les distributions du temps d'exécution de chaque opération dans les différents contextes. Pour chaque distribution, on présente la fréquence en fonction du temps d'exécution (ms). Les distributions ne sont pas présentées pour les combinaisons K-C (nombre insuffisant de valeurs, n=22) et P-P (impossible). On constate que plusieurs distributions sont asymétriques du côté droit. Ceci peut être dû à la présence d'actions implicites, notamment les *« homings »* et les pauses mentales. Les temps moyens $\hat{E}_{X,Y}$ (établis sur les sujets 1-10, tâches1-6 et essais 31-40) sont présentés dans le tableau XXIV. Ces temps sont utilisés lors de la prédiction effectuée avec le modèle *context-dependent* KPC.

Tableau XXIV

Temps moyens d'exécution (ms) du modèle *context-dependent* KPC

	K	P	C
K	312 σ=335, n=21025	1240 σ=459, n=1045	1497 σ=584, n=22
P	644 σ=409, n=549		230 σ=302, n=4884
C	748 σ=395, n=543	555 σ=689, n=3818	216 σ=152, n=253

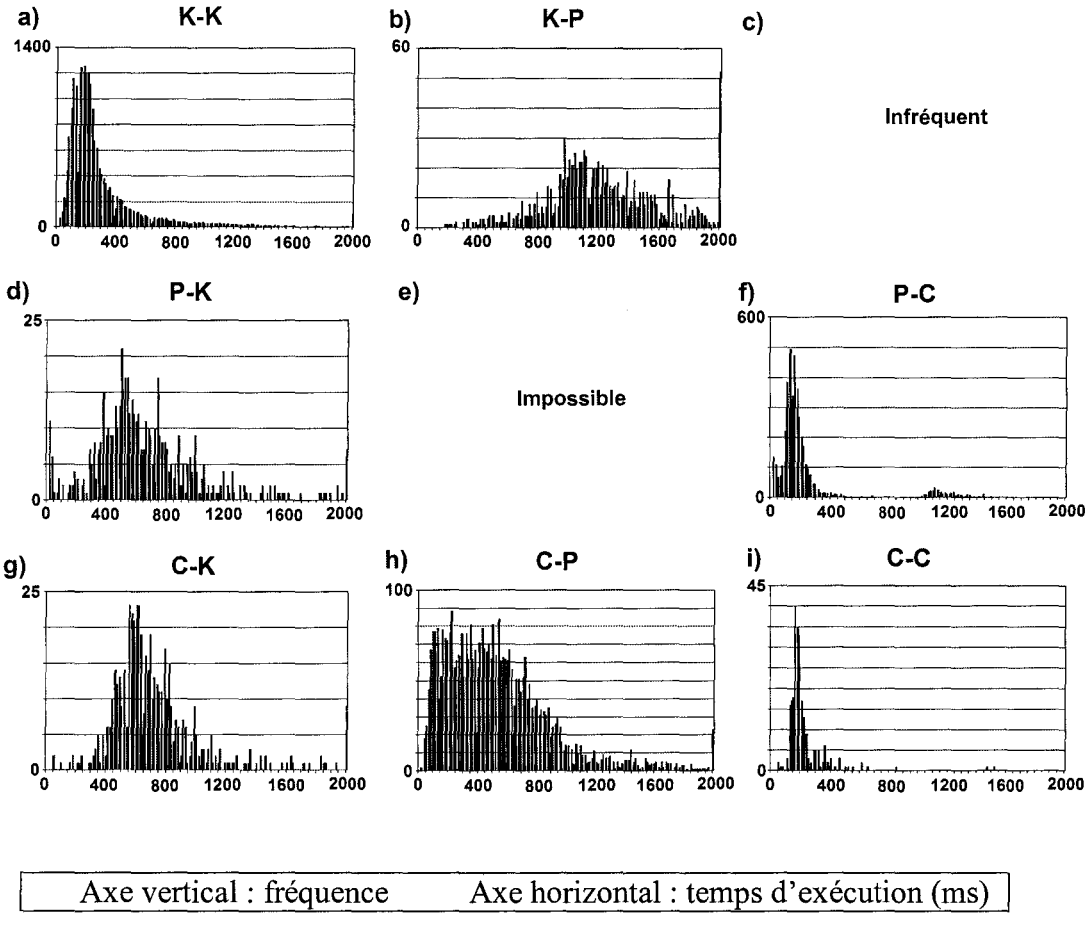


Figure 8 Distributions des temps d'exécution du modèle *context-dependent* KPC

5.14 Résultats – précision des modèles KPC, *context-dependent* KPC et KLM

On présente ici les résultats provenant des analyses présentées aux sections 5.10.1 à 5.10.3, inclusivement. On rappelle que la précision de prédiction a été calculée pour les modèles KPC, *context-dependent* KPC et KLM, et ce de deux manières différentes (erreur moyenne et biais). Selon les résultats obtenus, les modèles KPC et *context-dependent* KPC permettent de prédire le temps d'exécution des usagers habiles naïfs et entraînés avec des marges d'erreur moyennes ne dépassant pas 27 % et 19 %, respectivement. De plus, l'écart-type a été réduit de moitié pour les sujets habiles et entraînés comparativement aux sujets habiles et naïfs.

Pour ce qui est du KLM avec la séquence optimale (en temps d'exécution), les marges d'erreur sont de 33 % et 69 % pour les sujets habiles naïfs et entraînés, respectivement. Pour la séquence réelle mesurée, ces mêmes marges sont de 42 % et 73 %. Selon ces résultats, le KLM est plus précis pour les usagers habiles et naïfs que pour les usagers habiles et entraînés. Ceci est en apparente contradiction avec les objectifs originaux du KLM qui a été conçu pour prédire les performances d'un usager expert (équivalent à un usager habile et entraîné dans cette étude). Globalement, le KLM fournit de moins bonnes prédictions que le KPC et le *context-dependent* KPC. La figure 9 et le tableau XXV illustrent ces résultats.

La figure 10 présente ces mêmes erreurs de prédiction mais mesurées d'une façon différente, soit sous la forme de biais. Un biais représente la différence entre le temps d'exécution réel moyen et la prédiction du modèle, soit la différence des moyennes. Rappelons que l'erreur présentée à la figure 9 représente l'écart entre le temps d'exécution réel d'un essai et celui prédit par le modèle et ce moyenné sur tous les essais. Cette méthode semble fournir des résultats plus représentatifs de l'erreur que le biais puisqu'elle tient compte de la variabilité des séquences exécutées.

En regardant la figure 10, il est possible de constater que les trois modèles sont optimistes (biais négatif) pour les usagers habiles et naïfs. Pour les usagers habiles et entraînés, le KPC et le KLM sont tous les deux pessimistes (biais positif). Cependant, le biais pour le KPC est beaucoup moins grand que celui pour le KLM. Pour ce qui est du *context-dependent* KPC, il est pratiquement non biaisé (très précis).

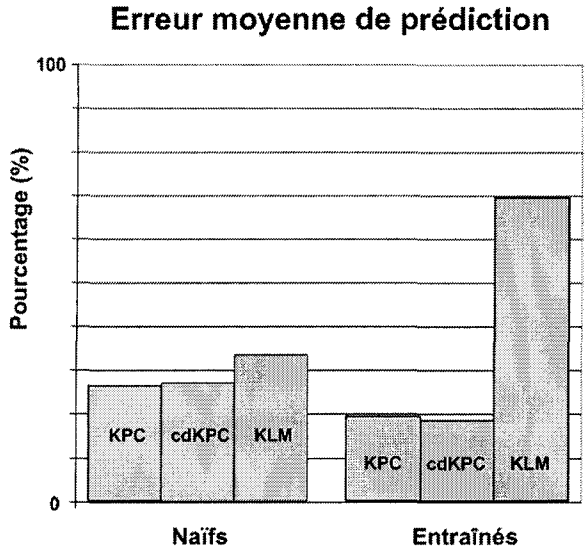


Figure 9 Erreur de prédiction pour les modèles KPC, *context-dependent* KPC (c-dKPC) et KLM sur les séquences réelles

Tableau XXV

Erreur de prédiction pour les modèles KPC, *context-dependent* KPC (c-dKPC) et KLM sur les séquences réelles

	Sujets habiles et naïfs	Sujets habiles et entraînés
	moyenne	moyenne
KPC	26% écart-type (σ)=40	19% σ=20
c-dKPC	27% σ=41	18% σ=20
KLM sur la séquence optimale	33% σ=50	69% σ=29
KLM sur la séquence réelle	42% σ=28	73% σ=38

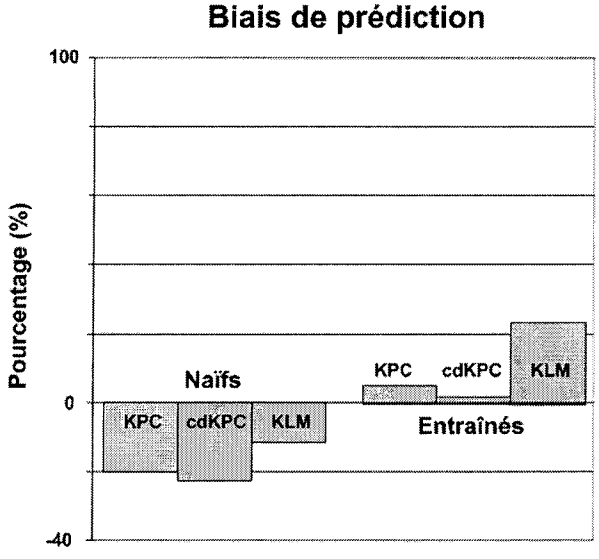


Figure 10 Biais de prédiction pour les modèles KPC, *context-dependent* KPC (c-dKPC) et KLM

5.15 Résultats – effets de la tâche, du sujet et de l'entraînement sur la précision du KPC

Les résultats présentés ici sont directement reliés à l'analyse expliquée à la section 5.10.4. On rappelle que le but de cette analyse était d'étudier la variation de la précision du KPC en fonction de la tâche, du sujet et de la pratique. Les effets de la tâche, du sujet et de l'entraînement ont tous été significatifs ($p<0.0005$) et les effets croisés on été relativement moins significatifs ($p<0.01$). Ces résultats sont présentés au tableau XXVI (**: significatif avec $p<0.0005$, *: significatif avec $p<0.01$, Δ = erreur (abs (TE - $\hat{E}_{KPC}$))). Ces effets sont présentés sous la forme de la distribution de Fisher-Snedecor d'une ANOVA qui est expliquée à la section 5.12. La présence d'un ou de deux astérix spécifie le degré avec lequel la différence des moyennes est significative. Les amplitudes des effets (différences des moyennes) ont été de 32 %, 38 % et 61 % pour la tâche, le sujet et le niveau d'entraînement, respectivement. Ces résultats sont acceptables considérant que le 61 % provient de la grande différence de pratique entre les essais 1 à 10 et 31 à 40. Le tableau XXVII résume ces résultats (gras: moyenne des différences entre moyennes marginales pour les paires de valeurs divisée par le temps moyen

d'exécution sur les sujets 11-20, tâches 1-6 et essais 1-10 (sujets habiles et naïfs) et 31-40 (sujets habiles et entraînés)).

Tableau XXVI

Effets de la tâche, du sujet et de l'entraînement sur la précision du KPC

	Tâche	Sujet	Entraînement	Tâche x Sujet	Tâche x Entraînement	Sujet x Entraîne-ment
Δ	F(5, 1080) = 7.96 **	F(9, 1080) = 8.02 *	F(1, 1080) = 58.14 **	F(45, 1080) = 1.59 *	F(5, 1080) = 4.89 **	F(9, 1080) = 8.24 **

Tableau XXVII

Amplitudes des effets de la tâche, du sujet et de l'entraînement sur la précision du KPC

	Tâche	Sujet	Entraînement
Δ	**32%** (1855 / 5789 ms)	**38%** (2217 / 5789 ms)	**61%** (3524 / 5789 ms)

5.16 Discussion

Modèle basé sur les opérations explicites

En utilisant les séquences réelles (exécutées par l'usager) constituées des opérations K, P et C, la précision de prédiction obtenue n'a pas dépassée 26 % et 19 % pour les sujets habiles naïfs et entraînés, respectivement. Ces résultats montrent qu'il est possible de ne tenir compte que des opérations explicites et de ne pas excéder de façon significative la marge d'erreur habituellement acceptable en modélisation analytique, soit

approximativement 20% (Card et al., 1980;1983, Ivory & Hearst, 2001, Hudson et al., 1999, John & Kieras, 1996a;1996b, Baumeister, John & Byrne, 2000, Kieras et al., 1993;1995, John et al., 2004).

Les estimés des temps moyens d'exécution qui ont été obtenus sont relativement similaires à ceux du KLM d'origine (1980) pour les opérations K et C. Pour ce qui est du P, nous avons obtenu un estimé de 700 ms comparativement à 1100 ms pour le KLM. Un résultat comparable à celui du KPC a été obtenu dans une étude plus récente (que celle de 1980) sur la détection des blocs d'activités mentales (*« mental chunks detection »*), soit 765 ms en moyenne pour des usagers utilisant la souris de façon significative (Santos & Badre, 1994). La différence peut être due à l'évolution des souris informatiques depuis 1980.

Il faut noter que les estimés obtenus n'ont pas changé significativement en fonction du groupe de sujets utilisé pour les calculer. En fait, l'écart-type ne changeait pas significativement lorsqu'on augmentait le nombre de sujets (5, 10, 15 et 20). Une telle observation justifie le choix de procéder de façon itérative afin de déterminer la taille du groupe de sujets participant à l'étude. Ainsi, une limite inférieure est choisie a priori et on incrémente le nombre de sujets jusqu'à ce que l'écart-type ne diminue plus de façon significative. Il a également été démontré dans cette étude que le modèle KPC n'est sensible que de façon modérée en ce qui a trait à la variabilité entre sujets, tâches et niveaux d'entraînement (nombre d'essais effectués). En d'autres mots, les estimés du modèle KPC peuvent être utilisés indépendamment du sujet, de la tâche et du niveau de pratique.

Avec le KPC, il n'est pas nécessaire de modéliser les opérations implicites (homing, pause mentale, recherche visuelle, etc.) en développant des règles d'insertion (ex. Card et al., 1980;1983, Hudson et al., 1999, John & Kieras, 1996a;1996b) et le résultat reste acceptable. En fait, il semble difficile de modéliser précisément la durée d'une pause

mentale. Pour en faire foi, cette durée a été déterminée à 1350 et 1200 ms selon deux études différentes (Card et al., 1980, Olson & Olson, 1990).

Précision du KPC

Le biais de prédiction (figure 10) obtenu est optimiste pour les sujets habiles et naïfs. Ceci peut être dû au fait que les sujets habiles et naïfs (essais 1-10) ont été plus actifs cognitivement (ex. vérification, hésitation, pause avant confirmation) parce qu'ils étaient dans la phase d'apprentissage de la tâche. En fait, selon Kieras (1993), les nouveaux usagers ont tendance à vérifier les différentes étapes qu'ils effectuent. De plus, ils n'ont pas encore tendance à entrelacer leurs actions cognitives et physiques (c'est-à-dire que les automatismes ne sont pas encore développés). Cependant, comme les sujets étaient déjà habiles, l'effet de l'état naïf sur la précision de prédiction n'a pas été significatif; la précision obtenue étant d'environ 26 %.

Pour les sujets habiles et entraînés, le biais de prédiction obtenu est légèrement pessimiste. En fait, il est d'environ une seconde pour le KPC et 250 ms pour le context-dependent KPC. Cette légère différence est entièrement due à la variabilité inter-individuelle (entre les sujets qui ont servi à établir les temps du KPC et ceux dont on a prédit les performances), puisque les deux groupes avaient le même niveau d'entraînement.

Les effets de la tâche, du sujet et de l'entraînement ainsi que les effets croisés sur la précision de prédiction ont tous été significatifs mais en majorité acceptables. Cependant, l'effet de l'entraînement (essais 1-10 vs essais 31-40) fût plus important que les autres démontrant nettement une différence de pratique entre les dix premiers et dix derniers essais.

Effet du contexte

Les opérations implicites d'un usager expert sont entrelacées avec ses opérations explicites (Kieras et al., 1993, Kieras et al., 1995). Les résultats obtenus (distributions des temps d'exécution) pour le modèle *context-dependen*t KPC semblent confirmer ceci également pour les usagers habiles et entraînés. Dans tous les contextes, la distribution est asymétrique à droite (voir figure 8). Ceci est probablement dû à la présence d'une opération implicite, soit un *« homing »* ou une activité cognitive.

Card (1980;1983) et Olson (1990) ont estimé pour une pause mentale des temps moyens de 1350 et 1200 ms, respectivement. Pour ce qui est du *« homing »*, le temps moyen est habituellement de 400 ms (KLM). D'après les résultats de la figure 8, on constate qu'il n'est pas facile de localiser les opérations implicites et surtout de leur assigner un estimé du temps d'exécution moyen. C'est pour cette raison que ces actions ne sont pas tenues en compte dans le modèle KPC. Toutefois, leur existence n'est pas niée. Les résultats ont démontré que la précision de prédiction du modèle *context-dependent* KPC pour des sujets habiles (naïfs et entraînés) est très similaire à celle du KPC. En fait, elle a été légèrement plus précise (quasiment non-biaisée) que le KPC pour les sujets habiles et entraînés (figure 10). Ainsi, il n'est pas nécessaire de considérer ces distributions pour la prédiction. Cependant, les contextes fournissent de l'information pertinente sur la présence des opérations implicites.

Précision du KLM

Selon les résultats obtenus, le KLM est plus précis pour les sujets habiles et naïfs que pour les sujets habiles et entraînés avec des précisions de 33 % (42 % avec la séquence réelle) et 69 % (73 % avec la séquence réelle), respectivement. Ces valeurs sont nettement supérieures à ce qui est normalement accepté en modélisation analytique (environ 20 %). Cependant, cet écart est peut être dû au mode de calcul. Dans le cas où

l'erreur est calculée à partir de la moyenne du groupe (ou la moyenne des temps d'exécution d'un sujet sur plusieurs exécutions), elle correspond plus au biais qu'a la définition utilisée ici.

Pour générer les prédictions avec le KLM, le temps moyen utilisé pour une pause mentale fût de 1200 ms (Hudson et al., 1999), ce qui est inférieur à l'estimé du KLM d'origine. L'optimisme dans le cas des sujets habiles et naïfs et le pessimisme pour les sujets habiles et entraînés sont justifiables de la même façon que pour le KPC et le *context-dependent* KPC.

La grande différence de précision entre le KLM et les autres modèles (KPC et *context-dependent* KPC) pour les sujets habiles et naïfs est justifiable par le fait que le KLM a été conçu pour prédire les performances d'usagers experts (qui sont entraînés avec la tâche). Cependant, la différence fût encore plus grande pour les sujets habiles et entraînés. Ces différences peuvent être dues en partie à l'estimé du temps moyen d'exécution du P qui est de 1100ms pour le KLM comparativement à 700 ms pour le KPC. Ceci peut être du aux changements physiques de la souris (Santos & Badre, 1994). Cependant, selon les résultats obtenus, les pointages ne représentent en moyenne que 17 % des actions des usagers. Cet écart n'explique donc pas entièrement ces différences de précision. Il semble surtout que le KLM ne modélise pas précisément les activités cognitives de l'usager habile et entraîné. Il est donc plus prudent d'utiliser le KPC qui reconnaît la présence des actions implicites sans toutefois les considérer dans sa prédiction.

Séquences utilisées et optimalité

Selon les résultats, l'usager habile n'utilise pas vraiment qu'une séquence optimale dans l'exécution d'une tâche (figure 7). Ceci peut justifier l'utilisation de la méthodologie de mise au point (section 5.2) qui consiste à enregistrer les séquences tel qu'exécutées par

l'usager via un prototype de l'interface. Si une séquence optimale peut être déterminée analytiquement, alors ce processus empirique ne sera pas nécessaire. Cependant, ceci semble peu probable puisque dans certains cas le développeur devra anticiper la stratégie de l'usager. Par exemple, certains usagers préfèrent maximiser l'usage du clavier au détriment de la souris, ce qui est difficile à prévoir. Ceci affectera la séquence d'opérations et possiblement aussi le temps d'exécution.

CHAPITRE 6

DISCUSSION

Le système AIR était initialement destiné aux étudiants dans le cadre de leurs activités de laboratoire de leur(s) cours d'interfaces. Cependant, le système a été conçu dès le départ comme un outil d'aide au développement et à l'évaluation d'interfaces usager-machine. Les premières sections de ce chapitre présentent les forces et faiblesses du langage AIR et du modèle KPC ainsi que les limitations des outils logiciels. Par la suite, on discute de la diffusion potentielle du système AIR en rappelant les facteurs qui ont limité les outils plus anciens à une diffusion marginale. Le chapitre se termine par une discussion sur le nouveau type d'usager et la notion d'optimalité dans les séquences.

6.1 Langage AIR

Forces

La principale force du langage AIR est qu'il permet de spécifier les interactions à n'importe quel niveau d'agrégation (niveau de détail) ou d'abstraction (niveau plus général). De plus, l'AIR permet une spécification concise grâce à l'utilisation de librairies de sous-scripts. Ceci permet d'éviter de réécrire certains scripts (tâches exécutées fréquemment) et allège ainsi la représentation.

L'enquête de l'annexe 9 a révélé que l'AIR est plus facile d'apprentissage, plus facile à écrire et plus facile à lire que l'UAN. De façon générale, 75 % des répondants ont préféré l'AIR à l'UAN. Ces résultats sont fort probablement dus au fait que la syntaxe du langage a été conçue afin de ressembler le plus possible à celle des langages de programmation structurée (fréquemment utilisés par les étudiants en génie). Cependant,

il faut s'abstenir de généraliser ces résultats, qui ont été obtenus dans un contexte académique, à l'ensemble des concepteurs d'interfaces.

Faiblesses

Bien que ses concepts de base soient faciles à apprendre, la syntaxe AIR est assez volumineuse. Ainsi, un certain temps d'apprentissage doit être pris afin de connaître la syntaxe de façon exhaustive. Ceci représente la principale faiblesse de l'AIR. En fait, l'enquête (annexe 9) a révélé qu'en moyenne les répondants ont pris plus de temps à écrire un script en langage AIR qu'avec l'UAN.

6.2 Modèle KPC

Forces

Le modèle KPC permet de représenter rapidement une tâche en une séquence d'opérations explicites (seulement 3 opérations à mémoriser). Il permet également de prédire le temps d'exécution d'un usager habile et entraîné peu importe le type de tâche, et ce avec une marge d'erreur respectable en modélisation analytique.

La validation expérimentale (chapitre 5) a révélé la présence d'opérations implicites (homing, pause mentale, recherche visuelle, etc.) dans certains contextes d'exécution. Cependant, il est difficile de déterminer leur type exact et de leur associer un temps d'exécution. Le KPC ne modélise donc pas les actions implicites de l'usager, leur temps étant compris dans l'estimé du temps d'exécution des actions explicites. En fait, les résultats obtenus ont révélé que les précisions de prédiction avec et sans considération du contexte sont similaires (à quelques pourcents près).

L'étude du chapitre 5 a révélé que le KLM est plus précis pour les usagers habiles et naïfs que pour les usagers habiles et entraînés, et ce même s'il est prévu pour un usager expert. Il semble donc que le KLM ne modélise pas correctement les activités cognitives d'un usager habile et entraîné. De plus, il est plus complexe à construire que le KPC, sans que la prédiction soit meilleure.

L'étude a également révélé qu'un usager habile utilise plus d'une séquence optimale (en terme du temps d'exécution et du nombre d'opérations) dans l'exécution d'une tâche donnée et ce même une fois entraîné. Ainsi, le KPC est utilisé sur la séquence enregistrée directement de l'usager contrairement aux cas pour lesquels la séquence est définie par l'analyste soit directement ou par l'entremise d'un prototype de l'interface. Cet enregistrement peut s'avérer plus coûteux en ressources mais il évite que certaines séquences échappent à l'analyste.

Faiblesses

Une première faiblesse du KPC se situe au niveau de la représentation de la tâche. En fait, il ne permet de spécifier que les actions de l'usager en négligeant tout détail relatif aux interactions (état interne du système et rétroaction de l'interface).

Comme modèle prédictif du temps d'exécution, le KPC est limité aux interfaces GUI et leur type d'usagers actuels. Il devra donc être mis au point à nouveau dans le cas d'un changement du profil de l'usager type ou pour supporter d'autres types d'interfaces (nouveaux composants d'interfaces ou dispositifs d'interaction).

Une dernière lacune est que le KPC ne permet pas de modéliser les actions implicites de l'usager. Bien qu'il permet de prédire le temps d'exécution précisément sans tenir compte des ces actions, la modélisation ne fournit aucun détail sur les activités cognitives exécutées par l'usager dans l'exécution de sa tâche.

6.3 Outils logiciels

Les outils logiciels qui ont été développés dans ce projet représentent un prototype de niveau universitaire. C'est-à-dire qu'ils n'ont pas été testés ni validés selon les standards de l'industrie. À l'exception de jeux de test utilisés (annexe 7), aucun test exhaustif n'a été effectué sur les modules logiciels. De plus, aucune étude d'usabilité ou d'ergonomie n'a été effectuée sur l'interface-usager. En fait, l'outil intégré dans son état actuel est rudimentaire, avec seulement trois modes d'opération : 1) Vérification syntaxique et estimation de temps intégrées. 2) Vérification syntaxique seulement. 3) Estimation de temps seulement. Plus en détails, l'outil de vérification syntaxique possède les limitations suivantes:

- La vérification est intracellulaire : elle est effectuée pour chaque cellule du script individuellement. En fait, la cohérence entre les différentes cellules n'est pas vérifiée.

- La sémantique des expressions n'est pas testée : il est possible d'écrire des scripts irréalistes, comme par exemple, deux presses (*mouse.press*) consécutives du même bouton de la souris ou deux pauses mentales (*user.think*) consécutives.

La principale limitation du module d'estimation du temps d'exécution réside dans l'estimation du temps d'une boucle. En fait, la flexibilité de la syntaxe rend difficile la détermination automatique du nombre d'itérations spécifié. Pour surmonter cette limitation, le temps total estimé de la boucle est obtenu sous la forme d'une expression du type *X ms * loopCounter* où X représente le temps estimé pour une itération seulement.

En termes généraux, l'outil intégré manque de fonctionnalités et son utilité est limitée par le fait qu'il n'est pas intégré à un environnement de développement logiciel, comme par exemple *Eclipse®* ou *MS .Net®*.

6.4 Diffusion du système AIR

La plupart des outils de spécification et de modélisation plus anciens n'ont été diffusés que de façon marginale. Cette faible diffusion résulterait, en partie, du faible degré d'automatisation. En fait, Ivory & Hearst (2001) relatent que la plupart des méthodes automatiques sont utilisées seulement en modélisation analytique et en simulation. Dans une étude comparative de trois outils d'aide à la construction de modèles GOMS (CATHCI, GLEAN3 et QGOMS), Baumeister, John & Byrne (2000) ont conclu que chacun de ces outils a des faiblesses qui ne lui permettraient pas d'être utilisé dans l'industrie. Ils expliquent que si un outil doit être diffusé largement, ses fonctionnalités doivent répondre aux standards des outils de développement (ex. gestion de *« crash »*, sauvegarde, réutilisation de modèles, copie de documents, etc.). Selon Myers et al. (2000), le manque de diffusion résulte du fait que la plupart de ces outils ne permettent pas d'être utilisés avec un minimum de ressources (*« low threshold »*) tout en étant complètement automatisés (*« high ceiling »*).

Les outils logiciels du système AIR nécessitent un investissement initial léger (spécification concise au niveau de détails, facilités d'apprentissage et d'utilisation et adaptabilité aux changements technologiques majeurs). Cependant, pour qu'ils puissent être diffusés, ils devraient être empaquetés (testés, validés et documentés) et leurs fonctionnalités augmentées. Leur diffusion pourrait se faire soit sous forme d'un logiciel indépendant ou d'un module ajouté (*« plug-in »*) à un environnement de développement.

Enfin, il faut souligner que le modèle KPC est un outil en soi, qui peut être utilisé sans environnement informatique. Il est un outil prédictif léger et simple qui permet de

construire des modèles KPC facilement à la main. Dans ce cas, la diffusion se ferait via le cycle normal de diffusion scientifique qui comprend les composantes suivantes : rédaction d'articles, conférences et/ou enseignement et supports Web (ex. tutoriaux, listes de discussion).

6.5 Points additionnels

Nouveau type d'usager

Dans ce projet, on a introduit la notion d'usager habile (*« skilled user »*). Ce dernier représente une personne utilisant régulièrement un ordinateur de type PC (*« Personal Computer »*) et ayant acquis un certain niveau d'habiletés avec les standards d'interfaces GUI. Ces standards évoluant lentement, les usagers sont devenus habiles sous l'effet d'un usage répété et cumulatif. Il est bien entendu que lorsqu'un usager habile est soumis à une tâche informatique nouvelle, il doit apprendre à exécuter cette dernière. Ainsi, il peut être considéré comme étant naïf ou entraîné (tout en restant habile) avec la tâche dépendamment de son niveau de pratique.

Les modèles prédictifs plus anciens (ex. KLM, GOMS) font plutôt référence à un usager expert (*« expert user »*). Il a été démontré que même après plus d'un an d'usage d'une application, un usager habile ne connaît toujours pas certaines fonctionnalités de base et est loin d'être considéré comme un expert au niveau quantitatif et qualitatif (Nilsen et al., 1993). Il a également été démontré qu'un usage efficient d'une application n'est pas toujours possible et ce même pour un usager expérimenté avec cette dernière (Bhavnani & John, 1997).

Alors que la notion d'usager expert semble parfois peu réaliste, la notion d'usager habile semble représenter l'usager type actuel d'interfaces GUI. Les usagers habiles ne sont pas toujours experts, ce qui pourrait restreindre l'utilisation pratique des modèles validés

pour les usagers experts. Le modèle KPC devra être révisé en cas de changement majeur dans les technologies ou les interfaces qui obligerait les usagers à réapprendre un nouveau style d'interaction (et pourrait de plus nécessiter des opérations élémentaires différentes).

Séquences utilisées et optimalité

L'étude du chapitre 5 a démontré que les sujets habiles utilisent plusieurs séquences optimales et ce même dans l'exécution de tâches simples. En fait, que ce soit lors de l'apprentissage ou une fois entraînés, les sujets ont utilisé plusieurs séquences optimales (en terme du temps d'exécution et du nombre d'opérations). On a même constaté que les sujets habiles et entraînés utilisent certaines séquences non optimales. Les modèles plus anciens basés sur une seule séquence optimale (ex. KLM, GOMS) ne semblent donc pas pouvoir supporter la variété de séquences utilisées par les usagers habiles.

Un phénomène similaire a été constaté dans une étude sur les stratégies utilisées dans le dessin assisté par ordinateur (Bhavnani & John, 1996). Selon les auteurs, les usagers expérimentés utilisent encore des stratégies non optimales même après quelques années d'usage. Des résultats similaires ont également été trouvés dans le cadre d'une étude sur l'effet de l'usage répété d'une application (*Lotus 123*) sur les performances de l'usager (Nilsen et al., 1993). Si la stratégie optimale n'est jamais démontrée à l'usager et qu'aucune rétroaction sur la non optimalité n'est effectuée, alors il sera peu probable que l'unique stratégie optimale, si existante, soit exécutée (Bhavnani & John, 1996). Bhavnani & John (1997) ont même étudié certaines méthodes tentant de faire migrer l'usager d'un usage suffisant à un usage efficient. Cependant, le mérite de notre étude est qu'elle a révélé que les usagers habiles et entraînés peuvent utiliser des stratégies non optimales et ce même pour des tâches très simples.

Il est bien entendu que si une séquence optimale peut être déterminée analytiquement, alors le processus empirique pour mesurer la séquence réelle peut être évité. Cependant, ceci semble peu probable puisque dans certains cas le développeur devra anticiper la stratégie de l'usager. En fait, certains usagers préfèrent maximiser l'usage du clavier au détriment de la souris, ce qui est difficile à prévoir. Ceci affectera la séquence d'opérations et possiblement aussi le temps d'exécution.

CONCLUSION ET RECOMMANDATIONS

Conclusion

La présente étude a introduit le système *Activity-oriented Interface Representation (AIR)*. L'utilisation du système AIR se divise en deux volets : 1) spécification des interactions (langage AIR) et 2) modélisation analytique (modèle KPC).

Afin d'atteindre les objectifs de la spécification, le langage AIR a été introduit. Dans un premier temps, il sert à spécifier une tâche en terme d'actions élémentaires exécutées par l'usager. Il permet également de spécifier le détail des interactions, soit les changements d'état interne du système et la rétroaction de l'interface vers l'usager. La durée de chaque étape de la tâche fait également partie de la spécification. La syntaxe du langage ressemble étroitement à celle des langages de programmation structurée (C, C++, Java). Elle a été conçue ainsi pour tenter de réduire les efforts d'apprentissage et de faciliter l'utilisation du langage.

Les outils logiciels du système AIR ont ensuite été présentés. Ces outils permettent principalement deux choses : 1) Vérifier automatiquement la syntaxe d'un script AIR. 2) De façon automatique, extraire la séquence KPC d'un script AIR et estimer le temps d'exécution.

Le modèle KPC a été introduit afin de modéliser les usagers habiles dans leur exécution de tâches sur interfaces GUI. Dans un premier temps, il permet de transformer la séquence d'actions (réellement exécutées) en une séquence d'opérations K, P et C. Ces opérations représentent une presse d'une clé du clavier, un mouvement de souris et un clic d'un bouton de souris, respectivement. Deuxièmement, il permet de prédire le temps d'exécution de la séquence en utilisant des estimés associés à chaque opération.

Une méthodologie de mise au point du KPC a été développée avec 10 sujets. La mise au point a révélé que les estimés du modèle sont valides peu importe la tâche, le sujet et son niveau d'entraînement. Il est possible que l'on veuille utiliser un tel modèle pour évaluer une interface de type autre que GUI ou étant utilisée par un usager autre que l'usager habile actuel. Dans ce cas, la méthodologie pourra être réutilisée afin de produire de nouvelles valeurs prédictives reflétant ces changements.

Le modèle KPC a également été validé expérimentalement sur 10 sujets (différents de ceux utilisés pour la mise au point). L'étude a révélé une erreur de précision près du standard accepté en modélisation analytique peu importe la tâche, le sujet et son niveau d'entraînement. La validation du KPC a soulevé quelques questionnements sur les notions d'usager expert et de séquence optimale. Il a été montré qu'aucune séquence optimale unique n'est utilisée par un usager habile (naïf ou entraîné) et ce même pour des tâches simples. Pour ce qui est des usagers d'interfaces actuels, ces derniers sont déjà habiles même s'ils sont naïfs avec l'application. Ceci est dû au fait qu'ils ont acquis certaines habiletés dues à un usage répété d'interfaces similaires. Dans cet ordre d'idées, le modèle KPC est plus robuste que certains modèles plus anciens basés sur un usager expert et une tâche optimale (ex. KLM, GOMS).

Recommandations

Les outils logiciels AIR devraient être intégrés dans un environnement de développement et de prototypage qui trouverait son utilité dans les différentes phases du cycle de développement d'interfaces. Afin d'y arriver, une fonctionnalité supplémentaire serait la création d'un script AIR (journal AIR vérifié syntaxiquement avec estimation du temps d'exécution) à partir de l'enregistrement des activités d'un usager dans l'exécution d'une tâche sur un prototype d'interface. Ceci permettrait de vérifier si la spécification d'origine reflète la réalité et si elle permet de rencontrer les exigences de

performances (en temps d'exécution) définies a priori. La figure 11 présente le cycle complet de développement proposé.

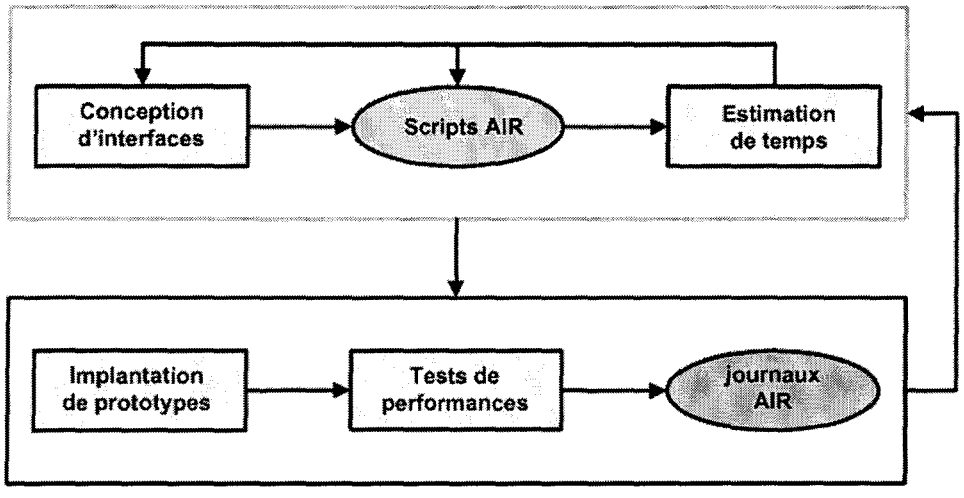


Figure 11 Outils AIR augmentés - cycle de développement proposé

Les outils AIR pourraient être diffusés dans la communauté universitaire gratuitement en utilisant le concept d'informatique libre (*« open source »*) via une licence de diffusion. Cette dernière devrait répondre à différents critères bien spécifiques touchant des sujets comme : la redistribution libre, le code source, l'intégrité du code source de l'auteur, les travaux dérivés, la discrimination entre les individus, etc.

Malgré ses limitations actuelles, le système AIR pourrait avoir plusieurs applications dans les domaines du design et de l'évaluation d'interfaces. Elles se divisent en deux volets, soit les volets académique et industriel.

Volet académique

Le système AIR pourrait être enseigné auprès de futurs concepteurs et/ou évaluateurs d'interfaces. La mission consisterait à :

1) Montrer l'importance de spécifier le détail des interactions : les actions de l'usager, les changements d'état et la rétroaction de l'interface en réponse à ces actions doivent être spécifiés.

2) Justifier l'utilisation d'un outil intégré de spécification et d'évaluation : il est important d'obtenir rapidement une idée sur : i) l'exactitude d'une spécification et ii) le niveau d'efficience de l'interface.

Volet industriel

Dans l'industrie, le système AIR pourrait être utilisé par un concepteur/évaluateur d'interfaces ou un expert/conseil en usabilité et/ou ergonomie des interfaces. Il pourrait même être utilisé en formation. Dans l'industrie, il est important d'avoir un prototype du produit rapidement. Un outil comme le système AIR permettrait d'effectuer un design et de l'évaluer plusieurs fois et ce de façon rapide et efficace avant même de réaliser un prototype de l'interface. Ainsi, on bouclerait moins souvent dans le cycle de conception-réalisation-évaluation. Autrement dit, l'utilisation d'un tel outil permettrait à une compagnie de sauver du temps et de l'argent et d'obtenir un produit plus rapidement.

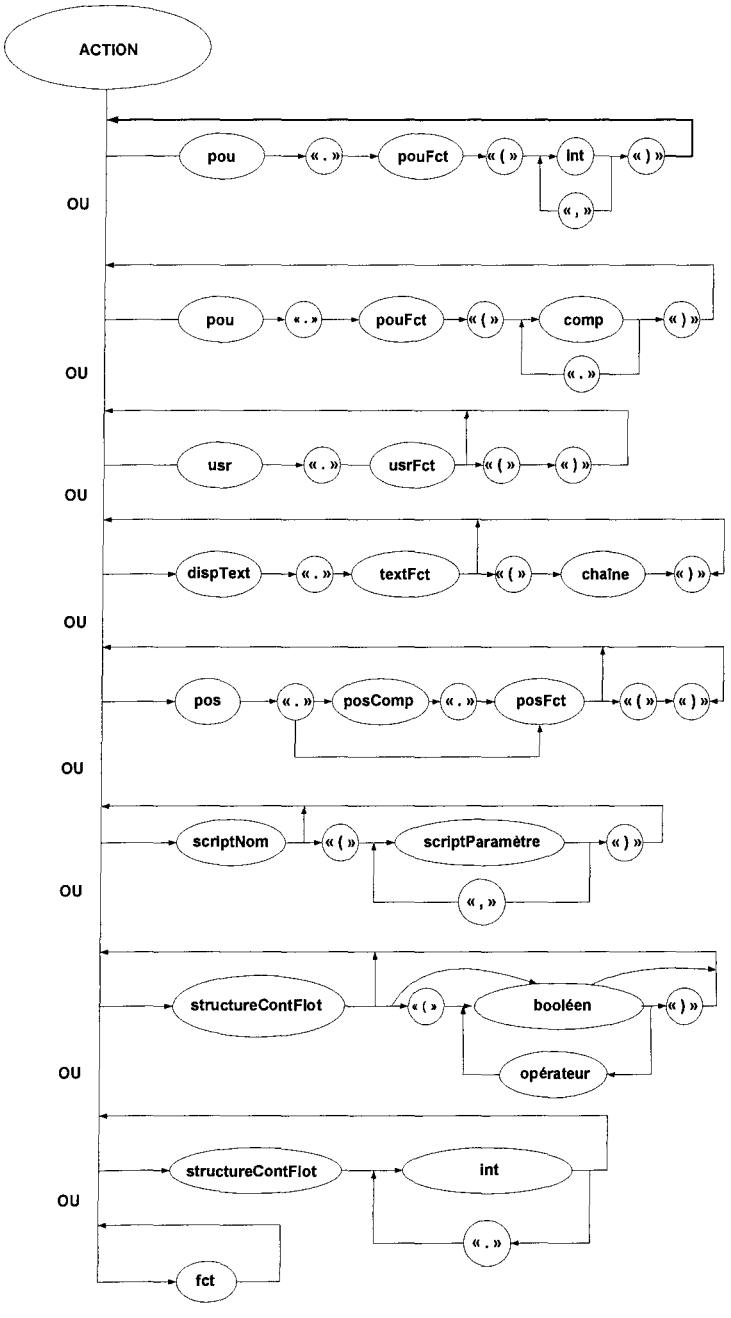
ANNEXE 1

Langage AIR 1.0 - diagrammes de grammaire de la syntaxe

Afin de faire le pont entre les spécifications du langage AIR (chapitre 2) et la définition de sa syntaxe, des diagrammes de grammaire ont été effectués. Les sections suivantes présentent ces diagrammes pour différents types d'expressions: action de l'usager, état interne du système et rétroaction de l'interface.

Colonne Action d'un script AIR

Le diagramme de la figure A1-1 présente les possibilités grammaticales pour une expression représentant une action de l'usager. Les sous-diagrammes représentent les différents cas possibles. Chaque cas peut représenter plusieurs expressions et se lit de gauche à droite et chaque flèche de retour signifie que l'expression peur se terminer. Un exemple pour le premier cas est le positionnement de la souris à une position absolue de l'écran : *mouse.goto(10, 200)*.



Légende

booléen : expression booléenne.
chaîne : chaîne de caractères.
comp : composant graphique
dispText : dispositif d'entrée de texte.
fct : fonction quelconque.
int : entier.
opérateur : opérateur binaire.
posComp : sous-composant matériel d'un pointeur ou d'un interrupteur.
posFct : action exécutable avec un pointeur ou un interrupteur.
pos : pointeur ou interrupteur.
pouFct : action exécutable avec un pointeur ou par l'usager lui-même.
pou : pointeur ou usager.
scriptNom : nom d'un sous-script.
scriptParameter : paramètre d'un sous-script.
structureContFlot : structure de contrôle.
textFct : fonction d'entrée de texte.
usrFct : action exécutable par l'usager.
usr : usager.

Figure A1-1 Langage AIR - diagramme de grammaire pour une expression de la colonne *Action*

Colonne State d'un script AIR

Le diagramme de la figure A1-2 présente les possibilités grammaticales pour une expression représentant un état ou changement d'état du système. L'interprétation du diagramme s'effectue de la même façon que pour les actions. Un exemple pour le premier cas est l'emmagasinage en mémoire du contenu d'une zone de texte : *stored = applicationWindow.textArea.content.*

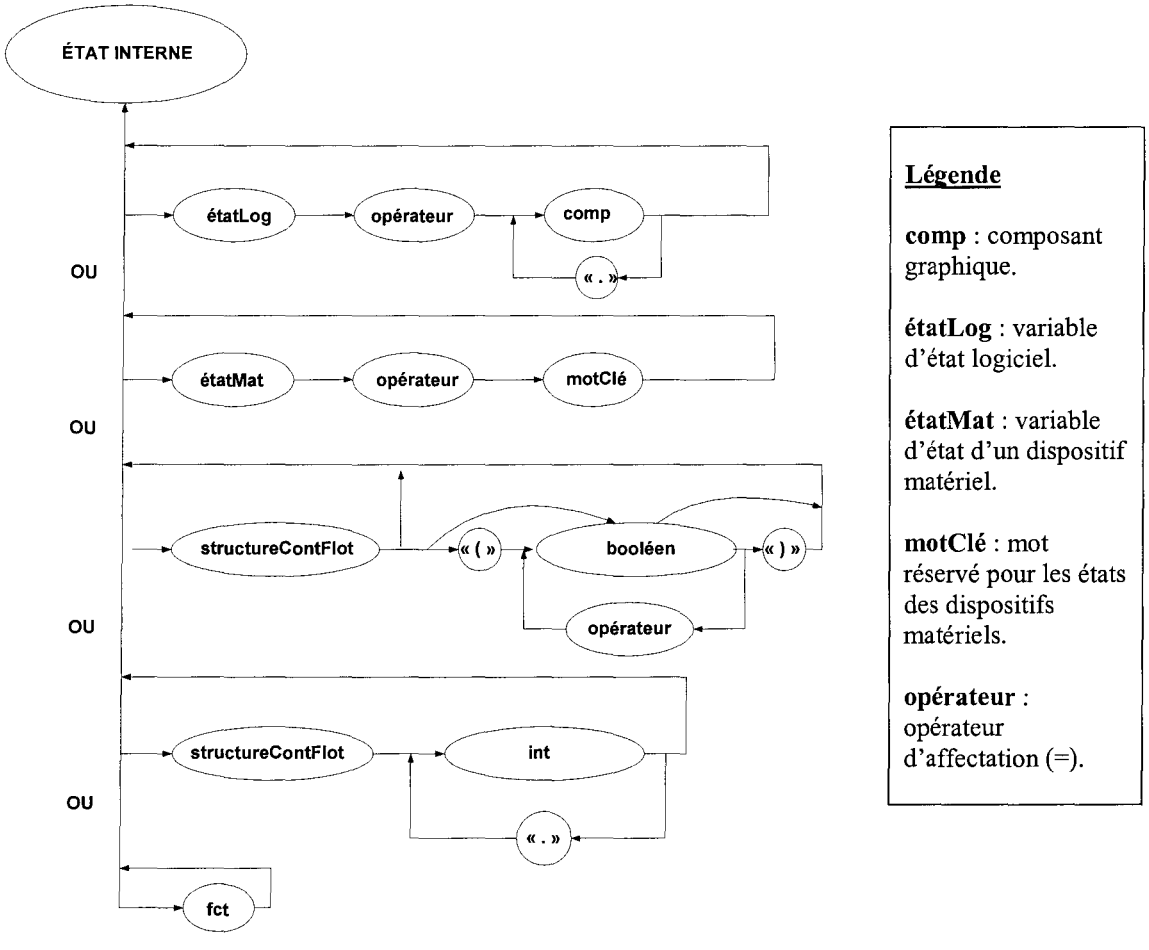


Figure A1-2 Langage AIR - diagramme de grammaire pour une expression de la colonne *State*

Colonne Feedback d'un script AIR

Le diagramme de la figure A1-3 présente les possibilités grammaticales pour une expression représentant une rétroaction de l'interface vers l'usager. L'interprétation du diagramme s'effectue de la même façon que pour les actions et les états. Un exemple pour la troisième possibilité grammaticale est l'affichage d'une boîte de dialogue : *window.dialogBox.display()*.

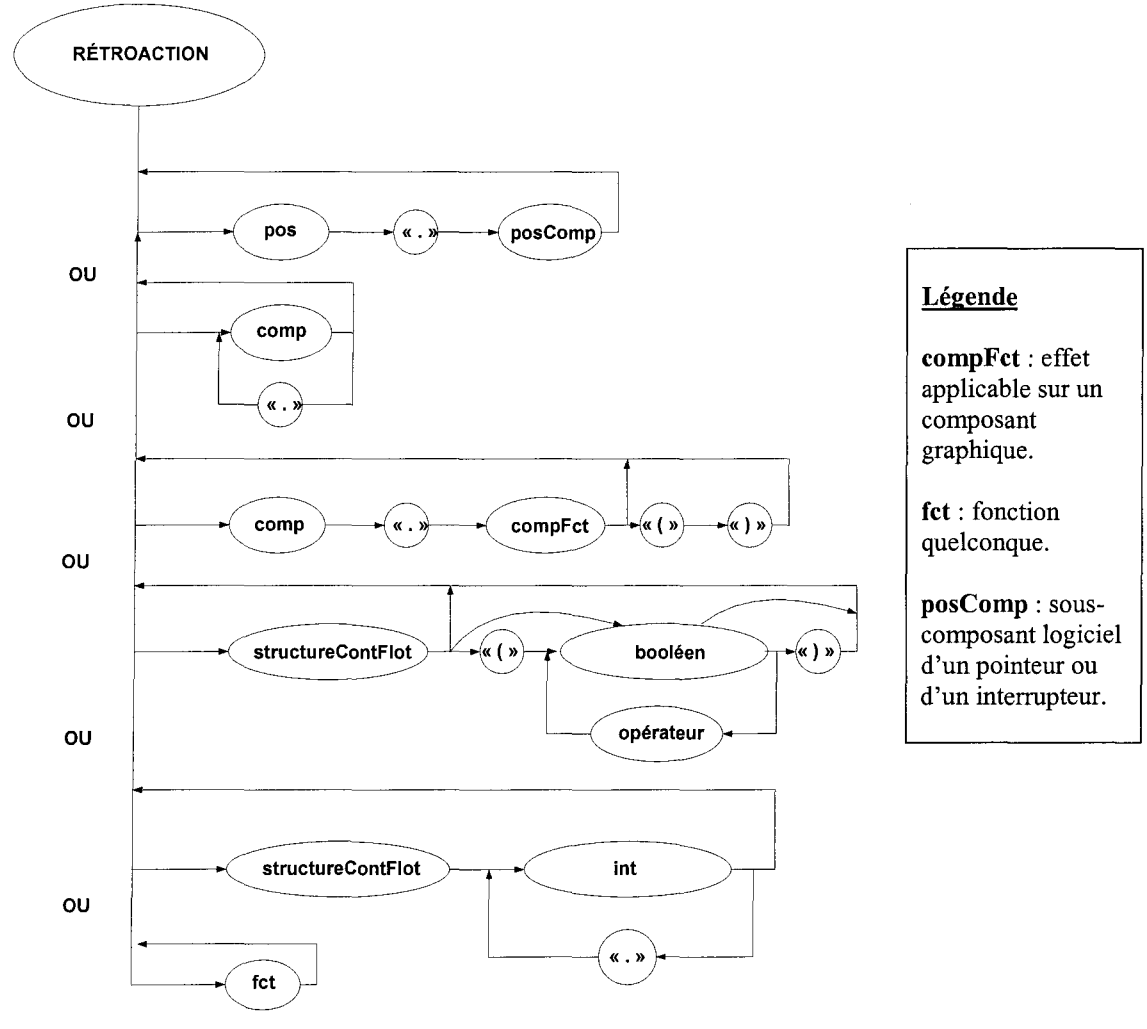


Figure A1-3 Langage AIR - diagramme de grammaire pour une expression de la colonne *Feedback*

ANNEXE 2

Langage AIR 1.0 – document de spécification du langage

1. Motivations

Cette annexe présente les éléments et règles de syntaxe du langage AIR (chapitre 2). Les objectifs du langage sont les suivants:

Objectifs primaires

1. Représenter les interfaces usager-machine sous la forme de scripts d'interactions.
2. Spécifier les interactions à différents niveaux d'abstraction et d'agrégation.
3. Permettre d'écrire des scripts de tâches comme partie des documents de spécification d'interfaces.
4. Permettre de décrire la réalisation de tâches lors d'observations (ex. tests d'usabilité).
5. Permettre d'extraire une spécification AIR à partir de journaux d'événements par le biais d'un processus semi-automatique.

Objectifs secondaires

1. Doit être concis.
2. Doit être facile à mémoriser et intuitif.
3. Doit rester proche de la syntaxe des langages de programmation structurée (ex. C++ et Java).

2. Présentation générale

En langage AIR, chaque façon de réaliser une tâche correspond à un script. Ce dernier se caractérise par un nom et des arguments, et se présente sous forme d'un tableau à 5 colonnes. Elles sont nommées de gauche à droite: numéro d'étape (colonne *Step*), action de l'usager (colonne *Action*), état ou changement d'état (colonne *State*), rétroaction de l'interface (colonne *Feedback*) et durée de l'action (colonne *Duration*). Les sous-

sections qui suivent présentent les différents éléments de la représentation d'une interface, soit les objets de l'interface (composants graphiques), les objets prédéfinis, les caractéristiques des composants, les variables d'état et les scripts d'interaction.

2.1 Objets de l'interface

Les objets de l'interface représentent les composants graphiques au sens usuel. Chacun est désigné par un *identificateur*. En cas d'ambiguïté, on le désigne en utilisant son parent (super-composant qui le contient). Un exemple de composant unique est le menu de sauvegarde qui est unique à toute une application (ex. *saveMenu*). Un exemple contraire est le suivant : *saveWindow.okButton*. En fait, il peut y avoir plusieurs boutons de confirmation différents sur différentes fenêtres d'une même application.

Pour les noms d'identificateurs, les règles à suivre sont les suivantes :

- Un identificateur est une suite de mots de longueur quelconque.
- Le premier mot est en minuscule tandis que tous les autres commencent par une majuscule, le reste étant en minuscule.
- Les espaces, soulignés ou caractères spéciaux sont interdits.

Remarque : si les normes suivies pour les identificateurs diffèrent ce qui précède, elles doivent être définies explicitement.

2.2 Objets prédéfinis

Les objets prédéfinis peuvent représenter soit l'interface, l'usager ou les dispositifs d'interaction utilisés. On les divise en cinq catégories :

- *Objets matériels ou logiciels* : i) Interface ou logiciel d'interface. ii) Composant logiciel actif (*focus*). iii) Curseur texte (*cursor*). iv) Écran (*screen*).

- *Usager* : i) Doigts (*finger*). ii) Main dominante (*dominantHand*) et non dominante (*nonDominantHand*). iii) Voix (*voice*). iv) Regard (*gaze*).

- *Dispositifs de pointage ou pointeurs* : i) Pointeur générique (*anyPointer*). ii) Souris (*mouse)*. iii) Manche à balai (*joystick*). La souris et le manche à balai ont les sous-composants suivants : bouton de gauche (*leftButton*), bouton du centre (*centerButton*), bouton de droite (*rightButton*) et bouton de déplacement (*scrollButton*). Ces sous-composants sont utilisés seulement si nécessaire.

- *Dispositifs d'entrée de texte* : i) Dispositif d'entrée de texte générique (*anyTextEntry*). ii) Clavier (*keyboard*). iii) Écran Tactile (*touchScreen*). iv) Voix (*voice*). v) Tablette de saisie ou de reconnaissance d'écriture (*writingPad*).

- *Dispositifs d'interruption ou interrupteurs* : Un seul type d'interrupteur est défini à l'état actuel. Il s'agit de l'interrupteur générique (*anySwitch*). Cependant, il faut noter que la souris est également considérée comme un interrupteur.

Remarque : le concepteur a la possibilité de rajouter d'autres objets.

2.3 Caractéristiques des composants

Les caractéristiques suivantes peuvent être spécifiées pour les composants graphiques (*anyComponent* représente un composant graphique générique):

- *anyComponent.position* : position absolue ou relative d'un composant graphique. En cas d'ambiguïté, il est possible de rajouter un préfixe à *position*. Ce préfixe

peut être soit *absolute* ou *relative*. On utilise *anyComponent* pour représenter un composant quelconque.

- *anyComponent.x, anyComponent.*y : position absolue.

- *(childComponent,parentComponent).x* : abscisse d'une position relative d'un composant.

- *(childComponent,parentComponent).y* : ordonnée d'une position relative d'un composant.

- *anyComponent.upperLeft* : coin supérieur gauche. Même chose pour les autres coins.

- *anyComponent.left* : segment de gauche. Même chose pour les autres segments.

- *anyComponent.sizeX, anyComponent.sizeY* : dimension en abscisse et en ordonnée d'un composant.

- *anyComponent.size* : dimension (x, y) d'un composant.

- *anyComponent.parent* : le parent d'un composant (ex. une fenêtre qui contient un bouton).

- *anyComponent.value* : valeur associée à un composant.

- *anyComponent.string(row,column,length)* : chaîne de caractères commençant à une position absolue et d'une longueur donnée.

- *anyComponent.char(row,column)* : caractère étant à une position absolue.

- *anyComponent.handle* : la poignée ou zone d'écran où il est possible de cliquer sur un composant. Cette zone est parfois plus grande que le composant. Quand elle coïncide avec le composant, il est inutile d'utiliser le mot *« handle »*.

- *anyComponent.outline* : le contour d'un composant. Cette caractéristique est parfois visible à l'écran lorsqu'un composant est déplacé avec la souris.

- *anyComponent.background* : arrière-plan d'un compoosant.

- *anyComponent.cursor* : curseur de texte spécifique au composant.

- *anyComponent.pointer* : pointeur spécifique au composant.

- *anyComponent.title* : barre de titre d'une fenêtre.

- Boutons d'ajustement de la taille d'une fenêtre (ex. *anyWindow.maximizeButton, anyWindow.sizeUpButton*).

- Groupe de composants (ex. *activatedIconGroup, window.allComponents*).

Remarque: si on rajoute des caractéristiques aux composants, il faut les définir explicitement.

2.4 Variables d'état de l'interface

Les variables d'état de l'interface servent à définir les états ou changements d'états de l'interface, qu'ils soient logiciels ou matériels. Ces variables peuvent être représentées sous plusieurs formes (ex. mot-clé, valeur numérique, valeur booléenne, chaîne de

caractères, composant graphique). Ces variables sont considérées comme publiques, partagées par tous les scripts.

2.5 Structure et exécution d'un script AIR

Un script AIR a un identificateur et des arguments, tout comme une fonction en programmation (ex. *identify(login,password)*). Un script est représenté par un tableau à cinq colonnes (*Step*, *Action*, *State*, *Feedback* et *Duration*). L'exécution d'un script s'effectue étape par étape. Les étapes sont exécutées dans l'ordre (de haut en bas). Dans une étape, l'ordre est le suivant de gauche à droite : i) action de l'usager, ii) état ou changement d'état interne et iii) rétroaction de l'interface vers l'usager. Ceci est illustré au tableau A2-I où l'ordre est le suivant : A1→A2→V1=X→V2=V1=X→Affiche 1→Affiche 2.

Tableau A2-I

Exécution d'un script AIR

Step	Action	State	Feedback	Duration
1	A1, A2	V1 = X V2 = V1	Affiche1, Affiche 2	
2				

On peut changer l'ordre d'exécution en utilisant des structures de contrôle du flot d'exécution et des appels de sous-scripts (voir plus loin). Les instructions de contrôle et les appels de sous-scripts sont mis sur des lignes séparées pour les différencier des séquences d'actions et de rétroactions.

2.6 Structures de contrôle du flot d'exécution

La syntaxe AIR contient trois types de structures de contrôle de base, soit : i) les structures conditionnelles (*if-else*), les structures itératives (*repeat, for* et *while*) et les sauts directs (*step*). Il y a également, pour chaque type, des structures dérivées (*switch, do...while, continue, return* et *break*).

Structures conditionnelles

La structure de contrôle conditionnelle principale (*if-else*) s'écrit de la façon suivante :

```
IF (condition)
        ...
ELSE
        ...
ENDIF
```

Le tableau A2-II présente un exemple d'utilisation de cette structure dans lequel la colonne *Duration* est omise. Dans cet exemple, lorsque l'usager positionne la souris sur le menu de fichiers, ce dernier devient actif.

Tableau A2-II

Exemple d'utilisation de la structure conditionnelle *if-else*

Step	Action	State	Feedback
1	if mouse ON fileMenu		
1.1			fileMenu.activate
1.2	endif		

La structure dérivée *switch* s'écrit de la manière suivante :

SWITCH (expression)
    CASE X : cas possible en réponse à l'expression
        ...
ENDSWITCH

Le tableau A2-III présente un exemple d'utilisation de cette structure dans lequel les colonnes *Feedback* et *Duration* sont omises. Dans cet exemple, la chaîne de caractères qui est entrée avec le clavier est emmagasinée en mémoire.

Tableau A2-III
Exemple d'utilisation de la structure conditionnelle *switch*

Step	Action	State
1	switch (keyboard.entered)	
1.1	case «Pierre»	stored=«Pierre»
1.2	case «Louis»	stored=«Louis»
1.3	case «Steve»	stored=«Steve»
1.4	endswitch	

Structures itératives (boucles)

Une première structure itérative est la structure *repeat*. Elle s'écrit de la façon suivante :

REPEAT X TIMES
        ...
ENDREPEAT

X peut prendre les valeurs suivantes :

- 0, 1, etc.: entier quelconque.
- + : une fois ou plus.
- * : un nombre quelconque de fois (positif ou nul).
- X1 ... X2 : entre X1 et X2 fois.

Un exemple est présenté au tableau A2-IV dans lequel les colonnes *State* et *Duration* sont omises. Dans cet exemple, on spécifie le déplacement aléatoire de la souris à l'écran.

Tableau A2-IV

Exemple d'utilisation de la structure itérative *repeat*

Step	Action	Feedback
1	repeat * times	
1.1	mouse.goto(x,y)	mouse.arrow
1.2	endrepeat	

Un deuxième type de structure itérative est la boucle *for*. Elle s'écrit de la façon suivante :

```
FOR (expression)
        ...
ENDFOR
```

Un exemple est présenté au tableau A2-V où les colonnes *Feedback* et *Duration* sont omises. Dans cet exemple, une chaîne de caractères est entrée avec le clavier un nombre défini de fois. Chaque saisie est emmagasinée en mémoire.

Tableau A2-V

Exemple d'utilisation d'une boucle *for*

Step	Action	State
1	for (i=0;i<nb_element;i++)	
1.1	keyboard.enter()	stored = keyboard.entered
1.2	endfor	

Les structures itératives comprennent également les boucles *while* et *do...while*. La forme d'une boucle *while* est la suivante :

WHILE (condition)

ENDWHILE

Un exemple d'une boucle *while* est présenté au tableau A2-VI où les colonnes *Feedback* et *Duration* sont omises. Dans cet exemple, le sous-script *processInfo* est réexécuté tant que la fenêtre de l'application a le focus de l'interface.

Tableau A2-VI

Exemple d'utilisation d'une boucle *while*

Step	Action	State
1	while	focus = applicationWindow
1.1	processInfo()	
1.2	endwhile	

La forme de la structure *do...while* est la suivante :

DO

...

WHILE (condition)

Comme en programmation, une itération est garantie, ce qui n'est pas le cas de la boucle *while*. Un exemple de la structure *do...while* est présenté au tableau A2-VII dans lequel les colonnes *Feedback* et *Duration* sont omises. L'exemple est similaire à celui du tableau A2-VI.

Tableau A2-VII

Exemple d'utilisation d'une boucle *do...while*

Step	Action	State
1	do	
1.1	keyboard.enter()	
1.2	while	focus=textArea

Sauts directs

Les quatre types de sauts directs sont les instructions suivantes : *step*, *continue*, *return* et *break*. Le saut de type *step* permet de sauter directement à l'étape de la colonne *Step* spécifiée. L'instruction *continue* permet de terminer l'itération courante de la boucle et aller immédiatement à l'itération suivante. L'instruction *return* permet de terminer l'exécution d'un script ou d'un sous-script en donnant le contrôle au script appelant. Elle doit être accompagnée d'une valeur de retour, si nécessaire. Finalement, *break* permet de sortir d'une boucle ou d'une structure conditionnelle et d'aller à la première étape qui la suit. Le tableau A2-VIII présente un exemple d'utilisation de l'instruction *step* où les

colonnes *State*, *Feedback* et *Duration* sont omises. Dans cet exemple, si la souris est positionnée sur le composant *« label »*, alors l'usager doit cliquer avec le bouton de gauche.

Tableau A2-VIII

Exemple d'utilisation de l'instruction *step*

Step	Action
1	If mouse ON label
1.1	step 2
1.2	else
1.3	break
1.4	endif
2	mouse.click

Remarque : Si on veut utiliser d'autres structures de contrôle, il faut les définir explicitement.

2.7 Appels de sous-scripts

La façon d'appeler un sous-script est d'écrire son nom avec ses arguments (ex. *searchGoogle(searchText)*). Il est possible d'appeler plusieurs sous-scripts. Les différentes façons de procéder sont les suivantes :

- M OF sous-script1, sous-script2, ..., sous-scriptN MOT_CLÉ
- $M_1..M_2$ OF sous-script1, sous-script2, ..., sous-scriptN MOT_CLÉ
- ALL OF sous-script1, sous-script2, ..., sous-scriptN MOT_CLÉ

Dans le premier cas, l'usager exécute M sous-scripts de la liste, en ordre quelconque. Dans le second cas, il en exécute entre M_1 et M_2, en ordre quelconque. Dans le troisième cas, il les exécute tous (M=N), en ordre quelconque. L'ordre des appels peut être géré en spécifiant un mode d'appel à l'aide d'un mot-clé ajouté à la suite de l'appel. Les modes possibles sont les suivants :

- Entrelacement (mot-clé : *interleaved*) : l'exécution est entrelacée, c'est-à-dire qu'une section de chaque script s'exécute tour à tour.

- Interruption (mot-clé : *interrupted*) : chaque script peut être interrompu par les autres et ce n'importe quand.

- Simultanéité (mot-clé : *concurrent*) : l'exécution est simultanée, c'est à dire que plusieurs scripts s'exécutent en même temps.

Si le mot-clé (*interleaved*, *interrupted* ou *concurrent*) est omis, le mode d'appel est quelconque (mode par défaut). Si on veut exécuter plusieurs sous-scripts un à la fois sans entrelacement, ni interruption, ni de façon simultanée, il faut utiliser le mot-clé *ONEBYONE* ou *1B1*. Dans ce cas, lorsqu'un script est commencé, il doit être terminé avant qu'un autre puisse être exécuté. Dans ce type d'appel, l'ordre reste quelconque (ce n'est pas séquentiel). Un exemple d'appel de plusieurs sous-scripts dans un mode et un ordre quelconques est présenté dans les tableaux A2-IX (les colonnes *State*, *Feedback* et *Duration* sont omises) et A2-X (les colonnes *State* et *Duration* sont omises).

Tableau A2-IX

Exemple d'appels de sous-scripts

Step	Action
1	mouse.goto(mainFrame)
2	mouse.click
3	all of write(name) write(age) write(nationality) write(academic degree)

Tableau A2-X

Sous-script appelé *write(parameter)*

Step	Action	Feedback
1	Tab.press()	
2	keyboard.enter(parameter)	currentTextField.text

2.8 Actions de l'usager

Les actions sont exécutables soit en utilisant des dispositifs d'interaction ou par l'usager lui-même. Les sections qui suivent présentent les actions par catégories.

2.8.1 Dispositifs de pointage ou pointeurs

Les actions exécutables avec un pointeur sont les suivantes :

- *anyPointer.press* : l'usager appuie un bouton du pointeur. Par défaut, il s'agit du bouton de gauche. Cependant, il est possible de spécifier un autre bouton.

- *anyPointer.release* : l'usager relâche un bouton du pointeur. Par défaut, il s'agit du bouton de gauche.

- *anyPointer.click* : l'usager clique (appuie et relâche immédiatement) un bouton du pointeur. Par défaut, il s'agit du bouton de gauche.

- *anyPointer.goto(anyComponent)* : l'usager déplace le pointeur sur un composant graphique.

- *anyPointer.goto(anyComponent.upperLeft)* : l'usager déplace le pointeur sur le coin supérieur gauche d'un composant carré ou rectangulaire (ex. icône). On procède de la même manière pour les autres coins.

- *anyPointer.goto(anyComponent.left)* : l'usager déplace le pointeur sur le segment de gauche d'un composant carré ou rectangulaire. On procède de la même manière pour les autres segments.

- *anyPointer.goto(row,column)* : l'usager déplace le pointeur à une position absolue de l'écran.

- *anyPointer.goto (row,column,anyComponent)* : l'usager déplace le pointeur à une position relative dans un composant.

Remarque : il est possible de spécifier un type de pointeur en particulier. Pour ce faire, il faut mettre son identificateur avant l'identificateur de l'action. Dans le cas présent, l'identificateur *anyPointer* signifie que l'action peut être effectuée avec n'importe quel type de pointeur.

Un exemple d'usage de la souris est présenté au tableau A2-XI dans lequel la colonne *Duration* est omise. Dans cet exemple, l'usager déplace la souris dans une fenêtre d'identification électronique jusqu'au champ où il doit entrer son nom d'usager. Ensuite, il clique dans le champ avec le bouton de gauche.

Tableau A2-XI

Exemple d'action exécutée avec un pointeur

Step	Action	State	Feedback
1	repeat * times		
1.1	mouse.goto(x,y)		mouse.arrow
1.2	endrepeat		
2	mouse.goto(window.loginField)		mouse.arrow
3	mouse.leftButton.click	focus=cursor	cursor.blink

2.8.2 Fonctions de lecture de position

Il est possible de tester la position d'un pointeur à l'écran par le biais d'expressions booléennes. Elles sont les suivantes:

- *anyPointer ON anyComponent* ou *anyPointer = anyComponent* : vérifie si le pointeur est sur un composant graphique.

- *anyPointer ON (row, column)* ou *anyPointer = (row, column)* : vérifie si le pointeur est à une position absolue de l'écran.

- *anyPointer ON (row, column, anyComponent)* ou *anyPointer = (row, column, anyComponent)* : vérifie si le pointeur est à une position relative dans un composant.

- *anyPointer ON anyComponent.upperLeft ou anyPointer = anyComponent.upperLeft* : vérifie si le pointeur est sur un coin d'un composant carré ou rectangulaire.
- *anyPointer ON anyComponent.left ou anyPointer = anyComponent.left* : vérifie si le pointeur est sur un segment d'un composant carré ou rectangulaire.

Ces expressions sont utilisables avec des structures de contrôle comme en fait foi l'exemple du tableau A2-XII où la colonne *Duration* est omise. Dans cet exemple, l'usager effectue un clic double du pointeur avec le bouton de gauche s'il a bien positionné le pointeur sur l'icône de l'application. À la suite du clic double, l'application démarre et sa fenêtre apparaît à l'écran.

Tableau A2-XII

Exemple de test de position d'un pointeur

Step	Action	State	Feedback
1	if (anyPointer ON applicationIcon)		
1.1	repeat 2 times		
1.1.1	anyPointer.click	selected = applicationWindow	applicationWindow.display
1.1.2	endrepeat		
1.2	endif		

Plusieurs fonctions servent à récupérer les coordonnées de la position d'un composant. Elles sont les suivantes:

- *anyComponent.getX* : position absolue en abscisse d'un composant.
- *anyComponent.getY* : position absolue en ordonnée d'un composant.
- *(childComponent,parentComponent).getX* : position relative en abscisse d'un composant.
- *(childComponent,parentComponent).getY* : position relative en ordonnée d'un composant.
- *anyComponent.getPosition*: position relative ou absolue d'un composant en abscisse et en ordonnée.

Remarque : en cas d'ambiguïtés, il est possible d'ajouter un préfixe à *position*. Le préfixe est soit *absolute* ou *relative*.

2.8.3 Dispositifs d'interruption ou interrupteurs

Les actions qui sont exécutables avec un interrupteur sont les suivantes :

- *anySwitch.press(level)* : enfoncer un bouton de l'interrupteur au niveau spécifié parmi N niveaux.

Remarque : s'il y a un seul niveau, il est possible d'omettre ce dernier.

- *anySwitch.release* : relâcher un bouton de l'interrupteur.

- *anySwitch.click(level)* : enfoncer puis relâcher immédiatement un bouton de l'interrupteur au niveau spécifié.

Des fonctions servent à obtenir l'état d'un bouton d'un interrupteur. Elles sont les suivantes:

- *anySwitch.pressed(level)* : vérifie si un bouton est enfoncé au niveau spécifié.

- *anySwitch.released* : vérifie si un bouton est relâché.

- *anySwitch.clicked(level)* : vérifie si un bouton a été cliqué au niveau spécifié.

2.8.4 Dispositifs d'entrée de texte

La syntaxe ne permet qu'une seule fonction pour spécifier une entrée de texte. Voici les deux expressions possibles:

- *anyTextEntry.enter()* : l'usager est libre d'entrer ce qu'il veut.

- *anyTextEntry.enter(*« allo »*)* : l'usager doit entrer la chaîne de caractères « allo ».

L'expression suivante sert à recueillir le texte saisi.

- *anyTextEntry.entered* : la chaîne de caractère qui est entrée.

Remarque : ce qui peut être saisi comprend les éventuelles corrections (ex. touches DEL et BKSP), mais pas les touches spéciales qui ont un effet de déplacement (ex. touches ENTER et TAB).

Les tableaux A2-XIII et A2-XIV présentent des exemples d'entrée de texte avec le clavier et un microphone, respectivement. Dans ces exemples, la colonne *Duration* est omise. Dans le premier exemple, l'usager s'identifie sur un ordinateur en entrant le nom d'usager («admin») et le mot de passe («adminPass») de l'administrateur. Dans le second, l'usager démarre une application en disant le mot «activate» à l'aide d'un microphone relié à son ordinateur.

Tableau A2-XIII

Exemple d'une saisie de texte exécutée avec le clavier

Step	Action	State	Feedback
1	mouse.goto(textfield)		mouse.arrow
2	mouse.click	focus=cursor	cursor.blink
3	keyboard.enter(«admin»)	stored = «admin»	textField.text
4	Tab.click		
5	keyboard.enter(«adminPass»)	stored = «adminPass»	textField.text

Tableau A2-XIV

Exemple d'une saisie de texte effectuée avec un microphone

Step	Action	State	Feedback
1		inputMicrophone = ON	
2	voice.enter()	stored= voice.entered	

Tableau A2-XIV (suite)

Step	Action	State	Feedback
3	if (voice.entered == «activate»)		
3.1	activateApplication()		applicationWindow.display()
3.2	endif		

2.9 Usager

L'usager effectue parfois des actions implicites, c'est-à-dire des actions qui n'ont pas un effet direct sur l'interface. Les expressions les utilisant peuvent avoir les formes suivantes :

- *user.think* : l'usager effectue une pause mentale (ex. hésitation, réflexion).

- *user.home* : l'usager déplace sa main du clavier vers la souris (et vice et versa) ou du clavier principal vers le clavier numérique (et vice et versa).

Remarque : il est possible de spécifier une main en particulier (ex. *user.dominantHand.home*). Par défaut, c'est la main dominante qui est considérée.

- *user.gaze(position)* : l'usager fixe son regard sur une position de l'écran.

- *user.read* : l'usager lit le contenu présent à l'écran.

- *user.seek* : l'usager effectue une recherche visuelle à l'écran.

- *user.wait*: l'usager attend volontairement. Le temps d'attente est indiqué dans la colonne *Duration*.

Le tableau A2-XV présente un exemple d'actions implicites exécutées par l'usager dans lequel la colonne *Duration* est omise. Après avoir entré du texte au clavier, l'usager réfléchit sur ce qu'il veut faire et cherche le menu d'aide. Ensuite, il déplace sa main du clavier vers la souris et pointe la souris sur le menu d'aide.

Tableau A2-XV

Exemple d'actions implicites de l'usager

Step	Action	State	Feedback
1	keyboard.enter()	stored = keyboard.entered	
2	user.think		
3	user.seek		
4	user.dominantHand.home		
5	mouse.goto(helpMenu)		mouse.arrow

2.10 Rétroaction de l'interface

Les effets applicables sur les composants graphiques sont les suivants :

- *anyComponent.display* : le composant est affiché à sa position courante. Le mot *display* n'est pas obligatoire.

- *anyComponent.position* : le composant est affiché à la position spécifiée par position.

- *anyComponentGroup* : plusieurs composants d'un même type sont affichés à leur position courante (ex. un groupe d'icônes ayant les mêmes propriétés).

- *anyComponent.erase* : le composant est supprimé de l'écran.

Remarque : l'affichage et la suppression ont un effet permanent qui ne pourra être changé que par les instructions d'affichage ou de suppression suivantes.

- *anyComponent.activate(level)* : le composant est activé via un effet sur son apparence (ex. le composant a une apparence grisée). Il est possible de spécifier un niveau particulier.

- *anyComponent.deactivate* : le composant est désactivé. Son apparence est la même qu'avant d'être activé.

- *anyComponent.freeze* : le composant est gelé.

- *anyComponent.unfreeze* : le composant est dégelé.

- *anyComponent.toggle* : le statut d'un composant à deux états (ex. ON, OFF) passe d'un état à l'autre. Ceci est utilisable que pour un composant à deux états d'activation (ex. LED « light emitting diode » graphique).

- *anyComponent.blink* : le composant clignote à l'écran. En d'autres termes, il s'agit de la fonction *toggle* qui se répète indéfiniment.

Il est également possible de tester l'état des effets appliqués. Les expressions peuvent être représentées comme suit :

- *anyComponent.displayed* : vérifie si le composant est affiché.

- *anyComponent.activated(level)* : vérifie si le composant est activé au niveau spécifié.

- *anyComponent.frozen* : vérifie si le composant est gelé.

- *anyComponent.blinking*: vérifie si le composant clignote.

Une autre fonction de rétroaction, autre qu'un effet sur un composant graphique, est la suivante :

delay: délai de réponse de l'interface. La durée est indiquée soit dans la colonne *Feedback* ou la colonne *Duration*.

L'exemple présenté au tableau A2-XVI illustre les effets visibles sur les composants graphiques lors de l'action de déplacement d'une icône. Dans cet exemple, les colonnes *State* et *Duration* sont omises.

Tableau A2-XVI

Exemple d'effets appliqués sur des composants graphiques

Step	Action	Feedback
1	mouse.goto(fileIcon)	mouse.arrow
2	mouse.press	fileIcon.activate
3	repeat * times	
3.1	mouse.goto(x,y)	mouse.arrow && fileIcon.outline
3.2	endrepeat	

Tableau A2-XVI (suite)

Step	Action	Feedback
4	mouse.release	fileIcon
5	mouse.goto(x,y)	mouse.arrow
6	mouse.click	fileIcon.deactivate

2.11 État ou changement d'état de l'interface

Dans la spécification d'un état ou changement d'état du système, un mot-clé, une valeur numérique ou une expression est assigné(e) à une variable d'état matériel ou logiciel via un opérateur d'affectation (l'opérateur d'égalité). Les expressions utilisant une variable d'état matériel ont la forme suivante :

- *inputDevice* = *ON* (ou *OFF*) : L'identificateur *inputDevice* représente tout dispositif d'interaction (ex. clavier, souris, écran). Les deux états possibles sont :

  ON : le composant matériel est sous le contrôle de l'usager. C'est la valeur utilisée par défaut.

  OFF : le composant matériel est sous le contrôle du système. Dans ce cas, cela doit être spécifié explicitement.

Les variables d'état logiciel sont :

- *focus* : le composant actif.

- *selected* : un composant qui est sélectionné

- *stored* : ce qui est emmagasiné en mémoire.
- *removed* : ce qui est effacé de la mémoire.
- *frozen* : un composant qui est en état de gel.
- *idle*: un composant qui est dans un état oisif (ou neutre).
- *idletime*: temps de l'état oisif.

Dans l'exemple du tableau A2-XVII (colonne *Duration* omise), l'usager positionne la souris sur un champ de texte et clique avec le bouton de gauche. Le champ est alors sélectionné dans le système. Ensuite, il entre du texte dans le champ. Le texte saisi est stocké en mémoire.

Tableau A2-XVII

Exemple de changement d'état du système

Step	Action	State	Feedback
1	mouse.goto(textField)		mouse.arrow
2	mouse.click	selected = textField	
3	keyboard.enter	stored = keyboard.entered	textField.txt

2.12 Durée des actions

La durée des différentes étapes d'un script AIR peut être représentée de trois façons différentes, soit :

- Spécifiée (mot-clé : *specified*) : la durée est spécifiée par le concepteur. C'est le type par défaut.

- Estimée (mot-clé : *estimated*) : la durée est obtenue par modélisation analytique (ex. KPC, KLM).

- Mesurée (mot-clé : *measured*): la durée est mesurée à partir d'un enregistrement des activités de l'usager dans un journal d'événements.

Dans l'exemple du tableau A2-XVIII (colonnes *State* et *Feedback* omises), on peut voir le temps d'exécution estimé par le modèle KPC (chapitre 4) pour la tâche de déplacement d'une icône. Il faut souligner que l'unité de temps par défaut est la seconde.

Tableau A2-XVIII

Exemple de représentation de la durée (temps d'exécution)

Step	Action	Duration
1	mouse.goto(fileIcon)	700 ms estimated
2	mouse.press	235 ms estimated
3	repeat * times	
3.1	mouse.goto(x,y)	0.7 s estimated
3.2	endrepeat	
4	mouse.release	0.235 estimated

3. Règles d'écriture

On propose ce qui suit comme règles d'écriture :

- Indenter le corps des structures de contrôle (recommandé).
- Ne pas dépasser trois niveaux d'indentation (obligatoire).
- Utiliser le plus souvent les structures de contrôle de base (facultatif).
- Si nécessaire, utiliser les structures dérivées (facultatif).
- Tous les noms d'identificateurs doivent respecter la convention établie (obligatoire).
- Dans le cas d'un dispositif ou d'un composant spécifique, on évite d'utiliser le préfixe *any* et on utilise un identificateur représentatif (recommandé).
- Lorsqu'un effet est appliqué sur plusieurs composants de même type, on utilise l'identificateur de groupe (ex. *anyButtonGroup*) (recommandé).
- Terminer une structure de contrôle avec le mot-clé approprié (ex. *endif*, *endrepeat*) (facultatif).
- Numéroter les étapes d'un script selon les normes spécifiées (obligatoire).
- Rédiger des glossaires pour les définitions de nouveaux dispositifs, de nouvelles caractéristiques de composants, de nouvelles actions et de nouveaux états du système (obligatoire).
- Les fonctions booléennes sont utilisées avec les structures de contrôle (obligatoire).

4. Compléments de syntaxe

Voici des possibilités syntaxiques additionnelles :

- Grouper des actions au moyen de parenthèses et les séparer au moyen de virgules ou d'espaces.

- On peut omettre les parenthèses partout pourvu qu'il n'y ait pas d'ambiguïté.

- Pour répéter des actions, on peut utiliser les exposants :

    - $action^{number}$: nombre prédéterminé de fois.
    - $action^{number..number}$: nombre de fois situé dans une plage.
    - $action^{>=number}$: nombre de fois supérieur ou égale à une constante.
    - $action^{<=number}$: nombre de fois inférieur ou égale à une constante.

Remarque : on peut aussi utiliser le symbole « ^ » pour indiquer un exposant (ex. action^2).

5. Glossaires d'identificateurs

Cette section présente des tableaux récapitulatifs de la syntaxe AIR. Ils contiennent des expressions génériques avec une description et parfois fournissent même une proposition de syntaxe alternative. Dans ces tableaux, le préfixe *any* n'est pas utilisé.

Tableau A2-XIX

Glossaire des actions exécutables avec un pointeur et celles exécutables avec un dispositif d'entrée de texte

Syntaxe	Description	Syntaxe alternative
pointer.goto (component)	Déplacement du pointeur sur la poignée du composant	*~[component]*
pointer.goto (component.upperLeft)	Déplacement du pointeur sur un coin d'un composant carré ou rectangulaire	*~[component]^{ul}*
pointer.goto (component.lowerLeft)	Idem	*~[component]^{ll}*
pointer.goto (component.upperRight)	Idem	*~[component]^{ur}*
pointer.goto (component.lowerRight)	Idem	*~[component]^{lr}*
pointer.goto (component.left)	Déplacement du pointeur sur un segment d'un composant carré ou rectangulaire	*~[component]^{l}*
pointer.goto (component.right)	Idem	*~[component]^{r}*
pointer.goto (component.bottom)	Idem	*~[component]^{b}*
pointer.goto (component.top)	Idem	*~[component]^{t}*
pointer.goto (row,column)	Déplacement du pointeur à une position absolue de l'écran	*~(row,column)*
pointer.goto (row,column,component)	Déplacement du pointeur à une position relative	*~(row,column,component)*
textEntry.enter	Texte saisi avec un dispositif d'entrée de texte (l'usager entre ce qu'il veut)	
textEntry.enter(string)	Texte saisi avec un dispositif d'entrée de texte (le texte est forcé)	
textEntry.entered(string)	Récupération du texte saisi	

Tableau A2-XX

Glossaire des fonctions de test de position d'un pointeur

Syntaxe	Description	Syntaxe alternative
pointer = component *pointer ON component*	Vérifie si le pointeur est sur la poignée du composant	*pointer ?* *ON [component]*
pointer = (row,column) *pointer ON (row,column)*	Vérifie si le pointeur est à la position absolue spécifiée	*pointer ?* *ON(row,column)*
pointer = *(row,column,component)* *pointer ON* *(row,column,component)*	Vérifie si le pointeur est à la position relative spécifiée	*pointer ?* *ON(row,column,component)*
pointer = component.upperLeft *pointer ON component.upperLeft*	Vérifie si le pointeur est sur un des coins d'un composant carré ou rectangulaire	*pointer ?* *ON[component]*^{ul}
pointer = component.upperRight *pointer ON* *component.upperRight*	Idem	*pointer ?* *ON[component]*^{ur}
pointer = component.lowerLeft *pointer ON component.lowerLeft*	idem	*pointer ?* *ON[component]*^{ll}
pointer = component.lowerRight *pointer ON* *component.lowerRight*	Idem	*pointer ?* *ON[component]*^{lr}
pointer = component.top *pointer ON component.top*	Vérifie si le pointeur est sur un des segments d'un composant carré ou rectangulaire	*pointer ?* *ON[component]*^{t}
pointer = component.bottom *pointer ON component.bottom*	Idem	*pointer ?* *ON[component]*^{b}
pointer = component.left *pointer ON component.left*	Idem	*pointer ?* *ON[component]*^{l}
pointer = component.right *pointer ON component.right*	idem	*pointer ?* *ON[component]*^{r}

Tableau A2-XXI

Glossaire des fonctions de lecture de position d'un composant graphique

Syntaxe	Description	Syntaxe alternative
component.getX	Récupération de la position absolue en abscisse d'un composant	*@X(component)*
component.getY	Idem mais en ordonnée	*@Y(component)*
(childComponent,parentComponent).getX	Récupération de la position relative en abscisse d'un composant	*@X(childComponent, parentComponent)*
(childComponent,parentComponent).getY	Idem mais en ordonnée	*@Y(childComponent, parentComponent)*
component.getPosition	Récupération de la position d'un composant (relative ou absolue)	*@component*

Tableau A2-XXII

Glossaire des actions exécutables avec un interrupteur

Syntaxe	Description	Syntaxe alternative
switch.press(level)	Presse d'un bouton de l'interrupteur au niveau spécifié	*X∨(level)*
switch.release	Relâchement d'un bouton de l'interrupteur	*X∧*
switch.click(level)	Clic d'un bouton de l'interrupteur au niveau spécifié	*X∨∧(level)*
switch.pressed(level)	Vérifie si un bouton de l'interrupteur est enfoncé au niveau spécifié	*? X∨(level)*
switch.released	Vérifie si un bouton de l'interrupteur est relâché	*? X∧*
switch.clicked(level)	Vérifie si un bouton de l'interrupteur a été cliqué au niveau spécifié	*? X∨∧(level)*

Tableau A2-XXIII

Glossaire des actions implicites exécutables par l'usager

Syntaxe	Description	Syntaxe alternative
user.gaze(position)	Le regard est fixé sur une position de l'écran (absolue ou relative)	*eye(position)* *look(position)*
user.think	Pause mentale	
user.read	Lecture à l'écran	
user.seek	Recherche visuelle à l'écran	*search*
user.wait	Attente volontaire	
user.home	Déplacement de la main du clavier vers la souris et vice et versa ou du clavier principal vers le clavier numérique et vice et versa	
user.dominantHand.home	Déplacement de la main dominante	
user.nonDominantHand.home	Déplacement de la main non dominante	

Tableau A2-XXIV

Glossaire des rétroactions de l'interface

Syntaxe	Description	Syntaxe alternative
component	Affichage d'un composant à sa position courante	
component.display	Affichage d'un composant à sa position courante	
component.position	Affichage d'un composant à la position spécifiée.	
component.displayed	Vérifie si le composant est affiché	
component.erase	Suppression du composant	
component.erased	Vérifie si le composant a été supprimé	
component.activate(level)	Activation du composant au niveau spécifié	*!(component, level)* *ON(component, level)*
component.activated(level)	Vérifie si le composant est activé au niveau spécifié	
component.deactivate	Désactivation du composant	*OFF(component)*
component.toggle	Composant à deux états passant d'un état à l'autre	
component.blink	Clignotement du composant	*! component !*

Tableau A2-XXIV (suite)

Syntaxe	Description	Syntaxe alternative
component.blinking	Vérifie si le composant clignote	
component.freeze	Composant en état de gel	
component.frozen	Vérifie si le composant est en état de gel	
component.unfreeze	Dégel du composant	
delay	Délai de réponse du système	

Tableau A2-XXV

Glossaire des variables d'état (matériel et logiciel) de l'interface

Variable d'état	Description
inputDevice	État matériel d'un dispositif d'interaction
focus	Composant actif
selected	Composant sélectionné
stored	Ce qui est stocké en mémoire
removed	Ce qui est effacé de la mémoire
frozen	Composant en état de gel
idle	Composant à l'état oisif ou neutre
idletime	Durée de l'état oisif

Tableau A2-XXVI

Glossaire des structures de contrôle du flot d'exécution

Nom	Type
IF	Conditionnelle (de base)
SWITCH	Conditionnelle (dérivée)
FOR	Iterative (de base)
REPEAT	Itérative (dérivée)
WHILE	Itérative (dérivée)
DO WHILE	Itérative (dérivée)
STEP	Saut direct (de base)
BREAK	Saut direct (dérivée)
CONTINUE	Saut direct (dérivée)
RETURN	Saut direct (dérivée)

Tableau A2-XXVII

Glossaire des opérateurs unaires et binaires

Opérateur	Signification	Ordre
+	Union	De gauche à droite
-	Différence	Idem
*	Intersection	Idem
()	Parenthèses	Idem
==	Égalité	Idem
!=	Inégalité	Idem
<	Inférieur	Idem
<=	Inférieur ou égal	Idem
>	Supérieur	Idem
>=	Supérieur ou égal	Idem
&&	ET logique	Idem
||	OU logique	Idem
-	Négation arithmétique	De droite à gauche
!	Négation logique	Idem
=	Affectation	Idem

ANNEXE 3

Vérificateur syntaxique – dictionnaire des identificateurs

Le dictionnaire utilisé par le programme de vérification syntaxique (chapitre 3) est présenté au tableau A3-I. Il contient tous les identificateurs de la syntaxe de base et leur type.

Tableau A3-I

Programme de vérification syntaxique – dictionnaire des identificateurs

Identificateur	Type
times, on, off, of	mot-clé (*keyword*)
specified, estimated, measured	type de la durée (*dtimetype*)
ms, s, m, h	type d'unité de la durée (*dtimeunit*)
if, else, endif, for, endfor, do, while, endwhile, switch, endswitch, repeat, endrepeat, case, break, continue, return, step	structure de contrôle de flot (*control_structure*)
+, -, ++, --, !, |, &	opérateur unaire (*uoperator*)
*, /, =, ==, <, >, ||, &&	opérateur binaire (*boperator*)
anyPointer, mouse, joystick	pointeur (*ptr*)
anyTextEntry, keyboard, voice, touchScreen, writingPad	dispositif d'entrée de texte (*tentry*)
anySwitch	interrupteur. (*switch*)
user, anyHand, dominantHand, nonDominantHand	usager (*usr*)
goto	action exécutable soit avec un pointeur ou par l'usager (*pou_fct*)
press, release, click	action exécutable soit avec un pointeur ou un interrupteur (*pos_fct*)
seek, read, gaze, think, home, wait	action exécutable par l'usager (*usr_fct*)
arrow, leftButton, centerButton, rightButton, scrollButton	sous-composant matériel d'un pointeur ou d'un interrupteur (*pos_comp*)
enter	fonction d'entrée de texte (*tentry_fct*)

Tableau A3-I (suite)

Identificateur	Type
selected, focus, stored, removed, idle, idleTime, frozen	variable d'état logiciel (*soft_st_op*)
inputMouse, inputKeyboard, inputVoice, inputTouchScreen, inputWritingPad	variable d'état d'un dispositif matériel (*hard_st_op*)
wait	fonction quelconque (*fct*)
activate, deactivate, activated, display, displayed, erase, erased, toggle, blink, unblink, blinking, freeze, unfreeze, frozen	effet applicable à un composant graphique ou état de l'effet (*i_comp_fct*)
delay	délai de réponse de l'interface (*response_fct*)

ANNEXE 4

Vérificateur syntaxique – type et spécificité des identificateurs

Les identificateurs utilisés dans un script AIR ont tous un type, qu'il soit défini dans le dictionnaire statique (fourni avec le langage), déclaré par l'usager ou défini par le programme de façon dynamique (voir chapitre 3). Les identificateurs qu'on peut déclarer sont ceux associés aux : i) dispositifs d'interaction, ii) composants graphiques, iii) effets associés aux composants graphiques et iv) sous-scripts et paramètres. Les identificateurs non déclarables sont ceux associés aux fonctions prédéfinies. Le tableau A4-I présente les différents types, leur signification et s'ils peuvent ou non être déclarés par l'usager.

Tableau A4-I

Programme de vérification syntaxique – types d'identificateurs et déclaration

Type d'identificateur	Signification	Déclarable (Oui/Non)
TOKEN	Type quelconque	Oui
POU (*« Pointer or user»*)	Pointeur ou usager	Oui
POS (*« Pointer or switch»*)	Pointeur ou interrupteur	Oui
POS_COMP (*« Pointer or switch component»*)	Composant matériel ou logiciel d'un pointeur ou d'un interrupteur	Oui
PTR (*« Pointer »*)	Pointeur	Oui
USR (*« User »*)	Usager	Oui
TENTRY (*« Text entry »*)	Dispositif d'entrée de texte	Oui
SWITCH	Interrupteur	Oui
FCT (*« Function »*)	Fonction quelconque (ex. : wait)	Non
USR_FCT (*« User function »*)	Action exécutable par l'usager	Non
POU_FCT (*« Pointer or user function »*)	Action exécutable avec un pointeur ou par l'usager	Non
POS_FCT (*« Pointer or switch function »*)	Action exécutable avec un pointeur ou un interrupteur	Non
TENTRY_FCT (*« Text entry function »*)	Fonction d'entrée de texte (*enter*)	Non
SOFT_ST_OP (*« Software state operator »*)	Variable d'état logiciel	Non
HARD_ST_OP (*« Hardware state operator »*)	Variable d'état matériel	Non
CONTROL_STRUCTURE (*« Control structure »*)	Structure de contrôle du flot d'exécution	Non

Tableau A4-I (suite)

Type d'identificateur	Signification	Déclarable (Oui/Non)
I_COMP (« *Interface component* »)	Composant graphique	Oui
I_COMP_FCT (« *Interface component function* »)	Effet applicable à un composant graphique	Oui
SCRIPTNAME (« Script name »)	Nom d'un sous-script	Oui
SCRIPTPARAMETER (« Script parameter »)	Paramètre d'un sous-script	Oui

Assignation dynamique de types

Lorsqu'un identificateur n'est pas présent dans le dictionnaire statique et qu'il n'est pas déclaré par l'usager, un type lui est assigné de façon dynamique par le programme de vérification syntaxique à partir de la nature de l'expression. Pour ce faire, le programme utilise une table de spécificité des identificateurs (tableau A4-II). Premièrement, le type le moins spécifique est associé (« *token* »). Si l'expression contient assez d'informations pour lui assigner un type plus spécifique (ambigu ou final), alors ce nouveau type lui est assigné. Dans le cas contraire, le type reste inchangé. Les niveaux de spécificité des types sont présentés au tableau A4-II. Plus le chiffre est petit, plus le type est spécifique. La figure A4-1 illustre le fonctionnement de l'assignation de types.

Tableau A4-II

Niveaux de spécificité des types

Type d'identificateur	Niveau de spécificité
ptr	1 (final)
usr	1
tentry	1
switch	1
i_comp	1
scriptname	1
scriptparameter	1
pou	2 (ambigu)
pos	2
token	3 (quelconque)

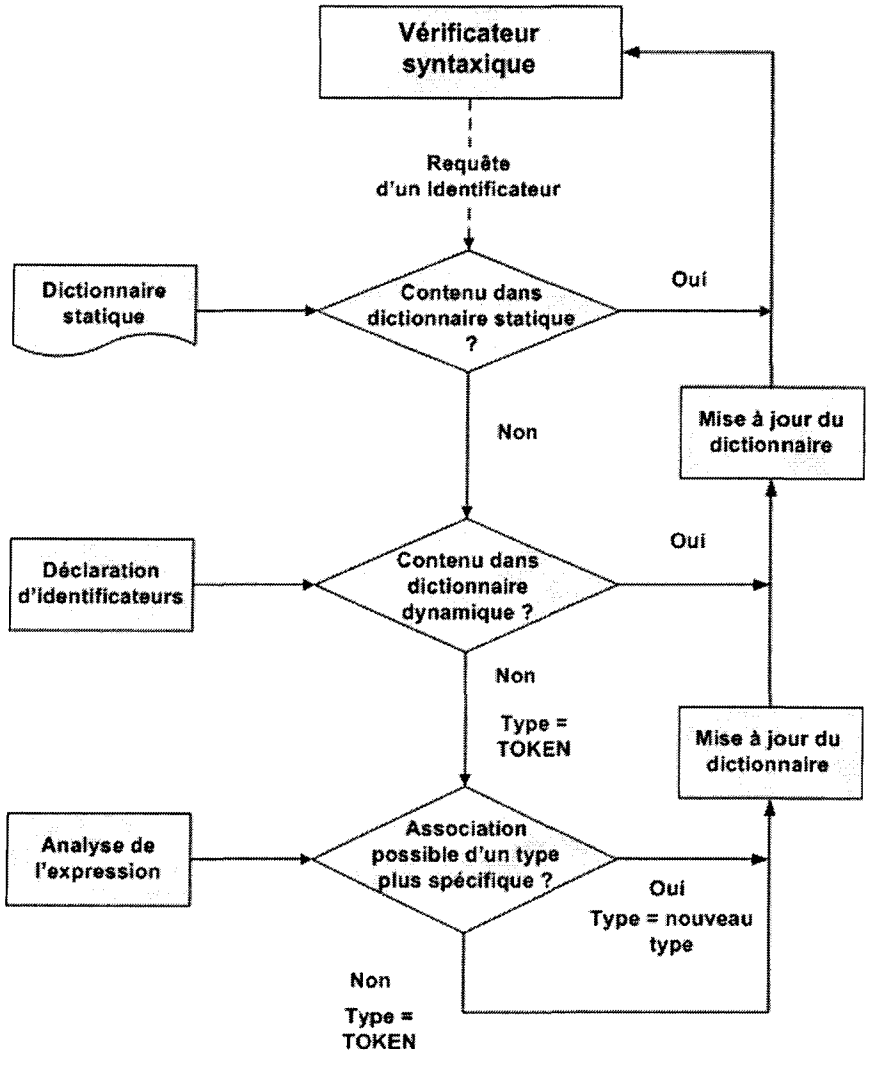


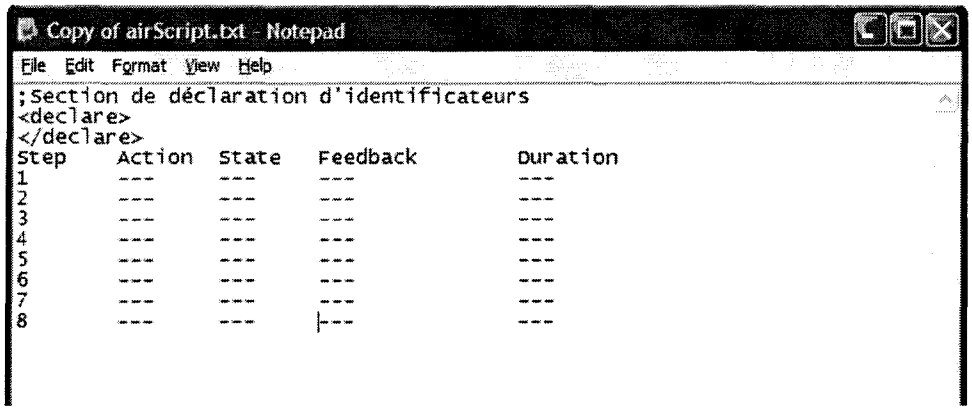
Figure A4-1 Assignation dynamique de types pour les identificateurs AIR

ANNEXE 5

Outils logiciels - patrons de fichiers d'entrée

La figure A5-1 présente les patrons de fichiers d'entrée à utiliser pour créer des scripts AIR soit en format TXT (texte) ou HTML (voir chapitre 3).

a) fichier TXT



b) fichier HTML

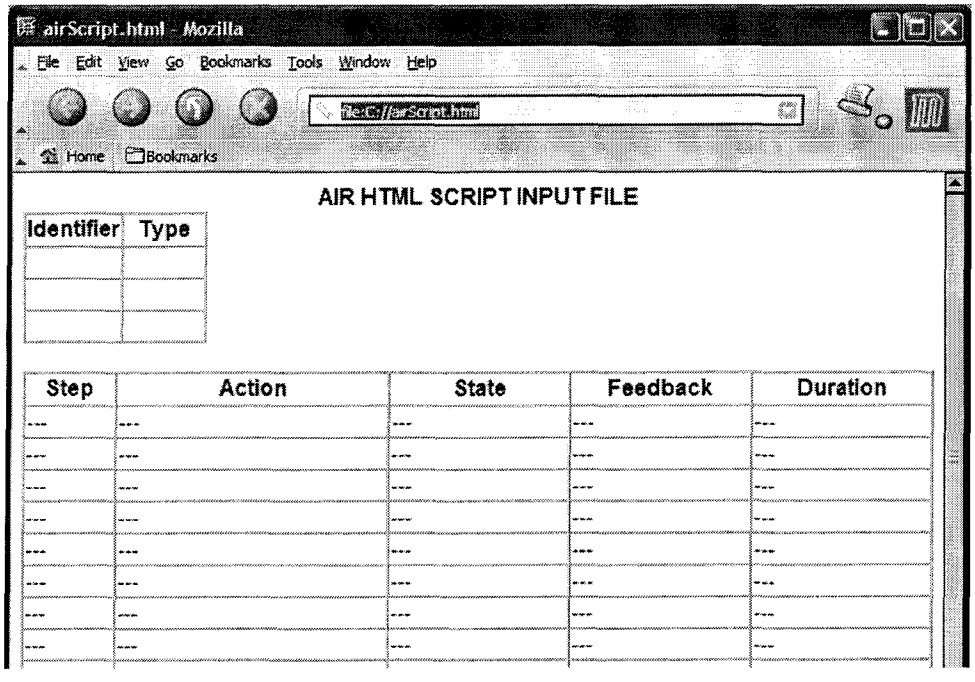


Figure A5-1 Outils logiciels – patrons de fichiers d'entrée

ANNEXE 6

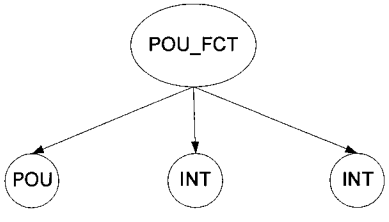
Vérificateur syntaxique – construction des arbres syntaxiques

Une vérification syntaxique d'un script AIR qui est réussie résulte en la création d'arbres syntaxiques à raison d'un par expression du script. Les arbres sont construits après vérification du type des identificateurs constituant une expression. Les sections suivantes présentent les arbres qu'il est possible de construire pour les types d'expressions suivantes : action de l'usager, état interne du système et rétroaction de l'interface. Il est à noter que la signification des identificateurs utilisés ici est la même que dans les annexes précédentes.

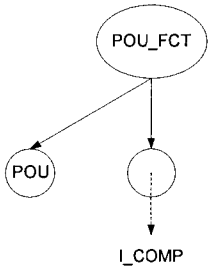
1. Colonne *Action*

Pointeur ou usager

a) exemple : *mouse.goto(10,20)*



b) exemple : *anyPointer.goto(window)*



c) exemple: *mouse.goto(window.okButton)*

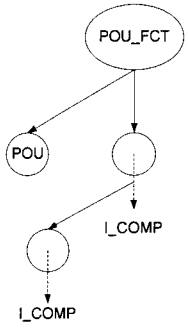


Figure A6-1 Vérificateur syntaxique - arbres syntaxiques pour les actions exécutables avec un pointeur ou par l'usager

Usager seulement

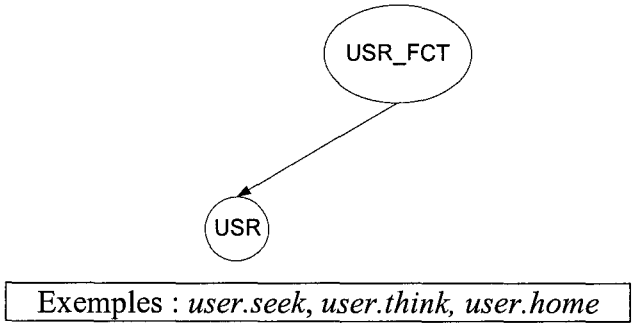


Figure A6-2 Vérificateur syntaxique – arbre syntaxique pour une action exécutable par l'usager

Dispositif d'entrée de texte

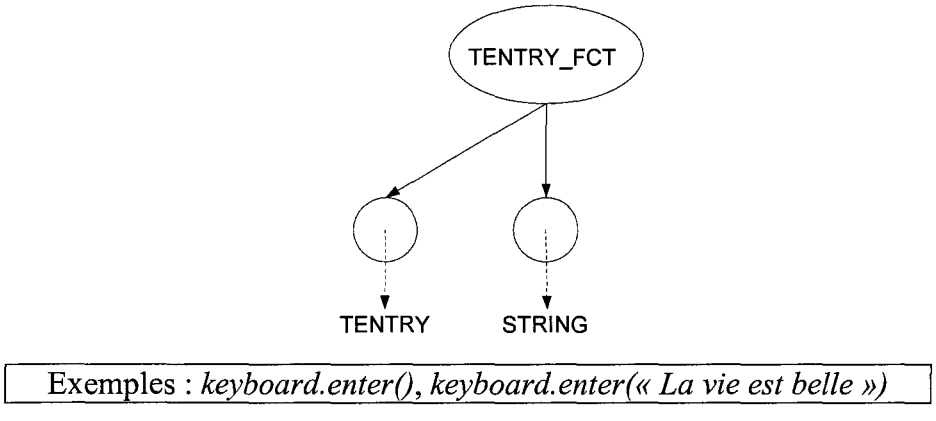
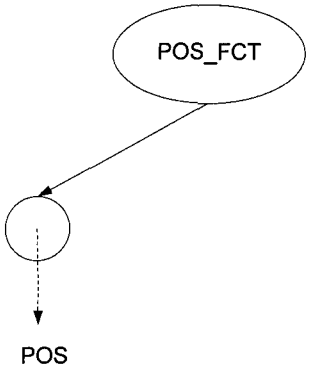


Figure A6-3 Vérificateur syntaxique - arbre syntaxique pour une action exécutable avec un dispositif d'entrée de texte

Pointeur ou interrupteur

a) exemple : *mouse.click*



b) exemple: *mouse.leftButton.press*

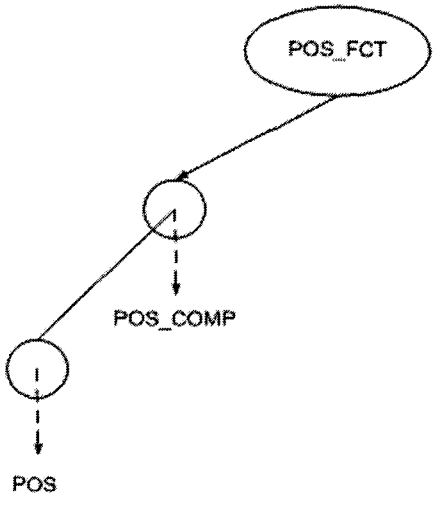


Figure A6-4 Vérificateur syntaxique - arbres syntaxiques pour une action exécutable avec un pointeur ou un interrupteur

Appel de script

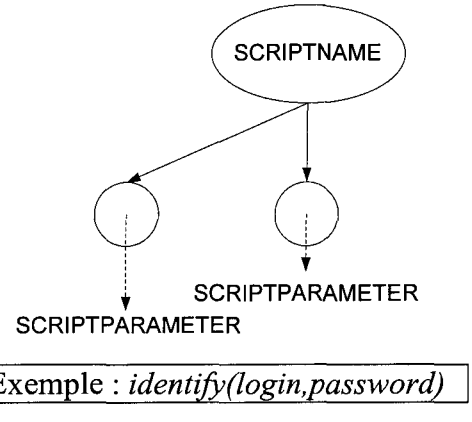
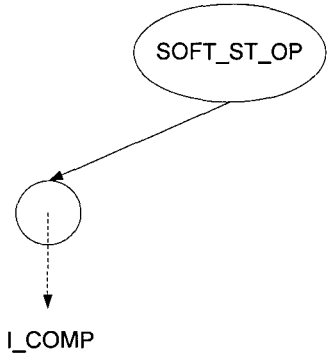


Figure A6-5 Vérificateur syntaxique - arbre syntaxique pour l'appel d'un sous-script

2. Colonne *State*

Variables d'état

a) exemple : *selected = okButton*



b)) exemple : *inputMouse = ON*

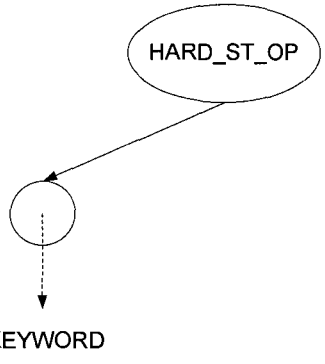


Figure A6-6 Vérificateur syntaxique - arbres syntaxiques pour la spécification d'un état ou changement d'état

3. Colonne *Feedback*

Sous-composant logiciel d'un pointeur ou d'un interrupteur

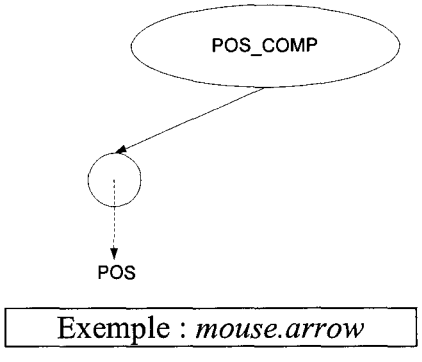
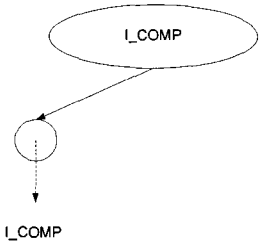


Figure A6-7 Vérificateur syntaxique - arbre syntaxique pour l'affichage d'un composant logiciel d'un pointeur ou d'un interrupteur

Composant graphique

a) ex. : *applicationWindow*

b) ex. : *applicationWindow.dialogBox*



c) exemple : *fileMenu.activate*

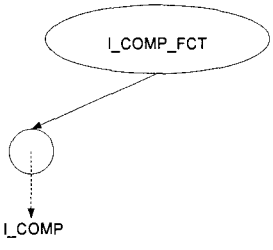
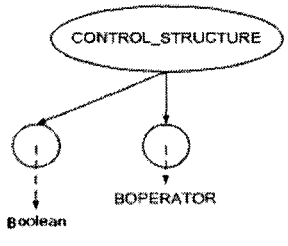


Figure A6-8 Vérificateur syntaxique - arbres syntaxiques pour les effets applicables sur des composants graphiques

4. Arbres communs à plusieurs colonnes (*Actions*, *State* et *Feedback)*

Structures de contrôle de flot d'exécution

a) exemple : *while (i < counter)*



b) exemple : *step 2.1*

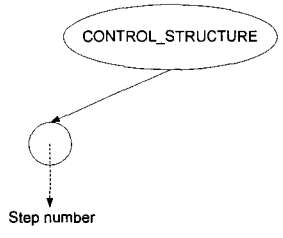


Figure A6-9 Vérificateur syntaxique - arbres syntaxiques pour les structures de contrôle du flot d'exécution

Fonction générique

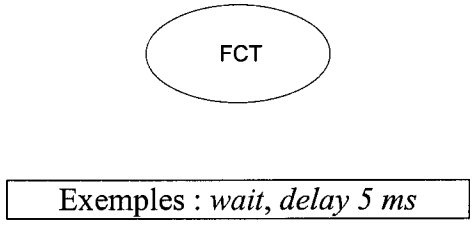


Figure A6-10 Vérificateur syntaxique - arbre syntaxique pour une fonction générique

ANNEXE 7

Test de l'outil de vérification syntaxique – jeux d'essai

Les prochaines sections présentent les jeux d'essai utilisés afin de tester le fonctionnement du programme de vérification syntaxique pour les cinq types d'expressions AIR.

Colonne Step

Le tableau A7-I présente le jeu d'essai utilisé pour une expression de la colonne *Step*. Le type *« int »* signifie une valeur entière.

Tableau A7-I

Vérificateur syntaxique –
jeu d'essai pour une expression de la colonne *Step*

Expression	Résultat attendu
int	Valide
int.int	Valide
int.int.int.	Valide

Colonne Action

Le tableau A7-II présente le jeu d'essai utilisé pour une expression de la colonne *Action*. Les types des identificateurs sont indiqués pour des raisons de clarté. Dans des cas d'ambiguïtés, un identificateur précis est spécifié. Les types *scriptname* et *scriptparameter* servent à spécifier un nom de script et un paramètre de script, respectivement.

Tableau A7-II

Vérificateur syntaxique – jeu d'essai pour une expression de la colonne *Action*

Expression	Résultat attendu
Les identificateurs sont soit dans le dictionnaire statique ou déclarés par l'usager	
fct.fct	invalide
i_comp.fct()	invalide
pou.pou_fct (	invalide
pou.pou_fct (i_comp)	valide
pou.pou_fct (i_comp	Invalide
pou.pou_fct (i_comp, i_comp)	valide
pou.pou_fct (int, int)	valide
pou .pou_fct()	valide
pou.pou_fct ()	valide
pou. pou_fct()	valide
pou . pou_fct()	valide
pou.pou_fct(int, int)	valide
pou.pou_fct (int, int)	valide
usr.usr_fct()	valide
usr.usr_fct(	invalide
usr .usr_fct()	valide
usr.usr_fct ()	valide
usr. usr_fct()	valide
usr . usr_fct()	valide
pos.pos_fct()	valide
pos.pos_comp.pos_fct()	valide
pos.pos_fct(	invalide
pos .pos_fct()	valide
pos.pos_fct ()	valide
pos. pos_fct()	valide
pos . pos_fct()	valide
tentry.tentry_fct()	valide
tentry.tentry_fct (	invalide
tentry.tentry_fct(« La vie est belle »)	valide
tentry . tentry_fct ()	valide
tentry. tentry_fct ()	valide
tentry. tentry_fct ()	valide
tentry . tentry_fct ()	valide
fct	valide
if i = int	valide
if i == int	valide
if (i = int)	valide
if (i ==int)	valide
if (i ==int	invalide
if i == int)	invalide
If pou ON i_comp	valide

Tableau A7-II (suite)

Expression	Résultat attendu
if pou ON i_comp	valide
for (i = 0; i < 10; i++)	valide
for (i = 0; i < 10; i++)	valide
switch X	valide
case X	valide
while (i== int)	valide
while (int)	valide
control_structure	valide
step int	valide
step int	valide
step int.int	valide
step int.int	valide
step int.int.int	valide
repeat * times	valide
repeat X times	valide
repeat * times	valide
repeat * times	valide
repeat * times	valide
repeat *	valide
repeat int	valide
scriptname(scriptparameter,scriptparameter)	valide
scriptname	valide
Certains des prochains identificateurs ne sont pas dans le dictionnaire statique ni déclarés par l'usager	
pou.pou_fct() pou non declaré	valide + message d'ambiguïté type assigné = pou
usr.usr_fct() usr non déclaré	valide type assigné = usr
pos.pos_fct pos non déclaré	valide + message d'ambiguïté type assigné = pos
tentry.tentry_fct tentry non declaré	valide type assigné = tentry
scriptname(scriptparameter,scriptparameter) scriptname non déclaré	valide type assigné = scriptname
scriptname(scriptparameter,scriptparameter) scriptparameter non déclaré	valide type assigné = scriptparameter

Colonne State

Le tableau A7-III présente le jeu d'essai utilisé pour une expression de la colonne *State.*

Tableau A7-III

Vérificateur syntaxique – jeu d'essai pour une expression de la colonne *State*

Expression	Résultat attendu
Les prochains identificateurs sont soit dans le dictionnaire statique ou déclarés par l'usager	
hard_st_op = ON/OFF	Valide
soft_st_op = i_comp	valide
soft_st_op = i_comp.i_comp	valide
if soft_st_op == i_comp	valide
if (soft_st_op == i_comp)	valide
if soft_st_op = i_comp	valide
if (soft_st_op = i_comp)	valide
Certains des prochains identificateurs ne sont pas dans le dictionnaire statique ni déclarés par l'usager	
hard_st_op = ON/OFF hard_st_op non déclaré	valide type assigné = hard_st_op
soft_st_op = i_comp soft_st_op non déclaré	valide type assigné = soft_st_op
soft_st_op = i_comp i_comp non déclaré	valide type assigné = i_comp

Colonne Feedback

Le tableau A7-IV présente le jeu d'essai utilisé pour une expression de la colonne *Feedback.*

Colonne Duration

Le tableau A7-V présente le jeu d'essai utilisé pour une expression de la colonne *Duration.* Le type *int* sert à spécifier une valeur entière alors que les secondes, millisecondes, minutes et heures sont spécifiées par les abréviations s, ms, m et h,

respectivement. Les mots-clés spécifiant le type de durée sont utilisés pour une raison de clarté.

Tableau A7-IV

Vérificateur syntaxique – jeu d'essai pour une expression de la colonne *Feedback*

Expression	Résultat attendu
Les prochains identificateurs sont soit dans le dictionnaire statique ou déclarés par l'usager	
i_comp	valide
i_comp. i_comp	valide
i_comp. i_comp. i_comp	valide
i_comp. i_comp_fct	valide
i_comp. i_comp_fct (	invalide
i_comp. i_comp. i_comp_fct ()	valide
if i_comp. i_comp_fct	valide
if (i_comp. i_comp_fct)	valide
Certains des prochains identificateurs ne sont pas dans le dictionnaire statique ni déclarés par l'usager	
i_comp.i_comp i_comp non déclaré	valide type assigné = i_comp
i_comp.i_comp_fct i_comp non declaré	valide type assigné = i_comp
i_comp_fct.i_comp_fct()	invalide

Tableau A7-V

Vérificateur syntaxique – jeu d'essai pour une expression de la colonne *Duration*

Expression	Résultat attendu
int	valide Note : l'unité de temps et le mode par défaut sont rajoutés (s et *specified*, respectivement)
int s	valide Note : le mode par défaut est rajouté
int *measured*	valide Note : l'unité de temps par défaut est rajoutée
int ms *estimated*	valide
int ms *measured*	valide
int ms *specified*	valide
float m *specified*	valide
int h *specified*	valide

ANNEXE 8

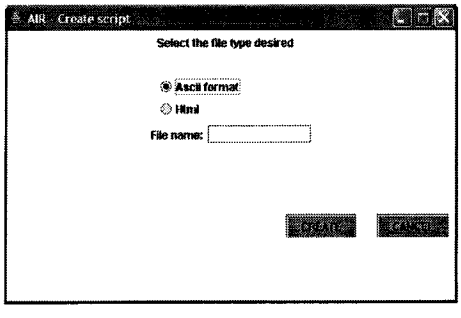
Outils logiciels – interface usager

On présente à la figure A8-1 les quatre écrans de l'interface usager des outils logiciels AIR (chapitre 3).

a) écran d'accueil

b) options de fichiers

c) création d'un script



d) options de l'outil intégré

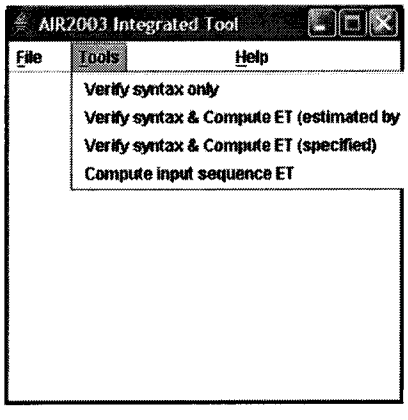


Figure A8-1 Outils logiciels AIR – interface usager

ANNEXE 9

Enquête comparative : AIR vs KLM et UAN

On présente ici l'enquête effectuée auprès d'usagers potentiels du langage AIR. Cette enquête visait à recueillir des informations quantitatives mais surtout qualitatives sur le langage AIR (chapitre 2) ainsi que sur la notation UAN et le modèle KLM (voir chapitre 1) afin de les comparer entre eux. Dans un premier temps, la méthodologie d'enquête est présentée. Ensuite, on présente les résultats obtenus. Finalement, on fait ressortir quelques limitations de cette enquête au niveau de la méthodologie.

1. Matériel et méthodes

Déroulement de l'enquête

Le sujet type choisi pour cette étude était un étudiant du baccalauréat en génie logiciel de l'ÉTS. Comme futur ingénieur, ce dernier devra effectuer de la conception d'interfaces centrée sur l'usager. Deux groupes de sujets ont participé à l'étude. Les tailles étaient de 40 (groupe 1) et 38 (groupe 2) sujets, respectivement. Les sujets de sexe féminin représentaient 10 % et 7.5 % de la taille des groupes 1 et 2, respectivement. Tous les sujets étaient des étudiants du cours *Analyse et conception d'interfaces usager (LOG340)* du département de génie logiciel de l'ÉTS.

Dans le cadre d'un devoir du cours, une ou deux tâches (dépendamment de la complexité) réalisables sur une interface GUI furent attribuées à chaque sujet. Une même tâche pouvait être attribuée à plus d'un sujet. Dans un premier temps, les sujets ont exécuté la ou les tâches eux-mêmes sur l'interface. À la suite de cette exécution, ils étaient capables de: i) décrire l'exécution de la tâche en terme d'actions élémentaires (niveau détaillé), ii) d'interpréter les changements internes du système en réponse à chaque action et iii) de décrire la rétroaction de l'interface en réponse à chaque action.

Deuxièmement, ils ont décrit la ou les tâches de trois façons différentes : i) Langage naturel. 2) KLM. 3) UAN. Ensuite, ils ont effectué la même chose avec le langage AIR.

Ils ont du utiliser l'outil de vérification syntaxique (chapitre 3) pour valider leur(s) script(s) AIR.

Il faut noter que les étudiants ont suivi un cours de trois heures sur le KLM et un cours de trois heures sur l'UAN. Cependant, ils n'ont eu qu'une introduction en classe d'environ trois heures sur le langage AIR. Ils ont du par la suite apprendre davantage l'AIR par eux-mêmes à l'aide d'un tutorial en ligne. L'AIR était donc défavorisé par rapport aux deux autres. De plus, l'enquête ne tient pas compte des erreurs d'écriture potentielles commises par les sujets pour le KLM et l'UAN. Les résultats obtenus (présentés plus loin) sont donc purement indicatifs.

Après avoir terminé les spécifications, chaque étudiant a rempli un questionnaire. La participation à cette partie du travail se faisait sur une base volontaire et n'avait aucune incidence sur la note du devoir. Dans le questionnaire, on leur demandait d'évaluer le KLM, l'UAN et l'AIR selon les quatre critères suivants :

1. Temps d'écriture de la tâche.
2. Facilité d'apprentissage du langage (échelle de 0 à 6, 0 : très difficile; 6 : très facile).
3. Facilité d'écriture (échelle de 0 à 6).
4. Facilité de lecture (échelle de 0 à 6).

Finalement, ils ont spécifié le langage qu'ils préféraient entre l'UAN et l'AIR. Tous les sujets (dont les données sont discutées dans ce chapitre) ont donné leur consentement écrit en remplissant un formulaire afin que leurs données puissent être publiées dans ce mémoire. Les patrons du questionnaire d'enquête et du formulaire de consentement sont présentés à l'annexe 10.

Récolte des données

Tous les sujets consentants ont envoyé leur questionnaire et leur formulaire par courriel sur le serveur du cours. Le questionnaire a alors été récupéré à des fins d'analyse.

Analyse pour la comparaison du KLM, de l'UAN et de l'AIR

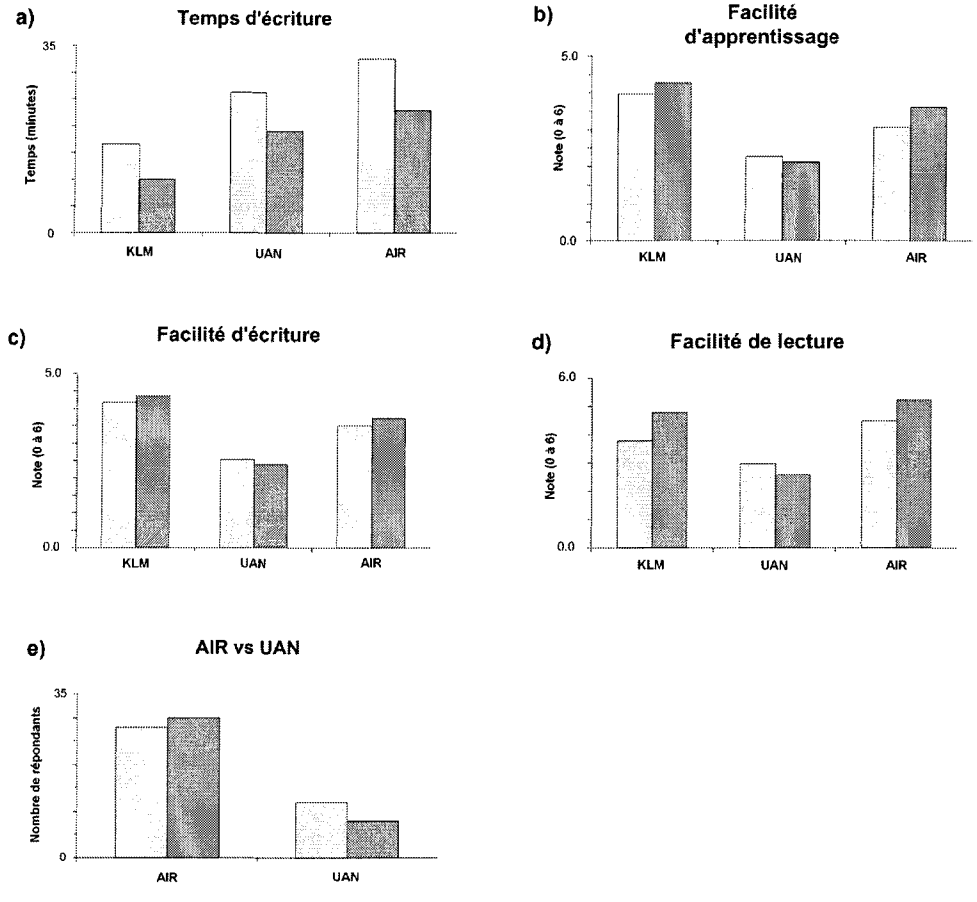
Premièrement, le temps moyen d'écriture pour le KLM, l'UAN et l'AIR a été calculé pour les deux groupes séparément. Pour la facilité d'apprentissage, d'écriture et de lecture, la note moyenne (échelle de 0 à 6) de chacun des langages fût calculée pour chaque groupe séparément. Finalement, le pourcentage de sujets préférant l'UAN à l'AIR et vice et versa a été calculé pour chaque groupe séparément.

2. Résultats

De façon générale, trois fois plus de répondants ont préféré le langage AIR à la notation UAN. L'AIR a été mieux apprécié que l'UAN en ce qui a trait aux facilités d'apprentissage, d'écriture et de lecture. Il a même été mieux apprécié que le KLM pour la facilité de lecture.

Plus en détail, les temps moyens d'écriture (en minutes) en ordre croissant furent ceux du KLM, de l'UAN et de l'AIR. Pour ce qui est de la facilité d'apprentissage, la note moyenne obtenue pour le KLM fût d'environ 4 (échelle de 0 à 6, 0 : très difficile; 6 : très facile) alors que celles pour l'UAN et l'AIR furent d'environ 2 et 3, respectivement. Des résultats similaires ont été obtenus pour la facilité d'écriture. C'est l'AIR qui a été noté comme le plus facile à lire tandis que l'UAN a obtenu la plus faible note. Finalement, les répondants ont préféré en moyenne le langage AIR à la notation UAN selon un rapport d'environ 3 répondants pour 1. Il est à noter que le KLM (tel qu'enseigné dans le cours

LOG340), contrairement à l'UAN et l'AIR, comporte seulement 4 opérateurs et ne spécifie que les actions de l'usager. Tous les résultats sont illustrés à la figure A9-1.



Gris pâle : groupe 1 (40 sujets)	Gris foncé: groupe 2 (38 sujets)

Figure A9-1 Comparaison du KLM, de l'UAN et de l'AIR

3. Limitations

L'enquête présentée ici semble posséder quelques lacunes au niveau de la méthodologie. Premièrement, tous les répondants étaient des étudiants du cours LOG340 enseigné par le professeur supervisant ce travail de recherche. Le langage AIR est enseigné dans ce cours. De plus, les répondants étaient pour la plupart familiers avec les langages de

programmation fréquemment utilisés. Cependant, ce n'est peut-être pas le cas pour tous les concepteurs d'interfaces usager. Ainsi, il se peut que les résultats aient été biaisés par cet échantillon précis. Il serait intéressant de poursuivre cette étude en agrandissant l'échantillon et surtout en le diversifiant davantage.

ANNEXE 10

Enquête comparative : AIR vs KLM et UAN – matériel pour sujets

Cette annexe présente la documentation remise à chaque sujet lors de l'étude comparative de l'AIR avec le KLM et l'UAN (chapitre 4).

1. Formulaire de consentement remis au sujet

Ce projet a pour but de développer un système de représentation du déroulement des tâches réalisées avec une interface usager, appelé AIR («Activity-oriented Interface Representation»).

L'AIR sera utilisé dans les cours d'interfaces de l'ÉTS, et éventuellement, il sera mis à la disposition de la communauté au moyen de publications scientifiques et d'outils logiciels gratuits.

- Le devoir que vous réalisez peut aider le développement de l'AIR de deux façons :

  1. Assurance qualité. Si vous êtes d'accord, votre questionnaire et les autres résultats de votre travail peuvent servir à une évaluation de l'usabilité de l'AIR et des outils d'aide.

     - Ces données seront traitées de façon anonyme (il sera impossible de vous identifier).

     - Les données qui permettraient de vous identifier seront conservées sur des supports informatiques qui ne permettent ni la consultation ni la copie (CDs ou DVDs).

     - Seuls les participant au projet AIR auront accès à ces données et ils ne les divulgueront sous aucun motif.

2. Construction d'une librairie de scripts d'usage public. Vos scripts peuvent être éventuellement intégrés dans cette librairie. Dans ce cas, vous serez reconnu comme l'auteur des scripts, et votre nom sera inscrit dans la librairie. Cependant, vous renoncez à vos droits intellectuels (« *copyright* ») sur ces scripts.

- Quelle que soit votre décision, vous n'aurez aucun risque ou inconvénient de quelque nature que ce soit. En particulier il n'y aura aucune incidence sur l'évaluation de votre devoir.

- Si vous acceptez, vous n'en retirerez aucun avantage particulier, si ce n'est d'avoir contribué à la mise au point d'un outil qui pourrait être utile aux autres étudiants, à l'ÉTS, et à la communauté qui développe des interfaces.

- Ce projet est réalisé sous la responsabilité du professeur Éric Fimbel de l'ÉTS (efimbel@ele.etsmtl.ca; (514) 396-8670 ; bureau 2542). Pour toute information, vous pourrez contacter le responsable.

Après avoir reçu et compris les explications relatives à ma participation au projet, et suite à la lecture du présent formulaire, je soussigné(e)

_____ consens :

- à ce que le résultat de mon travail soit utilisé pour évaluer l'usabilité de l'AIR _____

- à ce que mes scripts AIR soient inclus, le cas échéant dans la librairie de scripts d'usage public _____

2. Questionnaire rempli par le sujet

Tableau A10-I

Questionnaire rempli par le sujet dans le cadre de l'enquête comparative entre le langage AIR, le modèle prédictif KLM et la notation UAN

1	Date	date en format AAAA-MM-JJ	
2	Tâche	donner pour chaque tâche le numéro et la lettre	
3	Temps d'installation	temps mis pour installer les outils (arrondi en minutes)	
4	Temps d'apprentissage	temps passé sur le tutorial AIR avant de commencer (arrondi en minutes)	
5	Temps d'observation	temps mis pour observer la tâche (arrondi en minutes)	
6	Temps de description	temps mis pour décrire la tâche en langage naturel (arrondi en minutes)	
7	klmTemps	temps mis pour écrire le KLM (arrondi en minutes)	
8	uanTemps	temps mis pour écrire l'UAN (arrondi en minutes)	
9	airTemps	temps mis pour écrire l'AIR (arrondi en minutes)	
10	klmApprentissage	facilité d'apprentissage du KLM de 0(très dur) à 6 (très facile) (3= régulier)	
11		degré de certitude sur question précédente de 0 (pas certain du tout) à 6 (tout à fait certain)	
12	klmEcriture	facilité d'écriture du KLM de vos tâches de 0(très dur) à 6 (très facile) (3= régulier)	
13		degré de certitude sur question précédente de 0 (pas certain du tout) à 6 (tout à fait certain)	
14	klmLecture	facilité de relecture du KLM de vos tâches de 0(très dur) à 6 (très facile) (3= régulier)	
15		degré de certitude sur question précédente de 0 (pas certain du tout) à 6 (tout à fait certain)	
16	uanApprentissage	facilité d'apprentissage de l'UAN de 0(très dur) à 6 (très facile) (3= régulier)	
17		degré de certitude sur question précédente de 0 (pas certain du tout) à 6 (tout à fait certain)	
18	uanEcriture	facilité d'écriture de l'UAN de vos tâches de 0(très dur) à 6 (très facile) (3= régulier)	
19		degré de certitude sur question précédente de 0 (pas certain du tout) à 6 (tout à fait certain)	
20	uanLecture	facilité de relecture de l'UAN de vos tâches de 0(très dur) à 6 (très facile) (3= régulier)	
21		degré de certitude sur question précédente de 0 (pas certain du tout) à 6 (tout à fait certain)	
22	airApprentissage	facilité d'apprentissage de l'AIR de 0(très dur) à 6 (très facile) (3= régulier)	
23		degré de certitude sur question précédente de 0 (pas certain du tout) à 6 (tout à fait certain)	

Tableau A10-I (suite)

24	airEcriture	facilité d'écriture de l'AIR de vos tâches de 0(très dur) à 6 (très facile) (3= régulier)	
25		degré de certitude sur question précédente de 0 (pas certain du tout) à 6 (tout à fait certain)	
26	airLecture	facilité de relecture de l'AIR de vos tâches de 0(très dur) à 6 (très facile) (3= régulier)	
27		degré de certitude sur question précédente de 0 (pas certain du tout) à 6 (tout à fait certain)	
28	UanVsAir	pour le travail demandé dans ce devoir, avez vous préféré l'UAN ou l'AIR	
29	airPointsNegatifs	points négatifs et problèmes de l'AIR (si possible par ordre d'importance décroissante)	
30	airPointsPositifs	points positifs de l'AIR (si possible par ordre d'importance décroissante)	
31	airAmelioration	améliorations possibles (si possible par ordre de priorité décroissante)	

ANNEXE 11

Mise au point et validation du KPC – matériel pour sujets

Cette annexe présente la documentation remise à chaque sujet lors de l'étude pour la mise au point et la validation du modèle KPC (chapitre 5). On présente également des spécifications sur les interfaces utilisées.

1. Directives générales à l'attention du sujet

- Le présent test se déroulera en entier au Laboratoire d'Environnements de Synthèse et d'Interfaces Avancées (LESIA) du département de génie électrique de l'École de technologie supérieure (ÉTS).

- Votre poste de travail est muni d'une chaise ergonomique de choix pour l'exécution de tests avec sujet humains, d'un bureau, d'un ordinateur de type PC avec son écran ainsi qu'un clavier standard et une souris optique.

- Durant la période de test de 2 heures, vous aurez à exécuter un bloc de six tâches de façon répétitive. Le nombre de répétitions demandé est de 40.

- Chacune de ces six tâches devra être effectuée par le biais d'une interface GUI relativement simple à utiliser.

- Un bloc doit être exécuté manuellement par le biais d'un fichier .bat.

- Pour chaque bloc, les six tâches sont générées dans le même ordre.

- Dans la fenêtre déjà ouverte sur le bureau de travail de votre ordinateur, il y a quatre répertoires contenant chacun dix fichiers .bat représentant chacun un bloc. Un répertoire contenant deux blocs préliminaires est également présent.

- Vous aurez donc à exécuter 40 fichiers .bat. De cette façon, il vous sera possible de prendre des pauses volontaires entre les blocs. Vous pouvez prendre le nombre de pauses que vous désirez jusqu'à un maximum de 39 qui représente le nombre maximal de délais entre les blocs. La durée de chaque pause est également laissée à votre discrétion. Il est interdit de prendre une pause durant l'exécution d'un bloc. Entre chaque bloc, vous pourrez également poser des questions à l'expérimentateur.

- Pour chacune des tâches à effectuer, un dossier contenant la consigne vous sera remis. Vous devrez sans exception respecter l'ordre d'exécution de la tâche ainsi que les détails d'exécution. Il est également important d'exécuter les tâches le plus rapidement possible.

- Il vous est demandé de lire chaque consigne et de la remettre à l'intérieur de son dossier respectif avant son exécution et ce même à l'intérieur d'un bloc.

- Au besoin, il vous sera possible de consulter à nouveau la consigne lors de l'exécution de la tâche.

Le test se déroulera de la façon suivante :

- L'expérimentateur vous remettra et lira avec vous le formulaire de consentement à la participation à cette étude. Vous devrez alors remplir ce formulaire. Il vous sera possible de poser des questions.

- Ensuite, l'expérimentateur vous remettra et regardera avec vous un formulaire de renseignements personnels que vous devrez remplir. Vous pourrez poser des questions.

- Ensuite, on vous remettra les dossiers contenant les consignes des six tâches à effectuer. L'expérimentateur les survolera avec vous. Vous pourrez poser des questions.

- Une fois tout cela fait, vous pourrez bénéficier d'une période de pratique qui sera constitué de l'exécution de deux blocs maximum. Ceci est fait dans le but d'éliminer des erreurs dues au manque de pratique.

- Le bureau de travail (« desktop ») de l'ordinateur présent à votre poste de travail possède un dossier nommé *lesiaApplicationsAir2003UpmTesting*. Ce dossier sera déjà ouvert par l'expérimentateur à votre arrivée. Ce dossier comprend quarante fichiers .bat nommés *airUpmTestBlockX.bat* séparés en quatre sous-répertoires. Pour exécuter un bloc, il vous faut cliquer deux fois sur un fichier du type *airUpmTestBlockX.bat*.

- Les blocs doivent être exécutés dans l'ordre, c'est-à-dire du premier au quarantième. Vous pouvez prendre une pause entre chaque bloc d'une durée maximale de trente secondes. Si toutefois vous voulez répartir de plus grandes pauses, vous ne devez pas dépasser une durée maximale de trente minutes pour la durée totale de la séance de test de deux heures qui comprend les pauses.

- Avant de commencer, relisez attentivement les consignes associées aux six tâches sans toutefois les mémoriser.

- Pour chacune des tâches, relisez la consigne avant d'effectuer la tâche afin de ne pas faire des pauses inutiles lors de l'exécution et remettez-la dans son dossier. Si toutefois des détails de la tâche ne vous reviennent pas à l'esprit en cours d'exécution, vous pouvez consulter la consigne.

- Lors de l'exécution d'une tâche, il est primordial que vous fassiez tout ce qui est spécifié dans la consigne et de la manière demandée et ce, sans aucune exception. Ceci constitue un critère majeur dans la validité d'éventuelles analyses de données entre sujets et/ou entre tâches.

- Les consignes ont été mises au point à la suite de pré-tests. Si toutefois, vous ressentez le besoin d'obtenir de l'information additionnelle auprès de l'expérimentateur, vous pourrez le faire entre deux blocs seulement.

- N'oubliez pas que vous bénéficiez d'une période de pratique d'au plus deux blocs d'exécution avant d'entamer les blocs qui feront l'objet de l'analyse de données. Ils sont situés dans le dossier nommé *preliminaryBlocs* de la fenêtre déjà ouverte par l'expérimentateur.

- À la suite de cette période de pratique, vous pourrez en profiter pour poser les questions qui vous viennent à l'esprit.

- Suite à la séance de test, l'expérimentateur vous paiera votre compensation financière et vous devrez signer un reçu de preuve.

Rappel : si, pour une raison quelconque, vous ne vous sentez pas capable de poursuivre le déroulement du test jusqu'à son dénouement, il vous sera possible de vous retirer sans préjudice de la part de l'expérimentateur. De plus, votre compensation financière vous sera tout de même remise.

2. Formulaire de consentement remis au sujet

Mise au point de l'estimateur de temps du système de représentation des interactions usager-machine AIR et étude exploratoire de l'évolution sous l'effet de l'entraînement des séquences d'opérations et des temps d'exécution dans la réalisation de tâches par ordinateur

Formulaire d'information et de consentement

École de technologie supérieure

Pierre-Samuel Dubé
Étudiant à la maîtrise en génie électrique, École de technologie supérieure.

Eric Fimbel Ph.D. Ing. Jr.
Professeur, département de génie électrique, École de technologie supérieure
Chercheur, Institut universitaire de gériatrie de Montréal

- Ce projet vise à développer un langage de spécification des interactions usager-machine appelé AIR (*« Activity-oriented Interface Representation »*).

- Le but de la présente recherche est de déterminer comment les pauses mentales faites par un usager durant l'exécution d'une tâche évoluent avec la pratique.

- La séance de test à laquelle vous participez a lieu au Laboratoire d'environnements de synthèse et d'interfaces avancées (LESIA) de l'École de technologie supérieure (ÉTS).

Après avoir signé le présent formulaire et remplit une fiche de renseignements personnels, vous devrez réaliser six tâches différentes que vous répéterez pour la durée du test. L'ordre d'exécution des tâches est prédéterminé et reste le même pour toutes les répétitions.

Il s'agit de tâches simples exécutées par le biais d'interfaces GUI. Elles sont constituées de quelques actions élémentaires, soit des clics de souris, des saisies de texte, des choix d'un item parmi une liste, etc.

La séance de test est composée de 40 blocs de 6 tâches.

Vous pourrez vous reposer le temps que vous jugerez bon entre deux blocs. Il suffit que vous en avisiez l'expérimentateur. Un arrêt au milieu d'un bloc ou d'une tâche ne sera pas permis.

La durée du test est d'environ 2 heures, incluant les pauses que vous prendrez.

- Il n'y a pas de risque particulier lié à votre participation, sinon celui lié à l'usage d'un ordinateur durant environ deux heures.

- Il n'y a pas d'inconvénient particulier lié à votre participation, excepté le temps passé, la fatigue, le stress et la frustration que vous pourriez ressentir. Si ces inconvénients deviennent excessifs, vous êtes libre d'arrêter à tout moment (voir plus loin).

- Il n'y a pas d'avantage particulier lié à votre participation, excepté une petite compensation financière (voir plus loin), mais vous participerez à l'avancement des connaissances dans le domaine de la conception et l'évaluation d'interfaces usager-machine.

- Vous recevrez une compensation financière de 16 $ par session de test pour couvrir le temps passé et vos éventuels frais de transport.

- Vous pourrez vous retirer en tout temps de ce projet, et ce sans avoir à fournir de justification et sans préjudice. Dans ce cas, les données vous concernant seront détruites. La compensation financière prévue vous sera versée.

- Le projet pourra également être arrêté en tout temps par les chercheurs. Dans ce cas, les chercheurs vous informeront de la raison de cet arrêt. Votre participation future sera automatiquement annulée. Si vous êtes déjà en train de participer, la compensation financière vous sera versée.

- Votre participation à ce projet est strictement confidentielle. Ce formulaire de consentement et la fiche de renseignements sont conservés sous clé et seront détruits au plus tard cinq ans après la fin du projet. Les données et résultats enregistrés sur ordinateur ne permettront pas de vous identifier. Seuls les chercheurs et le personnel de recherche auront accès à ces données.

- On vous fournira toutes les informations que vous désirerez au sujet du présent projet, excepté celles qui pourraient permettre d'identifier les autres participants (voir plus loin les coordonnées des chercheurs).

Ce projet est réalisé sous la responsabilité du professeur Éric Fimbel de l'ÉTS et de Pierre-Samuel Dubé, étudiant à la maîtrise en génie électrique à l'ÉTS. Pour toute information, vous pourrez contacter l'un ou l'autre des responsables. En cas de questions, ou pour toute information concernant votre participation à l'étude, vous pouvez contacter une personne neutre, soit le président du Comité d'éthique de la recherche de l'École de technologie supérieure au (514) 396-8829.

Après avoir reçu et compris les explications relatives à ma participation au projet de recherche, et suite à la lecture du présent formulaire, je soussigné(e) _____ _____ consens librement à y participer.

_____ _____

Signature Date

Le chercheur

_____ _____

Signature Date

3. Fiche de renseignements personnels remplie par le sujet

Tableau A11-I

Fiche de renseignements personnels remplie par le sujet

Code sujet	
Type de test	
Date	
Prénom(s)	
Nom(s)	
Tel.	
Courriel	
Sexe	femme homme
Âge	
Latéralité	gaucher ambidextre droitier
Langue maternelle (ou pays d'origine)	
Activité (étudiant, employé, etc.)	
Études complétées	primaire secondaire cégep baccalauréat études supérieures
Fréquence d'usage hebdomadaire de votre ordinateur au cours de la dernière année	< 1 1-5 5-10 >10

Tableau A11-I (suite)

Familiarité avec les ordinateurs de type PC	Peu expert — Très expert
Familiarité avec les ordinateurs de type Macintosh	Peu expert — Très expert
Familiarité avec Internet	Peu expert — Très expert
Familiarité avec le système d'exploitation Linux	Peu expert — Très expert
Familiarité avec le système d'exploitation Windows	Peu expert — Très expert
Familiarité avec les jeux vidéo	Peu expert — Très expert

Tableau A11-I (suite)

À votre connaissance, êtes vous dans l'une des situations suivantes :	
Déficit de vision non corrigé	
Déficit d'audition non corrigé	
Problème(s) moteur(s)	
Épilepsie, troubles neurologiques ou dépressifs	
Autres problèmes de santé que vous voudriez mentionner	
Médications pouvant affecter votre niveau de vigilance ou votre fonctionnement cognitif ou moteur	

4. Points importants sur le déroulement des tâches

Pour chaque interface, le premier écran n'avait qu'un seul bouton. Les activités effectuées par le sujet avant de cliquer sur ce bouton n'ont pas été considérées dans l'extraction des séquences d'opérations. Ceci nous a donc permis de forcer la nature et surtout la position de la première opération effectuée.

Les interfaces et les consignes étaient dirigistes. Autrement dit, les sujets ne pouvaient sortir d'un écran sans avoir effectué leur tâche avec succès. Dans le cas d'une erreur, un message était généré mais la nature et l'emplacement de l'erreur n'étaient pas spécifiés.

5. Interface de test 1

Consigne remise au sujet

Les activités suivantes doivent être effectuées dans l'ordre et le plus rapidement possible:

1) Lorsque vous êtes prêt à commencer, appuyez sur le bouton START.
2) Choisir le type de paiement à crédit.

3) Positionner le pointeur sur le bouton NEXT et attendre la directive présente dans l'info-bulle avant d'aller plus loin.

4) Appuyer sur le bouton NEXT.

5) Dans le nouvel écran, remplir les champs comme suit :

   - Subject ID (0 :9) : Entrer l'entier indiqué dans l'info-bulle de la page précédente
   - *First Name*: Olivier
   - *Last Name*: Rochon
   - *Province*: Ontario
   - *Postal code*: G2P 1H1
   - *Email address*: olivier.rochon@hotmail.com
   - *Credit card type*: Visa
   - *Credit card number*: 123456789
   - *Expiration date*: 2005/01

6) Appuyer sur le bouton FINISH lorsque vous avez terminé.

Diagramme de transition de l'interface

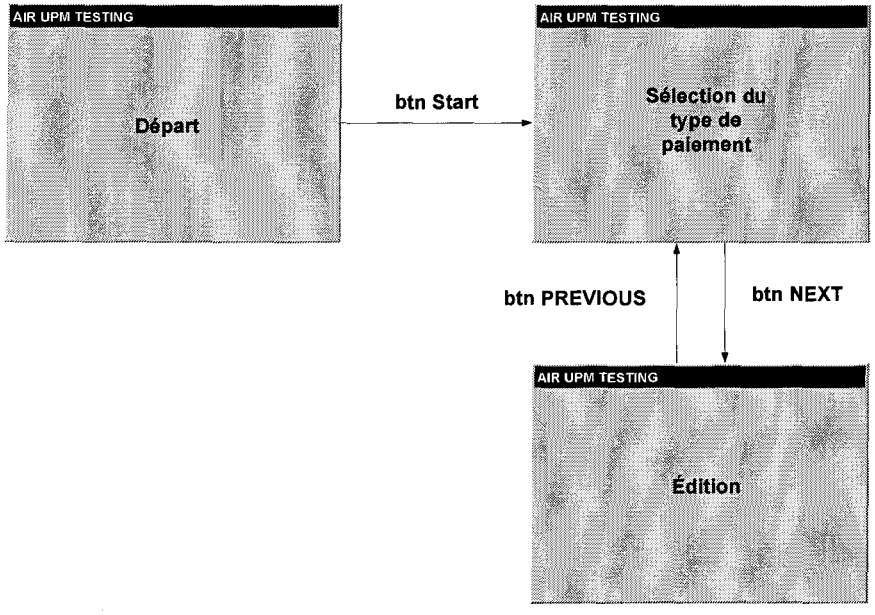
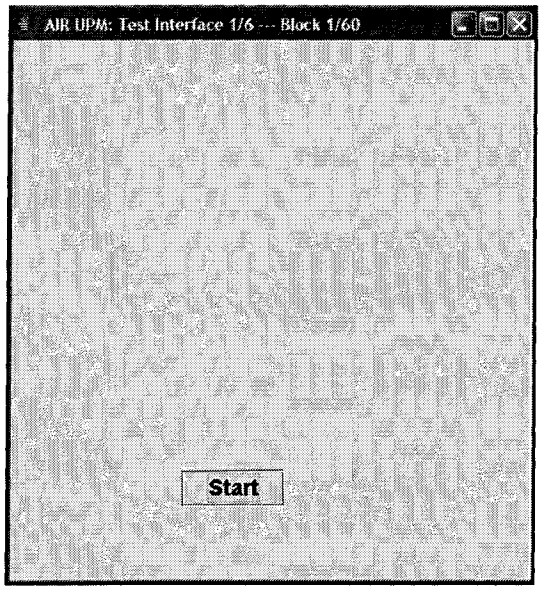


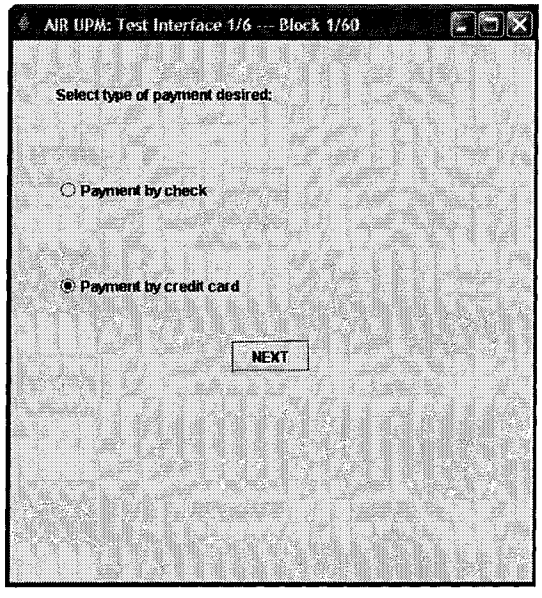
Figure A11-1 Diagramme de transition de l'interface de test 1

Écrans de l'interface

a) écran de départ



b) écran d'option



c) formulaire à remplir

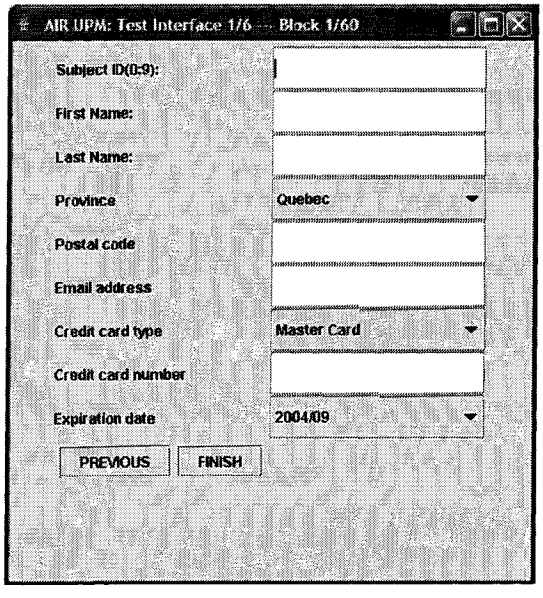


Figure A11-2 Écrans de l'interface de test 1

6. Interface de test 2

Consigne remise au sujet

Les activités suivantes doivent être effectuées dans l'ordre et le plus rapidement possible:

1) Lorsque vous êtes prêt à commencer, appuyez sur le bouton START.

2) Sélectionner tout le contenu de la zone de texte de gauche.

3) Positionner le pointeur sur le bouton « >> » situé au milieu entre les deux zones de texte et attendre l'info-bulle avant d'aller à l'étape 3.

4) Appuyer sur le bouton « >> » afin de copier le contenu dans la zone de texte de droite.

5) Ceci fait, remplir les champs dans la zone de texte de droite comme suit :

   - *Name*:Sylvie Dubé
   - *Sexe(M/F)*:F
   - *Date of birth (year/month/day)*:1980/01/01
   - *Nationality*:Canadian
   - *Occupation (S(Student), P(Professionnal), U(Unemployed))*:P

Note: Il est important de ne laisser aucun espace au début d'un champ et à la fin du texte entré.

6) Appuyer sur le bouton FINISH lorsque vous avez terminé.

Diagramme de transition de l'interface

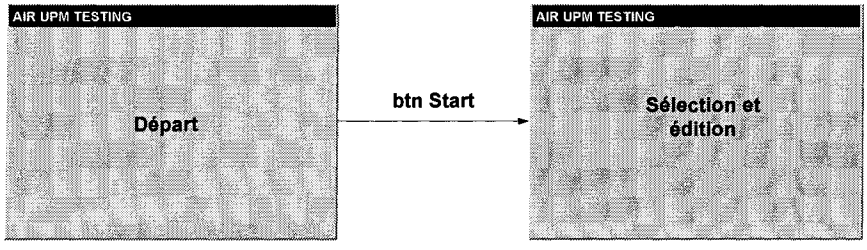


Figure A11-3 Diagramme de transition de l'interface de test 2

Écran de l'interface

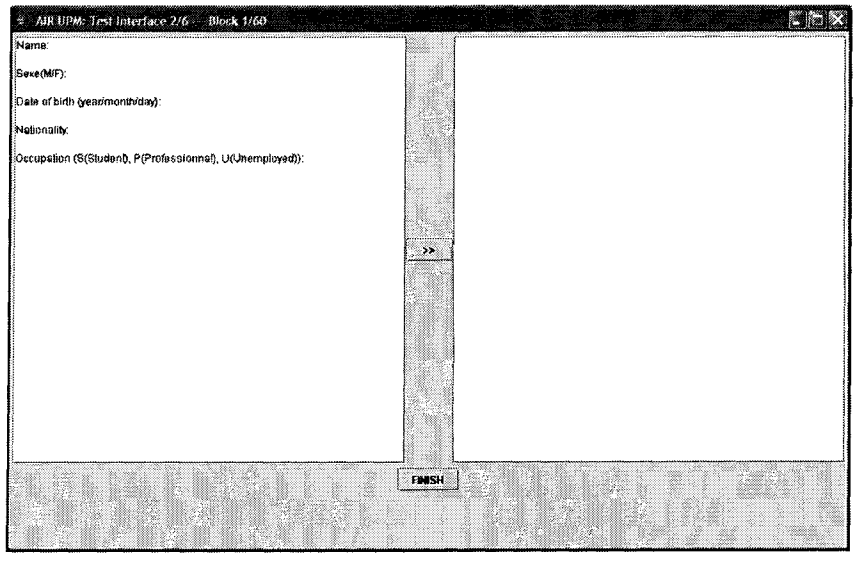


Figure A11-4 Écran de l'interface de test 2

7. Interface de test 3

Consigne remise au sujet

Les activités suivantes doivent être effectuées dans l'ordre et le plus rapidement possible:

1) Lorsque vous êtes prêts à commencer, appuyez sur le bouton START.

2) Sélectionner l'option d'installation ADVANCED et continuer.

3) Positionner le pointeur sur le bouton « ... » à la droite du champ de texte et attendre l'info-bulle avant d'aller à l'étape 4.

4) Appuyer sur le bouton « ... ».

5) Aller dans le dossier MY MUSIC en effectuant un clic double sur l'icône de ce dossier présent dans la fenêtre de sélection.

6) Cliquer sur l'icône de remontée (UP ONE LEVEL) afin de remonter d'un niveau dans la structure des répertoires. Cette icône est celle située la plus à gauche dans la barre d'icônes située en haut de la fenêtre de sélection et à la droite du champ de texte.

7) Aller dans le dossier PROGRAM FILES en effectuant un clic double sur l'icône de ce dossier présente dans la fenêtre de sélection.

8) Spécifier dans le champ FILE NAME le nom suivant : *myApplication.exe.*

9) Appuyer sur le bouton OPEN.

10) Appuyer sur le bouton NEXT.

11) Sélectionner les deux premières boîtes à cocher.

12) Appuyer sur le bouton FINISH lorsque vous avez terminé.

Diagramme de transition de l'interface

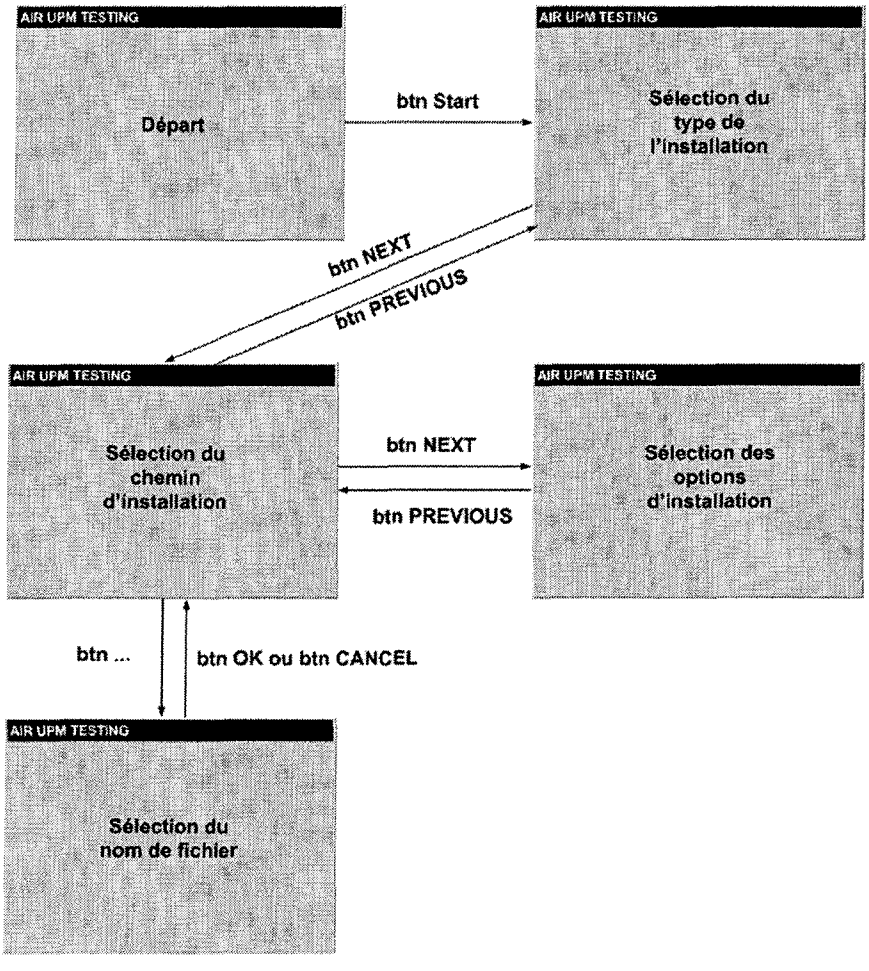
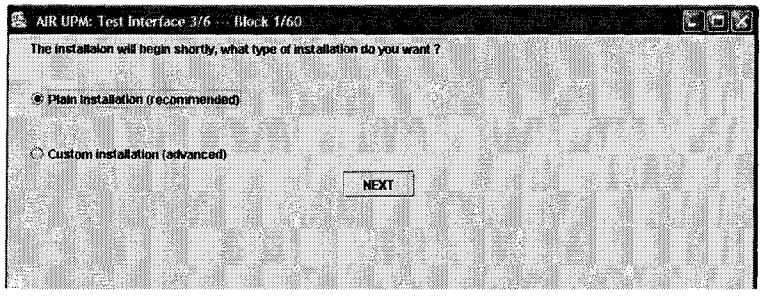


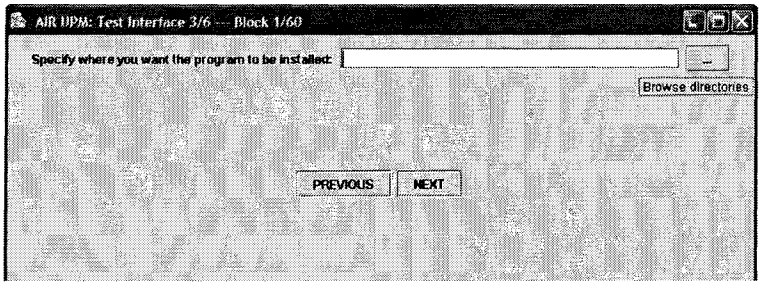
Figure A11-5 Diagramme de transition de l'interface de test 3

Écrans de l'interface

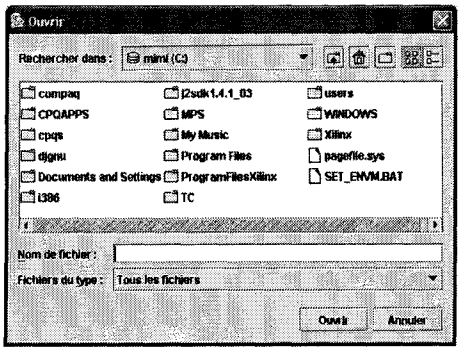
a) sélection du type d'installation



b) sélection du chemin d'installation



c) sélection du nom de fichier



d) sélection des options d'installation

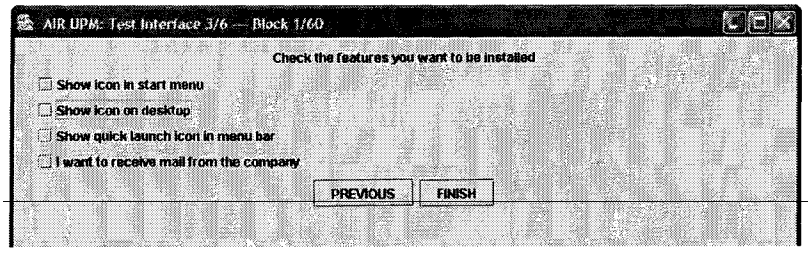


Figure A11-6 Écrans de l'interface de test 3

8. Interface de test 4

Consigne remise au sujet

Consigne sur papier :

Les actions demandées doivent être effectuées dans l'ordre et le plus rapidement possible:

1) Lorsque vous êtes prêt à commencer, appuyez sur le bouton START.

2) Sauvegarder le document présentement ouvert en utilisant le sous-menu SAVE AS du menu FILE.

3) Dans la fenêtre de sauvegarde, spécifier le chemin à un seul niveau suivant : *myDocuments*.

4) Dans la même fenêtre, spécifier le nom de document suivant : *myTestDocument.txt*.

5) Appuyer sur le bouton OK et confirmer l'action lorsque la fenêtre de dialogue apparaît.

6) Imprimer le document en utilisant le sous-menu PRINT DOCUMENT du menu FILE.

7) Dans la fenêtre du choix de l'imprimante, sélectionner l'imprimante par défaut et appuyer sur le bouton OK.

8) Confirmer l'action lorsque la fenêtre de dialogue apparaît.

9) Quitter l'application en utilisant le sous-menu EXIT du menu FILE.

Consigne à l'écran :

Les actions demandées doivent être effectuées dans l'ordre et le plus rapidement possible:

1) Consulter le sous-menu WHAT TO DO du menu HELP de l'écran principal.

2) Suivre les directives indiquées.

Diagramme de transition de l'interface

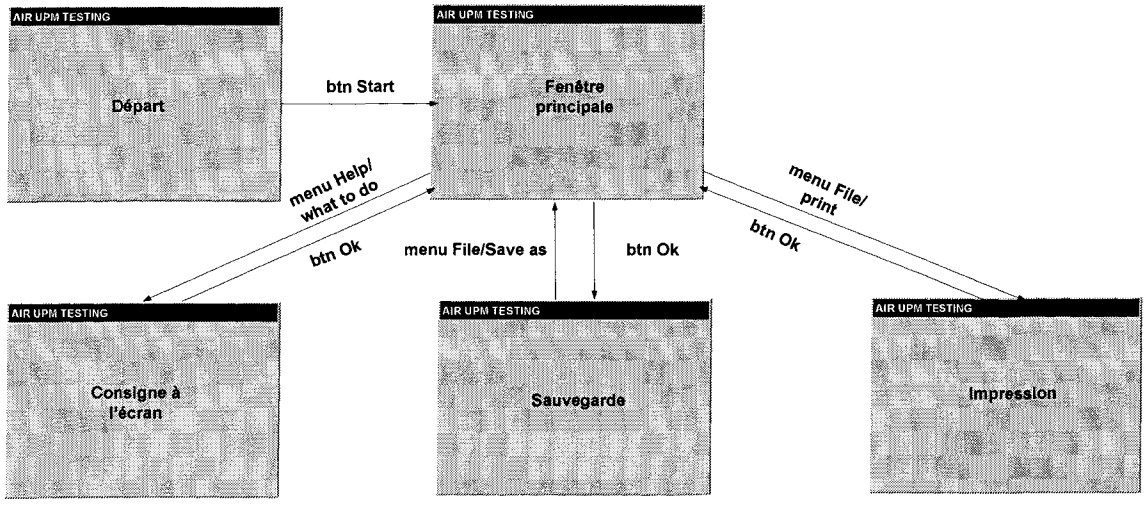
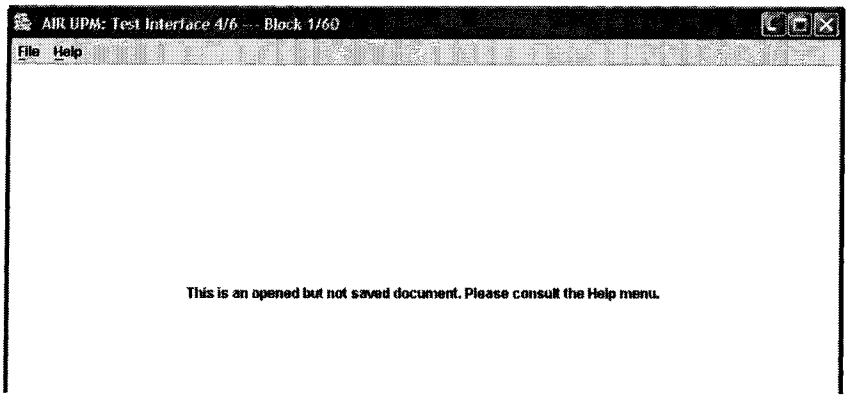


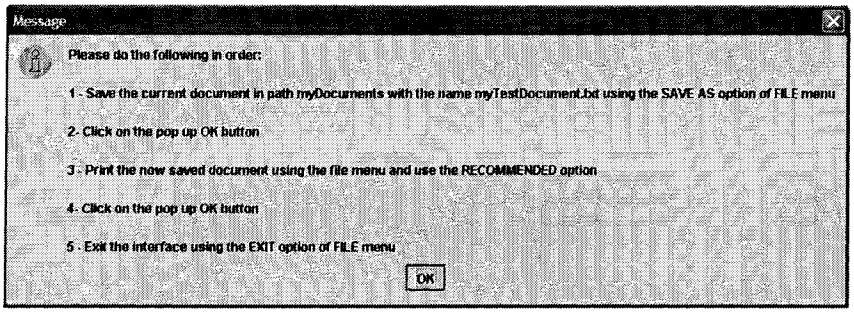
Figure A11-7 Diagramme de transition de l'interface de test 4

Écrans de l'interface

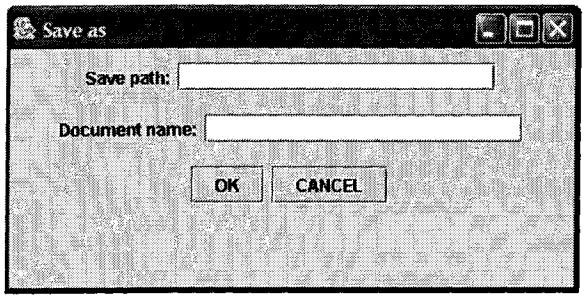
a) fenêtre principale



b) consigne à l'écran



c) écran de sauvegarde



d) écran pour les options d'impression

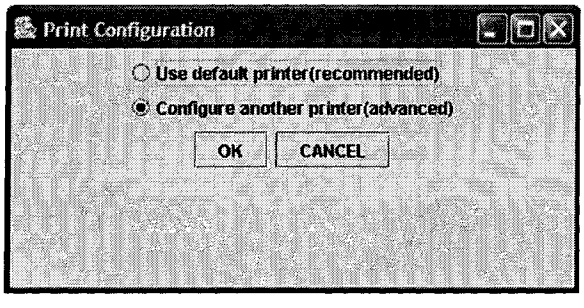


Figure A11-8 Écrans de l'interface de test 4

9. Interface de test 5

Consigne remise au sujet

Les activités suivantes doivent être effectuées dans l'ordre et le plus rapidement possible:

1) Lorsque vous êtes prêt à commencer, appuyez sur le bouton START.

2) Positionner le pointeur dans la région délimitée par un rectangle noir et arborant l'étiquette *Here.*

3) Entrer le nom d'usager *super_mike* et le mot de passe *Gru76e59* et continuer.

   Attention: Le caractère entre *super* et *mike* est un caractère de soulignement (signe _) et non un trait d'union (signe -).

4) Sélectionner l'option de recherche *Publication type.*

5) Entrer comme mot-clé le mot *Ergonomics.*

6) Positionner le pointeur sur le bouton P et attendre l'info-bulle avant d'aller à l'étape 7.

7) Positionner le pointeur sur le bouton F et attendre l'info-bulle avant d'aller à l'étape 8.

8) Appuyer sur le bouton F.

Diagramme de transition de l'interface

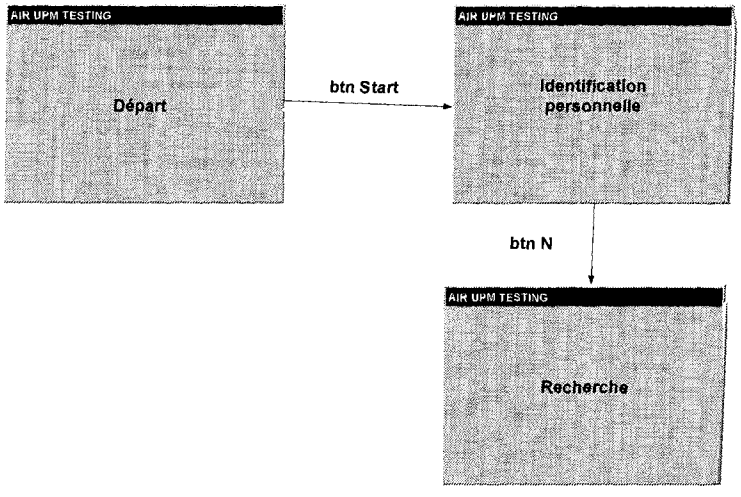
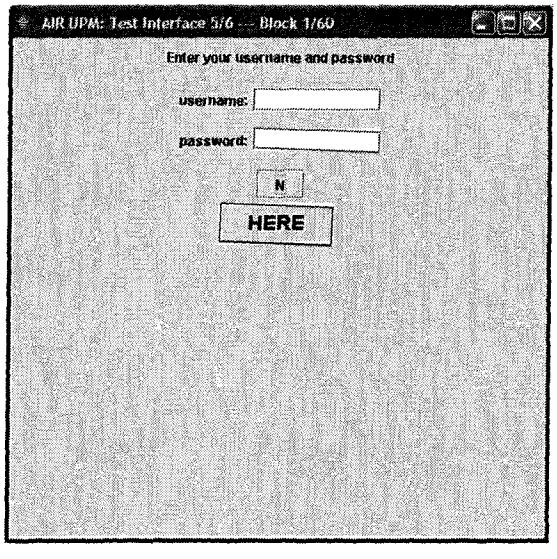


Figure A11-9 Diagramme de transition de l'interface de test 5

Écrans de l'interface

a) écran d'identification



b) écran de recherche

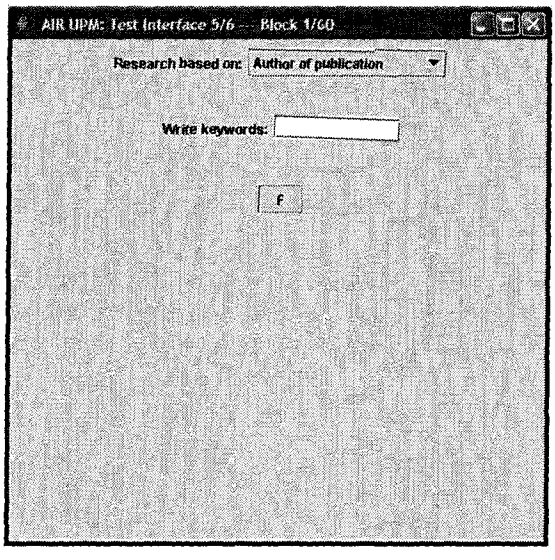


Figure A11-10 Écrans de l'interface de test 5

10. Interface de test 6

Consigne remise au sujet

Les activités suivantes doivent être effectuées dans l'ordre et le plus rapidement possible:

1) Lorsque vous êtes prêt à commencer, appuyez sur le bouton START.

2) Entrer le nom *Jean-Michel Denis* dans le champ de texte et appuyer sur ENTER.

   Attention: Le caractère entre *Jean* et *Michel* est un trait d'union (signe -), pas un caractère de soulignement (signe _).

3) Sélectionner *Pierre* dans la liste.

4) Appuyer sur le bouton CANCEL.

5) Appuyer sur le bouton CANCEL à nouveau.

6) Entrer le nom *Carl Breton* dans le champ de texte et appuyer sur ENTER.

7) Sélectionner *Marc* dans la liste.

8) Appuyer sur le bouton FINISH lorsque vous avez terminé.

Diagramme de transition de l'interface

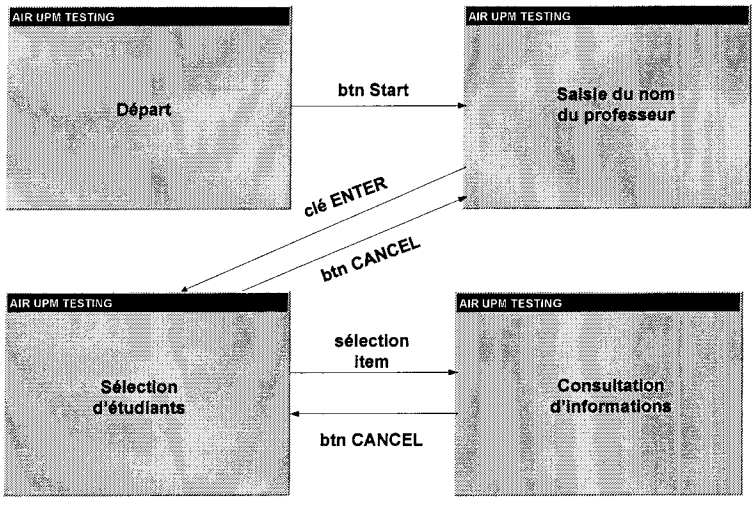
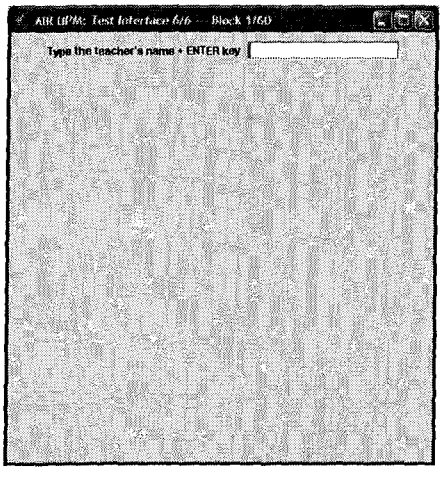


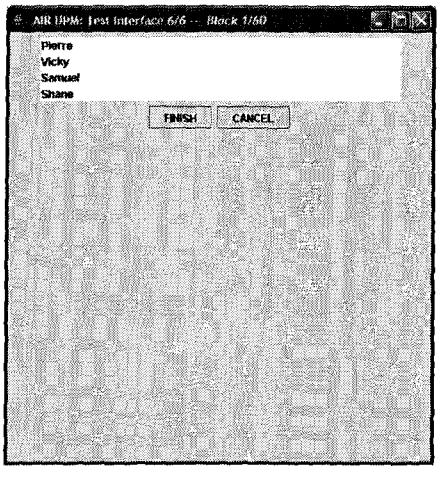
Figure A11-11 Diagramme de transition de l'interface de test 6

Écrans de l'interface

a) écran de saisie du nom du professeur



b) écran de sélection d'étudiants



c) écran de consultation d'informations

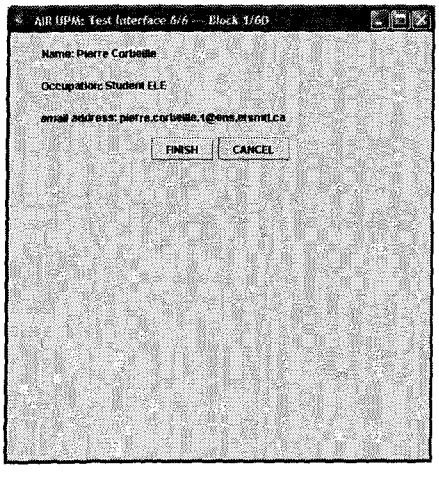


Figure A11-12 Écrans de l'interface de test 6

ANNEXE 12

"Predicting performance of skilled computer users by means of overt operations"

Astract

The KPC is a simplified Keystroke Level Model (KLM) aimed at predicting performance of *skilled users*, familiar with standard graphical user interfaces (GUIs). Skilled users presumably use automatisms, even when facing a novel task (or interface). Because automatisms require low mental and attention resources, it may not be necessary to model *covert* (invisible) *operations* like mental activities. In KPC, a task is modeled by sequences of *overt* (visible) *operations*: key types (K), pointing (P) and clicks (C). These sequences are recorded directly from skilled users by means of an interface prototype. In an experimental study, skilled users repetitively executed simple tasks. The prediction error of KPC was 26% (KLM: 42%) for untrained users, and 19% after extensive practice (KLM: 73%). The prediction error only varied moderately across Tasks and Users. Modeling covert operations (mental activities, hand movements) did not improve significantly the prediction. The study also showed that even after extensive practice, skilled users employ a variety of sequences for a given task, most of them being non-optimal.

Keywords: User interfaces; performance prediction; KLM; KPC; analytical modeling; overt operations; covert operations; skilled users

1. Introduction

Thirty years after their introduction, personal computers are ubiquitous in industrialized societies. Computers are omnipresent in people's day-to-day environment at work, at school, in public places, at home, and in the media, e.g., movies. Since the early generation of personal workstations and computers (Goldberg, 1988), the massive diffusion of graphical user interfaces (GUI) was accompanied by a convergent evolution of the interface standards. In current interfaces (2005), general commands (e.g., save file, copy-paste, drag-and-drop, search) are quite similar across look-and-feels, applications and types of computers (PC, MAC, SUN). As a consequence, even novice users, unfamiliar with a novel application, are usually *skilled* in the sense that they already know implicitly how to use it, i.e., they already have *automatisms* (e.g., Shiffrin & Schneider, 1977) developed from a cumulative use of similar interfaces. These automatisms take the form of stereotyped sequences of *overt operations* like pointing movements, button clicks and key presses. These sequences are *transferable*, i.e., reusable (either directly or with slight adaptations) across a wide range of applications (Murray & Häubl, 2003).

Automatisms represent the most advanced stage in the development of skills, in which psychomotor abilities become preponderant in relation with cognitive and perceptual abilities (e.g., Ackerman, 1988). Since the early work of Woodworth (1899), the development of automatisms and skills has been extensively studied in Experimental Psychology and Cognitive Science (e.g., Anderson, 1982) as well as in Motor Control (e.g., Welford, 1968). Like any other skill, computer skills follow the law of practice, i.e., performance evolve according to a power law (e.g., Newell & Rosenbloom, 1981). The acquisition and/or retention of computer skills have been studied on specific applications (e.g., Nilsen et al., 1993, Bhavnani et al., 1997), with specific entry devices and interface components (Durham & Emurian, 1998, Wiedenbeck, 1999) or as a function of prior level of expertise (e.g., Doane et al., 1990). However, at the beginning of the 21st century, young computer users have been exposed to graphical interfaces

since their early years. As a consequence, most of the development of their general computer skills escapes from scientific investigation.

The presence of computer skills may cause considerable differences in the learning process of a novel application (or new functionalities of some partially known application). Rather than trying to learn the novel application from scratch, we may reasonably posit that skilled users will tend to reuse their existing automatisms. In other terms, *skilled users naive (i.e., unfamiliar) with a novel application may avoid a significant amount of explicit learning*. Indeed, their automatisms may be slowed down (in terms of execution time) by additional *covert operations*, e.g., hesitations, mental explorations or visual searches. However, as long as the users have some knowledge of the domain of the application, they do not have to learn *what* they do. They just learn *how* to do it. In this case, we hypothesize that inasmuch as the interface is a standard GUI and that the user is skilled, additional covert operations will not cause major overtime and will quickly disappear with practice. This viewpoint is compatible with the fact that well-practiced automatisms elicit little mental workload (Welford, 1978).

From a modeler standpoint, it may thus be simpler to consider covert operations as a source of bias and/or variability in performance rather than modeling them explicitly. The execution time of skilled users, naive or not, could therefore be approximately predicted from the overt operations. However, the *sequence* of operations is a priori unknown. A possible approach is to assume that skilled users are "expert" and that they use "optimal sequences" for executing their tasks (Rationality Principle; Card, Moran & Newell, 1983). Unlike covert operations, the possible overt operations and the optimal sequences are strictly determined by the interface. Therefore, in the ideal case, the sequences executed by skilled users, and the corresponding execution time could theoretically be predicted from the interface specifications.

However, the concept of optimal sequence may be ambiguous, because the actual optimization criteria (parameters) are unknown (Embley & Nagy, 1981). The most straightforward parameters are the execution time and the number of operations. However, it is also possible that the user optimizes biomechanical or subjective

parameters like the mechanical action of the limbs (Lebedev et al., 2001) or the feeling of effort (Rosenbaum & Gregory, 2002). Users may also produce "sufficient" rather than "optimal" sequences, i.e., they keep some parameter(s) within a range of tolerance (e.g., Todorov & Jordan, 2002).

An additional difficulty in determining the optimal sequences for a given task is the size of the search space, i.e., the set of all the possible sequences. Modern GUIs are generally unconstrained, i.e., the same task can be executed in many different ways. Furthermore, applications run in open environments and allow usages that cannot be predicted from interface specifications (e.g., locating pages of a WEB site by means of generic search engines instead of using navigation links, Augustine & Greene, 2002). In summary, it may be difficult to analytically determine the sequences that will be used by skilled users in unconstrained interfaces and open environments. Direct observation and/or experimental approaches thus become valuable alternatives or complements to analytical modeling.

The main objective of this study is to propose a hybrid approach for predicting the execution time of skilled users, naive or trained. To that end, we introduce the *KPC model*, i.e., a simplified version of the Keystroke Level Model (KLM; Card, Newell, Moran, 1980) in which there are no covert operations, like hand movements or mental activities. In the KPC, the activity of the user is modeled as a sequence of overt operations: key types (K), pointing movements (P), and mouse button clicks (C). The average execution times of these operations (including those of intermediary covert operations) were determined experimentally, from skilled users executing repetitively a set of tasks with a standard GUI. These executions times are assumed to be general estimates of those of actual users whatever the application, the task and the degree of familiarity. Indeed, we restrict ourselves to standard GUI and skilled users that have some knowledge of the domain of application, so that they do not need to learn what to do, just how to do it.

The experimental part of our approach is the determination of the sequences and the prediction of execution time for a specific interface. It is done as follows. 1) Record the actual sequences of KPC operations executed by skilled users by means of an interface prototype. 2) Filter these sequences in order to keep only those that are optimal (or nearly optimal) in terms of execution time and/or number of operations. 3) For each sequence, predict the overall execution time from the estimates of the execution times of the K, P, C operations. The use of interface prototypes for determining the sequences of operations simplifies considerably the analytical modeling of performance. However, it is worth emphasizing that in previous approaches (Hudson et al.,1999, John et al., 2004) the sequences are demonstrated by the analyst, whereas in the present one, they are collected directly from users. The KPC model and the experimental approach are presented in Section 2.

The second objective of this study is to verify experimentally the hypotheses that underlie the approach. *Hypothesis 1*: the execution times of the K, P and C operations are only moderately affected by inter-individual variability (across skilled users), task contents and training level. If Hypothesis 1 is verified, the average execution times of K, P, C operations initially established may be used as *general* estimates. *Hypothesis 2*: the execution times of the K, P and C operations are only moderately affected by the presence of possible covert operations. If Hypothesis 2 is verified, the omission of the covert operations in the KPC model is justified. Note that covert operations occur preferentially in specific contexts, for instance hand movements between keyboard operations (K) and mouse operations (P, C). Hypothesis 2 can thus be verified by i) measuring the execution times of K, P, C operations in different contexts, and by ii) determining whether these context-dependent execution times bring additional accuracy for predicting the overall execution time of the sequences. *Hypothesis 3*: skilled users, naive or trained, execute a given task by means of a variety of sequences. If Hypothesis 3 is verified, it is worth capturing the sequences from actual users (instead of determining the optimal sequences analytically and/or demonstrating them). These issues were examined in an experiment (realized with the GUIs that were used to

establish the estimates of the execution times of K, P, C operations, but with a different group of subjects) presented in Section 3 (Methods) and Section 4 (Results). An overall discussion is presented in Section 5.

2. The KPC model

2.1. Task representation and execution time prediction

The KPC model is a simplified version of the KLM in which covert operations like homing (KLM's H operation) and mental activities (KLM's M operation) are not represented. The execution of a task is represented as a *sequence of overt operations at the keystroke level.* Those operations are key type (K), point (P) and click (C). Each sequence (e.g., PCKKKKPC) represents *one way* of executing a prescribed task. It is implicitly assumed that the task is correctly completed. However, sequences are not necessarily optimal. They may contain *deviations*, i.e., useless or redundant operations like clicking in an empty panel or moving the mouse while visually exploring the screen. They may also contain errors and corrections. Note that in most of the GUIs, there may exist infinitely many sequences for completing a given task (e.g., going back and forth an arbitrary number of times).

The KPC model provides an estimate $\hat{E}_{KPC}(S)$ of the execution time of a sequence S based on the numbers of K, P and C operations contained in S, i.e., $N_K(S)$, $N_P(S)$ and $N_C(S)$. To that end, the KPC uses execution time estimates of the three operations ($\hat{E}_K$, $\hat{E}_P$ and $\hat{E}_C$) which have been previously established (see Section 3). Eq. 1 shows how $\hat{E}_{KPC}(S)$ is computed.

$\hat{E}_{KPC}(S) = N_K(S)*\hat{E}_K + N_P(S)*\hat{E}_P + N_C(S)*\hat{E}_C$ with $\hat{E}_K = 330$ ms $\hat{E}_P = 700$ ms $\hat{E}_C = 235$ ms	(Eq. 1)

Table 1 depicts how the model is applied on a task that consists in entering a user name ("user") and a password ("passwd") in different text fields. The operation estimates ($\hat{E}_K$, $\hat{E}_P$ and $\hat{E}_C$) have been rounded off.

Table 1

Example of application of the KPC model. The user types the user name "user", the password "passwd" and clicks a confirmation button.

Sequence	Estimate of execution time (ms)	Description
P	700	Point to user name field
C	235	Click in user name field
5K	5 * 330 = 1650	Type user name "user"
K	330	Go to password field with Tab key
6K	6 * 330 = 1980	Type password "passwd"
P	700	Point to confirmation button
C	235	Click on confirmation button
$\hat{E}_{KPC}(S)$	**5830 ms**	

2.2. Experimental construction of the KPC model

The experimental approach proposed here is to determine experimentally the sequences that are optimal or nearly optimal from actual users. The approach consists in the following steps: 1) Develop a prototype of the interface equipped with a *sniffer*, i.e., a program that records the elementary actions of the user. 2) Record several skilled users while they execute repetitively some tasks with the prototype. 3) Filter the captured sequences in order to keep only those that are optimal or nearly optimal. Optimality may be defined in terms of number of operations and/or execution time (determined separately for each user, because comparing execution times across users may be pointless). It is worth mentioning that the two criteria, length of sequence and execution time, may be somehow independent. For instance, using the keyboard instead of the mouse may be faster, even when the number of key presses is higher than the number of mouse clicks. Fig. 1 depicts the approach, and presents a comparison with the classical use of the KLM model proposed in Card et al. (1980).

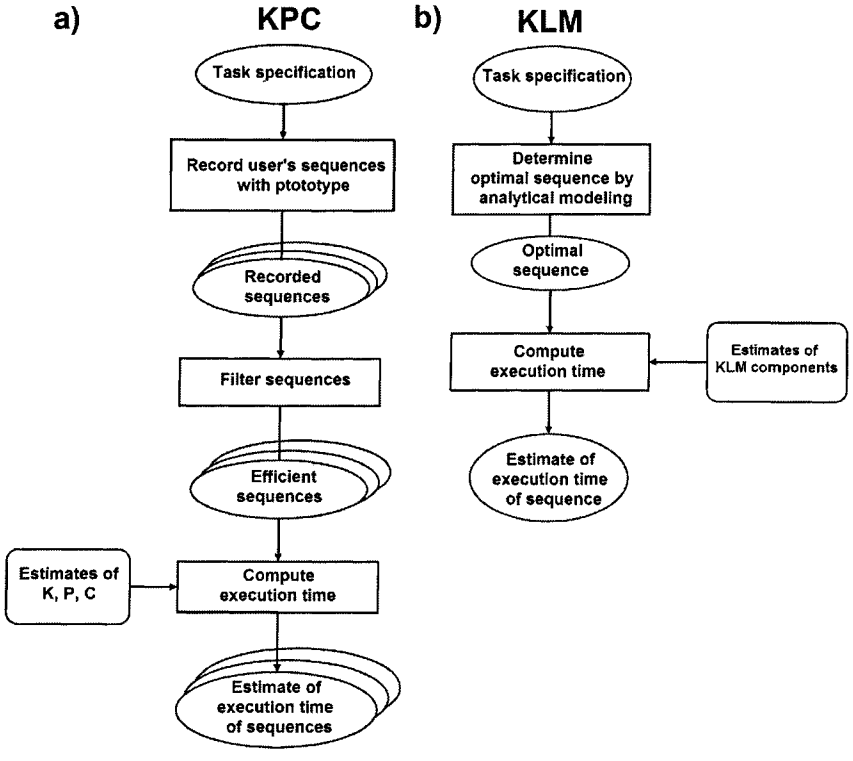


Fig. 1. Model construction process for a) KPC and b) KLM models.

3. Methods

This section presents the experimental protocol that was used for determining the parameters of the KPC model (i.e., determining the execution times of K, P, C operations) and for its validation (i.e., the verification of the three foregoing hypotheses). The results of half of the subjects were used for determining the parameters, whereas the other half was used for the validation.

3.1. Subjects and experiment

Twenty subjects (average age: 25 years (σ= 3.7), all right handed, 4 females, level of studies: CEGEP, i.e., USA high school + 1 year, or more) participated in this study. The subjects filled an auto-administered questionnaire about their experience with computers, and were selected as skilled users according to the following inclusion

criteria: i) Use of a personal computer more than 10 times per week during the past year. ii) Level of familiarity with the *Windows*® operating system self-rated as above average on an analog scale *unfamiliar-expert*. The exclusion criteria were: i) Uncorrected auditory and/or visual deficit. ii) Motor disabilities. iii) History of epilepsy, and/or neurological disorders. Subjects gave informed consent according to the ethics regulations of École de Technologie Supérieure (ÉTS).

Testing took place in a silent room on a desktop computer (Pentium 4, processor speed: 2 GHz, 512 MB RAM, OS: *Windows XP Professional*®, 17'' LCD screen, standard keyboard, 3 buttons optical mouse). Subjects were instructed to perform a sequence of six simple tasks (between 6 and 11 steps), e.g., form filling (Table 2). One practice sequence was allowed. Then, the sequence was repeated 40 times.

Tasks were performed on different graphical interfaces all having the Java Swing look-and-feel. Each task started with the press of a start button and ended with the release of a confirmation button. This allowed precise measurement of the execution time. The values of the fields (e.g., text fields like names or addresses, radio-buttons) were predefined in the instructions or displayed under the form of messages (tool-tips) during the task. Note that the values to be entered in text fields always had the same length across the trials. In case of error on a given screen, a generic message was displayed when the subject tried to go to the next screen. The nature and location of errors were not specified in the message therefore the subject had to localize the error in the current screen. By doing so, errors caused delays that were easy to detect.

Paper instructions were placed on the desk in closed folders, so that consultation was easy to detect (by the observer). During the training phase, subjects were encouraged to read carefully the instructions before each task. During the test, instructions could be consulted at any time. It was observed that instructions were consulted before the tasks only during the first 10 repetitions (some subjects only consulted 3 times). After that, they were only marginally consulted, mostly in case of error.

Table 2

Short description of the tasks. The steps are those indicated in the instructions (e.g., enter the value of a field, click a button...). The number of messages displayed dynamically during the tasks is indicated. The mean execution time and number of operations are those obtained during the study (20 subjects, 40 executions of each task).

Task	Description	Number of steps	Mean execution time (seconds)	Mean number of KPC operations
1	Fill an on line payment form.	12 1 message	35	88
2	Cut and paste a section of text. Then, modify the copied text.	8 1 message	25	54
3	Install an application on a computer.	11 1 message	19	49
4	Save a document, print it and exit the application.	8	22	58
5	Log on a database and perform a search of a book.	6 1 message	20	45
6	Consult list of students working with a professor.	7	19	49

3.2. Data capture and processing

Data recording and filtering. Each execution of a task (called here a *trial*) was recorded in a log file. The files contained events corresponding to the following elementary actions: *key typed, mouse moved* (elementary movement of the mouse), *mouse pressed and mouse released* (pressing and releasing of mouse buttons). Each event had a time stamp in milliseconds (with a temporal precision of about 15 ms).

Determination of the sequences of operations from the events. For each trial, the sequence of operations was constructed as follows: 1) A *K* was added for each *key typed.* 2) A *P* was added for each block of consecutive *mouse moved* corresponding to mouse displacements one pixel or more in length (in order to avoid introducing Ps for small

involuntary movements). 3) A *C* was added for each consecutive pair of *mouse pressed* and *mouse released*. When intermediary events occurred (e.g., drag: mouse *pressed - moved - released*), *pressed* and *released* were considered separate operations and a *C* was added for each of them. The time stamp of an operation was that of its last corresponding event, and the duration was the delay between its time stamp and that of the previous operation. Fig. 2 depicts a sequence extracted from a log file.

Operation	Arguments	Time stamp (ms)	Duration (ms)	Description
P	100,200	10 210	655	Point at user name field
C	userNameField	10 450	240	Click in user name field
K	"l"	10 751	301	Type a letter
K	"o"	11 061	310	Idem
K	"g"	11 339	278	Idem
K	"l"	11 628	289	Idem
K	"n"	11 887	259	Idem
K	Tab	12 576	689	Press Tab key
K	"p"	12 976	400	Type a letter
K	"a"	13 284	308	Idem
K	"s"	13 739	455	Idem
K	"s"	14 607	868	Idem
K	"w"	15 108	501	Idem
K	"d"	15 456	348	Idem
P	250,300	16 326	870	Point at OK button
C	okButton	16 545	219	Click on OK button

Fig. 2. Example of a KPC sequence extracted from a log file. The user types a user name, a password and clicks a button to confirm. The arguments and descriptions are provided for clarity.

Determination of the optimal sequences. Two types of optimal sequences were determined: *time-optimal*, i.e., minimal execution time, and *length-optimal*, i.e., minimal number of operations. First, the *time-optimal trials* were determined separately for each subject and each task as those with minimal execution time (counted in seconds so that all trials within the same second were considered equivalent). The *time-optimal sequences* were those appearing in the time-optimal trials. However, some of these sequences were mere permutations of the same operations, which leave the prediction of the KPC model invariant. Therefore, *generic sequences* were defined as 3-uples (Nk, Np, Nc) representing the number of K, P and C operations. The *time-optimal (*resp. *length-optimal) generic sequences* were determined from time-optimal (resp. length-optimal) trials.

3.3. Determination of the parameters of the KPC model

The estimates ($\hat{E}_K$, $\hat{E}_P$ and $\hat{E}_C$) of the execution time of K, P and C operations were determined with trained users, in order to avoid delays caused by visual explorations, instruction reading, hesitations, etc. The durations of the operations were averaged across the sequences executed by subjects 1-10, tasks 1-6 and trials 31-40 (600 sequences, more than 32000 operations). The stability of the estimates was assessed post-hoc by comparing the averages and the standard deviations of the execution time of K, P and C obtained with 5, 10, 15 and 20 subjects.

3.4. Data analysis on the KPC operations

Effect of Task, Subject and Training on the execution time of the KPC operations. In order to gain information about the robustness of the estimates ($\hat{E}_K$, $\hat{E}_P$ and $\hat{E}_C$), the effects of Task x Subject x Training on the execution time of K, P and C operations were determined by means of ANOVAs across 20 subjects, 6 tasks and 4 levels of training (trials 1-10, 11-20, 21-30 and 31-40). The amplitude of the main effects was estimated for each factor by averaging the difference between marginal Means across pairs of values. Significant differences between degrees of Training were examined

post-hoc (Dunnett T3), in order to determine whether performance stabilized before trials 31-40.

Effect of context on the execution time of the KPC operations. Covert operations like homing and mental activities may occur preferentially in specific contexts, for instance when there is a transition from keyboard to mouse or vice-versa. Therefore, it is likely that in a specific context, the execution time of K, P and C operations follow skewed or multimodal distributions because of the additional execution times due to covert operations. In first approximation, a context is defined as the operation (K, P or C) preceding the current operation. Table 3 presents the covert activities that may occur in the 9 combinations of Context x Operation in the interfaces used in the present study (Java Swing Look-and-feel). The distributions of execution times of K, P and C operations were determined in each context, across subjects 1-10, tasks 1-6 and trials 31-40. Context-dependent estimates of execution times $\hat{E}_{X,Y}$ were defined as the average execution time of operation Y when it is preceded by operation X for each possible combination X - Y.

Table 3
Contexts for each type of operation. *Rows*: context, i.e., previous operation. *Columns*: current operation. Each cell contains typical situations in Java Look-and-feel interfaces and the expected covert operations.

	K	P	C
K	Middle of the transcription of text field or sequence of commands.	Movement of the mouse after keyboard operation. *Possible covert operations*: Homing, Mental activity	Click without moving the mouse after keyboard operation (infrequent). *Possible covert operations*: Homing, Mental activity

	K	P	C
P	Move pointer away, resume use of keyboard. *Possible covert operations*: Homing, Mental activity.	Impossible	Point and click on a component. *Possible covert operations*: Mental activity
C	Click on a text field and begins transcription. *Possible covert operations*: Homing, Mental activity	Click on a component (e.g., radio button), then resumes pointing. *Possible covert operations*: Mental activity	Double click.

3.5. Data analysis on the performances of the KPC model

Accuracy of KPC model. The accuracy of the KPC was determined for both naive and trained users. For subjects 11-20, the sequences were determined for tasks 1-6, trials 1-10 (naive condition) and 31-40 (trained condition). The estimate $\hat{E}_{KPC}$ was determined for each sequence (see Section 2), using the estimates ($\hat{E}_K$, $\hat{E}_P$, $\hat{E}_C$) previously obtained with subjects 1-10 (see Section 3.3). Then, $\hat{E}_{KPC}$ was compared to the actual execution time ET in two ways: 1) Error of prediction for individual trials, $abs(ET-\hat{E}_{KPC})$. The mean and standard deviation of that error were determined across subjects, tasks and trials in the naive and trained conditions. 2) Bias, i.e., difference between average $\hat{E}_{KPC}$ and average ET, in naive and trained condition. A negative difference means that the model is optimistic, i.e. the execution time is underestimated, whereas a positive difference corresponds to a pessimistic model.

Comparison with KLM. The accuracy of the KLM was also measured for both naïve and trained users. We used a modified version of KLM (Hudson et al., 1999), equivalent to the original model (Card et al., 1980) in terms of predictions. However, in the modified version, the heuristic rules for inserting the mental activities *M* are simple and unambiguous, thus allowing their automation. For each subject, task and trial, the estimate of the execution time $\hat{E}_{KLM}$ (see Section 2) was determined from the actual

sequence. Then, for each task, the minimum (across subjects and trials) was retained. By doing so, we presumably obtained "the optimal sequence", as required by the rationale of the KLM.

Comparison with context-dependent KPC. We wanted to determine whether using context information improved the accuracy of prediction. Thus, we used the *context-dependent KPC (c-dKPC) model* to predict execution time. To that end, we determined an estimate $\hat{E}_{c\text{-}dKPC}$ obtained by assigning to each operation O_i the estimate $\hat{E}_{Oi\text{-}1,Oi}$, where $O_{i\text{-}1}$ is the previous operation in the sequence (see Section 3.4). When an operation was not immediately preceded by another operation, e.g., first operation of the sequence, the estimate $\hat{E}_{Oi}$ ($\hat{E}_K$, $\hat{E}_P$ or $\hat{E}_C$) was used instead of $\hat{E}_{Oi\text{-}1,Oi}$.

Effects of Task, Subject and Training on the accuracy of prediction of KPC model. Previous analyses determined the accuracy of the KPC, KLM and context-context dependent KPC for naive and trained subjects. However, the accuracy may vary according to the Task and the Subject, because the sequences may be different (in addition to the effect of these factors on the execution times of the elementary operations K, P and C). Thus, the effects of Task x Subject x Training on the error of prediction (dependent variable) were determined directly by means of ANOVAS, across 10 subjects (11-20), 6 tasks and 2 levels of Training (trials 1-10 and 31-40, like in the determination of the accuracy of prediction). The amplitude of the effects was estimated for each factor by averaging the difference between marginal Means across pairs of values.

4. Results

4.1. Optimal sequences

The numbers of time-optimal sequences for the six tasks (subjects 1-20 and trials 1-40) are shown in Table 4. The numbers of different time-optimal generic sequences

range between 15 and 21. This indicates that the users employed a variety of "optimal ways" for executing the tasks. Conversely, the proportion of optimal trials is relatively low, between 3.4% and 7.5% of all trials. In other terms, skilled users used optimal sequences less than 1 time out of ten.

Table 4
Numbers of time-optimal sequences (sequences that have the shortest execution time (counted in seconds) for a given subject. *Columns*: generic sequences, sequences, occurrences (trials in which an optimal sequence was used) and percentage of trials in which an optimal sequence was used.

Task	Time-optimal generic sequences (Nc, Nk, Np)	Time-optimal sequences	Time-optimal trials	Proportion of time-optimal trials
1	21	21	27	3.4%
2	19	23	42	5.3%
3	22	38	60	7.5%
4	29	33	37	4.6%
5	20	24	36	4.5%
6	15	26	50	6.3%

The numbers of length-optimal sequences for the six tasks are shown in Table 5. The numbers of length-optimal generic sequences range between 5 and 16. Like for time-optimal sequences, it is clear that there are different "optimal ways" of realizing the tasks. Although there are fewer length-optimal than time-optimal sequences, length-optimal trials are more frequent than time-optimal trials, representing from 5% to 15.9% of all trials. In other terms, subjects are more often length- than time- optimal. Care should be taken however in comparing these proportions, because length is discrete whereas time is continuous, i.e., the number of time-optimal trials may depend on the temporal precision used (here: 1 second).

Table 5
Numbers of length-optimal sequences (sequences that use a minimal number of operations K, P or C). *Columns*: generic sequences, sequences, occurrences (trials in which an optimal sequence was used) and percentage of trials in which an optimal sequence was used.

Task	Length-optimal generic sequences (Nc, Nk, Np)	Length-optimal sequences	Length-optimal trials	Proportion of length-optimal trials
1	14	14	62	7.8%
2	8	9	127	15.9%
3	10	15	59	7.4%
4	16	16	40	5.0%
5	9	16	118	14.8%
6	5	8	71	8.9%

Fig. 3 depicts the distribution of all trials (1-40) for two subjects and two tasks, in two dimensions (length vs. execution time). It can be observed that: 1) a significant proportion of the last 10 trials are not optimal, 2) some trials are length-optimal but not time optimal (e.g., Subject 1, Task 2, leftmost dots), 3) some trials are time- but not length- optimal (e.g., Subject 1, Task 1, lowest empty dot) and 4) some trials are both time- and length-optimal (e.g., Subject 1, Task 2, Subject 2 Tasks 1 and 2, lowest and leftmost dots).

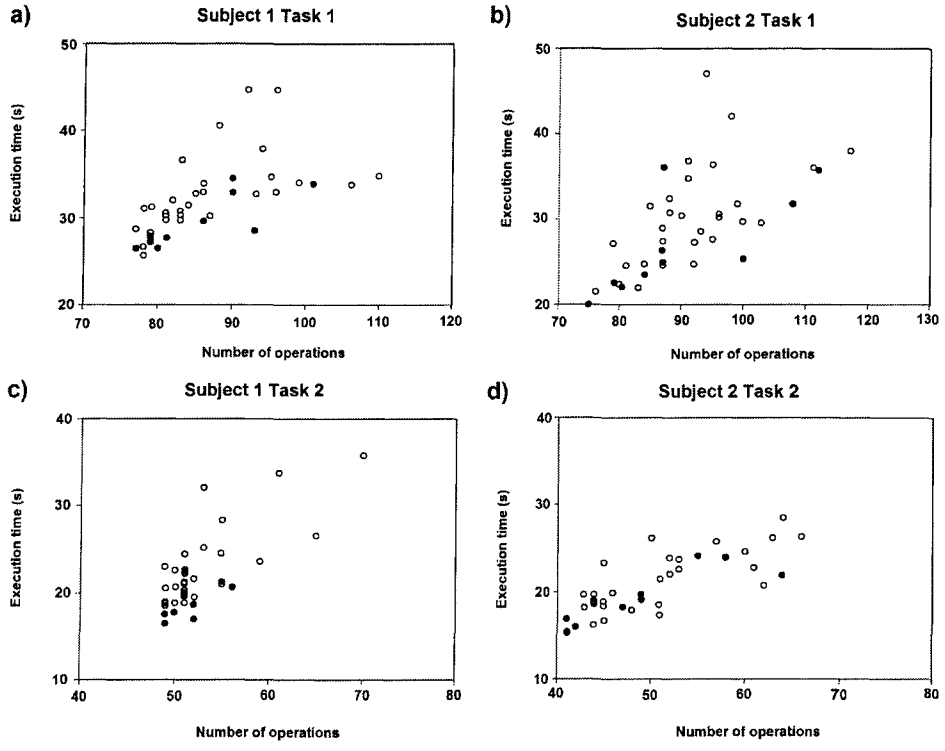


Fig. 3. Distribution of the trials for two subjects and two tasks. *Plain dots*: last ten trials (31-40), *empty dots*: trials 1-30. *Vertical*: execution time (seconds). *Horizontal*: number of operations. Time-optimal trials are at the bottom. Length-optimal trials are on the left. a) Subject 1, Task 1. b) Subject 2, Task1. c) Subject 1, Task 2. d) Subject 2, Task 2.

4.2. Estimates of the execution time of the KPC operations

For the needs of the present study, we obtained as estimates the values given in Table 6, namely $\hat{E}_K$ = 331 ms, $\hat{E}_P$=701 ms and $\hat{E}_C$=235 ms. Note that these values are rounded off for simplicity in Eq. 1 (330 ms, 700 ms and 235 ms, respectively).

Table 6

Average execution time of K, P and C operations. *Bold*: means established across subjects 1-10, tasks 1-6, trials 31-40, σ: standard deviation, n: number of observations.

Operation	Average execution time (ms)
K	**331** σ=348, n=22147
P	**701** σ=677, n=5433
C	**235** σ=301, n=5159

The execution time of K, P and C was relatively insensitive to the number of subjects used to calculate the average (Table 7). The dispersion of the data (standard deviation) did not decrease with the number of subjects and the differences between averages are quite below the dispersion. Although their statistical significance has not been established formally, these results justify post-hoc the decision of using only the data of 10 subjects for establishing the estimates $\hat{E}_K$, $\hat{E}_P$ and E_C.

Table 7

Average execution time of K, P, and C operations obtained with 5, 10, 15 and 20 subjects. *Bold*: means established across tasks 1-6, trials 31-40. σ: standard deviation, n: number observations. Δ: relative difference with the values of Table 6.

	Subjects 1-5	Subjects 1-10 (reference for Δ)	Subjects 1-15	Subjects 1-20
K	**301 , Δ=9%** σ=307, n=11339	**331 , -----** σ=348, n=22147	**334 , Δ=1%** σ=356, n=33571	**325 , Δ=2%** σ=357, n=44829
P	**660 , Δ=6%** σ=503, n=660	**701 , -----** σ=677, n=5433	**703 , Δ=0%** σ=749, n=8046	**689 , Δ=2%** σ=750, n=11648
C	**232 , Δ=1%** σ=314, n=2338	**235 , -----** σ=301, n=5159	**229 , Δ=3%** σ=306, n=7611	**208 , Δ=11%** σ=292, n=10924

4.3. Effect of Task, Subject and Training on execution time of the KPC operations

There were significant effects of Task, Subject and Training on the execution time of the KPC operations. All effects were significant at $p < 0.0005$ (Table 8). All

interactions between these factors were also significant at $p < 0.0005$ but one, namely the interaction Task x Training for pointing operations P (Table 8). However the amplitudes of these effects range between 9% and 36% of the execution time (Table 9). This is comparable to the errors of prediction reported in analytical modeling (see Discussion).

Specifically, the effects of Training, as estimated post-hoc (Dunnett T3) were as follows. For C and P operations, there were no significant difference between blocks 3 (trials 21-30) and 4 (trials 31-40), i.e., performance was stabilized. For *K* operations, the difference between blocks 3 and 4 was significant, but its amplitude was low (9 ms difference between blocks 3-4 compared to 40 ms difference between blocks 1-2). In other terms, performance was almost stabilized after 20 trials. It is worth recalling that the execution times of the KPC operations include covert operations (as well as the eventual consultation of instructions during the task). The effects of training may thus reflect a change in the number and/or the duration of covert operations, rather than a speed up in typing, pointing and clicking.

Table 8

Effects of Task, Subject and Training on the execution time of K, P and C operations (across subjects 1-20, tasks 1-6, trials 1-40). **: significant, $p<0.0005$. *ns.*: not significant ($p=0.1$).

	Task	Subject	Training	Task x Subject	Task x Training	Subject x Training
C	F(5, 44559) 244.78**	F(19, 44559) 41.20**	F(3, 44559) 50.34**	F(95, 44559) 6.47**	F(15, 44559) 3.49**	F(57,44559) 2.26**
K	F(5, 180591) 336.40**	F(19, 180591) 174.89**	F(3, 180591) 167.54**	F(95, 180591) 12.18**	F(15, 180591) 4.81**	F(3, 180591) 5.13**
P	F(5, 47727) 19.90**	F(19, 47727) 30.61**	F(3, 47727) 197.74**	F(95, 47727) 2.84**	F(15, 47727) 1.02 ns.	F(57, 47727) 3.09**

Table 9

Amplitude of the effect of Task, Subject and Training on the execution time of K, P and C operations (across subjects 1-20, tasks 1-6, trials 1-40). *Bold*: average across the pairs of values of the difference between marginal Means, divided by the average execution time across subjects 1-20, tasks 1-6, trials 1-40 (unlike the averages of section 4.2, which were obtained with trials 31-40). These values are indicated in parenthesis.

	Task	Subject	Training
C	**36%**	**20%**	**11%**
	(91 / 251 ms)	(51 / 251 ms)	(28 / 251 ms)
K	**20%**	**22%**	**10%**
	(66 / 340 ms)	(74 / 340 ms)	(33 / 340 ms)
P	**9%**	**18%**	**21%**
	70 / 807 ms	144 / 807 ms	167 / 807 ms

4.4. Context-dependent estimates of the execution time of the KPC operations

Table 10 presents the average execution time of the KPC operations in each context. These values will be adopted for the estimates of context-dependent execution times $\hat{E}_{X,Y}$. Fig. 4 depicts the distributions of execution times. All the distributions are skewed, i.e., they go far to the right, presumably because of the additional execution time due to covert operations. In the point-and-click (P-C) combination, the distribution of execution time shows a secondary peak around 1100 ms, which presumably represents a relatively frequent pause (or mental activity) just before clicking. Also, it can be observed that the distribution of execution time shifts markedly to the right when the context requires a homing, from keyboard to mouse or vice-versa (see the differences between combinations K-P and C-P and between P-K or C-K and K-K in Table 10 and Fig. 4).

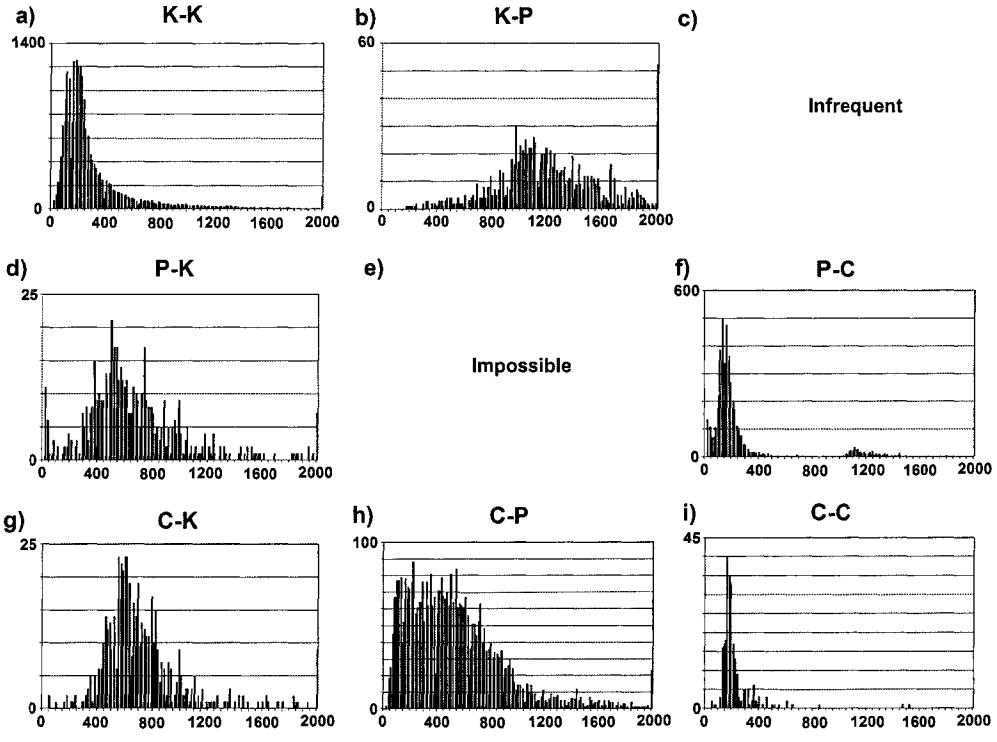


Fig. 4. Distributions of context-dependent execution time. *Rows*: contexts. *Columns*: current operations. *Vertical*: frequency. *Horizontal*: execution time (ms). Histograms not presented for the cases K-C (insufficient number of values, n=26) and P-P (impossible).

Table 10

Average context-dependent execution time (ms) of K, P, C operations. *Bold*: means established across subjects 1-10, tasks 1-6, trials 31-40. σ: standard deviation, n: number of observations. The operations that were not immediately preceded by another operation were not considered.

	K	P	C
K	**312** σ=335, n=21025	**1240** σ=459, n=1045	**1497** σ=584, n=22
P	**644** σ=409, n=549	Impossible	**230** σ=302, n=4884
C	**748** σ=395, n=543	**555** σ=689, n=3818	**216** σ=152, n=253

4.5. Accuracy of the KPC, KLM and context-dependent KPC models

The average error of the KPC model (context-dependent or not) is about 26% of the average execution time for naive users and 18% for trained users (Fig. 5 and Table 11). Also, the dispersion is markedly lower for trained (σ=20% of the execution time) than naive users (σ=40%). The error is markedly higher in the case of the KLM (naive: 33%; trained 69%). Unlike the KPC, in the present case, the KLM is more accurate for naive than trained users. In order to verify whether the poor accuracy of the KLM was caused by the use of a single optimal sequence, we also calculated the KLM with actual sequences (Table 11). The error was still higher (naive: 42%, trained: 73%).

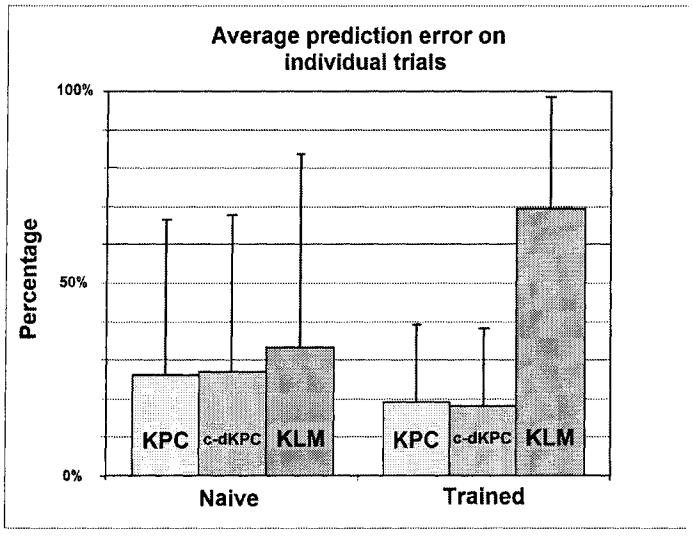


Fig. 5. Error in prediction for the KPC, context-dependent KPC (c-dKPC) and KLM models. *Vertical*: average error across subjects 11-20, tasks 1-6, trials 1-10 (naive) and 31-40 (skilled) divided by average execution time (same observations). *Thick bars*: average. *Thin bars*: standard deviation.

Table 11

Error in prediction for the KPC, context-dependent KPC, KLM and KLM with actual sequences models. *Bold*: average error across subjects 11-20, tasks 1-6, trials 1-10 (naive) and 31-40 (skilled) divided by average execution time (same observations). σ: standard deviation. Number of observations: 600.

	Naive Average	Trained Average
KPC	**26%**	**19%**
	σ=40	σ=20
context-dependent KPC	**27%**	**18%**
	σ=41	σ=20
KLM	**33%**	**69%**
	σ=50	σ=29
KLM with actual sequences	**42%**	**73%**
	σ=28	σ=38

All the models presented negative biases for naive subjects, i.e., all models underestimated the execution time (Fig. 6). The KLM was less biased for naive subjects than the KPC models (note that a low bias only indicates that the Means of the predicted and actual execution times are similar, even if their deviations are quite dissimilar. In other terms, the bias may be low even if the model does not capture accurately the factors that affect the execution time). Conversely, for trained subjects, the KLM presents a marked positive bias, i.e., it is pessimistic. The KPC is also pessimistic, but the bias is markedly lower than that of the KLM (less than 10%). The context-dependent KPC is almost unbiased.

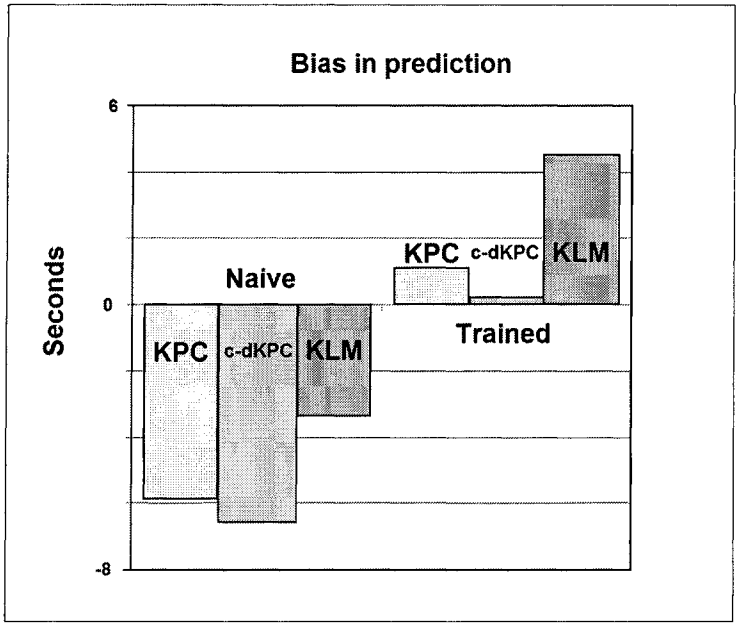


Fig. 6. Bias in prediction for KPC, context-dependent KPC (c-dKPC) and KLM models. Difference (signed) in seconds between averages of the estimate and of the actual execution time. Averages were computed across subjects 11-20, tasks 1-6, trials 1-10 (naive) and 31-40 (skilled).

4.6. Effects of Tasks, Subjects and Training on the accuracy of the KPC model

The factors Task, Subject and Training had a significant effect on the accuracy of the KPC model, $p<0.0005$. The interactions between these factors were also significant, $p<0.01$ (Table 12). The amplitude of the main effect of Task was 32% of the average error of prediction, and that of Subject was 38% of the error of prediction. The amplitude of the effect of Training was markedly higher, 61 %. However, this effect is artificially high because it characterizes the difference between the first 10 and last 10 trials without intermediary levels of training. Recall that intermediary trials 11-30 were removed in order to use the same data than for the estimation of the accuracy of the models.

Table 12

Effects of Task, Subject and Training on the accuracy of prediction of the KPC model (across subjects 11-20, tasks 1-6, trials 31-40). Δ = error, i.e., abs (ET - $\hat{E}_{KPC}$). **: significant at $p<0.0005$. *: significant at $p<0.01$.

	Task	Subject	Training	Task x Subject	Task x Training	Subject x Training
Δ	F(5,1080) 7.96**	F(9, 1080) 8.02*	F(1, 1080) 58.14**	F(45, 1080) 1.59*	F(5, 1080) 4.89**	F(9, 1080) 8.24**

Table 13

Amplitude of the effect of Task, Subject and Training on the accuracy of the KPC model. Bold: average across the pairs of values of the difference between marginal Means, divided by the average error across subjects 11-20, tasks 1-6, trials 1-10 (naive) and 31-40 (trained). These values are indicated in parenthesis.

	Task	Subject	Training
Δ	**32%** (1855 / 5789 ms)	**38%** (2217 / 5789 ms)	**61%** (3524 / 5789 ms)

5. Discussion

5.1. Predicting execution time from overt operations

The prediction error with the KPC model was 19% for trained users, and around 26% for naive users. The prediction seems robust, i.e., it was only moderately sensitive to inter-individual variability and to tasks content. In comparison, simple predictive models found in the literature typically present an error rate of 20% or above (see review in Ivory & Hearst, 2001). For instance, in Hudson et al. (1999), the prediction error was about 35% (18 seconds predicted, vs. 28 seconds measured on more than 100 users). The acceptable results of the KPC are all the more noticeable since the model only uses overt operations, i.e., the model can be constructed entirely from recorded experimental data.

The KPC model was negatively biased for naive users, i.e., it underestimated their execution time. This is compatible with the amount of covert activity that

accompanies the learning of a new task, e.g., conscious decisions, hesitations, visual explorations, etc. Conversely, the model was slightly pessimistic (positive bias) for trained users. This bias was eliminated when the context of the operations was considered (context-dependent KPC model), e.g., press key after clicking at the beginning of a text transcription. By considering the context, we implicitly accounted for the effects of covert operations, which occur preferentially in some situations, like hand movements and/or mental pauses. However, although the bias was eliminated, the overall prediction accuracy was not improved.

This suggests that the limits of prediction are settled by performance variability (across users, tasks and degree of training), and that fine modeling would bring little benefits at the expense of additional complexity. Note that the complexity of model building (and of updating models when the interface changes) may explain that analytical modeling is not widely used in interface development (Ivory & Hearst, 2001; John et al., 2004). In this context, the KPC model seems a good candidate for modeling simple tasks on standard graphical interfaces. Indeed, with complex tasks, the KPC model may be limited by the difficulty of collecting representative sequences of operations (see below) and by the possible increase of variability of covert operations due to a more difficult longer training and/or more complex mental activities.

5.2. Covert operations were present, but are they worth modeling?

In the present tasks, covert operations were present even after extensive training. Covert operations of any duration seemed to occur in all possible contexts, as indicated by the skewed distributions of execution time (Fig. 4). Covert operations were clearly identified in only two cases: just before clicking (presumably short mental activities of about 1100 ms) and in case of transition between mouse and keyboard operations (presumably hand homing). However, covert activities were by no means mandatory. For instance, we observed that some users avoided homing by leaving their dominant hand on the mouse while typing with the non-dominant hand.

Modeling covert operations of any duration and occurring in any context may bring only marginal benefits. The bias, but *not* the accuracy of prediction was improved when covert operations were incorporated to the KPC model. This may have been a consequence of the approach adopted here, i.e., model covert operations by assuming that they occur preferentially in some contexts, e.g., homing between a click and a key press, or a mental pause just before clicking.

The KLM, which explicitly models covert operations, was worse than the KPC model, whether context-dependent or in its basic form. The comparison between these two models is significant, because KPC and KLM only differ in the modeling of covert operations. The KLM widely overestimated the execution time of skilled trained users, which is paradoxical for a model initially limited to expert users. As expected, the bias was even worse when actual sequences were used instead of the optimal sequence. Indeed, KLM overestimated pointing time (1100 ms whereas the actual execution time was 700 ms), but pointing represented less than 20% of the operations, thus this difference does not explain the poor accuracy of the KLM. The most likely explanation is that KLM overestimated the number and duration of mental activities in skilled trained users (see similar results in a 16 months longitudinal study by Nilsen et al., 1993).

The "expert user" modeled by KLM may thus represent an intermediary phase between initial learning and sustained practice. This is compatible with the view that practiced skilled users make extensive use of visuo-motor automatisms, with low attention and cognitive demands (Woodworth, 1899; Shiffrin & Schneider, 1977). If this viewpoint is correct, model-based approaches may be limited by a trade-off between precision, robustness (i.e., validity for a wide range of degrees of training) and model complexity. Simple models (e.g., KLM, but see also GLEAN, Kieras et al., 1995, Apex-CPM, John et al., 2002, ACT-Simple, Salvucci & Lee, 2003) may be imprecise and robust, like KPC. Indeed, high accuracy may be attained with simple models, but only for a precise stage of the learning curve, because these models do not represent the evolution of the user's performance.

If both precision and robustness are required, it may be necessary to model (different stages of) the learning process in complex modeling frameworks (e.g., SOAR, Rosenbloom et al., 1991, EPIC, Meyer & Kieras, 1997, ACT-R, Anderson & Lebiere, 1998, UNICOM, Doane et al., 2000, ACT-R/PM, Byrne & Anderson, 2001, X-PRT, Tollinger et al., 2005). These frameworks provide many parameters (degrees of freedom), thus allowing a good fitting with experimental data. However, whatever the degree of refinement of a model, the inter- and intra-individual variability of performance settles a limit to the prediction accuracy (as measured by the mean error of prediction, not by the bias). Given the variability found even in the simple tasks of the present study, it is worth wondering whether complex modeling is justified. The KPC model represents the opposite approach: it is presumably one of the simplest predictive models, and precision is sacrificed for obtaining robustness, i.e., the capacity of predicting for a variety of tasks, users and degrees of training.

5.3. The quest for optimal sequences and the experimental approach

Predictive models like KLM and the GOMS family (see review in John & Kieras, 1996) are based upon the Principle of Rationality (Card, Moran & Newell, 1983): 1) users execute the tasks by moving in a problem space towards their goal, and 2) users are rational, i.e., they develop optimally efficient sequences of operations for reaching these goals. This principle allows the modeler to determine analytically the actions of the user from the functionalities of the interface.

However, modern GUIs and WEB interfaces are generally unconstrained, and they run in an open environment. This allows for many efficient ways of executing the tasks. The present findings support the viewpoint that even for simple tasks, there are many efficient sequences (time-optimal, length-optimal or both). These sequences are structurally different, i.e., they are not mere permutations of the same operations. In addition, the results do not indicate that skilled users restrict themselves to a specific type of optimality. Instead, the variety of sequences found is more compatible with the notion of sufficient sequences, i.e., maintaining some parameter(s) within an acceptable

range (Todorov & Jordan, 2002), or with the viewpoint that skilled users' behavior primarily satisfies a set of task-related, interaction and psychological constraints (Vera et al., 2004).

In addition, the present study provided two significant findings, as depicted by Fig. 3. They are the following: 1) *Even in simple tasks, training does not eliminate variety*, i.e., skilled, trained users did not use a unique optimal sequence. 2) *Even in simple tasks, training does not eliminate errors and inefficiency*, i.e., a significant proportion of the sequences was not optimal. The proportion of errors found in skilled trained users confirms the viewpoint of Card et al. (1983) and Norman (1988). Note that these effects (variety, non-optimality) may be even more important in applications requiring non-trivial execution strategies, which may still be sub-optimal after years of practice (Bhavnani & John, 1997).

In this context, it seems safer to collect experimentally the actual sequences from actual users than to determine them analytically. Indeed, the experimental approach proposed here is more time consuming than the analytical approaches in which the representative sequences are defined by the analyst, either directly (e.g., Kieras, 1993) or by means of a prototype (Hudson et al., 1999; John et al., 2004). However, the experimental approach allows collecting of actual sequences, even when they may have escaped to the developer and/or the analyst (e.g., using web search instead of internal navigation, Augustine & Greene, 2002). Also, by filtering the sequences according to some optimality criterion (time and/or length), inefficient sequences and sequences with errors are automatically discarded, without complex and time consuming error analysis (that cannot be entirely automated, Ivory & Hearst, 2001).

5.4. Skilled users vs. expert users

In conducting this study, we kept in mind that users have changed in recent years. Graphical interfaces become standardized, and people are exposed more and more frequently to standard interfaces. It is likely that a core of general automatisms may develop from the repetitive use of standard graphical interfaces. Users that have such

automatisms may be qualified as *skilled.* Studies have been conducted on the evolution of performance with practice on a specific interface (e.g., Nilsen et al., 1993), and the evolution of performance according to a type of components, e.g., icons vs. labels (Wiedenbeck, 1999). However, little is known about transfer phenomena, namely how practice with a specific interface or with specific components contributes to the development of general skills, or conversely how preexisting skills affect the learning and posterior use of a novel application (e.g., Bhavnani et al., 1997).

Skilled users have some of the features attributed to "experts", e.g., knowledge and practice, but unlike "experts" they can be naive with a given application or a given task. This situation is commonplace with applications and WEB sites that support hundreds of functionalities. In this case, the once used concept of expert user is ambiguous, and models of expert users may be inaccurate.

The present study brought some findings about skilled users. 1) They improve their performances on specific tasks, like any user. 2) They have multiple ways of executing a task, whether they are trained or not. In other terms, training does not eliminate variety. 3) They use optimal sequences only marginally. 4) They execute covert activities (presumably hand homing and mental activities) even after training. 5) Their activity is correctly modeled by the overt operations, even when covert operations are present. These findings are for now limited to the set of tasks used here. However, they may be a starting point for further studies on skilled users.

6. Concluding remarks

The KPC model presented here offers a simple and practical approach for usability evaluation: 1) Implement a prototype of the interface that records the overt operations of the users. 2) Perform a test with skilled users in order to determine the sequences of overt operations that are efficient and/or frequent. 3) Estimate execution time of skilled users on specific tasks from the numbers of overt operations in the sequences, namely pointing movements, clicks and key presses. This approach is compatible with an iterative, user-centered development process, such as proposed in the

ISO 13407 Norm. It does not require arbitrary assumptions on (or complex models of) the user's activity or cognitive functioning, or an exhaustive compilation of all the possible ways of realizing the tasks, whether these ways were predicted or not by the developers. In case of changes in hardware (input-output devices) and/or software (interface), this experimental approach allows updating easily the data and the predictions in order to support the changes. The whole approach relies on the assumption that, in industrialized countries, users have general skills with computer interfaces, because standardized interfaces are now part of people's technological environment.

Acknowledgements

This research was conducted at the Laboratoire d'Environnements de Synthèse et d'Interfaces Avancées (LESIA) and funded by a PSIRE grant 1-5428-000 of École de Technologie Supérieure (ÉTS), Montréal. The authors would like to thank Hassan Ait-El-Cadi for his valuable collaboration, and the participants that collaborated willingly to the study.

REFERENCES

Ackerman, P.L., 1988. Determinants of individual differences during skill acquisition: cognitive abilities and information processing. Journal of Experimental Psychology: General, 117(3), 288-318.

Anderson, J.R., 1982. Acquisition of cognitive skill. Psychological Review, 89(4), 369-406.

Anderson, J.R., Lebiere, C., 1998. The atomic components of thought. Hillsdale, NJ: Lawrence Erlbaum.

Augustine, S., and Greene, C., 2002. Discovering how students search a library web site: a usability case study. College and Research Libraries, 63(4), 355-365.

Bhavnani, S. K., John, B.E., 1997. From sufficient to efficient usage: an analysis of strategic knowledge. In Proceedings of the SIGCHI conference on human factors in computing systems (Atlanta, Georgia), 91-98.

Byrne, M. D., Anderson, J.D., 2001. Serial modules in parallel: the psychological refractory period and perfect time-sharing. Psychological Review, 108(4), 847-869.

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.

Card, S.K., Moran, T.P., Newell, A., 1983. The psychology of human-computer interactions. New Jersey: Lawrence Erlbaum.

Doane, S.M., Pellegrino, J.W., Klatzky, R.L., 1990. Expertise in a computer operating system: conceptualization and performance. Human-Computer Interaction, 5, 267-304.

Doane, S.M., Sohn, Y.S., McNamara, D.S., Adams, D., 2000. Comprehension-based skill acquisition. Cognitive Science, 24(1), 1-52.

Durham, A.G., Emurian, H.H., 1998. Learning and retention with a menu and a command line interface. Computers in Human Behavior, 14(4), 597-620.

Embley, D.W., & Nagy, G., 1981. Can we expect to improve text editing performance? In Proceedings of the 1982 conference on human factors in computing systems, Gaithersburg (Maryland), 152–156.

Goldberg, A., 1988. A history of personal workstations. NY: Addison-Wesley.

Hudson, S.E., John, B.E., Knudsen, K., Byrne, M.D., 1999. A tool for creating predictive performance models from user interface demonstrations. CHI letters, 1(1), 93-102.

Ivory, M.E., Hearst, M.A., 2001. The state of the art in automating usability evaluation of user interfaces. ACM Computing Surveys, 33(4), 470-516.

John, B.E., Kieras, D.E., 1996. The GOMS family of user interface analysis techniques: Comparison and contrast. ACM Transactions on Computer-Human Interaction, 3(4), 320-351.

John, B.E., Vera, A., Matessa, M., Freed, M., Remington, R., 2002. Automating CPM-GOMS. CHI letters, 4(1), 147-154.

John, B.E., Prevas, K., Salvucci, D. D., Koedinger, K., 2004. Predictive human performance modeling made easy. In Proceedings of the SIGCHI conference on human factors in computing systems (Vienna, Austria), 6(1), 455-462.

Kieras, D., 1993. Using the keystroke-level model to estimate execution times. Unpublished Report, University of Michigan. Available at http://www.pitt.edu/~cmlewis/KSM.pdf (last consulted 2005-08-16)

Kieras, D.E., Wood, S.D., Abotel, K., Hornof, A., 1995. GLEAN: A computer-based tool for rapid GOMS model usability evaluation of user interface designs. International Journal of Human-Computer Studies, 22, 365-394.

Lebedev, S, Tsui, WH, Van Gelder, P., 2001. Drawing movements as an outcome of the principle of least action. Journal of Mathematical Psychology, 45(1), 43-52.

Meyer, D.E., Kieras, D.E., 1997. A computational theory of executive cognitive processes and multiple-task performance: part 2. Accounts of psychological refractory-period phenomena. Psychological Review, 104(4), 749-791.

Murray, K.B., Häubl, G., 2003. A human capital perspective of skill acquisition and interface loyalty. Communications of the ACM, 46(12), 272-278

Nilsen, E., Jong, H., Olson, J.S., Rueter, H., Mutter, S., 1993. The growth of software skill: A longitudinal look at learning & performance. In Proceedings of the SIGCHI conference on human factors in computing systems (Amsterdam, The Netherlands), 149-156.

Newell, A., Rosenbloom, P.S., 1981. Mechanisms of skill acquisition and the law of practice, in: Anderson, J.R. (Ed.), Cognitive Skills and their Acquisition, Hillsdale, NJ: Lawrence Erlbaum, 1-55.

Norman, D. A., 1988. Categorization of action slips. Psychological Review, 88, 1-15.

Rosenbaum, D.A., Gregory, R.W., 2002. Development of a method for measuring movement-related effort: biomechanical considerations and implications for Fitts' law. Experimental Brain Research, 142(3), 365-373.

Rosenbloom, P.S., Laird, J.E., Newell, A., McCarl, R., 1991. A preliminary analysis of the SOAR architecture as a basis for general intelligence. Artificial Intelligence, 47, 289-325.

Salvucci, D.D., Lee, F.J., 2003. Simple cognitive modeling in a complex cognitive architecture. In Proceedings of the SIGCHI conference on human factors in computing systems, 5(1), 265-272.

Shiffrin, R. M., Schneider, W., 1977. Controlled and automatic information processing: II. Perception, learning, automatic attending and a general theory. Psychological Review, 84, 127-190.

Todorov, E, Jordan, M.I., 2002. Optimal feedback control as a theory of motor coordination. Nature Neuroscience, 5(11), 1226-1235.

Tollinger, I., Lewis, R.L., McCurdy, M., Tollinger, P., Vera, A., Howes, A., Pelton, L.J., 2005. Supporting efficient development of cognitive models at multiple skill levels: exploring recent advances in constraint-based modeling. In Proceedings of the SIGCHI conference on human factors in computing systems, 411-420.

Vera, A., Howes, A., McCurdy, M., Lewis, R.L., 2004. Constraint satisfaction approach to predicting skilled interactive cognition. In Proceedings of the SIGCHI conference on human factors in computing systems (Vienna, Austria), 6(1), 121-128.

Welford, A.T., 1968. Fundamentals of skill. London, Methuen.

Welford, A.T., 1978. Mental workload as a function of demand, capacity, strategy and skill. Ergonomics, 21 (3), 151-167.

Wiedenbeck, S., 1999. The use of icons and labels in an end user application program: an empirical study of learning and retention. Behaviour & Information Technology, 18(2), 68-82.

Woodworth, R.S., 1899. The accuracy of voluntary movement. Psychological Review, 3 (whole number 13).

BIBLIOGRAPHIE

Anderson, J. R. (1990). *The adaptive character of thought*. Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Anderson, J. (1993). *Rules of the mind*. Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Augustine, S. & Greene, C. (2002). Discovering how students search a library web site: a usability case study. *College and Research Libraries*, 63(4), 355-365.

Barnard, P.J. (1987). Cognitive resources and the learning of human-computer dialogs. *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction*, 112-158.

Barnard, P.J. & Teasdale, J.D. (1991). Interacting cognitive subsystems: a systemic approach to cognitive-affective interaction and change. *Cognition and Emotion*, 5, 1-39.

Baumeister, L.K., John, B.E. & Byrne, M.D. (2000). A comparison of tools for building GOMS models. *CHI letters*, 2(1), 502-509.

Beard, D.V., Smith, D.K. & Denelsbeck, K.M. (1996). Quick and dirty GOMS: a case study of computed tomography interpretation. *Human-Computer Interaction*, 11(2), 157 180.

Bhavnani, S.K. & John, B.E. (1996). Exploring the unrealized potential of computer aided drafting. *Proceedings of the CHI '96*, 332-339.

Bhavnani, S.K. & John, B.E. (1997). From sufficient to sufficient usage: an analysis of strategic knowledge. *Proceedings of the SIGCHI '97 conference on Human Factors in Computing Systems*, Atlanta, Georgia, 91-98.

Brock, D., Hix, D., Dievendorf, L. Jr. & Trafton, J.G. (1995). Extending the user action notation for research in individual differences. *Proceedings of the Human Factors and Ergonomics Society 39th annual meeting*, 253-257.

Byrne, M.D., Wood, S.D., Sukaviriya, P., Foley, J.D. & Kieras, D.E. (1994). Automating interface evaluation. *Proceedings of the CHI '94 conference on Human Factors in Computing Systems*. NY: Addison-Wesley, 232-237.

Byrne, M.D., John, B.E., Wehrle, N.S. & Crow, D.C. (1999). The tangled web we wove: a taskonomy of WWW use. *Proceedings of the CHI '99 conference on Human Factors in Computing Systems*. NY: Addison-Wesley, 544-551.

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.

Card, S.K., Moran, T.P. & Newell, A. (1983). *The psychology of human-computer interaction*. Hillsdale, NJ: Lawrence Erlbaum Associated, Inc.

Carr, D.A. (1996). Toward more understandable user interface specifications. *DSV-IS'96 Informal Proceedings*.

Chi, E.H., Pirolli, P. & Pitkow, J. (2000). The scent of a site: a system for analyzing and predicting information scent, usage, and usability of a web site. *Proceedings of the Conference on Human Factors in Computing Systems '2000*. The Hague, The Netherlands, April, 161-168.

Gallego, M.G., Rodriguez, M.G., Salan, A. & Fernandez, D.T. (2002). Automatic navigability testing system for web sites. *Simposio de Informatica y Telecommunicaciones SIT'02*, Sevilla, Spain, 169-180, [En ligne]. http://tdg.lsi.us.es/~sit02/res/papers/gonzalez-gallego.pdf (Consulté le 3 juin 2005).

Glenn, F.A., Schwartz, S.M. & Ross, L.V. (1992). Development of a human operator simulator version v (HOS-V): design and implementation. *U.S. Army Research Institute for the Behavioral and Social Sciences*, PERI-POX, Alexandria, VA.

Hammontree, M.L., Hendrickson, J.J. & Hensley, B.W. (1992). Integrated data capture and analysis tools for research and testing on graphical user interfaces. *Proceedings of the Conference on Human Factors in Computing Systems* (Monterey, CA, May), 431-432.

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*, 8(3), 181-203.

Hartson, H.R. & Mayo, K.A. (1994). A framework for precise reusable task abstractions. *Proceedings of the Eurographics Workshop on Design, Specification and Verification of Interactive Systems*, 147-164.

Hartson, H.R., Castillo, J.C., Kelso, J. & Neale, W.C. (1996). Remote evaluation: the network as an extension of the usability laboratory. *Proceedings of the Conference on Human Factors in Computing Systems* (Vancouver, Canada, April), 228-235, [En ligne]. http://www.acm.org/sigchi/chi96/proceedings/papers/Hartson/hrh_txt.htm (Consulté le 12 janvier 2005).

Hudson, S.E., John, B.E., Knudsen, K. & Byrne, M.D. (1999). A tool for creating predictive performance models from user interface demonstrations. *CHI letters*, 1(1), 93-102.

ISO/CEI 9126 (1991). Technologies de l'information – Évaluation des produits logiciels – Caractéristiques de qualité et directives d'utilisation.

Ivory, M.Y. & Hearst, M.A. (2001). The state of the art in automating usability evaluation of user interfaces. *ACM Computing Surveys*, 33(4), 470-516.

John, B.E. (1990). Extensions of GOMS analyses to expert performance requiring perception of dynamic visual and auditory information. *Human Factors in Computer systems, CHI '90*, 107-115.

John, B.E. & Kieras, D.E. (1996a). The GOMS family of user interface analysis techniques: comparison and contrast. *ACM Transactions on Computer-Human Interaction,* 3(4), 320-351.

John, B.E. & Kieras, D.E. (1996b). Using GOMS for user interface design and evaluation: which technique? *ACM Transactions on Computer-Human Interaction*, 3(4), 287-319.

John, B.E., Prevas, K., Salvucci, D. D. & Koedinger, K. (2004). Predictive human performance modeling made easy. *Proceedings of the CHI '2004 conference*, 6(1), 455-462.

Kasik, D.J. & George, H.G. (1996). Toward automatic generation of novice user test scripts. *Proceedings of the Conference on Human Factors in Computing Systems*, Volume 1 (Vancouver, Canada, April), 244-251, [En ligne]. http://www.acm.org/sigchi/chi96/proceedings/papers/Kasik/djk_txt.htm (Consulté le 20 juin 2005).

Kieras, D.E. & Polson, P.G. (1985). An approach to the formal analysis of user complexity. *International Journal of Man-Machine Studies*, 22(4), 365-394.

Kieras, D.E. & Polson, P.G. (1988). Towards a practical GOMS model methodology for user interface design. *Handbook of Human-Computer Interaction*, Elsevier science publishers, Amsterdam, 135-157.

Kieras, D. (1993). *Using the keystroke-level model to estimate execution times.* Unpublished Report, University of Michigan, [En ligne]. http://www.pitt.edu/~cmlewis/KSM.pdf (Consulté le 3 juin 2005).

Kieras, D.E., Wood, S.D., Abotel, K. & Hornof, A. (1995). GLEAN: a computer-based tool for rapid GOMS model usability evaluation of user interface designs. *Proceedings of the Eighth ACM Symposium on User Interface Software and Technology* (Pittsburgh, PA, November), 91-100.

Kieras, D.E. (1996). A guide to GOMS model usability evaluation using NGOMSL. *Handbook of Human-Computer Interaction*, 2nd edition. Amsterdam: North-Holland, 733-766.

Kieras, D. E., Wood, S.D. & Meyer, D.E. (1997). Predictive engineering models based on the EPIC architecture for a multimodal high-performance human-computer interaction task. *ACM Transactions on Computer-Human Interaction*, 4(3), 230-275.

Kirakowski, J. & Corbett, M. (1993). The software usability measurement inventory. *British Journal of Educational Technology*, 24(3), 210-212.

Kirakowski, J. & Claridge, N. (1998). Human centered measures of success in web site design. *Proceedings of the 4th Conference on Human Factors & the Web* (Baskin Ridge, NJ, June), [En ligne]. http://www.research.att.com/conf/hfweb/proceedings/kirakowski/index.html (Consulté le 3 juin 2005).

Laird, J.E. & Rosenbloom, P. (1996). The evolution of the Soar cognitive architecture. *Mind Matters: A Tribute to Allen Newell*, 1-50. Mahwah, NJ: Lawrence Erlbaum Associates, Inc.

May, J. & Barnard, P.J. (1995). The case for supportive evaluation during design. *Interacting with Computers*, 7(2), 115-143.

Moran, T.P. (1981). The command language grammar: a representation for the user interface of interactive computer systems. *International Journal of Man-Machine Studies*, 15(1), 3-50.

Myers, B.A., Hudson, S.E. & Pausch, R. (2000). Past, present, and future of user interface software tools. *ACM Transactions on Computer-Human Interaction*, 7(1), 3-28.

NF EN ISO 9241-11 (1998). Exigences ergonomiques pour travail de bureau avec terminaux à écrans de visualisation (TEV): lignes directrices concernant l'utilisabilité.

Nilsen, E., Jong, H., Olson, J.S., Biolsi, K., Rueter, H. & Mutter, S. (1993). The growth of software skill: a longitudinal look at learning & performance. *Proceedings of the SIGCHI '93 conference*, 149-156.

Olson, J.R. & Olson, G. M. (1990). The growth of cognitive modelling in human-computer interaction since GOMS. *Human-Computer Interaction*, 5, 221-265.

Payne, S. J. & Green, T.R.G. (1989). The structure of command languages: an experiment on task-action grammar. *International Journal of Man-Machine Studies*, 30(2), 213-234.

Polk, T.A. & Rosenbloom, P.S. (1994). Task-independent constraints on a unified theory of cognition. *Handbook of Neuropsychology,* Volume 9. Amsterdam, The Netherlands: Elsevier Science Publishers.

Rauterberg, M. & Aeppli, R. (1995). Learning in man-machine systems: the measurement of behavioural and cognitive complexity. Proceeding of the IEEE Conference on Systems, Man and Cybernetics (Vancouver, BC), 4685-4690.

Rosenbloom, P.S., Laird, J.E., Newell, A. & McCarl R. (1991). A preliminary analysis of the Soar architecture as a basis for general intelligence. *Artificial Intelligence*, 47, 289-325.

Rubin, J. (1994). *Handbook of usability testing*. New York: J. Wiley and Sons.

Santos, P.J. & Badre, A.N. (1994). Automatic chunk detection in human-computer interaction. *Proceedings of the workshop on advanced visual interfaces* (Bari, Italy), 69-77.

Shiffrin, R. M. & Schneider, W. (1977). Controlled and automatic information processing: II. Perception, learning, automatic attending and a general theory. *Psychological Review*, 84, 127-190.

Shneiderman, B. & Plaisant, C. (2004). *Designing the user interface: strategies for effective human-computer interactions*, 4th edition. Pearson/Addison Wesley, Inc.

Steriadis, C.E. & Constantinou, P. (2003). Designing human-computer interfaces for quadriplegic people. *ACM Transactions on Computer-Human Interaction*, 10(2), 87-118.

Tec-Ed, Inc. (1999). *Assessing WEB site usability from server log files*. White paper, [En ligne]. http://www.teced.com/PDFs/whitepap.pdf (Consulté le 20 juillet 2005).

Van Veenendaal, E.P.W.M. (1998). Questionnaire based usability testing. *Proceedings of the European Software Quality Week conference.*

Website Analysis and Measurement Inventory, [En ligne]. http://www.wammi.com (Consulté le 3 juin 2005).

Williams, K.E. (1993). Automating the cognitive task modeling process: an extension to GOMS for HCI. *Proceedings of the Conference on Human Factors in Computing* Systems, Volume 3 (Amsterdam, The Netherlands, April), 182.

Young, R.M., Green, T.R.G. & Simon, T. (1989). Programmable user models for predictive evaluation of interface designs. *Proceedings of the Conference on Human Factors in Computing Systems* (Austin, TX, April), 15-19.

Zachary, W., Mentec, J.-C. L. & Ryder, J. (1996). Interface agents in complex systems. *Human Interaction With Complex Systems: Conceptual Principles and Design Practice.* Dordrecht, The Netherlands: Kluwer Academic Publishers, 35-52.