Interface homme-ordinateur : conception et réalisation

Joëlle Coutaz

THESE

Présentée à

L'Université Joseph Fourier

pour obtenir le grade de
docteur ès sciences mathématiques

par

Joëlle COUTAZ

Interface Homme-Ordinateur : Conception et Réalisation

Thèse soutenue le 22 Décembre 1988 devant la commission d'examen.

Y. CHIARAMELLA	Président
P. BAUDELAIRE	
G. KAHN	
S. KRAKOWIAK	Examinateurs
A. MICHARD	
J. MOSSIERE	

UNIVERSITE Joseph FOURIER (GRENOBLE I)

Président de l'Université :
M. PAYAN Jean Jacques

Année Universitaire 1987 - 1988

MEMBRES DU CORPS ENSEIGNANT DE SCIENCES ET DE GEOGRAPHIE

PROFESSEURS DE 1ère Classe

ARNAUD Paul	Chimie Organique
ARVIEU ROBERT	Physique Nucléaire I.S.N.
AUBERT Guy	Physique C.N.R.S
AURIAULT Jean-Louis	Mécanique
AYANT Yves	Physique Approfondie
BARBIER Marie-Jeanne	Electrochimie
BARJON Robert	Physique Nucléaire ISN
BARNOUD Fernand	Biochimie Macromoléculaire Végétale
BARRA Jean-René	Statistiques-Mathématiques Appliquées
BECKER Pierre	Physique
BEGUIN Claude	Chimie Organique
BELORISKY Elie	Physique
BENZAKEN Claude	Mathématiques Pures
BERARD Pierre	Mathématiques Pures
BERNARD Alain	Mathématiques Pures
BERTRANDIAS Françoise	Mathématiques Pures
BERTRANDIAS Jean-Paul	Mathématiques Pures
BILLET Jean	Géographie
BOELHER Jean-Paul	Mécanique
BONNIER Jane Marie	Chimie Générale
BOUCHEZ Robert	Physique Nucléaire ISN
BRAVARD Yves	Géographie
CARLIER Georges	Biologie Végétale
CAUQUIS Georges	Chimie Organique
CHARDON Michel	Géographie
CHIBON Pierre	Biologie Animale
COHEN ADDAD Jean-Pierre	Physique
COLIN DE VERDIERE Yves	Mathématiques Pures
CYROT Michel	Physique du Solide
DEBELMAS Jacques	Géologie Générale
DEGRANGE Charles	Zoologie
DEMAILLY Jean-Pierre	Mathématiques Pures
DENEUVILLE Alain	Physique
DEPORTES Charles	Chimie Minérale
DOLIQUE Jean-Michel	Physique des Plasmas
DOUCE Roland	Physiologie Végétale
DUCROS Pierre	Cristallographie
FONTAINE Jean-Marc	Mathématiques Pures
GAGNAIRE Didier	Chimie Physique
GERMAIN Jean-Pierre	Mécanique,
GIRAUD Pierre	Géologie
HICTER Pierre	Chimie
IDELMAN Simon	Physiologie Animale
JANIN Bernard	Géographie
JOLY Jean-René	Mathématiques Pures
KAHANE André, détaché	Physique
KAHANE Josette	Physique
KRAKOWIAK Sacha	Mathématiques Appliquées

LAJZEROWICZ Jeanine	Physique
LAJZEROWICZ Joseph	Physique
LAURENT Pierre-Jean	Mathématiques Appliquées
LEBRETON Alain	Mathématiques Appliquées
DE LEIRIS Joël	Biologie
LHOMME Jean	Chimie
LLIBOUTRY Louis	Géophysique
LOISEAUX Jean-Marie	Sciences Nucléaires I.S.N.
LUNA Domingo	Mathématiques Pures
MACHE Régis	Physiologie Végétale
MASCLE Georges	Géologie
MAYNARD Roger	Physique du Solide
OMONT Alain	Astrophysique
OZENDA Paul	Botanique (Biologie Végétale)
PAYAN Jean-Jacques	Mathématiques Pures
PEBAY-PEYROULA Jean-Claude	Physique
PERRIER Guy	Géophysique
PIERRARD Jean-Marie	Mécanique
PIERRE Jean-Louis	Chimie Organique
RENARD Michel	Thermodynamique
RINAUDO Marguerite	Chimie CERMAV
ROSSI André	Biologie
SAXOD Raymond	Biologie Animale
SENGEL Philippe	Biologie Animale
SERGERAERT Francis	Mathématiques Pures
SOUCHIER Bernard	Biologie
SOUTIF Michel	Physique
STUTZ Pierre	Mécanique
TRILLING Laurent	Mathématiques Appliquées
VALENTIN Jacques	Physique Nucléaire I.S.N.
VAN CUTSEM Bernard	Mathématiques Appliquées
VIALON Pierre	Géologie

PROFESSEURS de 2ème Classe

ADIBA Michel	Mathématiques Pures
ANTOINE Pierre	Géologie
ARMAND Gilbert	Géographie
BARET Paul	Chimie
BLANCHI J.Pierre	STAPS
BLUM Jacques	Mathématiques Appliquées
BOITET Christian	Mathématiques Appliquées
BORNAREL Jean	Physique
BRUANDET J.François	Physique
BRUGAL Gérard	Biologie
BRUN Gilbert	Biologie
CASTAING Bernard	Physique
CERFF Rudiger	Biologie
CHIARAMELLA Yves	Mathématiques Appliquées
COURT Jean	Chimie
DUFRESNOY Alain	Mathématiques Pures
GASPARD François	Physique
GAUTRON René	Chimie
GENIES Eugène	Chimie
GIDON Maurice	Géologie
GIGNOUX Claude	Sciences Nucléaires
GILLARD Roland	Mathématiques Pures
GIORNI Alain	Sciences Nucléaires
GONZALEZ SPRINBERG Gérardo	Mathématiques Pures
GUIGO Maryse	Géographie
GUMUCHAIN Hervé	Géographie
GUITTON Jacques	Chimie

HACQUES Gérard	Mathématiques Appliquées
HERBIN Jacky	Géographie
HERAULT Jeanny	Physique
JARDON Pierre	Chimie
JOSELEAU Jean-Paul	Biochimie
KERCKHOVE Claude	Géologie
LONGEQUEUE Nicole	Sciences Nucléaires I.S.N.
LUCAS Robert	Physique
MANDARON Paul	Biologie
MARTINEZ Francis	Mathématiques Appliquées
NEMOZ Alain	Thermodymamique CNRS - CRTBT
OUDET Bruno	Mathématiques Appliquées
PECHER Arnaud	Géologie
PELMONT Jean	Biochimie
PERRIN Claude	Sciences Nucléaires I.S.N.
PFISTER Jean-Claude	Physique du Solide
PIBOULE Michel	Géologie
RAYNAUD Hervé	Mathématiques Appliquées
RICHARD Jean-Marc	Physique
RIEDTMANN Christine	Mathématiques Pures
ROBERT Gilles	Mathématiques Pures
ROBERT Jean-Bernard	Chimie Physique
SARROT-REYNAULD Jean	Géologie
SAYETAT Françoise	Physique
SERVE Denis	Chimie
STOECKEL Frédéric	Physique
SCHOLL Pierre-Claude	Mathématiques Appliquées
SUBRA Robert	Chimie
VALLADE Marcel	Physique
VIDAL Michel	Chimie Organique
VIVIAN Robert	Géographie
VOTTERO Philippe	Chimie

MEMBRES DU CORPS ENSEIGNANT DE L' IUT 1

PROFESSEURS de 1ère Classe

BUISSON Roger	Physique IUT 1
DODU Jacques	Mécanique Appliquée IUT 1
NEGRE Robert	Génie Civil IUT 1
NOUGARET Marcel	Automatique IUT 1
PERARD Jacques	EEA. IUT 1

PROFESSEURS de 2ème classe

BOUTHINON Michel	EEA. IUT 1
CHAMBON René	Génie Mécanique IUT 1
CHEHIKIAN Alain	EEA. IUT 1
CHENAVAS Jean	Physique IUT 1
CHOUTEAU Gérard	Physique IUT 1
CONTE René	Physique IUT 1
GOSSE Jean-Pierre	EEA.IUT 1
GROS Yves	Physique IUT 1
KUHN Gérard, (Détaché)	Physique IUT 1
MAZUER Jean	Physique IUT 1
MICHOULIER Jean	Physique IUT 1
MONLLOR Christian	EEA.IUT 1
PEFFEN René	Métallurgie IUT 1
PERRAUD Robert	Chimie IUT 1
PIERRE Gérard	Chimie IUT 1
TERRIEZ Jean-Michel	Génie Mécanique IUT 1
TOUZAIN Philippe	Chimie IUT 1
VINCENDON Marc	Chimie IUT 1

PROFESSEURS DE PHARMACIE

AGNIUS-DELORD Claudine	Physique	Faculté La Tronche
ALARY Josette	Chimie Analytique	Faculté La Tronche
BERIEL Hélène	Physiologie et Pharmacologie	Faculté La Tronche
CUSSAC Max	Chimie Therapeutique	Faculté La Tronche
DEMENGE Pierre	Pharmacodynamie	Faculté La Tronche
FAVIER Alain	Biochimie	C.H.R.G.
JEANNIN Charles	Pharmacie Galénique	Faculté Meylan
LATURAZE Jean	Biochimie	Faculté La Tronche
LUU DUC Cuong	Chimie Générale	Faculté La Tronche
MARIOTTE Anne-Marie	Pharmacognosie	Faculté La Tronche
MARZIN Daniel	Toxicologie	Faculté Meylan
RENAUDET Jacqueline	Bactériologie	Faculté La Tronche
ROCHAT Jacques	Hygiéne et Hydrologie	Faculté La Tronche
SEIGLE-MURANDI Françoise	Botanique et Cryptogamie	Faculté Meylan
VERAIN Alice	Pharmacie Galénique	Faculté Meylan

MEMBRES DU CORPS ENSEIGNANT DE MEDECINE

PROFESSEURS CLASSE EXEPTIONNELLE ET 1ère CLASSE

AMBLARD Pierre	Dermatologie	C.H.R.G.
AMBROISE-THOMAS Pierre	Parasitologie	C.H.R.G.
BEAUDOING André	Pédiatrie-Puericulture	C.H.R.G.
BEZEZ Henri	Orthopédie-Traumatologie	Hopital SUD
BONNET Jean-Louis	Ophtalmologie	C.H.R.G.
BOUCHET Yves	Anatomie	Faculté La Merci
	Chirurgie Générale et Digestive	C.H.R.G.
BUTEL Jean	Orthopédie-Traumatologie	C.H.R.G.
CHAMBAZ Edmond	Biochimie	C.H.R.G.
CHAMPETIER Jean	Anatomie-Topographique et Appliquée	C.H.R.G.
CHARACHON Robert	O.R.L.	C.H.R.G.
COLOMB Maurice	Immunologie	C.H.R.G.
COUDERC Pierre	Anatomie-Pathologique	Hopital sud
DELORMAS Pierre	Pneumophtisiologie	C.H.R.G.
DENIS Bernard	Cardiologie	C.H.R.G.
GAVEND Michel	Pharmacologie	C.H.R.G.
HOLLARD Daniel	Hématologie	Faculté La Merci
LATREILLE René	Chirurgie Thoracique et Cardiovasculaire	C.H.R.G.
LE NOC Pierre	Bactériologie-Virologie	C.H.R.G.
MALINAS Yves	Gynécologie et Obstétrique	C.H.R.G.
MALLION Jean-Michel	Médecine du Travail	C.H.R.G.
MICOUD Max	Clinique Médicale et Maladies Infectieuses	C.H.R.G.
MOURIQUAND Claude	Histologie	Faculté La Merci
PARAMELLE Bernard	Pneumologie	C.H.R.G.
PERRET Jean	Neurologie	C.H.R.G.
RACHAIL Michel	Hépato-Gastro-Entérologie	C.H.R.G.
DE ROUGEMONT Jacques	Neurochirurgie	C.H.R.G.
SARRAZIN Roger	Clinique Chirurgicale	C.H.R.G.
STIEGLITZ Paul	Anestésiologie	C.H.R.G.
TANCHE Maurice	Physiologie	Faculté La Merci
VIGNAIS Pierre	Biochimie	Faculté La Merci

PROFESSEURS 2ème CLASSE

BACHELOT Yvan	Endocrinologie	C.H.R.G.
BARGE Michel	Neurochirurgie	C.H.R.G.
BENABID Alim Louis	Biophysique	Faculté La Merci
BENSA Jean-Claude	Immunologie	Hopital Sud
BERNARD Pierre	Gynécologie-Obstétrique	C.H.R.G.
BESSARD Germain	Pharmacologie	ABIDJAN
BOLLA Michel	Radiothérapie	C.H.R.G.
BOST Michel	Pédiatrie	C.H.R.G.
BOUCHARLAT Jacques	Psychiatrie Adultes	Hopital Sud
BRAMBILLA Christian	Pneumologie	C.H.R.G.
CHIROSSEL Jean-Paul	Anatomie-Neurochirurgie	C.H.R.G.
COMET Michel	Biophysique	Faculté La Merci
CONTAMIN Charles	Chirurgie Thoracique et Cardiovasculaire	C.H.R.G.
CORDONNIER Daniel	Néphrologie	C.H.R.G.
COULOMB Max	Radiologie	C.H.R.G.
CROUZET Guy	Radiologie	C.H.R.G.
DEBRU Jean-Luc	Médecine Interne et Toxicologie	C.H.R.G.
DEMONGEOT Jacques	Biostatistiques et Informatique Médicale	Faculté La Merci
DUPRE Alain	Chirurgie Générale	C.H.R.G.
DYON Jean-François	Chirurgie Infantile	C.H.R.G.
ETERRADOSSI Jacqueline	Physiologie	Faculté La Merci
FAURE Claude	Anatomie et Organogénèse	C.H.R.G.
FAURE Gilbert	Urologie	C.H.R.G.
FOURNET Jacques	Hépato-Gastro-Entérologie	C.H.R.G.
FRANCO Alain	Médecine Interne	C.H.R.G.
GIRARDET Pierre	Anesthésiologie	C.H.R.G.
GUIDICELLI Henri	Chirurgie Générale et Vasculaire	C.H.R.G.
GUIGNIER Michel	Thérapeutique et Réanimation Médicale	C.H.R.G.
HADJIAN Arthur	Biochimie	Faculté La Merci
HALIMI Serge	Endocrinologie et Maladies Métaboliques	C.H.R.G.
HOSTEIN Jean	Hépato-Gastro-Entérologie	C.H.R.G.
HUGONOT Robert	Médecine Interne	C.H.R.G.
JALBERT Pierre	Histologie-Cytogénétique	C.H.R.G.
JUNIEN-LAVILLAUROY Claude	O.R.L.	C.H.R.G.
KOLODIE Lucien	Hématologie Biologique	C.H.R.G.
LETOUBLON Christian	Chirurgie Générale	C.H.R.G.
MACHECOURT Jacques	Cardiologie et Maladies Vasculaires	C.H.R.G.
MAGNIN Robert	Hygiène	C.H.R.G.
MASSOT Christian	Médecine Interne	C.H.R.G.
MOUILLON Michel	Ophtalmologie	C.H.R.G.
PELLAT Jacques	Neurologie	C.H.R.G.
PHELIP Xavier	Rhumatologie	C.H.R.G.
RACINET Claude	Gynécologie-Obstétrique	Hopital Sud
RAMBAUD Pierre	Pédiatrie	C.H.R.G.
RAPHAEL Bernard	Stomatologie	C.H.R.G.
SCHAERER René	Cancérologie	C.H.R.G.
SEIGNEURIN Jean-Marie	Bactériologie-Virologie	Faculté La Merci
SELE Bernard	Cytogénétique	Faculté La Merci
SOTTO Jean-Jacques	Hématologie	C.H.R.G.
STOEBNER Pierre	Anatomie Pathologique	C.H.R.G.
VROUSOS Constantin	Radiothérapie	C.H.R.G.

Je tiens à remercier ici l'ensemble des personnes qui, par leurs conseils, leurs encouragements et leur participation, ont contribué à l'aboutissement de ce travail :

Yves Chiaramella, Professeur à l'Université Joseph Fourier et Directeur du Laboratoire de Génie Informatique où s'est effectuée cette étude, qui me fait l'honneur de présider le jury,

Gilles Kahn, Directeur de Recherche à l'I.N.R.I.A., qui a accepté d'être le rapporteur de cette thèse et qui m'a permis de l'améliorer,

Patrick Baudelaire, Directeur du Centre de Recherche de Paris de Digital Equipment, qui a bien voulu évaluer cette étude et qui a effectué une lecture scrupuleuse du manuscrit,

Sacha Krakowiak, Professeur à l'Université Joseph Fourier de Grenoble, qui suit mon travail depuis de nombreuses années et qui a accepté de le diriger,

Alain Michard, Directeur de Recherche à l'I.N.R.I.A., qui a manifesté son intérêt pour ce travail donnant ainsi l'exemple d'un rapprochement possible entre ergonomes et informaticiens,

Jacques Mossière, Directeur de l'E.N.S.I.M.A.G., qui a constamment soutenu ce travail de ses encouragements cordiaux,

Bernard Senach, chercheur en ergonomie cognitive à l'INRIA, qui a bien voulu lire la première section de la thèse,

Les étudiants du D.E.S.S. de Génie informatique de l'Université Joseph Fourier et de l'E.N.S.I.M.A.G. sans qui une partie de ce travail n'aurait pas existé :
Jean-Claude Alletru, Frédéric Fournier, Olivier Roussel et Agnès Vial pour le projet IRENE,
Christian Doaré, Dominique Solomon, Vafa Sabete et Anne Valette pour le projet ELOISE,
Sophie Bernard, Laurent Lys, Christophe Po et Philippe Marchal pour le projet MOUSE,
Christophe Geiben et Paul-André Tourtier pour le projet CONCEPT,

Muriel Chandelon, Geneviève Warnant et Jean-Claude Ruche, étudiants de Maîtrise de l'Université de Namur (Belgique) pour leur participation à l'application Thermo,

L'ensemble des participants au projet GUIDE qui m'ont exprimé leur soutien amical,

Françoise Renzetti et l'équipe de la Médiathèque qui m'ont apporté l'information recherchée avec efficacité et gentillesse et qui ont souvent devancé mes besoins,

Daniel Iglésias et l'équipe du service de reprographie qui m'ont toujours accordé leur précieux concours.

Une pensée toute particulière pour ma petite famille qui a participé quotidiennement à l'évolution de mon travail et spécialement pour Jim, à l'origine de cette étude, et qui a partagé la responsabilité de l'encadrement des projets ELOISE et IRENE, qui a été aussi le premier utilisateur d'APEX.

Un dernier remerciement à mon Macintosh, le supporter matériel de tous les instants.

A ma petite famille.

Introduction

Le sujet
Interface et interaction
Les lacunes de la situation
L'objet et l'organisation de la thèse

Section I : L'apport des sciences cognitives

Introduction
1. Le Processeur Humain
2. GOMS et KEYSTROKE
3. Modélisation de l'Action
4. Principes pratiques de l'ergonomie
5. Pour une méthode de conception

Section II : Architecture Logicielle des Systèmes Interactifs

Introduction
6. Les Composants Fonctionnels
7. Les Modèles d'Architecture
8. Le Modèle PAC
9. Les Applications du modèle PAC

Section III : Outils pour la Construction des Systèmes Interacifs

Introduction
10. Boîtes à outils
11. Machines à Images Abstraites
12. Squelettes d'Application
13. Générateurs d'Interfaces Interactives

Conclusion

Contribution de la thèse
Perspectives

Introduction

Le sujet : l'Interaction Homme-Ordinateur

Ce travail concerne l'interaction homme-ordinateur, un domaine en pleine évolution, à la confluence de deux disciplines : les sciences cognitives et les techniques informatiques. Pour les ergonomes et les psychologues, l'interaction homme-ordinateur désigne l'ensemble des phénomènes physiques et cognitifs qui participent à la réalisation de tâches informatisées. Chez les informaticiens, concernés par la technologie de l'interaction, on parle d'interface homme-ordinateur ou encore, par abus de langage, d'interface homme-machine. Une telle interface est un assemblage de composants logiciels et matériels qui permet l'accomplissement de tâches avec le concours d'un ordinateur.

Nous observons ici la cohabitation de deux vues distinctes de l'interaction homme-ordinateur, l'une animée par les aspects psychologiques de la communication, l'autre dirigée par les composantes techniques de la réalisation. Jusqu'à ces dernières années, les deux approches ont évolué sur des chemins parallèles conduisant chacune à des enseignements intéressants mais souvent sans caractère unitaire et sans transfert de savoir-faire. L'objet de cette thèse est un premier pas en direction d'un rapprochement entre les deux points de vue car tous deux trouvent un terrain d'analyse complémentaire dans l'étude des interfaces et de l'interaction.

Interface et interaction

Au sens large, une interface est un dispositif qui sert de limite commune à plusieurs entités communicantes. Chaque entité s'exprime dans un langage spécifique : signal électrique, force mécanique, langue artificielle ou naturelle, etc. Pour que la communication soit possible, le dispositif doit assurer à la fois la connexion physique entre les entités et effectuer des opérations de traduction entre les formalismes. Dans le cas de l'interface homme-ordinateur, la connexion a lieu entre l'image du système (c'est-à-dire sa manifestation externe) et les organes sensorimoteurs de l'utilisateur ; la traduction s'effectue entre les formalismes du système et ceux de l'utilisateur. Du côté système, une

traduction est assurée entre les représentations internes adaptées aux traitements du problème et la représentation externe qui participe à la définition de l'image. Une opération analogue se produit du côté utilisateur entre la représentation mentale de la situation perçue ou à atteindre et les actions physiques à entreprendre. Une fois la communication établie, l'interaction peut avoir lieu.

La réalisation d'une interface suppose donc la connaissance précise du comportement de chacune des entités à relier. Lorsque ces entités sont des objets artificiels, le comportement et les formalismes sont généralement bien maîtrisés ; lorsque l'une des entités est le sujet humain, la définition de l'interface devient une tâche complexe, souvent teintée d'empirisme et d'arbitraire. Il y a trois raisons à cela : la puissance fonctionnelle des systèmes informatiques, le peu d'intérêt des informaticiens pour les sciences cognitives et l'absence de modèles psychologiques adaptés à l'interaction homme-ordinateur.

Les lacunes de la situation

En raison de son potentiel fonctionnel, l'ordinateur peut s'entrevoir, non pas comme un simple outil, mais comme un collaborateur. Un outil est un instrument sans pouvoir décisionnel. Il est conçu pour être manipulé. A l'inverse, un collaborateur participe activement à la réalisation de l'œuvre commune. Il est conçu pour aider à la résolution d'un problème. L'efficacité du soutien dépend étroitement de la connaissance des facultés et des mécanismes de résolution du partenaire. Dans ces conditions, le concepteur d'un système interactif doit élaborer une description aussi précise que possible du problème et des processus cognitifs de l'utilisateur. Il lui faut ensuite concrétiser aussi fidèlement que possible cette représentation dans le logiciel. Si la description est "correcte" et si la transcription logicielle conserve cette "correction", le système peut être l'extension électronique des facultés intellectuelles de l'utilisateur tout comme la machine-outil est devenue l'extension mécanique des facultés musculaires et motrices de l'homme.

L'ensemble "système informatique-utilisateur" fonctionne comme des vases communicants : si l'utilisateur ne va pas vers le système, le système parcourt la branche de chemin qui conduit à la collaboration ; à l'inverse, si le système ne va pas jusqu'à l'utilisateur, l'utilisateur doit faire l'effort d'adaptation complémentaire. Lorsque l'effort demandé dépasse les capacités ou les motivations, l'interaction s'achève en situation d'échec. L'objectif du concepteur est donc de déterminer les limites du chemin mental de l'utilisateur. Cette détermination fait précisément appel au domaine des sciences cognitives.

Par ignorance, par réticence, ou par contrainte conjoncturelle et technologique, la conception des systèmes se plie trop souvent aux exigences de l'informatique avant de répondre à celles de l'utilisateur. L'informaticien s'applique avant tout à définir les fonctions logicielles d'un système sans vraiment se préoccuper des besoins et des limites cognitives de l'utilisateur. Une autre composante de cette approche erronnée est la croyance encore trop répandue qu'un habillage racoleur suffit à rendre le système "facile à apprendre" et "facile à utiliser". Si les fonctions d'un système ne sont pas de nature à compléter les facultés de l'utilisateur, si leur organisation ne correspond pas à la structure mentale de résolution, aucun effet de présentation ne pourra durablement maquiller les carences de fond. A la décharge de l'informaticien, les enseignements des sciences cognitives ne permettent pas, à l'heure actuelle, de déterminer scientifiquement les caractéristiques cognitives de l'utilisateur.

Les sciences cognitives proposent des théories séduisantes mais trop restreintes pour envisager de modéliser l'ensemble des processus psychologiques de l'interaction homme-machine ; elles comprennent des modèles quantitatifs formels mais trop réducteurs pour être applicables au cas complexe de l'interaction ; il existe des modèles qualitatifs généraux mais trop informels pour guider les choix sur une voie scientifiquement sûre ; on dispose d'une multitude de recommandations pratiques qui conduisent parfois à des situations contradictoires. Les hypothèses et les démonstrations expérimentales prolifèrent mais aucune méthode de conception ne s'adresse véritablement à l'informaticien. De cette diversité, il est possible néanmoins de retenir des enseignements formateurs et constructifs. C'est ce que j'ai tenté d'identifier dans le travail ici présenté.

L'objet et l'organisation de la thèse

L'un des objets de la thèse est de sensibiliser les informaticiens à la nécessité d'ouvrir les logiciels aux concepts de l'ergonomie cognitive. La mise en pratique de cette ouverture ne relève pas d'une méthode éprouvée mais d'une compétence. La thèse présente de manière informelle une part de ce savoir-faire. Elle comprend trois sections. La première organise la diversité des connaissances du domaine des sciences cognitives en une synthèse de points de référence simples et utiles à la conception des systèmes interactifs ; les sections restantes concernent les aspects logiciels de l'interaction. Elles montrent en particulier comment construire des systèmes capables de répondre aux principes de l'ergonomie.

La section I reflète les deux tendances du domaine des sciences cognitives en présentant trois théories influentes et un condensé de principes pratiques.

- Le modèle du Processeur Humain, exposé au chapitre 1, définit un cadre fédérateur à la diversité des connaissances en psychologie et utilise une terminologie adaptée à l'informaticien.
- Le chapitre 2 présente les modèles GOMS et Keystroke, deux applications directes du modèle du Processeur Humain résolument orientées vers les performances.
- Le chapitre 3 complète l'approche behavioriste de GOMS et de Keystroke avec le point de vue cognitiviste de la théorie de l'action. Cette dernière est un modèle explicatif du comportement utile à la mise en évidence des points clés de l'interaction.
- L'approche théorique trouve son complément au chapitre 4 avec l'orientation pratique des principes ergonomiques. Ceux-ci sont innombrables et difficiles à embrasser dans leur totalité. Dans cette présentation, l'objectif est de retenir les éléments les plus marquants mais aussi les plus simples à mettre en œuvre par un informaticien.
- Cette première section conclut au chapitre 5 sur la nécessité d'un schéma directeur de méthode de conception.

Après avoir identifié l'apport des sciences cognitives à la conception des systèmes interactifs, ce travail présente les aspects logiciels de l'interaction. La section II traite de l'organisation des abstractions logicielles impliquées dans la réalisation des systèmes interactifs. Sur ce thème, la litérature est parcimonieuse et confuse.

- Le chapitre 6 identifie les fonctions essentielles d'un système interactif et précise la terminologie.
- Le chapitre 7 est une clarification des connaissances relatives aux modèles d'architecture. Il fait observer que ces modèles ne permettent pas toujours de satisfaire aux exigences psycho-cognitives de l'interaction.
- Le modèle PAC, au centre de ma contribution au domaine, vise à combler les lacunes des propositions précédentes. Il fait l'objet du chapitre 8.
- Au chapitre 9, la description de trois applications réalisées selon ce modèle en démontre l'intérêt.

La section III présente les outils utiles à la construction des systèmes interactifs : les boîtes à outils, les techniques d'affichage, les squelettes d'application et les générateurs d'interfaces.

- Le chapitre 10 propose une classification et une étude comparative des boîtes à outils actuelles. Il en identifie les avantages et les limitations en examinant leurs effets sur la qualité ergonomique du logiciel et en considérant le point de vue du réalisateur.

- Le chapitre 11 concerne les problèmes de l'affichage auxquels je me suis intéressée en développant la technique des boîtes.

- Le chapitre 12 est consacré aux squelettes d'application qui viennent utilement combler les lacunes des boîtes à outils. La technique des squelettes est un exemple de réutilisation de logiciel. Elle est récente et encore mal maîtrisée. L'objet de ce chapitre est d'en dégager les principes directeurs. APEX, réalisé selon le modèle PAC, illustre leur mise en application.

- Le dernier chapitre développe les aspects essentiels de la génération automatique des interfaces. Il en indique les éléments prometteurs et les lacunes.

La conclusion dresse un bilan des retombées de ce travail et présente une voie de collaboration entre ergonomes et informaticiens. L'ensemble de cette étude débouche sur deux résultats utiles :

- une synthèse pédagogique qui intègre les aspects essentiels de l'interaction homme-ordinateur et qui montre comment les sciences cognitives peuvent utilement s'insérer dans les techniques informatiques ;

- des réponses à quelques-uns des problèmes fondamentaux de la réalisation des systèmes interactifs, en particulier l'architecture et la réutilisation de logiciels d'interaction.

Les prolongements de ce travail pourraient se situer dans la perspective à long terme d'un système d'aide à la conception et à la réalisation d'interfaces interactives qui devrait véhiculer le savoir-faire de l'ergonome et de l'informaticien.

Section I

L'Apport des Sciences Cognitives

La conception des systèmes interactifs ne doit plus s'effectuer sur la seule base du jugement intuitif mais doit exploiter les connaissances dans le domaine de l'interaction homme-machine. Cette connaissance s'organise progressivement sous forme de modèles, de principes pratiques et de méthodes.

Les modèles sont de deux sortes : théoriques et appliqués. Les modèles théoriques formalisent (ou tentent de formaliser) les mécanismes qui régissent l'interaction entre un individu et l'ordinateur. Ils sont le reflet d'une théorie explicative ou prédictive du comportement cognitif de l'utilisateur. Ils ne résolvent pas les problèmes de la conception mais aident le concepteur à comprendre la structure cognitive de l'individu, à identifier ses besoins, à analyser les propriétés d'une description formelle et à contrôler différents aspects de la conception.

Les modèles appliqués, à mi-chemin entre la théorie et le concret, sont tournés vers l'évaluation prédictive des performances. Ils ne fournissent pas davantage de réponse sur le processus de conception mais, pour une conception donnée, permettent d'effectuer des comparaisons analytiques entre plusieurs voies possibles.

Les principes pratiques sont des règles et des propositions de bon sens établies le plus souvent expérimentalement. Ces principes ne constituent pas les éléments déterminants du processus de conception mais servent de guide à la réalisation des compromis.

Les méthodes de conception s'organisent selon deux tendances : les méthodes qui s'appuient sur une théorie de la communication homme-machine et les méthodes expérimentales dirigées par l'observation.

Cette section sur les sciences cognitives reflète les deux tendances : théorie et expérimentalisme. Elle est organisée en cinq chapitres.

Les chapitres 1 et 2 illustrent l'approche formelle à l'étude de l'interaction homme-ordinateur. Les modèles du Processeur Humain, GOMS et Keystroke en sont les éléments marquants.

Le chapitre 3 représente la tendance inverse. La théorie de l'Action de D. Norman décrit de manière informelle les éléments mis en jeu dans l'accomplissement d'une tâche.

Au chapitre 4, nous abandonnons le point de vue du théoricien au profit du praticien. Nous ne passons pas en revue les catalogues des principes pratiques mais nous identifions des règles générales faciles à appréhender.

Au chapitre 5, nous concluons cette première section avec l'assemblage des enseignements théoriques et pratiques au sein de méthodes de conception.

Chapitre 1

Le Processeur Humain

1. Introduction aux Eléments du Modèle

Dans leur modèle, "The Model Human Processor", S. Card, T. Moran et A. Newell représentent l'individu comme un système de traitement d'informations régi par des règles [Card 83].

Le processeur humain comprend trois sous-systèmes interdépendants : les systèmes sensoriel, moteur et cognitif. Chacun d'eux dispose d'une mémoire et d'un processeur dont les performances se caractérisent à l'aide de paramètres. Pour une mémoire, les paramètres essentiels sont :

- μ, la capacité (c.-à-d. le nombre d'éléments d'information mémorisés),
- δ, la persistance (c.-à-d. le temps au bout duquel la probabilité de retrouver un élément d'information est inférieure à 0.5),
- κ, le type d'information mémorisée (physique, symbolique, etc.).

Pour un processeur, le paramètre important est

- τ, le cycle de base qui inclut le cycle d'accès à sa mémoire locale.

Les règles qui régissent le comportement du processeur humain sont répertoriées au paragraphe 5. Certaines traduisent les caractéristiques des sous-systèmes considérés isolément, d'autres expriment leur interdépendance. Voyons dans les paragraphes 2, 3 et 4 comment chaque sous-système opère et coopère.

2. Le Système Sensoriel

Le système sensoriel désigne l'ensemble des sous-systèmes spécialisés chacun dans le traitement d'une classe de stimuli. Un stimulus est un phénomène physique détectable par un sous-système sensoriel. Chaque sous-système dispose d'une mémoire spécifique, dite mémoire sensorielle, et d'un mécanisme de traitement intégré.

Les stimuli sont codés dans la mémoire sensorielle. Ce codage exprime les propriétés physiques du phénomène. Par exemple, dans la mémoire visuelle présentée dans la figure 1.1, le codage κ_S de la lettre P traduit les courbures et les dimensions de la lettre mais n'exprime pas sa reconnaissance.

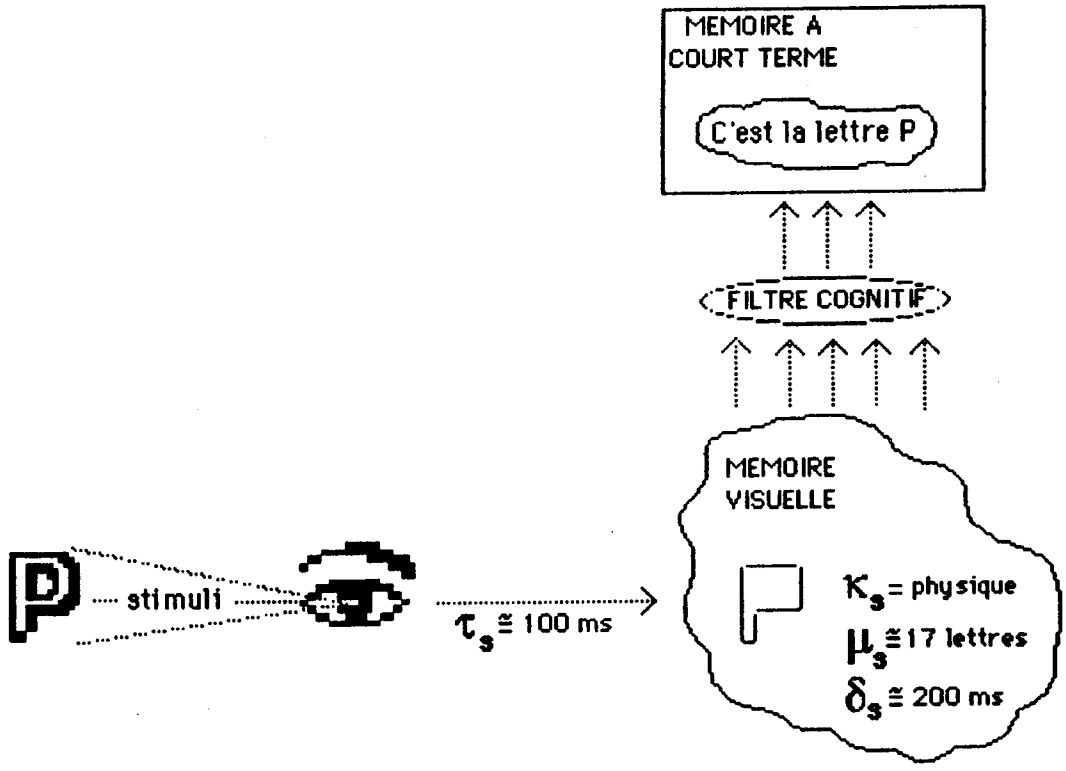


Figure 1.1 : Le système sensoriel visuel et sa relation avec le système cognitif.

Les mémoires sensorielles sont intimement liées à la mémoire à court terme du système cognitif. Ce dernier dispose d'un filtre qui détermine la nature des informations à transférer entre les mémoires sensorielles et la mémoire à court terme. Lorsque, par suite de transferts, la capacité d'absorption de la mémoire à court terme atteint la limite de saturation, les informations des mémoires sensorielles ne sont plus transmises et se dégradent. La persistance δ_S des mémoires sensorielles est de l'ordre de 200 msec pour la mémoire visuelle et légèrement supérieure, 1500 msec, pour la mémoire auditive.

Le cycle de base τ_S d'un processeur sensoriel est de l'ordre de 100 msec et varie inversement avec l'intensité du stimulus. Ceci signifie qu'il faut en moyenne 100 msec pour qu'un stimulus soit représenté dans une mémoire sensorielle (c'est-à-dire pour que l'individu ait la sensation de percevoir) et que la sensation de percevoir se manifeste plus rapidement lorsque le stimulus est intense. En conséquence, deux événements sensoriels similaires survenant dans le même cycle sont combinés en un seul mais la durée de ce cycle est sensible à l'intensité du stimulus (de 50 à 200 msec, voire largement en dehors de ces limites si les conditions sont extrêmes).

Nous verrons au paragraphe 6 consacré à l'évaluation du modèle pourquoi ces valeurs concernent directement la réalisation des systèmes interactifs.

3. Le Système Moteur

Le système moteur est responsable des mouvements. Dans le cadre de l'interaction homme-ordinateur, les mouvements d'intérêt sont les manipulations des unités physiques de commande, telles que les claviers, les écrans et les dispositifs de désignation.

Un mouvement n'est pas continu mais est constitué d'une suite de micromouvements discrets. Chaque micromouvement s'accomplit en moyenne en 70 msec. Ce temps constitue le cycle de base τ_m du processeur du système moteur.

Partant de l'hypothèse qu'un mouvement est le résultat de plusieurs micromouvements, il est possible de déterminer le temps théorique T nécessaire au placement de la main sur une cible donnée, soit :

$$T = I \log_2 2D/L \qquad (1)$$

où

- D est la distance à parcourir par la main,
- L est la largeur de la cible, et
- I une constante évaluée à 0,1 sec.

La figure 1.2 illustre la situation initiale.

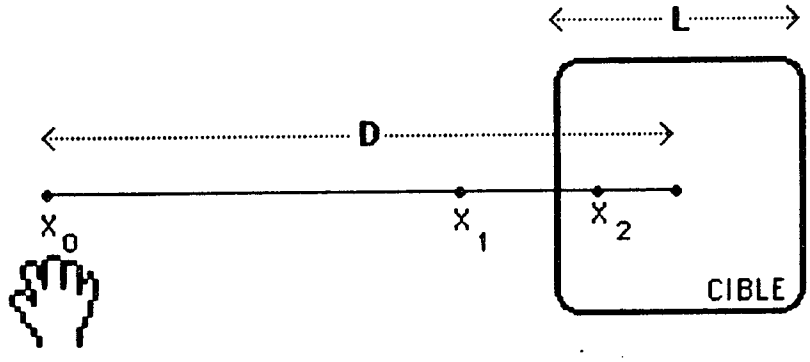


Figure 1.2 : Déplacement de la main vers une cible. X0, X1, X2 indiquent les emplacements successifs de la main au cours des micromouvements.

L'égalité (1) définit la loi de Fitts. Nous verrons son application au paragraphe 6.

4. Le Système Cognitif

La mémoire du système cognitif comprend la mémoire à court terme (appelée également mémoire de travail) et la mémoire à long terme. La mémoire à court terme détient les informations en cours de manipulation tandis que la mémoire à long terme est le lieu de stockage de la connaissance permanente. Le processeur du système cognitif contrôle le comportement de l'individu en fonction du contenu de ces mémoires.

4.1. La mémoire à court terme

La mémoire à court terme se comporte comme les registres d'un calculateur : elle contient les opérandes d'entrée et les résultats intermédiaires des traitements en cours. Les opérandes proviennent des mémoires sensorielles et/ou de la mémoire à long terme.

Les informations d'origine sensorielle sont représentées sous forme symbolique. Contrairement au codage des stimuli en mémoire sensorielle, elles ne sont plus affectées des caractéristiques physiques. Par exemple, la représentation de P dans la mémoire à court terme traduit le fait qu'il s'agit de la lettre P.

Les informations en provenance de la mémoire à long terme sont des mnèmes (en anglais, "chunks") activés par le processeur cognitif. Un mnème est une unité cognitive symbolique, une abstraction qui peut être associée à d'autres unités. Les associations et la nature d'un mnème dépendent de la tâche en cours et des connaissances de l'individu. Par exemple, la suite de lettres "S.N.C.F." constitue un mnème pour la plupart d'entre nous mais forme quatre mnèmes pour toute personne qui ignore la signification de ce sigle.

L'activation d'un mnème entraîne sa mise à disposition dans la mémoire à court terme. Cette activation se propage aux mnèmes associés ajoutant de nouveaux éléments dans la mémoire à court terme. La capacité de cette mémoire est estimée, selon la célèbre formule de Georges Miller [Miller 75], à 7 ± 2 mnèmes[1]. Lorsque la mémoire à court terme est saturée, l'activation de nouveaux mnèmes efface de la mémoire ceux qui n'ont pas fait l'objet d'une réactivation. Ce phénomène de conflit s'appelle interférence avec, pour conséquence, la dégradation de l'information au bout d'un certain temps[2].

L'interférence se manifeste dès qu'il y a partage de ressource. Il est intéressant de relever la similitude entre le modèle de fonctionnement de la mémoire à court terme selon lequel les éléments inutilisés se dégradent, et la technique usuelle des systèmes de pagination qui réquisitionnent les pages les plus anciennes et les moins référencées (c.-à-d. les moins "réactivées").

Nous verrons au paragraphe 6 les conséquences de la faible capacité de la mémoire à court terme et de sa persistence sur l'interaction homme-ordinateur.

4.2. La mémoire à long terme

La mémoire à long terme a le même rôle que les mémoires centrales et secondaires d'un calculateur : elle contient l'information de masse ; elle peut être lue ou modifiée ; son contenu est un réseau de mnèmes qui représente des procédures et des données appelées respectivement connaissance procédurale (ou savoir-faire) et connaissance factuelle (ou connaissance de faits).

---

1. En toute rigueur, deux valeurs traduisent la capacité de la mémoire à court terme :

- la capacité pure : 3[2.5 ~ 4.1] mnèmes, déterminée, par exemple, à partir du nombre de chiffres que l'on est capable de retrouver au sein d'une longue suite qui s'arrête brusquement,
- la capacité effective qui tient compte de l'utilisation de la mémoire à long terme : 7±2 mnèmes, estimée, par exemple, comme étant le nombre de mots répétés d'une liste.

2. La persistence δ_C varie inversement avec le nombre de mnèmes que l'utilisateur tente de retrouver simultanément. Card et Newell retiennent pour δ_C les moyennes suivantes :

- δ_C [1mnème] = 73 sec.
- δ_C [3 mnèmes] = 7 sec.

Une opération de lecture dans la mémoire à long terme consiste à rechercher un mnème. Le succès de cette recherche transfère le mnème dans la mémoire à court terme avec un degré d'activation donné. L'échec a deux causes principales : ou bien aucune association n'est trouvée, ou bien plusieurs mnèmes interfèrent avec le mnème cible. Dans le second cas, il y a interférence parce que plusieurs mnèmes répondent aux mêmes critères de recherche ; ils répondent aux mêmes critères de recherche parce qu'ils ont été enregistrés avec des critères d'association identiques. Les mnèmes qui interfèrent, acquis plus récemment que la cible, ont un degré d'activation supérieur ; de ce fait, ils masquent l'accès à la cible. En résumé, l'information ne disparaît pas de la mémoire à long terme. Elle est simplement inaccessible. En conséquence, on considère que la persistance des informations de la mémoire à long terme est infinie.

Les opérations d'écriture dans la mémoire à long terme s'effectuent par association. Afin d'être inscrit dans la mémoire à long terme, un mnème de la mémoire à court terme doit être associé, selon des critères définis par l'individu, à un ou plusieurs mnèmes de la mémoire à long terme. Le principe de discrimination abordé précédemment indique que les chances de retrouver ce mnème croissent avec le nombre d'associations discriminantes. L'intuition pousse à penser que ces chances augmentent également avec le temps t disponible pour effectuer ces associations. Ceci est vrai tant que $t \leq \delta_c$. Au delà de δ_c, un mnème se dégrade s'il n'est pas réactivé. Or, la mise en place d'une association active des mnèmes de la mémoire à long terme qui entrent en compétition avec le mnème initial pour la mémoire à court terme. Ces mnèmes "plus frais" ont un degré d'activation plus élevé que le mnème initial qui s'efface. δ_c définit le rythme optimal d'arrivée de nouveaux mnèmes dans la mémoire à court terme pour qu'ils soient associés à des éléments de la mémoire à long terme. La création de nouvelles associations est toujours possible. On considère donc que la capacité de la mémoire à long terme est infinie.

4.3. Le Processeur Cognitif

Le fonctionnement du processeur cognitif est calqué sur le modèle des systèmes de production. Il opère selon le cycle "Reconnaissance-Action", analogue au cycle "Recherche-Exécution" des calculateurs usuels. Dans la phase de Reconnaissance, le processeur détermine les actions de la mémoire à long terme associées aux mnèmes de la mémoire à court terme. Dans la seconde phase, l'Exécution, ces actions sont effectuées provoquant une modification du contenu de la mémoire à court terme.

Le cycle de base du processeur cognitif τ_c semble similaire aux cycles des autres processeurs :

τ_c = 70 msec.

Dans la description qui précède, nous avons énoncé quelques-uns des principes de fonctionnement du système cognitif. Nous les rappelons au paragraphe suivant.

5. Quelques Principes Opératoires

Nous distinguons deux types de principes : ceux qui reflètent la théorie du Processeur Humain et ceux qui formalisent des observations empiriques en accord avec cette théorie. Ne sont cités ici que les principes utiles à l'exposé de la thèse.

Principe du fonctionnement cyclique du processeur cognitif

Le système cognitif procède selon le cycle "Reconnaissance-Action".

Principe de Discrimination

La difficulté de retrouver une information est liée au nombre de candidats répondant aux mêmes indicateurs d'accès.

Principe de Rationalité

Pour atteindre un objectif, l'individu agit de manière rationnelle. Le comportement rationnel d'un individu résulte de l'union des ensembles : buts à atteindre, structure de la tâche à réaliser, connaissance.

Loi de Fitts

Le temps T pour placer la main sur une cible dépend uniquement de la précision requise, c'est-à-dire du rapport entre la distance et la dimension de la cible, soit :

$T = I \log_2(D/L + 0.5)$[1] où $I = -(\tau_s + \tau_c + \tau_m) / \log_2 \varepsilon$.

ε a été évalué à .07 (Card 83 p. 53), ce qui donne une valeur moyenne de I = 100 msec.

---

1. Cette formule est une variante de la loi de Fitts que nous devons à Welford (1968). Diverses expérimentations tendent à confirmer sa validité.

6. Evaluation du Modèle du Processeur Humain

Le modèle du Processeur Humain présente trois avantages : il constitue un cadre fédérateur à la diversité des connaissances en psychologie ; il utilise la terminologie de l'informaticien et, surtout, franchit un pas important en direction d'une psychologie appliquée. En revanche, le niveau d'abstraction des éléments théoriques du modèle ne correspond pas aux besoins de conception globale d'une interface homme-machine.

6.1. L'intérêt du Processeur Humain

Le modèle du Processeur Humain représente l'individu comme un système de traitement d'informations. Les concepts attachés à ces systèmes sont précisément ceux que les informaticiens manipulent quotidiennement. Pour cette raison, ce modèle constitue une introduction simple et séduisante au domaine de la psychologie cognitive.

Au delà d'une terminologie adaptée, le modèle du Processeur Humain définit un cadre fédérateur à la diversité des connaissances en psychologie. Sur cette charpente, l'informaticien peut progressivement greffer ses connaissances concernant tel ou tel aspect de l'individu. L'objectif des auteurs du modèle est plus ambitieux que d'offir un simple cadre de réflexion.

L'objectif de Newell est de créer une discipline qui allierait à la fois les caractéristiques de la science fondamentale et les caractéristiques de la science appliquée. Elle permettrait, comme la Physique, d'effectuer des évaluations approchées. A partir d'une théorie technique [Newell 86], il serait possible de construire des modèles qui répondraient aux questions que le concepteur se poserait sur des phénomènes précis de l'interaction homme-ordinateur.

Le Processeur Humain est une tentative en direction de cette théorie technique : des paramètres (capacité, persistance, cycle) formalisent les performances élémentaires des sous-systèmes ; des lois (comme celle de Fitts) expriment, par des formules mathématiques, les performances globales du processeur humain. Ces paramètres et ces lois ont été confirmés par l'expérimentation ; ils constituent des approximations réalistes formelles utiles à la prise de décision des compromis inhérents à la conception d'interface homme-ordinateur. Citons quelques exemples de questions qui trouvent leur réponse avec le modèle.

6.2. Trois Exemples d'Approximation avec le Modèle

Les trois exemples choisis concernent chacun un sous-système différent. Le premier démontre le lien entre le cycle du processeur sensoriel et le rafraîchissement de l'écran ; le second relève la dépendance entre les performances du système moteur et la conception d'unités de désignation ; le dernier exemple met en garde le concepteur contre la faible capacité de la mémoire à court terme.

6.2.1. τ_s, le cycle du processeur sensoriel et le rafraîchissement de l'écran

D'après le modèle, les images produites dans un même cycle du processeur visuel sont confondues. L'utilisateur a donc l'illusion d'un dessin animé si le taux d'images produites est supérieur à $1/\tau_s$. En choisissant pour τ_s la valeur moyenne de 100 msec, le système répond aux conditions s'il est capable de produire au moins 10 images/sec. La satisfaction de cette contrainte dépend, pour un matériel donné, des techniques de réalisation.

John Uebbing et Charles Young montrent dans [Uebbing 1986] comment combiner judicieusement la technique de la programmation par objets et celle des langages algorithmiques traditionnels. En effet, il est souvent reproché aux langages à objets le coût de la communication par message. Dans le cas qui nous intéresse, l'objectif est de déterminer les conditions où ces coûts sont compatibles avec les performances du système visuel. Connaissant :

- τ_m, le temps de transfert d'un message entre deux objets, et
- n le nombre d'objets graphiques constituants d'un objet O,

l'affichage de O implique un temps $t = n\ \tau_m$ de communication par messages. Si t dépasse un seuil, fonction de τ_s, alors il est souhaitable de :

- redéfinir l'organisation structurelle de O afin de réduire le nombre de messages nécessaires à l'affichage de O,
- produire tout ou partie de O en appliquant des techniques brutales comme l'usage de l'assembleur.

Le rapport entre le coût de transfert d'un message et le cycle du processeur visuel nous permet de constater que le signal d'un événement souris est tout à fait réalisable en utilisant la communication par message. A titre d'illustration, X [Scheifler 86], le système de fenêtrage conçu à l'origine dans le cadre du projet Athena [Balkovich 85], est capable de gérer sur un réseau local l'affichage de rectangles élastiques avec des performances telles, que l'utilisateur n'imagine pas l'existence d'un calculateur distant.

6.2.2. La loi de Fitts et la souris du Star

[Card 83, p. 252-253] rapporte une anecdote intéressante sur l'application de la loi de Fitts à la conception matérielle de la souris du Star [Smith 82].

Initialement, les concepteurs du Star souhaitaient que la vitesse maximale du suivi du curseur fût de 50 cm/sec pour un écran de 35 cm de diagonale. Les évaluations effectuées sur la base du modèle, ont révélé que la vitesse envisagée était de 2 à 3 fois inférieure à la vitesse théorique de la main. Le délai entre les mouvements de la souris et ceux du curseur, se serait traduit, pour l'utilisateur, en une difficulté réelle à désigner rapidement et avec précision. La conception matérielle de la souris du Star a donc été revue.

6.2.3. Les limites de la mémoire cognitive et la mémorisation

Si un utilisateur lit une suite d'informations arbitraires, par exemple une quinzaine de noms de commande dans une langue inconnue, comment cet utilisateur va-t-il se comporter sur le plan mémorisation immédiatement après avoir cessé de consulter la liste?

La réponse est liée à la capacité de la mémoire à court terme (7 ± 2 mnèmes) et aux temps de persistance moyens δ_s, δ_c et δ_{lt} des mémoires :

- δ_s = 200 msec pour la mémoire sensorielle,
- δ_c [3 mnèmes] = 7 sec pour la mémoire à court terme, et
- $\delta_{lt} = \infty$ pour la mémoire à long terme.

Puisque, pour l'utilisateur, les mots n'ont pas de relation (ce sont des mots d'une langue étrangère), chaque mot constitue un mnème. La liste comportant plus de 7 mnèmes, il est clair que l'utilisateur va faire des oublis. Les mots oubliés seront ceux du milieu de la liste. Pourquoi cela?

Les premiers mots présentés, s'ils ont été mémorisés, sont accessibles depuis la mémoire à long terme car, au delà de δ_c, ou bien l'information est perdue. ou bien elle est disponible dans la mémoire à long terme. Les derniers éléments de la liste, s'ils ont été transférés depuis la mémoire sensorielle, sont encore présents dans la mémoire à court terme.

Cette analyse prédit le comportement de l'utilisateur immédiatement après avoir cessé de consulter la liste des noms de commande. Si on le soumet au même test quelques dizaines de secondes plus tard, les premiers noms de la liste seront tout aussi bien retrouvés (puisque $\delta_{lt}= \infty$) ; en revanche, les derniers mots ne seront pas retrouvés avec autant de succès que précédemment : l'utilisateur, dans son effort de recherche, aura réactivé de nouveaux mnèmes qui interfèreront avec les derniers éléments de la liste.

Des expérimentations (Card 83 p.77] confirment cette analyse. La conséquence, pour le concepteur d'interface, est de suppléer aux limitations de la mémoire à court terme. Une première technique, celle des menus et des formulaires, constitue une extension satisfaisante à titre de support syntaxique. Un second moyen concerne le choix d'une terminologie adaptée qui facilite l'association de mnèmes et qui donc réduit le nombre d'éléments arbitraires isolés.

6.3. Les Limites du Modèle

Les quelques exemples qui précèdent démontrent l'utilité du modèle pour expliquer et prédire les performances d'un utilisateur. Ces performances, bien qu'évaluées de manière approximative, permettent de répondre à des questions précises. Les réponses permettent à leur tour de déterminer les contraintes qui devront être respectées lors de la réalisation du système.

Le modèle présente cependant deux sortes de limitations :

1. Les questions possibles concernent les performances motrices et perceptuelles mais ne peuvent se rapporter aux structures cognitives du sujet humain. Le modèle passe sous silence les processus cognitifs les plus fondamentaux : la conceptualisation et la reconstruction mnésiques qui interviennent dans l'apprentissage et la résolution de problèmes, la capacité à traiter plusieurs niveaux d'interruption, le phénomène du parallélisme et les erreurs. Le concepteur n'a donc pas la possibilité de vérifier que l'organisation du système qu'il vient de créer est compatible avec la structure cognitive de l'utilisateur.

2. Le Processeur Humain ne véhicule pas de méthode de conception. Aucun élément du modèle n'indique comment satisfaire les contraintes de performance qu'il permet cependant de déduire. Les modèles GOMS et Keystroke présentés au chapitre suivant tentent de combler cette lacune.

EN RESUME :

Le modèle du Processeur Humain présente le sujet humain comme un système de traitement d'informations. Ce système comprend trois sous-systèmes interdépendants (sensoriel, moteur et cognitif) munis chacun d'un processeur et d'une mémoire. Le cycle de base de ces processeurs est lent (70 msec), la capacité de la mémoire à court terme est faible (7 ± 2 mnèmes), et les performances motrices qui interviennent dans l'interaction homme-ordinateur peuvent s'exprimer formellement avec la loi de Fitts.

Le modèle du Processeur Humain constitue un cadre général de pensée et un point de départ culturel simple dans le domaine de l'ergonomie cognitive. Sa représentation de l'individu sous forme d'un système de traitement de l'information sert de paradigme à la psychologie contemporaine. Cette vision, et la théorie des systèmes de production sur laquelle le modèle s'appuie, ont contribué pour une large part à rénover les techniques de laboratoire de la psychologie traditionnelle.

La simplicité du modèle a sa contrepartie. Le Processeur Humain ne fournit aucune indication sur les représentations mentales alors que ces structures ont une incidence directe sur le comportement ou la prédiction de ce comportement. Bien qu'il se situe à un haut niveau d'abstraction, il ne permet de prédire ou d'expliquer que des phénomènes de performance de bas niveau. Le modèle du Processeur Humain est donc trop réducteur pour assister l'informaticien dans sa tâche de conception de systèmes interactifs. En fait, il ne véhicule aucune méthode de conception.

Chapitre 2

Les Modèles GOMS et Keystroke

1. Introduction

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

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

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

2. Le Modèle GOMS

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

2.1. Les Eléments du modèle

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

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

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

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

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

Si <condition-sur-la-situation-actuelle-est-vraie> alors utiliser la méthode M.

Par exemple, pour un éditeur pleine-page qui permet à la fois les commandes à-la-emacs et les commandes à-la-Macwrite, il existe plusieurs façons de déplacer le curseur : soit avec la souris, soit

avec le clavier, soit par une combinaison des deux techniques. Le choix entre les deux méthodes peut s'exprimer en fonction de la distance entre la position actuelle du curseur et la localisation visée :

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

Si le but à atteindre est placer le curseur au bas de la fenêtre
et
Si la position actuelle du curseur est près du bas de la page
alors utiliser la méthode M2

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

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

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

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

Card, Moran et Newell définissent quatre niveaux d'analyse : les niveaux tâche, fonctionnel, argument et physique. Le niveau tâche structure l'espace de travail en une hiérarchie de sous-tâches dont la nature dépend uniquement du domaine. Les éléments terminaux de la décomposition sont des tâches conceptuelles élémentaires. L'analyse fonctionnelle modélise les tâches élémentaires en termes de fonctions du système. A ce niveau de modélisation, l'accomplissement d'une tâche est décrit par une suite de fonctions. Le niveau argument précise, pour chaque fonction, sa réalisation par une suite de commandes. A ce niveau de modélisation, l'accomplissement d'une tâche est décrit par une suite de commandes. Le niveau physique décrit en termes d'actions physiques la spécification des commandes. Keystroke présenté au paragraphe 3 est une illustration de cette classe de modélisation.

2.2. Evaluation du Modèle GOMS

Les points forts de GOMS concernent les aspects méthode de conception et technique d'évaluation mais les cognitivistes regrettent que GOMS ne soit qu'un modèle de performance.

2.2.1. L'Apport de GOMS

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

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

2.2.2. Les Limites de GOMS

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

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

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

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

3. Le Modèle Keystroke

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

3.1 Les Eléments du Modèle Keystroke

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

Etant donné

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

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

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

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

3.1.1. Les Opérateurs

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

- K ("Keystroking", frappe de touches du clavier ou de la souris),
- P ("Pointing", désignation),
- H ("Homing", rapatriement de la main),
- D ("Drawing", action de dessiner), et
- M ("Mental activity, activité mentale).
- R ("Response time", temps de réponse du système)

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

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

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

$$t_K = (\text{durée totale des tests}) / (\text{nombre total de touches frappées sans erreur})$$

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

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

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

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

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

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

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

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

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

L'opérateur D représente l'utilisation de la souris pour construire un dessin sur l'écran. Le temps t_D nécessaire à la construction d'un dessin dépend des fonctions disponibles. Si le tracé de droite est la seule possibilité, alors t_D est une fonction linéaire du nombre n de segments tracés et de la somme l

de leur longueur. En appliquant la technique des moindres carrés à un ensemble de données expérimentales recueillies sur des tests de tracé de segments de droite, Card, Moran et Newell ont obtenu :

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

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

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

$$t_M = 1.35 \text{ sec.}$$

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

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

t_R vaut ceci :

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

3.1.2. Codage des Méthodes

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

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

Si maintenant, la commande **ls** est spécifiée avec la souris, la méthode devient :

H[souris] **M** **P**[souris] **K**[bouton-souris] **H**[clavier]

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

Règle 0

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

Règle 1

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

Règle 2

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

Règle 3

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

Règle 4

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

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

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

3.2 Exemple d'Application du Modèle Keystroke

Soit la tâche évoquée au paragraphe 2.1 qui consiste à déplacer le curseur vers le bas de la fenêtre. Cette tâche peut être effectuée selon deux méthodes M1 et M2. Le but est de prédire dans quelles conditions M1 est préférable à M2. Nous dirons que M1 est préférable à M2 si, pour un état initial identique de l'environnement, T_{M1}, le temps d'exécution de M1 est inférieur au temps d'exécution T_{M2} de M2.

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

1. codage de la méthode avec les opérateurs physiques uniquement, puis
2. application de la règle 0 pour introduire M, puis
3. application des règles 1, 2, 3 et 4 pour éliminer M.

Soient M1 et M2 exprimées de manière informelle :

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

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

M1 et M2 se transcrivent ainsi :

Méthode M1 :

- étape 1 :
  H[souris] P[souris] K[bouton-souris] H[clavier]
- étape 2, règle 0 : insertion de M devant les K et P
  H[souris] M P[souris] M K[bouton-souris] H[clavier]
- étape 3, règle 1 : PMK = PK
  H[souris] M P[souris] K[bouton-souris] H[clavier]
- $T_{M1} = 2t_H + t_P + t_K + t_M$

Méthode M2 :

- étape 1 : sachant que l'utilisateur a tapé m fois †n et p fois esc-f
  K[touche control] m {K[touche n]} p {K[touche esc] K[touche f]}
- étape 2, règle 0 : insertion de M devant les K
  M K[touche control] m {M K[touche n]} p {M K[touche esc] M K[touche f]}

- étape 3, règle 2 :
  M K[touche control] m {K[touche n]} M p {K[touche esc] K[touche f]}
- $T_{M2} = 2t_M + (m+ 2p +1)\, t_K$

M1 est préférable à M2 si :

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

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

t_H = 0.4 sec. [Card 83, p. 263]

t_P = 1.1 sec. [Card 83, p. 234, p. 262],

t_K = 0.2 sec. [Card 83, p. 266],

t_M = 1.35 sec [Card 83, p. 263].

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

$2t_H + t_P + t_K + t_M < 2t_M + (m+ 2p +1)\, t_K$

$m+2p > (2t_H + t_P - t_M) / t_K$

$m+2p > 2.75$ (1)

Ce résultat indique que pour m=1 et p=1 (passer à la ligne suivante puis au mot suivant) l'utilisation de la souris est mieux adaptée. A mon sens, cette conclusion est un peu hâtive. En effet, dans la relation : $m+2p > (2t_H + t_P - t_M) / t_K$, le facteur influent est t_M (1.35 sec.). Si l'on considère le cas d'un utilisateur expert en Emacs, t_M est non seulement surévalué mais il est très probable que cet utilisateur ne marque pas de pause entre l'exécution de la commande "ligne-suivante" et celle de "mot-suivant". Si l'on considère que cette suite d'actions est une information compilée donc un mnène unique, t_M doit être éliminé de la relation. On obtient alors :

$m+2p > (2t_H + t_P) / t_K$ soit :

$m+2p > 9.5$ (2)

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

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

3.3 Evaluation du Modèle Keystroke

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

3.3.1. Avantages

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

Le second avantage de Keystroke est sa simplicité. La terminologie et la technique de modélisation sont compréhensibles pour un non-spécialiste en psychologie cognitive. Toutefois, la tentative de modélisation du déplacement du curseur présentée au paragraphe 3.2 montre que cette simplicité n'est qu'apparente.

3.3.2. Précautions d'utilisation

Le placement de l'opérateur M est une première source de difficultés : la notion d'opération mentale présentée par Keystroke comme "une pause entre deux mnènes", est trop imprécise. La nature des mnènes varie avec le temps et les individus, d'où l'écart des résultats du paragraphe 3.2 entre l'heuristique "officielle" et la version "personnalisée". L'ajustement des règles générales au cas d'une étude particulière requiert des compétences que l'informaticien n'a généralement pas.

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

- Dans la modélisation du paragraphe 3.2, les temps de réponse du système ont été ignorés. Ceci suppose que le système est suffisamment rapide pour ne pas faire attendre l'utilisateur. Dans le cas inverse, il faut être capable de déterminer t_R. Cette évaluation est certainement imprécise dans le cas d'un système en cours de conception.
- Keystroke ne fait aucune distinction entre la saisie d'un texte ordinaire et la frappe des touches

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

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

EN RESUME

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

- de buts organisés en hiérarchie,
- d'opérateurs (ou actions élémentaires),
- de méthodes (ou procédures de réalisation d'un but), et
- de règles de sélection de méthode lorsque plusieurs d'entre elles permettent d'atteindre un même but.

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

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

GOMS constitue néanmoins une étape marquante vers la conception scientifique d'interface et Keystroke est un outil de mesure objective pour comparer, par exemple, les différentes solutions syntaxiques et lexicales d'un langage de commande.

Chapitre 3

Modélisation de l'Action

1. Introduction

Du point de vue de l'informaticien concepteur, le Processeur Humain définit un canevas théorique utile à la compréhension et à l'évaluation quantitative des mécanismes généraux de l'interaction homme-ordinateur. GOMS et Keystroke remplissent un point précis du canevas pour le cas particulier de la modélisation du comportement de l'utilisateur expert engagé dans des tâches de routine. Le point de vue behavioriste de ces modèles doit être complété avec la perspective cognitiviste : GOMS et Keystroke modélisent le comportement observé ; l'objet de ce chapitre, "vers une théorie de l'action", est d'analyser les processus psychologiques qui conduisent à ce comportement.

Le point de départ de la théorie de l'action de D. Norman repose sur l'hypothèse que l'individu élabore des modèles conceptuels et que ces modèles sont les données déterminantes du comportement [Norman 86]. Nous verrons que cette hypothèse n'est pas en contradiction avec celle de H. Simon exposée au chapitre précédent, selon laquelle c'est l'environnement qui détermine le comportement. L'organisation de ce chapitre suit les deux éléments principaux de l'hypothèse de D. Norman : la première partie introduit la notion de modèle conceptuel ; la seconde décrit la structure du processus de manipulation du modèle. Ce chapitre s'achève par une évaluation du modèle.

2. Notion de Modèle Conceptuel

Un modèle conceptuel est une représentation mentale. Il dépend étroitement de la connaissance déjà acquise et de la compréhension de la situation présente. Il évolue avec l'expérience. Il incomplet et imprécis mais il guide l'essentiel du comportement.

On distingue deux formes de modèle : le Modèle de Conception et le Modèle de l'Utilisateur.

- Le Modèle de Conception est le modèle conceptuel de l'outil. Puisque la raison d'être d'un outil est d'aider l'utilisateur à accomplir un ensemble de tâches, le modèle de conception doit résulter d'une étude appronfondie des besoins, des possibilités et des limitations de l'utilisateur-type.

- Le Modèle de l'Utilisateur est la représentation mentale que l'utilisateur élabore à propos de l'outil. Il résulte de l'interprétation que l'utilisateur fait de l'Image.

L'Image d'un outil est sa présentation physique, son interface d'utilisation. Le concepteur a donc pour tâche de définir une Image qui conduise l'utilisateur à construire, au cours de l'interaction avec l'outil, un modèle compatible avec le Modèle de Conception. Si l'Image est explicite, cohérente et intelligible, alors on peut espérer que l'utilisateur dévelope le modèle adéquat.

Ces définitions et remarques s'appliquent à tout outil, à l'ordinateur en particulier. Dans le cas de l'ordinateur, un troisième type de modèle peut intervenir : le modèle qu'un programme "intelligent" élabore pour représenter l'utilisateur. Ce modèle sert de base aux interfaces capables d'évoluer dynamiquement en fonction des caractéristiques et de l'état mental actuel de l'individu.

La figure 3.1 situe les diverses classes de modèles conceptuels et montre le rôle central de l'Image, passerelle entre le monde physique du système informatique et le monde psychologique de l'utilisateur. Chacun des deux mondes s'exprime dans un langage spécifique. L'Image assure le passage entre les deux langages. L'expérience montre que ce passage n'est pas toujours facile à franchir même dans les cas les plus simples de la vie courante.

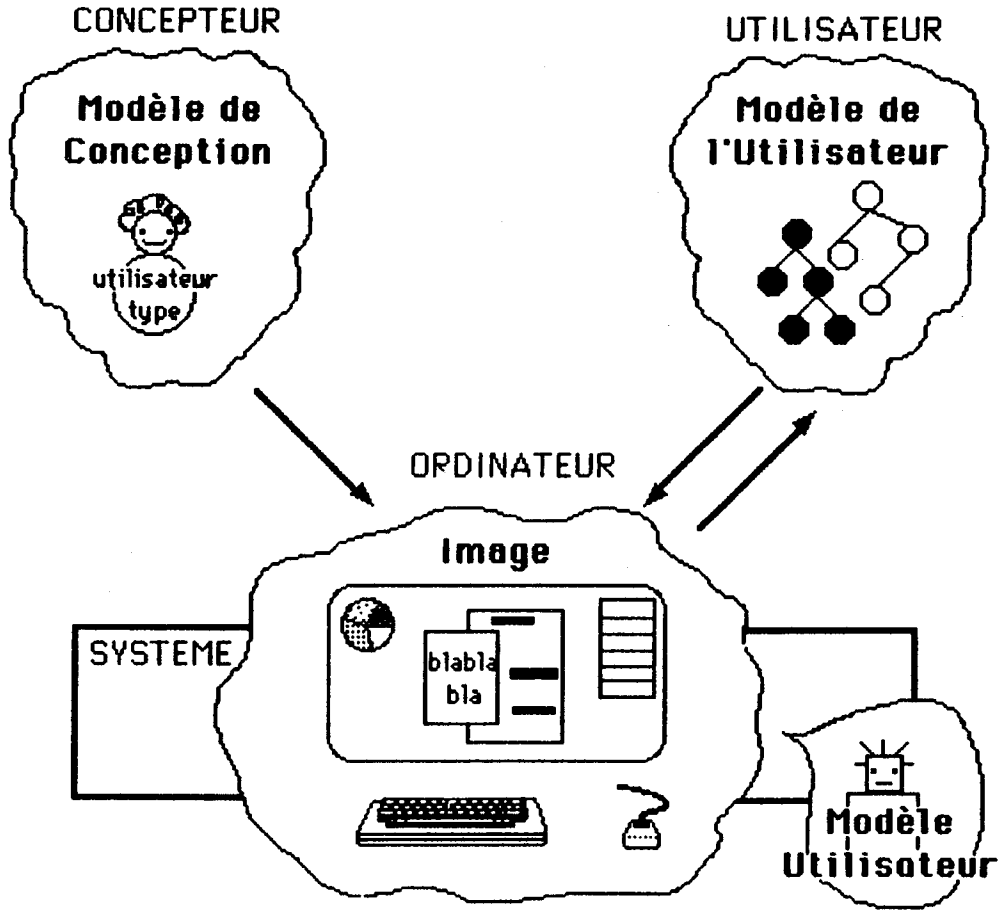


Figure 3.1 : Modèles Conceptuels et notion d'Image.

Prenons comme exemple une illustration inspirée de [Norman 86]. La tâche considérée consiste à remplir une baignoire équipée de deux robinets indépendants, l'un pour l'eau chaude, l'autre pour l'eau froide. Du point de vue de l'utilisateur,

- les éléments signifiants comprennent :

  - d, le débit total de l'eau et
  - t, la température du bain.

  d et t sont les variables psychologiques qui interviennent dans l'expression du but visé.

- l'outil "baignoire à 2 robinets indépendants" se manifeste par quatre variables physiques :

  - d_c, le débit du robinet d'eau chaude,
  - t_c, la température du robinet d'eau chaude,
  - d_f, le débit du robinet d'eau froide,
  - t_f, la température du robinet d'eau froide.

- le robinet d'eau chaude sert à contrôler d_c et le robinet d'eau froide permet d'agir sur d_f. Les robinets sont des dispositifs physiques de commande pour agir sur les variables physiques d_c et d_f.

- les équations (1) et (2) expriment les relations entre les variables psychologiques et les variables physiques :

$$d = d_f + d_c \qquad (1)$$

$$t = (d_c t_c + d_f t_f)/(d_f + d_c) \qquad (2)$$

La formalisation de la tâche "remplir la baignoire" met en évidence plusieurs difficultés pour l'utilisateur :

1. *Un problème de correspondance entre les variables physiques et les dispositifs physiques de commande.*

   Dans sa représentation mentale, l'utilisateur sait, en raison de son expérience passée, que d_c et d_f sont contrôlables avec les robinets. Deux questions se posent alors pour réaliser ce contrôle. Comment dans l'Image distinguer le robinet d'eau froide du robinet d'eau chaude? Dans quel sens faut-il tourner un robinet pour agir sur le débit?

2. *L'inadéquation des relations entre les variables physiques et les variables psychologiques.*

   Les équations (1) et (2) montrent que pour refroidir la température du bain tout en conservant le même débit, il faut simultanément augmenter d_f (pour faire plus froid) et diminuer d_c (pour conserver d constant) ou bien diminuer d_c (pour faire moins chaud) et augmenter d_f (pour conserver d constant). Quel que soit le raisonnement choisi, l'utilisateur se voit dans cette situation difficile où il lui faut manipuler simultanément deux robinets dans des sens contraires. Le but choisi est difficile à atteindre en raison de l'inadéquation des relations entre les variables physiques et psychologiques

3. *Un problème d'évaluation des résultats.*

   Avec deux becs verseurs, il est difficile d'évaluer le débit total. Il est encore plus difficile (voire sensoriellement douloureux) de déterminer la température obtenue!

Cette analyse indique qu'un robinet mélangeur résoud le problème de l'évaluation et qu'un robinet

thermostatique définit un isomorphisme entre les variables physiques et les variables psychologiques.

Nous venons de mentionner les notions de variables et les activités de contrôle et d'évaluation. Ces notions et ces activités forment un sous-ensemble des aspects d'une tâche.

3. Les Aspects d'une Tâche

D. Norman dans [Norman 86] indique que la réalisation d'une tâche met en jeu au moins sept activités :

1. l'établissement d'un but,
2. la formation d'une intention,
3. la spécification d'une suite d'actions,
4. l'exécution des actions,
5. la perception de l'état du système,
6. l'interprétation de l'état du système,
7. l'évaluation de l'état du système par rapport au but.

Précisons la définition de but, d'état et de mécanisme de contrôle.

1. Un but est une représentation mentale de l'état à atteindre. Par exemple, " intervertir deux mots" d'un texte constitue un but. Dans le cas où le but ne peut être directement réalisé avec les commandes du système, l'utilisateur élabore un plan de résolution dans lequel le but est décomposé en une hiérarchie de sous-buts plus simples jusqu'à ce qu'ils soient réalisables avec une commande.

2. L'état d'un système se définit par un ensemble de variables physiques. Par exemple, le placement de la fenêtre (lucarne sur le contenu d'un document) et la localisation du curseur (point d'insertion de texte dans un document), sont des variables physiques qui intéressent directement l'utilisateur dans le but "intervertir deux mots".

3. Un mécanisme de contrôle est un dispositif physique qui permet d'agir sur les variables physiques. Par exemple, la souris permet de placer le curseur à l'endroit voulu ou de déplacer la fenêtre sur le contenu du document.

Supposons qu'un utilisateur rédigeant une lettre avec MacWrite, ait pour but B : "Intervertir mot_1 et mot_2". Le système se trouve dans l'état physique $E_\phi(V1_\phi, V2_\phi)$. La variable physique $V1_\phi$ désigne la localisation actuelle du curseur et $V2_\phi$ représente la suite de mots constituant la lettre soit : $V2_\phi = \{m_1\ m_2 \ldots m_i\ mot_1\ mot_2\ m_{i+3} \ldots m_n\}$. $V1_\phi$ et $V2_\phi$ sont contrôlables au moyen des dispositifs physiques souris et clavier. Pour simplifier l'exemple, nous supposerons que le contenu de la lettre est entièrement visible sur l'écran. Reprenons maintenant les étapes cognitives mises en jeu pour satisfaire B.

1. *Etablissement d'un but .*

Soit B_μ, la représentation mentale de B.

2. *Formation d'une intention .*

L'utilisateur examine l'état du système et le compare au but recherché. Pour que la comparaison soit possible, état et but doivent être exprimés dans le même formalisme : ici, celui de la représentation mentale. L'état $E_\phi(V1_\phi, V2_\phi)$ du système est traduit en une représentation mentale équivalente $E_\mu(V1_\mu, V2_\mu)$ où $V1_\mu$ et $V2_\mu$ sont les représentations mentales de $V1_\phi$, $V2_\phi$. La différence (ou la distance) entre la représentation mentale $E_\mu(V1_\mu, V2_\mu)$ de l'état du système et le but B_μ donne naissance à une intention. Cette intention est la décision d'agir pour atteindre B.

3. *Spécification d'une suite d'actions .*

L'intention doit se concrétiser en une suite d'actions sur les dispositifs physiques de commande. Pour cela, les spécifications internes du but doivent être exprimées dans le formalisme de l'Image : B_μ est donc traduit en l'état physique souhaité $E'_\phi(V1_\phi, V2_\phi)$ où $V2_\phi = \{m_1\ m_2 \ldots m_i\ mot_2\ mot_1\ m_{i+3} \ldots m_n\}$. $V2_\phi$ désigne la suite de mots du document dans lequel mot1 et mot2 se trouvent maintenant inversés. Cette traduction requiert la connaissance des relations entre les variables psychologiques et les variables physiques. Elle nécessite aussi la connaissance des liens entre les variables physiques et les dispositifs de commande qui permettent de les modifier. En l'occurrence, l'utilisateur doit associer la variable psychologique "lieu d'insertion" à la variable physique "curseur" et il doit connaître le lien entre le curseur et le dispositif de commande "souris". Le résultat de la traduction est un plan de résolution, spécification mentale des actions à exécuter.

4 et 5. *Exécution des actions et Perception de l'état du système.*

L'exécution des actions sont les actes moteurs qui conduisent au changement de l'état physique du système : $E_\phi(V1_\phi, V2_\phi)$ devient $E''_\phi(V1_\phi, V2_\phi)$. Si l'Image du système est sensible aux actions de l'utilisateur, le changement de l'état est immédiatement perceptible (on parle alors de retour immédiat d'information). La perception de l'état est une représentation mentale de E''_ϕ, soit : $E''_\mu(V1_\mu, V2_\mu)$.

6. *Interprétation de l'état physique perçu.*

$E''_\mu(V1_\mu, V2_\mu)$ est maintenant interprété dans les termes des variables psychologiques d'intérêt, de celles notamment qui interviennent dans B_μ.

7. *Evaluation.*

La comparaison entre l'interprétation de l'état perçu $E"_{\mu}(V1_{\mu}, V2_{\mu})$ et B_{μ} constitue l'évaluation. Cette évaluation peut conduire à la révision du plan et produire de nouveaux buts et intentions.

Les sept phases identifiées par D. Norman ne sont pas nécessairement présentes dans l'accomplissement d'une tâche ni forcément appliquées dans cet ordre. En particulier, les trois premières étapes sont récursives car au cours de la résolution, il toujours possible de revenir sur le but, les intentions et la spécification du plan. L'intérêt premier de cette classification est d'identifier les points critiques, notamment la distance entre les variables psychologiques et les variables physiques, distance qui complique les opérations de traduction et, par voie de conséquence, l'utilisation du système. D. Norman distingue la distance d'exécution de la distance d'évaluation.

4. Distance d'Exécution et distance d'Evaluation

La notion de distance exprime la dissimilitude entre la représentation de l'Image et celle maintenue dans le Modèle de l'Utilisateur. La distance d'exécution traduit l'effort de mise en correspondance entre la représentation mentale interne de la tâche à effectuer et la représentation physique externe imposée par l'Image. La distance d'évaluation traduit l'effort cognitif inverse (voir la figure 3.2). Puisque, de fait, ces distances sont inévitables, l'objectif du concepteur est d'en raccourcir la longueur par l'intermédiaire de l'Image.

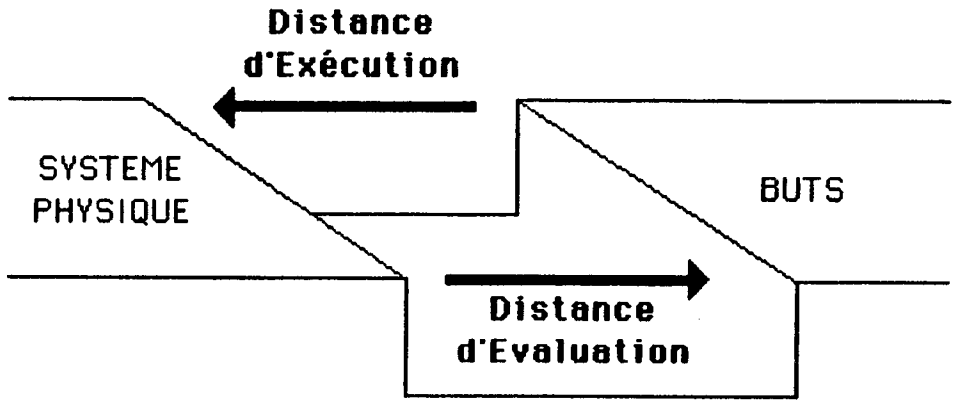


Figure 3.2 : Les distances d'exécution et d'évaluation.

Du point de vue linguistique, l'Image est un langage d'interface constitué de deux dialectes : l'un pour spécifier les expressions d'entrée, l'autre pour présenter les concepts du système sous forme d'expressions de sortie. Elle peut donc s'étudier selon les deux composantes : sémantique et forme. Hutchins et ses co-auteurs intègrent ces deux vues et parlent de distance sémantique et de distance articulatoire [Hutchins 86] :

- la distance sémantique exprime les relations entre les buts que l'utilisateur s'est fixé et la signification des expressions du langage d'interface. Elle traduit l'effort de mise en correspondance entre la connaissance sémantique que l'utilisateur a dans le domaine de la tâche et la connaissance sémantique qu'il possède sur le système ;

- la distance articulatoire reflète les relations entre la signification d'une expression du langage et sa forme. Elle traduit l'effort de traduction entre la connaissance sémantique que l'utilisateur a du système et sa connaissance syntaxique des éléments de présentation du système.

La figure 3.3 résume les relations entre buts, signification et forme, et montre comment les distances d'exécution et d'évaluation se décomposent en distance sémantique et en distance articulatoire. La distance d'exécution comprend la distance sémantique d'entrée et la distance articulatoire d'entrée. Dans l'autre sens, la distance d'évaluation inclut la distance articulatoire de sortie et la distance sémantique de sortie :

- la distance sémantique d'entrée définit la différence entre le but recherché (par exemple, supprimer mot1) et la signification de l'expression d'entrée (par exemple, la commande couper). Cette distance est parcourue par l'activité qui aboutit à la formation d'une intention. Elle fait intervenir deux formes de connaissances sémantiques : celle sur le domaine et celle sur le système ;

- la distance articulatoire d'entrée représente la mise en correspondance entre la signification de l'expression d'entrée (commande couper) et la forme de cette expression (par exemple, sélectionner mot1 avec la souris puis taper la touche "retour-arrière"). Elle est franchie par la spécification des actions physiques. Elle fait intervenir les connaissances sémantiques et syntaxiques que l'utilisateur possède sur le système. L'exécution des actions entraîne la mise à jour (ou l'apparition) d'expressions de sortie. La perception de ces expressions engendre une forme (par exemple, mot1 vient de disparaître) ;

- la distance articulatoire de sortie définit la différence entre la forme d'une expression de sortie (mot1 a disparu) et sa signification (par exemple, mot1 est détruit). Elle est franchie dans l'étape d'interprétation ;

- la distance sémantique de sortie établit la relation entre la signification de l'expression de sortie (mot1 est détruit) et l'objectif de l'utilisateur (supprimer mot1). Elle est franchie dans l'étape d'évaluation.

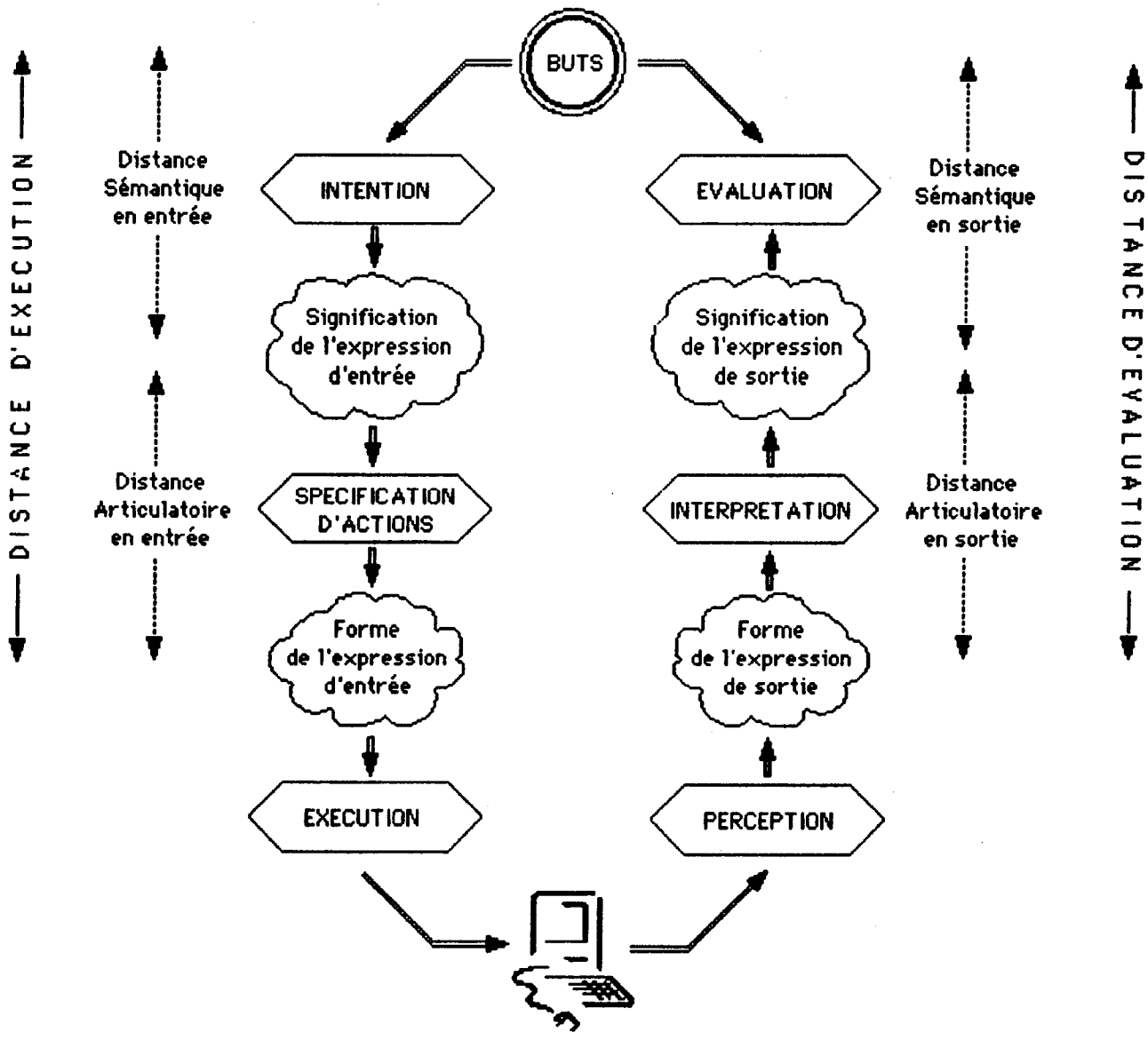


Figure 3.3 : Distances Sémantiques et Distances Articulatoires.

5. Evaluation de la Théorie de l'Action

La théorie qui vient d'être présentée ne prétend pas être la théorie de l'action mais une théorie sur les processus cognitifs sous-jacents à la réalisation d'une tâche. Avec son point de vue cognitif, cette théorie complète les modèles GOMS et le Processeur Humain en plusieurs points : elle précise la notion d'état, elle prend en compte les erreurs, elle explique les difficultés de l'utilisateur et justifie l'utilité de la notion de modèle conceptuel.

D'une théorie à l'autre, nous retrouvons la décomposition hiérarchique d'une tâche en buts et sous-buts jusqu'à ce que les sous-buts correspondent à des unités cognitives procédurales, un savoir-faire compilé. Alors que GOMS se contente de définir un but "comme une structure symbolique qui décrit un état à atteindre", la Théorie de l'Action précise la notion d'état. Elle distingue l'état effectif de l'état perçu. L'état effectif est une fonction portant sur des variables physiques, caractéristiques du modèle conceptuel du système, tandis que l'état perçu est la traduction de l'état effectif sous forme de variables psychologiques, caractéristiques du modèle conceptuel de l'utilisateur.

La différence de représentation entre le monde physique et le monde mental met en évidence la nécessité, pour l'utilisateur, d'effectuer des traductions. Contrairement à GOMS qui a une vue synthétique du comportement, la Théorie de l'Action analyse le processus de traduction qui sous-tend le comportement. Cette analyse identifie des phases auxquelles correspondent des besoins. Lorsque les besoins ne sont pas ou sont mal satisfaits, il y a risque d'erreur. Une erreur est commise soit parce que l'Image n'explicite pas l'état effectif du système, soit parce que la correspondance entre les variables physiques et psychologiques est complexe, soit encore parce que les dispositifs de contrôle des variables physiques ne sont pas adaptés à la tâche. Tandis que GOMS se cantonne au cas idéal de l'utilisateur infaillible, la Théorie de l'Action choisit le parti réaliste d'identifier les sources d'erreur et d'expliquer les difficultés rencontrées par l'utilisateur.

Erreurs, difficultés et réussites tiennent au modèle conceptuel que forge l'utilisateur avec l'expérience. Cette utilisation en lecture et en écriture du modèle conceptuel traduit bien l'aspect "système adaptatif" qui sert d'hypothèse à GOMS. En effet, l'adaptation revêt deux aspects complémentaires : l'assimilation et l'accommodation. Assimiler, c'est modifier l'environnement pour qu'il corresponde à soi. Accommoder, c'est se changer pour correspondre à l'environnement. L'utilisateur assimile lorsqu'il franchit la distance d'évaluation : dans les phases de perception et d'interprétation, il traduit une représentation de l'état de l'environnement en une représentation mentale qui peut éventuellement modifier le modèle conceptuel. L'utilisateur accommode lorsqu'il franchit la distance d'exécution : dans les phases de spécification d'actions et d'exécution, il utilise en lecture le modèle conceptuel pour traduire l'objectif mental en une suite d'actions conformes à l'environnement.

Jusqu'ici, nous avons présenté la Théorie de l'Action comme un modèle explicatif des réussites, des difficultés et des erreurs de l'utilisateur. Nous verrons dans le chapitre sur les principes pratiques, l'utilité de ces explications pour classer les besoins de l'utilisateur et attirer l'attention du concepteur sur la nature de ces besoins.

La Théorie de l'Action n'est pas seulement un modèle explicatif. Elle peut, comme GOMS, Keystroke et le Processeur Humain, servir de support à l'évaluation prédictive comme en témoigne ETIT. ETIT est une technique d'analyse que Moran a appliquée à l'étude des tâches de deux éditeurs de texte [Moran 83]. Moran part de la distance qui sépare la représentation mentale de la représentation physique. Il exprime sous forme de règles la mise en correspondance entre les concepts mentaux et les concepts du système. L'examen des règles fournit une métrique de la difficulté d'utilisation et d'apprentissage.

EN RESUME :

La Théorie de l'Action assimile la réalisation d'une tâche au parcours d'une distance. Cette distance qui traduit la dissimilitude entre le monde mental et le monde physique fait intervenir les notions de modèles conceptuels et d'Image du système ; elle comprend sept étapes.

Un modèle conceptuel est une représentation mentale. On distingue deux modèles conceptuels :

- le modèle de l'utilisateur qui est la représentation mentale que l'utilisateur a du système,
- le modèle de conception qui est la représentation mentale que le concepteur a du système pour un utilisateur-type.

L'Image a pour rôle d'expliciter le modèle de conception afin que l'utilisateur élabore un modèle mental en accord avec le modèle de conception.

Les sept étapes du processus cognitif qui conduit à la réalisation d'une tâche sont :

1. l'établissement d'un but. Un but est la représentation mentale de l'objectif en termes de variables psychologiques ;
2. la formation d'une intention. Elle résulte de la différence entre le but et la représentation mentale de l'état actuel du système ;
3. la spécification d'une suite d'actions. Cette spécification est une représentation mentale du plan de résolution ; elle résulte de la traduction du but en l'Image physique souhaitée ; elle fait intervenir la connaissance des relations entre les variables psychologiques et physiques et entre les variables physiques et les dispositifs de commande ;
4. l'exécution des actions. Elle consiste à manipuler les dispositifs physiques de commande pour modifier les variables physiques ;
5. la perception de l'état du système. Elle est une représentation mentale de la nouvelle Image du système ;
6. l'Interprétation de l'état du système. Elle s'effectue sur l'état perçu en fonction des variables psychologiques impliquées dans le but ;
7. l'évaluation. Elle compare le résultat de l'interprétation au but recherché.

Chapitre 4

Principes Pratiques Ergonomiques

1. Introduction

Jusqu'ici, nous avons pris connaissance des quelques théories marquantes du domaine de l'interaction homme-ordinateur. Ce chapitre présente un point de vue complémentaire : celui de l'approche expérimentale. Bien que les deux approches aient le même objectif (l'amélioration de la qualité des interfaces), les méthodes et l'esprit diffèrent. Les théories des chapitres précédents tentent d'expliquer et de formaliser les mécanismes cognitifs qui régissent l'utilisation d'un système informatique. Elles fournissent une structure générale au processus de conception des interfaces. A l'inverse, l'expérimentalisme consiste à émettre une hypothèse sur un aspect très particulier de l'interaction puis à la vérifier sur le terrain [Michard 82]. Cette hypothèse peut concerner l'environnement de travail, le matériel, les caractéristiques de l'utilisateur, la structure des tâches à informatiser, la nature et la forme des informations que l'utilisateur et le système vont échanger, etc.

La variété des aspects étudiés en ergonomie et, pour chaque aspect, la multitude des paramètres envisageables se traduisent en une quantité phénoménale de recommandations qui ne cessent de croître en nombre et en complexité. Il est clair que l'informaticien n'a pas la compétence, ni le temps, d'embrasser cette connaissance de manière exhaustive. L'objectif de ce chapitre est de présenter une synthèse qui soit facile à appréhender et qui montre que la conception d'interface n'est pas uniquement une affaire de bon sens. Ce chapitre n'a pas la prétentation d'être un guide complet de l'ergonomie. On trouvera dans [Scapin 87] et [Shneiderman 87] un complément utile à cette introduction.

Le premier des principes est que toute conception d'interface doit placer l'utilisateur au centre de l'étude. Une conception qui a pour élément directeur la spécificité de l'utilisateur peut néanmoins faire usage de points de repère généraux. Ce chapitre présente successivement cinq points de référence fondamentaux : la cohérence, la concision, les retours d'information, la structuration des activités et la flexibilité. Ces points ont été classés en fonction de leur facilité de mise en application, non pas en raison de leur incidence sur la qualité de l'interaction. En matière de qualité, les priorités dépendent étroitement de chaque cas. Une définition dans l'absolu de l'importance d'une règle par rapport à son effet sur la qualité de l'interaction est par conséquent illusoire.

2. Règle de la Cohérence

La cohérence repose sur le caractère unitaire des constituants de l'interface, c'est-à-dire sur l'absence d'exceptions. Une métaphore d'interaction peut servir d'élément fédérateur à l'homogénéité de l'interface. La cohérence concerne toutes les étapes de réalisation d'une tâche, en particulier la

spécification du plan, l'exécution, la perception et l'interprétation.

2.1. Cohérence et choix d'une métaphore d'interaction

En linguistique, une métaphore est "l'emploi d'un terme concret pour exprimer une notion abstraite par substitution analogique ... " (Grand Dictionnaire Encyclopédique Larousse). La création poétique mais aussi la langue familière en font un grand usage. Dans le domaine de l'interaction homme-ordinateur, la métaphore est également considérée comme un facteur prédominant de créativité, et notamment pendant l'apprentissage.

L'apprentissage, estiment certains cogniticiens, est structuré par des comparaisons métaphoriques [Carroll 85b]. Par exemple, un "répertoire est un dossier". Une métaphore permet à l'apprenant d'initialiser le modèle mental d'une notion mal connue, par exemple la notion de répertoire, avec le modèle d'une notion connue, par exemple celle de dossier. L'expérience aidant, l'utilisateur émet des hypothèses, les vérifie et complète la représentation initiale. La métaphore est un modèle prédictif du comportement du monde. C'est un stimulateur d'inférences qui permet d'identifier les traits communs mais aussi les différences entre les notions connue et moins connue. La métaphore est un véhicule cognitif précieux que l'on commence à exploiter fructueusement dans le domaine de l'interaction homme-ordinateur.

Dans l'interaction homme-ordinateur, on compte essentiellement deux classes de métaphores : celles qui s'inspirent du monde réel et celles qui parlent d'un monde abstrait [Hutchins 86]. Dans la première catégorie, nous trouvons la métaphore du "desktop" lancée par Smalltalk [Golberg 84] mais rendue célèbre par le Star de Xerox [Smith 82] et reprise ensuite par les machines Lisa et Macintosh d'Apple. Sur ces machines, l'interface s'efforce d'être la reproduction miniature du monde du bureau. Par exemple, un dossier électronique a la même présentation externe qu'un dossier réel : il comporte une chemise et un nom. Comme le dossier du monde réel, il peut être ouvert, reproduit, déplacé sur le bureau, rangé dans un autre dossier, etc. Toutes ces actions s'effectuent généralement avec la souris qui sert de substitut électronique de la main. Avec la souris (ou tout autre dispositif de pointage), l'utilisateur a l'impression d'agir sur les objets. Il n'y a pas d'intermédiaire. L'utilisateur est engagé dans l'action de manière directe.

Contrairement aux interfaces de "manipulation directe", les interfaces de la seconde classe montrent une image dans laquelle le monde n'est plus représenté explicitement. L'interface n'est plus le monde miniaturisé électroniquement mais devient un médium linguistique. L'utilisateur et le système sont engagés dans une conversation sur un monde supposé. L'interface est alors un intermédiaire entre l'utilisateur et le monde dont il est question. Les langages de commande

traditionnels et les interfaces en langue naturelle entrent dans cette catégorie. Avec ce type d'interface, l'utilisateur n'effectue pas l'action sur un objet visible mais il décrit une action sur un objet supposé. L'utilisateur n'est pas en contact direct avec l'objet mais manipule des structures linguistiques.

On peut s'interroger sur les avantages respectifs de ces deux classes de métaphores. Si l'on considère la distance sémantique, et si l'on accepte l'hypothèse que la pensée procède par métaphores, les interfaces qui reproduisent le monde réel facilitent probablement l'apprentissage mais ne réduisent pas forcément la distance articulatoire.

Quelle que soient ses avantages ou ses inconvénients, une métaphore définit un style d'interaction. Elle constitue l'unité de cohérence générale pour l'Image.

2.2. Cohérence et spécification du plan

D'une tâche à l'autre, que ce soit au sein d'une application ou entre applications, il faut s'assurer que la suite des commandes à exécuter est identique pour réaliser une sous-tâche commune. Par exemple, dans les environnements, traitement de texte et courrier électronique, l'accomplissement de la sous-tâche "dupliquer-imprimer un texte" devrait utiliser la même suite de commandes.

2.3. Cohérence et exécution

L'étape d'exécution concerne les trois niveaux syntaxique, lexical et pragmatique.

2.3.1. Aspects syntaxiques

La définition de la structure syntaxique des commandes appelle au moins deux questions :

1. Quel ordre choisir pour les arguments? Les ergonomes estiment que d'une commande à l'autre, l'ordre de spécification des arguments communs doit être identique [Barnard 81]. Cette règle s'applique aussi bien aux langages de commandes textuels qu'aux langages de type iconique.

2. Le préfixage est-il préférable à une notation postfixée? La réponse dépend du type de langage. Pour les langages textuels, Barnard aboutit à la conclusion que la forme "Verbe Objet" est la structure qui convient. Pour les langages de type iconique, Foley suggère la forme inverse, "Objet Verbe".

Le problème d'ordre ne se pose plus si le système le prend en charge. Par exemple, dans les formulaires, l'ordre de spécification des arguments est généralement libre ; les systèmes Concerto [Conchon 86] et EzWin [Lieberman 85] admettent à la fois les formes préfixée et postfixée.

Contrairement à l'intuition, cette liberté ne joue pas toujours en faveur de l'utilisateur : Barnard rapporte à propos de son expérimentation sur le préfixage et le postfixage que les utilisateurs de la syntaxe libre avaient eu davantage recours à l'assistance d'un expert.

2.3.2. Aspects lexicaux

Une première difficulté rencontrée dans la définition du lexique est le choix d'une nomenclature. Très fréquemment la terminologie du système est floue ou ne correspond pas au vocabulaire de l'utilisateur. L'imprécision des termes d'un menu peut conduire l'utilisateur débutant à formuler une intention inappropriée. (Au restaurant, si le menu comporte "Plat du Jour" ou "Patisserie Maison", on peut avoir recours aux compétences du maître d'hôtel.)

Les ergonomes recommandent de choisir des termes précis, didactiques et pertinents. Par exemple: DETRUIRE est plus précis que ENLEVER ; les intitulés des champs d'entrée HEURE [HH MM SS] et DISTANCE [en km] qui précisent le format et l'unité de mesure, sont plus didactiques que les simples noms HEURE et DISTANCE ; OUI/NON (ou O/N) est généralement plus signifiant pour l'utilisateur que 0/1.

Contrairement aux apparences, cette activité de dénomination n'est ni marginale ni facile. Dans un domaine autre que celui de l'interaction homme-ordinateur, les programmeurs connaissent bien l'incidence du choix des identificateurs sur la lisibilité de leurs programmes. Ils savent aussi combien ce choix est difficile et conflictuel. En outre, le choix est invariablement influencé par l'héritage culturel du concepteur.

Le second problème est l'élimination des incohérences lexicales. Il faut veiller à ce qu'une fonction sémantique ou un concept soient toujours désignés par le même nom et ceci quelque soit le contexte. L'exemple de la fonction "terminer" dans l'environnement Unix est particulièrement évocateur : pour spécifier la fin de saisie d'un message électronique, la notation lexicale usuelle est "."; elle est "q" pour quitter la messagerie et "logout" pour terminer la session.

2.3.3. Aspects pragmatiques

La cohérence pragmatique concerne essentiellement la localisation et l'organisation spatiales des informations. La cohérence spatiale aide l'utilisateur à acquérir une connaissance motrice qui permet d'anticiper les actions physiques. Ainsi, l'utilisateur sait par avance où diriger son regard, comment orienter les mouvements de la souris ou quel doigt actionner. La cohérence pragmatique aide à franchir la distance articulatoire.

Le principe général à retenir est le maintien d'une certaine forme de stabilité de l'écran et de l'utilisation des dispositifs physiques de commande. Stabilité n'est pas ici synonyme de permanence mais d'expectative : les informations doivent être là où l'utilisateur les attend et les dispositifs physiques doivent se comporter conformément à l'attente.

Nous avons relevé quelques critères de cohérence pragmatique ayant rapport avec les boutons de la souris, les menus et les formulaires. Il en est bien d'autres et des bien controversés comme le nombre idéal de boutons pour la souris [Price 83]!

- Dans le cas de la souris, chacun sait qu'un bouton doit être lié en permanence à la même fonction (ou à la même classe de fonctions). Pourtant, chacun de nous a pu constater les incohérences les plus frappantes sur l'affectation de ces boutons.

- Concernant les menus, la littérature abonde de recommandations et d'évaluations. Nous retiendrons au moins ceci : afficher les menus à la même place (ou bien à l'emplacement désigné par la souris) et ordonner les options en conservant un ordre fixe. Cet ordre est, en priorité, l'ordre logique défini par la tâche. S'il n'y a pas d'ordre logique, alors utiliser la fréquence des options ou, à défaut, l'ordre alphabétique. Noter que dans le cas d'un système multilingue où le changement de langue est possible en cours de session, la règle de l'ordre alphabétique risque de contrarier celle de la cohérence spatiale.

- Pour les formulaires, les ergonomes nous enseignent au moins ceci : sauf si l'ordre logique de la tâche s'y oppose, grouper en haut du formulaire les entrées requises et en bas, les entrées optionnelles. Cet assemblage a pour effet de concentrer les déplacements donc de réduire les mouvements soumis à la loi de Fitts. On constatera avec satisfaction que l'annuaire téléphonique du Minitel répond à cette règle.

2.4. Cohérence et étapes de perception et d'interprétation

On retrouve pour les étapes de perception et d'interprétation les mêmes règles que pour la spécification et l'exécution : en effet, les éléments d'entrée peuvent aussi servir de données de sortie et réciproquement. Bien que ces règles soient particulièrement simples à mettre en œuvre, elles se trouvent fréquemment transgressées. L'exemple du gestionnaire de fenêtres du système Andrew [Gosling 86c] est à ce titre illustratif.

Dans le système Andrew, les fenêtres ne se recouvrent pas : les concepteurs ont décidé, sur des bases purement intuitives, que le partitionnement de l'écran "était bon pour l'utilisateur". En réalité, lorsque l'utilisateur crée, détruit ou change les dimensions d'une fenêtre, le système, afin de maintenir le partitionnement de l'écran, modifie la taille, voire la localisation des fenêtres. Le bruit visuel ainsi créé gêne la phase d'interprétation ; les variations des emplacements des variables physiques empêchent l'anticipation gestuelle et l'utilisateur est contraint de corriger les choix stratégiques du système lorsqu'ils sont contraires aux besoins de la tâche. Un contre-exemple nous vient du toolbox du Macintosh [Rose 86] qui encourage la stabilité dans la présentation des variables physiques et qui suggère une sorte de norme pour les réalisateurs d'applications Macintosh.

2.5. Conclusion sur la cohérence

Avec un peu de motivation et de vigilance, les règles sur la cohérence sont faciles à satisfaire et les incohérences lexicales sont généralement réparables sans trop de difficulté. Les boîtes à outils, qui tendent à définir un style d'interaction homogène, et l'organisation modulaire des logiciels qui facilite l'ajustement des éléments lexicaux, participent à cet objectif de cohérence.

3. Règle sur la concision

La concision repose sur l'harmonieuse combinaison du bref et de l'expressif. Dans le contexte de l'interface homme-ordinateur, la concision a deux effets : éviter les surcharges d'information de sortie et réduire le nombre d'actions physiques nécessaires à la spécification des expressions d'entrée. Le niveau de concision des expressions de sortie dépend de l'utilisateur et de la tâche en cours. Il est donc directement lié à la structuration et à la flexibilité du système, deux thèmes que nous traitons aux paragraphes 5 et 6. Nous considérons ici la concision des expressions d'entrée.

Les techniques qui permettent de réduire le nombre d'actions physiques sont connues mais pas toujours intégrées de manière harmonieuse. Les abréviations, les macrocommandes, le couper-coller, les valeurs par défaut et les fonctions refaire et défaire sont parmi les plus fréquentes.

3.1. Concision et abréviations

Les abréviations s'adressent essentiellement aux utilisateurs expérimentés. L'exemple du menu en démontre l'utilité. Un menu facilite l'étape de spécification d'un plan mais n'en favorise pas l'exécution. S'il agit comme une extension de la mémoire à court terme (au lieu de reconstituer l'élément mental recherché, l'utilisateur se contente de le reconnaître dans la liste des options proposées), il prend du temps à s'afficher et à disparaître et il impose des changements de dispositifs de contrôle (notamment lorsque, dans la tâche, l'usage du clavier est dominant). L'abréviation telle que la proposent les menus du Macintosh, concilie la facilité de spécification et la rapidité d'exécution. Dans l'exemple de la figure 4.1, la commande **New Folder** peut être spécifiée par la frappe simultanée des touches ⌘ et **N**.

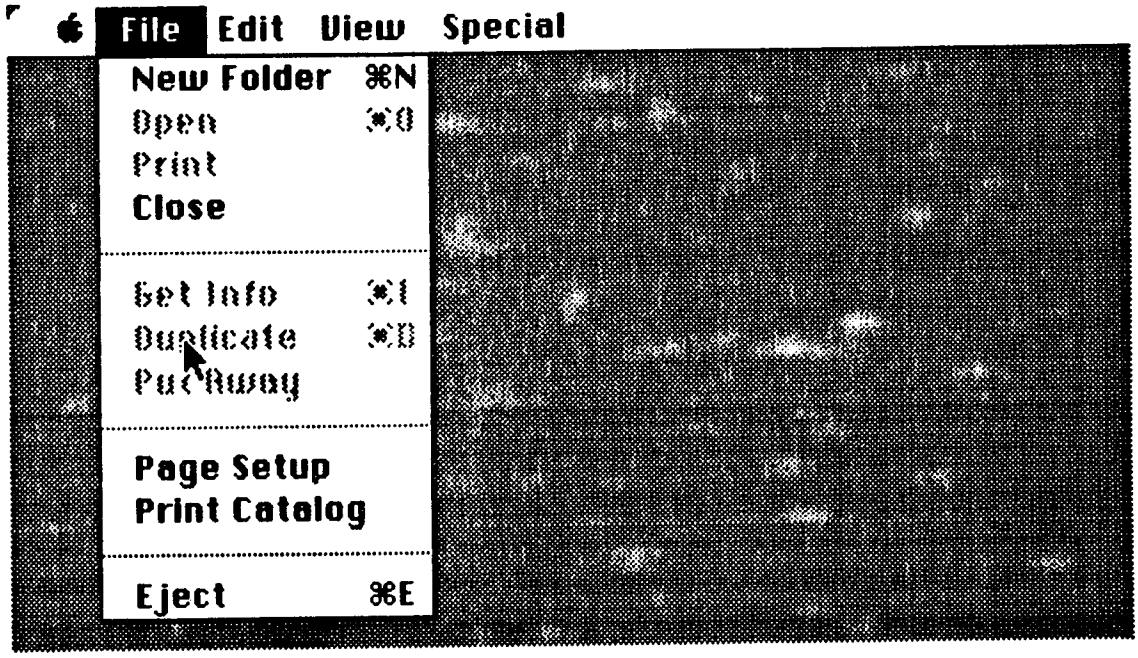


Figure 4.1 : Un menu du Macintosh. Les caractères d'abréviations sont indiqués à droite des éléments de menu. L'ajout, le retrait et la modification des raccourcis sont accessibles à l'utilisateur sans recompilation de l'appplication.

Si les abréviations sont utiles, encore faut-il que leur forme soit compréhensible, c'est-à-dire dérivable selon des règles précises. Il existe plusieurs règles en usage parmi lesquelles un caractère spécial suivi de l'initiale d'une commande, la suppression des voyelles ou encore les troncatures non ambiguës. Bonnie John dans [John 85, John 87] effectue une excellente analyse des temps nécessaires à la spécification des commandes abrégées. L'hypothèse choisie est celle de GOMS : les tâches de routine (comme l'est l'usage des abréviations par un utilisateur expert) sont effectuées en exécutant des algorithmes compilés. Ces algorithmes s'expriment en termes d'opérateurs perceptuels, cognitifs et moteurs (voir Keystroke, par exemple). Connaissant le temps d'exécution des opérateurs, il est possible de prédire les temps nécessaires à la reconstitution mnésique d'une abréviation et à sa saisie. La distinction entre ces deux temps permet de repérer les exceptions, par exemple une abréviation qui serait facile à retrouver mais difficile à saisir.

3.2. Concision et macrocommandes

La macrocommande est aux langages d'interaction ce que la procédure est aux langages de programmation. C'est un mécanisme d'abstraction et une technique d'extension. En tant que mécanisme d'abstraction, elle répond au phénomène cognitif de l'apprentissage qui, avec l'expérience, encapsule dans un mnème un ensemble organisé de connaissances plus élémentaires. En tant que technique d'extension, elle permet de concilier le particularisme et la généralité.

Le particularisme désigne ici les besoins spécifiques de l'utilisateur pour l'expression de ses intentions. On se souvient de la distance sémantique à parcourir entre la formation de l'intention et la spécification du plan de commandes. Une façon de raccourcir cette distance est de fournir à l'utilisateur un langage de haut niveau qui lui permet d'exprimer directement les structures de décomposition des problèmes les plus fréquents. Ce langage, façonné à l'expression des tâches usuelles, perd la puissance de la généralité et rend difficile, voire impossible, la spécification des tâches qui ne se décomposent pas dans ses termes. Ce conflit entre particularisme et généralité est résolu dans les systèmes Unix et les environnements Lisp grâce à l'extensibilité. Ces systèmes, avec leurs notions de macrocommande et de fonction, permettent à l'utilisateur de rapprocher l'interface de leur conception de la tâche.

La macrocommande est donc une technique qui répond convenablement aux besoins de spécificité et de généralité. Encore faut-il que la spécification des macrocommandes soit aisée! Chacun d'entre nous reconnaît la puissance d'expression du shell d'Unix mais chacun a dû faire face (ou renoncer) à la phase d'apprentissage. La non-interactivité du langage explique en partie cette difficulté. Dans Emacs, l'utilisateur définit une macrocommande par démonstration : dans le contexte "définition de macro", toute commande est immédiatement exécutée. L'utilisateur peut ainsi évaluer de manière incrémentale l'effet de la future macrocommande.

La construction par démonstration est une façon de raccourcir la distance sémantique qui sépare l'objet conceptuel à réaliser de l'expression de son élaboration. Cette technique, bien qu'attrayante, n'est pas toujours facile à mettre en œuvre. Dans Emacs, les constituants de la construction sont des commandes textuelles non ambiguës. Dans le cas des interfaces graphiques, la gestuelle est plus difficile à interpréter. Il est clair que des travaux méritent d'être entrepris dans cette direction.

3.3. Concision et les fonctions Couper-Coller

Couper-Coller est la version électronique de cette méthode manuelle de construction qui consiste à découper et à assembler des morceaux de documents. Le Couper-Coller électronique a deux raisons d'être : reproduire des informations et suppléer au manque d'intégration des outils de production d'informations.

La reproduction d'informations a été rendue possible dès les premiers éditeurs de texte. Certes, l'interface d'utilisation présentait alors une longue distance d'exécution et d'évaluation (voir le couple de commandes *kill buffer* et *Yank* d'Emacs). Avec l'arrivée des écrans graphiques et des dispositifs de désignation, cette fonction est devenue plus facile à utiliser et s'est étendue à d'autres logiciels que les éditeurs de texte. Il est aujourd'hui possible de construire un dessin avec un logiciel graphique,

d'écrire une formule mathématique avec un éditeur de formule [Quint 83] et de recopier les informations produites dans le texte d'un document. En réalité, cet exemple met en évidence le manque d'intégration des logiciels de production de documents.

Bien que des systèmes multimédia comme Tigre [Lopez 83], Grif [Quint 85] ou Athena [Balkovich 85] commencent à voir le jour, les postes de travail sont encore conçus comme des juxtapositions d'applications fermées. Chaque logiciel a sa spécialité avec, pour inévitable conséquence, des structures de données incompatibles avec celles des logiciels voisins. La technique pratiquée habituellement pour contourner cette difficulté consiste à laisser l'application réceptrice enrober l'information intruse d'un film protecteur : l'information est intégrée comme un tout et l'utilisateur n'a plus la possibilité d'en changer le contenu. Ce procédé cache un problème de fond non résolu (celui de la conversion de types que nous évoquerons au chapitre 10, paragraphe 2.5.2.) ; il est aussi source d'incohérences : par suite d'un transfert, une entité peut changer de comportement. Par exemple, l'aller-retour d'une entité "cercle" entre Macdraw et Macpaint transforme le cercle en une surface qui restitue un cercle mais dont les attributs de cercle ne sont plus accessibles à l'utilisateur.

3.4. Concision et valeurs par défaut

Les valeurs par défaut évitent les répétitions de saisie et proposent des valeurs sémantiquement correctes. Elles contribuent ainsi à réduire les erreurs de typographie et de choix sémantique. Elles se répartissent en deux catégories : les valeurs dynamiques et les valeurs semi-statiques.

Les valeurs par défaut dynamiques sont automatiquement réévaluées et mémorisées en cours de session. La figure 4.2 montre l'exemple d'une demande de sauvegarde de fichier. La valeur par défaut dynamique est le nom du fichier en cours d'édition. L'heuristique du concepteur repose ici sur l'observation qu'une sauvegarde de fichier s'effectue généralement sous le même nom ou sous un nom voisin.

Alors que les modifications des valeurs par défaut dynamiques sont décidées par le système, les valeurs par défaut semi-statiques ne varient qu'à la demande explicite de l'utilisateur. Les fichiers "profil", qui regroupent les options des applications, entrent dans cette catégorie. Les valeurs par défaut semi-statiques contribuent modestement mais simplement à la personnalisation du comportement des logiciels.

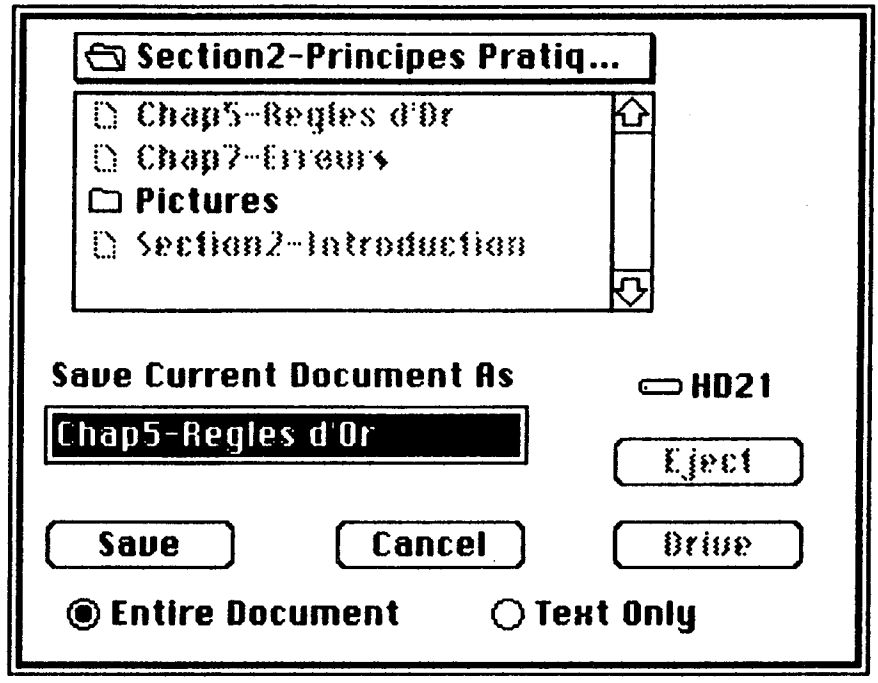


Figure 4.2 : Exemple de valeur par défaut dynamique : le nom du fichier que l'utilisateur éditait au moment de la spécification de la commande de sauvegarde.

3.5. Concision et fonctions Défaire-Refaire

Les commandes Défaire et Refaire, qui désignent implicitement la dernière commande exécutée, réduisent le nombre d'actions physiques. Elles interviennent directement dans la réparation des erreurs.

La fonction Défaire est un moyen de correction immédiate, simple et directe. Elle permet d'annuler l'effet d'une action involontaire ou erronée. Elle encourage par conséquent l'exploration et les tentatives incertaines. Dans la plupart des cas, la sémantique de Défaire s'entend comme la fonction qui place le système dans l'état immédiatement antérieur. Ainsi, appliquée à l'état du système, la suite "Défaire.Défaire" a l'effet de la fonction Identité. Cette définition est trop restrictive : elle ne répond pas au cas fréquent où l'erreur, parce qu'elle n'a pas été détectée immédiatement, ne peut plus être résorbée. Dans ce cas, la sémantique souhaitée est celle qui permet de "remonter le temps" dans l'espace des états. Soient e_0, e_1, e_2, ..., e_n, les n+1 derniers états du système enregistrés dans cet ordre et notons Défaire^i, i applications successives de Défaire, alors :

Défaire(e_n) $= e_{n-1}$,

Défaire$^2(e_n)$ $= e_{n-2}$. et

Défaire$^n(e_n)$ $= e_0$.

Cette définition de Défaire permet de remonter dans l'espace des états mais ne permet pas de refaire ce qui a été défait. A cette fin, le système doit pouvoir proposer une fonction Refaire telle que si $e_0, e_1, e_2, ..., e_n$ sont les n+1 derniers états du système enregistrés dans cet ordre puis défaits dans l'ordre inverse, alors :

$\text{Refaire}(e_0) = e_1$,
$\text{Refaire}^2(e_0) = e_2$, et
$\text{Refaire}^n(e_0) = e_n$.

Lieberman dans [Lieberman 87] donne une application intéressante de cette navigation dans l'espace des états : les "interpréteurs réversibles". Ces interpréteurs, capables de mémoriser les valeurs successives des expressions, sont applicables à la mise au point de programmes.

La mise en œuvre des fonctions Défaire et Refaire repose sur l'existence d'un service d'historique sophistiqué. Concrètement, historique et fonctions Défaire/Refaire appartiennent au catalogue des vœux pieux de l'informaticien bien intentionné. Les services actuels d'historique ne recueillent qu'une vue partielle de l'espace de travail de l'utilisateur. Par exemple, sous Unix, l'historique ne mémorise que la suite des commandes reconnues par le shell ; il ne reconnaît ni les touches élémentaires ni les commandes du programme lancé à partir du shell. L'historique n'est pas une fonction locale. Il est omniprésent, c'est-à-dire réparti à tous les niveaux d'abstraction du système. Il concerne donc l'ensemble des variables physiques de l'interaction. Chaque variable devrait être capable de mémoriser sa propre évolution et contribuer à l'histoire générale par le biais des relations structurelles entre variables physiques. Nous verrons au chapitre 8 l'apport du modèle PAC à la mise en œuvre de la répartition.

4. Règle sur les retours d'information

Un retour d'information (en anglais, feedback) est une réaction du système aux actions de l'utilisateur. Cette réaction se manifeste dans l'Image sous forme d'expressions de sortie dont l'interprétation permet d'évaluer la situation, de la comparer au but recherché, puis, selon le cas, de poursuivre ou modifier le plan d'actions. Le retour d'information a donc la responsabilité de refléter l'état du système, non pas l'état interne au service de la logique du logiciel, mais l'état qui répond à l'attente de l'utilisateur, voire anticipe ses intentions. Dans cet état, les variables psychologiques, centres d'intérêt de l'utilisateur, doivent trouver leurs correspondants dans les variables physiques de l'Image. Si cette condition est respectée, le retour d'information est dit immédiat et informatif.

L'attente ou les intentions de l'utilisateur peuvent être satisfaites de diverses façons. Dans ce paragraphe nous en retenons trois : informer pour rassurer, informer pour réduire la charge cognitive et informer pour remédier aux erreurs.

4.1. Informer pour rassurer

Certaines commandes activent des fonctions du système dont le temps de réponse est long, c.-à-d. au delà de quelques secondes. Par exemple, une compilation ou un transfert de fichier ne s'effectuent généralement pas de manière instantanée. Dans le cas de la compilation, l'attente est simplement gênante (le compilateur devrait être plus rapide), mais dans le cas d'opérations à risque, comme le transfert de fichiers sur un réseau local, l'attente vire le plus souvent à l'inquiétude (le fichier n'est peut-être que partiellement transmis).

Que l'opération soit dangereuse ou non, l'utilisateur a besoin d'être informé sur l'état interne de la boîte noire. Les figures 4.3 et 4.4 montrent deux exemples de retour d'information œuvrant dans ce sens. Dans la figure 4.3, l'Image montre l'évolution d'un entier, variable physique isomorphe à une variable psychologique d'un utilisateur type pendant la recopie de fichiers.

Figure 4.3 : Message affiché par le système de gestion de fichiers du Macintosh lors de la copie de fichiers. A tout instant, l'utilisateur est informé du nombre de fichiers restant à traiter.

La figure 4.4 est l'exemple presque parfait de message informatif. On y trouve la représentation des variables psychologiques impliquées dans la tâche de transfert de fichiers entre calculateurs, à savoir l'identité et la taille du fichier, le pourcentage d'octets déjà transférés et le temps prévu pour achever le transfert. Notez l'indication des unités de mesure. La progression de la barre de pourcentage et l'évolution du temps estimé permettent à l'utilisateur d'évaluer approximativement les conditions de transfert. Si ces conditions sont jugées inacceptables, la commande peut être annulée. Le message anticipe ce besoin en indiquant comment arrêter l'opération. La forme du suiveur, un sablier, indique que la situation d'attente est normale. Informé de la progression de l'opération, l'utilisateur est en mesure d'évaluer objectivement la situation. Il ne sera pas tenté d'interrompre un transfert de fichier jugé à tort anormal sous le seul prétexte de la lenteur apparente de l'opération.

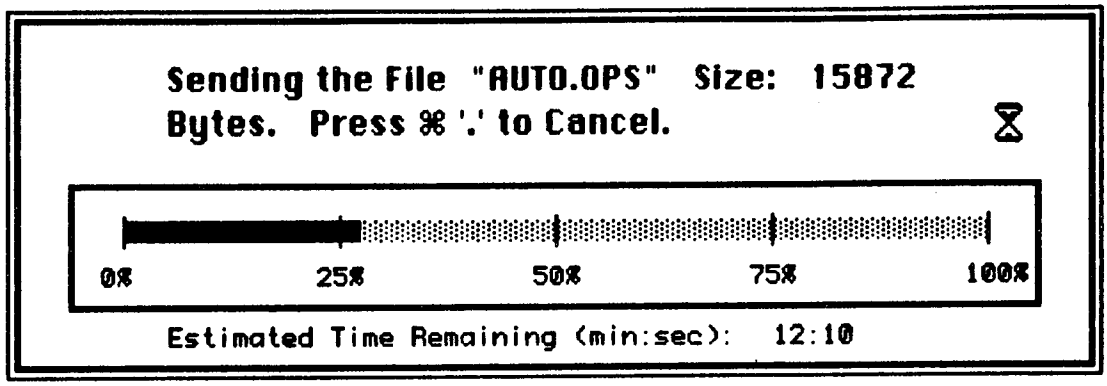


Figure 4.4 : Le message de Versaterm3 sur Macintosh pendant le transfert d'un fichier vers un calculateur distant.

4.2. Informer pour réduire la charge cognitive

Au cours d'un raisonnement, il est fréquent de noter par écrit des résultats intermédiaires. Nous fabriquons ainsi une extension physique de notre mémoire à court terme pour en réduire la charge cognitive. L'ordinateur, dont la mémoire est infaillible (aux pannes matérielles près), a un rôle essentiel à jouer comme extension de la mémoire à court terme. En particulier, il lui est possible de rappeler à l'utilisateur son contexte de travail, de l'aider à se situer dans son espace de travail et de lui indiquer les options possibles.

4.2.1. Retour d'information et rappel du contexte de travail

Il nous arrive fréquemment de changer mentalement de contexte de travail mais d'oublier de le signifier au système. Cette forme d'oubli est quelquefois liée à la fatigue mais se manifeste généralement lorsque l'état du système n'est pas suffisamment explicite.

Pour les systèmes organisés en états, la solution généralement admise est l'association d'une forme de suiveur ou de curseur à chaque mode d'interprétation des actions d'entrée. Dans ces conditions, le suiveur et le curseur s'imposent visuellement à l'utilisateur et signalent de manière permanente l'état interne du système. De nombreux logiciels exploitent depuis longtemps cette technique. Encore faut-il que les formes choisies expriment clairement la sémantique de l'état actuel ou mieux, qu'elles puissent être personnalisées sans programmation ni recompilation de l'application!

Pour les systèmes qui supportent le multifenêtrage et les activités multiples, aucune solution n'est vraiment satisfaisante. Du point de vue de l'utilisateur, le partage du poste de travail repose sur la notion de fenêtre courante. La fenêtre courante est celle qui reçoit toutes les actions d'entrée jusqu'à ce que l'utilisateur désigne une autre fenêtre. La présentation de la fenêtre courante repose à peu près toujours sur la même technique qui consiste à la mettre en évidence par un cadre particulier. Les variations entre systèmes se manifestent essentiellement sur le changement de profondeur de la

fenêtre. Dans certains systèmes comme le Macintosh, la fenêtre courante passe au premier plan. Pour d'autres, comme Sapphire [Myers 84], la fenêtre courante conserve sa place dans l'empilement. Dans le premier cas, le mouvement visuel causé par le changement de plan peut paraître gênant ou au contraire peut servir de signal de mise en garde. Dans le second cas, l'écran est stable mais induit des oublis de changement de fenêtre. Le choix entre les deux techniques relève du compromis.

4.2.2. Retour d'information et navigation

Les retours d'information peuvent aider l'utilisateur à se situer dans son espace de travail. Dans un espace linéaire, comme celui des systèmes de traitement de texte traditionnels, l'utilisateur n'a que deux degrés de liberté : en avant ou en arrière par rapport à la localisation actuelle. La représentation des déplacements et de la localisation s'effectue selon des techniques que l'on croît bien maîtriser : les commandes de recherche et les barres de défilement. Les exemples de la figure 4.5 montrent que les barres de défilement n'ont pas toujours la qualité informative requise.

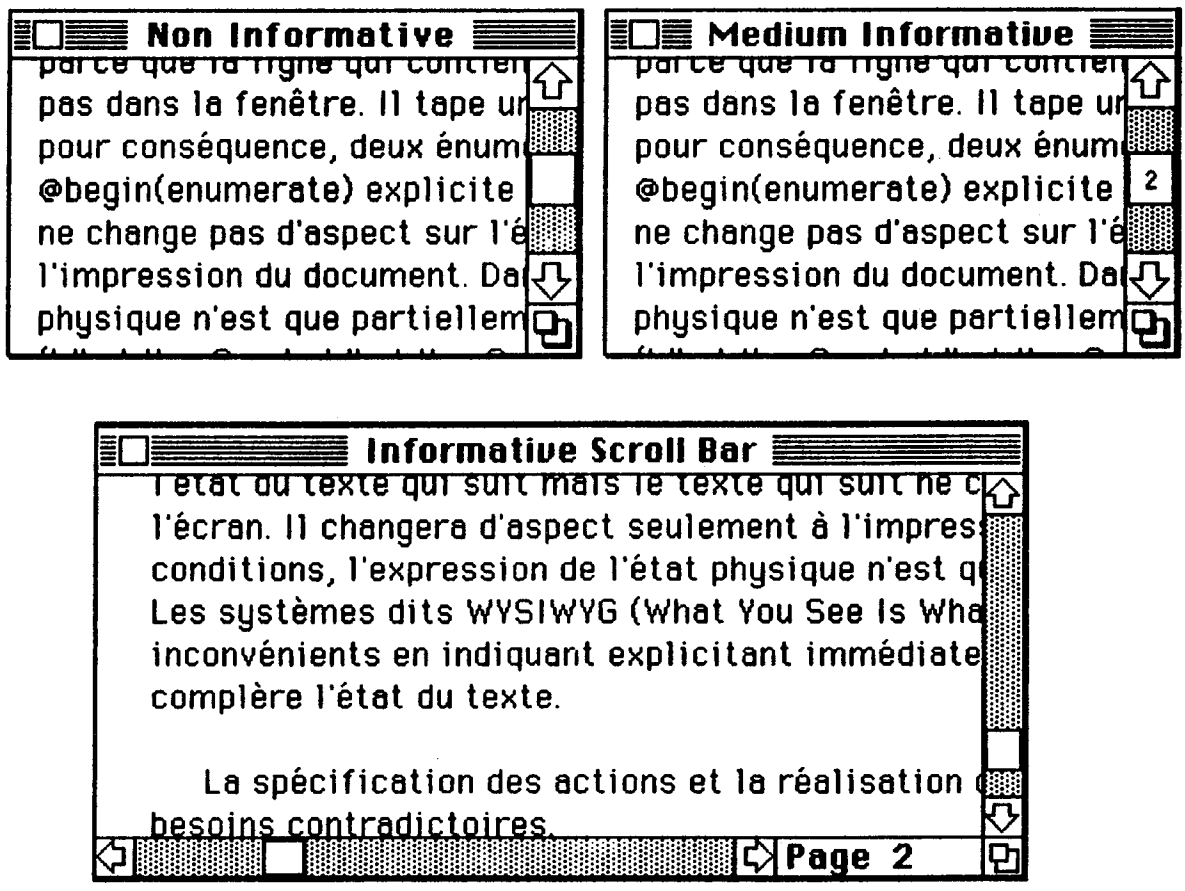


Figure 4.5 : Qualité informative de trois barres de défilement. En haut à gauche, la barre de défilement ne fournit aucune indication sur le numéro de page actuellement visible dans la fenêtre. En haut à droite, l'utilisateur sait à quelle page il se trouve à condition qu'il ait fait le lien entre le numéro affiché dans l'ascenseur de la barre et le numéro de page du document. En bas, la barre de défilement idéale : l'utilisateur même débutant sait qu'il se trouve en page 2.

Dans les systèmes où l'espace de navigation est vaste et complexe, la navigation pose un problème non résolu. Les systèmes dits "hypertextes" appartiennent à cette catégorie [Conklin 87]. L'utilisateur d'un tel système peut créer et modifier de manière interactive des liens entre des entités. Ces entités constituent les noeuds d'un réseau sémantique et servent à représenter "n'importe quoi" : une idée, une image, une procédure de programme, un morceau musical. L'idée directrice des systèmes hypertextes est de donner à l'utilisateur les moyens d'organiser et de communiquer ses pensées et ses connaissances. L'idée n'est pas nouvelle : les systèmes d'aide et les "browsers" en ont été les premières applications.

Aujourd'hui, on parle de technologie hypertexte dont NoteCards [Halasz 87, Trigg 87] symbolise la tendance. Malheureusement, le potentiel prometteur de cette technologie se trouve actuellement limité par des problèmes cruciaux de navigation. Dans un réseau de 1000 noeuds, il devient difficile de retrouver une information ou de se souvenir de l'existence d'une information. Il est clair que l'avenir de ces systèmes est lié à l'existence de techniques graphiques qui permettraient de produire des représentations graphiques souples capables de maîtriser la complexité. Aucune technique actuelle (méthodes holophrastiques, représentations en trois dimensions) n'offre de solution satisfaisante.

4.2.3. Retour d'information et présentation des options

La présentation des options résulte du compromis entre la visibilité, la validité et la concision. La visibilité définit la présence de l'option dans l'Image pour que l'utilisateur en apprenne l'existence. La validité concerne la pertinence de l'option dans le contexte actuel pour que l'utilisateur sache s'il a le droit de la manipuler. La concision vise à réduire la surcharge cognitive sans perte d'information. Ces trois conditions apparemment contradictoires sont cependant conciliables.

La condition de visibilité est satisfaite en présentant dans l'Image toutes les options ; l'expression de l'invalidité est satisfaite en décorant les options invalides d'attributs lexicaux pertinents, par exemple l'affichage en grisé clair et l'absence de réaction aux actions de l'utilisateur ; la concision est obtenue en regroupant les options dans des menus déroulants ou des menus fantômes qui ont l'avantage de n'apparaître qu'à la demande. Si ces menus comportent trop d'options, ils peuvent à leur tour faire l'objet d'une décomposition hiérarchique.

L'exemple de la figure 4.2 illustre la mise en application de ces techniques : le système indique le choix entre effectuer ou annuler la sauvegarde, changer de support physique, accepter ou surcharger la valeur par défaut. Si la valeur par défaut doit être changée, la liste des noms déjà utilisés peut être consultée dans un menu déroulant afin d'éviter une double définition. Le bouton *Eject* présenté en grisé clair et insensible aux actions de la souris, indique que la fonction de déchargement de disquette est actuellement invalide.

4.3. Informer pour la détection des erreurs et les remèdes

Les erreurs sont inévitables mais le système peut en faciliter la détection, la détermination et l réparation.

- La détection de l'erreur est déclenchée par le caractère immédiat du retour d'information.
- La détermination de la cause dépend de la qualité informative de l'Image. Les messages, tel "Erreur de syntaxe" ou "Turkey!" (extrait de Gosling-Emacs), ne contribuent certainement pa au succès de la collaboration entre l'utilisateur et le système. Le message de la figure 4.6 qu explicite la cause de l'erreur et qui fournit les indications utiles au remède, est au contraire u bon exemple de retour d'information.
- Le remède à l'erreur dépend des fonctions du système, et notamment de l'existence de commandes Défaire et Refaire.

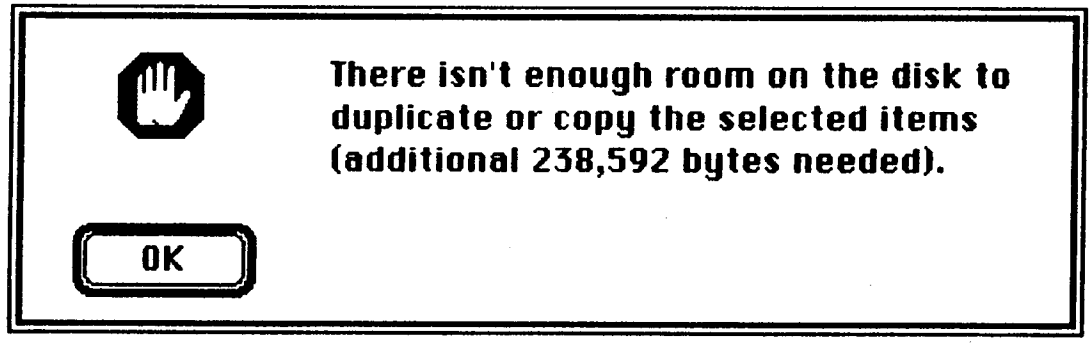


Figure 4.6 : Un message d'erreur informatif qui comprend une explication claire du problème (il n'y a pas assez de place pour effectuer la copie) et un renseignement complémentaire (le nombre d'octets manquants) utile à la mise en place d'un plan de résolution.

On conçoit aisément que l'absence de retour d'information conduise l'utilisateur à spécifier des commandes illégales, à oublier le mode d'interprétation des entrées ou à avoir des difficultés à identifier la cause d'une erreur. L'absence de retour d'information peut aussi conduire l'utilisateur à inférer des faits inexacts que l'Image renforce par l'absence de contradiction.

5. Règle sur la structuration des activités

Structurer les activités consiste à organiser l'espace de travail de l'utilisateur en accord avec ses compétences. Dans ce but, Carroll propose la méthode des "roulettes de sécurité" (en anglais, training wheels) qui consiste à grouper les fonctions d'un système en niveaux de complexité [Carroll 83, Carroll 84, Carroll 85a]. Cette organisation peut se pratiquer selon deux axes : le fond et la forme.

5.1. La technique des "Roulettes de sécurité"

Dans la vie réelle, on fixe des roulettes de part et d'autre de la roue arrière des vélos afin que les enfants évoluent sans risque de chute. On retire les roulettes dès que l'enfant maîtrise l'équilibre.

En interaction homme-ordinateur, on peut appliquer une technique similaire : laisser l'utilisateur "apprendre en faisant" mais sans échec. De même que l'enfant peut d'emblée faire du vélo sans risque de blessure, de même l'utilisateur doit pouvoir réaliser dès le départ des tâches simples sans erreur. Pour cela, Carroll propose d'organiser les fonctions du système en niveaux de complexité croissante. Chaque niveau définit des garde-fous cognitifs. Lorsque l'utilisateur a compris toutes les fonctions d'un niveau, il a droit au niveau supérieur, et ceci jusqu'au libre accès à toutes les fonctions du système. Les résultats analytiques obtenus par Carroll sont assez convaincants [Carroll 84].

5.2. Structurer le fond

Sur le fond, structurer l'interaction c'est définir une hiérarchie d'environnements. Chaque environnement est un contexte qui se caractérise en particulier par l'ensemble des fonctions accessibles. Au sommet de la hiérarchie, l'environnement initial, le plus simple possible. Au bas de la hiérarchie, les environnements les plus complexes. La difficulté est d'allier fréquence et simplicité. Il n'est pas certain que les fonctions fondamentales du système, c'est-à-dire les fonctions les plus fréquemment utilisées, soient, du point de vue cognitif, les plus simples. Le concepteur doit alors user de compromis pour structurer l'espace des commandes.

Structurer l'espace des commandes consiste à définir la granularité des commandes. Cette activité de conception n'est pas une tâche simple : si le niveau d'abstraction est trop fin, l'utilisateur doit prendre en charge la composition des commandes pour accomplir les tâches élémentaires. Si le niveau est trop élaboré, les tâches usuelles s'expriment directement mais l'utilisateur risque de rencontrer de graves difficultés pour l'expression des tâches qui ne se décomposent pas dans les termes du langage de commande.

En interface homme-ordinateur comme en génie logiciel, une structuration correcte résulte d'une analyse approfondie du problème et d'une compétence qui permet de repérer les indices révélateurs d'anomalies. Concernant les commandes et la charge cognitive qu'elles impliquent, on peut utiliser comme détecteur d'anomalies le nombre d'options : si ce nombre est nettement supérieur à la capacité de la mémoire à court terme, alors il faut chercher à raffiner la commande ou, si la logique de l'application n'autorise pas la décomposition, il faut jouer sur la forme en rendant les options explicites.

5.3. Structurer la "forme"

La structuration de la forme intervient partout où il est nécessaire d'organiser la complexité visuelle, de réduire la charge cognitive tout en restant informatif. Elle concerne la présentation des fonctions, des menus et des messages.

5.3.1. Présentation des fonctions

l'Image doit à la fois mettre en évidence les fonctions usuelles simples et voiler (sans toutefois les cacher) celles qui le sont moins. Lorsque la logique du système le permet, toutes doivent être assorties de valeurs par défaut raisonnables. Dans ces conditions, l'utilisateur débutant est prompt à employer telles quelles les fonctions qui s'imposent dans l'Image. D'emblée, il se trouve capable de réaliser les tâches les plus courantes et ceci à moindre effort. L'expérience aidant, l'utilisateur va se définir des objectifs plus exigeants qui demanderont la recherche de fonctions adaptées, celles précisément que le système voilait. Les Images des systèmes MacPaint et HyperCard [Harvey 88] appliquent cette technique.

- L'Image de MacPaint montre, au moyen de deux palettes, le minimum d'informations pour construire un dessin correct. L'une propose les formes de tracés géométriques, les outils de dessin (crayon, pistolet, etc.), et les épaisseurs de trait ; l'autre montre l'éventail des couleurs. Les fonctions, utiles à l'expert mais non fondamentales pour le novice, ne sont pas immédiatement explicitées par l'Image. Par exemple, la fonction de fabrication d'une couleur n'est pas directement visible : elle est activée par sélection d'un élément de menu ou bien par un double-clic sur un élément de la palette des couleurs.

- Le système HyperCard est organisé en une hiérarchie de trois environnements de complexité croissante : navigation, création sans programmation et programmation. Cette structure du fond est complétée par l'Image qui élimine de la présentation les fonctions inaccessibles dans l'environnement actif et qui voile l'accès au formulaire de changement de mode de travail. En mode navigation, l'utilisateur peut parcourir sans risque la base de piles de cartes. C'est le mode initial par défaut. Avec l'expérience, l'utilisateur viendra à découvrir dans l'Image la possibilité de changer de mode. Il faut supposer que l'utilisateur a alors assez d'expérience pour s'engager dans la modification de la base d'informations.

Cette description un peu idéaliste du processus d'évolution de l'utilisateur, est souvent contredite par la réalité. Rosson constate que les utilisateurs même expérimentés se limitent souvent dans leur routine à un sous-ensemble des possibilités du système [Rosson 83]. Le remède pourrait être la présence d'un observateur logiciel qui, analysant les commandes de l'utilisateur, serait capable d'inférer les objectifs recherchés, de détecter les maladresses et de proposer, sans les imposer, de nouvelles solutions.

5.3.2. Structuration des menus

Lorsqu'un menu est trop large (il comporte trop d'éléments), l'utilisateur a des difficultés à

reconnaître l'option qui correspond à l'objectif. Le menu doit alors être organisé en une hiérarchie de sous-menus. Lorsque le menu est trop profond (il comporte trop de niveaux de décomposition), l'utilisateur a des difficultés à se situer. Les observations expérimentales semblent converger vers les résultats suivants [Miller 81, Kiger 84, Tullis 85] :

1. les menus équilibrés permettent de meilleures performances,

2. si un compromis doit être effectué entre la largeur et la profondeur, préférer la largeur.

A l'exception des systèmes d'exploitation qui comprennent entre 500 et 1000 commandes, les applications usuelles devraient pouvoir se satisfaire d'une profondeur 3. En tout état de cause, quelle que soit la profondeur d'un menu, il est bon que l'utilisateur puisse facilement déterminer sa position dans la hiérarchie. La figure 4.7 montre un exemple de menu hiérarchisé remplissant cette condition : toute entrée de sous-menu est marquée d'un petit triangle et le sous-menu est collé à l'entrée mère.

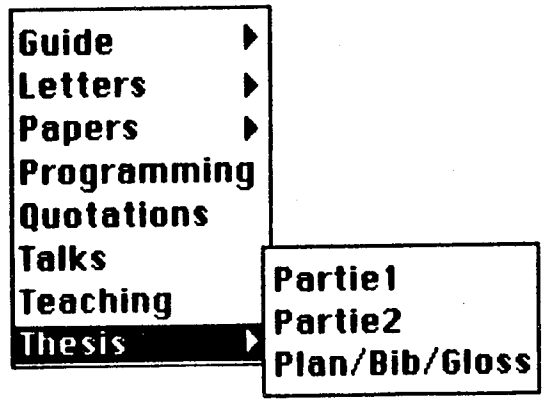


Figure 4.7 : Hiérarchie de menus dans le système Interlisp. Les entrées Guide, Letters, Thesis, marquées d'un petit triangle, sont des en-têtes de sous-menus. Pour faire apparaître le sous-menu de Thesis, l'utilisateur a placé la souris sur le triangle de l'entrée Thesis puis en est sorti par la droite.

5.3.3. Structuration des messages

Les messages posent deux problèmes : celui de la qualité informative et celui de la concision. Laconique et informatif pour l'expert, un message est imprécis et sibyllin pour le débutant. S'il éclaire la compréhension du débutant, il vient encombrer l'espace de travail de l'expert. Ou encore, pour un utilisateur donné, le niveau de détail requis peut varier avec les préoccupations du moment.

La conciliation entre la concision et le détail peut se résoudre en organisant les messages en niveaux de raffinement et en donnant à l'utilisateur le moyen d'accéder au degré de détail voulu. Cette solution est acceptable à condition que l'utilisateur ne puisse se perdre dans le labyrinthe de la hiérarchie (ou du réseau) [Robertson 81]. Nous verrons au chapitre 9 comment le système expert IRENE a su concilier concision et précision.

6. Règle sur la flexibilité

La flexibilité d'une interface désigne sa faculté d'ajustement aux variations de l'environnement, et notamment à l'utilisateur. Elle peut être automatique (on parle alors d'interface adaptative) ou manuelle (on parle alors d'interface adaptable).

Une interface adaptative repose sur une modélisation dynamique de l'utilisateur. Elle reprend les lois des théories cognitives et s'inspire généralement des techniques de l'intelligence artificielle. Pour cette raison, on parle également d'interface intelligente. A l'heure actuelle, les interfaces adaptatives n'en sont qu'à leurs balbutiements.

Une interface est adaptable lorsqu'elle est modifiable sur intervention explicite de l'utilisateur. L'adaptabilité n'implique pas forcément la modification radicale, par ailleurs discutable, de l'interface mais une personnalisation raisonnable qui garantit la cohérence de la présentation et facilite l'accomplissement des tâches. Dans ce paragraphe, nous avons retenu quatre éléments en faveur des interfaces adaptables.

6.1 Flexibilité et réparations lexicales

Le lexique d'un système est flexible lorsqu'il est modifiable sans qu'il soit nécessaire d'altérer le logiciel source. L'intérêt de cette possibilité est d'adapter la terminologie, de corriger les incohérences, de reformuler les énoncés des messages, de redéfinir les raccourcis, de surcharger l'association fonctionnelle des touches de la souris et du clavier, etc., sans l'intervention d'un concepteur ou d'un informaticien. Cet ajustement n'est possible que si les éléments lexicaux sont des valeurs permanentes externes non pas, comme c'est encore trop souvent le cas, des constantes "câblées" dans le programme.

Les "alias" et les fichiers de commande du système Unix, de même que les fichiers de ressources du Macintosh, vont dans cette direction. Nous verrons au chapitre 9 comment le système IRENE produit des messages dynamiques à partir d'un lexique externe.

6.2. Flexibilité et valeurs par défaut

Les valeurs par défaut semi-statiques constituent avec les lexiques externes une seconde façon simple de personnaliser le comportement des logiciels. Les fichiers de profil entrent dans cette

catégorie.

6.3. Flexibilité et choix de l'initiative du dialogue

Laisser l'initiative à l'utilisateur est une règle à observer avec circonspection. Lorsqu'il est incompétent, l'utilisateur va préférer le guide rassurant des demandes du système. Dans la situation inverse, le système doit proposer et l'utilisateur doit pouvoir disposer. Dans ce mode, l'utilisateur ne veut pas être modélisé comme un fichier séquentiel.

Modéliser l'utilisateur comme un fichier séquentiel c'est avoir ordonné a priori les actions possibles de l'utilisateur. Ce procédé, bien qu'il soit commode pour les informaticiens, tue par avance l'élaboration de plans dynamiques non linéaires, plans dans lesquels l'utilisateur, pour accomplir une tâche, commence par se diriger vers un sous-but, met de côté ce début de résolution, entreprend de se rapprocher vers d'autres sous-buts, récupère les résultats partiels d'un chemin pour reprendre la résolution du premier sous-but, etc. Il est fréquent par exemple qu'un système, faisant suite à une commande de sauvegarde, demande de spécifier un nom de fichier. L'utilisateur, qui ne se souvient pas des noms des fichiers existants, devra se détourner temporairement de son objectif pour obtenir une liste des fichiers. Dans un système où l'utilisateur est un fichier séquentiel, l'opération de sauvegarde n'autorise pas de déviation et l'utilisateur se trouve contraint de spécifier un nom de fichier coûte que coûte. Parfois il en coûte cher!

Nous verrons avec le modèle PAC exposé aux chapitres 8 et 9, comment un logiciel d'interaction doit prendre en compte la liberté d'action de l'utilisateur.

6.4. Flexibilité et représentations multiples

Selon la tâche, il arrive qu'une variable psychologique ait besoin d'être concrétisée sous plusieurs formes dans l'Image. Chaque forme de présentation a son utilité. Par exemple, au cours du tracé interactif d'un rectangle, l'Image peut suggérer les contours de la figure à l'aide d'une forme élastique et montrer les valeurs exactes des dimensions à l'aide de deux entiers. Les deux représentations ont un rôle informatif complémentaire : le dessin concrétise et les nombres précisent. Dans le cas d'un document, l'utilisateur doit pouvoir consulter la table des matières et un chapitre particulier, l'une permettant par exemple, d'évaluer la structure générale du texte, l'autre permettant par exemple, de vérifier les détails d'une phrase.

De manière générale, chaque représentation a ses qualités expressives et sert d'instrument à la réflexion. A ce propos, Edward Tufte énonce dans un ouvrage passionnant [Tufte 83], les propriétés et les règles de conception des représentations visuelles des données numériques. Ces représentations doivent conduire l'observateur à raisonner sur la substance des données ; elles doivent être capables de présenter d'énormes ensembles de données sur une petite surface et encourager l'œil à faire des comparaisons, etc. Cette excellence n'est pas le fruit d'un art intuitif mais d'une technique. Cette technique nous enseigne en particulier qu'une représentation doit conserver l'intégrité des données. La figure 4.8 illustre un cas fréquent de "mensonge" où un motif, ici un carré et du texte, ne croît pas dans les mêmes proportions que les données numériques.

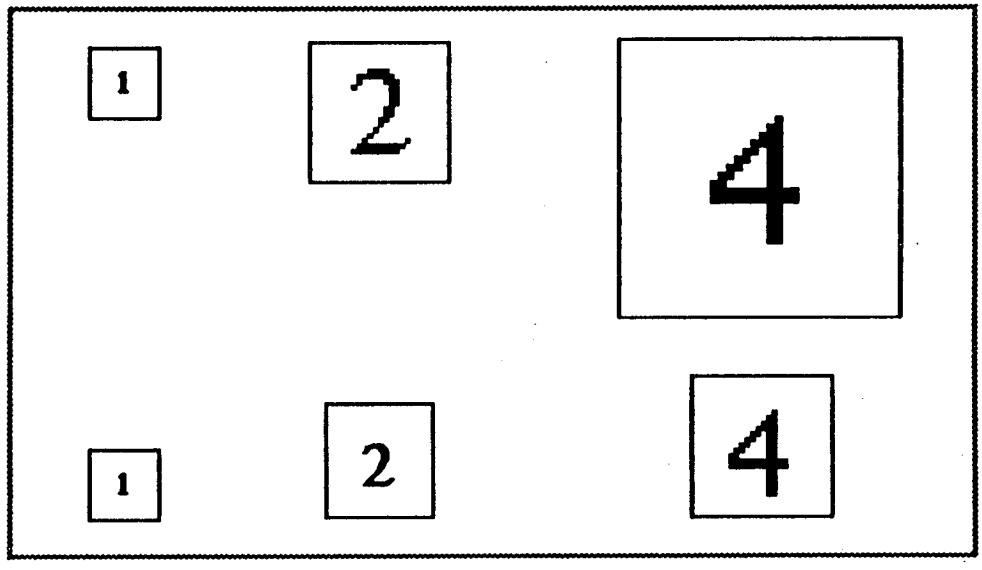


Figure 4.8 : En haut, une représentation mensongère de données numériques : les côtés des carrés sont égaux aux données numériques donnant par l'effet de la surface une impression inflationiste ; en bas, une représentation qui préserve l'intégrité des données : la surface d'un carré et la taille d'un caractère sont égales à la donnée numérique qu'elles représentent.

On distingue deux techniques de base pour les représentations multiples. La première consiste à associer plusieurs objets de présentation à un concept donné. Chaque objet a sa puissance d'expression et vient compléter la compétence des autres. Nous verrons aux chapitres 8 et 9 comment le modèle PAC permet l'association multiple. La seconde technique consiste à produire des variations sur une forme de base unique. Le chapitre 11 sur l'affichage abstrait présentera les principes de mise en œuvre de ce procédé.

EN RESUME

Les cinq règles ici présentées contribuent toutes à améliorer l'usage des systèmes informatiques ; elles visent à limiter l'occurrence des erreurs et à faciliter leur réparation ; elles résultent d'observations expérimentales et ne dérivent généralement pas des propriétés d'un modèle prédictif de l'utilisateur.

La cohérence permet à l'utilisateur d'élaborer un modèle mental débarrassé des cas d'exception. Une métaphore d'interaction définit le caractère unitaire de l'Image dont le comportement général doit respecter les quatre principes suivants :

- à but identique, séquence de commandes identiques,
- à arguments identiques, ordre de spécification identique,
- à sémantique identique, dénomination identique,
- à utilisation identique, localisation identique.

La concision vise essentiellement à réduire le nombre des actions physiques et avec elles, les erreurs typographiques. Elle revêt quatre formes :

- les abréviations qui doivent être dérivables selon des règles précises,
- les macrocommandes, technique d'abstraction qui répond au mécanisme de construction de mnèmes,
- les fonctions Couper-Coller pour la réutilisation d'informations,
- les valeurs par défaut comme extensions de la mémoire à court terme,
- les fonctions Défaire-Refaire pour les réparations physiques et cognitives des erreurs.

Les retours d'information doivent être

- immédiats : le système doit réagir à toute action ou suite d'actions pertinentes pour l'utilisateur,
- informatifs : l'Image doit montrer l'état des variables physiques en correspondance avec les variables psychologiques d'intérêt.

La structuration des activités consiste à organiser la complexité du système. Elle concerne

- le fond qui doit, si possible être structuré en niveaux de complexité croissante, chaque niveau définissant un garde-fou cognitif,

- la forme qui met en évidence les fonctions simples mais qui voile les fonctions complexes, qui propose d'afficher les messages au niveau de détail voulu, et qui équilibre la largeur et la profondeur des menus.

La flexibilité recouvre l'idée d'adaptabilité. Elle peut être automatique (c'est une interface adaptative) ou manuelle (c'est une interface adaptable). On doit pouvoir

- adapter le lexique sans qu'il soit nécessaire d'altérer le logiciel. Il faut pour cela que les données lexicales soient des entités externes permanentes,
- modifier les valeurs par défaut semi-statiques qui offrent une façon simple de personnaliser le comportement du système,
- laisser l'utilisateur maître de l'initiative du dialogue (par exemple, ne pas le modéliser comme un automate),
- permettre la représentation multiple d'un même concept.

Chapitre 5

Méthodes de Conception

1. Introduction

Ce chapitre résume l'apport des sciences cognitives à la conception des systèmes interactifs et s'interroge sur les méthodes possibles.

Le paragraphe suivant évalue la contribution des théories et des modèles présentés en prenant le point de vue de l'informaticien. Nous constatons au paragraphe 3 l'absence d'une méthode de conception. Au paragraphe 4, les principes directeurs d'un outil de description, CLG, nous servent à identifier les grandes étapes de la conception. Nous concluons en constatant que les étapes de conception sont marquées par des décisions déterminantes pour la qualité de l'interaction et pour lesquelles les outils actuels n'offrent aucun support.

2. Evaluation de l'apport des sciences cognitives

Une théorie permet d'expliquer un phénomène ou d'en prédire l'occurrence. Dans le domaine de la psychologie cognitive, il n'existe pas de théorie explicative ou prédictive unique qui embrasse tous les aspects du sujet humain. Il existe des théories se rapportant chacune à des phénomènes précis : théories sur la résolution de problèmes, théories sur les représentations mentales, théorie sur l'apprentissage, théorie des langages, etc.

Chaque théorie sensibilise l'informaticien à une classe particulière de concepts et de processus cognitifs, mais la dispersion a comme résultat une somme impressionnante d'enseignements partiels difficilement organisables dans un cadre de pensée homogène. Cette situation est probablement tolérable pour le cogniticien qui étudie un aspect précis du sujet humain. Elle ne l'est plus dès l'instant où l'étude porte sur le sujet humain dans sa totalité comme c'est le cas pour l'informaticien. Ce dernier doit concevoir un système, non pas pour un sous-ensemble de l'utilisateur, mais bien pour un individu à part entière! De toute évidence, il n'existe pas de théorie adaptée à la conception des interfaces homme-ordinateur quand bien même le modèle du Processeur Humain et la théorie de l'Action de Norman constituent des étapes marquantes dans cette direction.

Le modèle du Processeur Humain voudrait se rapprocher des avantages d'une théorie technique [Newell 86] en formalisant les performances du sujet humain à l'aide de paramètres et de lois mathématiques. Malheureusement, la loi de Fitts et les notions de cycle et de capacité concernent des phénomènes de comportement de bas niveau. Ils ne fournissent aucune indication sur la nature des représentations mentales (en particulier sur la conceptualisation et la reconstitution mnésiques) ; ils

n'expliquent pas les mécanismes d'apprentissage ; ils passent sous silence la résolution de problèmes. Le modèle se contente de présenter très succintement la résolution de problème (accomplir une tâche avec un ordinateur, c'est effectivement une résolution de problème) en introduisant des règles et des principes opératoires. J'ai retenu le principe de rationalité qui sert de base au modèle GOMS.

L'idée centrale de GOMS et qui présente un intérêt en matière de conception est la décomposition d'une tâche en une hiérarchie de buts et de sous-buts jusqu'à ce que les sous-buts soient atteints par l'exécution d'opérateurs élémentaires. Tout comme le processeur humain, GOMS est extrêmement réducteur. Il ne décrit qu'un aspect limité des mécanismes cognitifs : celui de l'accomplissement de tâches de routine sans aucune possibilité d'erreur ou d'interruption. Cette vue simplifiée qui s'attache essentiellement aux performances est complétée par la perspective cognitiviste de la Théorie de l'Action.

La Théorie de l'Action de D. Norman [Norman 86] complète en plusieurs points les modèles de performance GOMS et du Processeur humain. D'une théorie à l'autre, nous retrouvons la décomposition hiérarchique d'une tâche en buts et sous-buts jusqu'à ce que les sous-buts correspondent à des unités cognitives procédurales, un savoir-faire compilé. Alors que GOMS se contente de définir un but "comme une structure symbolique qui décrit un état à atteindre", la Théorie de l'Action précise la notion d'état. Elle distingue l'état effectif de l'état perçu. L'état effectif est une fonction portant sur des variables physiques, caractéristiques du modèle conceptuel du système, tandis que l'état perçu est la traduction de l'état effectif sous forme de variables psychologiques, caractéristiques de la représentation mentale de l'utilisateur.

La différence de représentations entre le monde physique et le monde mental met en évidence la nécessité, pour l'utilisateur, d'effectuer des traductions. Contrairement à GOMS qui a une vue synthétique du comportement, la Théorie de l'Action analyse le processus de traduction qui sous-tend le comportement. Cette analyse identifie des phases auxquelles correspondent des besoins. Lorsque les besoins ne sont pas ou sont mal satisfaits, il y a risque d'erreur. Tandis que GOMS se cantonne au cas idéal de l'utilisateur infaillible, la Théorie de l'Action choisit le parti réaliste d'identifier les possibilités d'erreur et d'expliquer les difficultés rencontrées par l'utilisateur.

La Théorie de l'Action est un modèle explicatif du comportement. Elle nourrit l'esprit du concepteur-informaticien d'un schéma de pensée utile à la mise en évidence des points clés de l'interaction. Malheureusement, son enseignement s'effectue sans support formel. La description un peu intuitive de la Théorie de l'Action se voit complétée par la formalisation de ACT*.

ACT* [Anderson 83] est une première réponse à l'absence de cohésion entre les diverses théories

cognitives, à l'excès de simplification des phénomènes, aux modélisations réductrices et au manque de support formel. Cette théorie, qui s'appuie sur le formalisme et le principe des systèmes de productions, s'attache à la représentation de la connaissance, à la reconstitution mnésique, à la résolution de problème et à l'apprentissage (des langages en particulier). ACT* permet de construire des modèles de simulation exécutables. Des modèles ACT* ont permis, par exemple, de représenter les processus d'apprentissage du langage Lisp et de définir, à partir de là, un didacticiel intelligent ("The Lisp Tutor"). Bien que la description d'un modèle de simulation s'exprime dans un langage de programmation (à-la-OPS5), la définition du modèle proprement dit dépasse les compétences d'un informaticien. La théorie ACT, bien que formelle, est trop générale (elle ne s'attache pas à l'interaction homme-ordinateur) et trop difficile à appliquer.

Les modèles jugent, évaluent ou expliquent de manière formelle ou informelle, avec imprécision ou de manière approchée, le résultat d'une conception. Ils n'indiquent nullement le procédé qui devrait conduire à une conception raisonnable.

3. En quête d'une méthode de conception

Les méthodes de conception s'organisent selon deux tendances : les méthodes qui s'appuient sur une théorie de la communication homme-machine et les méthodes expérimentales dirigées par l'observation. Les premières nécessitent une connaissance approfondie de la psychologie de l'utilisateur tandis que les secondes évoluent vers cette connaissance par essais successifs. Sans théorie uniforme facile à maîtriser par l'informaticien, le recours simplificateur à quelques grands thèmes fondamentaux est inévitable. Je considère en particulier comme centrales à la conception : la notion de modèle mental, la nécessité de combler les distances d'évaluation et d'exécution et l'hypothèse que la réalisation d'une tâche se décompose en sous-tâches avec pour chacune d'elles un ensemble de variables psychologiques utiles à la conduite du travail. Dans ce canevas de points de repère théoriques, je propose d'insérer les paramètres de l'observation expérimentale [Coutaz 88a].

Les composantes observables concernent l'utilisateur effectif, le domaine d'application et le matériel. La diversité des utilisateurs, les facettes évolutives d'un même individu, la variété des applications et la multiplicité des matériels se combinent en une complexité explosive de données contradictoires. Il n'est donc pas surprenant qu'un choix technique soit bon pour une dimension et mauvais pour une autre. Les problèmes de conception d'interfaces n'ont donc pas de réponses absolues mais trouvent leur solution dans un dosage difficile de compromis. Comment procéder sur un terrain aussi insidieux?

Une méthode possible consiste à définir des priorités dans chacune des trois étapes suivantes [Schneiderman 86] :

1. admettre la diversité mais définir un profil type d'utilisateur,

2. envisager toutes les tâches réalisables dans le domaine mais identifier les plus fréquentes,

3. choisir un style d'interaction qui satisfait les priorités établies dans les deux étapes précédentes.

La définition d'un profil type d'utilisateur peut utilement s'appuyer sur la classification générique en utilisateurs novices, utilisateurs experts et utilisateurs compétents mais occasionnels. A cette dimension, il faut ajouter la catégorie socio-professionnelle et le type de connaissance (connaissances dans le domaine d'application et dans le domaine informatique). Une fois l'utilisateur type déterminé, viennent l'analyse de tâche et en tout dernier lieu, le choix d'une présentation. A ma connaissance, le premier outil qui ait supporté le processus de conception depuis l'analyse de tâche jusqu'à la spécification lexicale de l'interaction et qui soit compréhensible pour un informaticien est CLG.

4. Command Language Grammar (CLG)

CLG, ou Command Language Grammar [Moran 81], est une structure grammaticale qui permet de représenter un système informatique à différents niveaux d'abstraction : niveaux tâche, sémantique, syntaxe et interaction. Chaque niveau est une représentation complète du système, un raffinement de la vue immédiatement supérieure dans la hiérarchie. Une représentation CLG peut se voir à la fois comme la représentation des étapes de conception d'un système et comme la description de la représentation mentale que l'utilisateur a (ou est supposé avoir) du système. On trouvera dans l'annexe 1, un exemple complet d'une description CLG pour un système simple. Nous indiquons ici les principes directeurs de la méthode de conception retenue de CLG.

4.1. Niveau tâche

L'utilisateur ou le concepteur (selon que l'on prend le point de vue psychologie ou le point de vue conception), commence par identifier les tâches du domaine et organiser leur décomposition en sous-tâches. Une tâche est un objectif que l'utilisateur peut atteindre avec l'aide du système. Elle agit sur des paramètres qui représentent des concepts du domaine. La décomposition hiérarchique des tâches (que l'on retrouve dans GOMS) organise l'espace de travail de l'utilisateur. Elle est essentielle puisqu'elle définit la forme générale de la collaboration entre le système et l'utilisateur.

Une description du niveau tâche ne fait pas référence au système. Elle définit l'utilité du système du point de vue de l'utilisateur non pas du point de vue de la réalisation informatique. En réalité, les tâches atomiques et les fonctions du système définissent le point de rencontre entre la démarche descendante qui va de l'utilisateur au système, et la démarche ascendante qui va du système à l'utilisateur. Cette jonction détermine la distance sémantique exposée au chapitre 3. Elle doit être dirigée par la sémantique des tâches non pas par des critères de commodité logicielle!

Les concepts et les opérations du système sont définis au niveau de description suivant : le niveau sémantique.

4.2. Le niveau sémantique

Le niveau sémantique décrit l'accomplissement des tâches identifiées au niveau tâche en termes d'entités et d'opérateurs conceptuels. Une entité conceptuelle est un objet du système tel que l'utilisateur le perçoit, non pas tel que le système le modélise. Certaines entités représentent des concepts du domaine tandis que d'autres correspondent à des notions spécifiques au système. Les opérations conceptuelles sont les actions élémentaires applicables aux entités conceptuelles. Elles se répartissent en deux catégories : les unes sont propres au système, les autres représentent des activités de l'utilisateur telle la recherche d'une information sur l'écran. Ceci signifie que le concepteur introduit les notions du système sans exclure l'utilisateur de la représentation, ou, si l'on prend le point de vue psychologie, une représentation CLG du niveau sémantique est la connaissance sémantique que l'utilisateur élabore à propos du système pour l'accomplissement des tâches du domaine.

Ici s'achève la composante conceptuelle de CLG. Les échanges effectifs entre l'utilisateur et le système sont conçus à deux niveaux de description : les niveaux syntaxe et interaction.

4.3. Niveau syntaxique

Le niveau syntaxique définit la structure du langage de commande avec les notions de commande et de contexte. Les commandes représentent les requêtes de l'utilisateur. Leur définition comprend la description de leur effet et la spécification des arguments.

L'effet d'une commande, exprimé avec les opérations conceptuelles du niveau sémantique,

explicite la correspondance entre la notion de commande du niveau syntaxique et la notion d'opération conceptuelle du niveau sémantique. Cette correspondance définit l'action atomique du système du point de vue de l'utilisateur. Au chapitre 4 nous avons évoqué à propos de la structuration des activités, le problème de la granularité des commandes (qui ne doit être ni trop fine ni trop grossière). C'est à ce niveau de la définition du système que ce problème doit être résolu. Malheureusement CLG ne dit pas comment!

Les arguments désignent les entités conceptuelles référencées dans une commande. Un argument est défini par une sorte de descripteur qui précise :

- un type de valeur, c'est-à-dire la classe de l'entité système que l'argument représente,
- une valeur par défaut pour le cas où l'utilisateur ne spécifie pas de valeur pour l'argument,
- la notation employée par l'utilisateur pour spécifier la valeur de l'argument, et
- une procédure d'interprétation utilisée par le système pour accéder à l'entité système depuis la notation.

Un contexte est un élément de structuration de l'interaction. Il comprend essentiellement les commandes qui provoquent son activation et sa désactivation, les commandes permises quand il est actif et des variables de contrôle. La notion de contexte telle que l'introduit CLG a un double rôle : du point de vue du concepteur, il constitue un élément de structuration du dialogue et un récipient d'informations de portée déterminée. Du point de vue de l'utilisateur, un contexte représente l'ensemble des variables psychologiques à mémoriser. L'intérêt de CLG est de rapprocher les deux vues en incitant le concepteur à voir le contexte comme l'extension électronique de la mémoire à court terme de l'utilisateur.

Avec les notions de contexte et de commande, le niveau syntaxique définit la structure générale de l'interaction entre l'utilisateur et le système, mais il ne définit pas les actions physiques élémentaires nécessaires à la spécification des commandes. Cette spécification est l'objet du niveau interaction.

4.4. Niveau interaction

Le niveau interaction, qui situe la description au niveau des actions physiques, s'apparente à une spécification lexicale. Il décrit, en termes d'actions, comment l'utilisateur et le système interagissent pour aboutir à la spécification des éléments syntaxiques ; il indique quand le système doit exécuter les procédures d'interprétation syntaxiques et vérifie que l'utilisateur respecte l'ordre d'occurence des éléments syntaxiques.

5. Conclusion

Le principe directeur de CLG est un enseignement de valeur pour les informaticiens : l'interface homme-machine n'est pas seulement un langage de commande et des dispositifs matériels donnés, mais elle comprend tout ce que l'utilisateur sait et perçoit d'un système. La connaissance que l'utilisateur a d'un système est organisée en catégories. CLG reprend cette structuration théorique.

La structuration de CLG n'est pas une dichotomie mais un continuum logique de projections de vues depuis la représentation des tâches, jusqu'aux actions physiques élémentaires. A cette représentation du modèle mental, CLG superpose une approche descendante de la conception d'interfaces.

Une description CLG peut se voir à la fois comme la représentation de la connaissance que l'utilisateur a du système et comme la représentation des étapes de conception d'un système. Cette décomposition en étapes organise la tâche du concepteur en sous-domaines moins complexes, mais à chaque étape interviennent des décisions essentielles pour la qualité de l'interaction. Sur ce point, CLG et d'autres outils d'aide à la conception tels TAG [Payne 86] et les travaux de Mark Green [Green 85], sont totalement inopérants.

On relève en particulier l'absence de moyens d'expression du parallélisme et des interruptions et l'absence d'aide à la structuration. L'expression du parallélisme et des interruptions est indispensable à la modélisation de l'accomplissement d'une tâche. On entend par interruption tout déroutement du cours normal de la résolution : par exemple, le traitement des erreurs et l'accomplissement de tâches syntaxiques comme le déplacement d'une fenêtre qui cache une information importante.

La décomposition hiérarchique des tâches et l'organisation de l'espace des commandes déterminent la forme de la collaboration entre l'utilisateur et le système. Sur ce point, CLG comme tous les systèmes d'aide à la conception n'apportent aucun éléments de réponse. Ils sont incapables à partir d'une description du problème, de proposer une décomposition. Tout au plus sont-ils capables de juger une conception [Barnard 86, Barnard 87]. Le transfert entre la représentation de la connaissance telle que la propose ACT* et la décomposition d'une tâche qui plaque à la représentation ACT*, dépasse largement les compétences de l'informaticien.

Pour conclure cette première section, l'ergonomie cognitive n'offre pas d'outils concrets simples directement utilisables par les informaticiens. Elle apporte néanmoins des "outils pour la pensée" : des

abstractions, des principes et des hypothèses qui aident à faire des choix ou à détecter des ereurs de conception. Les informaticiens, confrontés au problème tangible de la réalisation de système, ont mis en place des outils logiciels qui intègrent autant que possible, les enseignements de l'ergonomie. Ces aspects sont analysés dans les sections qui suivent.

Section II

Architecture Logicielle des Systèmes Interactifs

La conception et la réalisation logicielles des systèmes interactifs se heurtent à deux sources de difficulté : la diversité de l'environnement et la dispersion de l'expression des échanges. L'environnement inclut des applications aux besoins les plus divers, une grande variété de matériels et des utilisateurs qui ne cessent d'évoluer. Dans un système interactif, le caractère immédiat et informatif des échanges met à contribution tous les composants logiciels avec pour conséquence, le risque (ou la tentation) de faire un amalgame confus entre l'expression des détails de l'interaction qui produisent l'Image et l'expression des échanges qui relève du domaine. Dans ces conditions, le comportement du système est difficile à ajuster aux variations de l'environnement.

En génie logiciel, la complexité trouve son remède dans l'usage de la modularité. Si les principes de cette technique sont connus, leur application est difficile à maîtriser en l'absence de modèle. Jusqu'à ces dernières années, on ne disposait pas de modèle de référence pour la construction des systèmes interactifs. Sans cadre directeur pour appliquer la modularité, les réalisateurs ont créé des logiciels modulaires, certes, mais inadaptés à l'évolution itérative des interfaces.

Les modèles d'architecture proposés actuellement pour les systèmes interactifs reposent tous sur la distinction entre les fonctions spécifiques à un domaine (les services) et les mécanismes de présentation (l'Image du système). J'appelle désormais :

- *application*, l'ensemble des fonctions ou services spécifiques à un domaine,
- *interface interactive*, le logiciel qui définit l'Image du système et qui assure la communication d'informations entre l'application et l'utilisateur,
- *système interactif*, l'ensemble constitué d'une application reliée à une interface interactive.

La distinction "application, interface interactive" définit un premier niveau de modularité : un module réunit l'expertise du domaine tandis qu'un autre se spécialise dans la gestion des détails de l'interaction. La modularité est alors exploitée comme technique structurelle. Elle est aussi utilisable comme technique d'abstraction pour masquer les variations de l'environnement.

Nous venons d'esquisser le thème de cette seconde section : la construction des systèmes interactifs avec ses problèmes d'architecture. Nous allons traiter ces aspects en détail dans les quatre chapitres qui suivent. Le chapitre 6 identifie les fonctions d'un système interactif et, à fin pédagogique, propose une terminologie et une organisation de ces fonctions en niveaux d'abstraction. Le chapitre 7 décrit les modèles d'architecture développés au cours de ces dernières années. Nous verrons que ces propositions ont des retombées intéressantes mais qu'elles ne permettent pas toujours de satisfaire aux exigences grandissantes de l'interaction ; le chapitre 8 présente ma contribution avec le modèle PAC qui vise à combler les lacunes des modèles précédents ; le chapitre 9 réunit un ensemble de réalisations construites selon le modèle PAC.

Chapitre 6

Les Composants Fonctionnels

1. Introduction

2. Système de fenêtrage

2.1. Terminologie et concepts
- 2.1.1. Indépendance physique
- 2.1.2. Surface d'affichage
- 2.1.3. Fenêtre

2.2. Distinction entre services de base et présentation

2.3. Nécessité des surfaces d'affichage virtuelles
- 2.3.1. Surface virtuelle et rafraîchissement
- 2.3.2. Surface virtuelle et vue multiple

2.4. Utilité des fenêtres hiérarchiques

2.5. Nécessité d'un protocole de communication ouvert
- 2.5.1. Expression des stratégies de repliement
- 2.5.2. Datation des événements
- 2.5.3. Classification des événements

2.6. Les retours d'information
- 2.6.1. Echo statique
- 2.6.2. Echo dynamique

2.7. Temps de réponse
- 2.7.1. Compétition au sein du système de fenêtrage
- 2.7.2. Héritage en provenance du système d'exploitation

3. Affichage et désignation

3.1. Machines graphiques élémentaires

3.2. Machines graphiques à images simples

3.3. Machines graphiques à images abstraites

3.4. Acquisition d'information par désignation

4. Gestion du Dialogue

EN RESUME

---

1. Introduction

L'objectif de ce chapitre est de préciser le rôle, les concepts et la terminologie des "logiciels de base de l'interaction".

Il y a une quinzaine d'années, les programmeurs ne disposaient, pour exprimer les échanges entre un utilisateur et un système, que des instructions d'entrée/sortie (E/S) d'un langage de programmation. Par exemple, en langage Pascal, le positionnement du curseur au point *1;2* d'un écran, se traduisait, pour un VT100, en l'instruction *writeln("ESC[1;2f")*. L'expression de ce contrôle direct exigeait une connaissance précise du protocole de programmation dont le niveau d'abstraction était loin de correspondre à la nature des entités manipulées dans la solution du problème.

Une seconde difficulté s'est présentée lorsque les terminaux se sont enrichis de nouvelles fonctions et se sont diversifiés. L'hétérogénéité des parcs de terminaux relançait le problème épineux de la portabilité : la substitution d'un VT100 par un écran à points invalidait de manière fatale les paramètres des instructions *writeln* déjà mentionnées.

Une troisième difficulté est apparue lorsque la notion de processus est devenue accessible à l'utilisateur. Ce dernier pouvait désormais exécuter plusieurs programmes en parallèle avec l'avantage de mener de front des activités multiples. Malheureusement, les processus ainsi créés produisaient et consommaient des informations par le biais des mêmes ressources physiques. Dès lors, les informations affichées sur l'écran par les *writeln* d'un processus risquaient de se trouver effacées par les instructions graphiques des processus reliés au même terminal.

Une dernière difficulté, liée à l'apparition des calculateurs individuels, s'est manifestée avec l'exigence grandissante des utilisateurs et avec la constatation que la bonne conduite d'une tâche informatisée s'accommodait difficilement d'une simple suite d'échanges figée par avance. La gestion du dialogue, restée longtemps dans l'ombre, est devenue explicite.

Nous venons d'identifier quatre classes de problèmes : l'expression du contenu des échanges, la diversité du fonctionnement des terminaux, l'allocation des ressources physiques de l'interaction et la gestion du dialogue. A chaque classe, le logiciel de l'interaction fait correspondre une machine abstraite spécialisée : les systèmes de fenêtrage qui ont la double fonction d'assurer l'indépendance physique et l'allocation des ressources, les machines à images abstraites qui permettent l'expression des échanges tout en restant proches de la nature des entités du système interactif et les gestionnaires

de dialogue qui contrôlent la structure de l'interaction. Nous développons chacun de ces aspects dans les paragraphes qui suivent : au paragraphe 2, les systèmes de fenêtrage ; au paragraphe 3, les problèmes de l'affichage et, au paragraphe 4, les aspects relatifs à la gestion du dialogue.

2. Systèmes de fenêtrage

Dans la plupart des systèmes de fenêtrage, le partage des ressources de l'interaction et le masquage du fonctionnement du terminal physique s'appuient sur des concepts similaires. Si les concepts sont voisins, la terminologie et les mises en œuvre se caractérisent au contraire par la diversité. Ce paragraphe introduit une terminologie qui reflète la tendance actuelle de la technologie des systèmes de fenêtrage ; il identifie ensuite les caractéristiques d'un système de fenêtrage qui jouent en faveur de l'utilisateur et du programmeur : la distinction entre les services de base et les fonctions de présentation, la nécessité des concepts de surface d'affichage virtuelle et de fenêtres hiérarchiques, l'avantage d'un protocole de communication ouvert, le rôle à jouer dans la réalisation des retours d'information et les implications du temps de réponse.

2.1. Terminologie et concepts

Dans la gestion des ressources du terminal, l'écran fait l'objet d'une attention particulière. Après la présentation du principe de réalisation de l'indépendance physique, on introduit les concepts essentiels à la gestion de l'écran : les surfaces d'affichage et les fenêtres.

2.1.1. Indépendance physique

On assure l'indépendance d'un programme client vis-à-vis d'un appareillage physique en définissant un appareil abstrait dont les primitives masquent le fonctionnement de l'appareil réel. L'interprétation de ces primitives donne lieu à une génération d'instructions dépendantes du matériel.

Le système de fenêtrage X Window [Scheifler 86] (ou plus simplement X) inclut une couche qui garantit l'indépendance des services de gestion des ressources vis-à-vis du matériel. Dans le domaine des normes graphiques, CGI [ISO 86b] définit un appareil graphique abstrait.

La définition d'un appareil abstrait est souvent le résultat de compromis entre la généralité, la facilité de mise en œuvre et une véritable indépendance physique. Par exemple,

- dans ses algorithmes de restitution, le protocole de X considère que les points des écrans sont carrés. Cette hypothèse est généralement vraie mais si les points sont rectangulaires, un cercle prend la forme d'une ellipse :
- pour les entrées, il faut être capable de modéliser les dispositifs de commande les plus

courants. Pour cela, on organise leur diversité en classes appelées unités logiques dont la nature dépend du type d'applications ou de terminaux visés. On trouve généralement les notions de localisateur (en anglais, locator) pour désigner un point de l'écran, les notions de clé et de choix que Core [GSPC 79] et GKS [ISO 85] ont contribuées à normaliser.

2.1.2. Surfaces d'affichage

Afin de s'abstraire des limites physiques de l'écran, les systèmes de fenêtrage ont introduit la notion de surface d'affichage. On parle de *grafport* dans le Macintosh [Rose 86], de *canvas* dans NeWS [SUN 87, Gosling 86b] et de *drawables* dans X [MIT 87].

Une surface d'affichage est un dispositif abstrait caractérisé par un système de coordonnées et capable d'interpréter des requêtes graphiques élémentaires (par exemple, "dessiner-un-segment-de-droite"). L'effet de ces interprétations (dans l'exemple, une suite de points alignés) est mémorisé dans un espace plan dont la taille peut être supérieure à celle de l'écran. Pour tenir compte de la couleur, cet espace est modélisé sous forme d'un ensemble de plans parallèles. Cet ensemble définit les pixels de la surface d'affichage. Un pixel (de l'anglais, "picture element") est l'unité d'affichage d'une surface d'affichage.

On distingue deux formes de surfaces d'affichage : celles qui ne peuvent être rendues visibles et celles que le système de fenêtrage est capable de projeter sur l'écran. Les premières permettent de mémoriser des images dont la taille peut dépasser celle de l'écran. Pour cette raison, je les appelle surfaces d'affichage virtuelles. Les secondes servent à construire des fenêtres.

2.1.3. Notion de fenêtre

La notion de fenêtre varie selon les systèmes. On trouve essentiellement deux définitions de base : la fenêtre élémentaire et la fenêtre composée.

1. Une fenêtre élémentaire est une surface d'affichage projetée sur l'écran. Elle a comme propriétaire le processus client qui en a demandé la création.

2. Une fenêtre composée est un assemblage particulier de surfaces d'affichage projetées sur l'écran. L'une d'entre elles définit le contenu de la fenêtre. Elle appartient au processus à l'origine de la création de la fenêtre. Elle est le lieu de communication entre l'utilisateur et ce processus. Les autres surfaces définissent le cadre. Elles n'appartiennent pas au créateur de la fenêtre mais au composant du système de fenêtrage chargé de gérer la présentation des fenêtres à l'utilisateur. Elles servent aux échanges entre l'utilisateur et le système de fenêtrage. L'assemblage géométrique des surfaces qui constituent une fenêtre est assuré par le système de fenêtrage.

Dans certains systèmes comme X, les deux formes de fenêtres peuvent cohabiter. Dans d'autres, comme le Macintosh, seule la fenêtre composée est permise. La figure 6.1 en illustre le principe.

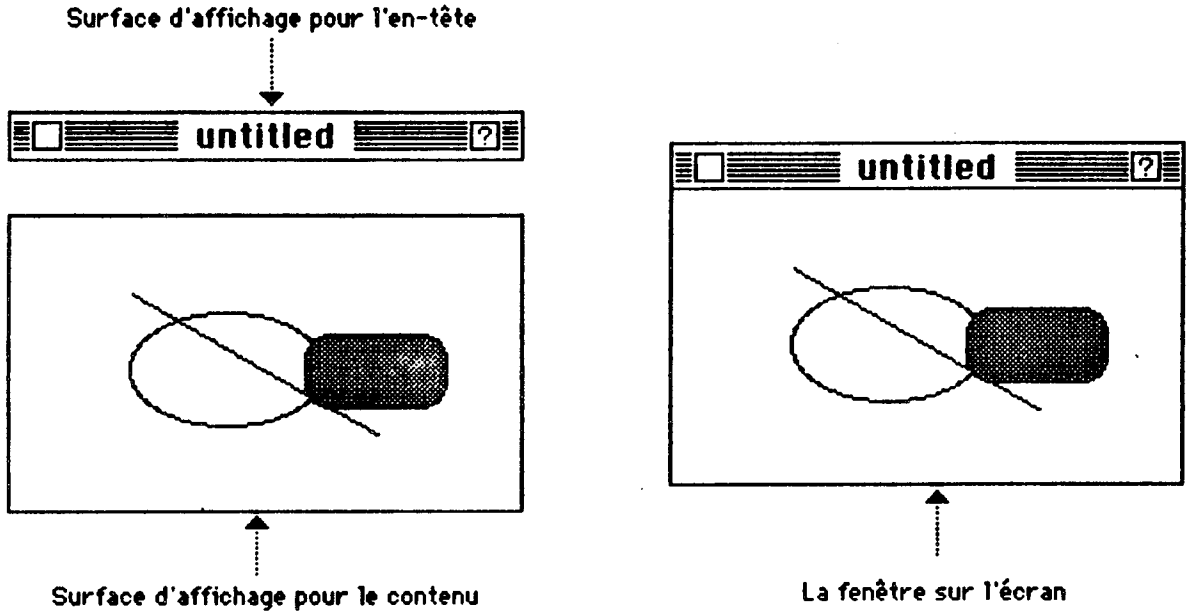


Figure 6.1 : Une fenêtre Macintosh est une composition de deux surfaces d'affichage projetées sur l'écran.

2.2. Distinction entre les services de base et la présentation

La distinction entre la fenêtre, simple surface d'affichage, et la fenêtre, assemblage de surfaces d'affichage, suggère la séparation modulaire des services de base de ceux de la présentation.

Les services de base gèrent l'allocation des ressources de l'interaction tandis que les services de présentation définissent l'Image (au sens de D. Norman) du système de fenêtrage. L'avantage de cette organisation est de permettre le remplacement ou l'ajustement de l'Image, voire la cohabitation de plusieurs (superposition et carrelage de fenêtres [Cohen 86]) et d'évaluer leurs qualités ergonomiques respectives [Bly 86], sans mettre en cause les services de base. L'environnement graphique de Brown distinguent les services de base, ASH, des services de présentation, MAPPLE [Pato 84]. X et NeWS appliquent ce même principe en s'appuyant sur le modèle du client-serveur :

- dans X, un processus serveur centralise et synchronise toutes les requêtes ayant trait aux services de base tandis que les services de présentation sont assurés par un processus client : le gestionnaire de fenêtres ;
- dans NeWS, les processus clients adressent au serveur des requêtes personnalisées sous forme de programmes PostScript [Adobe 85]. Ils peuvent ainsi télécharger toute forme de politique y compris des techniques de présentation.

Dans des systèmes de fenêtrage plus anciens tels SunWindows [SUN 85], le système de WhiteChapel [Sweetman 86], Sapphire [Myers 84] et le gestionnaire de fenêtres du système Andrew [Gosling 86c], les services de base et de présentation sont indissociables. Ces systèmes se

différencient par la façon de répartir les fonctions de fenêtrage dans l'espace d'exécution des processus : soit dans l'espace d'adressage de chaque processus client, soit dans le noyau du système, soit dans un processus serveur. Gosling dans [Gosling 86a] compare ces trois techniques :

- Le fenêtrage dans l'espace d'adressage des processus clients a l'avantage d'éviter les appels au noyau du système d'exploitation mais il impose aux clients de gérer les conflits d'accès aux ressources. En outre, en l'absence de mécanisme de partage de bibliothèques, le système de fenêtrage est reproduit en plusieurs exemplaires. La conséquence possible est une sollicitation excessive du système de gestion de la mémoire. SunWindows pratique cette technique.

- Le fenêtrage dans le noyau du système résoud les problèmes de synchronisation et d'arbitrage mais compromet la fiabilité et l'intégrité du noyau. Le système de WhiteChapel relève de cette catégorie.

- Le fenêtrage dans un processus serveur satisfait les contraintes de synchronisation sans mettre en péril l'intégrité du noyau mais ses performances dépendent de l'efficacité du mécanisme de communication interprocessus. (Dans le cas idéal, le rafraîchissement d'une fenêtre devrait s'effectuer en moins de 0,1 seconde ceci afin de se conformer au cycle du processeur sensoriel.) Inversement, un service de communication qui intègre les fonctions du réseau, tels les ports de MACH-1 [Acetta 86] ou les appels de procédure à distance, rend possibles la manipulation de fenêtres à distance et l'exécution du serveur et des clients sur des sites distincts. Le gestionnaire de fenêtres du système Andrew, Sapphire, X et NeWS sont organisés autour de la notion de serveur.

2.3. Nécessité des surfaces d'affichage virtuelles

Les surfaces d'affichage virtuelles sont des mémoires d'image. Elles ont deux applications intéressantes : rafraîchir efficacement le contenu des fenêtres et montrer plusieurs portions d'une même image dans des fenêtres distinctes.

2.3.1. Surface virtuelle et rafraîchissement

Une opération de rafraîchissement consiste à restaurer les pixels d'une fenêtre dont la surface visible vient d'être modifiée. Deux techniques de base sont envisageables : gérer un double des pixels dans une surface virtuelle d'affichage ou enregistrer dans ce qu'on appelle une liste d'affichage (en anglais, display list), les requêtes graphiques ayant produit l'image. La première méthode est illustrée dans la figure 6.2. L'image est produite par le programme client sur la surface virtuelle. A cette surface, est associée une lucarne rectangulaire de taille identique à la celle de la fenêtre et de localisation variable par rapport au système de coordonnées de la surface. Par simple opération graphique dite "raster", l'ensemble ou une partie des pixels de la lucarne est transféré vers la fenêtre.

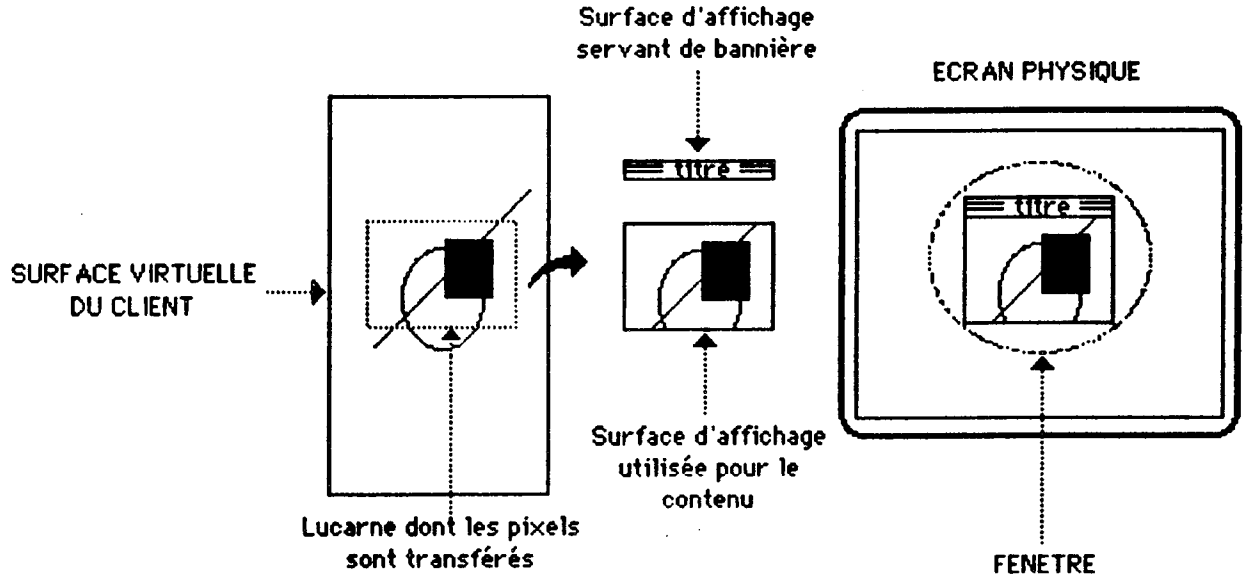


Figure 6.2 : Rafraîchissement du contenu d'une fenêtre à partir d'une surface d'affichage virtuelle.

La restauration des pixels à partir d'une surface virtuelle est gourmande en mémoire centrale mais l'image est reconstituée en une seule opération graphique. Une liste d'affichage est plus compacte mais son interprétation est plus coûteuse (surtout si elle contient des requêtes qui se défont!). Du point de vue de l'utilisateur, la première technique donne l'effet d'une restauration quasi instantanée et globale alors qu'avec la méthode des listes d'affichage, l'utilisateur peut suivre l'effet local de chaque opération graphique. Lorsque les temps de réponse le permettent (c.-à-d. lorsqu'ils sont de l'ordre du cycle du processeur sensoriel), il est possible d'envisager une combinaison des deux techniques de base : réinterpréter tout ou partie de la liste d'affichage en produisant l'image sur une petite surface virtuelle puis extraire de cette surface les parties de la fenêtre à restaurer.

2.3.2. Surface virtuelle et vue multiple

La figure 6.3 montre l'utilisation d'une surface virtuelle pour présenter des vues multiples. On pratique ici une extension de la technique précédente : le programme client dessine l'image de l'attelage dans une surface virtuelle mais gère trois lucarnes pour alimenter des fenêtres distinctes. L'utilisateur obtient ainsi des vues multiples complémentaires sur un document trop volumineux pour être entièrement visible dans une fenêtre unique.

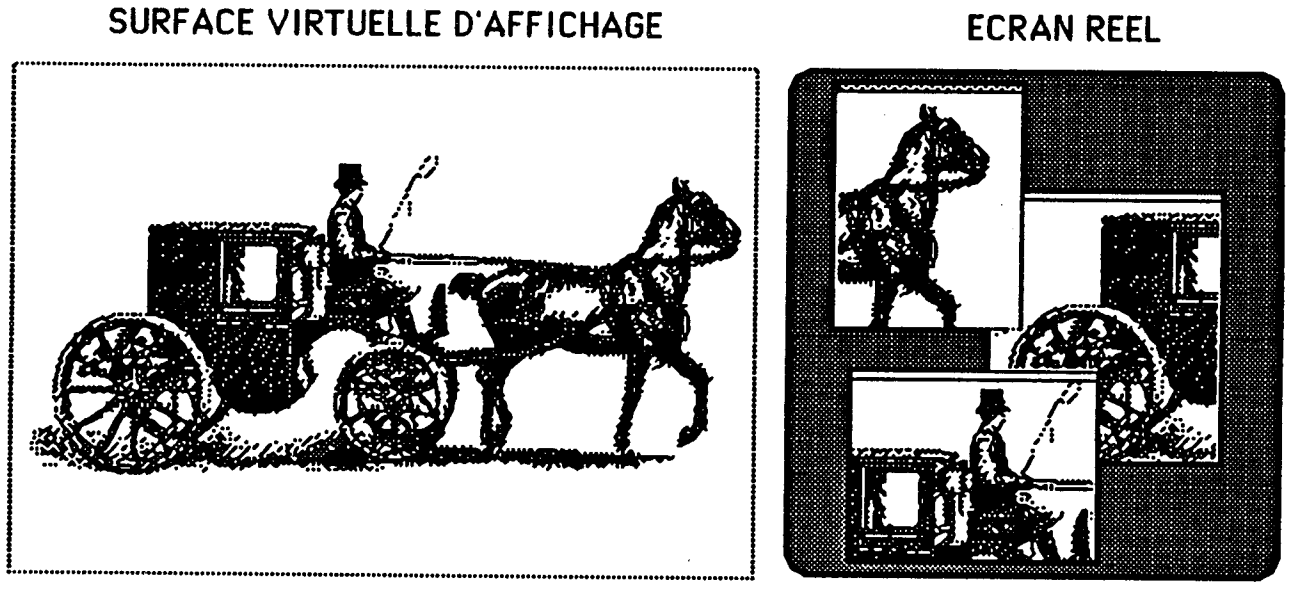


Figure 6.3 : Surface d'affichage virtuelle et sa projection simultanée dans trois fenêtres distinctes.

Les surfaces d'affichage virtuelles sont pour l'écran ce que la mémoire virtuelle est à la mémoire centrale : une fenêtre montre une vue partielle de la surface virtuelle tout comme une page en mémoire centrale contient une part de la mémoire virtuelle. De même que l'exécution d'un programme peut conduire au remplacement d'une page en mémoire centrale par une autre, de même l'utilisateur peut provoquer la visualisation d'une nouvelle portion de la surface virtuelle. Alors que la gestion de la mémoire virtuelle est intégrée depuis longtemps aux systèmes d'exploitation, la gestion du rafraîchissement des fenêtres est encore à la charge du programmeur!

S'il est vrai qu'un système de fenêtrage ne peut fournir un mécanisme de rafraîchissement à la fois général et optimimal, le programmeur doit néanmoins disposer de la notion de surface virtuelle. Il est alors possible de définir une sous-classe de fenêtres adaptée aux besoins d'affichage des applications visées mais capable de gérer automatiquement le rafraîchissement. A titre d'exemple, j'ai appliqué cette technique à la définition d'une classe de fenêtres au dessus du système de fenêtrage du Macintosh. Cette classe se trouve être intégrée à APEX, un squelette d'application décrit au chapitre 12.

Lorsque le système de fenêtrage, telle la version 10 de X, ne fournit pas de véritables surfaces d'affichage virtuelles (les primitives graphiques de dessin sont inexistantes), la réinterpréation des listes d'affichage, qui entraîne parfois une gêne visuelle pour l'utilisateur, est inévitable.

2.4. Utilité des fenêtres hiérarchiques

La création, sans restriction, de hiérarchies de fenêtres est une fonction importante pour la mise en œuvre des interfaces de manipulation directe. Si ce type d'interface facilite la vie de l'utilisateur, il complique sensiblement celle du programmeur : la manipulation d'un objet peut impliquer des déplacements, des modifications de taille, donc éventuellement des recouvrements et des changements d'exposition. Or le traitement de ces phénomènes fait précisément partie des fonctions d'un système de fenêtrage. Avec un mécanisme de sous-fenêtrage, chaque objet de l'interface peut utiliser une sous-fenêtre comme support de restitution et hériter du même coup de leurs propriétés et de leurs opérateurs. Par exemple, les widgets de la boîte à outils réalisée au dessus de X sont des spécialisations de fenêtres. Sans mécanisme de sous-fenêtrage, le programmeur d'application est contraint de reproduire une partie des algorithmes du système de fenêtrage.

2.5. Nécessité d'un protocole de communication ouvert

Un protocole ouvert permet d'étendre les services de base tout en respectant un modèle directeur. Dans les systèmes de fenêtrage, il faut être attentif à deux formes d'extensibilité : l'expression des stratégies de repliement et, pour les événements, la datation et la classification.

2.5.1. Expression des stratégies de repliement

La possibilité d'exprimer des stratégies de repliement accroît la robustesse des programmes clients. Contrairement aux ressources gérées par les systèmes d'exploitation qui, une fois allouées, sont sûres, les ressources allouées par un système de fenêtrage ne sont que partiellement garanties. Par exemple, une fenêtre dont la taille peut être modifiée à tout instant est une ressource variable. Cette instabilité peut poser des problèmes au client qui suppose l'existence d'un minimum de pixels, d'une police de caractères ou d'une couleur. Le client doit pouvoir exprimer des stratégies de repliement. La forme de l'expression peut aller de la simple requête dont la sémantique est fixée par le système de fenêtrage comme dans X, à un véritable programme élaboré par le client comme dans NeWS.

2.5.2. Datation des événements

L'intérêt d'associer une date aux événements est la synthèse d'événements. A son tour, celle-ci permet de contourner certaines limitations du matériel, et notamment la séquentialité du système d'interruption et le nombre restreint de boutons sur les dispositifs de pointage :

- les actions simultanées de l'opérateur sur plusieurs dispositifs de commande (par exemple une touche du clavier et un bouton de la souris) sont traitées en séquence par le système d'interruption puis traduites en une suite d'événements. La comparaison des dates des événements permet de reconstituer la simultanéité des actions ;

- de même, il est possible de simuler une souris à n boutons avec une souris à un bouton en donnant un sens à la réception de n clics successifs. Le temps écoulé entre deux clics lève l'ambiguïté entre les n événements considérés comme une action unique et la répétition de la même action. Afin de tenir compte de la rapidité motrice de l'opérateur, le Macintosh gère une variable globale par dispositif de commande. Toutes sont accessibles à l'opérateur par l'intermédiaire du panneau de contrôle.

L'usage combiné de plusieurs dispositifs de commande étant admis (après tout, nous conduisons en utilisant nos pieds et nos mains), la datation des événements est devenue une nécessité. On regrettera que cet attribut ne soit pas toujours modélisé par les systèmes de fenêtrage.

2.5.3. Classification des événements

La classification des événements est un autre facteur d'extension du protocole de communication. La classe d'un événement est un type énuméré. Dans certains systèmes de fenêtrage, tel le gestionnaire de fenêtres du Macintosh, le sens de certaines valeurs du type est laissé au choix du client. L'intérêt de cette ouverture est de permettre à des clients d'échanger des messages personnalisés en utilisant les services de communication du système de fenêtrage.

2.6. Retours d'information

Le système de fenêtrage intervient directement dans la réalisation des retours d'information liés aux dispositifs de pointage. A ce niveau d'implication, on parle d'écho et l'on distingue l'écho statique et l'écho dynamique.

2.6.1. Echo statique

L'écho statique s'effectue sans faire intervenir le client et sa forme est fixée par avance. Le service de gestion du curseur de certains systèmes de fenêtrage rentre dans cette catégorie. Dans ces systèmes, les clients ont la possibilité d'associer une forme de curseur à une surface d'affichage (ou à une région de cette surface) et le système de fenêtrage se charge de l'écho : le curseur suit les mouvements de la souris et change automatiquement de forme au franchissement d'une frontière de surface (ou de région). L'avantage de cette approche est que le système de fenêtrage peut tirer profit du matériel lorsque des dispositifs spécifiques le permettent (voir les *sprites* de l'Amiga [Mical 86]). Cependant, l'image du curseur, fixée à l'avance, ne peut qu'exprimer une sémantique statique telle que le mode actuel de l'interaction. Nombreuses sont les circonstances où cette image doit être

calculée à la volée.

2.6.2. Echo dynamique

L'écho dynamique s'effectue en faisant intervenir le client pour qu'il en détermine dynamiquement la forme. Les objets élastiques rentrent dans cette catégorie. Deux techniques logicielles sont envisageables pour réaliser ce type d'écho :

1. soit effectuer le cycle

```
tant que continuer faire
    signaler au client un événement "déplacement-souris";
    traiter la requête graphique du client qui produit l'écho;
fin;
```

2. soit exécuter une procédure fournie par le client et adaptée à la circonstance.

La première méthode est celle adoptée par X. Elle a le mérite d'être totalement dynamique puisqu'elle fait appel au client à chaque événement souris. A l'inverse, elle est tributaire des performances du réseau si le serveur et le client s'exécutent sur des sites différents. La seconde méthode s'apparente à la technique de téléchargement de procédures pratiquée par NeWS. Elle garantit le rafraîchissement en temps réel puisque la procédure d'interprétation, une fois transmise, est locale au serveur. Cette technique marche bien pour des échos semi-dynamiques, telle la génération du rectangle élastique, qui se suffisent à eux-mêmes. Elle s'effrite dès l'instant où l'écho se met à dépendre de structures de données profondément ancrées dans le client et qui, à leur tour, évoluent avec la position de la souris.

De manière idéale, l'écho du suivi de la souris devrait intégrer les deux types de techniques : pour l'écho statique et semi-dynamique, une procédure spécifique au système de fenêtrage mais surchargeable par le client ; pour l'écho dynamique, le mécanisme usuel des aller-retour "événement-requête" à condition qu'il garantisse un taux de rafraîchissement d'au moins 10 images par secondes.

2.7. Temps de réponse

Sans retour d'information instantané, l'utilisateur risque de s'enliser dans des situations d'erreur dont il est parfois difficile de s'extirper. Par exemple, l'utilisateur actionne un dispositif de commande mais ne perçoit aucune réaction dans l'Image. Il en déduit que le système n'a pas reçu son message. Il actionne le dispositif une seconde fois et constate que l'action est interprétée dans un environnement inattendu. Le problème peut provenir d'une compétition au sein du système de fenêtrage qui

désorganise l'agencement des événements ; il peut aussi trouver ses racines dans la structure même du système d'exploitation.

2.7.1. Compétition au sein du système de fenêtrage

L'architecture de X induit une forme de compétition entre le gestionnaire de fenêtres, le processus client chargé de la présentation des fenêtres, et les clients ordinaires du serveur. Parce que tous fonctionnent entre eux de manière asynchrone, les requêtes d'un client ordinaire risquent à tort de devancer celles du gestionnaire de fenêtre. Par exemple, l'utilisateur peut exprimer un changement de fenêtre courante et commencer à utiliser le clavier avant que le gestionnaire de fenêtre ait pu signifier le changement au serveur : la saisie n'est pas transmise au contexte désiré.

NeWS contourne ce problème en organisant différemment l'espace d'exécution : tout processus client a un représentant local au serveur qui exprime son intérêt pour des classes d'événements données. Comme dans X, le serveur centralise et sérialise les événements d'entrée sur une file unique. A l'inverse de X, l'événement de tête est diffusé à tous les processus ayant manifesté un intérêt pour ce type d'événement. Les processus candidats ont ainsi la garantie de s'exécuter avant la diffusion de l'événement suivant. Cette sérialisation n'est cependant pas assez stricte car il se peut qu'un processus n'ait pu exprimer son intérêt à temps. Par exemple, on peut imaginer qu'à la détection de l'enfoncement d'un bouton de la souris, le processus récepteur lance un processus "menu". Pour que ce dernier reçoive tous les événements le concernant, il faut qu'il exprime son intérêt avant même que le bouton de la souris soit relâché. En toute rigueur, tout changement de la liste des intérêts doit s'effectuer en exclusion mutuelle. Pour ce faire, NeWS propose une primitive de verrouillage de la file des événements qui élimine la diffusion jusqu'au déverrouillage ou à l'expiration d'un délai paramétrable.

2.7.2. Héritage en provenance du système d'exploitation

Certains systèmes d'exploitation, tels les environnements Smalltalk et Lisp, sont contraints d'interrompre le cours normal des opérations pour réorganiser l'allocation de la mémoire centrale. Les algorithmes de ramasse-miettes sont généralement coûteux au point d'être perceptibles à l'utilisateur. Par exemple, une action qui devrait provoquer l'affichage d'un menu fantôme est sans effet tant que l'opération de nettoyage est en cours. Si cette opération doit se répéter fréquemment et, en outre, s'accompagner de vidage de pages, (cas du système Interlisp-D de la machine Xerox 1186), alors les temps de réponse compromettent l'utilisation du système. Tous les logiciels développés au dessus de ces environnements héritent irrémédiablement de la lenteur des performances du noyau.

La difficulté pour l'utilisateur n'est pas uniquement la lenteur mais sa variation imprévisible : le

déclenchement du ramasse-miettes n'est pas toujours contrôlable depuis la surface. En conséquence, une même action peut avoir un écho immédiat ou au contraire se manifester au bout de quelques secondes. Cette variation aléatoire ne facilite pas l'élaboration de stratégies de comportement moteur.

3. Affichage et désignation

L'affichage d'informations peut se voir comme une cascade de transformations qui, à partir de structures de données internes, produit une image sur un support de restitution. L'acquisition des informations implique une suite de transformations inverses. La succession des transformations nécessaires à l'affichage comme à l'acquisition indique l'existence de niveaux d'abstraction. L'objectif de ce paragraphe est d'introduire les concepts et les techniques qui définissent les niveaux de machines graphiques répertoriées dans la figure 6.4 : les machines élémentaires, les machines à images simples et les machines à images abstraites.

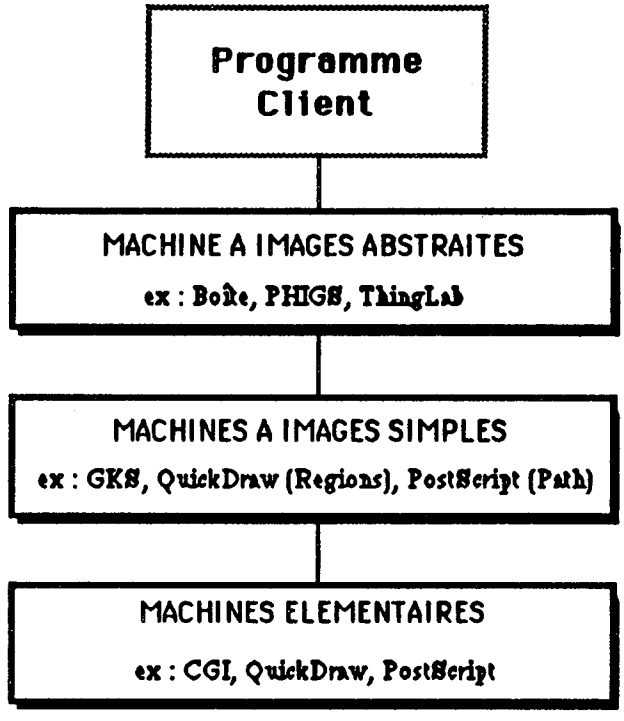


Figure 6.4. Les niveaux de machines graphiques.

3.1. Machines graphiques élémentaires

Les concepts et le répertoire d'instructions d'une machine élémentaire définissent un appareil abstrait graphique dont le fonctionnement reste proche des caractéristiques d'une gamme de matériels graphiques. De telles machines permettent de dessiner des entités graphiques simples tels que des

segments de droite, des arcs, et des motifs de remplissage. CGI (Computer Graphics Interface [ISO 86b], ex Virtual Device Interface) sert de norme à ce niveau d'abstraction. QuickDraw [Rose 86] et PostScript [Adobe 85] sont deux exemples intéressants.

Quickdraw est la machine graphique élémentaire du Macintosh. Son originalité tient à la richesse des opérations définies sur les concepts de *grafport* , de *region* et de *picture* . Pour l'instant, nous retenons celui de grafport. Un grafport est un contexte d'affichage, c'est-à-dire une surface d'affichage à laquelle sont associés des paramètres graphiques (grosseur, forme et position actuelles du pinceau, motif de remplissage, police et corps des caractères, couleur du fond, etc.). Lorsque la surface d'un grafport peut être projetée sur l'écran (on l'appelle "grafport sur écran"), les requêtes graphiques ont un effet perceptible. Dans le cas contraire, le grafport est un contexte d'affichage virtuel.

L'effet visible d'une requête graphique se produit dans la surface résultat de l'intersection de trois ensembles : un portrect, une région visible et une région de coupure (en anglais, clipping region). Le portrect est une lucarne de la surface d'affichage. La région visible est la surface non recouverte de la surface d'affichage. Elle est automatiquement gérée par le mécanisme des recouvrements. La région de coupure dont la forme est arbitraire (courbe spline fermée), est laissée à l'appréciation du programme client pour limiter l'effet des requêtes graphiques à un sous ensemble du portrect. Cette possibilité évite l'ajustement, parfois complexe, des paramètres des requêtes graphiques en vue de restreindre leur portée. Le rôle complémentaire du portrect, de la région visible et de la région de coupure permet de cadrer exactement les zones à rafraîchir ce qui améliore sensiblement la qualité visuelle du rafraîchissement.

PostScript est la machine graphique de base de NeWS. Son originalité tient à son modèle d'image qui part de l'idée de pochoir [Warnock 82]. Dans le domaine de l'art graphique, un pochoir est une plaque de métal (ou de carton) découpée sur laquelle on passe une brosse ou un pinceau pour produire des dessins. PostScript reprend cette technique mais profite de la puissance de calcul de l'ordinateur pour associer au pochoir des opérateurs graphiques évolués (rotation, zoom, déformation). Dans PostScript, un pochoir a deux représentations possibles : un *path* et un *mask* . Un path est un ensemble de courbes splines fermées qui modélisent les contours des orifices du pochoir. Un mask est une matrice de bits dont chaque 1 représente un trou dans le pochoir. La peinture est une structure qui modélise une couleur, une texture ou même une image. La figure 6.5 illustre le principe du modèle.

Le modèle d'image de PostScript est intéressant pour sa simplicité et sa généralité. En particulier,

- les polices de caractères ne justifient pas de traitement spécial.
- la notion de path permet de définir des formes arbitraires et de les manipuler comme un tout : à ces formes, il est possible d'appliquer des opérateurs sophistiqués pour pratiquer simplement des déformations d'apparence complexe. Le path, qui peut aussi servir de région de coupure, permet d'ajuster les raffraîchissements et les effets visuels.

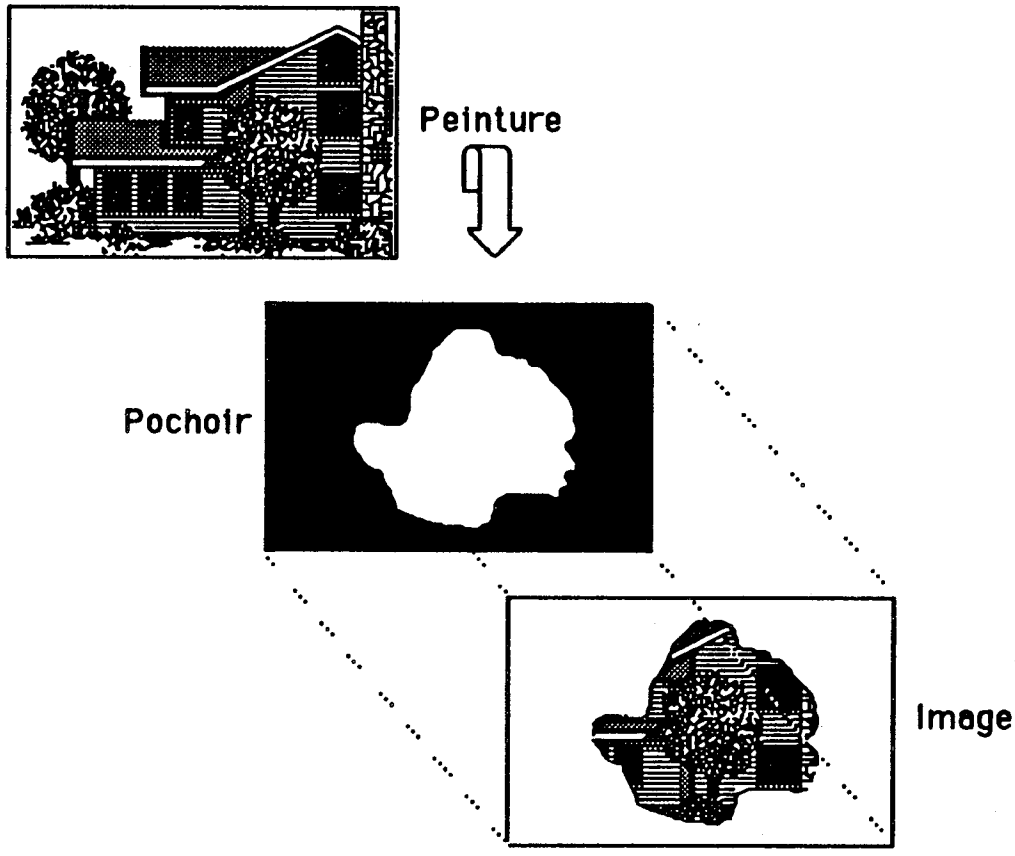


Figure 6.5 : Le modèle d'image de PostScript.

3.2. Machines graphiques à images simples

Une machine à images simples introduit, par rapport aux machines élémentaires, la possibilité de construire de nouvelles unités graphiques. Le mécanisme d'abstraction proposé s'apparente à la technique des macroinstructions qui encapsule un ensemble de primitives en une entité manipulable comme un tout. Le niveau d'abstraction de ces primitives est celui des instructions des machines graphiques élémentaires : une unité graphique ne peut contenir d'autres unités. GKS, en tant que norme, peut être retenu comme exemple. La region et la picture de QuickDraw, de même que le path de PostScript permettent de considérer ces deux systèmes comme des machines à images simples.

Dans GKS, l'unité construite est un *segment*. Alors que QuickDraw et PostScript s'en tiennent à la restitution, GKS prend à sa charge le multiplexage des événements d'entrée. Son modèle de la

gestion des entrées, qui s'appuie sur la notion d'unité logique, masque le fonctionnement des dispositifs physiques mais ne répond pas vraiment aux besoins de l'interactivité. En particulier, un programme client ne peut pas attendre d'événement sur plusieurs fenêtres simultanément. Le fait que GKS prenne à sa charge des fonctions attribuées aujourd'hui aux systèmes de fenêtrage s'explique par la carence de ces systèmes au moment de sa définition.

Dans QuickDraw, une *region* est un assemblage de pixels au sein d'une courbe spline fermée. La possibilité de manipuler des formes arbitraires contribue à la construction d'images expressives et originales. La figure 6.6 illustre l'utilisation d'une region comme écho semi-dynamique. La figure 6.7 montre le programme de construction correspondant. Une *picture* se définit comme une region sous forme d'une macroinstruction graphique. Comme la region, une picture ne peut contenir d'autres pictures. Alors que la region permet, comme le path, de jouer sur les subtilités fines de l'affichage, une picture est un objet graphique destiné à être montré. Elle sert de matrice à l'incarnation d'images similaires au facteur d'échelle près. La figure 6.8 montre trois exemplaires d'une picture.

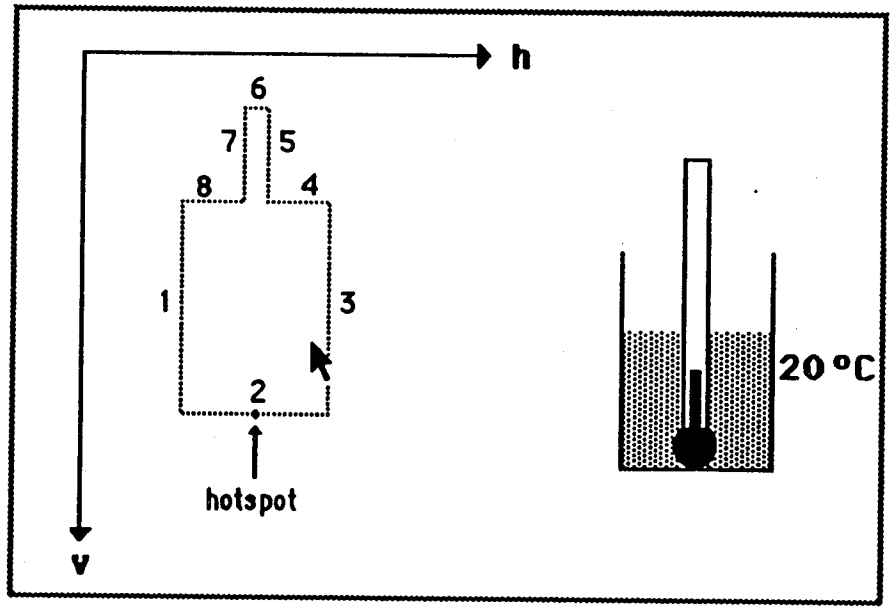


Figure 6.6 : A gauche, en pointillé, les contours de la region tels qu'ils apparaîssent lorsqu'elle est liée aux mouvements de la souris. Les chiffres correspondent aux numéros des segments de droite commentés dans le programme de la figure 6.7. Le hotspot sert de point de référence. Les axes de coordonnées, les numéros des segments et le hotspot ne figurent pas sur l'écran réel. Ils sont présentés dans la figure afin de faciliter la compréhension du programme de la figure 6.7. A droite, un objet obtenu en dessinant la region, une colonne de mercure, de l'eau et une valeur numérique. Cet objet s'inscrit dans une interface de type manipulation directe présentée au chapitre 9. Lorsqu'il est déplacé par l'utilisateur à l'aide de la souris, la region est automatiquement liée aux mouvements de la souris afin de concrétiser à tout instant la localisation future de l'objet. La region est utilisée ici comme écho semi-dynamique.

```
myThermoGlass = NewRgn();-- créer une region
OpenRgn();                    -- Commencer à mémoriser les requêtes graphiques

MoveTo(hotspot.h-GLASSWIDTH/2 , hotspot.v-GLASSHEIGHT);
LineTo (hotspot.h-GLASSWIDTH/2, hotspot.v);                    -- tracer le segment 1
LineTo (hotspot.h+GLASSWIDTH/2, hotspot.v);                    -- tracer le segment 2
LineTo (hotspot.h+GLASSWIDTH/2, hotspot.v-GLASSHEIGHT);        -- tracer le segment 3

PenSize(0,0);                 -- les tracés suivants sont invisibles
LineTo (hotspot.h+PIPEWIDTH/2, hotspot.v-GLASSHEIGHT);         -- tracer le segment 4
LineTo (hotspot.h+PIPEWIDTH/2, hotspot.v-PIPEHEIGHT);          -- tracer le segment 5
LineTo (hotspot.h-PIPEWIDTH/2, hotspot.v-PIPEHEIGHT);          -- tracer le segment 6
LineTo (hotspot.h-PIPEWIDTH/2, hotspot.v-GLASSHEIGHT);         -- tracer le segment 7
LineTo (hotspot.h-GLASSWIDTH/2, hotspot.v-GLASSHEIGHT);        -- tracer le segment 8

PenNormal();                  -- le crayon retrouve ses valeurs standard

CloseRgn(myThermoGlass); -- associer les requêtes graphiques à myThermoGlass

DragGrayRgn(myThermoGlass); -- associer la region aux mouvements de la souris
```

Figure 6.7 : Le programme de construction de la region de la figure 6.6. La region est définie par l'ensemble des instructions graphiques comprises entre les instructions *OpenRgn* et *CloseRgn*.

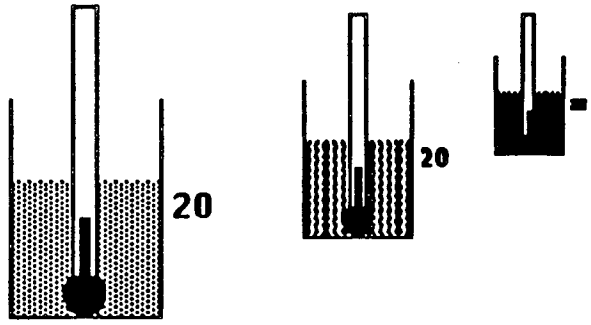


Figure 6.8 : Trois exemplaires d'une picture de QuickDraw.

3.3. Machines graphiques à images abstraites

Une machine à images abstraites introduit, par rapport aux machines à images simples, des possibilités de structuration et d'expression de contraintes et de décorations :

- une unité graphique peut inclure d'autres unités graphiques. Par exemple, l'image d'une maison comprend deux images plus simples : les murs et le toit ;
- les unités graphiques, simples ou composées, peuvent être liées par des relations géométriques. Par exemple, le toit d'une maison est toujours au dessus des murs ;
- une unité est décorée d'attributs de restitution dont le niveau d'abstraction est plus élevé que celui des attributs graphiques des machines à images simples. Par exemple, l'attribut "mise en

évidence" du toit peut être traduit, selon le cas, par un tracé épais, un clignotement ou une vidéo inverse.

La norme PHIGS [ISO 86a, Shuey 86, Shuey 87], le concept de boîtes [Knuth 79, Mikelsons 81, Coutaz 84b, Hibbard 84, Nanard 84, Quint 85, Alhers 86, Morcos 86a, Morcos 86b] et les vérificateurs de contraintes géométriques tel ThingLab [Borning 86a] permettent de construire des images abstraites. Le principe et les avantages de ces techniques sont décrits au chapitre 11.

3.4. Acquisition d'information par désignation

L'acquisition d'information est le cheminement des transformations représentées dans la figure 6.9, depuis la désignation d'un point de l'écran jusqu'au concept de l'application.

Le pilote de la souris relève le point (x,y) de l'espace de coordonnées physiques de l'écran. Le système de fenêtrage identifie la fenêtre propriétaire du point. A ce niveau d'abstraction, l'identification du point s'exprime maintenant avec le triplet (identification-fenêtre, x', y'). Lorsque la technique de restitution utilise une surface virtuelle, le point doit être repéré dans le système de coordonnées de cette surface, soit : (identification-surface, x", y"). Cette identification doit permettre de déterminer l'élément de l'image abstraite propriétaire du point (ici, le toit), élément qui représente un concept du domaine (le Toit).

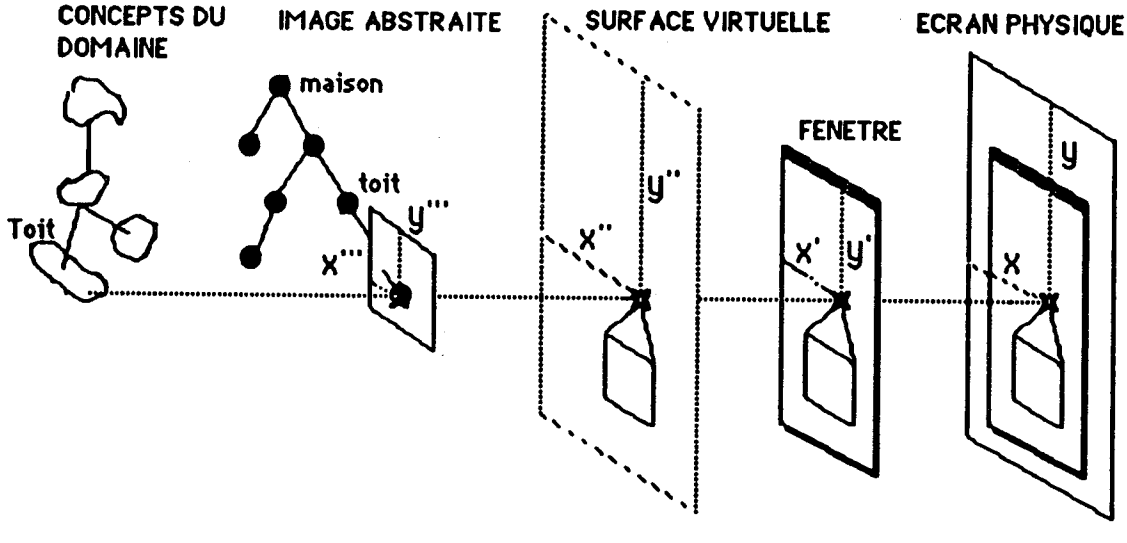


Figure 6.9 : Les étapes de traduction de l'adresse d'un point depuis l'écran jusqu'au concept du domaine.

Le schéma de la figure 6.9 donne une idée de la complexité du processus de traduction de l'adresse physique d'un point en l'identification d'une entité du domaine. En l'absence de machines à images abstraites, qui est le cas usuel, ce travail revient au programmeur d'application.

Si les mécanismes d'affichage et de désignation abstraits automatisent la traduction d'adresses entre les points de l'écran et les concepts du domaine, ils ne contrôlent pas pour autant la communication entre l'utilisateur et les services du système. Cette tâche relève de la gestion du dialogue.

4. Gestion du Dialogue

La littérature sur l'interaction homme-machine fait souvent référence aux notions de dialogue et de gestion de dialogue sans trop préciser de quoi il retourne. En s'inspirant des caractéristiques dominantes des dialogues entre individus, nous allons proposer un premier schéma de solution et relever les points clés de la modélisation logicielle du dialogue : le niveau d'abstraction des échanges entre l'application et l'interface, la localisation du contrôle de l'interaction au sein du système et la modélisation du fonctionnement de ce contrôle.

4.1. Observation du monde réel

Au sens général, un dialogue est une conversation entre plusieurs partenaires sur un sujet déterminé en vue d'aboutir à un accord ; c'est aussi le contenu de la conversation, c'est-à-dire les propos échangés. Si l'on s'en tient à cette double définition, la manifestation externe d'un dialogue est un ensemble d'expressions dont l'émission est contrôlée de manière répartie par un ensemble d'interlocuteurs.

Un contrôle implicite règle l'alternance des échanges. L'un des partenaires prend l'initiative du dialogue, soumet une expression (ou une suite d'expressions) à ses interlocuteurs. En raison du parallélisme des traitements, il arrive que les expressions se recouvrent dans l'espace temporel. L'élaboration des expressions ne s'effectue pas au hasard. Leur contenu sémantique et leur forme syntaxique dépendent du modèle conceptuel que chaque intervenant a de ses partenaires, du but recherché et de l'état de la conversation.

L'interaction entre un individu et un système informatique devrait s'organiser de manière similaire. Pour cela, il faut que le système soit capable de gérer un modèle de l'interaction qui respecte les phénomènes importants des dialogues entre sujets humains.

4.2. Un premier schéma de solution : Application et Interface

Un schéma de solution nous est fourni avec la notion d'Image et de variables psychologiques que D. Norman introduit dans sa Théorie de l'Action. Cette notion conduit à définir un système interactif comme l'assemblage de deux composants logiciels (voir la figure 6.10) : l'application et l'interface interactive.

- L'application regroupe, pour un domaine donné, l'ensemble des concepts qui correspondent aux variables psychologiques de l'utilisateur.
- L'interface interactive présente les concepts dans l'Image et assure la communication d'informations entre l'application et l'utilisateur.

Cette définition qui suggère de répartir le modèle conceptuel du système et l'état de l'interaction entre l'application et l'interface interactive, est compatible avec la méthode de conception et de description de CLG.

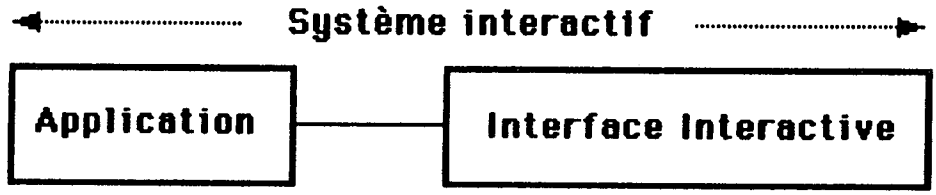


Figure 6.10 : Organisation schématique d'un système interactif.

4.2.1. Gestion du dialogue dans l'application

Dans l'application, le modèle conceptuel est l'ensemble des abstractions spécifiques au domaine qui satisfont aux besoins des tâches de l'utilisateur (ou variables psychologiques). Cet ensemble correspond aux entités et aux opérations conceptuelles tel que le définit le niveau de description sémantique de CLG. Il détermine la sémantique statique du système. La sémantique dynamique s'exprime par l'enchaînement des états de l'interaction. A chaque état correspondent les abstractions accessibles (ou inaccessibles) au monde externe. Pour l'application, l'interface interactive est le représentant du monde externe. C'est avec lui qu'elle communique. C'est de lui qu'elle reçoit les demandes d'accès aux abstractions (par exemple sous forme d'appels de fonction) ; c'est à lui qu'elle exprime ses changements d'état ou ses besoins en données. De même que le terminal abstrait rend un programme client indépendant des contraintes physiques, de même l'interface interactive sert à masquer le détail des échanges avec l'utilisateur.

4.2.2. Gestion du dialogue dans l'interface interactive

Dans l'interface interactive, le modèle conceptuel et l'état de l'interaction sont exprimés à l'aide d'abstractions spécialisées dans la communication homme-machine. Ces abstractions correspondent aux entités syntaxiques et aux éléments d'interaction tels que les définissent les niveaux syntaxique et interaction de CLG. Ces abstractions jouent le rôle de tampon entre les abstractions conceptuelles de l'application et les actions de l'utilisateur. Chacune participe à la réalisation de l'interface interactive. Chacune est une mini-interface interactive qui représente une part du modèle conceptuel et une part de l'état de l'interaction. Un assemblage particulier de telles entités définit un exemplaire d'interface interactive pour une application. Au niveau macroscopique, cette interface sert de traducteur entre les formalismes de l'application et la forme des expressions d'entrée.

Une fois le langage d'entrée fixé (qui s'appuie, on le rappelle, sur le choix préalable d'une métaphore d'interaction), la difficulté pour les réalisateurs de systèmes interactifs est de déterminer le formalisme de communication de l'application, et notamment le niveau d'abstraction des échanges entre l'application et l'interface.

4.3. Niveau d'abstraction des échanges entre l'application et l'interface

Le choix du niveau d'abstraction des échanges entre l'application et l'interface est important en raison de son incidence sur la possibilité d'ajuster l'Image par itération. Lorsque le niveau d'abstraction est bas (par exemple, un événement de type "clic souris"), les traitements nécessaires à la mise à niveau des expressions de l'interface avec les structures de l'application sont dilués dans le code de l'application. Lorsque les unités d'échange sont des entités de même niveau d'abstraction que celles de l'application, alors l'application est indépendante des techniques de présentation.

Pour certaines classes d'applications, la frontière entre les concepts du domaine et le formalisme de l'interface est nette. Par exemple, dans le domaine de l'édition structurée, l'application gère l'arbre abstrait et l'interface le présente sous une forme syntaxique adaptée (généralement, un texte formaté). Dans d'autres domaines, comme les systèmes de base de données, on observe un continuum de traductions plutôt qu'une délimitation bien marquée. Nous verrons au chapitre 8 comment le modèle PAC répond à cette idée nouvelle de "continuum de traducteurs".

Une fois le niveau d'abstraction des échanges déterminé, le réalisateur doit identifier la localisation du contrôle.

4.4. Localisation du contrôle au sein du système

La localisation du contrôle précise le centre d'arbitrage du flux des informations. Si l'on considère, en première approximation, que le système est organisé en deux composants, le choix se situe entre l'application et l'interface.

P. Tanner dans [Tanner 83] distingue trois formes de contrôle : interne, externe ou mixte. Le contrôle est interne lorsqu'il réside dans l'application. Il est externe lorsqu'il est maintenu par l'interface interactive. Il est mixte lorsqu'il est géré en alternance par l'application et l'interface.

4.4.1. Contrôle interne

Le contrôle interne présente deux inconvénients majeurs : enraciné dans l'application, il compromet la séparation modulaire entre les fonctions du domaine et les techniques de présentation ; mal utilisé, il ouvre la voie à des situations contraignantes pour l'utilisateur. A titre d'exemple, prenons le cas d'une application en attente d'une réponse. L'application, bloquée mais disposant du contrôle, ne peut traiter une requête intermédiaire qui aurait permis à l'utilisateur de répondre correctement à la première question.

4.4.2. Contrôle externe

Lorsque le contrôle est externe, l'interface interactive a la responsabilité d'appeler les fonctions de l'application en réponse aux actions de l'utilisateur. L'application peut alors se voir comme un "serveur sémantique". Le contrôle externe assure la séparation entre les fonctions du domaine et celles de la présentation. Il conduit à une meilleure décomposition modulaire bien qu'il risque d'imposer à l'implémenteur de l'application un style de programmation particulier. Du côté de l'utilisateur, le contrôle externe devrait éviter les situations contraignantes mentionnées précédemment. Contrairement au contrôle interne qui induit un dialogue dirigé par l'application, le contrôle externe est ouvert à la mise en œuvre de dialogues dirigés par l'utilisateur.

4.4.3. Contrôle mixte

Le contrôle mixte permet au système interactif de naviguer entre les deux extrêmes. Cette possibilité est utile en trois circonstances :

1. pour traduire des événements asynchrones par rapport aux actions de l'utilisateur (par exemple, l'arrivée d'un message électronique ou encore la sonnerie du réveil) ;

2. pour demander des renseignements complémentaires nécessaires à la poursuite du traitement (cas de la commande "more" sous unix ou cas des systèmes experts qui, typiquement,

demandent à l'utilisateur d'intervenir pour résoudre les conflits) ;

3. pour faciliter la réutilisation d'applications existantes.

Le contrôle mixte présente cependant un inconvénient. Mal utilisé, il ouvre la voie à une mauvaise structuration du système (engendrant des problèmes similaires au "goto" des langages de programmation). Un compromis raisonnable est un contrôle externe qui simule les effets du contrôle mixte : l'application n'a pas le contrôle mais exprime ses besoins auprès de l'interface. L'interface reflète immédiatement l'état de l'application sous forme d'expressions de sortie mais laisse autant que possible l'utilisateur libre de ses décisions. Nous verrons le détail d'une mise en œuvre de cette solution au chapitre 12 lors de la description d'APEX.

Une fois l'arbitre du dialogue choisi, il reste à déterminer un modèle de fonctionnement qui soit en accord avec le comportement de l'utilisateur.

4.5. Modélisation du fonctionnement du contrôle du dialogue

Le fonctionnement du contrôle du dialogue peut se modéliser selon deux techniques essentielles : les automates et les traitants d'événements.

La technique des automates est séduisante parce qu'on la maîtrise bien. En revanche, il faut savoir l'appliquer avec discernement de façon à tenir compte du caractère imprévisible du comportement de l'utilisateur. En effet, le système doit être capable, dans chaque état, de répondre avec précision à toutes les actions de l'utilisateur. Reportée à la programmation (ou à la spécification d'interfaces), cette condition se traduit par la programmation (ou la spécification) de toutes les actions utilisateur imaginables dans un état donné. Le caractère fastidieux et difficile de cette tâche a pour conséquence une programmation (ou une spécification) incomplète avec, à la clef, un utilisateur mal informé qui a comme seul choix le cheminement restreint de l'automate.

La notion de traitant d'événements est utilisée depuis longtemps dans les systèmes d'exploitation. Elle a l'avantage d'organiser les traitements d'un système en une multitude d'agents coopérants, prêts à réagir aux événements pertinents. En conception de système interactif, un agent peut se voir comme un interlocuteur de l'utilisateur. Comme dans un dialogue entre individus, l'utilisateur peut changer d'interlocuteur ou s'adresser à plusieurs partenaires simultanément. Le modèle multiagent qui encourage une conception de l'interface dirigée par les actions de l'utilisateur et le parallélisme, est en accord avec les principes ergonomiques énoncés au chapitre 4. Il sera décrit au chapitre 7 puis repris en détail au chapitre 8 avec la présentation du modèle PAC.

EN RESUME :

La figure 6.11 montre l'empilement des machines abstraites impliquées dans la réalisation d'un système interactif :

- les pilotes gèrent chacun une classe de terminaux physiques. Ils font partie du système d'exploitation.
- L'indépendance vis-à-vis du matériel se manifeste sous la forme d'un appareil abstrait intégré, en général, dans les systèmes de fenêtrage.
- La gestion des ressources de l'interaction est assurée par les systèmes de fenêtrage dont les concepts visibles aux programmes clients incluent les événements, les surfaces d'affichage, les fenêtres, etc.
- Les services d'affichage et de désignation sont réalisés à divers niveaux d'abstraction depuis la machine graphique élémentaire jusqu'aux machines à images abstraites.
- La gestion du dialogue s'effectue au sein d'un gestionnaire qui contrôle un ensemble d'entités de présentation au moyen d'un modèle de l'interaction.

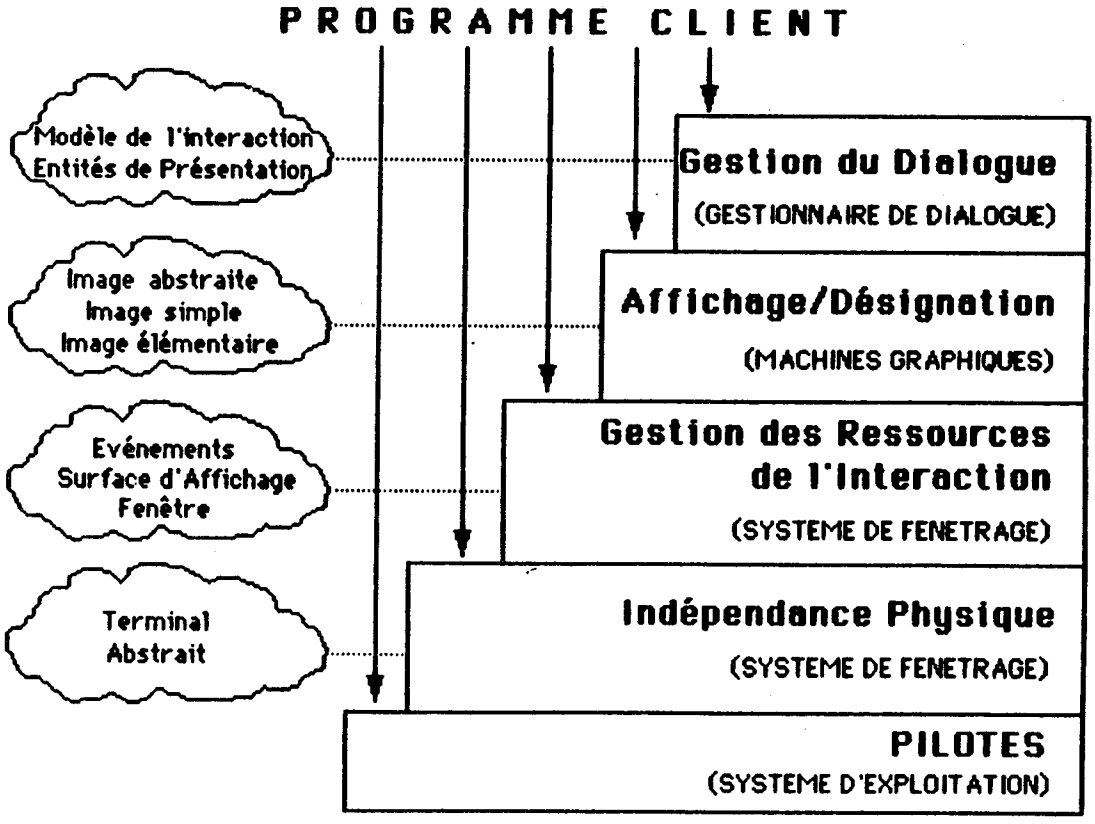


Figure 6.11 : Les niveaux de machines impliquées dans la réalisation d'un système interactif. Les flèches verticales symbolisent les appels directs d'un programme client aux fonctions d'une machine. Sur la gauche de la figure, les concepts manipulés dans chaque niveau. En caractère gras, les classes de services rendus. Entre parenthèses, les noms usuels des machines.

Aujourd'hui, on maîtrise assez bien les concepts des machines abstraites des couches les plus basses, et notamment ceux des systèmes de fenêtrage, des machines graphiques élémentaires et des machines à images simples. Ces logiciels constituent ce que j'appelle le "logiciel de base de l'interaction". A l'inverse, l'affichage abstrait et l'organisation du dialogue font encore l'objet de recherches actives. C'est précisément sur ces deux thèmes que s'est effectué l'essentiel de mon travail.

Chapitre 7

Les Modèles d'Architecture

1. Introduction

Les besoins en modèle d'architecture pour systèmes interactifs ne se sont manifestés que très récemment. En 1984, un groupe de travail SIGGRAPH définit un premier modèle d'interaction homme-ordinateur : le modèle Seeheim [Pfaff 85] présenté dans ce chapitre. L'année suivante, dans le cadre de la conférence "Computer Human Interaction", je participe à un groupe de travail sur le thème "User Interface Reference Models". Ce groupe comprend une vingtaine de chercheurs et d'industriels retenus pour leurs travaux dans le domaine de l'interaction homme-ordinateur. Gene Lynch dans [Lynch 86] donne un compte-rendu détaillé de l'approche suivie ainsi qu'une liste de résultats et de conclusions. En voici les éléments essentiels.

Avant même de définir un modèle, deux questions s'imposent : "Quel est le destinataire et quelle est l'utilité du modèle?". Le client privilégié d'un modèle d'architecture de systèmes interactifs est un réalisateur mais les utilisateurs de ces systèmes doivent en tirer le meilleur bénéfice. Cette observation indique que l'"intégration de l'utilisateur" doit constamment servir de principe directeur à l'élaboration du modèle. L'objet du modèle est de fournir au réalisateur une structure générique à partir de laquelle il est possible de construire un système interactif particulier. Cette structure décrit le flux des données entre l'utilisateur et l'application ; elle identifie les étapes de transformation des données et détermine l'agencement des composants qui assurent ces transformations. Ces composants sont des abstractions qui correspondent à des processus généraux ; ils sont indépendants des techniques de réalisation.

La nature des composants, leur arrangement et les transformations qu'ils opèrent constituent une solution au problème de la construction d'un système interactif. Ce problème se situe dans un espace à quatre dimensions : l'application, l'utilisateur, l'environnement de travail, et les niveaux de communication. A chaque dimension correspondent des propriétés intrinsèques et des besoins à satisfaire. Le tableau de la figure 7.1 donne quelques exemples. Nous relevons qu'une application peut avoir comme propriété intéressante la décentralisation de ses ressources et qu'elle doit être capable de résister aux pannes ; que l'utilisateur est caractérisé par un niveau d'expertise et qu'il a le droit de prendre l'initiative du dialogue ; que l'environnement comprend un type de terminal particulier et qu'il impose des limitations d'ordre administratif ; que les niveaux de communication exigent cohérence et concision.

Domaine de Définition	**APPLICATION**	BESOINS Gestion des erreurs Portabilité Maintenance Lancement Terminaison, etc. CARACTERISTIQUES Centralisation/Distribution des ressources Multi/Mono utilisateur Multi/Mono tâche Multi/Mono espace de données Interaction avec d'autres applications Temps réel/traitement par lots, etc.
	UTILISATEUR	BESOINS Droit à l'initiative Protection contre les erreurs Guidage, etc. CARACTERISTIQUES Niveaux d'expertise Limites sensorimotrices et cognitives, etc.
	ENVIRONNEMENT	CARACTERISTIQUES Bruit, confort Dispositifs physiques d'interaction Procédures administratives, etc.
	NIVEAUX DE COMMUNICATION	CARACTERISTIQUES Lexicales Syntaxiques Sémantiques BESOINS Cohérence Concision, etc.

Figure 7.1 : Problème de la réalisation d'un système interactif : son domaine de définition.

La structure générique servant de modèle de référence propose une solution au problème. Cette solution comprend l'identification des classes de service et la détermination des stratégies de répartition de ces services parmi les constituants logiciels. Les constituants logiciels sont reliés par des relations précises définies par un protocole d'échange. La figure 7.2 montre quelques exemples de services et de répartitions.

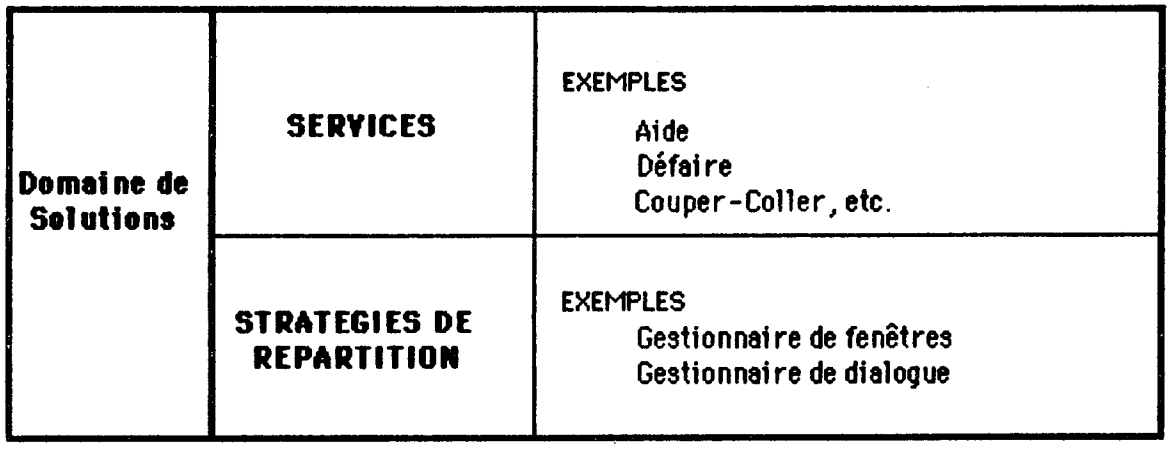


Figure 7.2 : Problème de la réalisation d'un système interactif : son domaine de solutions.

A cet objectif idéal s'oppose la réalité des propositions actuelles en matière de modèles d'architecture. Dans ce chapitre, nous envisageons trois modèles d'architecture qui offrent chacun une vue complémentaire sur la façon d'organiser les éléments constitutifs d'un système interactif : le modèle langage qui voit une analogie entre le dialogue humain et l'interaction homme-ordinateur ; le modèle entrée/sortie qui assemble le logiciel de l'interaction en niveaux fonctionnels ; le modèle multiagent qui organise un système interactif en un assemblage de réactifs à des stimuli. Le chapitre suivant est consacré au modèle PAC qui tend à se rapprocher de la définition du modèle idéal.

2. Le Modèle Langage

Les modèles d'architecture de type langage s'inspirent de l'analogie entre l'interaction homme-machine et les dialogues entre individus. Un dialogue s'appuie sur un langage qui permet d'externaliser la pensée. Deux individus vont pouvoir échanger leurs idées s'ils parlent le même langage. De la même façon, l'utilisateur et le système communiquent par l'intermédiaire d'un langage commun. Le paragraphe suivant indique les principes directeurs de la classe des modèles langage ; un exemple marquant de cette classe, le modèle Seeheim, sera ensuite présenté.

2.1. Principes directeurs

A l'image des modèles linguistiques, le langage utilisé par le système et l'utilisateur se définit selon trois composantes : sémantique, syntaxe et lexique.

- La sémantique définit la signification des phrases. Elle représente les concepts et le savoir-faire

dans un domaine donné. La sémantique est la description précise des classes d'objet auxquelles l'utilisateur et le système font référence dans leur conversation. Par exemple, si la tâche consiste à étudier des phénomènes thermiques, les classes d'objet intervenant dans les échanges comprennent la température, la quantité de chaleur et les opérations qui permettent de mettre en évidence les relations entre ces notions. Cet ensemble de concepts exportés vers l'utilisateur est un sous-ensemble des objets intervenant dans l'accomplissement des tâches pour lesquelles le système est prévu.

- La syntaxe définit la construction des phrases du langage à partir des éléments syntaxiques. Un élément syntaxique est une unité qui ne peut être décomposée plus avant sans perdre sa signification. Par exemple, les symboles graphiques qui représentent un thermomètre et un réchaud n'expriment plus l'idée de température et de chaleur si on les décompose en termes de primitives graphiques. Tout comme dans le domaine linguistique traditionnel, la validité syntaxique d'une phrase n'entraîne pas forcément sa validité sémantique. Par exemple, la phrase constituée du mouvement du thermomètre dans l'espace est syntaxiquement correcte. Elle n'a cependant aucun effet sémantique tant que le thermomètre n'est pas placé sur le réchaud.

- Le lexique définit la production des unités syntaxiques à partir d'un vocabulaire. Dans le cas du langage homme-ordinateur, ce vocabulaire comprend tout ce qu'il est possible de créer à partir des dispositifs d'entrée et de sortie : les caractères du clavier, les sons élémentaires d'un générateur, les primitives graphiques d'une bibliothèque. Par exemple, les primitives de tracé de droite et de remplissage de régions permettent de construire les éléments syntaxiques thermomètre et réchaud.

Foley et Van Dam furent les premiers à plaquer une vue linguistique sur l'interaction homme-ordinateur [Foley 84] mais le modèle Seeheim [Pfaff 85] a su le premier exprimer cette source d'inspiration sous une forme générique utilisable.

2.2. Le Modèle Seeheim

Le modèle Seeheim tient son nom du lieu où il a été défini. La figure 7.3 illustre la structure proposée. Celle-ci comprend trois composants logiques : la présentation, le contrôle du dialogue et l'interface avec l'application.

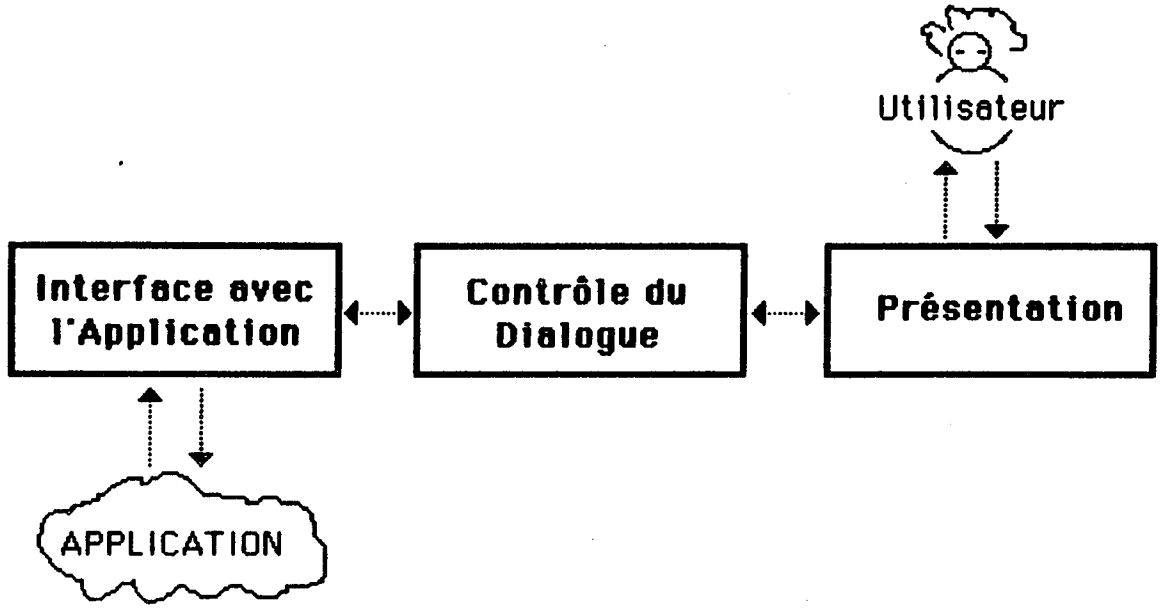


Figure 7.3 : Le modèle Seeheim.

2.2.1. Le Composant Présentation

Le composant Présentation est responsable de la représentation physique du système. Dans la terminologie de D. Norman, il en définit l'Image. Du point de vue linguistique, il effectue l'analyse lexicale du langage d'interaction. En tant que processeur, il convertit les informations entre le format du monde physique externe et celui du monde informatique interne. Plus précisément, la Présentation:

- lit les données en provenance des dispositifs physiques d'entrée. Ces données constituent les unités lexicales d'entrée du langage d'interaction.
- traduit les unités lexicales d'entrée en une forme abstraite compatible avec le monde informatique interne. Ces représentations définissent les unités syntaxiques du langage.
- reçoit des expressions abstraites en provenance du Contrôleur du Dialogue. Ces abstractions définissent les unités syntaxiques de sortie.
- interprète les unités syntaxiques de sortie dans les termes du lexique de sortie (primitives graphiques, sons élémentaires, etc.).

La Présentation est le seul composant du système à manipuler directement les dispositifs physiques d'entrée/sortie. Les autres constituants passent nécessairement par l'intermédiaire de la Présentation pour effectuer des échanges avec l'utilisateur. Ces opérations s'expriment en termes abstraits, les unités syntaxiques, dégagés des détails de bas niveau de l'interaction. Inversement, la Présentation ne s'adresse jamais directement à l'application. Les informations qu'elle émet sont nécessairement filtrées et enrichies par le Contrôleur du Dialogue.

2.2.2. Le Composant Contrôle du Dialogue

Le Contrôleur du dialogue joue le rôle de médiateur entre la Présentation et l'Interface de l'application. En tant que processeur linguistique, il est responsable de l'analyse syntaxique du langage d'interaction : les unités syntaxiques reçues de la Présentation sont assemblées en phrases. Les phrases syntaxiquement correctes correspondent à des requêtes et des données que l'utilisateur souhaite transmettre à l'application. Dans le sens inverse, le Contrôleur reçoit des phrases de sortie abstraites qu'il ventile vers les éléments spécialisés de la Présentation.

A ce rôle de médiateur et d'analyseur syntaxique, le Contrôleur ajoute celui de gestionnaire de l'état de l'interaction. L'ensemble des états possibles, leurs relations et la composition des phrases définissent la structure du dialogue entre l'utilisateur et l'application. La définition d'un état peut inclure l'ensemble des requêtes permises. Dans ce cas, le Contrôleur est en mesure d'effectuer une partie des vérifications sémantiques avant d'envoyer une requête à l'Interface de l'application.

2.2.3. Le Composant Interface avec l'Application

L'Interface de l'Application est le représentant de l'application auprès du Contrôleur. Il définit la vue que le Contrôleur a de l'application. Rappelons que l'application réunit les concepts et les opérations propres à l'accomplissement de tâches dans un domaine précis. Ces concepts et leurs opérateurs définissent la sémantique du langage d'interaction.

L'interface propose un mécanisme de conversion entre les concepts de l'application et les phrases du Contrôleur. Par exemple, il peut contenir les structures de données et les points d'entrée des fonctions que l'application exporte aux fins de l'interaction. L'Interface maintient alors la correspondance entre les requêtes abstraites élaborées par le Contrôleur et les concepts exportés de l'application. Il gère aussi la correspondance inverse entre les expressions produites par l'application et celles qui sont transmises au Contrôleur. L'Interface peut aussi contenir une description des contraintes d'utilisation des concepts exportés. Cette description permet au Contrôleur de vérifier la validité des requêtes de l'utilisateur avant d'appeler les services de l'application.

2.3. Evaluation des Modèles Langage

Le modèle langage a l'avantage de s'appuyer sur des concepts connus mais ceux-ci ne conviennent pas toujours au cas de l'interaction homme-ordinateur. Ces deux aspects contradictoires sont abordés dans les paragraphes qui suivent.

2.3.1. L'apport

L'apport du modèle langage se résume ainsi : définition d'un cadre de pensée, conception itérative des interfaces utilisateur, généricité et généralité.

- *Un cadre de pensée.* L'analogie dont s'inspirent les modèles langage fournit une explication simple du principe de fonctionnement des systèmes interactifs. Les notions de lexique, de syntaxe et de sémantique sont aujourd'hui bien comprises des informaticiens. Ceux-ci peuvent donc s'y référer utilement pour jeter les bases structurelles du système à construire.

- *Conception itérative des interfaces interactives.* La modularité sous-jacente au modèle, qui distingue les aspects lexicaux, syntaxiques et sémantiques, rend possible l'ajustement itératif de l'interface utilisateur : en théorie, la modification de la Présentation peut être effectuée sans mettre en cause les fonctions de l'application et réciproquement.

- *Généricité.* Le modèle Seeheim ne fait aucune hypothèse sur la nature des applications et n'est lié à aucune technique de réalisation. Son principe de base, qui distingue les techniques de présentation des services d'un domaine, a servi de guide à la réalisation d'un grand nombre de systèmes interactifs.

- *Généralité.* Le modèle langage n'est pas seulement applicable à la construction des systèmes interactifs. Il est également utile à leur conception et à leur réalisation automatisée.

  Parmi les méthodes de conception qui s'appuient sur la décomposition en niveaux de communication, nous retenons CLG [Moran 81], les travaux de Foley et Van Dam [Foley 84] et ceux de Mark Green [Green 85]. Toutes ces méthodes mettent en évidence les avantages de la décomposition en niveaux d'abstraction, depuis les concepts du domaine jusqu'aux fins détails de l'interaction : à chaque niveau correspondent des besoins, des compromis et des décisions auxquels le concepteur peut localement porter l'attention au lieu d'embrasser le problème dans un espace plus grand et non structuré.

  Le modèle langage a servi de base à de nombreux outils de génération automatique d'interfaces. A chacun des composants du modèle, il est naturel de faire correspondre un outil d'aide à sa création. Parmi les constituants de l'interface, le Contrôleur est celui qui a reçu le plus d'attention. Ce fait n'est pas surprenant si l'on considère la maîtrise des techniques de génération automatique d'analyseurs syntaxiques. Les outils de génération d'interface seront décrits au chapitre 13. Pour l'instant, indiquons qu'ils sont capables de créer le Contrôleur à partir d'une description de la syntaxe. La syntaxe s'exprime dans un formalisme adapté. Nous aborderons ces notations au chapitre 13. La sémantique et le lexique sont généralement exprimés dans un langage de programmation traditionnel. Les éléments syntaxiques sont le plus souvent prêts à l'emploi dans des bibliothèques spécialisées.

Conceptuellement, la distinction explicite entre sémantique, syntaxe et lexique, est séduisante. En pratique, cette vue qui considère l'interface utilisateur comme un serveur syntaxique et l'application comme un serveur sémantique, ne marche pas vraiment.

2.3.2. Les limites

Le modèle présente quatre inconvénients majeurs : la priorité est donnée à la forme de l'information, le traitement des formes est centralisée, le langage d'interaction est découpé en deux

sous-ensembles distincts, et le niveau d'abstraction des protocoles de communication entre les composants est trop imprécis.

- *Priorité à la forme des échanges.* Par définition, l'approche langage concentre l'attention sur la forme des échanges au détriment de leur dynamique. Pour le compilateur, qui travaille sur des expressions statiques, cette propriété n'est pas gênante. Pour l'interface utilisateur, la forme d'une expression d'entrée peut évoluer en parallèle avec la production d'une expression de sortie. Ou bien encore, plusieurs expressions d'entrée, éventuellement indépendantes, peuvent être spécifiées en parallèle. Contrairement au compilateur qui effectue, en séquence, la réception des expressions d'entrée, l'analyse des expressions, et la production des résultats, l'interface doit permettre l'entrelacement d'expressions d'entrée partiellement élaborées avec la production d'expressions de sortie. Les interfaces de manipulation directe qui exigent une réaction "immédiate et informative", illustrent parfaitement cette particularité.

- *Centralisation du traitement des formes.* La centralisation des traitements sémantiques, syntaxiques et lexicaux est inadaptée aux besoins de l'interactivité. Même les détails lexicaux les plus simples requièrent parfois un minimum de connaissance sémantique. Par exemple, les limites d'une boîte élastique peut dépendre de données de l'application. En outre, à l'observation de l'expérience, on constate que la frontière entre l'application (la sémantique) et l'interface (la syntaxe et le lexique) n'est pas toujours facile à déterminer. Au lieu d'une dichotomie en trois niveaux d'abstraction, un continuum d'abstractions semble correspondre davantage au processus de modélisation du "processeur humain".

- *Découpage du langage d'interaction en deux sous-langages.* Certaines applications du modèle, en particulier les générateurs d'interface, ou encore certaines interprétations du modèle comme celle de Foley et Van Dam [Foley 84, p. 220], décomposent le langage d'interaction en deux sous-langages : le langage d'entrée et le langage de sortie. L'un permet à l'utilisateur de communiquer avec le système, l'autre au système de communiquer avec l'utilisateur. Cette dichotomie du langage de l'interaction vient en contradiction avec des besoins courants en communication. En particulier, une expression produite par le système doit pouvoir servir d'expression d'entrée. Cette possibilité est l'essence même des opérations couper-coller. L'une des conséquences regrettables de la distinction entre langage d'entrée et langage de sortie est la dissymétrie fonctionnelle des générateurs d'interfaces. Ceux-ci ne supportent en réalité que la moitié du dialogue, c'est-à-dire le langage d'entrée.

- *Imprécision des protocoles de communication entre les composants.* Le découpage fonctionnel en trois niveaux est clair mais le modèle reste volontairement vague sur la nature des échanges entre les différents niveaux. Cette imprécision rend difficile la détermination des frontières entre les constituants. La conséquence fréquente de ce problème est un logiciel où s'enchevêtrent décisions sémantiques, syntaxiques et lexicales. L'exemple suivant illustre cette confusion. Il s'agit d'une version simplifiée d'un exemple tiré du chapitre 2 de [Foley 84]. Le système proposé permet à l'utilisateur d'organiser l'ameublement d'une pièce.

La figure 7.4 a) montre l'image initiale du système. Une fenêtre contient les contours de la pièce et le menu principal. Ce dernier indique les choix possibles entre l'ajout, le retrait d'un meuble, une demande de renseignements ou la terminaison. La figure 7.4 b) montre l'image lorsque l'utilisateur a fait le choix d'ajouter un meuble : le menu principal disparaît au profit d'une liste de symboles représentant chacun une catégorie de meuble (chaise, lit, table, etc.). En 7.4 c) l'utilisateur vient de sélectionner la catégorie fauteuil et se trouve placer un exemplaire de fauteuil à l'aide de la souris. Les figures suivantes indiquent le principe de la réalisation logicielle du système.

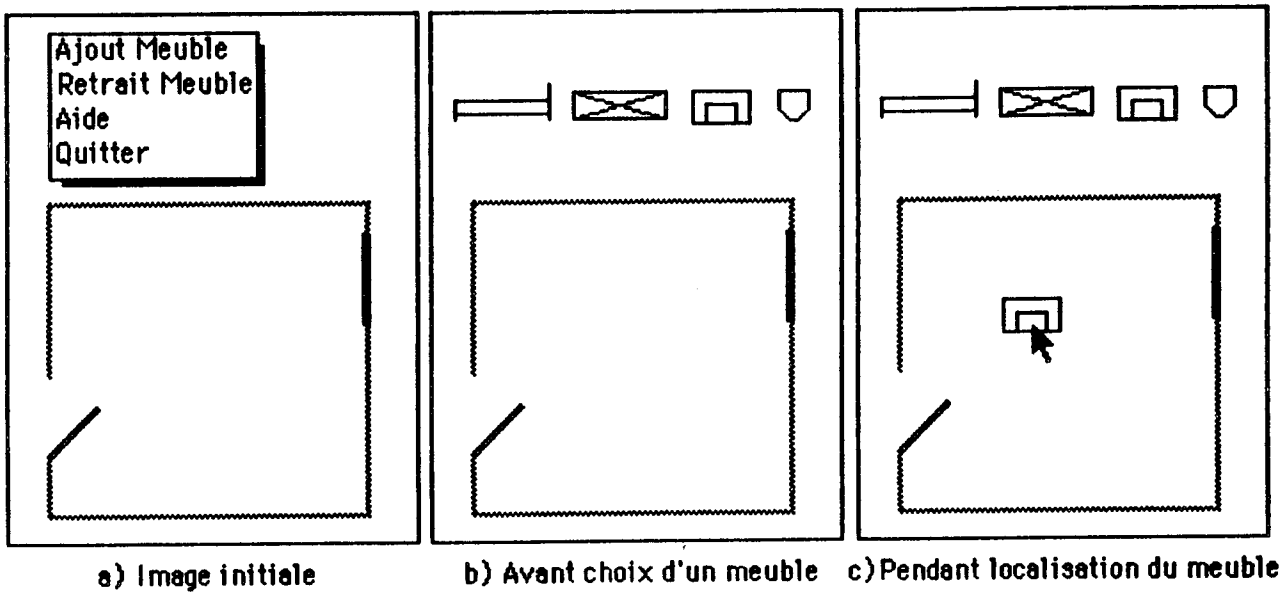


Figure 7.4 : Système d'ameublement d'une pièce : l'Image.

La figure 7.5 montre le programme principal : **controle** effectue les initialisations d'usage, crée la fenêtre et les menus, dessine les contours de la pièce, et affiche le menu principal. Il entre ensuite dans une boucle qui s'achève lorsque l'utilisateur sélectionne l'élément **Quitter** du menu principal. Cette boucle comprend l'attente d'une action utilisateur suivie du traitement de l'action. Par exemple, la sélection de l'élément **Ajout Meuble** du menu principal entraîne l'appel de la procédure **AjoutMeuble.**

```
programme controle
 ..... ;
debut
   finprog = FAUX;
   CreerFenetre (50, 50, 150, 200);
   CreerMenu (mPrincipal, "Ajout Meuble", "Retrait Meuble", "Aide", "Quitter");
   CreerMenu (mMeuble, lit, table, fauteuil, chaise, ...);
   DessinerPiece;
   AfficherMenu (mPrincipal, 10, 10);
   repeter
      AttendreAction (action);
      choix action parmi
         ajoutmeuble : AjoutMeuble;
         retraitmeuble : RetraitMeuble;
         ..... ;
         quitter : finprog  = VRAI;
         autres : AfficherTexte ("Choix illégal, essayer à nouveau", ...);
      finchoix ;
   jusqua finprog;
fin
```

Figure 7.5 : Le Contrôleur du dialogue du système d'ameublement.

La figure 7.6 décrit brièvement la procédure qui traite l'ajout d'un meuble. **AjoutMeuble** affiche le menu des symboles et procède de manière similaire au programme principal : attente d'une action puis interprétation de l'action. Le choix d'un symbole appelle la procédure **PlacerMeuble.**

```
procedure AjoutMeuble
 ..... ;
debut
    EffacerMenu (mPrincipal);
    AfficherMenu (mMeuble);
    ajoutFait = FAUX;
    repeter
        AttendreAction (action);
        choix action parmi
            lit : PlacerMeuble (lit);
            chaise : PlacerMeuble (chaise);
            ..... ;
            autres : AfficherTexte ("Choix illégal, essayer à nouveau", ...);
        finchoix ;
    jusqua ajoutFait;
fin
```

Figure 7.6 : Traitement des actions utilisateur principales dans le système d'ameublement.

La figure 7.7 correspond au traitement de la création d'un nouveau meuble avec la détermination de son emplacement dans la pièce. **PlacerMeuble** vérifie que la création d'un meuble est possible. Si l'autorisation est accordée, elle attend la désignation de l'emplacement. Cet emplacement est spécifié par la position du curseur lorsque l'utilisateur appuie sur l'un des boutons de la souris.

```
procedure PlacerMeuble (meuble)
 ..... ;
debut
   LocChoisie = FAUX;
   si non CreerMeuble (meuble)
   alors
     debut
       AfficherTexte ("Creation impossible");
       retour;
     fin
   repeter
       AttendreAction (action);
       choix action parmi
         loc : AfficherMeuble (loc.x, loc.y);
         bouton : LocChoisie = VRAI;
         autres : AfficherTexte ("action illégale",....);
       finchoix ;
   jusqua LocChoisie;
fin
```

Figure 7.7 : Désignation de l'emplacement d'un meuble du système d'ameublement.

Cet exemple logiciel ne tient pas compte de tous les cas d'annulation et de reprise mais suffit à révéler quelques entorses au modèle langage. **controle**, le programme principal, joue le rôle de contrôleur du dialogue. Il maintient un état en proposant un enchaînement des actions possibles : la décomposition hiérarchique en procédures et sous-procédures correspond à la réalisation d'un automate d'états finis. En tant que contrôleur, il ne devrait pas effectuer de choix lexicaux, ni opérer sur des entités lexicales. En réalité, la création de la fenêtre exige la spécification d'un emplacement et d'une dimension qui s'expriment en pixels, c'est-à-dire en termes du niveau lexical. Dans la procédure **PlacerMeuble**, la réception de la position du meuble nouvellement créé relève également du niveau lexical. Par cohérence avec le modèle nous pouvons considérer que cette procédure appartient au niveau lexical. Cette décision vient malheureusement en contradiction avec la présence de l'appel d'une action du niveau sémantique : **CreerMeuble**.

La confusion entre les niveaux de description fragilise la maintenance du logiciel. Elle est sans doute acceptable pour les systèmes simples comme celui que nous venons de décrire. Dans le cas général, elle rend impossible la mise au point itérative des interfaces. Le modèle langage définit donc un cadre utile à la compréhension générale de l'organisation d'un système interactif. En pratique, son imprécision le rend inopérant pour la réalisation de gros logiciels interactifs.

Le modèle langage s'inspire des théories et techniques du traitement des langages. Le modèle décrit au paragraphe suivant s'appuie sur les techniques de traitement des opérations d'entrée/sortie.

3. Le Modèle entrée/sortie

Les modèles d'architecture de type entrée/sortie mettent l'accent sur l'échange et la transformation d'informations. La forme des échanges et la nature des transformations définissent le découpage fonctionnel du système. Ce découpage peut s'envisager comme un empilement de machines abstraites effectuant chacune des opérations d'entrée/sortie et des traitements à un niveau d'abstraction donné. Il peut s'envisager aussi comme une organisation de processus communiquant selon des protocoles précis. Nous allons considérer ces deux perspectives dans les paragraphes suivants puis donner quelques éléments d'évaluation.

3.1. Principes directeurs : vue en couches

La vue en couches modélise un échange avec l'utilisateur comme une succession d'opérations d'entrée/sortie et de traitements. Cette suite de transformations est assurée par une hiérarchie de machines abstraites [Coutaz 85b, Coutaz 86a, Coutaz 86b, Coutaz 86c]. La machine de plus bas niveau interprète les actions physiques de l'utilisateur. Celles-ci sont progressivement transformées dans la hiérarchie en notions abstraites interprétables par l'application. Dans le sens inverse, les concepts de l'application sont traduits en direction de l'utilisateur en plusieurs étapes jusqu'à la rencontre de la machine physique. Cette modélisation d'une opération d'entrée/sortie à plusieurs niveaux d'abstraction a déjà été présentée au chapitre précédent. La figure 7.8 en rappelle la définition.

La décomposition en niveaux d'abstraction fournit une base saine à la définition d'architectures modulaires et de protocoles de communication. Elle imite en cela le modèle OSI, à l'origine de la série des protocoles ISO pour la connexion d'ordinateurs et de systèmes distants. A la différence de la structure en couches strictes du modèle OSI qui impose à une machine de niveau i de ne communiquer qu'avec les machines adjacentes de niveaux i+1 ou i-1, le modèle entrée/sortie suppose, pour chaque machine, l'existence d'une interface accessible directement à l'application. Les flèches verticales de la figure 7.8 symbolisent cette possibilité. L'intérêt est de pouvoir intégrer dans le modèle les applications qui n'ont pas besoin de tous les niveaux de services proposés. Par exemple, la présentation des concepts d'une application ne justifie pas systématiquement l'usage d'une machine à images abstraites, ou encore les concepts manipulés par une application ne se situent pas forcément au dessus des entités de dialogue mais peuvent représenter des entités proches de l'appareil physique.

Sans la possibilité d'accès direct, les performances de telles applications souffriraient inutilement des coûts de transfert entre couches logicielles.

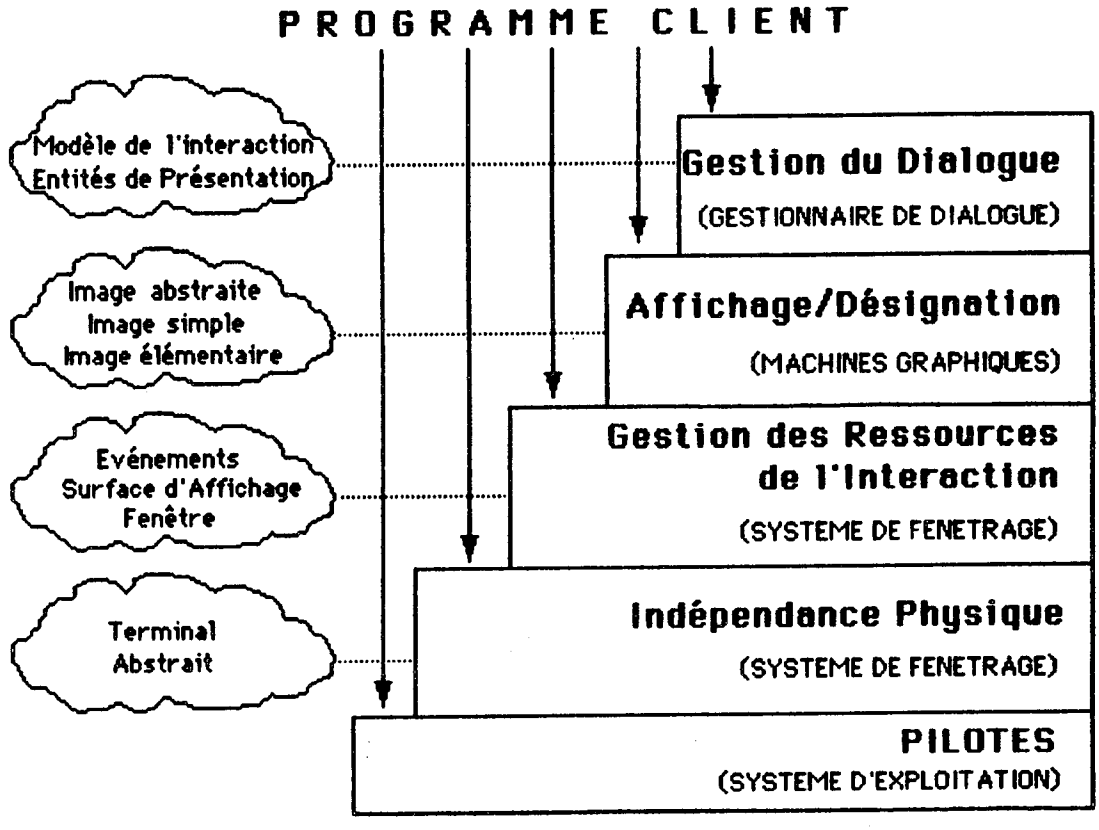


Figure 7.8 : Le modèle entrée/sortie : vue en couches.

En permettant à l'application d'avoir accès à tous les niveaux d'abstraction, le modèle évolue d'une vision en niveaux vers une organisation modulaire.

3.2. Principes directeurs : vue modulaire

La figure 7.9 présente une vue modulaire de l'architecture logicielle d'un système interactif. Cette vue complète le modèle de référence proposé par Keith Lantz dans [Lantz 86] dans lequel la notion de serveur d'images n'apparaît pas.

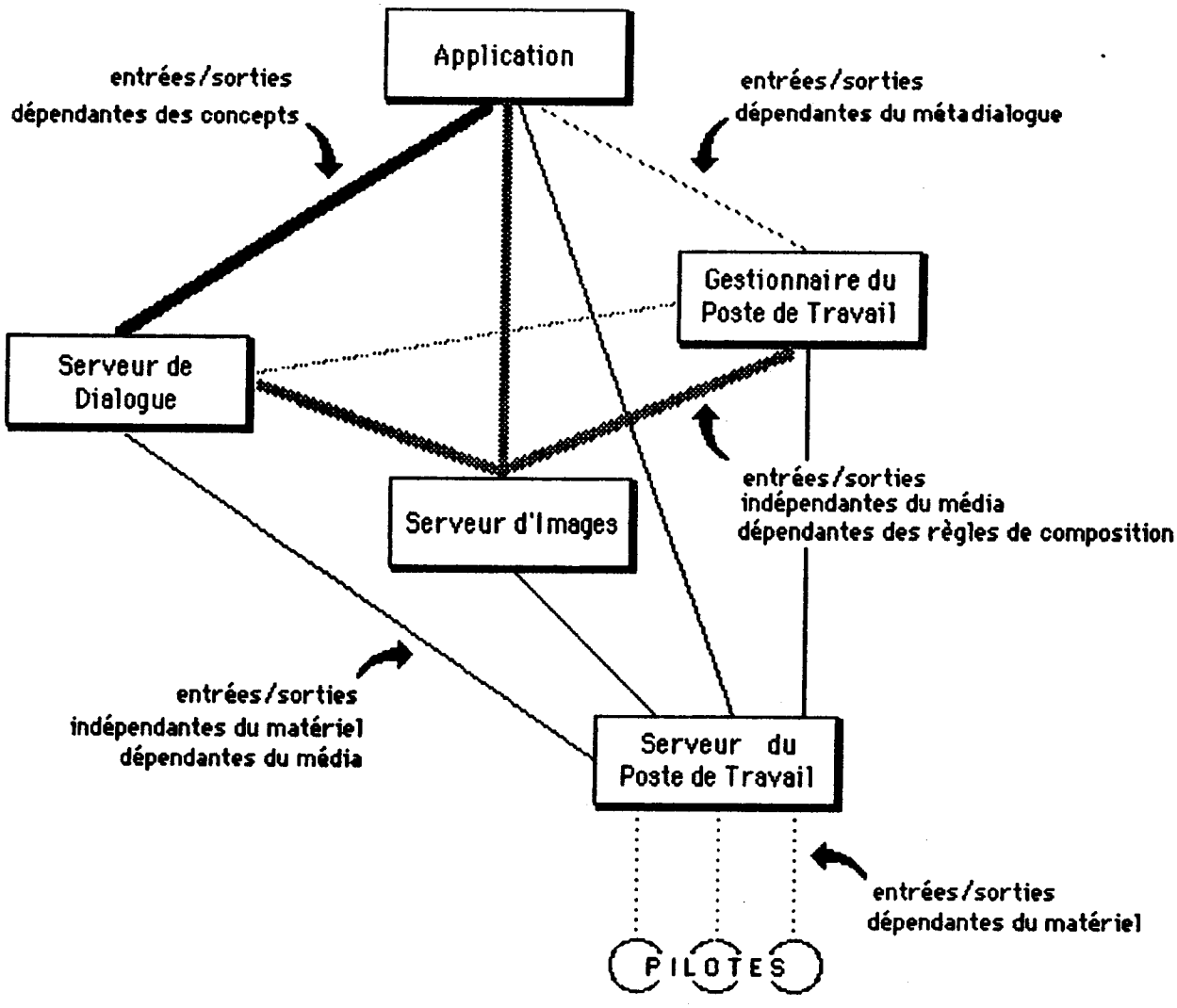


Figure 7.9 : Le modèle entrée/sortie : vue modulaire.

Les constituants nécessaires à la réalisation de l'interface homme-ordinateur d'un poste de travail comprennent : les pilotes, le serveur du poste de travail, le serveur d'images abstraites, le serveur de dialogue et l'application :

- un pilote effectue des opérations d'entrée ou de sortie sur une unité physique d'échange. Son interface est directement liée aux caractéristiques de fonctionnement de l'unité. Au niveau des pilotes, l'expression des entrées et des sorties dépend donc du matériel.

- le serveur du poste de travail gère l'ensemble des ressources physiques impliquées dans les échanges (les unités physiques), assure leur partage entre les clients et masque leur fonctionnement. L'interface qu'il propose aux clients est indépendante du matériel mais conserve la notion de classes d'unités. Une telle classe définit une catégorie de moyen de communication : graphique, son, vidéo. Les expressions d'entrée/sortie adressées au serveur du poste sont donc indépendantes du matériel mais dépendent des médias.

- le serveur d'images abstraites définit un média structuré abstrait. Alors que le serveur du poste de travail propose dans son protocole, des requêtes d'échange élémentaires, le serveur d'images organise ces éléments en entités composées. Les attributs de restitution de ces entités sont abstraits vis-à-vis des supports de communication car ils ne dépendent pas des

caractéristiques des médias définis par le serveur du poste de travail. Par exemple, l'attribut abstrait "mise en évidence" d'un constituant d'image peut se traduire par une vidéo inverse ou encore par un signal sonore. La vidéo inverse et le signal sonore relèvent tous deux du protocole du serveur du poste. Les entrées/sorties effectuées entre le client et le serveur d'images sont donc indépendantes des médias mais dépendent des règles de composition disponibles dans le serveur d'images.

- le serveur de dialogue définit un dialogue abstrait : il masque au client la présentation de ses concepts ainsi que le détail de l'enchaînement du dialogue avec l'utilisateur. Si l'on reprend le modèle langage, ce serveur inclut le Contrôleur du dialogue ainsi que la Présentation. Pour s'exprimer, celle-ci peut faire appel aux fonctions du serveur d'images ou bien directement au serveur du poste de travail. En tout état de cause, les entrées/sorties effectuées entre le client et le serveur de dialogue sont indépendantes de tout choix de présentation mais dépendent des concepts du client.

- l'application définit un serveur de concepts pour un domaine donné. Une application particulière dans un système multitâche est le gestionnaire du poste de travail. Un système multitâche permet à l'utilisateur d'exécuter simultanément plusieurs applications (applications prises ici au sens usuel). Le domaine du gestionnaire du poste de travail concerne le lancement et la terminaison d'applications, le changement d'application courante c.-à-d. de l'interlocuteur privilégié de l'utilisateur à un instant donné. A ces fonctions correspond nécessairement une interface interactive. Cette interface assure une sorte de métadialogue : un dialogue au dessus des dialogues entretenus avec chacune des applications. Le shell d'Unix et les "desktop managers" sont des exemples connus de gestionnaires de poste de travail.

Le modèle de la figure 7.9 représente le cas d'architecture le plus courant. Sur ce schéma de base, viennent se greffer des extensions : exploitation de l'environnement de multiprogrammation, raffinement de certains serveurs, enrichissement du protocole de communication.

- un environnement de multiprogrammation rend possible la cohabitation de plusieurs serveurs de la même classe. En particulier, les applications en cours d'utilisation ont chacune leur serveur de dialogue activé. L'ensemble de ces serveurs est alors contrôlé par le gestionnaire du poste de travail. La multiprogrammation permet aussi d'exécuter simultanément plusieurs gestionnaires de poste de travail. A ce propos, rappelons que le serveur de base de X (qui est un serveur de poste de travail) accepte l'exécution simultanée de plusieurs gestionnaires de fenêtre (qui sont chacun des serveurs de poste de travail).

- une seconde possibilité, non représentée dans le schéma mais détaillée dans la figure 7.10, est le découpage du serveur du poste de travail en deux niveaux : un premier niveau définit un terminal abstrait indépendant de l'appareillage physique ; le second niveau est un éventail de serveurs de poste de travail. A titre d'illustration, SUN MicroSystems s'emploie à définir un niveau logique sur lequel peuvent simultanément cohabiter les serveurs de base X et NeWS.

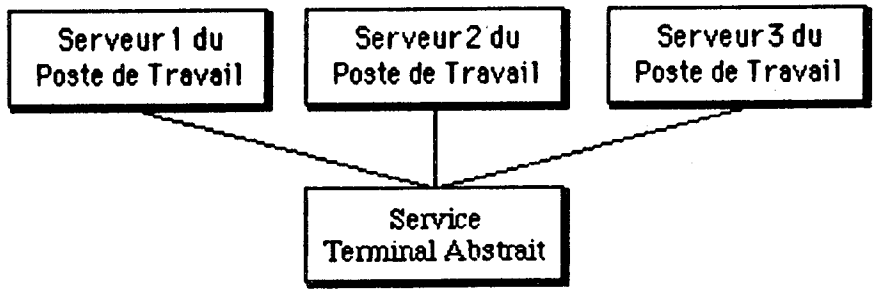


Figure 7.10 : Cohabitation de plusieurs serveurs de poste de travail.

- la dernière subtilité concerne la puissance d'expression des informations échangées au cours d'une opération d'entrée/sortie. L'information peut être une requête, c'est-à-dire une instruction simple aussitôt interprétable par le récepteur, ou bien un assemblage cohérent d'instructions composées, c'est-à-dire un véritable programme. Dans le second cas, l'émetteur a la possibilité de télécharger des concepts et des comportements spécifiques. Le serveur est alors une machine programmable. Cette technique du téléchargement a été exploitée pour la première fois par NeWS qui utilise PostScript comme langage de communication.

A la différence de la vue en couches qui met l'accent sur les niveaux d'abstraction, la vue modulaire structure l'espace des logiciels interactifs en serveurs communicants. Un serveur est ici considéré comme un ensemble de services cohérents. Il peut se voir comme un module relié à l'espace d'exécution du client ou bien comme un lieu d'activités dans un espace d'exécution différent de celui du client. Le choix entre les deux techniques dépend de la structure d'accueil et de la vocation du serveur. Dans un environnement de monoprogrammation, comme celui du Macintosh, on aura souvent recours à la solution du module lié (ou de la bibliothèque partagé) alors que dans un environnement multitâche, comme celui d'Unix, un serveur sera le plus souvent un processus du système hôte. A ces deux solutions de base s'ajoute la technique des processus légers qui permet de créer des activités parallèles au sein d'un processus système.

En raison de leurs fonctions, certains serveurs doivent être capables de mener de front plusieurs activités. Par exemple, un serveur de dialogue doit être capable de piloter simultanément une multitude d'objets de présentation et traiter les requêtes de l'application. Il est commode, dans ce cas, de déléguer les traitements à des microprocessus sans avoir à payer le prix de la gestion des contextes du système hôte. Le serveur NeWS et le squelette d'application APEX [Coutaz 87b], qui inclut un serveur de dialogue, comprennent un noyau de processus légers.

3.3. Evaluation du modèle entrée/sortie

3.3.1. L'apport

Le modèle entrée/sortie concentre la description sur les constituants logiciels qui interviennent nécessairement dans l'utilisation d'un poste de travail. Cette description fournit une vision globale des transformations appliquées aux informations d'échange, depuis l'action de l'utilisateur jusqu'à son effet dans l'application et inversement, depuis les concepts de l'application jusqu'aux primitives de fonctionnement du matériel.

La structure en couches est utile à la compréhension du rôle des différents niveaux de traitement tandis que la vue modulaire se rapproche davantage de la dynamique des échanges et des problèmes

de réalisation. Cependant, la généricité du modèle a ses contreparties.

3.3.2. Les limites

Le modèle entrée/sortie indique les grands niveaux de traitement. Certains, tels les serveurs et les gestionnaires de poste de travail, correspondent à des services généraux dont on maîtrise assez bien la réalisation. Ces services sont généralement disponibles sur toutes les stations de travail. D'autres, comme les serveurs de dialogue, se rapprochent au contraire des applications. Ils nécessitent alors une conception sur mesure. Pour ces serveurs, le modèle entrée/sortie ne fournit malheureusement aucune indication sur la façon de les organiser.

En résumé, le modèle entrée/sortie répartit davantage les échanges et les traitements que le modèle langage mais les niveaux d'abstraction de cette répartition sont encore trop grossiers pour guider le réalisateur dans la mise en place des détails fins de l'interaction. Le modèle multiagent, présenté au paragraphe suivant, offre le degré de souplesse voulue.

4. Le Modèle Multiagent

Le modèle multiagent s'inspire du fonctionnement des systèmes de type stimuli-réponses. Un tel système est un ensemble organisé d'agents capables de réagir à des phénomènes externes déterminés (des stimuli) et de produire à leur tour des stimuli. Cette notion d'agent réactif est à rapprocher du concept d'objet. Les paragraphes suivants exposent respectivement les principes directeurs du modèle et l'application de la programmation par objets à sa mise en œuvre. L'intérêt de cette approche est indiqué dans un dernier paragraphe. Il sera développé au chapitre suivant avec la description du modèle PAC.

4.1. Principes directeurs

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

- un événement est l'équivalent d'un stimulus : il appartient à une classe et véhicule une information dépendante de la classe ; il est produit par un émetteur et se détecte par des récepteurs ;
- un agent est un système de traitement de l'information : il possède des récepteurs et des émetteurs pour acquérir et produire des événements ; il dispose d'une mémoire à deux niveaux: l'une pour enregistrer les événements détectés par les récepteurs, l'autre pour mémoriser un

état; il comprend un processeur cyclique spécialisé dans le traitement d'une ou plusieurs classes d'événements ; le processeur ne traite qu'un événement à la fois. Lorsque le traitement d'un événement est achevé, le processeur extrait, s'il en existe un, l'événement enregistré suivant. La politique de séquencement pratiquée sur les événements est spécifique à l'implémentation ;

- lorsqu'un émetteur produit un événement, les récepteurs sensibles à la classe de l'événement sont activés. L'événement est enregistré chez les agents propriétaires des récepteurs actifs. L'événement est traité par chaque agent qui l'a enregistré. Le résultat d'un traitement se traduit généralement par un changement d'état de l'agent et par l'émission de nouveaux événements. L'état fournit à tout instant une image de l'évolution d'un agent. Parmi les informations qui participent à la dynamique du comportement, citons le filtre qui détermine, à un instant donné, les classes d'événements auxquels l'agent est sensible.

Le modèle multiagent se caractérise par une organisation fortement modulaire, des traitements exécutés en parallèle et une communication par événements. Ce modèle abstrait est à rapprocher des modèles à objets pour lesquels il existe des outils de réalisation.

4.2. Le modèle multiagent et le modèle à objets

Le modèle à objets tel que l'appliquent les langages de programmation par objets, organise le monde en classes et en exemplaires. Une classe est une matrice qui permet de produire des exemplaires aux propriétés identiques. Les propriétés comprennent des attributs, des opérateurs et des contraintes :

- un attribut est une unité de description déclarative. Il est manipulable par l'exécution d'opérateurs ;

- un opérateur est une unité de description procédurale. Sa signature, qui comprend un nom symbolique et une spécification de paramètres, définit son format d'utilisation. Des contraintes traduisent son effet ;

- les contraintes expriment des relations permanentes entre attributs. Elles se répartissent en deux catégories : les unes expriment les conditions d'activation des opérateurs, les autres des assertions valides après leur éxécution. Toutes servent à la vérification automatique du bon fonctionnement de l'exemplaire.

Les attributs, les opérateurs et les contraintes déterminent le comportement visible des exemplaires d'un type mais ne font aucune hypothèse sur les mécanismes qui le réalisent. La distinction entre comportement externe et mécanismes internes est à la base de toute technique d'abstraction. Celle-ci a été largement exploitée, avec quelques extensions, dans les langages de programmation par objets.

Les exemplaires de classe, appelés le plus souvent "objets", communiquent par messages. Un message contient un objet destinataire, un sélecteur, et des données. Le sélecteur est un nom symbolique qui désigne une méthode du destinataire. Les données du message sont utilisées comme

paramètres de la méthode. A ce modèle de base, les langages par objets ajoutent la possibilité d'héritage.

L'héritage permet d'exprimer le fait qu'une classe n'est, selon l'expression de B. Meyer, "ni vraiment identique, ni vraiment différente" d'une autre classe [Meyer 87]. Ce fait s'observe fréquemment dans la vie courante : un chat n'est pas vraiment un chien mais tous les deux sont des carnivores ; un histogramme n'est pas vraiment un camembert mais tous deux représentent une énumération de valeurs. Un langage à objets permet de définir le chien comme une sous-classe de la superclasse carnivore. On dit aussi que le chien est une spécialisation de la notion de carnivore.

Une sous-classe hérite des attributs et des méthodes définis dans la superclasse. A ce patrimoine, la sous-classe ajoute des propriétés spécifiques. Ces propriétés sont de nouveaux attributs et de nouvelles méthodes ; elles peuvent être aussi des redéfinitions d'attributs et de méthodes de la superclasse. Une redéfinition s'appelle une surcharge. Selon les langages, l'héritage est simple ou multiple. L'héritage est simple lorsqu'une classe est au plus sous-classe d'une classe. C'est le cas des langages Smalltalk [Goldberg 84] et Guide [Krakowiak 87]. L'héritage est multiple lorsque, comme dans Loops [Bobrow 83], une classe peut être sous-classe de plusieurs classes. L'avantage de l'héritage simple est la clarté sémantique, celui de l'héritage multiple, la commodité d'expression. De manière générale, l'héritage encourage l'écriture de logiciels modulaires : tout comportement est réutilisable, extensible et adaptable. Nous verrons au chapitre 12 que ces possibilités s'appliquent directement à la réalisation de squelettes d'application, noyaux réutilisables de gestionnaires de dialogue.

Si la programmation par objets est une technique adaptée à la réalisation d'outils logiciels interactifs, la notion de classe présente des propriétés intéressantes pour la mise en œuvre du modèle multiagent. En effet, une classe définit une catégorie d'agents :

- les opérateurs d'une classe constituent le répertoire d'instructions d'un processeur d'agent. Les paramètres décrivent les opérandes d'une instruction.
- les attributs constituent les éléments de la mémoire. Ils modélisent l'état de l'agent.
- les contraintes spécifient la sémantique des instructions du processeur. Une contrainte de préconditions, qui régit l'activation d'un opérateur, peut se voir aussi comme un filtre.
- l'appel d'un opérateur ou l'envoi d'un message modélisent l'émission d'événements.

Le modèle multiagent retrouve dans l'approche par objets les thèmes de la modularité et de la communication. Un objet et un agent sont tous deux des entités aux traitements extrêmement spécialisés. Tous deux "décident" de leur état : celui-ci ne peut être manipulé directement par autrui mais résulte d'un traitement sollicité par autrui.

4.3. Evaluation du modèle multiagent

L'évaluation du modèle entrée/sortie du paragraphe 3.3 fait ressortir l'absence d'indication sur la façon de structurer le serveur de dialogue. Le modèle langage nous apporte un début de réponse en proposant une décomposition en deux unités : le Contrôleur et la Présentation. Cette dichotomie suppose que les entrées soient traitées comme un flot séquentiel d'entités lexicales et syntaxiques. La séquentialité est une propriété intéressante pour le concepteur et le réalisateur d'interfaces interactives mais elle ne correspond pas au comportement imprévisible de l'utilisateur. Ce comportement se traduit en particulier par la production entrelacée d'expressions : l'utilisateur commence la spécification d'une expression, passe à une seconde expression, revient à la première, etc. en fonction de l'état de son plan de résolution. Chaque minidialogue correspond à l'un des fils d'activité mentale de l'utilisateur.

A chaque fil d'activité, il est naturel d'associer un agent spécialisé dans la conduite du minidialogue. L'état de l'agent permet de modéliser celui du minidialogue. Le fil est alors interruptible: l'agent dispose de l'information nécessaire à la poursuite de l'activité. La reprise a lieu à la détection d'un événement. Les événements véhiculent de l'information. Ils permettent en particulier de modéliser les actions de l'utilisateur. Lorsqu'un fil d'activité est trop complexe ou trop riche, il est raisonnable de le représenter, non pas comme un agent unique, mais comme un assemblage d'agents qui coopèrent dans la conduite de l'activité.

En résumé, le modèle multiagent remplace le serveur dichotomique séquentiel par une organisation d'acteurs qui travaillent en parallèle et qui coopèrent. A la gestion centralisée de l'état de l'interaction et de la présentation se substitue une totale répartition des charges. Ce parallélisme et cette répartition sont les conditions nécessaires à la prise en compte des méthodes de résolution des utilisateurs. La répartition présente en sus deux retombées intéressantes : celui d'une forte modularité et l'ouverture vers les traitements physiquement distribués :

- l'agent définit la granularité de la modularité. Il est donc possible de modifier localement un comportement sans mettre en cause le fonctionnement de l'ensemble. La centralisation du modèle langage rend au contraire cet ajustement particulièrement périlleux.
- l'agent définit l'unité d'exécution. Il est donc possible de l'exécuter sur un calculateur physiquement distant du lieu de création. Cette propriété est essentielle à la réalisation d'interfaces homme-ordinateur véritablement réparties tel Colab [Stefik 87].

Le modèle multiagent structure un système interactif en unités de traitement qui coopèrent dans la conduite de l'interaction. Il ne fournit cependant aucune indication sur la façon de l'organiser. Le modèle PAC, décrit au chapitre suivant, montre comment élaborer une telle organisation.

EN RESUME :

Les modèles ici présentés s'adressent aux réalisateurs de systèmes interactifs. Ils ont pour objet la fourniture de structures génériques à partir desquelles il est possible de construire des exemplaires de systèmes interactifs. Nous avons envisagé trois organisations complémentaires : celles des modèles entrée/sortie, langage et multiagent.

- *Le modèle entrée/sortie* représente l'interaction comme une succession d'opérations d'échange et de traitement s'effectuant à divers niveaux d'abstraction : depuis les actions physiques de l'utilisateur jusqu'aux concepts de l'application et, inversement, depuis les concepts jusqu'aux caractéristiques de fonctionnement des unités physiques de sortie. Le découpage fonctionnel en niveaux d'abstraction peut se voir comme un empilement de machines abstraites ou bien comme un ensemble de modules communicants. La vue modulaire lève la contrainte qui consiste à ne communiquer qu'avec les niveaux d'abstraction adjacents.

  Le modèle entrée/sortie offre une vue synthétique de l'ensemble des logiciels utiles à la construction des systèmes interactifs mais il ne fournit aucune indication sur la façon d'organiser les composants chargés du dialogue proprement dit.

- *Le modèle langage* concentre la représentation sur le dialogue. Il s'inspire de l'analogie entre le dialogue humain et l'interaction homme-ordinateur : un dialogue a lieu dans un langage. A l'image des modèles linguistiques, le langage utilisé par le système et l'utilisateur se définit selon trois composantes : sémantique, syntaxe et lexique. A ces trois aspects, le modèle langage fait correspondre un constituant logiciel : la Présentation effectue l'analyse lexicale ; le Contrôleur de Dialogue est responsable de l'analyse syntaxique et gère l'état de l'interaction ; l'Interface avec l'application définit auprès du Contrôleur la vue sémantique du système. L'application réalise la sémantique proprement dite.

  Le modèle langage définit un cadre de pensée simple sur le fonctionnement des systèmes interactifs. Il ouvre aussi la voie à la conception itérative des interfaces. En pratique, la centralisation des traitements sémantiques, syntaxiques et lexicaux suppose que les expressions d'entrée soient traitées comme un flot séquentiel. Malheureusement, cette séquentialité ne correspond pas au comportement opportuniste de l'utilisateur qui produit ses expressions selon plusieurs fils d'activité.

- *Le modèle multiagent* concentre l'effort sur la mise en œuvre de dialogues à plusieurs fils d'activité. Il représente un système interactif comme un ensemble d'acteurs spécialisés chacun dans la gestion d'un fil de dialogue. Un agent est un système de traitement complet doté d'une mémoire, d'un état, d'un processeur et de récepteurs-émetteurs d'événements. Il réagit aux événements externes auxquels ses récepteurs sont sensibles. La sensibilité d'un récepteur dépend de l'état de l'agent et s'exprime au moyen d'un filtre.

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

Chapitre 8

Le Modèle PAC

1. Introduction

Nous avons jusque-là énuméré les classes de modèles utiles à la construction de logiciels interactifs : le modèle entrée/sortie offre une vue synthétique générale mais superficielle de l'interaction ; le modèle langage fournit davantage d'indications sur la façon d'organiser les traitements de l'interaction mais il est inadapté aux dialogues à plusieurs fils d'activité ; le modèle multiagent est résolument orienté vers les activités parallèles mais il ne précise pas comment les organiser.

L'objet du modèle PAC [Coutaz 87c, Coutaz 87d] est d'intégrer les aspects remarquables du modèle langage aux avantages du modèle multiagent. Le réalisateur de systèmes interactifs peut ainsi s'appuyer sur une technique connue : la décomposition des traitements en niveaux lexical, syntaxique et sémantique, tout en tirant profit des possibilités de la modularité et du parallélisme. Rappelons que la modularité et le parallélisme contribuent tous deux à une meilleure prise en compte de l'utilisateur. Le premier favorise l'ajustement iteratif de l'interface, le second ouvre la voie à des interfaces plus souples dirigées par l'utilisateur. Faciliter le travail du réalisateur au meilleur bénéfice de l'utilisateur sont précisément les attributs recherchés d'un modèle d'architecture.

Ce chapitre s'organise comme suit : une définition du modèle PAC est présentée au paragraphe 2. Celle-ci est suivie au paragraphe 3 d'une description de l'intérêt du modèle. Le chapitre s'achève par une comparaison de PAC avec des modèles approchants. Des expériences effectuées avec le modèle font l'objet du chapitre suivant.

2. Définition du modèle PAC

PAC s'appuie sur une combinaison des vues langage, en couche et multiagent. Le modèle de Norwood Sisson [Sisson 86], qui intègre l'utilisateur aux vues en couche et langage, va servir de point de départ à la présentation. Il nous suffira ensuite d'ajouter la dimension du parallélisme.

2.1. Le modèle de N. Sisson

Le schéma de la figure 8.1 est une représentation du modèle de N. Sisson adaptée à notre terminologie.

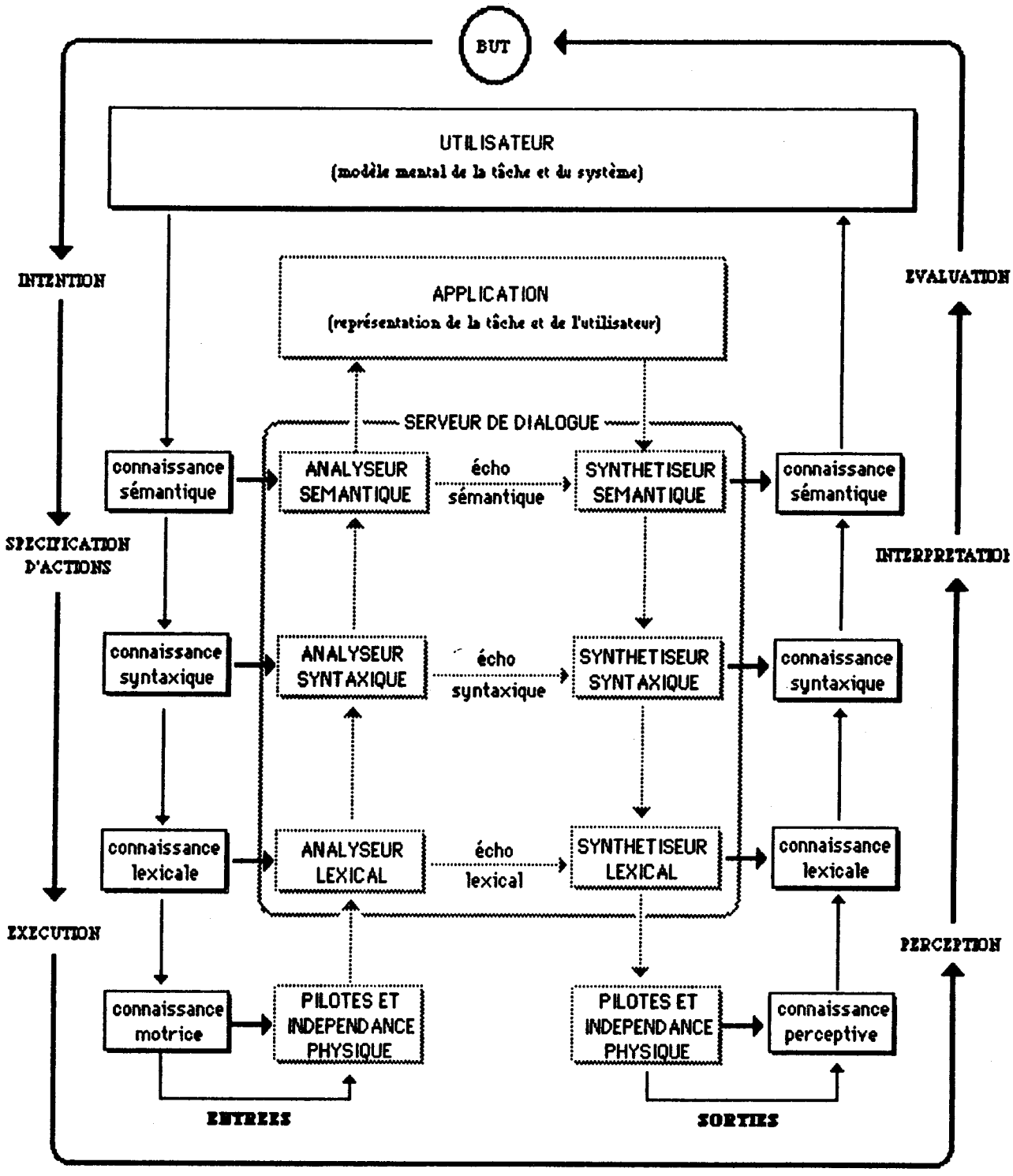


Figure 8.1 : Le modèle de N. Sisson. Les flèches décrivent le flux de l'information. Tous les éléments en pointillé (flèches et rectangles) se rapportent au système. Les éléments en trait plein concernent l'utilisateur.

L'élément frappant du modèle est la mise en évidence des liens entre les activités mentales de l'utilisateur et les traitements du système : le modèle mental de l'utilisateur maintient une représentation de la tâche à accomplir avec le système ; parallèlement, le système maintient une représentation de la tâche à réaliser avec l'utilisateur. La représentation mentale se subdivise en

connaissances sémantiques, syntaxiques, lexicales, motrices et perceptives. Chaque classe de connaissances a son utilité dans l'activité mentale et a un correspondant dans les traitements logiciels. Par exemple, les connaissances sémantiques, c'est-à-dire la modélisation des concepts du domaine et du système, interviennent dans l'activité mentale qui fait passer de la formation d'une intention à la spécification du plan d'actions. A ces connaissances sémantiques correspondent deux constituants logiciels : l'analyseur sémantique pour les expressions d'entrée, et le synthétiseur sémantique pour les expressions de sortie.

L'intérêt du modèle de Sisson est de présenter une vue synthétique qui intègre simultanément :

- la théorie de l'action de D. Norman décrite au chapitre 3 et dont nous retrouvons les étapes : formation d'une intention sollicitée par le but à atteindre, spécification d'un plan d'actions conduisant au but, exécution des actions du plan, perception des phénomènes physiques produits par le système, interprétation des phénomènes et évaluation de l'Image du système en comparant la situation avec le but recherché ;
- la notion de connaissances sémantiques et syntaxiques introduite au chapitre 3 ;
- le modèle langage dont on retrouve les constituants linguistiques, et
- le modèle entrée/sortie sous ses deux formes : le flux des échanges figuré par les flèches en pointillé, concrétise les transformations que subit l'information à chaque traversée d'un niveau d'abstraction. A cette vue en couches, se superpose une partie de la description modulaire présentée au chapitre précédent : les pilotes et les modules qui assurent l'indépendance logicielle vis-à-vis de l'appareillage physique, le serveur de dialogue et l'application.

Le modèle de Sisson est une représentation d'un monde idéal où le flux de l'information serait un flot séquentiel. Malheureusement, le raisonnement humain ne procède pas de manière séquentielle. PAC reprend la décomposition verticale et la correspondance horizontale du modèle de Sisson en y incorporant la composante multiagent qui permet de modéliser non seulement la non séquentialité mais aussi les activités parallèles.

2.2. Le modèle PAC

2.2.1. Les éléments de structuration

Comme le montre la figure 8.2, le modèle PAC [Coutaz 87a, Coutaz 87b] structure l'architecture d'un système interactif en trois constituants : la Présentation, l'Abstraction et le Contrôle.

- la Présentation définit l'Image du système, c'est-à-dire son comportement en entrée comme en sortie vis-à-vis de l'utilisateur ;
- l'Abstraction désigne les concepts et les fonctions du système ;
- le Contrôle maintient la cohérence entre les facettes Présentation et Abstraction.

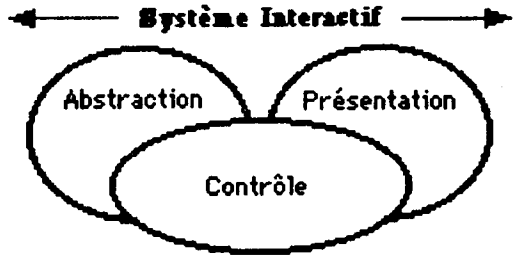


Figure 8.2 : Les constituants du modèle PAC : Présentation-Abstraction-Contrôle que l'on ne doit pas confondre avec le "Model View Controller" de Smalltalk présenté au paragraphe 4.1.

Si l'on s'en tient à cette définition, PAC paraît présenter peu de différence avec le modèle Seeheim. L'Abstraction désigne l'Application de Seeheim. Elle se charge de réaliser le niveau sémantique. Le Contrôle inclut les fonctions de l'Interface de Seeheim mais, nous le verrons par la suite, sa situation centrale lui vaut d'autres responsabilités. La Présentation de PAC rentre mal dans le cadre figé du Contrôleur syntaxique et de la Présentation lexicale de Seeheim.

La Présentation de PAC est réalisée par un ensemble d'agents spécialisés dans l'interaction avec l'utilisateur. Pour ces raisons, je les appelle objets interactifs. Tout objet interactif adhère au modèle PAC.

2.2.2. Notion d'objet Interactif

Un objet interactif PAC se caractérise par :

- une Image qui définit un comportement perceptible (c'est la Présentation visible de l'utilisateur),
- des fonctions ou des attributs fonctionnels (c'est le côté Abstrait visible des autres constituants logiciels), et
- la gestion des liens entre côtés abstrait et présentation (c'est le Contrôle).

L'exemple de l'histogramme de la figure 8.3 illustre la structure PAC d'un objet interactif :

1. la Présentation comprend
   - en sortie : une forme circulaire d'une taille donnée et deux couleurs servant à distinguer l'histogramme global de la tranche de camembert,
   - en entrée : les actions de l'utilisateur effectuées par l'intermédiaire des dispositifs de commande. Par exemple, on peut imaginer que l'utilisateur a la possibilité de modifier la part de camembert ou sa localisation par manipulation directe à l'aide de la souris ;
2. le côté Abstraction est défini par une **Valeur** entière comprise entre deux bornes **Min** et **Max**. L'objet histogramme ignore l'interprétation que d'autres constituants logiciels peuvent faire de ces trois éléments abstraits. De même, ces éléments ignorent la façon dont ils sont présentés et manipulés par l'utilisateur ;

3. le Contrôle consiste à maintenir la cohérence entre les éléments abstraits et la Présentation. Il est l'arbitre et le centre de décisions. Il décide en particulier des techniques de traduction entre les formes abstraites et concrètes d'un objet. Par exemple, lorsque l'utilisateur modifie la tranche de camembert, la Présentation de l'histogramme se charge du traitement des actions physiques et signale au Contrôle l'état actuel de la forme concrète. Le Contrôle interprète cet état et reflète ses traitements par la mise à jour de la valeur abstraite. Dans le sens inverse, si la valeur abstraite est modifiée par un constituant logiciel, le Contrôle avertit la Présentation d'ajuster la dimension de la tranche.

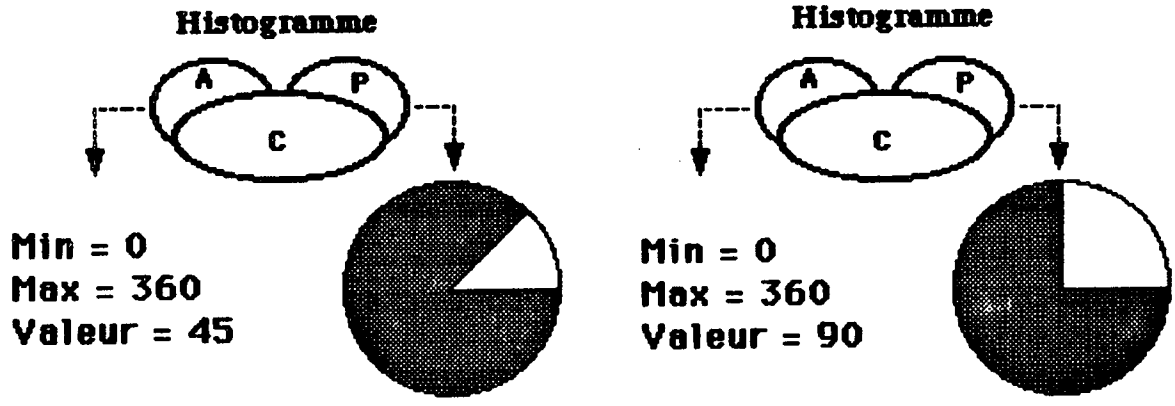


Figure 8.3 : Un exemplaire d'histogramme présenté sous forme de camembert : du côté Abstraction, l'histogramme est une valeur entière comprise entre 0 et 360. Du côté Présentation, c'est une surface circulaire comportant 2 parties de couleurs différentes et des règles de manipulation de la souris (non représentées dans la figure). Les dimensions relatives des deux parts de camembert reflètent l'état de la valeur abstraite Valeur. Dans l'exemple, Valeur passe de 45 à 90. La partie blanche est agrandie proportionnellement au changement de Valeur. Cette cohérence entre Abstraction et Présentation est assurée par le Contrôle.

Un objet interactif peut être élémentaire (comme l'histogramme de la figure 8.3) ou composé.

2.2.3. Objet interactif composé

Un objet composé définit un agent d'interaction structuré dont le comportement dépend de lui-même mais aussi des objets qui le constituent. Sa Présentation hérite des propriétés de présentation des constituants mais ajoute ses propres caractéristiques. Son Abstraction lui est spécifique. Son Contrôle vérifie la cohérence entre les côtés abstrait et présentation mais en sus gère la collaboration ou la dépendance entre les objets composants. L'exemple suivant illustre la composition d'objets interactifs.

Le superhistogramme de la figure 8.4 est constitué de deux objets élémentaires : l'histogramme de l'exemple précédent et une chaîne qui présente, sous forme numérique, la valeur abstraite de l'histogramme élémentaire. Parmi les attributs de présentation de la chaîne, relevons la police de caractères. La Présentation du superhistogramme applique des contraintes spatiales aux Présentations élémentaires : celles-ci sont alignées verticalement. A cela, s'ajoutent ses propres éléments de

décoration : ici, un cadre. La valeur du superhistogramme peut être modifiée de plusieurs façons : par un constituant logiciel ou par l'utilisateur.

- Lorsque la modification provient d'un constituant logiciel, son Contrôle traduit l'expression de la modification dans les termes de chacun des objets composants concernés par le changement. Chacun de ces objets reçoit une notification du changement. Les Contrôles des objets sollicités reflètent à leur tour la modification sur leur Abstraction et leur Présentation. Dans le cas du superhistogramme, les opérations de traduction sont inutiles (toutes les abstractions des objets composants sont des entiers qui recouvrent le même sens), mais le Contrôle du superhistogramme demande la modification des valeurs abstraites de l'histogramme élémentaire et de la chaîne. Les Contrôles élémentaires, avertis de la modification des abstractions, demandent aux Présentations élémentaires de s'actualiser.

- Lorsque la modification est une conséquence des actions de l'utilisateur, le cheminement des modifications s'effectue depuis l'objet élémentaire manipulé par l'utilisateur jusqu'à l'objet composé qui décide à son tour des effets de bord sur les autres constituants. Dans le cas du superhistogramme, l'utilisateur peut agir directement sur la présentation de l'histogramme élémentaire ou bien éditer la valeur présentée par la chaîne. Dans le premier cas, le Contrôle de l'histogramme élémentaire signale le changement de sa valeur abstraite au Contrôle de l'objet composé. Ce dernier interprète le signal reçu, convertit l'information dans les termes abstraits de l'objet chaîne et force cette chaîne à prendre une nouvelle valeur. La modification de l'abstraction de la chaîne est finalement reflétée par sa Présentation à la demande de son Contrôle. Dans le cas de l'édition, c'est le Contrôle de la chaîne qui avertit le supercontrôle de sa nouvelle valeur abstraite.

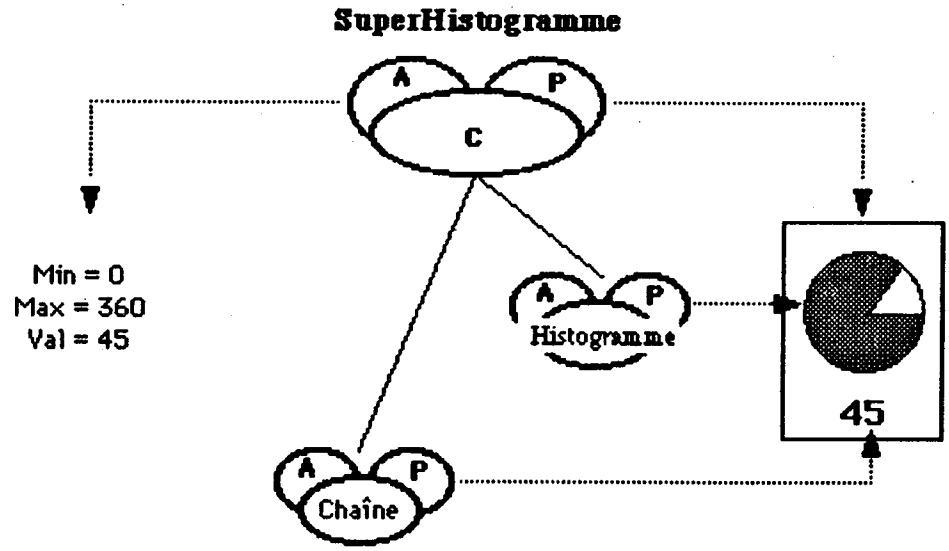


Figure 8.4 : Un Objet Interactif Composé de l'Histogramme de la figure 8.3 et d'une Chaîne Numérique. Sa Présentation hérite des Présentations des objets composants qu'elle aligne verticalement et qu'elle entoure d'un cadre. L'Abstraction de la chaîne est une valeur entière. Les attributs de sa Présentation incluent la police de caractères.

2.2.4. Vue complète d'un système interactif

Par application systématique des techniques de raffinement et/ou de composition, il est toujours possible de structurer un système interactif en une hiérarchie d'objets PAC. Un système interactif est donc un objet PAC composé. La figure 8.5 met en évidence cette organisation. Au sommet de la

hiérarchie, nous retrouvons les éléments introduits au paragraphe 2.2.1 : l'Abstraction qui constitue l'application, la Présentation qui organise l'Image du système, et le Contrôle qui joue le rôle d'intermédiaire entre l'application et la Présentation.

Du point de vue du Contrôle du niveau le plus haut, l'application est un producteur/consommateur de valeurs qui modélisent les concepts du domaine. Ces représentations abstraites conviennent aux traitements de l'application mais, présentées telles quelles à l'écran, ne facilitent pas forcément la tâche de l'opérateur humain. A la présentation brute de concepts se substituent des intermédiaires adaptés à l'interaction avec l'utilisateur : les objets interactifs. Une des fonctions du Contrôle du niveau le plus haut est de mettre en correspondance les concepts de l'application avec des abstractions d'objets interactifs. Ainsi, lorsque l'application modifie un concept, le Contrôle exprime le changement dans les termes des abstractions des objets qui lui sont associés. Les Contrôles de ces objets prennent le relais jusqu'à ce que l'effet se traduise dans les Présentations des objets élémentaires. Dans le sens inverse, les actions de l'utilisateur sont interprétées par les Présentations de plus bas niveau, reflétées dans les abstractions de plus bas niveau puis colportées en remontant la hiérarchie jusqu'à ce que des abstractions d'objets interactifs soient en liaison directe avec des concepts de l'application.

L'Image du système est définie par la Présentation du niveau le plus haut. Comme pour tout objet composé, cette Présentation hérite des Présentations des objets composants auxquelles s'ajoutent des propriétés intrinsèques. Nous verrons des exemples concrets au chapitre suivant.

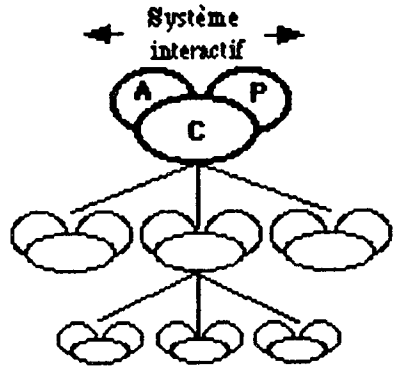


Figure 8.5 : Structuration d'un système interactif en objets PAC.

La technique de structuration PAC est également applicable au système interactif particulier qu'est le gestionnaire du poste de travail. Rappelons que ce gestionnaire s'occupe du métadialogue (voir chapitre précédent) : lancement et terminaison de systèmes, changement du centre d'intérêt, etc. La figure 8.6 précise l'organisation générale des activités menées par l'utilisateur sur son poste de travail.

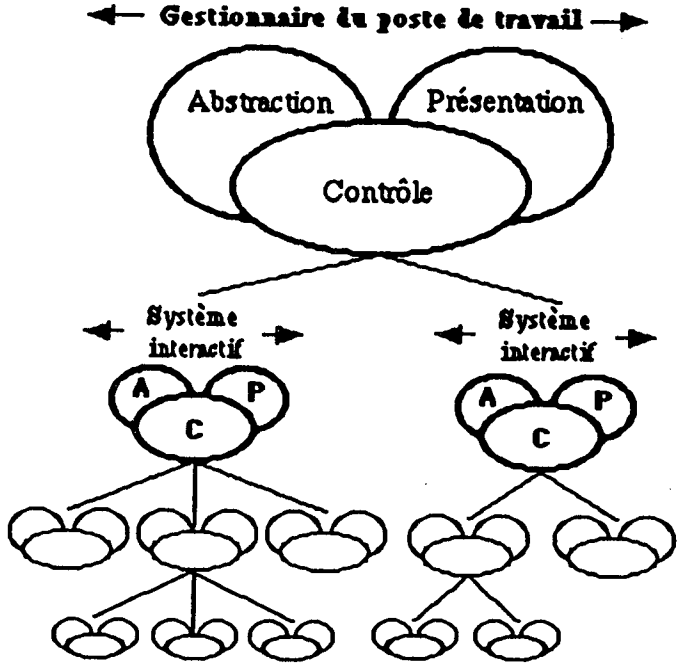


Figure 8.6 : Structuration PAC du gestionnaire du poste de travail.

Au sommet de la hiérarchie, l'Abstraction désigne les fonctions et les concepts propres à un gestionnaire de poste de travail : lancement d'un système interactif, copie de fichiers, services généraux tels le couper-coller et l'impression. La Présentation inclut celle des systèmes interactifs en cours d'utilisation et celle du métadialogue : par exemple un "shell" textuel à-la Unix ou un "desktop" graphique à-la Macintosh. Le Contrôle remplit le rôle habituel d'intermédiaire mais aussi rend possible la cohabitation de plusieurs univers : celui du gestionnaire et ceux des divers systèmes interactifs en présence. Nous verrons un autre exemple de cohabitation au chapitre suivant.

3. Intérêt du modèle PAC

L'intérêt du modèle PAC est triple : PAC s'appuie sur la notion d'agent ; il est applicable à tous les niveaux d'abstraction d'un système interactif ; il distingue fonctions et techniques de présentation mais il introduit la notion de Contrôle.

3.1. Les agents PAC

PAC est un modèle multiagent qui permet de créer à partir d'agents élémentaires (les objets interactifs élémentaires) des agents multiprocesseurs (les objets interactifs composés). En qualité de modèle multiagent, il offre les avantages de la modularité et de la prise en compte des dialogues à plusieurs fils d'activité.

3.1.1. Objet interactif élémentaire et la notion d'agent

Un objet interactif élémentaire est l'unité de compétence et d'exécution. Comme le montre la figure 8.7, il doit se voir comme une grappe indivisible de trois coprocesseurs, l'Abstraction A, le Contrôle C et la Présentation P, dotés chacun d'une mémoire : mémoire d'état abstrait, mémoire de contrôle et mémoire d'état graphique. Le répertoire d'instructions du Contrôle définit l'objet interactif du point de vue des autres objets. Ses instructions permettent de traiter deux classes d'événements : les requêtes en provenance d'autres objets interactifs et les actions de l'utilisateur. Leur interprétation fait appel aux instructions des processeurs Abstraction et Présentation. Le répertoire d'instructions de l'Abstraction contient les fonctions abstraites qui travaillent sur la mémoire d'état, celui de la Présentation, les primitives de restitution de l'objet à l'écran et l'interprétation des actions de l'utilisateur. Le Contrôle est donc une élaboration au dessus de l'Abstraction et de la Présentation.

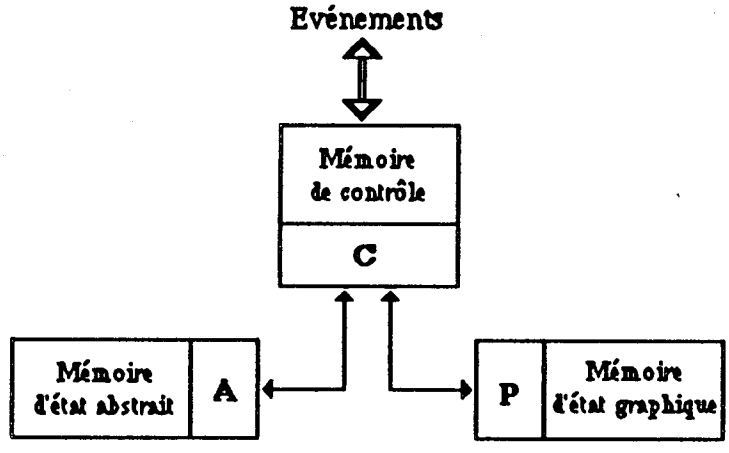


Figure 8.7 : Objet interactif élémentaire PAC vu comme un agent. Les flèches traduisent le flux des échanges.

Un objet interactif élémentaire est donc un agent structuré dont la compétence se répartit sur trois processeurs liés. Parmi ces processeurs, le Contrôle est le moteur central, les deux autres servant de périphériques spécialisés. Le rôle du Contrôle sera précisé au paragraphe 3.3.

3.1.2. Objet interactif composé et la notion d'agent

Un objet interactif composé est une organisation multiagent assemblée selon le schéma de la figure 8.8. L'objet composé comprend, comme l'objet élémentaire, une grappe de trois coprocesseurs principaux (notés A, C et P dans le schéma) dotés chacun d'une mémoire privée. Il comprend en plus les processeurs des objets composants. Comme pour l'objet élémentaire, le répertoire des instructions de C définit l'agent composé vis-à-vis des autres objets interactifs. L'interprétation des instructions de C fait appel aux répertoires de A et de P mais aussi à ceux de C1 et C2, les contrôles des objets composants.

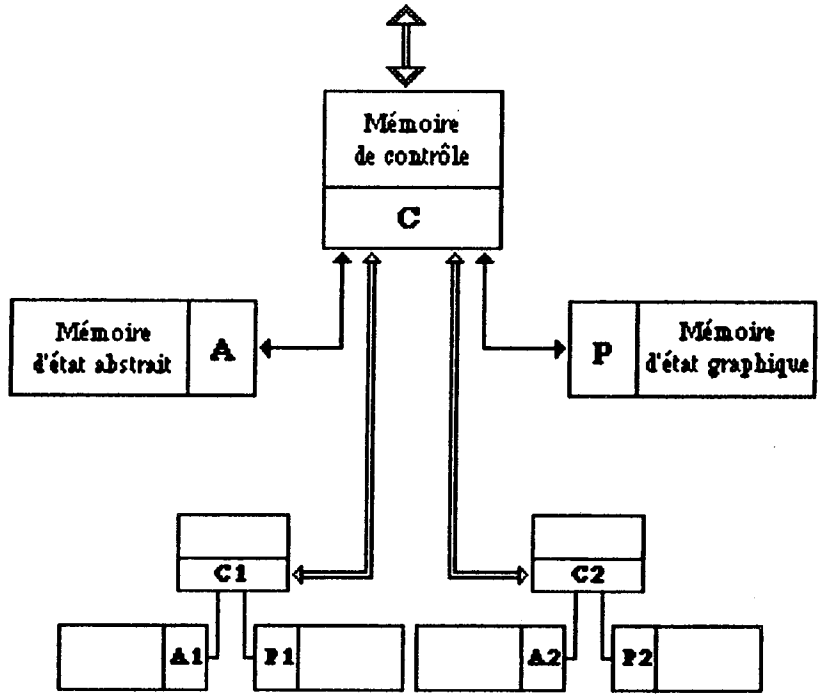


Figure 8.8 : Objet interactif PAC vu comme une organisation multiagent. Ici, un objet composé de deux objets élémentaires.

L'objet composé est donc une élaboration au dessus du fonctionnement d'agents composants. En ce sens, il définit un nouveau niveau d'abstraction. Les objets composants conservent néanmoins leur identité. En ce sens, ils peuvent être référencés par d'autres Contrôles. La construction hiérarchique pure a des avantages sur lesquels nous reviendrons mais, par définition, elle ne traduit pas toutes les formes de relations. Il arrive fréquemment que des objets PAC aient des relations hiérarchiques pour un sous-ensemble seulement de leurs compétences. Par exemple, deux objets élémentaires O1 et O2 ont besoin d'un arbitre pour gouverner leurs relations spatiales alors que leurs abstractions sont indépendantes. Dans ce cas, il est naturel d'introduire un objet O composé de O1 et O2 qui contrôle les relations spatiales mais il est maladroit d'imposer aux objets qui influencent les valeurs des abstractions de O1 et de O2 de transiter par O. Nous verrons des exemples d'utilisation de cette possibilité au chapitre suivant.

Nous observons qu'un objet interactif élémentaire est un agent structuré en trois formes de compétence dont deux, l'Abstraction et la Présentation ne communiquent jamais directement. Cette indépendance ou plus exactement, le mécanisme de référence indirecte assuré par le Contrôle, permet de modifier ou de remplacer la réalisation d'une classe de compétences sans mettre en cause le fonctionnement de l'autre.

Nous observons aussi que les constructions PAC conservent l'indépendance des traitements et des modélisations entre les fonctions abstraites, les services de contrôle et les techniques de présentation. Les modifications logicielles pratiquées au niveau élémentaire sont donc applicables à tous les niveaux d'abstraction. En particulier, il est aisé de remplacer la présentation d'un concept par d'autres objets interactifs ou d'ajuster localement l'Image d'un objet élémentaire.

3.1.3. Objets PAC et dialogues à plusieurs fils d'activité

Les dialogues à plusieurs fils d'activité se caractérisent par une interaction dirigée par l'utilisateur, des actions d'entrée et de sortie simultanées. En d'autres termes : l'utilisateur a le droit de changer de centre d'intérêt au gré de son plan d'actions, il peut agir simultanément sur la présentation de plusieurs concepts, et plusieurs présentations de concept peuvent s'exprimer simultanément. Tous ces phénomènes se traduisent aisément à l'aide d'objets PAC.

- *Manipulation interruptible.* Chaque objet PAC élémentaire est un lieu d'activités dont l'état est actualisé en permanence par les trois coprocesseurs dans les trois mémoires locales. Un objet composé est un lieu d'activités dont l'état est réparti à divers niveaux d'abstraction dans les coprocesseurs principaux et dans les objets composants. L'ensemble de ces états modélise l'état de l'interaction. Grâce à cette représentation répartie, à tout instant, tout objet, élémentaire ou composé, est en mesure d'être "abandonné" par l'utilisateur puis d'en être à nouveau le centre d'intérêt.

- *Parallélisme des entrées.* La présentation d'un concept s'effectue à l'aide d'un objet PAC. Celui-ci est à tout instant en mesure de traiter les événements parmi lesquels se comptent les actions de l'utilisateur. Si l'utilisateur agit sur deux présentations simultanément, il constatera des réactions de la part des objets sollicités. La simultanéité des réactions risquent cependant d'être limitée par la façon dont le serveur du poste de travail modélise les événements physiques. Cet aspect a été évoqué au chapitre 6 où nous avons relevé la nécessité de la datation des événements pour encapsulater en une seule unité événementielle les interruptions physiques (nécessairement séquentielles) produites par les actions que l'utilisateur perçoit comme simultanées.

- *Parallélisme des sorties.* Un objet PAC est à l'écoute de l'utilisateur tout comme il est à l'écoute des requêtes d'autres objets ou de lui-même. Toute requête modifiant une abstraction a un effet sur la présentation. Sans aucune sollicitation extérieure, son état peut le conduire à s'exprimer en sortie pour produire des effets d'animation. Nous verrons des exemples pratiques de ces possibilités au chapitre suivant.

3.2. La récursivité PAC

Le fait que PAC soit applicable à tous les niveaux d'abstraction d'un système interactif présente quelques retombées intéressantes : il fournit un cadre systématique pour la conception d'architecture et rend possible la répartition des notions de sémantique et de syntaxe.

3.2.1. Cadre de construction systématique

PAC se situe dans la ligne directe des techniques de structuration modulaire mais n'impose aucun choix entre les méthodes d'analyse ascendante ou descendante. A cet égard, le plan de résolution du problème n'est pas pris en charge par PAC et respecte l'heuristique habituelle du concepteur.

Cependant PAC demande au concepteur de penser en termes d'agents c'est-à-dire en termes d'objets au sens usuel de la programmation par objets. Dans la programmation par objets, on ne met pas les données dans les procédures mais on attache des procédures à des structures de données (qui modélisent l'état de l'objet). En outre, tout objet PAC doit se voir comme une cohabitation de deux formalismes par l'intermédiaire d'un traducteur : l'un abstrait dirigé par les besoins internes du logiciel, l'autre concret adapté à l'utilisateur. Il est important de noter que ces deux formalismes, celui de l'Abstraction et celui de la Présentation, "ne se comprennent pas" et que la communication entre les deux langages passe par le Contrôle. Cette rigueur systématique doit utilement servir le concepteur à définir l'organisation du logiciel interactif.

3.2.2. Répartition de la sémantique et de la syntaxe

Une forme syntaxique, sa sémantique et le rapport entre les deux n'existent chez un individu que par leur représentation mentale. Par analogie, l'Abstraction d'un objet interactif élémentaire peut se voir comme la représentation d'une sémantique pour laquelle la Présentation élabore une forme syntaxique. Le rapport entre les deux est assuré par le Contrôle. Un système interactif, qui est un objet interactif composé, voit son comportement sémantique et syntaxique distribué parmi ses constituants. Cette répartition trouve son utilité sous plusieurs formes. Nous retiendrons ici : la possibilité d'échanger des informations au niveau d'abstraction voulu et l'élimination du problème de la séparation entre application et interface interactive.

- *Echanges au niveau d'abstraction voulu.* L'Abstraction au sommet de la hiérarchie PAC d'un système interactif définit la sémantique du domaine d'application. Elle décrit un savoir-faire abstrait indépendant de toute considération syntaxique. L'Abstraction doit donc communiquer avec le Contrôle du niveau le plus haut dans les termes qui lui sont propres. Par exemple, si elle manipule des entiers, elle doit pouvoir communiquer et recevoir des valeurs entières. Un exemple moins trivial est celui des erreurs car généralement la formulation textuelle des messages d'erreur se trouve câblée dans l'application. Lorsque l'Abstraction de plus haut niveau détecte des anomalies sémantiques, celles-ci doivent être exprimées de manière

symbolique. En aucun cas, elles ne doivent avoir à ce niveau une forme syntaxique finale. Cette dernière sera décidée par des objets interactifs compétents qui, connaissant par exemple la classe de l'utilisateur auxquels ils s'adressent, trouveront une formulation adaptée. Nous verrons au chapitre suivant la mise en pratique de ces exemples.

- *Elimination de la frontière entre application et interface.* Dans certains domaines, la limite fonctionnelle entre l'application et l'interface est difficile à déterminer. PAC apporte un élément de réponse à ce problème en acceptant de répartir le savoir-faire. Cette répartition, qui s'effectue de manière systématique et structurée, ne conduit pas à la désorganisation qui empêcherait l'ajustement itératif du système. L'élimination du problème de frontière permet d'effectuer proprement des réparations logicielles, de limiter la fréquence des échanges en déléguant à d'autres objets PAC une part de compétence, de répartir la sématique comme dans les FA formulaires de C. Collet [Collet 87] ou de faire cohabiter des domaines d'application distincts.

  - Réparation : l'élimination des frontières rend possible l'extension des fonctions de l'Abstraction sans en modifier la réalisation. Il arrive fréquemment qu'à l'utilisation d'un système, une fonction du niveau du domaine fasse défaut : par exemple, l'indication du nombre total de messages électroniques que l'utilisateur n'a pas encore lus. Ce comptage, qui n'a pas été prévu dans l'Abstraction, peut être effectué par un objet PAC de la Présentation : chaque fois que le Contrôle de plus haut niveau reçoit de son Abstraction un message de la forme "voilà un courrier, provenant de telle personne, traitant de tel sujet, etc.", celui-ci sollicite l'objet compteur qui met à jour son Abstraction et sa Présentation. Nous verrons un exemple de réparation sémantique au chapitre suivant.

  - Réduction des échanges : l'élimination des frontières rend possible la délégation d'une part de savoir-faire de l'application vers des objets de présentation. Il arrive fréquemment dans les objets composés de type formulaires ou tableaux de bord que la saisie d'une valeur nécessite une vérification sémantique immédiate relevant de l'Abstraction de plus haut niveau. Afin d'éviter un aller-retour coûteux, il est parfois possible de télécharger cette vérification dans l'Abstraction du tableau de bord. Nous verrons une application de cette possibilité au chapitre suivant.

  - Cohabitation de domaines : la répartition permet aux techniques de présentation d'exprimer non seulement la sémantique du domaine d'application mais aussi celle d'un domaine totalement indépendant. Cette situation se rencontre fréquemment dans les interfaces imitant le monde réel. Dans ces interfaces, les concepts du domaine qui rappellent les propriétés d'objets du monde réel, sont présentés par des imitations électroniques. Lorsqu'il est utile de pousser le message métaphorique jusqu'au bout, les reproductions électroniques doivent se comporter comme les objets réels. Or, il arrive que ce comportement réponde à des lois spécifiques indépendantes de celles des concepts à présenter. Par exemple, le réchaud, source de chaleur, et le thermomètre, indicateur de température, illustrent, au nom des principes de la métaphore du monde réel, les relations entre les concepts de chaleur et de température. Tous deux obéissent aussi à la loi de la gravitation totalement étrangère aux règles de la thermodynamique. Grâce à la répartition encouragée dans PAC, il est possible de faire cohabiter des mondes abstraits indépendants. En l'occurrence, les relations gravitationnelles qui régissent le comportement spatial du thermomètre et du réchaud peuvent s'exprimer dans un objet PAC extérieur à l'Abstraction spécialisée en thermodynamique. Nous reviendrons sur cet exemple au chapitre suivant.

Jusqu'ici nous avons vu qu'un objet interactif était un agent et que le modèle s'appliquait récursivement aux niveaux d'abstraction voulue. Ces deux caractéristiques s'articulent autour de la notion de Contrôle.

3.3. La notion de Contrôle PAC

Nous verrons au paragraphe 4 que d'autres modèles d'architecture s'inspirent du concept d'agent (ou d'objet) sans toutefois introduire la notion de Contrôle. Tous s'accordent sur le principe de séparation entre fonctions du domaine et techniques de présentation mais tous se taisent sur la façon d'y parvenir. La notion de Contrôle telle que l'entend le modèle PAC sert avant tout de pont entre deux mondes dont les rôles et les formalismes sont distincts. Nous allons identifier les fonctions de ce pont répertoriées dans la figure 8.9 : traduction et arbitrage.

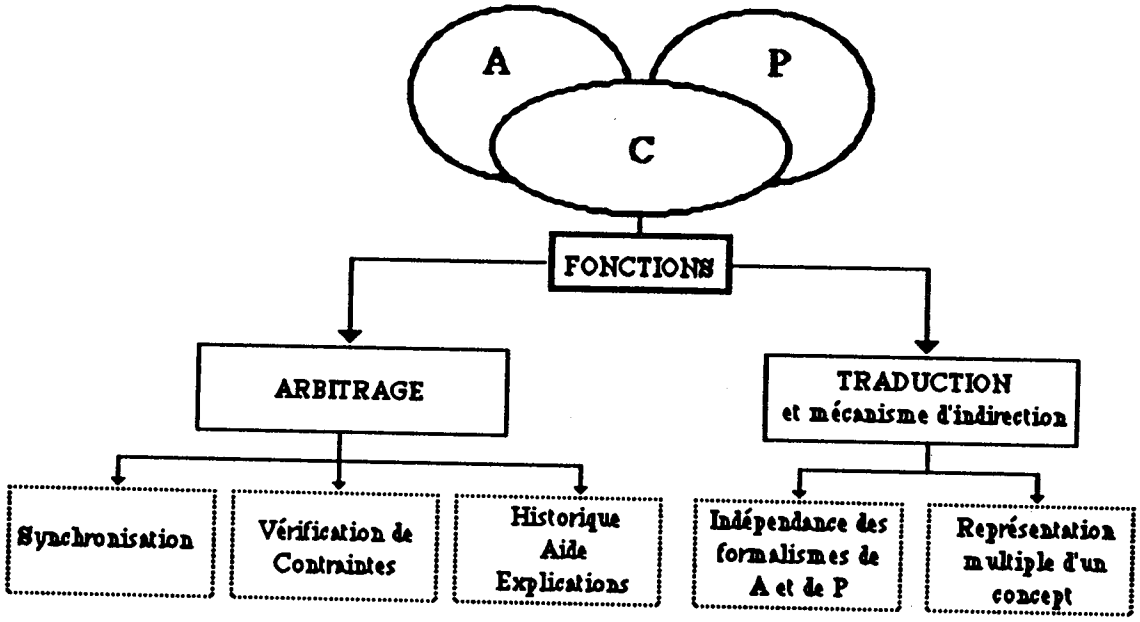


Figure 8.9 : Les fonctions de la notion de Contrôle.

3.3.1. Traduction et mécanisme d'indirection

La fonction de traducteur du Contrôle a été évoquée à plusieurs reprises : le modèle PAC impose aux machines abstraites Abstraction et Présentation des répertoires d'instruction distincts et ne leur accorde aucun moyen de communiquer directement, ceci afin de garantir la séparation modulaire des rôles : l'un dirigé par le fonctionnement logiciel (la sémantique), l'autre orienté vers l'utilisateur (la syntaxe). L'une et l'autre de ces machines se référencent de manière indirecte, le lien d'indirection étant géré au niveau du Contrôle. L'intérêt de l'indirection est la possibilité de substituer l'une des machines sans remettre en cause la réalisation de la partenaire. Il suffit pour cela de modifier la liaison dans le Contrôle. Le Contrôle sert donc d'intermédiaire entre le formalisme de représentation de la sémantique et celui utilisé pour la syntaxe.

Il est courant de constater qu'à une sémantique donnée correspondent plusieurs formes syntaxiques. Le Contrôle sachant maintenir le rapport entre les deux peut aussi servir à associer

plusieurs présentations à une même abstraction. Toute modification d'abstraction étant signalée au Contrôle, celui-ci sait en traduire l'effet sur l'ensemble des techniques de présentation liées à cette abstraction. L'utilité des vues multiples d'un concept a été évoquée au chapitre 4 à propos de la flexibilité des interfaces : adéquation à l'étape d'évaluation actuelle de la tâche, adaptation à la classe d'utilisateurs. La notion de Contrôle fournit un moyen simple et naturel de mettre en œuvre ce principe ergonomique. Nous verrons une application de cette possibilité au chapitre suivant.

3.3.2. Arbitrage

Dans un objet PAC, le Contrôle est le processeur principal par lequel tout transite. Il peut donc être utilement exploité à enrichir les deux mondes qu'il sert (l'Abstraction et la Présentation), assurer notamment toute forme d'arbitrage, en particulier la synchronisation, la vérification de contraintes et la gestion d'informations contextuelles.

- *Synchronisation.* Dans un environnement à activités multiples, il est nécessaire de définir des points de synchronisation. Généralement, la solution choisie s'appuie sur l'existence d'un arbitre qui centralise les requêtes et qui décide de l'ordre d'activation des agents demandeurs. En raison de sa position centrale, le Contrôle est le candidat naturel à cette fonction : il peut décider de l'ordre de traitement des signaux en provenance de ses coprocesseurs et, s'il est le Contrôle d'un objet composé, il peut aussi gérer la coopération entre les objets composants. Dans un système interactif PAC, la synchronisation est donc répartie sur l'ensemble des Contrôles, chaque Contrôle assurant une gestion adaptée aux besoins locaux. Ces besoins peuvent s'exprimer sous forme de contraintes.

- *Vérification de contraintes.* Le comportement d'une entité dépend essentiellement des contraintes imposées par son environnement. La remarque de Herbert Simon dans [Simon 84] à propos du comportement de la fourmi confirme cette hypothèse. Selon Simon, la complexité du comportement de la fourmi sur la plage de sable n'est pas due à la complexité de ses processus internes mais bien à celle de l'environnement. Avec cette hypothèse, il convient que le comportement d'un objet interactif soit contrôlé en ultime ressort par l'environnement. L'environnement immédiat d'un objet interactif est le Contrôle auquel il est subordonné. L'environnement effectif est la suite des Contrôles, depuis le Contrôle de l'environnement immédiat jusqu'au sommet de la hiérarchie. L'avantage de la décision ultime laissée à la hiérarchie des Contrôles est la réutilisation d'objets interactifs d'environnement en environnement : un objet ne doit jamais faire référence explicite à d'autres objets mais doit toujours notifier le Contrôle dont il dépend de ses changements d'état. Ce Contrôle évalue la nouvelle situation et détermine les effets de bord sur l'ensemble des objets qu'il contrôle. Nous verrons une illustration de ces principes au chapitre suivant.

- *Gestion d'informations contextuelles.* J'entends par contexte un état qui définit les conditions d'un traitement. Nous avons déjà défini l'état d'un objet élémentaire comme l'assemblage des trois états mémorisés dans chacune des mémoires locales. Parmi les informations d'utilité générale et qui nous viennent des ergonomes sont celles qui servent à la mise en place d'un mécanisme d'historique. Par exemple, la mémoire d'état abstrait peut mémoriser les valeurs successives prises par les abstractions ; la mémoire d'état graphique peut mémoriser les positions successives de l'objet ; la mémoire de contrôle peut modéliser les liens entre les deux historiques et participer à la fonction d'historique répartie sur l'ensemble des Contrôles. Les fonctions d'aide et d'explication devraient pouvoir être mises en œuvre de façon similaire.

Après la description des principes du modèle PAC, nous sommes en mesure d'en présenter les différences avec des modèles approchants.

4. Autres modèles multiagent

D'autres modèles s'appuient, comme PAC, sur la notion d'agent (ou d'objet) et font la distinction entre fonctions et présentation. Ce sont les modèles : MVC (Model, View, Controller) de Smalltalk [Goldberg 84], InterViews de Mark Linton [Linton 86], PPS (Primitive Presentation System) d'Eugene Ciccarelli [Cicarelli 84] et GWUIMS de l'Université de Georges Washington [Sibert 86].

4.1. MVC

Dans MVC, le Modèle, la Vue et le Contrôleur sont trois objets Smalltalk. Le Modèle réunit l'ensemble des fonctions abstraites, la Vue définit la présentation de sortie, et le Contrôleur interprète les entrées en provenance de l'utilisateur.

PAC et MVC diffèrent en plusieurs points :

- PAC se veut libre des contraintes pratiques de l'outil de réalisation. MVC est au contraire très lié aux techniques de la programmation par objets lorsqu'il impose de concrétiser la distinction entre services fonctionnels et services de présentation sous forme de trois objets Smalltalk. PAC laisse le choix de la séparation modulaire à l'appréciation du programmeur en fonction des propriétés de l'environnement, du niveau d'abstraction choisi ou des contraintes de performance (le coût de la communication par message peut être incompatible avec une demande de réaction immédiate).

- MVC utilise deux objets distincts, la Vue et le Contrôleur, pour réaliser la notion équivalente de Présentation dans PAC. La distinction entre les comportements d'entrée et de sortie a l'avantage de la souplesse : elle permet de modifier l'un indépendamment de l'autre. Cependant, dans les couches basses de l'interaction, on observe fréquemment l'asservissement mutuel des expressions de sortie et des événements d'entrée. Par exemple, le suivi de la souris par un contour ou l'affichage en vidéo inverse de l'objet visité par la souris nécessitent tous deux une réaction immédiate de la part de l'entité concernée. La distinction telle que la propose MVC entraîne nécessairement des échanges par messages qui risquent d'allonger le temps de réaction du système. PAC, avec son composant Présentation, n'impose pas la séparation modulaire au bénéfice de la performance mais au risque d'une implémentation où s'enchevêtrent entrées et sorties.

- PAC explicite la notion de Contrôle, organe arbitre et traducteur entre le monde abstrait et le monde concret. Dans MVC, le contrôle est inexistant. Ses fonctions sont diluées entre le Modèle, la Vue et le Contrôleur. Sans lieu explicite de synchronisation ou d'arbitrage, le concepteur du système n'a pas de cadre de référence pour l'expression de la coopération entre les agents de l'interaction.

4.2. Interviews

Interviews insiste sur la distinction entre abstraction et interface mais regroupe entrées et sortie au sein d'un même objet. Interviews introduit donc deux classes d'objet : les "Subjects" (ou Sujets) et les "Views" (ou Vues). Un Sujet est une abstraction équivalente au Modèle de MVC et à l'Abstractio de PAC. Une Vue réunit les fonctions de la Vue et du Contrôleur MVC et correspond à l Présentation PAC. Interviews est donc très voisin de PAC mais n'introduit pas la notion de Contrôle

4.3. PPS

PPS est organisé autour des notions de Présentateur (Presenter) et de Reconnaisseu (Recognizer) agissant sur deux bases de données : la base Système et la base de Présentation. La bas Système définit l'interface abstrait tandis que la base de Présentation présente à l'écran des élémen de la base Système. Le Présentateur crée la base de Présentation à partir de la base Systèm L'utilisateur édite la Présentation et le Reconnaisseur modifie la base Système en fonction des actior de l'utilisateur.

Présentateur et Reconnaisseur ne sont pas des objets mais des processeurs de traduction qui for le pont entre l'abstrait et le concret : le premier effectue la traduction de l'abstrait vers le concret, l second du concret vers l'abstrait. Les bases Système et Présentation sont des sortes de réseau sémantiques regroupant des définitions de classes et des exemplaires d'objets. Dans la réalisatio actuelle, les bases sont confondues en un seul réseau sémantique qui exprime les liens entre objets d présentation et objets abstraits. Le modèle PPS est applicable itérativement et récursivement : une bas de Présentation peut jouer le rôle de base Système pour un autre couple Présentateur-Reconnaisseu qui engendre et gère une nouvelle base de Présentation ; l'état d'un Présentateur P (ou d'u Reconnaisseur R) constitue une base abstraite à laquelle on peut associer un Présentateur, u Reconnaisseur et une base de Présentation qui permettent de contrôler P (ou R).

Tout comme MVC, PPS introduit une dichotomie entre comportement d'entrée et comportemer de sortie mais dans PPS, ces comportements ne sont pas véhiculés par des objets. Les objets PP sont des entités passives manipulées par les Présentateurs et les Reconnaisseurs tandis que dans PAC MVC et Interviews, les objets sont des agents qui décident de leur comportement. PPS choisit le poi de vue base de connaissances. Les objets représentent la connaissance factuelle et les processeu décrivent la connaissance procédurale nécessaire à la manipulation des faits.

4.4. GWUIMS

GWUIMS (George Washington User Interface Management System) structure un système interactif en cinq classes d'objets : les Objets-A (A pour abstraction) contiennent la sémantique de l'application ; les Objets-I (I pour interface) font le pont entre les Objets-A et les Objets-R. Les Objets-R (R pour représentation) contrôlent la présentation. Ils font le pont entre les aspects syntaxiques et lexicaux. Les aspects lexicaux d'entrée sont traités par les Objets-T (T pour "typing") tandis que les Objets-G (G pour graphique) s'occupent des aspects lexicaux de sortie.

A la différence de PAC, GWUIMS n'est pas récursif mais fige la profondeur des relations. Comme le modèle langage, il se limite aux trois niveaux de base : sémantique (avec les objets-A), syntaxique (avec les objets R) et lexical (avec les objets-T et G). Entre chaque niveau, GWUIMS insère un pont : les objets-I entre la sémantique et la syntaxe, et les objets-R entre la syntaxe et le lexique. Nous avons vu, avec la notion de Contrôle l'intérêt de ces ponts, lieux d'arbitrage entre les parties. Alors que PAC laisse toute liberté sur la fonction exacte des Contrôles et sur la hiérarchie à envisager, GWUIMS impose un rôle précis aux intermédiaires et restreint explicitement l'architecture à trois niveaux d'abstraction. Les réalisations décrites au chapitre suivant mettront en évidence la nécessité d'une organisation plus structurée sans restriction sur la profondeur.

EN RESUME :

Le modèle PAC structure de manière récursive l'architecture d'un système interactif en agents (o objets interactifs) organisés chacun en trois classes de compétence :

- la Présentation définit l'Image de l'agent, c.-à-d. son comportement perceptible de l'utilisateur
- l'Abstraction définit les fonctions et les attributs internes de l'agent, c.-à-d. son comportemer vis-à-vis d'autres agents,
- le Contrôle maintient la cohérence entre les deux perspectives. Il est :
  - . un arbitre : son rôle central fait de lui le candidat naturel à la résolution des conflits entr constituants. Il peut servir de lieu de synchronisation, d'allocateur de l'unité centrale et d l'écran pour le réglage des animations et des rafraîchissements, de gestionnaire de l'éta local pour les dialogues à plusieurs fils d'activité,
  - . un traducteur : une Présentation et son Abstraction ne communiquent jamais directemer mais s'adressent toujours à leur Contrôle dans le formalisme qui leur convient. L mécanisme d'indirection aménagé dans un Contrôle est également utile à la présentatio multiple de concept.

Un objet interactif peut être élémentaire ou composé :

- un objet interactif élémentaire est une unité de compétence et d'exécution répartie en troi classes de fonctions : Présentation, Abstraction, Contrôle ;
- un objet composé définit une nouvelle unité de compétence organisée selon les mêmes critère qu'un objet élémentaire mais qui, en sus, hérite du comportement des objets constituants et le soumet à ses propres règles.

Le modèle PAC présente les avantages suivants :

- il définit un cadre de construction systématique applicable à tous les niveaux d'abstraction d'u système interactif,
- il distingue les services abstraits des techniques d'interaction, mais il introduit la notion d Contrôle qui permet :
  - . à deux perspectives de communiquer par indirection avec l'avantage de modifier l'un sans mettre en cause le fonctionnement de l'autre,
  - . à l'état de l'interaction d'être géré par un ensemble de Contrôles locaux coopérants ave l'avantage de rendre possibles les dialogues à plusieurs fils d'activité,
  - . aux informations de nature contextuelle d'être maintenues à divers niveaux d'abstractio avec l'avantage d'être exploitables par des services d'aide et d'historique.
- il encourage la répartition des traitements sémantiques et syntaxiques. Cette propriété présente son tour plusieurs retombées intéressantes :

. les constituants de l'interface communiquent au niveau d'abstraction voulu. En particulie l'application peut (et doit) s'exprimer dans les termes qui lui sont propre indépendamment de toute considération de présentation,

. l'élimination de la frontière entre application et interface rend possibles les réparation sémantiques (sans mettre en péril l'esthétique architecturale) ; permet la cohabitation d domaines sémantiquement distincts (sans mettre en cause leur indépendanc fonctionnelle), et encourage la délégation de fonctions du domaine dans l'interface (dans l but d'améliorer les performances du système).

Chapitre 9

Applications du Modèle PAC

5. Le système expert IRENE

5.1. L'objet du système

5.2. Les fonctions du système et son Image
- 5.2.1. Reproduction des procédures du monde réel
- 5.2.2. Aide à la conception : vérification
- 5.2.3. Aide à la conception : génération

5.3. L'architecture PAC du système

5.4. Analyse syntaxique à divers niveaux d'abstraction
- 5.4.1. Premier niveau d'analyse : les fenêtres
- 5.4.2. Deuxième niveau d'analyse : l'objet multifenêtre

5.5. Création dynamique d'entités

5.6. Génération des messages d'erreur sémantique

5.7. Délégation de pouvoirs sémantiques dans l'interface
- 5.7.1. Réparations sémantiques
- 5.7.2. Amélioration des performances

EN RESUME

1. Introduction

Ce chapitre illustre l'utilisation du modèle PAC avec la description concrète de quatre systèmes interactifs. L'intérêt de cette présentation est double : montrer le transfert de règles de l'ergonomie cognitive dans la conception logicielle, et montrer l'utilisation du modèle PAC à la résolution des problèmes les plus courants en architecture d'interfaces interactives. Nous verrons que la diversité des systèmes ici décrits, tant par la nature des fonctions que par les outils de réalisation, s'estompe dans le cadre général et unificateur du modèle PAC.

Le premier système est un simulateur du fonctionnement d'une centrale nucléaire. La conception de l'architecture logicielle de ce système a été réalisée par des étudiants de l'I.N.S.A. de Lyon dans le cadre de leur projet de fin d'étude. Le second système, Thermo, est une application jouet démontrant quelques lois simples de la thermodynamique. Il a été réalisé en langage C dans un environnement monotâche au dessus de la boîte à outils du Macintosh. Le troisième système, ELOISE (Environnement Langage Objet d'Implémentation de Systèmes Experts), est un environnement de développement de systèmes experts [Doaré 87]. Il a été réalisé par des étudiants du DESS de Génie Informatique de l'Université de Grenoble et programmé en Loops [Bobrow 83] au dessus d'Interlisp sur une machine Xerox 1186. Le dernier système, IRENE (Inference Rules for EtherNet Expertise), est un système expert d'aide à la configuration de réseaux XNS (Xerox Network System) [Alletru 88]. Il a été réalisé par des étudiants du DESS de Génie Informatique de l'Université de Grenoble en utilisant les avantages respectifs des langages ELOISE et Loops.

Les paragraphes qui suivent décrivent successivement ces quatre systèmes expérimentaux.

2. Le simulateur de centrale nucléaire

Mon intervention dans la conception de ce système s'est limitée à la définition de l'architecture logicielle. L'étude des besoins de l'opérateur n'est donc pas l'élément initial de ma participation. Aussi, les aspects cognitifs seront peu abordés dans la présentation qui suit.

2.1. L'objet du système et son Image

Le problème considéré ici est la représentation de l'état d'un liquide circulant dans un tuyau de centrale nucléaire. Dans les installations de ce type, un tuyau est constitué de sections. Dans certains cas, l'opérateur se contente d'une évaluation approximative de l'état par simple consultation d'un

indicateur de pression. En d'autres circonstances, il a besoin d'observer en détail l'état du liquide dans les sections de son choix. La figure 9.1 illustre les deux situations.

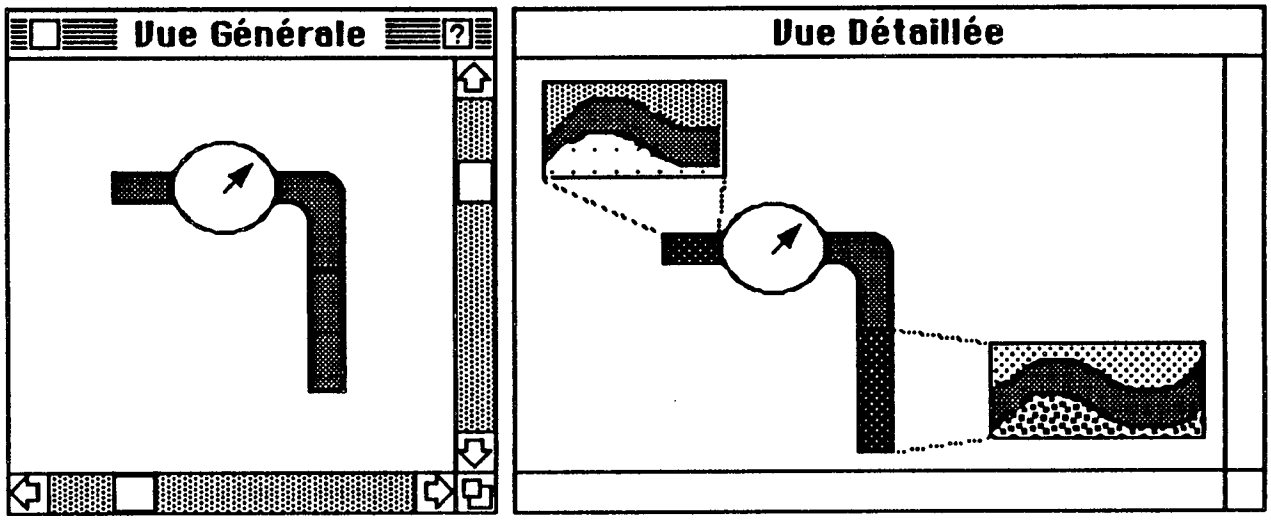


Figure 9.1 : Présentation des concepts d'un simulateur de centrale nucléaire. A gauche, vue générale, à droite vue détaillée.

Nous supposons que cette présentation de la notion d'état est adaptée à la tâche de surveillance de l'opérateur. Nous envisageons maintenant son organisation logicielle en termes de composants PAC.

2.2. L'architecture PAC du système

La figure 9.2 montre l'architecture générale du système. Le comportement détaillé des constituants est ensuite présenté.

2.2.1. Architecture générale

Le simulateur est l'Abstraction qui figure au sommet de la hiérarchie PAC. Il constitue l'application. Du point de vue du Contrôle du niveau le plus haut, le simulateur est un producteur de valeurs numériques qui modélisent l'état général d'un tuyau et l'état de chacune des sections. Ces états sont des représentations abstraites qui conviennent aux traitements du simulateur mais qui doivent être présentées à l'opérateur humain sous une forme adaptée à la tâche de surveillance.

Dans notre exemple, la représentation abstraite de l'état général d'un tuyau est concrétisée par l'objet composé **Tuyau** et l'état d'une section par un objet élémentaire **Section**. Le Contrôle du sommet de la hiérarchie se charge de gérer la correspondance entre les états abstraits produits par le simulateur et les abstractions des Sections et de Tuyau. En particulier, toute modification émise par le

simulateur est reflétée aux abstractions correspondantes des objets PAC. Dans la figure 9.2, on assiste à une correspondance biunivoque entre les structures du simulateur et les abstractions de objets PAC.

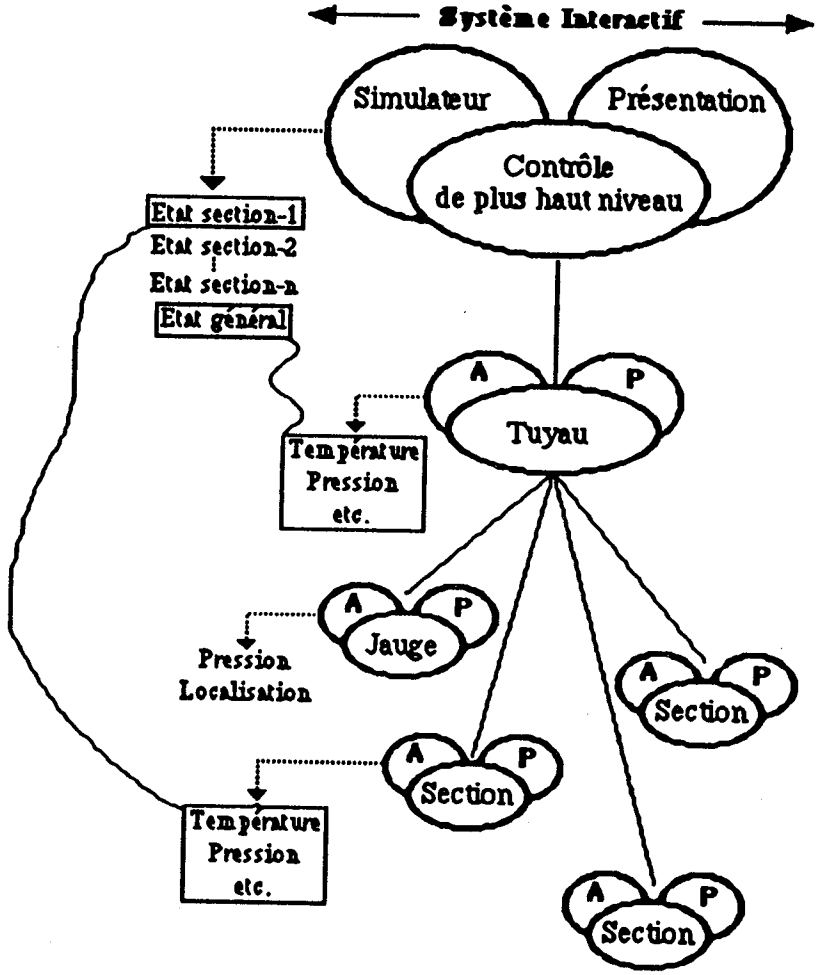


Figure 9.2 : Organisation d'objets PAC dans laquelle les structures de données du simulateur correspondent de manière biunivoque à un objet PAC. Les traits souples mettent en évidence l'association maintenue par le Contrôle du niveau le plus haut entre les concepts du simulateur et les abstractions d'objets PAC. Par exemple, la structure "Etat section-1" est liée aux abstractions d'un objet PAC de classe Section.

En réalité, un concept structuré d'une application n'est pas contraint d'être présenté par un objet composé PAC. Dans la figure 9.3, la notion de pression générale, qui fait pourtant partie de l'état général du concept de tuyau, n'est pas associée à une abstraction de Tuyau mais est liée à une abstraction de l'objet **Jauge**. La conséquence de ce choix est que la correspondance entre la pression générale et sa présentation par la jauge est gérée par le Contrôle du niveau le plus haut. Dans le cas de l'association à travers une abstraction de Tuyau, la correspondance est assurée à deux niveaux : une première fois par le Contrôle du sommet de la hiérarchie entre les éléments de l'état général et les abstractions de Tuyau ; une seconde fois par le Contrôle de Tuyau entre ses abstractions et celles de

Jauge. L'intérêt de la première solution est l'efficacité puisqu'il y a un seul niveau de liens à assurer ; celui de la deuxième solution est la répartition des traitements à des niveaux d'abstraction différents. Dans le cas présent, l'absence de traitement joue en faveur du second choix.

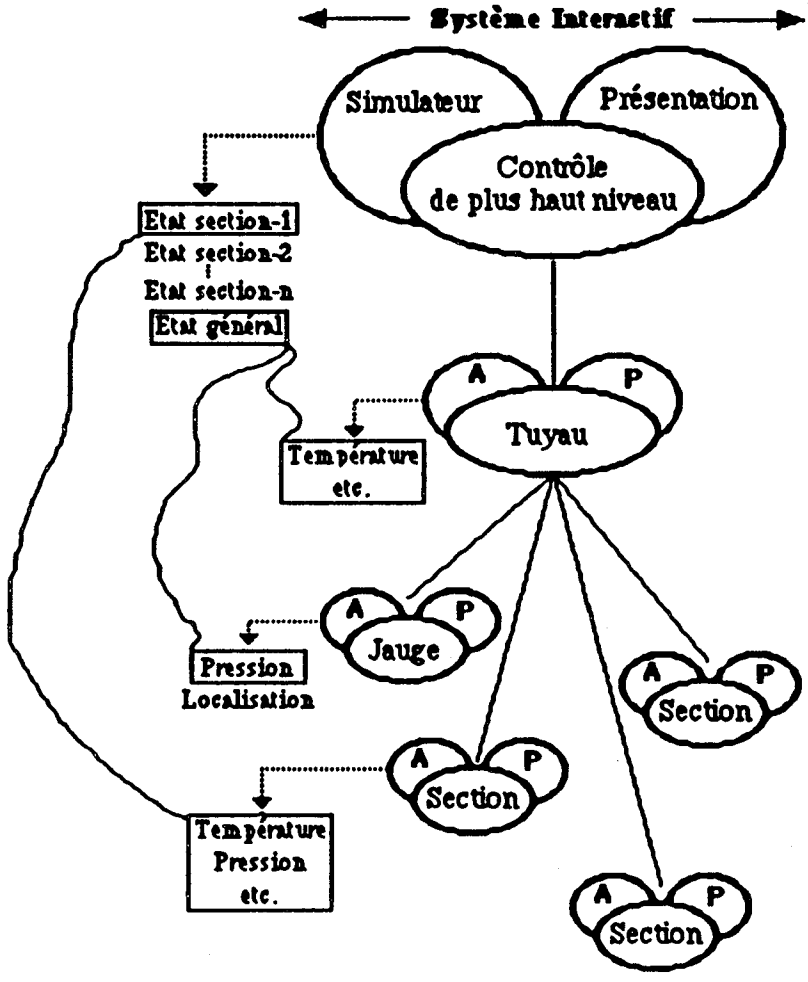


Figure 9.3 : Autre organisation où les structures de données du simulateur sont liées de manière distribuée à des abstractions d'objets PAC.

Après la description structurelle des objets et des liaisons avec les concepts du simulateur, nous abordons leur comportement de présentation.

2.2.2. Comportement des objets de présentation

La figure 9.4 illustre la répartition des rôles de présentation entre le tuyau, la jauge et les sections.

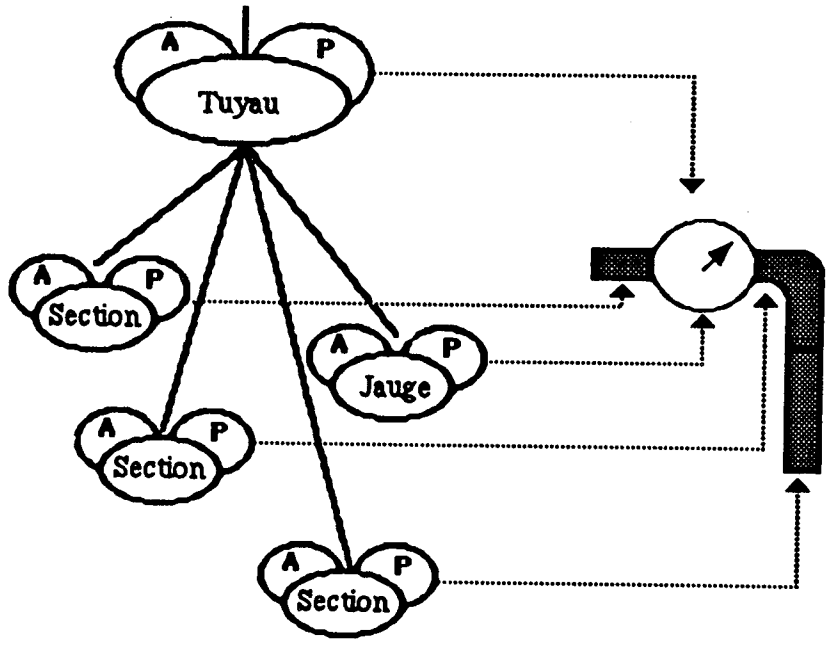


Figure 9.4 : L'organisation PAC et la répartition des rôles dans les tâches de présentation.

En sortie, la Présentation de Tuyau a la charge d'assembler les sections en une forme tubulaire, d'y fixer la jauge et de maintenir à l'intérieur du tube un flux animé de particules.

- L'assemblage graphique peut s'exprimer sous forme de contraintes spatiales appliquées aux Présentations des objets constituants. Chaque Présentation de section détermine une forme calculée par le Contrôle de section à partir des Abstractions de section. La Présentation de Tuyau, qui est un vérificateur de contraintes spécialisées et qui connaît l'ordre des sections, détermine la position et la taille de chaque forme élémentaire. L'ordre peut être modélisé implicitement selon la suite des créations des sections PAC, ou bien explicitement par un numéro associé à l'Abstraction de chaque section. La taille d'une section pourrait être fixe. L'intérêt de la modéliser par des paramètres est une souplesse accrue de l'affichage. Par exemple, on peut imaginer que la forme globale du tuyau soit proportionnelle à la taille de la fenêtre de restitution. A cette fin, la Présentation de Tuyau se chargerait de déterminer les paramètres qui modélisent les dimensions d'une section et demanderait aux sections de se dessiner selon ces valeurs.

- La localisation et la taille de la jauge sont soumises aux mêmes contraintes que celles des sections. Toutefois, la localisation justifie quelques commentaires. Si le simulateur manipule la notion de jauge, alors on peut imaginer que la localisation abstraite d'une jauge s'exprime par l'identification des sections adjacentes. Dans ces conditions, l'identification est liée à une abstraction de Jauge. Cette information permettra à la Présentation de Tuyau de déterminer la position graphique de Jauge. Si le simulateur n'a pas de concept de jauge mais ne manipule que la notion de pression, alors la localisation de la jauge est fixée a priori par la Présentation de Tuyau. On peut aussi imaginer que l'opérateur ait quelques droit de regard sur cette localisation. Nous reviendrons sur cette hypothèse.

- Le flux animé à l'intérieur du tuyau peut s'envisager de deux façons. Ou bien, chaque présentation de section effectue sa part ou bien Tuyau dessine l'ensemble du flux. La première option réutilise la compétence des sections mais impose une synchronisation dans l'affichage : les sections doivent se dessiner dans l'ordre et la sortie du flux d'une section doit coïncider avec le point d'entrée dans la section suivante. Cette synchronisation peut être facilement assurée par Tuyau qui connaît l'ordre des sections et qui peut interroger chaque section sur "l'état graphique" du flux au moment de la sortie. La seconde façon de créer le flux n'utilise

pas les compétences des sections. La Présentation de Tuyau effectue le remplissage dans la forme globale à partir des informations mémorisées dans l'Abstraction de Tuyau. Cette solution n'est possible que si l'Abstraction de Tuyau contient les informations nécessaires. Dans le cas inverse, Tuyau devrait interroger les sections afin de synthétiser les données abstraites manquantes.

Jusqu'ici, nous avons décrit le comportement en sortie de la Présentation du simulateur. Pour les entrées, nous faisons l'hypothèse que seuls les clics souris ont un sens. Un clic sur la forme tubulaire est interprété comme une demande d'observation détaillée d'une section. La Présentation de la section réceptrice du clic s'affiche en vidéo inverse et offre en sus une vue détaillée de son état. La figure 9.5 montre l'Image une fois la désignation complètement traitée. Le pointillé graphique qui traduit le lien entre les deux vues est géré par la Présentation de la section. L'enfoncement d'un bouton de la souris suivi d'un mouvement provoque le déplacement de la version détaillée jusqu'au relâchement du bouton. Le lien est redessiné en conséquence. Un clic sur la version détaillée ou sur l'élément en vidéo inverse du tuyau ferme la vue détaillée. Comme le suggère le schéma de la figure 9.5, ces phénomènes sont, en principe, entièrement gérés par la Présentation de la section manipulée. En réalité, cette gestion demande généralement l'intervention d'un arbitre pour effectuer des réparations graphiques.

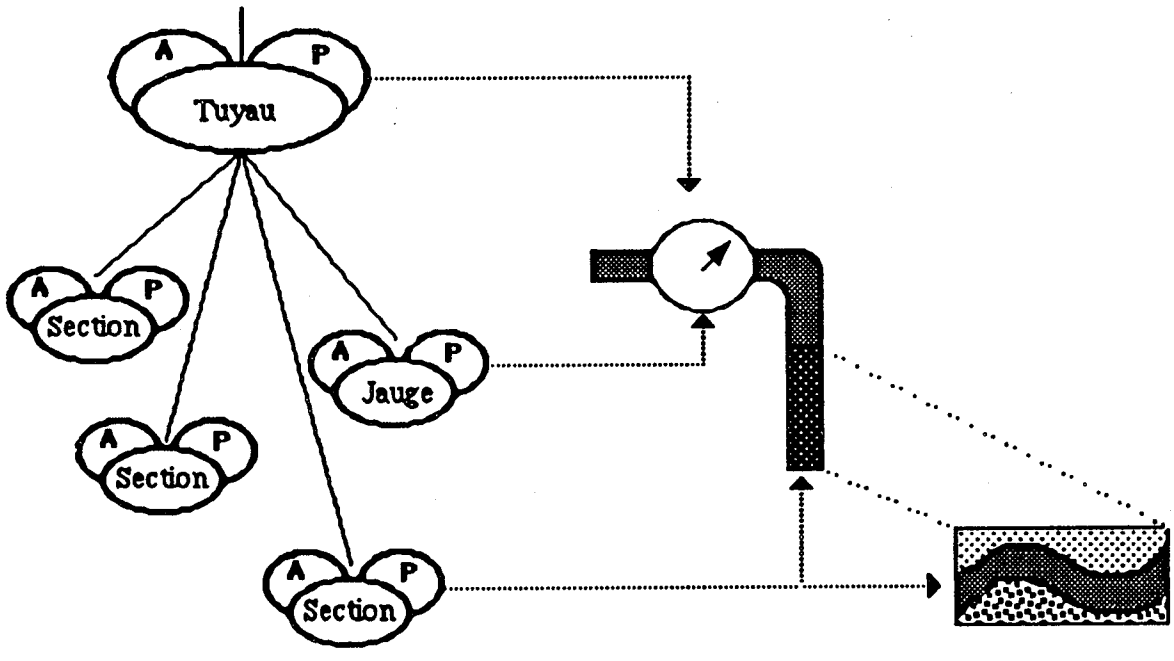


Figure 9.5 : Vue synthétique et vue détaillée.

Une conséquence directe du mouvement, de l'ouverture et de la fermeture des vues détaillées est le traitement de la superposition. Selon les outils de réalisation, la restauration des parties endommagées est automatique ou au contraire doit être programmée. Nous verrons dans la section II dédiée aux outils de construction d'interfaces que ce type de service n'est généralement pas disponible ou que, s'il existe, ne répond que partiellement aux besoins. La Présentation de Tuyau, qui gère

l'organisation géométrique de l'ensemble, est le candidat privilégié pour s'acquitter des tâches de réparation. En effet, lorsqu'une vue de section est déplacée ou fermée, la Présentation de la section en avertit Tuyau. Ce dernier détermine les dommages et demande aux Présentations détériorées de se redessiner dans l'ordre voulu. Le traitement du déplacement de la jauge, évoqué plus haut, s'effectue selon les mêmes principes : les événements qui reflètent les actions physiques de l'utilisateur sont interprétés par la Présentation de Jauge. Un dessin de jauge suit les mouvements de la souris, Tuyau est averti du changement et demande aux sections endommagées de se redessiner. Ce processus se répète jusqu'à ce que le bouton de la souris soit relâché.

Cet exemple montre comment répartir les aspects sémantiques, syntaxiques et lexicaux de l'interaction à divers niveaux d'abstraction. Le simulateur est libéré des tâches syntaxiques et sémantiques de bas niveau. Celles-ci sont déléguées à des objets interactifs. Par exemple, la désignation d'une section est une tâche syntaxique entièrement traitée par la hiérarchie d'objets PAC. Le changement du centre d'intérêt de l'utilisateur pour une section particulière est une tâche sémantique de bas niveau gérée en totalité dans l'interface.

3. Thermo, un système jouet

Ce système jouet va nous servir à illustrer une large part des possibilités du modèle PAC : communication entre l'application et son interface à un haut niveau d'abstraction, répartition du comportement sémantique, arbitrage entre les éléments de présentation grâce à la notion de Contrôle, parallélisme et dialogue à plusieurs fils, et représentation multiple d'un même concept. Avant d'aborder la mise en œuvre de ces possibilités avec le modèle PAC, nous présentons le comportement du système et son architecture.

3.1. L'objet du système et son Image

L'objet de ce système est de présenter les relations entre les notions de chaleur, de température et de volume d'eau. Nous avons choisi pour cela d'appliquer une métaphore du monde réel : un thermomètre indique la température actuelle, un réchaud fait office de source de chaleur, un récipient sert à recevoir un certain volume de liquide et un distributeur permet d'obtenir de l'eau. Au delà des relations entre ces notions, il se peut que l'utilisateur soit intéressé par l'évolution de la température. A cette fin, nous utilisons un objet spécialisé dans le tracé de courbe tel que l'axe horizontal représente le temps, et l'axe vertical, la température. Notre dernière contrainte est de présenter le système dans le contexte international. Plusieurs langues sont donc admises et il doit être possible de passer de l'une à l'autre en cours d'exécution. La figure 9.6 donne une idée de l'Image du système lorsque l'utilisateur travaille en langue anglaise.

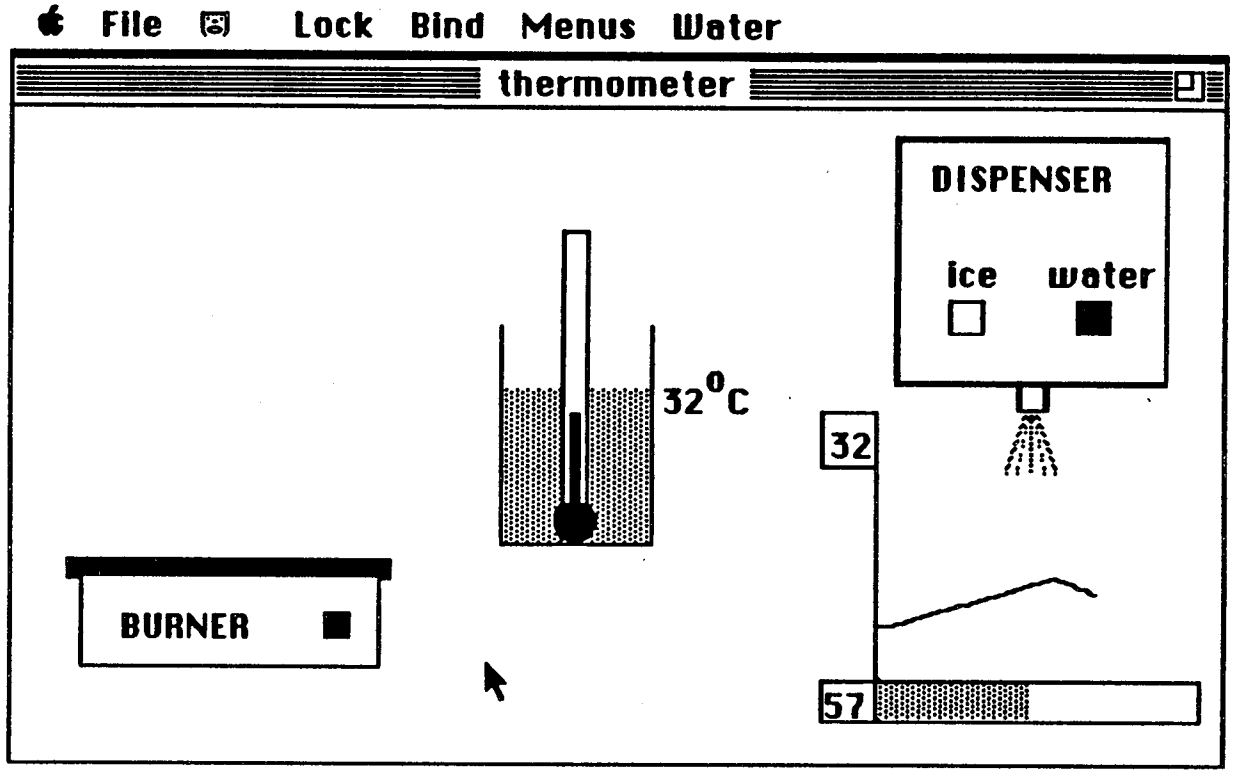


Figure 9.6 : Présentation des concepts de température, de chaleur et de volume à l'aide d'objets du monde réel. En bas et à droite, présentation de l'évolution de la température au cours du temps.

Comme dans le monde physique, l'utilisateur déplace les objets et actionne les interrupteurs. Ici, la souris sert d'extension électronique de la main pour reproduire les sensations de la manipulation directe. Par exemple :

- un clic-souris sur l'interrupteur du réchaud le met en marche ou au contraire, s'il est en marche l'arrête. L'interrupteur change de couleur pour indiquer l'état de marche ou d'arrêt. Lorsque le réchaud est assez chaud, des effluves se manifestent à la surface. Leur hauteur exprime le niveau de chaleur du réchaud. La figure 9.7 montre les principales présentations du réchaud en fonction de son état.

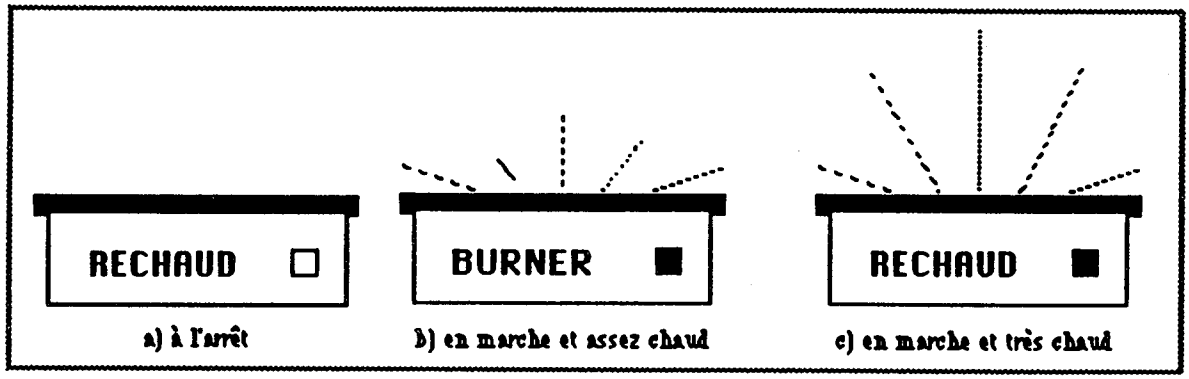


Figure 9.7 : Les différentes présentations du réchaud en fonction de son état. Noter au centre la version anglaise du réchaud lorsque la Présentation du système doit s'exprimer en langue anglaise.

Dans le monde réel, lorsque l'on dépose un objet sur la surface d'un réchaud, les effluves, s'i
y en avait, ne sont plus visibles. Ce phénomène est reproduit dans le système comme e
témoigne la figure 9.8.

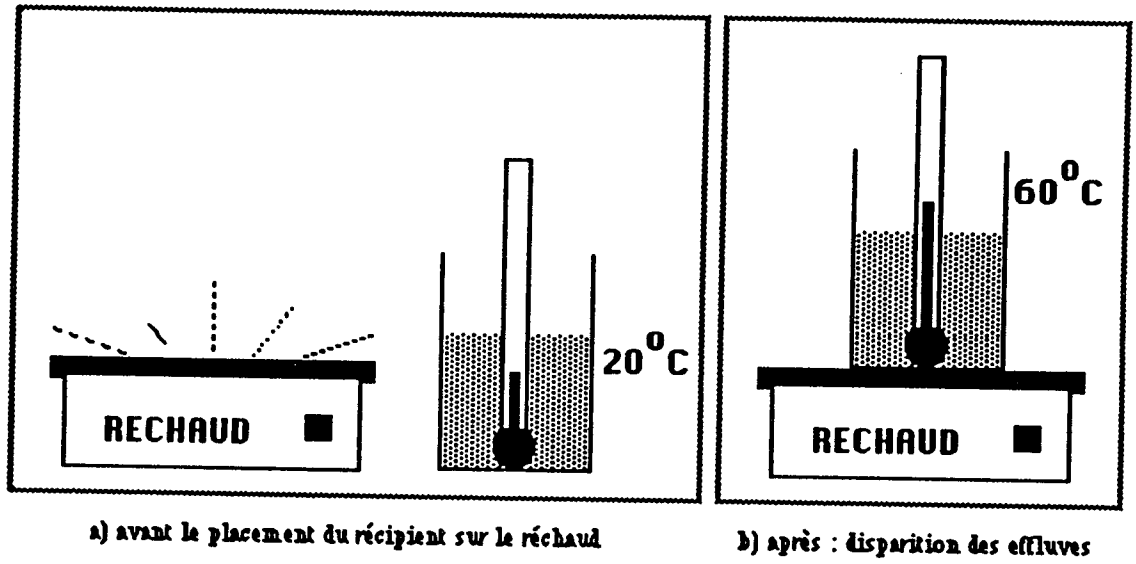


Figure 9.8 : Le placement du récipient sur le réchaud masque les effluves.

- un clic-souris sur l'indicateur d'eau du distributeur provoque l'écoulement d'une quantité fixe de liquide. Si le récipient est à ce moment-là situé sous le bec du distributuer, son volume d'eau monte progressivement. La figure 9.9 montre les différentes présentations du distributeur selon les circonstances.

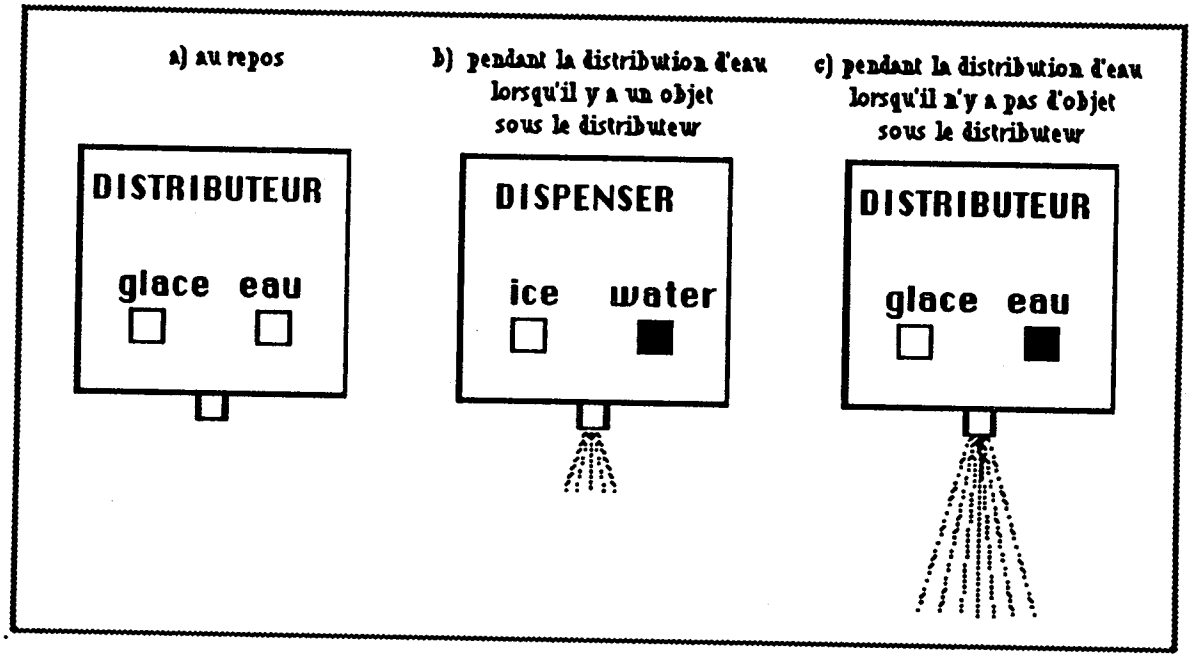


Figure 9.9 : Les présentations du distributeur selon son état et l'environnement. Noter au centre, la version anglaise du distributeur lorsque la Présentation du système doit s'exprimer en langue anglaise.

- une désignation accompagnée d'un balayage de la souris permet de déplacer le réchaud ou le récipient. Les objets en cours de déplacement suivent les mouvements de la souris. La même

action sur le distributeur n'a aucun effet : dans le monde réel, celui-ci est fixé au mur. Lorsque le bouton de la souris est relâché, l'objet manipulé cesse son mouvement mais subit les lois de l'environnement. Par exemple, si le récipient est relâché au voisinage du réchaud, il est automatiquement placé sur la surface chauffante mais si le réchaud est placé à proximité du distributeur, il revient automatiquement à sa place : dans le monde réel, il est préférable de ne pas placer un réchaud sous un distributeur! La figure 9.10 montre une vue du récipient en cours de déplacement.

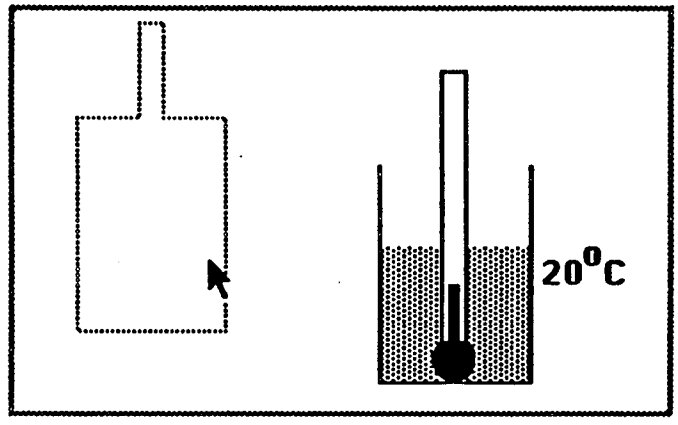


Figure 9.10 : Une vue du récipient lors du balayage de la souris. Afin de faciliter l'évaluation de la localisation de l'objet, le contour en pointillé est lié aux mouvements de la souris pendant toute l'opération.

- La variation du volume d'eau peut s'effectuer de trois manières : soit placer le récipient sous le distributeur et actionner l'indicateur d'eau ; soit désigner la surface du liquide et effectuer un balayage vertical de la souris pour accroître ou diminuer le volume ; soit utiliser la voie traditionnelle de la sélection des fonctions "add" ou "remove" du menu "water" de la figure 9.6. Lorsque l'eau atteint l'état d'ébullition, des bulles évoluent dans le récipient. Les figures 9.11 a) et 9.11 b) résument les états principaux du récipient.

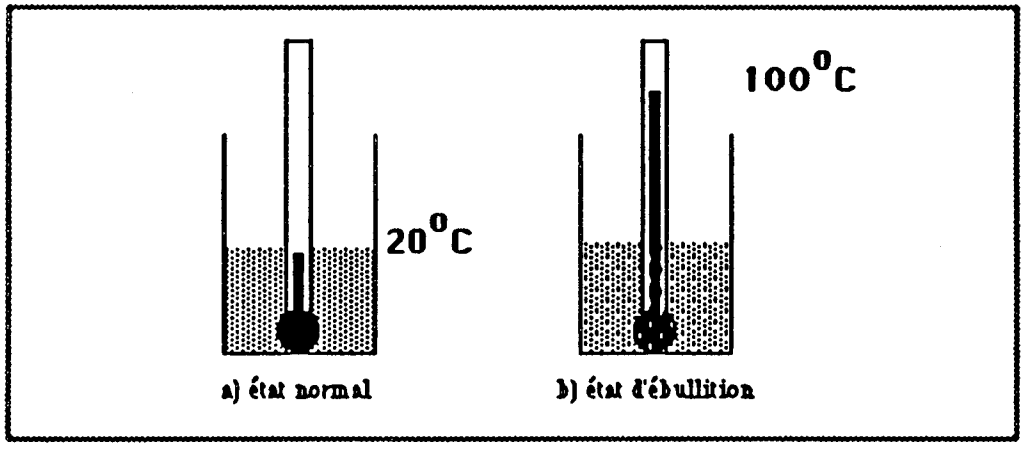


Figure 9.11 a) : Les présentations du récipient selon son état. Les états du liquide.

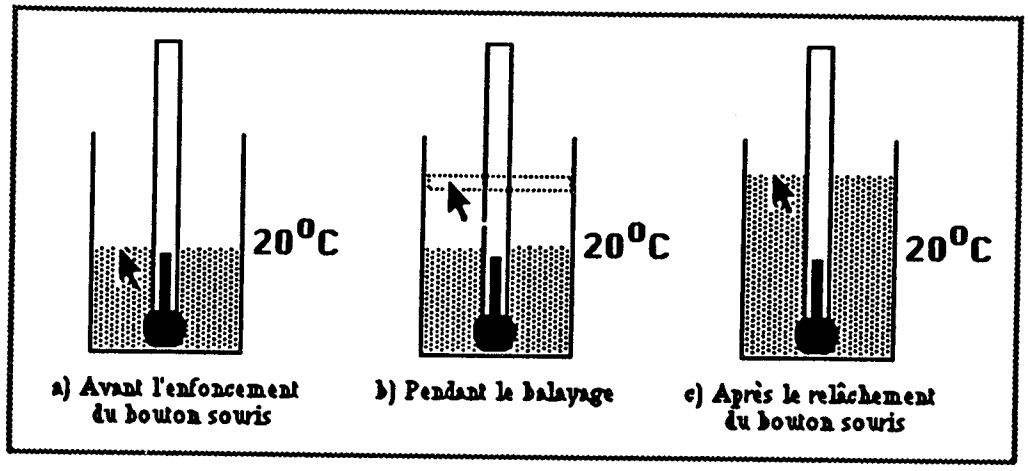


Figure 9.11 b) : Les présentations du récipient selon son état. La manipulation du volume avec la souris. Noter le retour d'information en pointillé qui, lié aux mouvements de la souris, concrétise la surface mobile du liquide.

3.2. L'architecture PAC du système

La figure 9.12 décrit l'architecture PAC du système Thermo. Au sommet de la hiérarchie, nous retrouvons les constituants habituels : l'Abstraction du niveau le plus haut modélise les lois de la thermodynamique. Elle est capable de déterminer à tout instant la chaleur produite par une résistance, de calculer la température d'un volume d'eau et de déterminer si cette eau est en état d'ébullition ou non. La Présentation du niveau le plus haut n'a pas d'attributs particuliers. Nous pouvons la considérer comme inexistante. Le Contrôle remplit ses fonctions usuelles. Celles-ci seront détaillées au cours de l'exposé.

Les objets composants se répartissent en deux catégories : ceux qui reproduisent des objets du monde réel en rapport étroit avec les fonctions du système tels le réchaud, le récipient et le distributeur, et ceux d'utilité générale tels l'aide, les boutons et les menus. Nous ne reviendrons pas sur le fonctionnement des objets généraux dont on trouve des réalisations prêtes à l'emploi dans la plupart des boîtes à outils.

La réalisation des objets élémentaires de la première catégorie respectent rigoureusement le modèle PAC. Par exemple, pour le réchaud :

- la Présentation sait dessiner une forme de réchaud à une certaine localisation sur une surface de restitution. Selon le point désigné avec la souris, elle interprète les actions de l'utilisateur soit comme un déplacement, soit comme une pression sur l'interrupteur. Nous avons vu l'effet visuel du déplacement : un contour en pointillé du réchaud se met à suivre les mouvements de la souris ; une pression sur l'interrupteur provoque un changement de couleur. Tous ces phénomènes visuels sont entièrement réalisés par la Présentation du réchaud et l'interprétation de l'action de l'utilisateur (déplacement ou pression) est signalée à son Contrôle ;

- le Contrôle traduit une pression de l'interrupteur par une négation de la valeur booléenn **EnMarche** de l'Abstraction. Le déplacement est soumis à une vérification qui relève d l'environnement. Nous verrons son traitement plus loin. Le Contrôle traduit aussi la valeu d'état abstrait **Chaleur** en une hauteur. Cette hauteur est utilisée par la Présentation pour dessine et animer les effluves ;

- l'Abstraction, avec ses valeurs d'état **EnMarche** et **Chaleur**, constitue un modèle de l'état d'u réchaud. Noter qu'elle ne communique jamais directement avec la Présentation (e réciproquement), le Contrôle servant de traducteur entre les deux perspectives de l'objet.

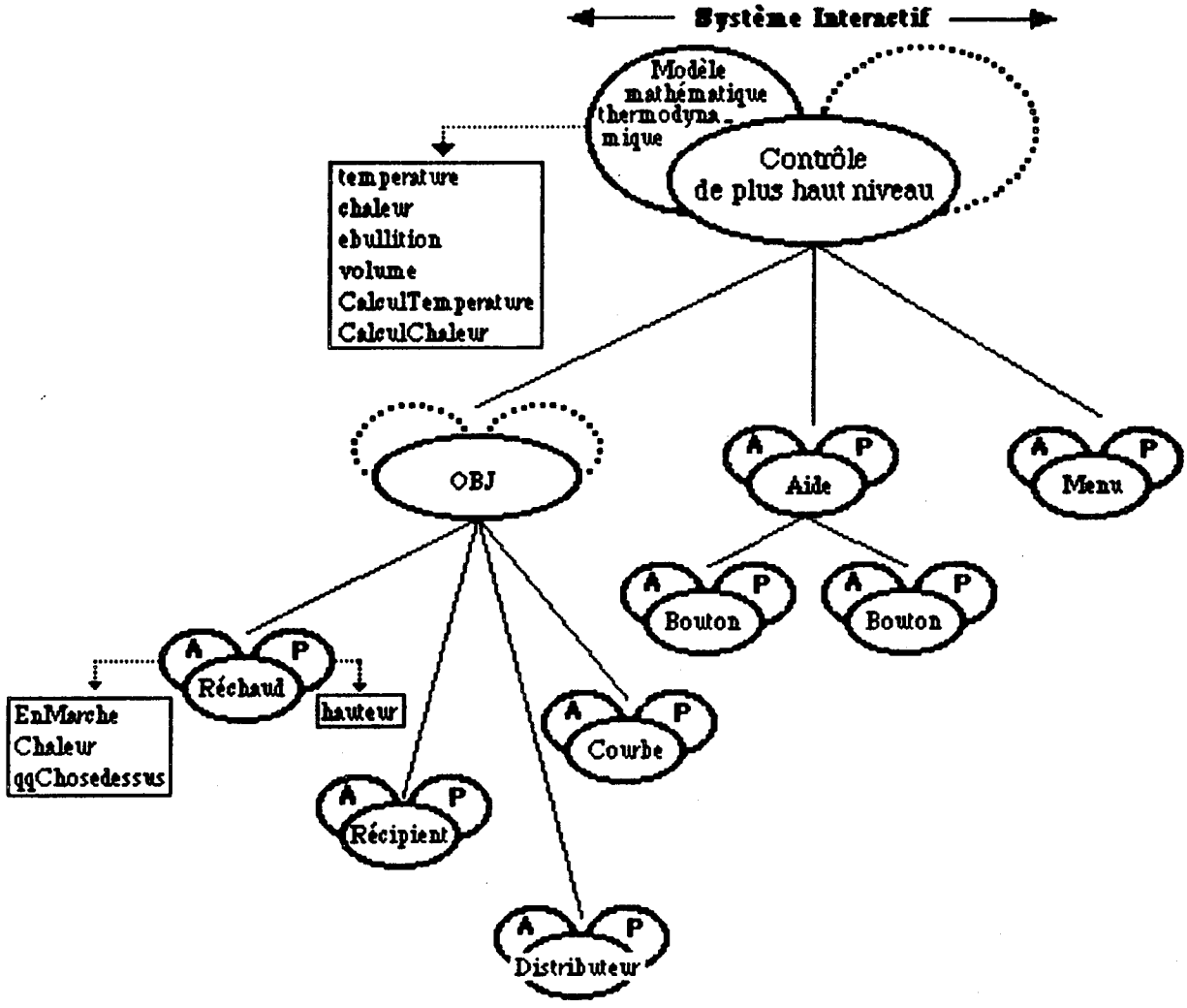


Figure 9.12 : L'architecture PAC du système Thermo.

Le récipient et le distributeur PAC sont conçus selon le même modèle. Nous verrons au chapitre 12 que ce modèle a fait l'objet d'une réalisation sous forme de squelette réutilisable.

Chaque objet élémentaire tend à reproduire le comportement intrinsèque d'un objet physique Tout comportement spécifique reste néanmoins sous l'influence des contraintes venues de l'environnement. Dans le cas précis de notre système, l'environnement voit cohabiter un réchaud, ur

récipient et un distributeur. Afin que cette cohabitation respecte les lois du monde physique, un agent doit la modéliser et la faire respecter. L'agent OBJ de la figure 9.12 remplit cette fonction d'arbitre. Il n'a pas de Présentation intrinsèque à proposer ni d'Abstraction : ses fonctions relèvent du Contrôle.

Après avoir considéré l'architecture générale du système, nous pouvons aborder les détails intéressants de la réalisation : le niveau des échanges entre l'application et l'interface, la répartition du comportement sémantique, les fonctions d'arbitrage des Contrôles, le dialogue à plusieurs fils d'activité et la représentation multiple de concepts.

3.3. Echanges entre l'application et l'interface à un haut niveau d'abstraction

Nous avons déjà insisté sur la nécessité pour l'application et son interface de communiquer en termes de concepts de l'application. Nous avons indiqué au chapitre précédent que le modèle PAC rendait possible ce niveau d'échange. L'architecture du système Thermo, tout comme celle du simulateur, indique comment exploiter le mécanisme d'indirection aménagé dans les Contrôles.

L'indirection permet à deux interlocuteurs de communiquer sans référence explicite. Dans le cas du système Thermo, l'application "parle" de température mais doit tout ignorer des techniques de présentation : afficher une température ne relève pas du modèle mathématique qui rend compte des phénomènes de la thermodynamique. De même, le thermomètre du récipient peut servir à présenter d'autres valeurs entières que celle de notre système. L'application et le récipient ne doivent donc jamais se désigner directement mais doivent passer par une indirection gérée dans le Contrôle du niveau le plus haut.

La technique de réalisation des indirections n'est pas unique : tables de liaison, valeurs actives à-la Loops, systèmes de règles. Le choix dépend essentiellement des outils de réalisation. Dans le cas de Thermo dont les objets s'exécutent dans un espace d'adressage unique, la technique des tables de correspondance convient parfaitement. Le schéma de la figure 9.13 indique les liaisons les plus significatives. On observe une table pour chacun des deux sens des échanges. Dans le sens application-interface, l'entier **temperature**, qui modélise la température, est lié à l'abstraction **Temp** d'un objet de classe récipient ; l'entier **chaleur** est lié à l'abstraction **Chaleur** d'un objet de classe réchaud. Dans le sens interface-application, l'abstraction **EnMarche** d'un objet de classe réchaud est liée à la fonction **CalculChaleur** de l'application et l'abstraction **qqChoseDessus**, à la fonction **CalculTemperature**. Ainsi, tout signal de modification de **temperature** par l'application est automatiquement traduit par le Contrôle du niveau le plus haut dans les termes de l'objet qui lui est associé. Dans le sens inverse, tout signal de modification d'abstractions d'objets de l'interface est reflété dans l'application.

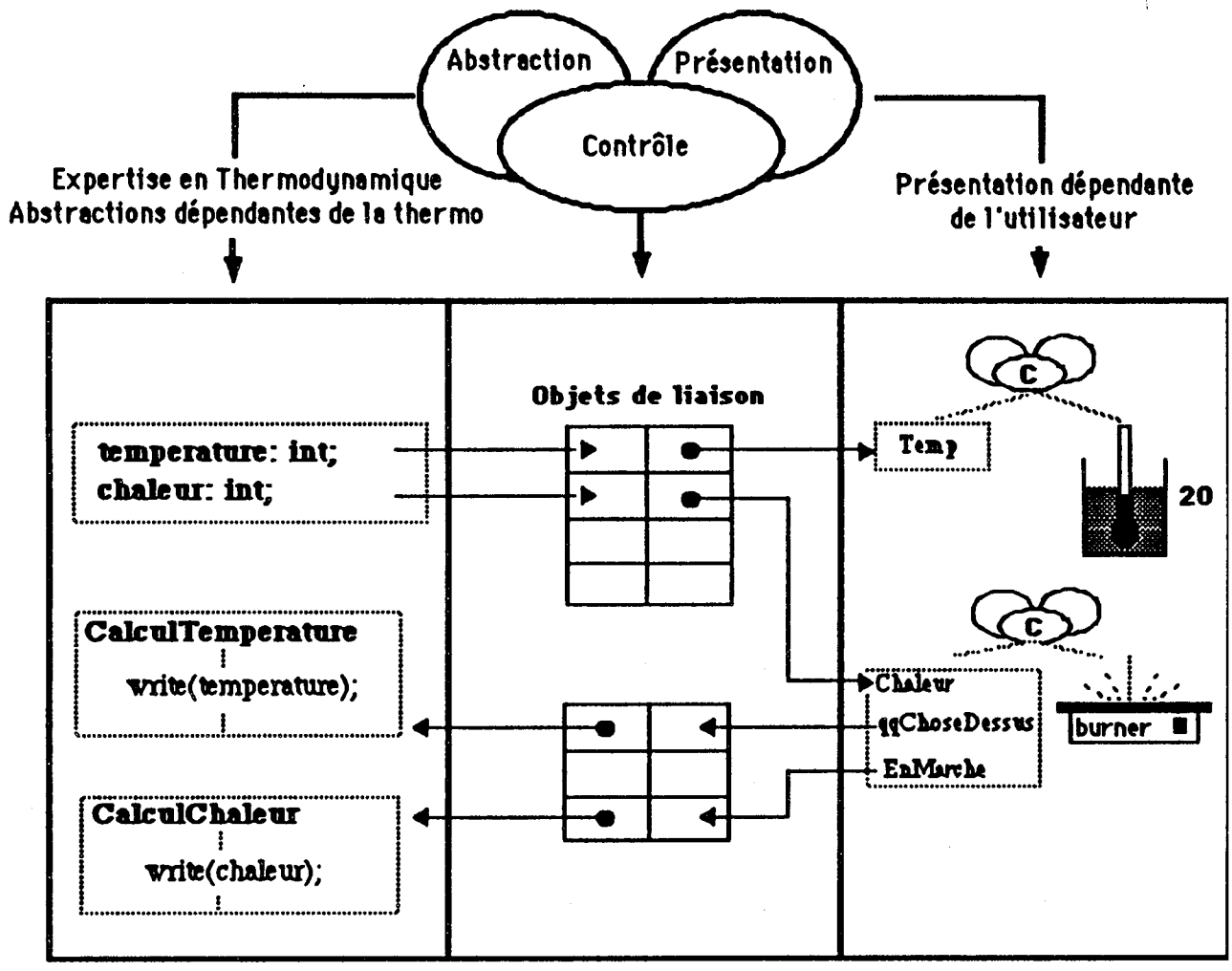


Figure 9.13 : Tables d'indirection entre abstractions de l'application et abstractions d'objets de présentation.

Les modifications qui interviennent pour une abstraction ne concernent pas seulement les changements de valeur. Elles peuvent aussi servir à traduire des changements de droit d'accès. Par exemple, il se peut qu'à partir d'un certain temps, l'expérience de thermodynamique n'autorise plus de modifier la valeur de **température**. Pour cela, il suffit que l'application signale au Contrôle du niveau le plus haut les nouveaux droits d'accès à **température**. La conséquence d'une telle invalidation est l'interdiction, par l'intermédiaire des objets de présentation, de toute action utilisateur qui aurait comme conséquence l'activation de la fonction **CalculTemperature**. La propagation de cette interdiction est assurée par la répartition du comportement sémantique présentée au paragraphe suivant.

En résumé, quelle que soit la nature du signal entre l'application et le Contrôle du niveau le plus haut, le sujet doit toujours être un concept de l'application, jamais un élément de présentation. Nous verrons au chapitre 12, un exemple de réalisation de protocole d'échange de haut niveau incluant la lecture, l'écriture, l'invalidation, la validation et la liaison de concepts.

3.4. Répartition du comportement sémantique

Le système Thermo met en scène deux univers : des concepts de thermodynamique et des objets physiques du quotidien. Il se trouve que ces objets sont le lieu de phénomènes thermiques. Pour cette raison, ils conviennent à la présentation de notions de thermodynamique. Il se trouve aussi que ces objets obéissent à des lois étrangères à la thermodynamique comme celle de la gravitation et de l'usage commun. L'expression de ces lois relève du niveau sémantique mais concernent des domaines distincts. Le modèle PAC, qui permet de répartir les fonctions sémantiques, suggère d'introduire un objet composé, OBJ, qui gouverne les relations des objets du quotidien selon les lois de la gravitation et de l'usage commun.

Ainsi, lorsque le réchaud (ou le récipient) est déplacé, la Présentation du réchaud se charge du suivi de la souris. Lorsque le déplacement est achevé, le Contrôle du réchaud avertit OBJ de la position envisagée. Ce dernier vérifie l'adéquation du nouvel emplacement. Si l'emplacement est accepté, le réchaud se dessine à la position prévue. Lorsque l'emplacement est refusé, OBJ suggère au réchaud une nouvelle position. La vérification de l'emplacement fait intervenir des connaissances sur la gravitation et des connaissances sur l'usage quotidien des objets mis en jeu.

- Pour la gravitation, OBJ, qui a connaissance de tous les objets de l'environnement, demande à chacun si l'emplacement envisagé par l'objet déplacé appartient à sa zone d'attraction. Si c'est le cas, l'objet interrogé indique son point d'attraction privilégié. Par exemple, pour un distributeur, le point d'attraction est sous le bec verseur alors que pour un réchaud, le point se situe au dessus de la plaque chauffante. Le point d'attraction définit la localisation définitive de l'objet déplacé.

- Pour l'usage quotidien, OBJ doit effectuer deux types de vérification. En complément de la gravitation, il doit s'assurer qu'un réchaud ne rentre pas dans le domaine d'attraction d'un distributeur, auquel cas, le réchaud reçoit l'ordre de rester à sa position initiale. La seconde forme de vérification a trait à la traduction dans le monde physique des interdictions (ou permissions) du monde de l'application. Par exemple, lorsqu'un réchaud reçoit un récipient (ou perd son récipient), OBJ l'en avertit. Grâce à ce signal, la Présentation du réchaud peut cesser (ou bien recommencer) de dessiner des effluves, l'abstraction **qqChoseDessus** du réchaud peut changer de valeur et la fonction **CalculTemperature** liée à cette abstraction peut être activée. Un second exemple est l'interdiction pour l'utilisateur de modifier un concept de l'application par l'intermédiaire des objets physiques. Pour ce faire, il suffit de refuser toutes les actions qui conduisent à modifier la ou les abstractions des objets de présentation en liaison avec le concept interdit.

Ainsi la répartition conjuguée à l'esprit de hiérarchie rend possible la cohabitation d'univers disjoints. La notion de hiérarchie conduit naturellement à celle d'arbitrage.

3.5. Les fonctions d'arbitrage des Contrôles

Avec la gravitation et les règles d'usage, OBJ remplit un premier rôle d'arbitre. A l'évidence, ces

fonctions d'arbitrage dépendent d'un domaine particulier. Nous allons maintenant nous pencher sur des fonctions générales comme l'allocation des ressources. Les ressources envisagées ici sont l'unité centrale et l'écran.

3.5.1. Allocation de l'unité centrale

Tout objet PAC est un lieu d'activités. Dans un environnement de réalisation multitâche, il est naturel d'associer un processus à chaque objet élémentaire et de laisser au noyau de gestion des processus la charge d'allouer l'unité centrale. Ce procédé convient lorsqu'il s'agit d'un noyau de processus légers à-la NeWS. Malheureusement, la plupart des noyaux, tel Unix, ne sont pas adaptés aux exigences de l'interaction : les changements de contexte sont trop coûteux et les règles d'allocation favorisent les ressources matérielles au dépend de l'utilisateur qui attend! Enfin, il se peut que le système d'accueil soit monotâche. Ces deux derniers cas, les plus fréquents, imposent l'écriture d'un miniallocateur d'unité centrale.

Le Contrôle du niveau le plus haut, ou plus précisément, le constituant qui reçoit les événements en provenance du système hôte, peut inclure cette fonction d'allocateur. Nous verrons les détails de la réalisation au chapitre 12. Pour l'instant, indiquons qu'avant de livrer l'événement suivant à son destinataire, le Contrôle du niveau le plus haut alloue successivement l'unité centrale aux interlocuteurs qui en ont fait la demande. Dans le cas de Thermo, les interlocuteurs sont l'application et OBJ. L'application a besoin de l'unité centrale pour évaluer la température et la chaleur tant que celles-ci n'atteignent pas un état stable. OBJ a besoin de l'unité centrale dès que l'un de ses constituants veut produire un effet d'animation. OBJ règle à son tour la priorité entre les demandeurs locaux. La figure 9.14 regroupe quelques effets d'animation dans le système Thermo.

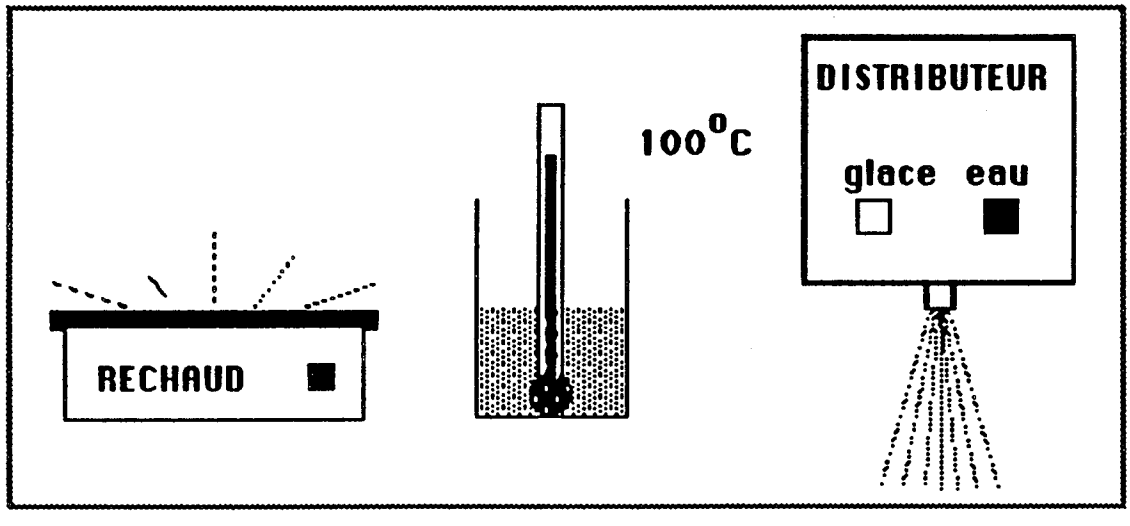


Figure 9.14 : le réchaud, le récipient et le distributeur ont besoin de l'unité centrale afin que leur Présentation produise des effets d'animation : effluves, ébullition, écoulement.

Un Contrôle peut donc utilement servir d'allocateur d'unité centrale. L'ensemble des allocateur ainsi répartis dans la hiérarchie PAC permet d'ajuster finement l'allocation.

3.5.2. Allocation de l'écran.

L'une des caractéristiques des interfaces de manipulation directe est la possibilité pou l'utilisateur de déplacer des objets. Ces mouvements ont une conséquence gênante pour le réalisateu d'interfaces : la gestion de la superposition d'objets. Celle-ci consiste à détecter les recouvrements et à déterminer l'ordre des rafraîchissements. Ainsi, dans l'exemple de la figure 9.15, l'afficheur d courbe est placé au dessus du récipient. Le récipient doit donc se dessiner sur l'écran avant l'afficheu de courbe. La détection des superpositions sort des compétences des objets concernés (qui de toute façon ignorent la présence de leurs prochains) mais relève de la gestion de l'environnement. OBJ chargé de l'environnement local, trouve dans le rafraîchissement de l'écran une nouvelle fonctior d'arbitrage.

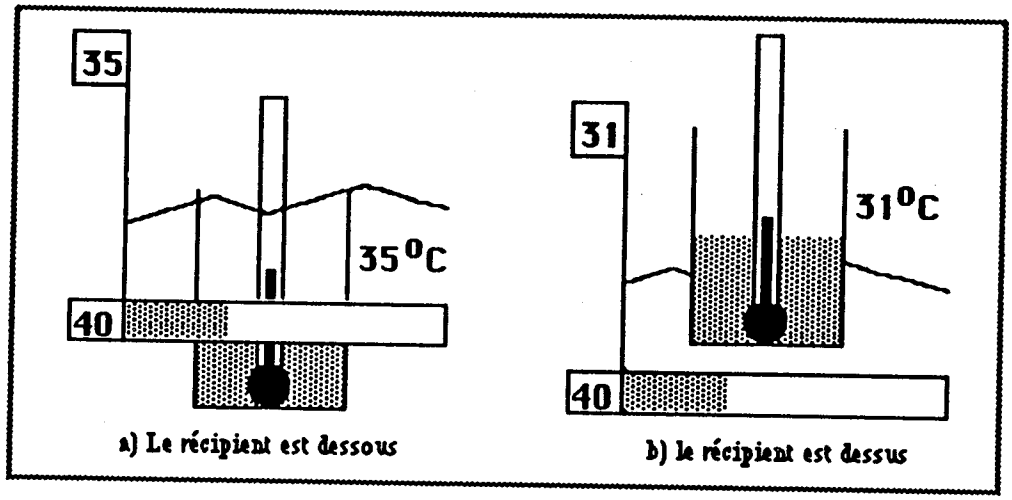


Figure 9.15 : Superposition d'objets : OBJ détecte le phénomène de superposition et règle l'ordre de rafraîchissement. Noter que l'afficheur de courbe est partiellement transparent alors que le récipient est opaque.

3.6. Dialogue à plusieurs fils d'activité

Tout objet PAC mémorise un état laissant à l'utilisateur l'initiative de la conduite du dialogue. Au cours du dialogue, il arrive que l'utilisateur fasse appel à un service qui, à son tour, a besoin de l'intervention de l'utilisateur pour poursuivre sa tâche. Par exemple, la figure 9.16 montre une Image du système Thermo après une demande de renseignements sur l'utilisation du réchaud. Le service d'aide présente quelques informations préliminaires et attend l'intervention de l'utilisateur pour compléter sa fonction.

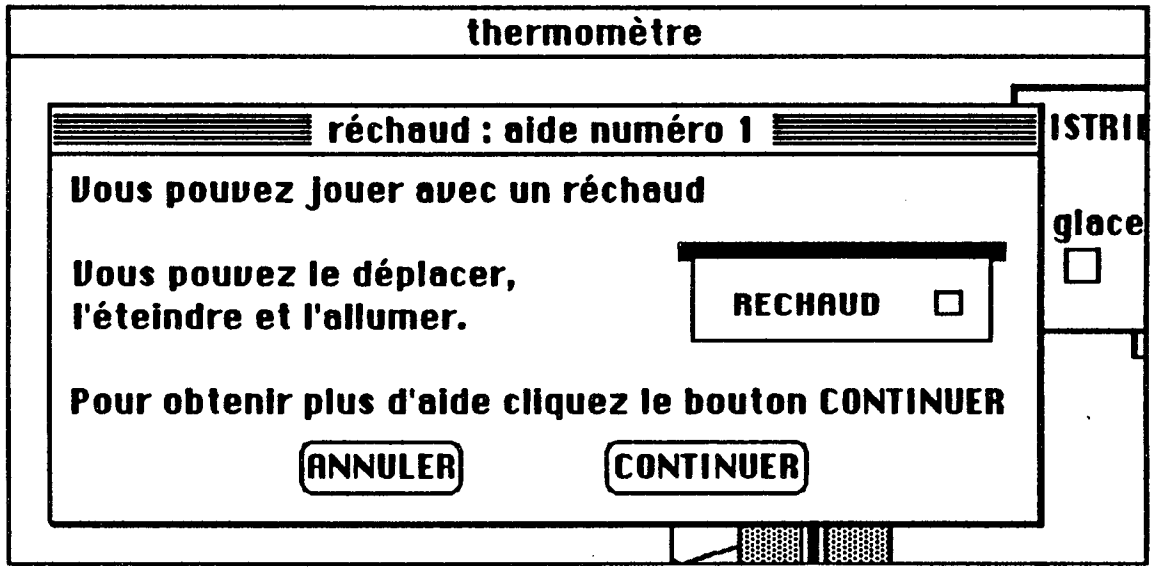


Figure 9.16 : Dialogue à plusieurs fils : l'utilisateur n'est pas contraint de répondre à la demande d'aide complémentaire. Il peut reprendre ses actions sur d'autres objets PAC.

Le Contrôle du service d'aide maintient la liaison entre la valeur booléenne **more** et les abstractions des deux boutons **ANNULER** et **CONTINUER**. Selon, le bouton sollicité, le Contrôle affecte à **more** la valeur faux ou vrai qui, à son tour, détermine la suite à donner. L'état d'attente étant modélisé dans le Contrôle du service d'aide, rien n'impose à l'utilisateur de répondre immédiatement. Celui-ci peut à sa guise poursuivre la manipulation d'autres objets avant de revenir au service d'aide.

3.7. Représentation multiple de concepts

Une des retombées du mécanisme d'indirection est la possibilité de présenter un concept par une multitude d'objets. La liaison simple présentée à propos de la communication à un haut niveau d'abstraction peut être facilement étendue à une liaison multiple comme le montre la figure 9.17. Dans l'exemple, toute modification de **temperature** est immédiatement reportée sur l'ensemble des abstractions d'objet interactif qui lui sont associées. L'ajout ou le retrait d'une liaison peut s'effectuer dans le cadre du protocole de haut niveau déjà évoqué.

En résumé, le système Thermo, illustre des problèmes courants, parfois difficiles à satisfaire mais que le modèle PAC permet de résoudre simplement. Nous avons vu en particulier comment séparer les fonctions de l'application des techniques de présentation tout en assurant la cohabitation de domaines sémantiquement distincts. Nous avons observé le rôle central mais réparti des Contrôles dans les diverses facettes de l'arbitrage : allocation de ressources, vérification de contraintes de toute nature et représentation multiple. Nous retrouvons quelques unes de ces propriétés dans le système ELOISE.

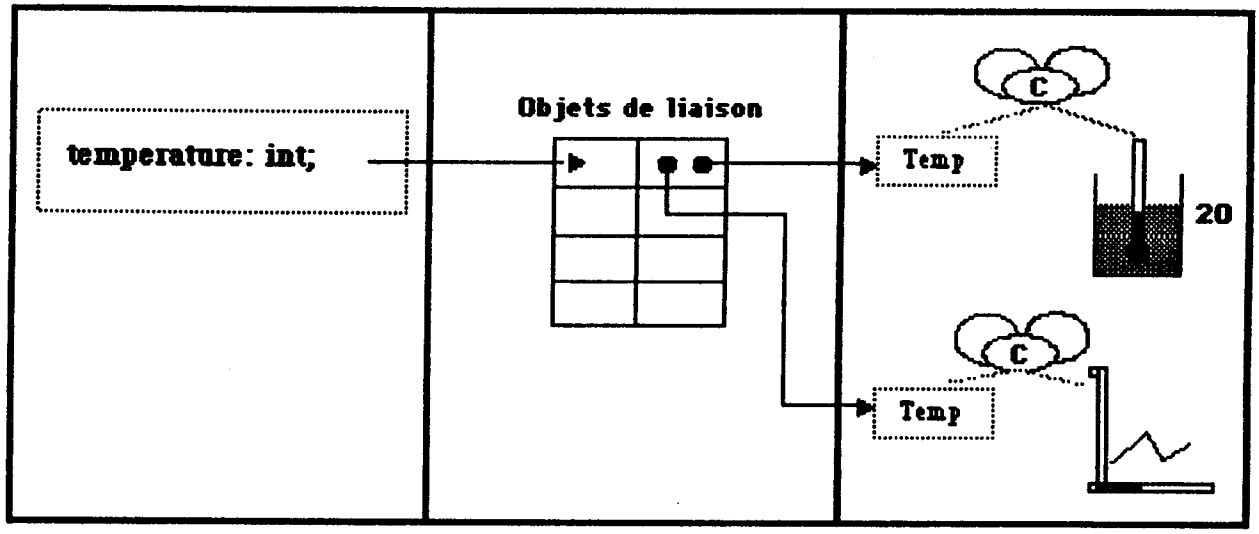


Figure 9.17 : Liaison multiple entre concepts de l'application et abstractions d'objets interactifs.

4. L'environnement de programmation ELOISE

A la différence du système Thermo réalisé spécialement pour illustrer les possibilités de PAC, ELOISE correspond à un besoin réel : disposer d'un environnement de programmation pour l'écriture de systèmes experts. ELOISE a été l'occasion d'appliquer les méthodes informelles de conception présentées dans la section 1. Celles-ci recommandent de procéder en premier lieu par l'analyse de tâche (ici, l'activité de programmer), puis de définir les fonctions du système convenant aux concepts mis en jeu dans la tâche, de concevoir enfin une Image adaptée à l'utilisateur type du système. Ces trois aspects font l'objet des paragraphes suivants. L'architecture PAC de l'ensemble est ensuite présentée.

4.1. Analyse des besoins

Toute tâche de programmation est soumise au concours de trois facteurs influents : l'adéquation du langage à l'expression du problème, les fonctions de l'environnement pour la construction et la mise au point des programmes et la qualité de l'Image.

4.1.1. Adéquation du langage

La classe des problèmes envisagés avec ELOISE est la modélisation de connaissances mal structurées. Si le cadre rigoureux de l'algorithmique est inadapté à la représentation de ce type d'information, les systèmes de productions conviennent parfaitement à la situation : une règle définit

un quantum de savoir-faire et un élément de la mémoire de travail représente un fait élémentaire. ELOISE reprend le modèle des systèmes de productions [Buchanan 84] dont la figure 9.18 schématise le fonctionnement : le raisonnement est mis en œuvre par une base (ou mémoire) de règles qui agit sur les éléments de la mémoire de travail. L'algorithme d'unification détermine un ensemble de règles activables "instanciées" avec les éléments de la mémoire de travail qui en satisfont les conditions. L'algorithme de contrôle sélectionne dans cet ensemble la règle à exécuter. L'exécution d'une règle peut à la fois modifier, créer ou supprimer des éléments de la mémoire de travail, créer ou supprimer des règles de la base de règles et échanger des informations avec l'utilisateur.

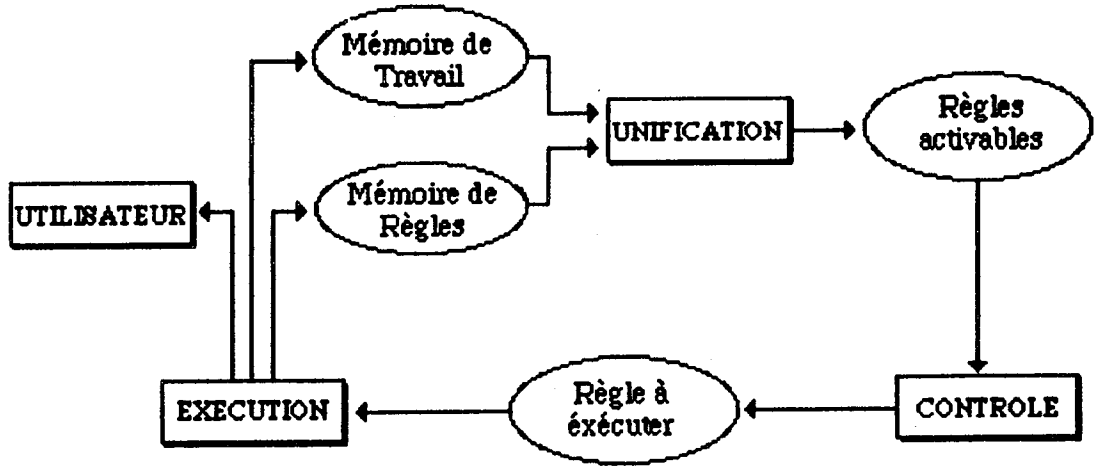


Figure 9.18 : Principe de fonctionnement d'un système de productions.

Le langage ELOISE réunit les avantages de la représentation de la connaissance structurée avec la notion de schéma [Schank 77] et la puissance du contrôle d'inférence avec la notion de règle. Le programmeur d'un système expert a ainsi la possibilité d'utiliser simultanément chacun des deux formalismes pour ses points forts. Son adéquation à la formalisation de la connaissance experte a pu être mesurée en vraie grandeur avec la réalisation d'IRENE, un système expert pour la configuration logicielle et matérielle de réseaux locaux XNS (Xerox Network System).

4.1.2. Adéquation des fonctions de l'environnement

Les fonctions de l'environnement constituent le second facteur marquant de la productivité du programmeur. Celles d'ELOISE sont le fruit de nos observations d'environnements de programmation par règles. Ces expériences nous ont permis de constater trois inconvénients majeurs dans les environnements usuels : le manque de retour d'information immédiat, le manque de support syntaxique et le manque d'intégration entre les services.

Absence de retour d'information : dans l'environnement OPS5 [Brownston 85], l'utilisateur peut modifier la mémoire de travail et la base de règles de manière incrémentale, tout comme il a la possibilité d'exécuter un programme pas à pas, mais dans les deux cas, l'environnement ne fournit aucune indication sur l'état du système. S'il souhaite être informé de la nouvelle situation, l'utilisateur doit émettre les commandes qui présentent le contenu des mémoires de travail et de règles. En outre, l'affichage s'effectuant sur une surface de restitution élémentaire, les informations disparaissent de l'écran à mesure que de nouvelles informations sont affichées. Dans ces conditions, l'utilisateur ne peut évaluer de manière précise l'état de l'espace d'exécution. Il se voit noter sur une feuille de papier les résultats intermédiaires intéressants avant que ceux-ci ne disparaissent de l'écran. De toute évidence, l'environnement devrait prendre à sa charge la fonction d'extension de la mémoire à court terme assurée manuellement avec la feuille de papier.

La deuxième lacune fréquemment rencontrée dans les environnements de programmation par règles est l'absence de support syntaxique. La syntaxe d'un langage de programmation définit un protocole que le système impose à l'utilisateur. Par définition, ce protocole est artificiel. Il entrave l'objectif de l'utilisateur qui est l'expression d'une certaine sémantique. L'environnement, qui connaît les conventions d'écriture, devrait naturellement apporter son concours à la construction de la forme syntaxique. L'écriture de programmes assistée par ordinateur est à l'origine de nombreux éditeurs de structures. Contrairement à toute attente, certains de ces éditeurs n'ont pas toujours eu le succès escompté : si l'objectif était louable (éliminer les erreurs de syntaxe et certaines erreurs sémantiques), l'Image imposait au programmeur un mode de travail qui n'était plus le sien (suivi de l'arbre abstrait, présentation inhabituelle) [Neal 87]. A la lumière de ces expériences, nous voulons pour ELOISE un support syntaxique libéré de toute forme de contrainte pour l'utilisateur.

Le dernier inconvénient souvent relevé dans les environnements de programmation est le manque d'intégration entre les services. L'utilisateur observe des phases ou des modes de travail, typiquement: l'édition, la liaison, l'exécution et le deboggage. Pour ELOISE, il n'y aura qu'un seul mode : celui de l'édition. Ceci est rendu possible par l'interprétation incrémentale du langage. La souplesse des services d'un interpréteur s'effectue au prix d'une perte en efficacité. A cela, ELOISE répond par la possibilité de produire du code "compilé" en OPS5 lorsque le programme est considéré comme correct.

Les fonctions d'ELOISE sont maintenant abordées à travers l'Image présentée à l'utilisateur.

4.2. Les fonctions et l'Image du système

Afin d'assister le programmeur dans sa tâche, tous les concepts qui présentent un intérêt sont à tout moment consultables et manipulables : la mémoire de travail, la base de règles et l'ensemble des

règles activables peuvent être présentées dans des fenêtres distinctes automatiquement mises à jour à chaque modification. Parmi ces concepts, l'ensemble des règles activables intervient de manière essentielle dans la mise au point d'un programme. Il est aux langages par règles ce que la pile est aux langages de programmation classiques.

Les figures 9.19 et 9.20 montrent un exemple d'évolution de la mémoire de travail et de l'ensemble des règles activables pour le système expert IRENE. Dans ces Images, nous observons les fenêtres ELOISE ouvertes pour les besoins de la mise au point : **Action sur MT** qui présente une partie des éléments de la Mémoire de Travail et **Visu Règles Activables** qui montre le contenu actuel de l'ensemble des règles activables. Les fenêtres **PALETTE** et **CONFIGURATION** appartiennent à la Présentation du système expert. Leur rôle sera précisé au paragraphe 5 avec la description de ce système. Pour l'instant, indiquons que les actions de l'opérateur sur les entités de la Présentation d'IRENE changent l'état de l'expertise exprimée dans le langage ELOISE. L'environnement ELOISE permet à l'opérateur d'en observer l'évolution.

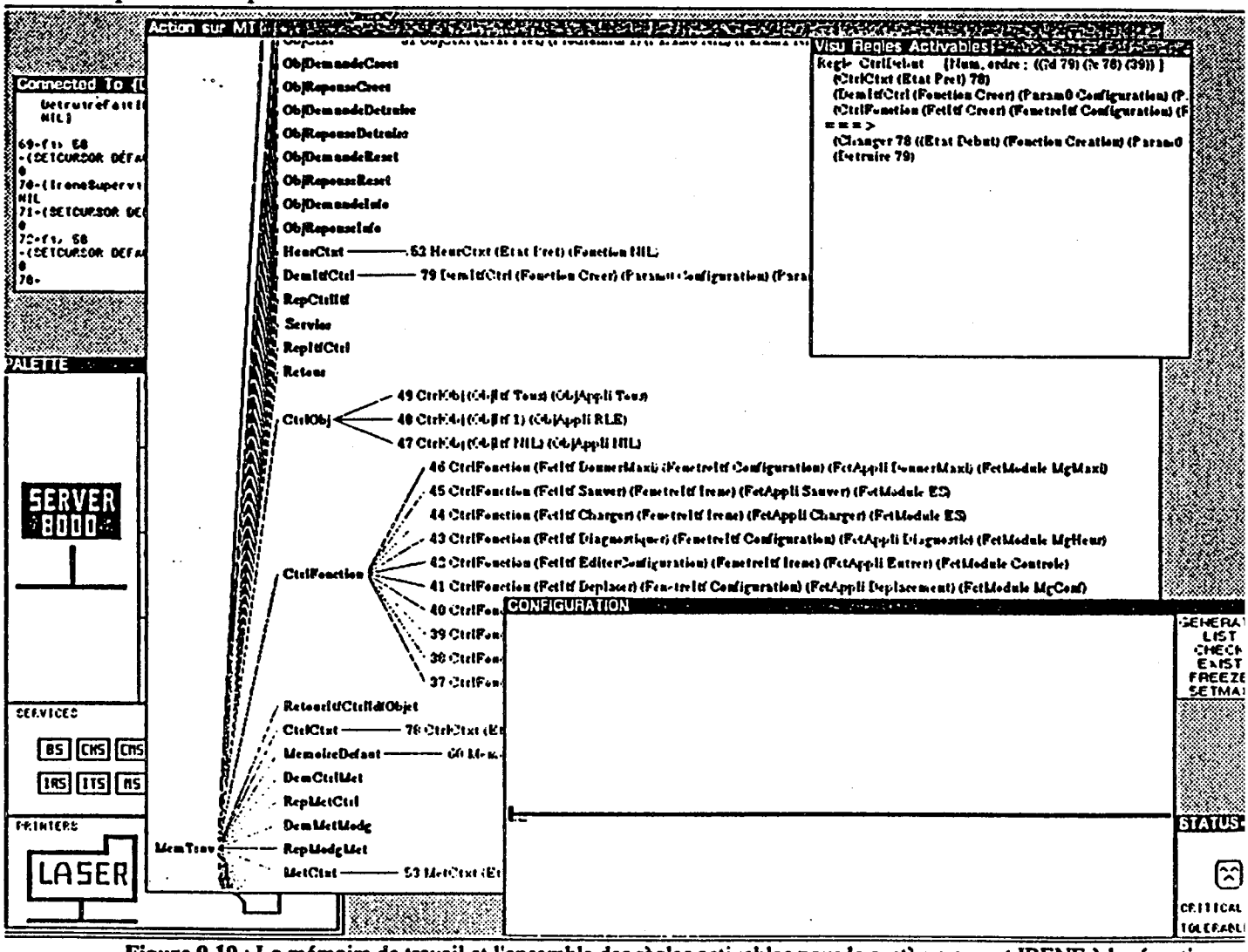


Figure 9.19 : La mémoire de travail et l'ensemble des règles activables pour le système expert IRENE à la réception d'un demande de création d'un fait.

Dans l'Image initiale de la figure 9.19, l'opérateur vient de demander la création d'un exemplaire de SERVEUR 8000. La Présentation d'IRENE vient d'afficher l'icône sélectionnée en vidéo inverse et vient d'adresser à l'environnement ELOISE une demande de création d'un fait de classe serveur. L'Image actuelle est celle après l'exécution d'un cycle du moteur d'inférence. La mémoire de travail ne contient pour l'instant aucun exemplaire de la classe serveur et la règle CtrlDebut est activable.

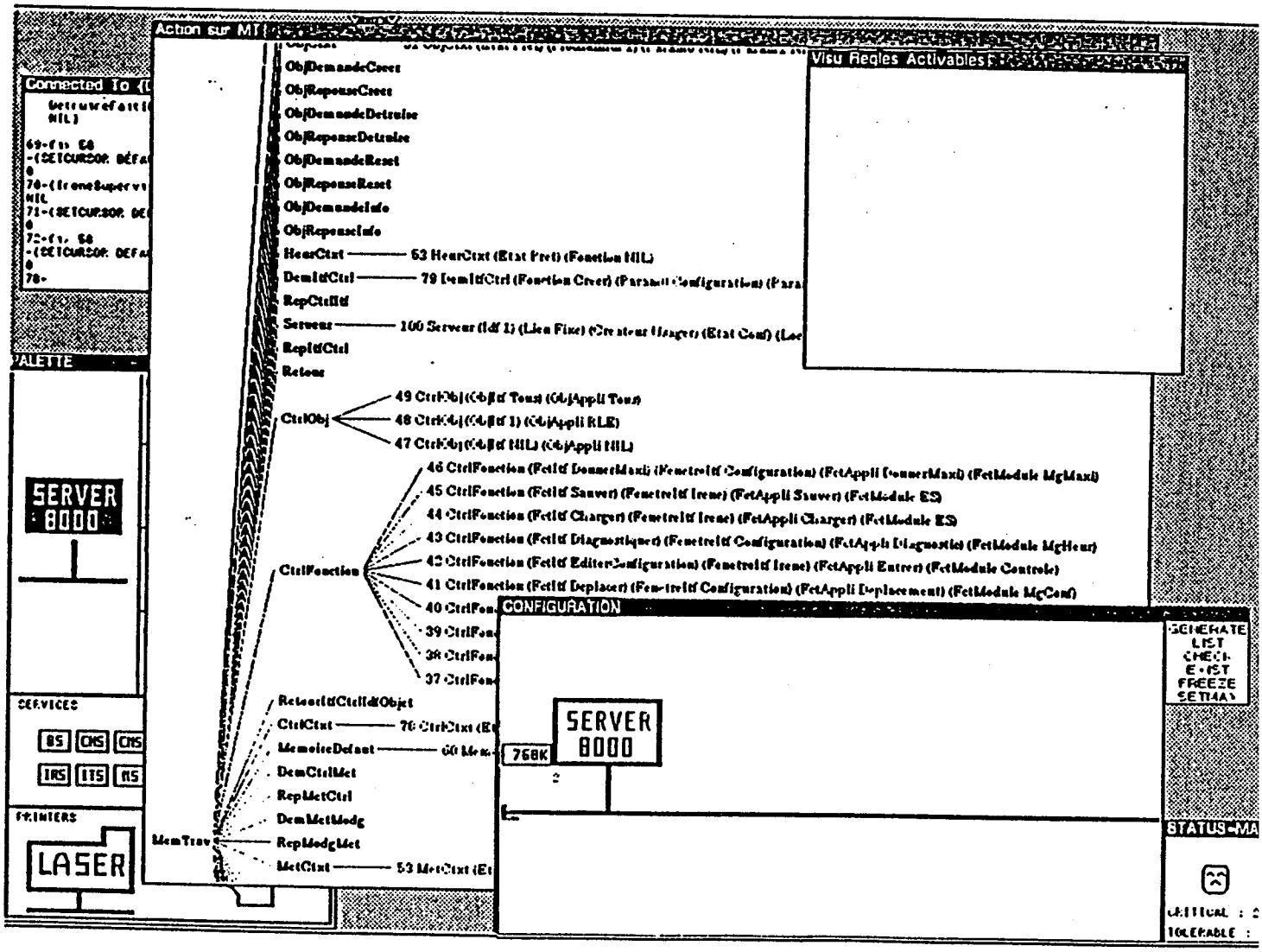


Figure 9.20 : La mémoire de travail et l'ensemble des règles activables pour le système expert IRENE après la création du fait.

La figure 9.20 montre l'Image après le traitement complet de la demande de création : la mémoire de travail s'est enrichie d'un exemplaire de serveur et il n'y a plus de règle activable puisque le moteur d'inférence vient d'achever le traitement de la demande. Les effets de bord sur la Présentation du système IRENE sont pour l'instant hors de notre propos, mais il est important de noter l'intégration totale entre l'exécution, la mise au point et l'élaboration du programme expert.

Rendre les concepts explicites est un premier effort. Les rendre explicites mais aussi utiles relève sensiblement le défi. Pour être utile, la présentation d'un concept doit évoluer avec la tâche alors que les besoins d'une tâche sont difficilement évaluables de manière exhaustive. Pour ELOISE, nous nous sommes inspirés des lacunes d'OPS5 et des méthodes de mise au point sous OPS5. Nous avons ainsi identifié la nature des représentations multiples. Par exemple, comme en témoigne la figure 9.21, le contenu de la mémoire de travail peut simultanément être présenté sous forme d'une arborescence dans une première fenêtre, sous forme textuelle triée par classe de faits dans une seconde fenêtre ou encore triée selon la valeur de l'estampille dans une troisième fenêtre. Il se trouve que ces trois catégories de présentation correspondent chacune à une perspective utile au raisonnement: l'arborescence indique les relations de dépendance hiérarchique entre les éléments, le tri par classe permet de consulter l'ensemble des faits de même nature et le tri par estampille indique l'ordre de création des faits et leur incidence sur les prochaines activations de règles.

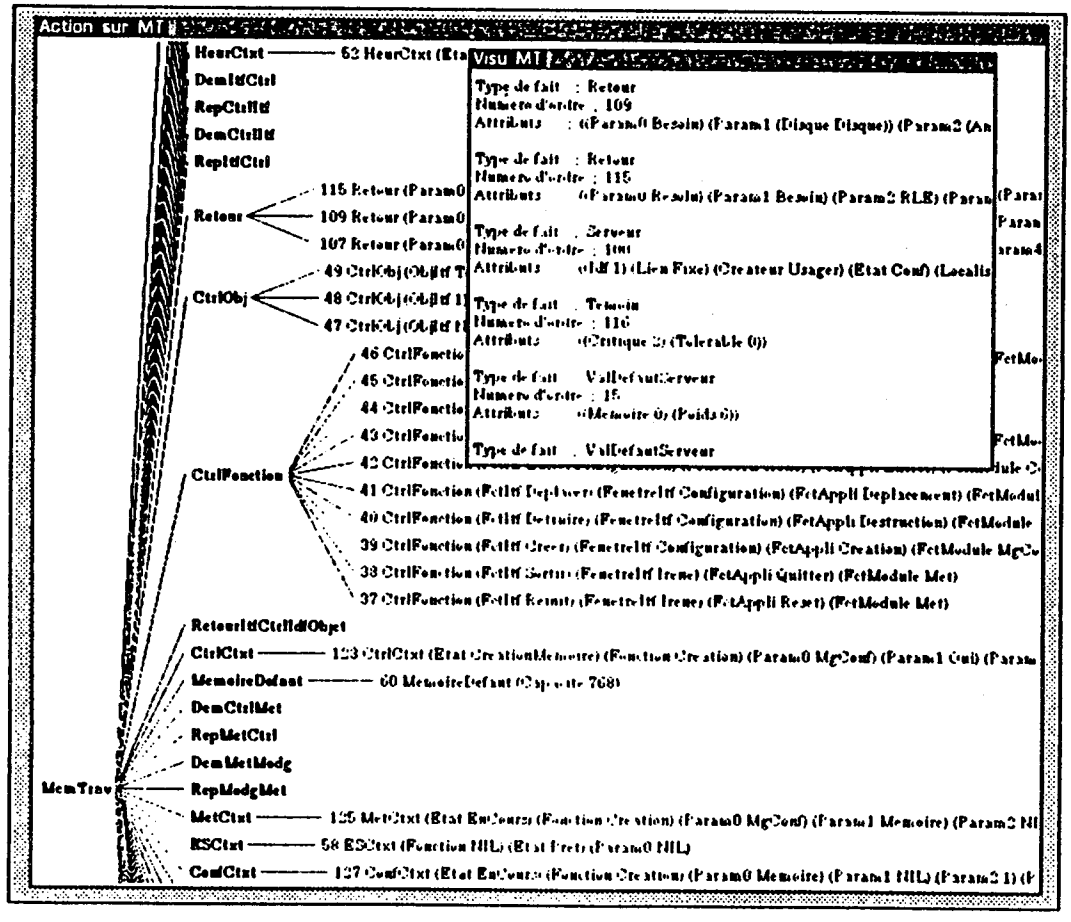


Figure 9.21 : Vues multiples simultanées du contenu de la mémoire de travail
a)- Mémoire de travail sous forme arborescente et mémoire de travail présentée par type de faits.

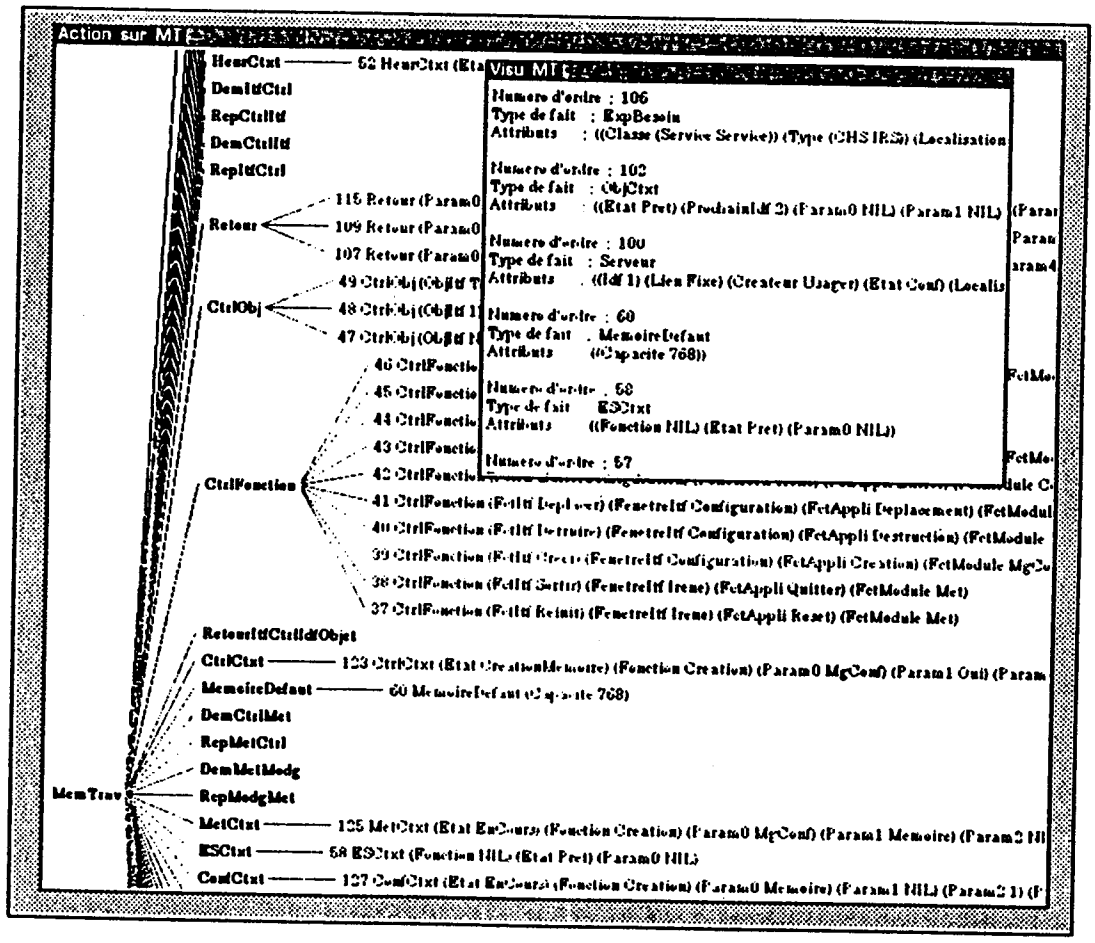


Figure 9.21 : Vues multiples simultanées du contenu de la mémoire de travail.
b)- Mémoire de travail sous forme arborescente et mémoire de travail présentée par numéro d'ordre de création (estampille).

Alors qu'une consultation générale s'appuie essentiellement sur l'observation de vues synthétiques comme celles de la figure 9.21, l'édition nécessite au contraire des représentations détaillées. La figure 9.22 montre la vue d'un élément de la mémoire de travail en cours d'édition.

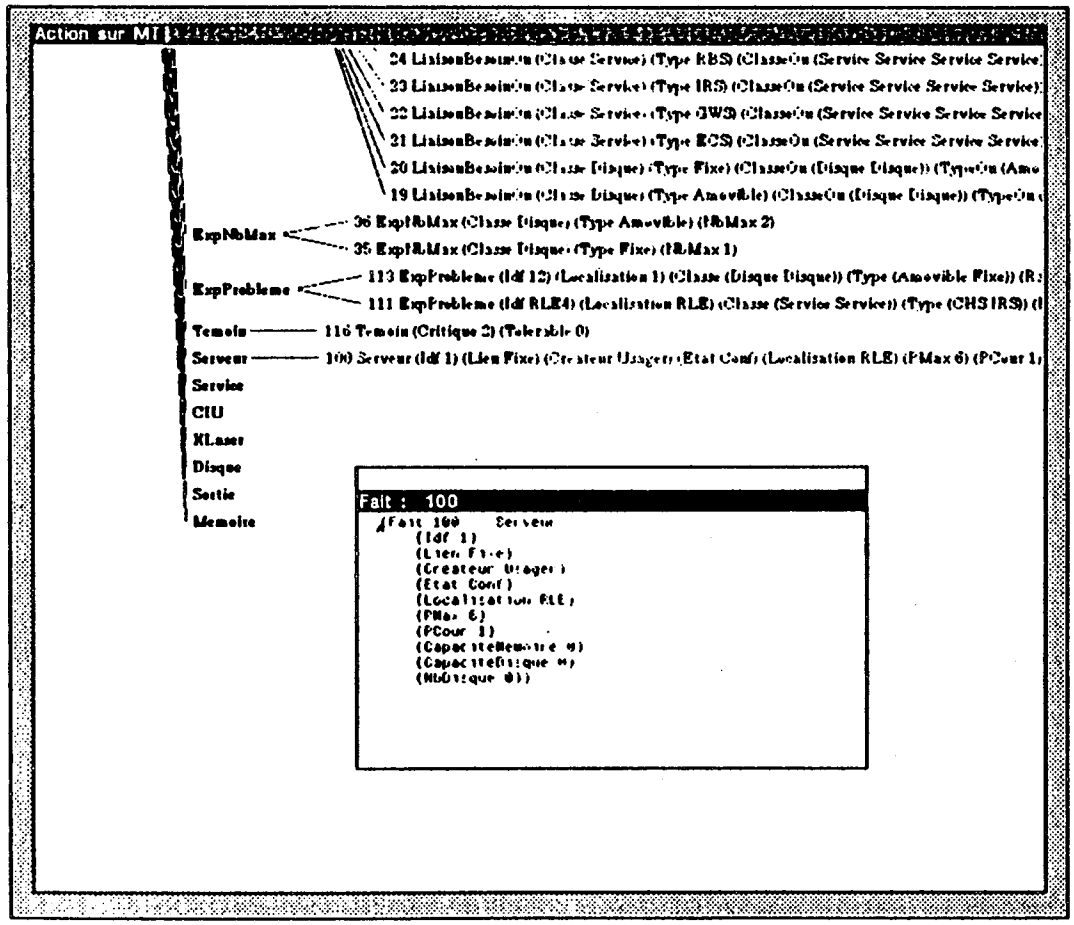


Figure 9.22 : La vue détaillée d'un élément de la mémoire de travail : ici, l'exemplaire de serveur qui se trouve être le fait d'estampille 100 de la mémoire de travail.

De même, une règle a plusieurs présentations : une version compacte pour les vérifications générales et, pour les besoins de l'édition, une forme détaillée. La figure 9.23 montre une version générale de la mémoire de productions et une vue précise sur une règle particulière.

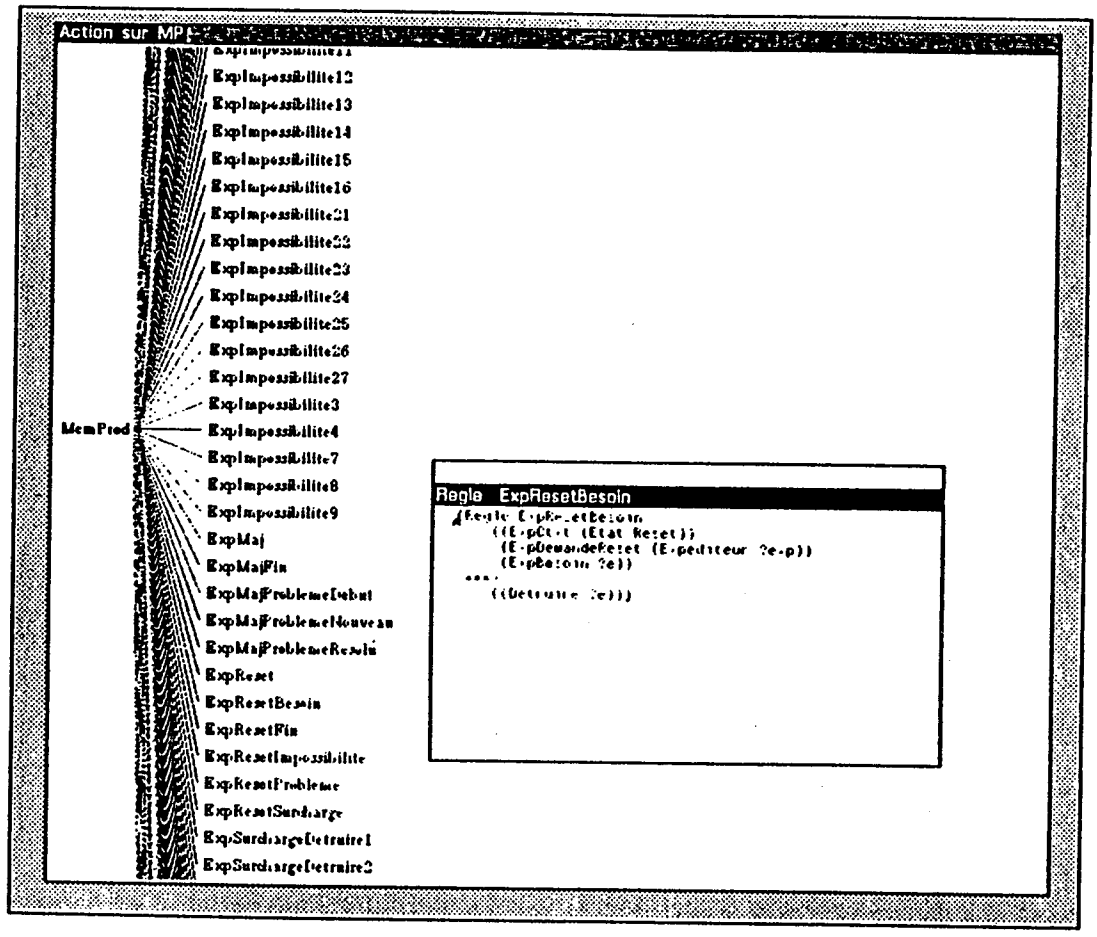


Figure 9.23 : Vues multiples simultanées du contenu de la base de règles : sous forme arborescente, l'ensemble des règles de la mémoire de productions et dans une fenêtre éditable, une règle particulière, ExpResetBesoin.

Jusqu'ici, nous avons considéré l'Image de sortie d'ELOISE. Pour les entrées, l'Image reprend la métaphore de la manipulation directe. Toutes les commandes ELOISE s'effectuent en désignant, avec la souris, les concepts visibles ou des éléments de menu. Lorsque l'utilisateur doit saisir un texte, le système propose un squelette du format attendu. Par exemple, dans la figure 9.24, nous observons les squelettes d'une règle et d'un élément de la mémoire de travail. Tous deux sont éditables sans contrainte structurelle à l'aide de l'éditeur de texte du poste de travail. Tous deux font suite à une demande de création, respectivement, d'une règle et d'un exemplaire de classe. Cette technique simple à mettre en œuvre permet de libérer l'utilisateur des détails syntaxiques sans le soumettre à la rigueur d'un éditeur de structures.

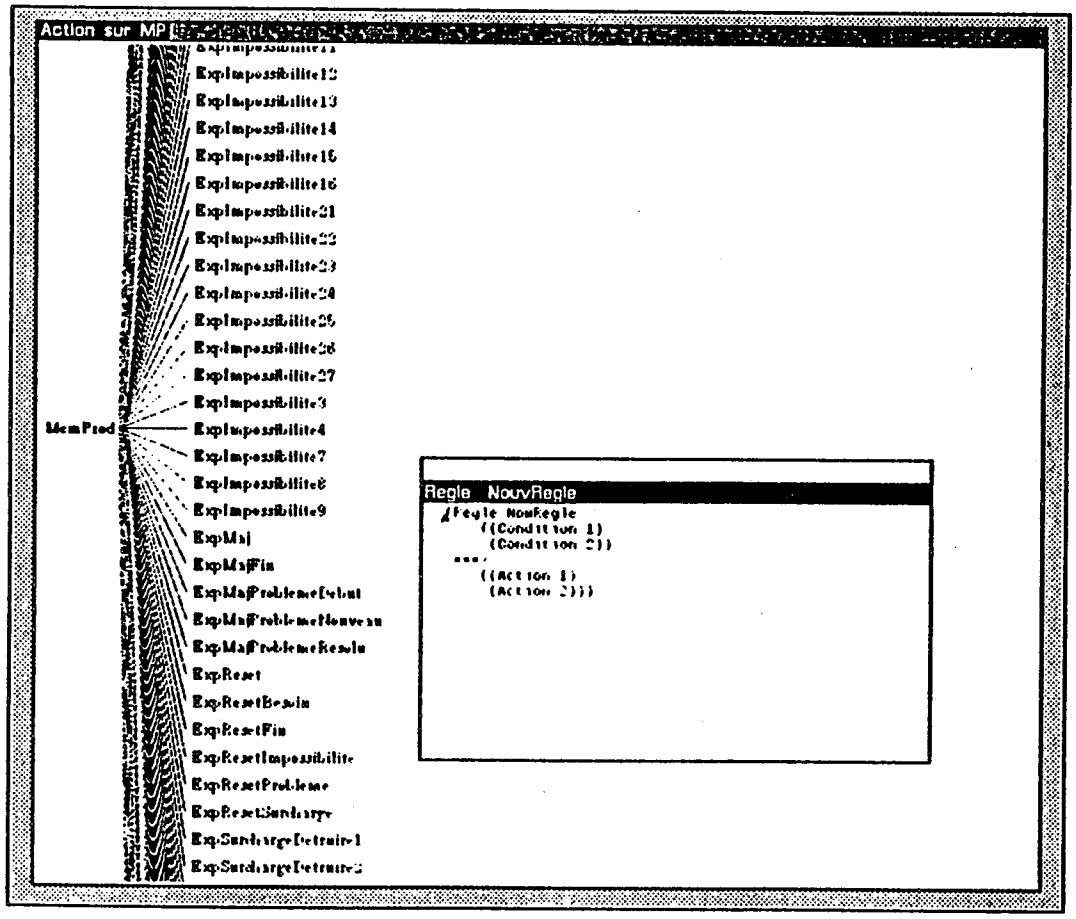


Figure 9.24 : Fenêtres de squelettes de règle et d'éléments de la mémoire de travail.
a)- L'opérateur vient d'appliquer l'opérateur Créer à l'entité **MemProd** qui représente la mémoire de productions. (La désignation de Memprod fait apparaître un menu fantôme des opérateurs applicables à cette entité.) Le squelette "Règle NouvRegle" apparaît prêt pour l'édition. Un menu fantôme attaché à la fenêtre du squelette permet à l'opérateur de confirmer la création ou au contraire de l'annuler.

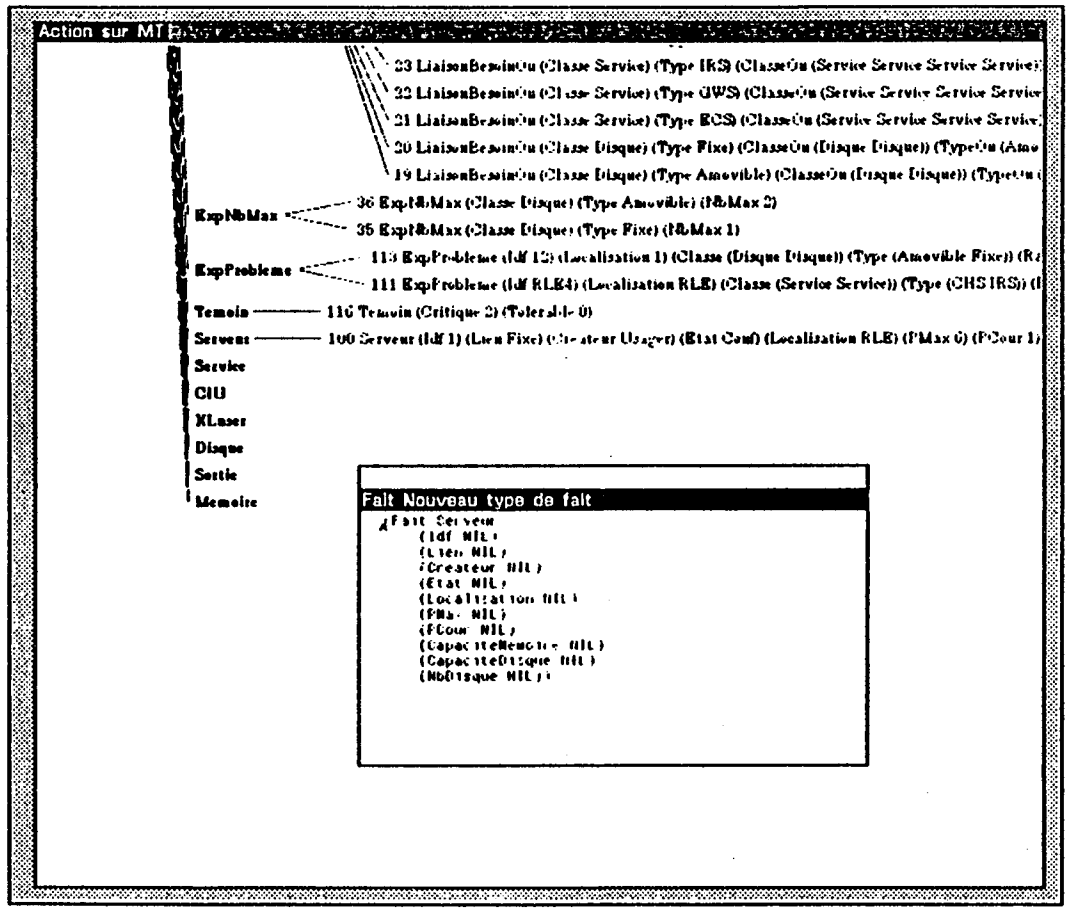


Figure 9.24 : Fenêtres de squelettes de règle et d'éléments de la mémoire de travail.
b)- L'opérateur vient d'appliquer l'opérateur Créer à l'entité **Serveur** de la fenêtre **Action sur MT**. (La désignation de **Serveur** fait apparaître un menu fantôme des opérateurs applicables à cette entité.) Un squelette décrivant un élément de la classe apparaît prêt pour l'édition. Un menu fantôme attaché à la fenêtre du squelette permet à l'opérateur de confirmer la création ou au contraire de l'annuler.

Les commandes ELOISE se répartissent en deux catégories : les actions sur les mémoires de travail et de règles et les requêtes d'exécution.

Les actions sur les mémoires s'effectuent par manipulation directe : l'utilisateur désigne un élément (par exemple, une règle de la fenêtre **Action sur MP** de la figure 9.24-a, ou un fait de la fenêtre **Action sur MT** de la figure 9.24-b). Un menu fantôme s'affiche à l'emplacement de l'élément désigné et propose la liste des actions possibles : création, destruction, modification, etc. Le choix d'une action se concrétise par l'affichage d'une fenêtre qui en illustre la sémantique. Par exemple, la sélection de l'action **modify** sur la règle **ExpResetBesoin** de la figure 9.24-a entraîne l'apparition de sa version détaillée éditable (c'est l'Image de la figure 9.23). Contrairement à de nombreux systèmes, l'apparition d'une fenêtre d'édition n'implique pas de la part de l'utilisateur des actions d'édition

immédiates. L'utilisateur d'ELOISE n'est jamais bloqué dans un sous-automate! A sa convenance, l'utilisateur indiquera au système la fin de l'édition ou son annulation grâce au menu qu'il trouve systématiquement attaché à toute fenêtre d'édition. Pour les besoins de la cohérence, lorsque la règle éditée est visible dans une fenêtre de la mémoire de règles, cette dernière est automatiquement rafraîchie.

Comme le montre la figure 9.25, les requêtes d'exécution sont toutes regroupées dans le menu de la fenêtre Eloise. Cette fenêtre est un "télétype-version écran" (glass-tty) qui reçoit les messages d'erreur du système et qui sert aussi à la saisie des commandes sous forme d'expressions Lisp. La disponibilité simultanée de la manipulation directe et de la saisie textuelle permet à la fois au programmeur débutant de contourner la barrière linguistique des langages de commande traditionnels et au programmeur expérimenté de conserver la puissance d'expression du langage Lisp. Parmi les requêtes d'exécution présentées dans la figure 9.25, deux méritent quelques commentaires supplémentaires : **Show Windows et Run.**

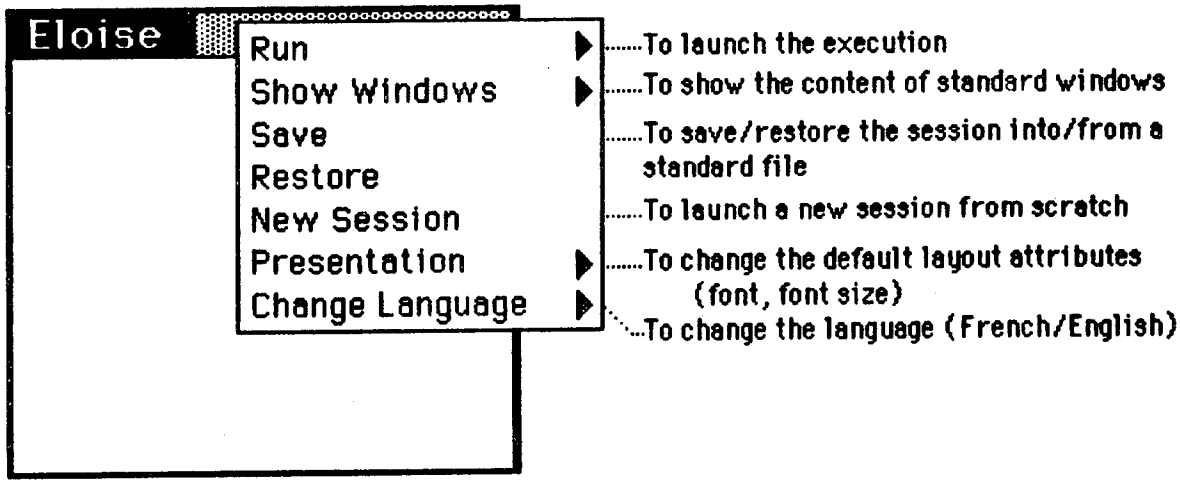


Figure 9.25 : Les requêtes d'exécution sont attachées à la fenêtre système ELOISE. Cette fenêtre est un "glass-tty" pour l'affichage des messages d'erreur et sert d'alternative textuelle à la saisie des commandes par manipulation directe.

Comme l'indique le petit triangle (notation usuelle sur les postes de travail Interlisp), **Show Windows** conduit à un sous-menu où sont regroupés les noms des fenêtres ELOISE standard : les fenêtres des mémoires de travail et de règles, et l'ensemble des règles activables. L'utilisateur peut les ouvrir, les fermer, les déplacer et les retailler à sa convenance. **Run** conduit également à un sous-menu permettant de choisir entre trois options d'exécution : exécution du programme dans sa totalité, éxécution d'un cycle ou bien exécution d'un nombre quelconque de cycles. Dans ce dernier cas, la valeur est saisie à l'aide d'une minicalculette héritée de l'environnement InterLisp. Tandis que le programme s'exécute, le programmeur peut observer en parallèle l'évolution des différentes mémoires.

Pour juger de la qualité des fonctions et de l'Image d'ELOISE, nous pouvons nous reporter à l'évaluation critique d'utilisateurs autres que les concepteurs et les réalisateurs du système. Ces utilisateurs sont quatre étudiants du DESS de Génie Informatique dont trois ont une connaissance littéraire du langage OPS5. Leur tâche est la réalisation du système expert IRENE qui comprendra finalement près de 500 règles et quelques classes pour les éléments de la mémoire de travail. Ces chiffres indiquent qu'il ne s'agit pas d'un simple exercice mais bien d'un véritable système qui peut rivaliser en importance avec nombre de systèmes experts déployés sur le marché.

Le comportement des utilisateurs d'ELOISE n'a pas fait l'objet d'une analyse "scientifique". Aucune source d'informations objectives comme la mesure chronométrée ou la technique de la caméra invisible, n'a été utilisée. Notre observation peut paraître naïvement informelle à l'ergonome averti, mais elle se résume simplement : aucun utilisateur n'a suggéré de modifications. Une anecdote tout aussi "non scientifique" mais tout aussi parlante est le refus catégorique de la part des étudiants d'utiliser l'environnement OPS5 à la place d'ELOISE!

4.3. L'architecture PAC du système

Alors que les fonctions et l'Image d'ELOISE sont loin d'être triviales, l'architecture, organisée selon le modèle PAC, est remarquablement simple.

L'Abstraction du niveau le plus haut est le noyau du système qui comprend la mécanique nécessaire à la mise en œuvre d'un système de productions. Contrairement au principe général des systèmes de productions, l'algorithme d'unification d'ELOISE ne travaille pas sur l'ensemble des règles de la mémoire de règles, mais procède de manière incrémentale sur les variations de la mémoire de travail : toute modification de la mémoire de travail détermine une variation de l'ensemble des règles activables. Cette technique d'unification est connue sous le nom d'algorithme de Rete [Forgy 82]. L'originalité d'ELOISE tient à l'élégance de la réalisation du réseau d'unification Rete sous forme d'une hiérarchie d'objets Loops communicants.

Le Contrôle du niveau le plus haut est un objet Loops de nom **ELOISE** qui sert de porte entre le noyau et la Présentation. **ELOISE** reçoit de la Présentation des messages dont le format convient au noyau. Par exemple, dans le cas d'une création de règle, la Présentation émet un message Loops de la forme :

(<-$ELOISE **CreateRule** *NomdeRegle Conditions Actions*)

où <-$ELOISE désigne le destinataire (c'est-à-dire **ELOISE**), **CreateRule** spécifie le sélecteur de message, et le reste décrit les éléments syntaxiques intervenant dans la définition d'une règle.

La Présentation est une organisation d'objets hérités des environnements LOOPS et InterLisp tels que les fenêtres, les menus, l'éditeur de texte, l'afficheur d'arbres et la calculette, etc. Afin de conserver le "style Xerox", nous avons évité les surcharges de comportement. Seules les actions de l'utilisateur à effet de bord sémantique sont dirigées vers ELOISE sous la forme déjà indiquée.

La richesse des environnements Loops et Interlisp a rendu possible la réalisation de la Présentation en deux mois par une seule personne. La difficulté essentielle a été la conception de l'Image et, pour les aspects réalisation, la détermination des services réutilisables de l'environnement.

5. Le système expert IRENE

IRENE (Inference Rules for EtherNet Expertise) est un système d'aide à la configuration de réseaux locaux XNS (Xerox Network System) réalisé dans l'environnement ELOISE. Comme pour ELOISE, la conception s'est appuyée sur une analyse de tâche approfondie effectuée avec le concours d'un futur utilisateur, expert dans le domaine. Cette partie reprend les thèmes du chapitre avec la description des objectifs du système, de son Image et de son architecture. Elle présente ensuite l'apport du modèle PAC à la résolution de quelques problèmes intéressants : analyse syntaxique à plusieurs niveaux d'abstractions, création dynamique de concepts, génération de messages d'erreur sémantique et délégation de pouvoirs sémantiques dans l'interface.

5.1. L'objet du système

Un réseau local XNS (Xerox Network System) ne se construit pas au hasard des besoins du client. Les composants matériels et les services logiciels d'un réseau sont au contraire agencés selon des règles précises. Ces règles, consignées dans d'énormes répertoires, constituent l'unique apport à la conception. Le concepteur procède par essais successifs, élaborant les agencements possibles à l'aide de schémas esquissés sur des feuilles de papier. Au delà de moyens quelque peu surannés, les solutions envisagées, qui échappent à toute forme de vérification automatique, ne sont pas à l'abri d'erreurs de conception.

L'objectif d'IRENE est d'informatiser l'activité de configuration de réseaux XNS en mettant à la disposition du concepteur le savoir-faire du domaine. L'expertise d'un système tel qu'IRENE peut se manifester sous deux formes selon les besoins stratégiques du concepteur : vérifier la validité d'une proposition ou suggérer des solutions.

- l'utilisateur, compétent dans le domaine, souhaite élaborer une solution comme il le pratique dans le monde réel. Le rôle du système est alors de vérifier que la proposition est conforme aux règles de fonctionnement ;
- l'utilisateur, incertain quant à la solution à donner au problème, se contente d'indiquer les composants obligatoires de la configuration. Le système doit alors proposer une organisation valide qui inclut les constituants spécifiés.

La figure 9.26 résume les deux rôles essentiels du système IRENE en tant que collaborateur du concepteur. A ces deux fonctions de base, s'ajoute la gestion du travail élaboré en commun : sauvegardes et reprises, versions multiples d'une solution et impression sur papier.

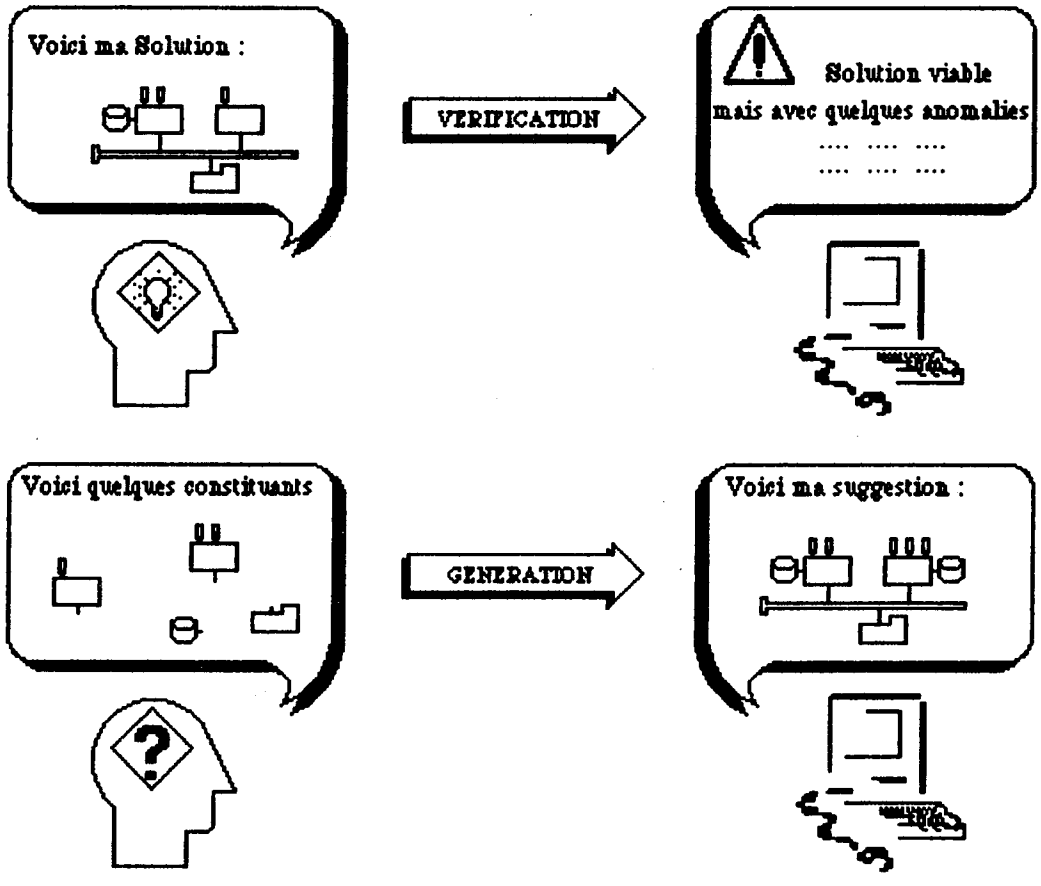


Figure 9.26 : Le système IRENE se comporte comme un collaborateur : il vérifie la validité des propositions de l'opérateur et lui suggère des solutions.

5.2. Les fonctions du système et son Image

La figure 9.27 donne un aperçu de l'Image du système IRENE. Cette Image a deux objectifs : reproduire au mieux le monde familier du concepteur et présenter son savoir-faire sous une forme utile.

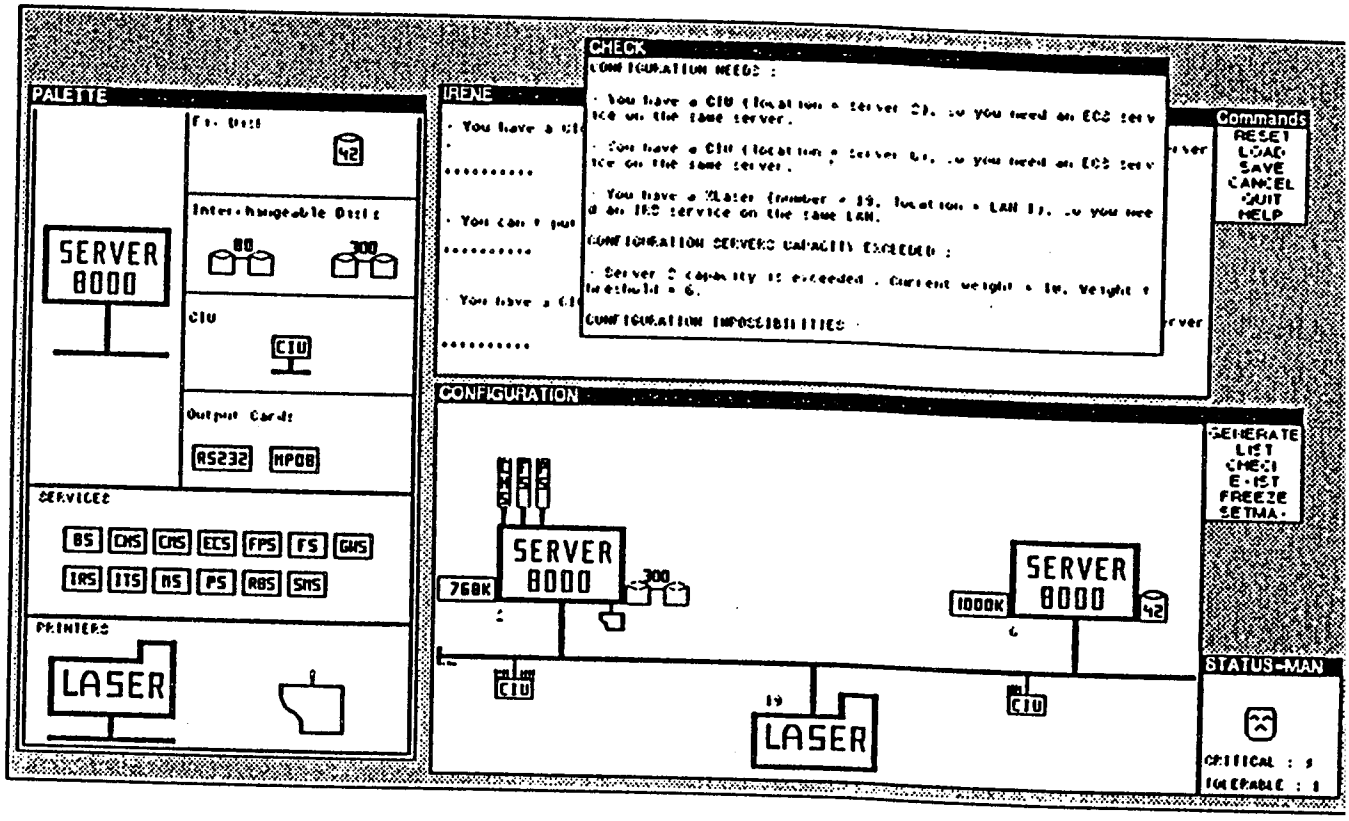


Figure 9.27 : Un aperçu de l'Image du système IRENE.

5.2.1. Reproduction des procédures du monde réel

Puisqu'il s'agit d'informatiser une activité du monde réel, l'Image doit en respecter les éléments marquants. Ces éléments, nous les avons identifiés au cours d'entretiens avec un praticien du domaine. Nous avons en particulier relevé le rôle central des schémas dans l'élaboration d'une solution. A la suite de ces observations, nous avons organisé l'Image du système autour d'une fenêtre dite de **CONFIGURATION**, feuille de brouillon commune à l'utilisateur et au système sur laquelle est restituée une représentation graphique de la solution. Les éléments graphiques sont les reproductions des symboles utilisés dans le monde physique.

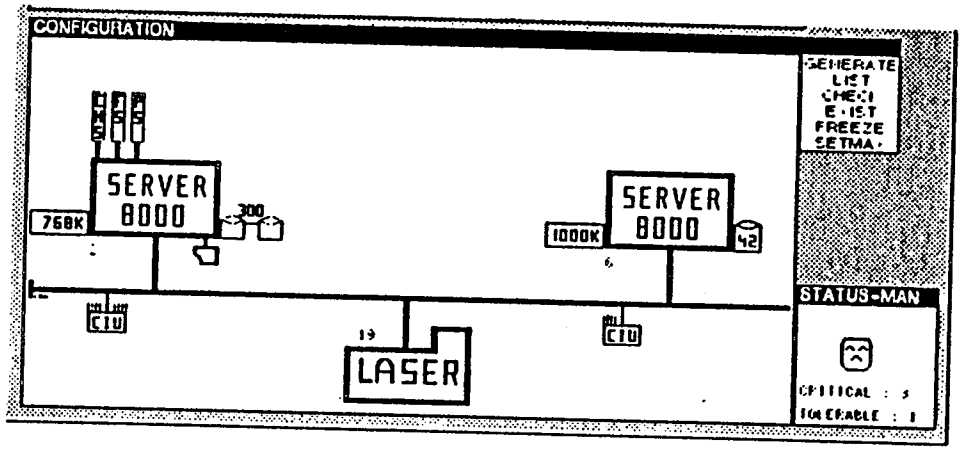


Figure 9.28 : la fenêtre de configuration présente la solution actuelle.

La configuration actuelle présentée dans la figure 9.28 comprend deux unités centrales (désignés sous le terme de **server 8000**). Le serveur de gauche est doté d'une carte mémoire de 768 K octets et d'un disque amovible de 300 méga octets. Ce serveur est le lieu d'exécution de deux services logiciels: **CHS** (Clearing House Service) et **FS** (File Service). Les sigles semblent ésotériques. En réalité, ils respectent rigoureusement la terminologie de l'expert humain.

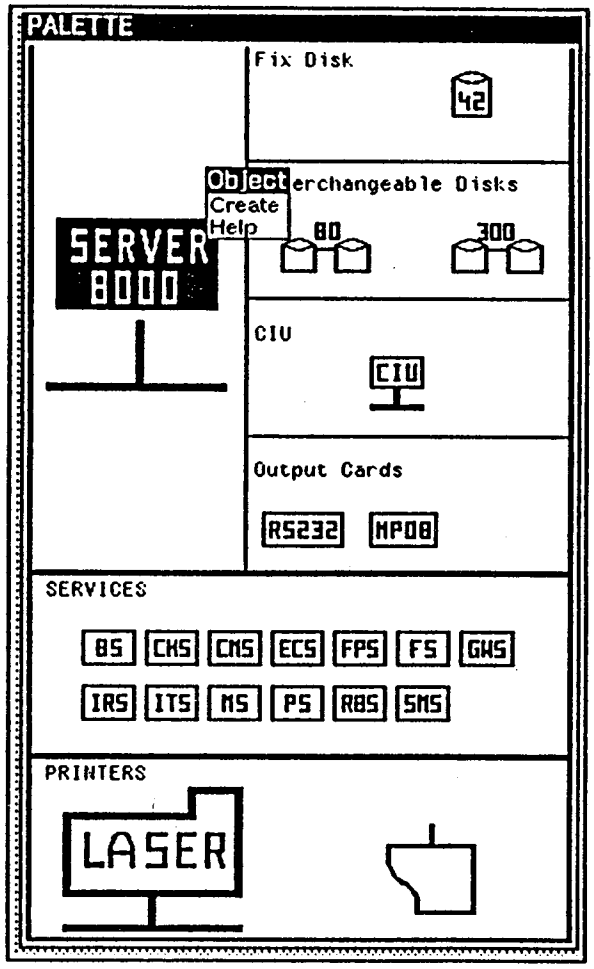


Figure 9.29 : Les concepts du domaine sont regroupés dans une palette et les opérateurs mentaux applicables aux concepts ont leur équivalent dans l'Image sous forme de menus fantômes.

Dans le monde réel, le concepteur décide de la création d'un serveur et de l'installation d'accessoires matériels et logiciels. Ces décisions se concrétisent par l'ajout, le retrait, le déplacement de symboles graphiques sur la feuille de papier. Le monde électronique d'IRENE reproduit cette méthode de travail. Les symboles graphiques sont réunis dans une **PALETTE** qui concrétise les classes des concepts du domaine. Les manipulations mentales que le concepteur applique aux classes se retrouvent sous la forme de menus fantômes liés aux symboles graphiques. Un menu apparaît dès que l'utilisateur désigne un symbole de la palette. Par exemple, dans la figure 9.29, l'opérateur vient de désigner le symbole qui représente le concept de serveur. Un menu apparaît près de l'emplacement du symbole proposant une liste des opérations possibles : création, aide.

Le résultat d'une décision mentale sur un exemplaire de concept, par exemple la suppressio d'un serveur, se traduit dans le monde réel par une modification du dessin. IRENE reproduit l procédure physique : toute opération mentale conduisant à la modification du dessin a comm équivalent électronique la manipulation directe d'un symbole de la fenêtre de configuration. Pa exemple, la suppression du serveur s'effectue en désignant le symbole qui le représente dans l fenêtre de configuration. Un menu fantôme apparaît et l'utilisateur sélectionne l'opérateur d destruction. De même, le déplacement d'un service vers un autre serveur s'effectue par manipulatio directe. La figure 9.30 illustre la succession des états graphiques pour une opération de transfert.

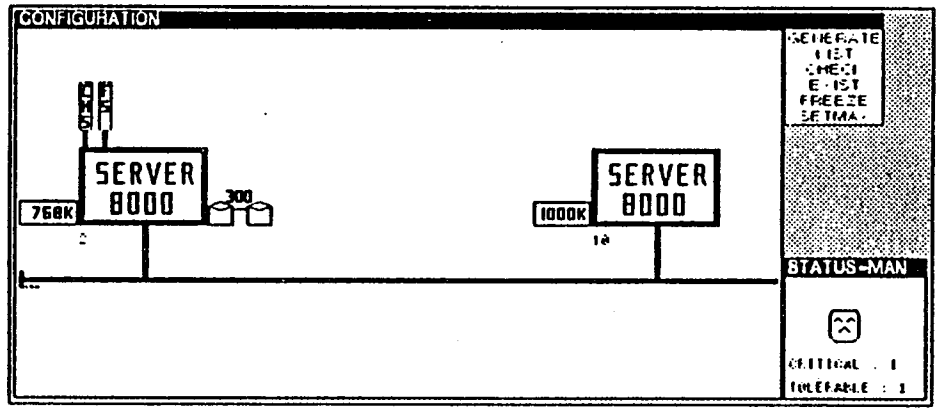


Figure 9.30 : Le transfert d'un service logiciel s'effectue par manipulation directe.
a) Vue initiale de la fenêtre de configuration. L'utilisateur décide de transférer le service FS sur l'autre serveur.

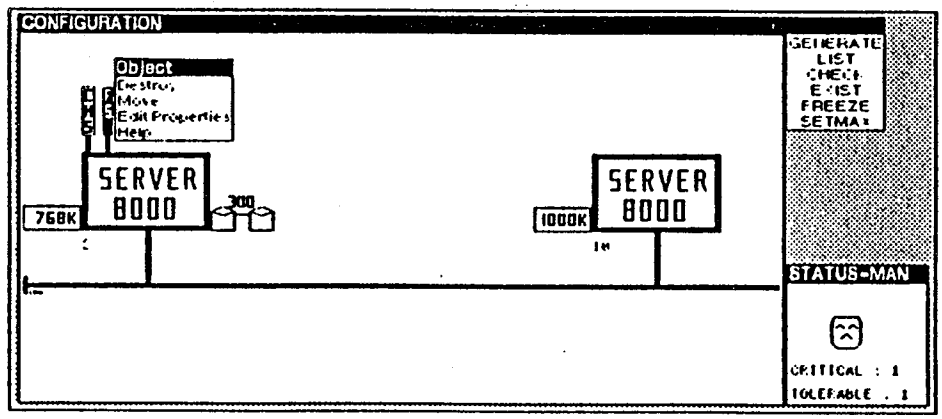


Figure 9.30 : Le transfert d'un service logiciel s'effectue par manipulation directe.
b) L'utilisateur vient de désigner le symbole du service FS : ce dernier apparaît en vidéo inverse et le menu des actions applicables à ce service apparaît.

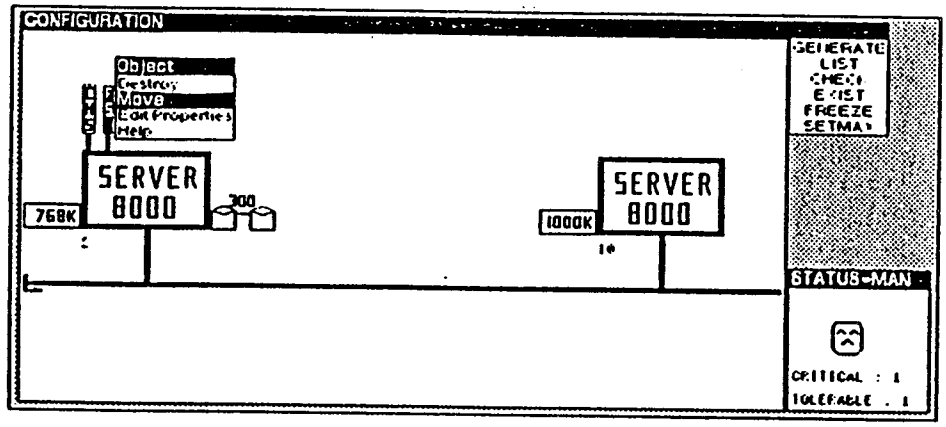


Figure 9.30 : Le transfert d'un service logiciel s'effectue par manipulation directe.
c) L'utilisateur place la souris sur l'opérateur "move" du menu.

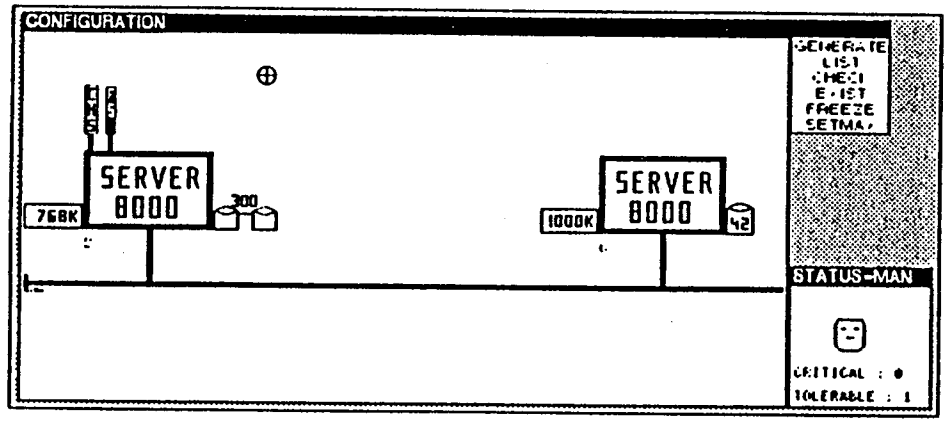


Figure 9.30 : Le transfert d'un service logiciel s'effectue par manipulation directe.
d) L'utilisateur vient de sélectionner l'opérateur "move". Le suiveur prend la forme d'une mire pour exprimer qu'une désignation de lieu est attendue. Le symbole graphique de FS est toujours en vidéo inverse pour montrer qu'il est encore impliqué dans l'action.

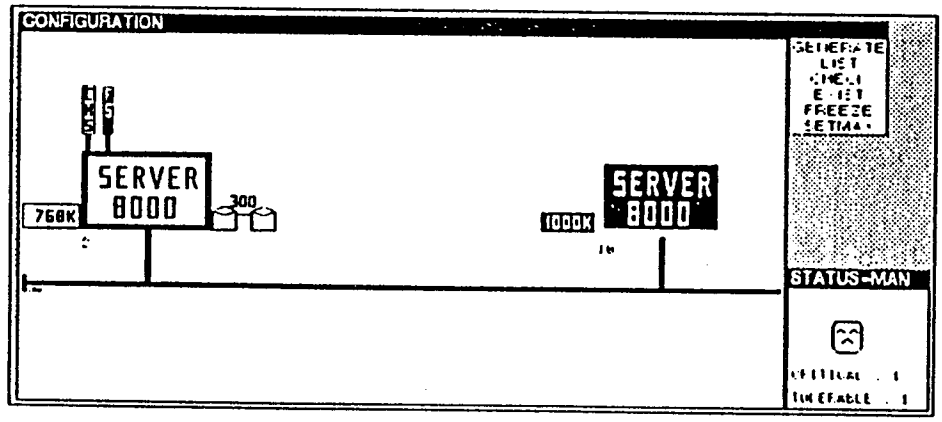


Figure 9.30 : Le transfert d'un service logiciel s'effectue par manipulation directe.
e) L'utilisateur désigne le serveur destinataire. Son symbole graphique répond à l'action par une vidéo inverse.

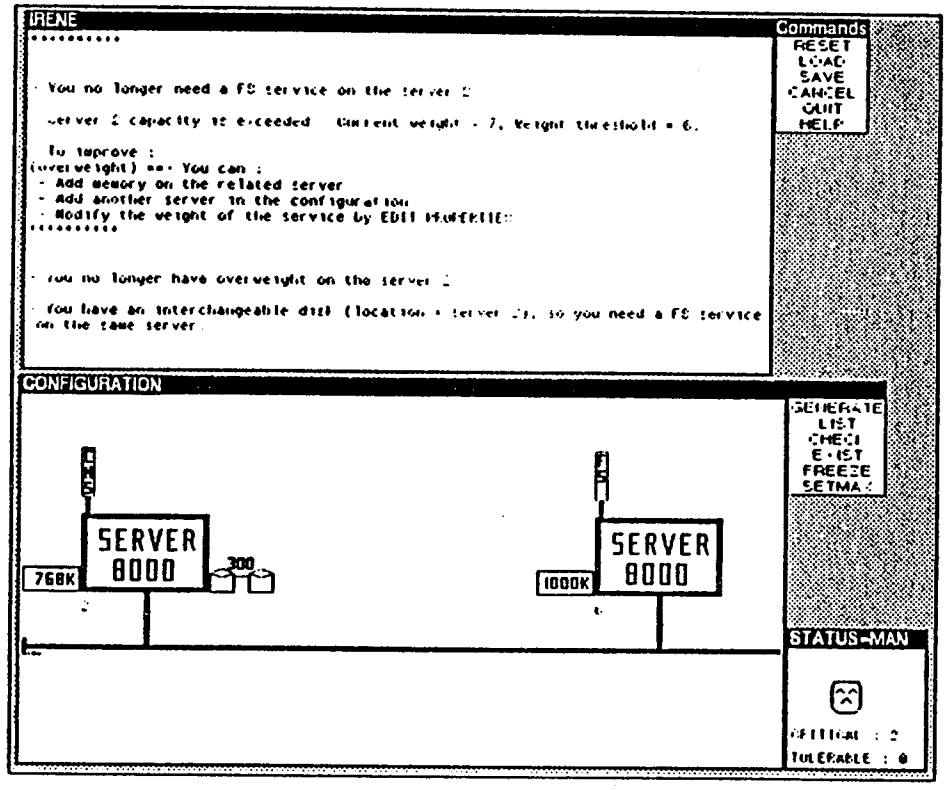


Figure 9.30 : Le transfert d'un service logiciel s'effectue par manipulation directe.
f) L'expertise accepte le transfert. Le symbole FS apparaît maintenant sur le niveau site.

Noter que toute action élémentaire de l'utilisateur s'accompagne d'un retour d'information immédiat : la désignation du symbole à transférer provoque son affichage en vidéo inverse et l'apparition d'un menu fantôme ; à la sélection de l'opérateur move, le curseur prend la forme d'une mire pour indiquer qu'une destination est attendue. (Nous aurions souhaité un retour d'information plus expressif tel que l'asservissement du symbole aux mouvements de la souris. Les outils graphiques de l'environnement, contrairement à ceux du Macintosh, ne nous ont pas permis un tel effet.) La spécification de la destination affiche le symbole récepteur en vidéo inverse. Cette succession d'états graphiques correspond à un automate local chargé de la saisie des informations nécessaires à l'expertise. Afin d'éviter les blocages dans des sous-dialogues, l'annulation est à tout instant possible.

Jusqu'ici, nous avons décrit la reconstitution par l'Image du monde familier au concepteur. Nous allons maintenant analyser la contribution d'IRENE à la tâche de conception. L'expertise agit à titre de vérificateur ou de générateur de solutions. Le comportement entre les deux attitudes est laissé au choix de l'utilisateur.

5.2.2. Aide à la conception : vérification

Une vérification experte est effectuée dès qu'une manipulation de l'Image s'accompagne d'un effet de bord sémantique. Le résultat s'exprime systématiquement par une série de messages explicatifs consultables dans la fenêtre système **IRENE**. La figure 9.31 montre le contenu de cette fenêtre avant et après le transfert du service FS de l'exemple précédent. Nous observons qu'IRENE signale les problèmes résolus par l'intervention de l'opérateur, qu'il identifie la cause des nouvelles anomalies et suggère les remèdes mais qu'il ne prend pas l'initiative de la correction : IRENE est un collaborateur qui n'impose jamais de solution.

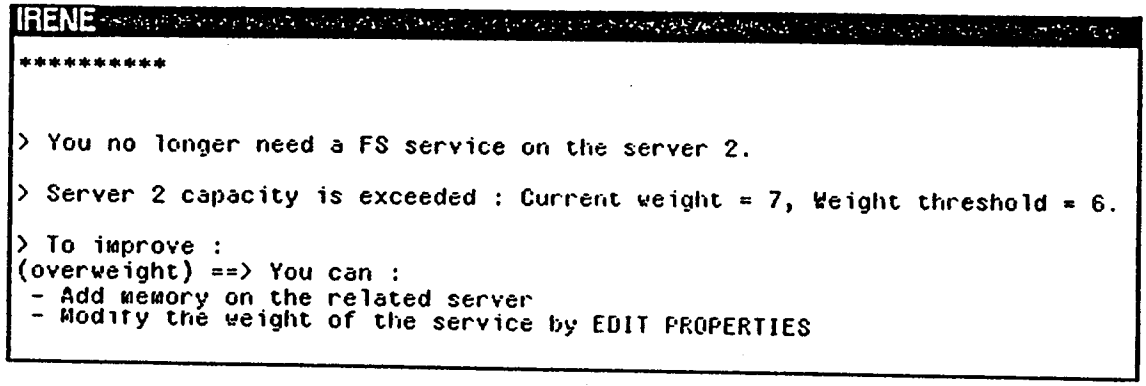


Figure 9.31 : La fenêtre des messages IRENE informe le concepteur à chaque manipulation sémantique : identification des problèmes résolus, indication des nouvelles difficultés et proposition de solution.
a) Avant le transfert de FS : le premier message indique qu'un problème vient d'être résolu (le serveur 2 nécessitait la présence d'un service FS). Le second message signale une nouvelle difficulté (surcharge du serveur 2) et le dernier message propose plusieurs façons d'améliorer la situation (notamment d'augmenter la capacité mémoire).

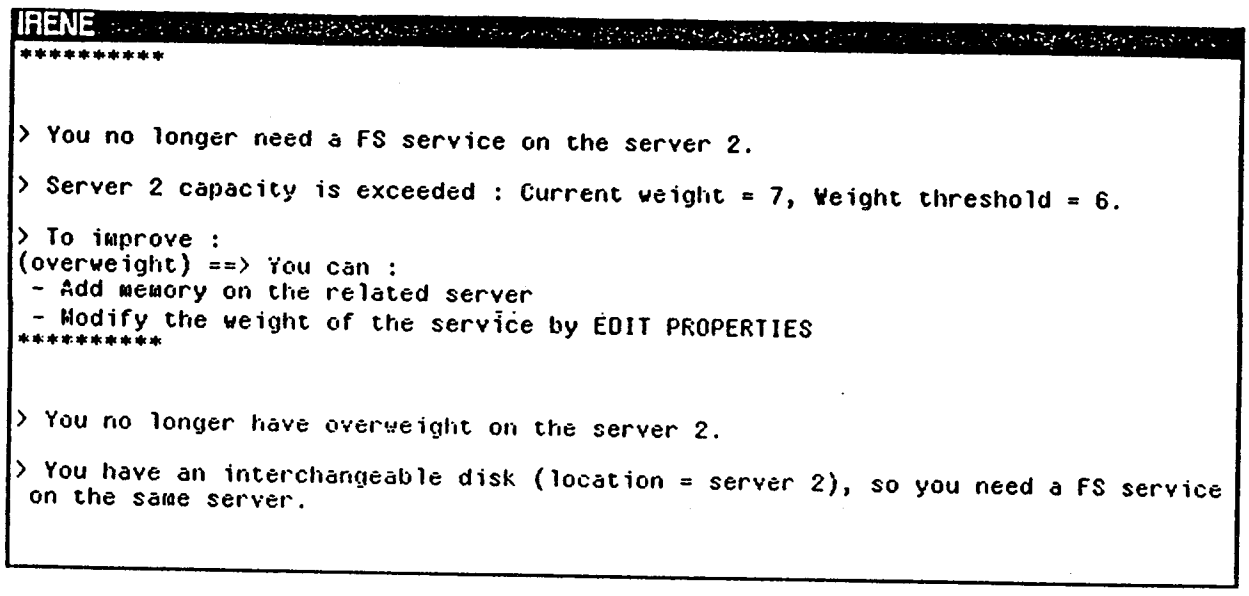


Figure 9.31 : La fenêtre des messages IRENE informe le concepteur à chaque manipulation sémantique : identification des problèmes résolus, indication des nouvelles difficultés et proposition de solution.
b) Après le transfert de FS : avec un service de moins, le serveur 2 n'est plus en état de surcharge mais la présence d'un disque amovible exige celle d'un FS (File service). Cette Image est une vue grossie de la fenêtre IRENE de la figure 9.30-f.

Les anomalies sont mémorisées par le système : le concepteur n'est pas contraint de le supprimer sur le champ et n'a pas non plus la charge mentale de s'en souvenir. Dans ce but, l système fournit trois services complémentaires : les messages signalant la disparition des anomalie le STATUS-MAN et la commande CHECK :

- Les messages de signalement de disparition d'anomalie sont importants à deux titres : il confirment au concepteur le bien fondé de ses actes et lui rappellent l'existence d'erreur oubliées. Ces messages sont présentés, nous l'avons vu, dans la fenêtre IRENE.

- Le STATUS-MAN est un indicateur général de la validité de la solution actuelle. La figure 9.3 en montre les trois états possibles : le visage est souriant si la solution est sémantiquemen correcte ; il est neutre lorsque la configuration peut fonctionner sans toutefois se présenter dan les meilleures conditions : un indicateur précise alors le nombre d'anomalies tolérables ; l visage est triste lorsque la configuration est invendable telle quelle : un indicateur précise l nombre d'erreurs critiques. Cette représentation synthétique permet une évaluation globale simple et rapide de l'état de la configuration.

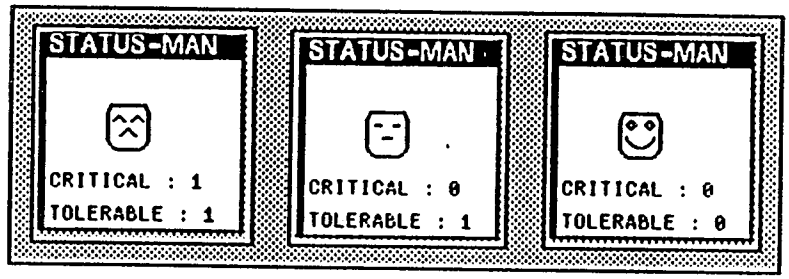


Figure 9.32 : Les trois présentations du STATUS-MAN permettent une évaluation globale simple et rapide de l'état de la configuration.

- La commande CHECK fournit au contraire des renseignements précis sur la situation. Elle est activée en sélectionnant l'élément CHECK du menu de la fenêtre de configuration. Elle fournit dans la fenêtre CHECK la liste des anomalies classées par gravité mais elle donne aussi des enseignements comme en témoigne l'exemple de la figure 9.33.

```
CHECK
CONFIGURATION NEEDS :

> You have a server (number = 6, location = LAN 1), so you need
 an interchangeable disk or a fix disk on the same server.

CONFIGURATION SERVERS CAPACITY EXCEEDED :

> Server 2 capacity is exceeded : Current weight = 7, Weight th
reshold = 6.

CONFIGURATION IMPOSSIBILITIES :

> You won't be able to have a MPOB with the interchangeable dis
k (location = server 2) on the same server.

> You won't be able to have a fix disk with the interchangeable
 disk (location = server 2) on the same server.
```

Figure 9.33 : Le résultat de la commande CHECK lancée avant le transfert du service FS. Par ordre de gravité : les besoins, les dépassements de capacité et les enseignements : le serveur 6 doit être équipé de mémoire secondaire ; le serveur 2 est en état de surcharge ; la présence d'un disque amovible sur le server 2 exclut l'installation d'un MPOB ou d'un disque fixe.

Après avoir décrit IRENE comme outil de vérification, nous abordons ses fonctions de génération automatique.

5.2.3. Aide à la conception : génération automatique

La génération automatique est souhaitable lorsque l'opérateur est un néophyte dans le domaine ou bien lorsqu'il lance une nouvelle étude à partir de quelques constituants connus. Comme pour la fenêtre de configuration, il suffit à l'opérateur de placer les constituants dans la fenêtre **CONFIGURATION AUTOMATIQUE**. La figure 9.34 montre un exemple. En haut, la fenêtre **CONFIGURATION** contient la solution actuelle partiellement élaborée. En bas, dans la fenêtre de configuration automatique, l'opérateur vient de placer quelques éléments simples mais il y a aussi construit un serveur doté de périphériques. La construction d'un élément composé comme ce serveur est toujours soumise à l'avis de l'expertise quelle que soit la fenêtre de travail. A la sélection de l'élément **GENERATE** du menu permanent de la fenêtre **CONFIGURATION**, l'expertise se met en quête de produire une solution qui intègre dans la configuration actuelle les éléments de la fenêtre de configuration automatique.

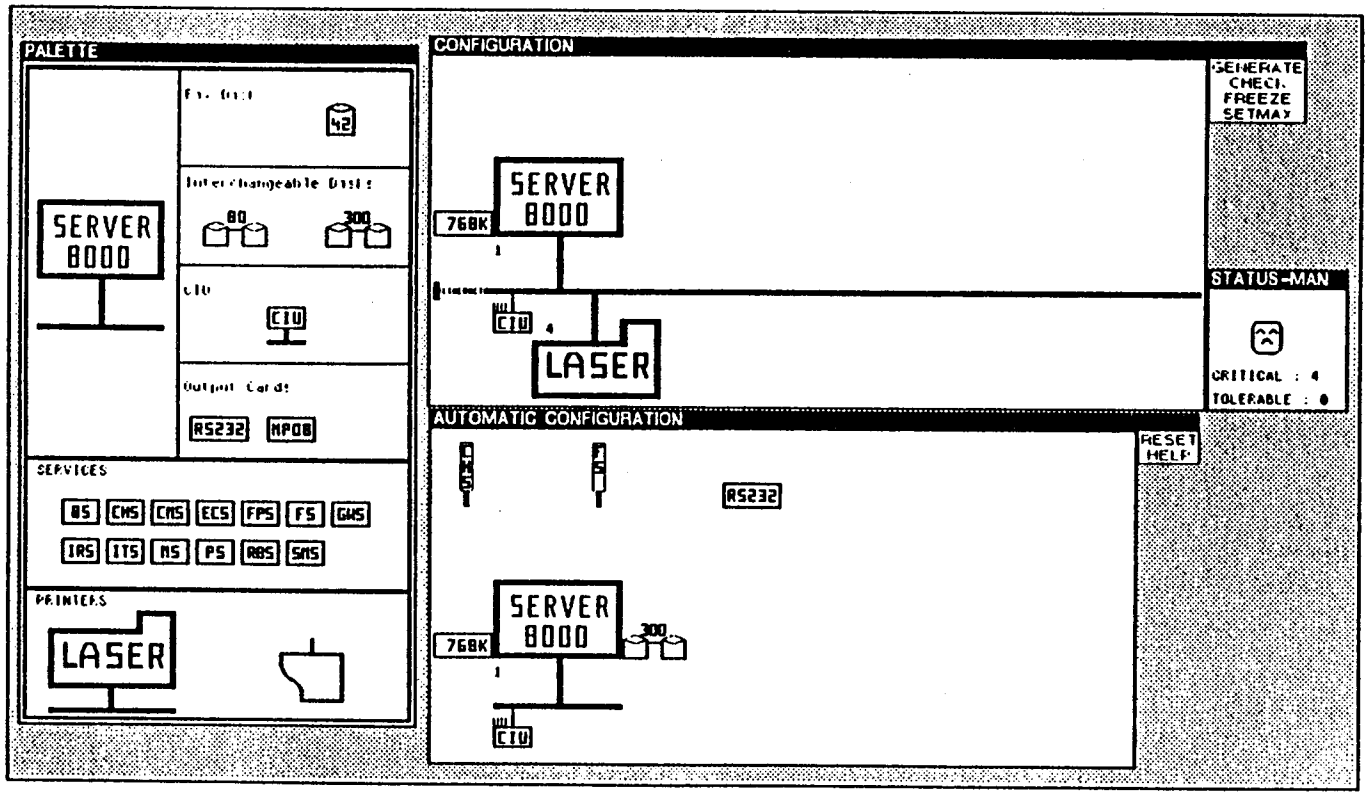


Figure 9.34 : Le contenu de la fenêtre **AUTOMATIC CONFIGURATION** est un ensemble partiellement organisé de constituants. L'Image du système avant l'application de l'opérateur **GENERATE** (menu permanent de la fenêtre CONFIGURATION).

La solution trouvée par la génération automatique relève d'une heuristique : elle satisfait les contraintes sémantiques du domaine mais elle ne garantit pas d'être optimale. La figure 9.35 montre la solution proposée dans la fenêtre de configuration. La fenêtre de configuration automatique est vide : tous les constituants demandés ont pu être placés. L'opérateur peut maintenant affiner la proposition et poursuivre l'édition selon la technique usuelle.

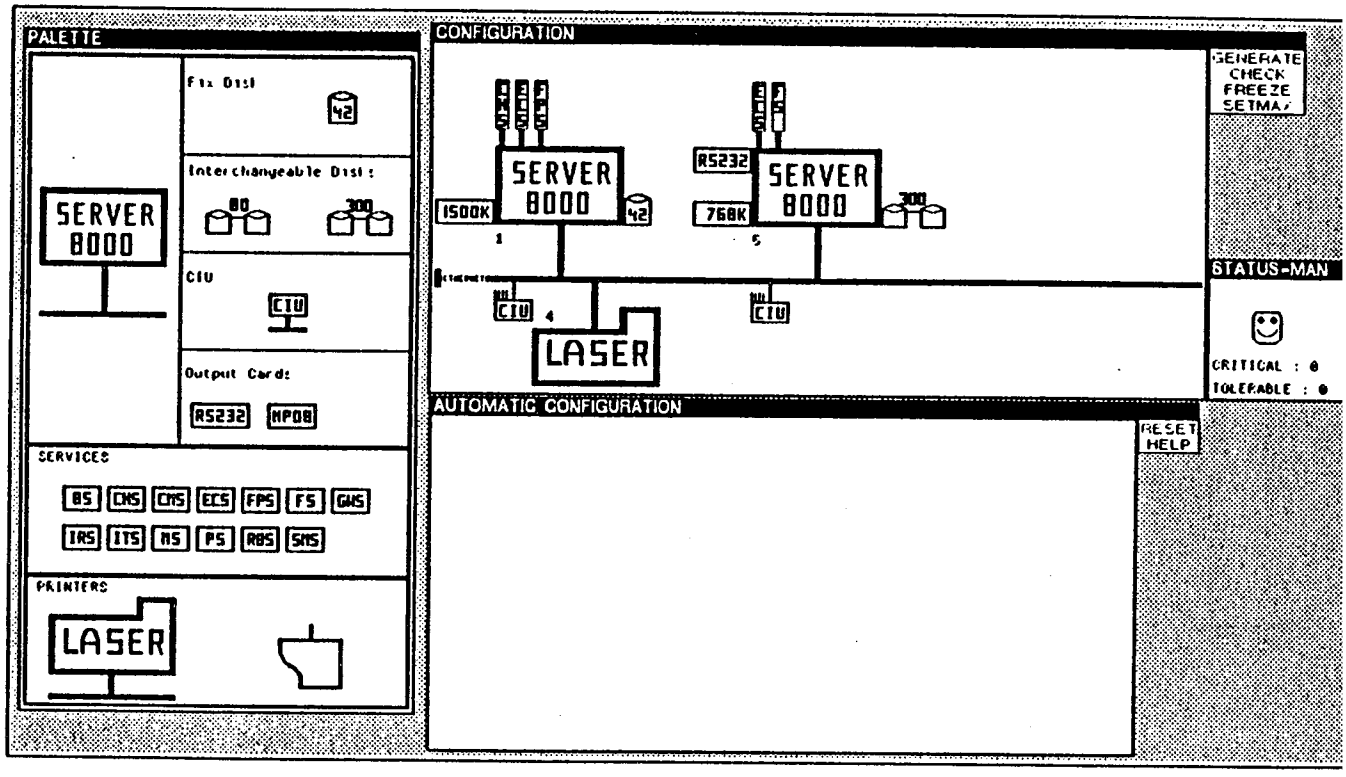


Figure 9.35 : Image après la génération automatique : la solution proposée combine les constituants de la fenêtre de configuration automatique à ceux de l'ancienne configuration.

A la conception d'IRENE, nous nous sommes inspirés de l'image métaphorique de l'entonnoir pour désigner la fonction de la fenêtre de configuration automatique : un entonnoir est bien à la fois un réceptacle d'information désorganisée et un producteur de flot contrôlé. Nous aurions souhaité reproduire cet effet à l'écran. Les techniques graphiques de l'environnement de réalisation d'IRENE nous ont contraints à une présentation moins parlante. NeWS, qui permet de créer des surfaces d'affichage quelconques, nous aurait donné les moyens de produire une Image comme celle de la figure 9.36 mieux adaptée à la situation.

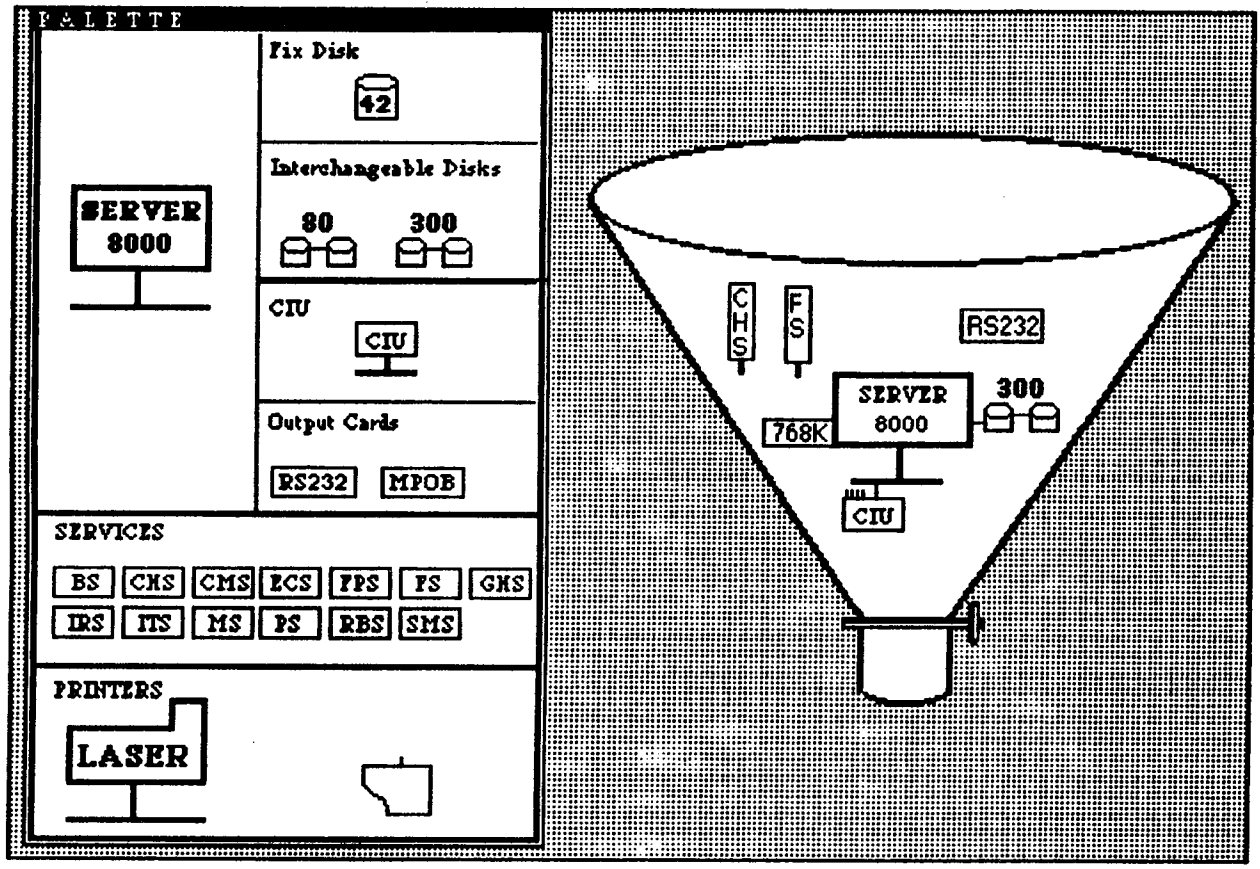


Figure 9.36 : Image souhaitée mais non réalisée de la fenêtre de configuration automatique sous forme d'un **ENTONNOIR**. La génération automatique aurait été activée par l'ouverture du robinet.

5.3. L'architecture PAC du système

L'architecture logicielle d'IRENE reprend fidèlement le modèle PAC. La figure 9.37 en indique les éléments essentiels. L'Abstraction du niveau le plus haut réalise le savoir-faire dans le domaine de la configuration des réseaux XNS. La description de l'organisation interne de ce composant ne relève pas de nos propos. Il nous faut cependant préciser quelques points éclairants sur le savoir-faire et la représentation de la configuration :

- le savoir-faire est modélisé sous forme de règles ELOISE. Les unes expriment les lois du bon fonctionnement des répertoires officiels. Par exemple, elles traduisent des contraintes de la forme : "S'il existe un disque sur un serveur, alors il faut un service de gestion de fichiers" ; d'autres règles modélisent l'heuristique personnelle de l'expert qui a participé à la conception. Celles-ci traduisent des contraintes de la forme : "Si la somme des poids des services installés sur un serveur dépasse la capacité du serveur, alors le serveur fonctionnera en mode dégradé". (Le poids est une notion personnelle à notre expert qui synthétise le coût d'un service en tenant compte de sa fréquence d'utilisation et de sa taille.) D'autres règles servent au contrôle du parcours de graphe nécessaire à la détermination d'une solution pour la génération automatique. La recherche n'est pas exhaustive (le domaine d'application ne l'exige pas) et s'effectue par profondeur (sans limite sur la profondeur).

- la configuration est modélisée par des éléments de la mémoire de travail. Certains représentent les constituants de la configuration, d'autres jouent le rôle de variables de contrôle. Tout

constituant (présenté à l'écran sous forme d'un symbole graphique) est modélisé dan l'Abstraction par un élément de la mémoire de travail. Les liens entre ces éléments exprimen l'organisation actuelle de la configuration. Les éléments de contrôle ne modélisent pas l connaissance experte mais servent à marquer les étapes intermédiaires du raisonnement et mémoriser des observations, telles les anomalies, utiles à l'utilisateur. Ce dernier point es important : on évoque en permanence la séparation entre application et interface ; cett séparation marque une frontière entre formalismes mais elle n'est pas un gouffre entr fonctionnalités. Toutes les fonctions du système coopèrent pour le bien de l'utilisateur. E conséquence, s'il est vrai que certaines fonctions et structures de données de l'application son conçues pour modéliser au mieux un problème donné, il est tout aussi nécessaire que d'autre soient strictement dirigées vers l'utilisateur c.-à-d. vers l'interface. En l'occurrence, l modélisation des anomalies est une nécessité à la fois pour la résolution du problème et pou l'utilisateur. En conséquence, tout phénomène concernant une anomalie doit être communiqué l'interface. Sans cela, l'utilisateur ne poura être averti convenablement. De toute évidence, l difficulté première, au moment de la conception de l'Abstraction, est la détermination de informations abstraites nécessaires à l'utilisateur.

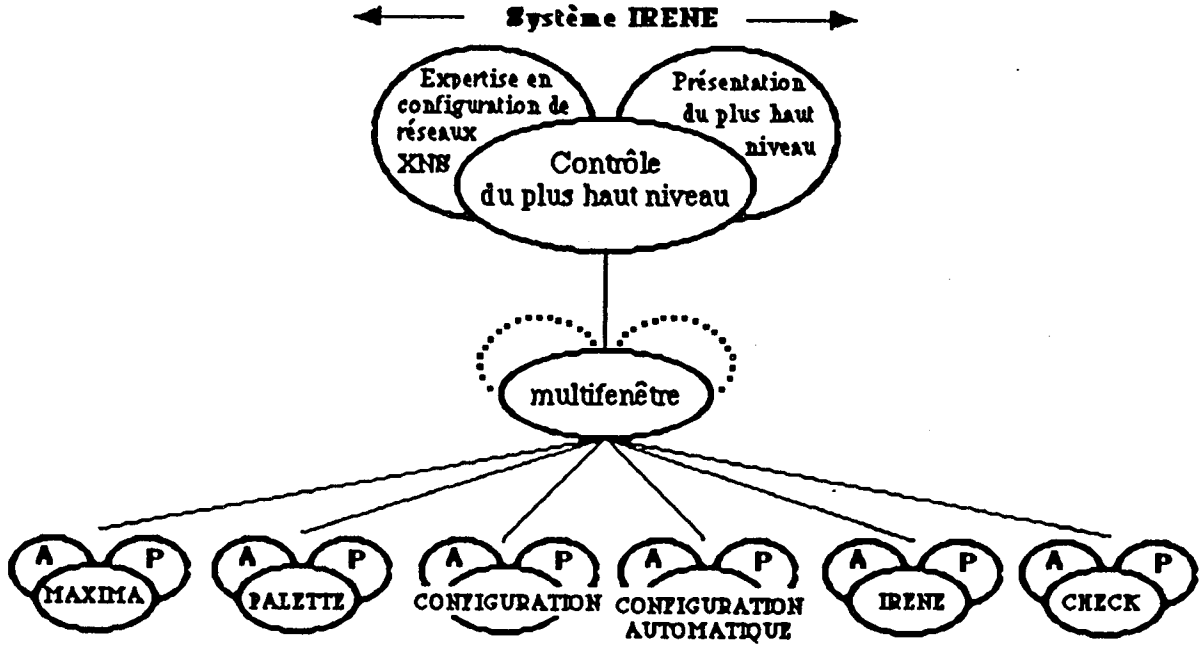


Figure 9.37 : L'architecture PAC du système IRENE.

Le Contrôle du niveau le plus haut joue son rôle habituel d'intermédiaire avec le maintien des liens entre les concepts du domaine et les objets de présentation. Dans le cas d'IRENE, il illustre parfaitement la fonction de traducteur entre le formalisme de l'Abstraction et celui de la Présentation. En l'occurrence, les différences ne concernent pas seulement la dissimilitude de vue mais aussi celle des langages de réalisation : l'Abstraction est programmée dans le langage ELOISE alors que la Présentation est entièrement réalisée en Loops. Ce fait s'observe couramment dans d'autres réalisations telles que PIMS [Lunati 88]. Le Contrôle du niveau le plus haut est le lieu adéquat à la réalisation de ce double transfert : changement de perspective et changement de langage d'implémentation. Dans le cas précis d'IRENE, les langages Loops et ELOISE reposent tous deux

sur InterLisp. Cette base commune offre une technique simple et naturelle de communication au moyen de fonctions Lisp.

La Présentation a peu de propriétés personnelles. Elle se contente de définir la vue initiale du système sous forme d'une icône (figure 9.38). Une fois l'icône ouverte, l'Image est entièrement assurée par les constituants de l'interface.

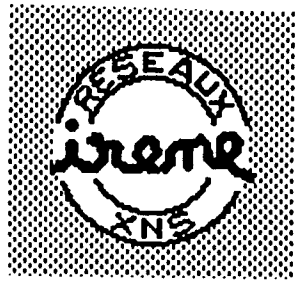


Figure 9.38 : La vue initiale du système est gérée par la Présentation du niveau le plus haut.

Les constituants de l'interface illustrent la réalisation de quelques éléments originaux de l'interaction homme-ordinateur : l'analyse syntaxique à divers niveaux d'abstraction, la création dynamique d'exemplaires de concepts, la génération des messages d'erreur sémantique et la délégation de "pouvoirs sémantiques" dans l'interface. Nous allons successivement reprendre ces quatre points.

5.4. Analyse syntaxique à divers niveaux d'abstraction

La création d'un exemplaire de concept donne lieu à une analyse syntaxique répartie sur plusieurs niveaux d'abstraction. Nous rappelons que la demande d'une création s'exprime par la séquence d'actions physiques suivantes : désignation d'un symbole graphique dans la palette, sélection de l'opérateur de création dans le menu fantôme et désignation d'un site destinataire dans la fenêtre de configuration. L'exemple est intéressant parce qu'il fait intervenir des événements dispersés dans l'Image à la fois dans la palette et dans la configuration. Le premier niveau d'abstraction de l'analyse syntaxique a lieu dans les fenêtres. Le second a lieu dans l'objet **multifenetre** de la figure 9.37.

5.4.1. Premier niveau d'analyse : les fenêtres

Toutes les fenêtres du système (**PALETTE, CONFIGURATION, IRENE, CHECK**, etc.) sont des entités PAC réalisées chacune sous forme de trois objets Loops (un par constituant : Présentation, Abstraction, Contrôle). Cette organisation permet de réutiliser les services de l'environnement. Par exemple, les Présentations des fenêtres d'IRENE sont des spécialisations de fenêtres Loops. Le

traitement des événements devient alors très dépendant du système de fenêtrage hôte. Dans ce système, tout événement est dirigé vers la fenêtre concernée, c'est-à-dire, dans notre cas, vers une Présentation de fenêtre IRENE. Nous distinguons deux classes de fenêtres IRENE : les fenêtres graphiques et les fenêtres textuelles. L'exemple de la création fait intervenir des exemplaires de fenêtres graphiques. Nous reviendrons sur les fenêtres textuelles à propos de l'affichage des messages.

Une Présentation de fenêtre graphique a essentiellement les compétences d'une fenêtre Loops : elle affiche des "bitmaps" et des titres ; elle est capable de rafraîchir sa surface à la suite d'opérations de défilement, de changements de taille ou d'exposition ; elle reçoit les événements en provenance du système hôte mais en avertit immédiatement son Contrôle.

Le Contrôle d'une fenêtre graphique gère l'état de la fenêtre. Il est donc en mesure de vérifier le droit d'interpréter un événement. Si l'événement est acceptable, le Contrôle demande à son Abstraction de déterminer l'objet propriétaire de l'événement. L'Abstraction regroupe ici l'une des fonctions usuelles du Contrôle : celle de gérer la création et la destruction des constituants. Elle laisse au Contrôle le rôle d'intermédiaire et de gestionnaire local de l'état de l'interaction.

L'Abstraction d'une fenêtre graphique gère la liste des objets graphiques qui y sont présents. Un objet graphique n'est pas une entité PAC mais un simple objet Loops sans activité personnelle. Il se comporte comme une structure de données passive dont les attributs essentiels sont la localisation dans la fenêtre et des bitmaps de symboles graphiques. Les objets élémentaires d'IRENE ont une présentation essentiellement statique (pas d'animation). Il est donc inutile de leur "donner une vie propre". En conséquence, contrairement aux objets PAC du système Thermo, ils n'ont pas de méthode pour traiter un événement et prendre des décisions. Ils sont simplement capables de répondre à des questions (lectures de valeur d'attribut) et d'obéir à des ordres (affectations de valeur à un attribut).

Dans l'exemple de la création d'un exemplaire de concept, l'événement qui traduit la désignation d'un point de la palette est reçu par la Présentation de la palette. Son Contrôle est averti de l'événement, en vérifie la pertinence et demande à son Abstraction de déterminer l'objet graphique concerné. Si la réponse désigne un objet, le Contrôle l'interroge sur les attributs utiles à la situation ("bitmap", localisation, type), demande à la Présentation d'afficher ce bitmap à telle localisation et en vidéo inverse et fait apparaître le menu correspondant au type d'objet. La réponse du menu est reçue par le Contrôle et entraîne la sortie de l'automate local du traitement de l'interaction : le Contrôle émet un signal de création d'exemplaire en direction de l'objet **multifenêtre**. Le traitement de la commande entre dans le deuxième niveau d'abstraction.

5.4.2. Deuxième niveau d'analyse : l'objet multifenêtre

L'objet multifenêtre cimente les actions réparties sur l'Image. Elle n'a pas de Présentation (à l'exception de la gestion de la forme du curseur pour traduire l'activité ou au contraire la disponiblité du système) ni d'Abstraction : c'est un Contrôle qui gère l'état de l'interaction à un niveau supérieur (celui des commandes), et qui influence le comportement de ses constituants (les fenêtres).

Une commande est une structure abstraite qui comprend un type et des éléments de description. Par exemple, une commande de type création comprend deux éléments : la classe de l'objet à créer et l'objet de la configuration récepteur du futur exemplaire. Un exemplaire de commande est créé à la réception d'un signal de création de commande. Dans notre exemple, la palette signale la demande de l'utilisateur par une création de commande de type "création" et précise la classe de l'objet désiré. La commande ainsi créée est incomplète : l'objet destinataire doit être spécifié. A cette fin, **multifenêtre** impose un état aux deux fenêtres impliquées dans l'élaboration de la commande.

A tout instant, multifenêtre contrôle le mode d'interprétation des événements dans les fenêtres constituantes. Dans notre exemple, tant que le paramètre manquant n'est pas spécifié, PALETTE devra interpréter un clic souris comme une annulation de commande, CONFIGURATION devra interpréter un clic souris sur un symbole graphique comme une désignation de localisation et considérer tout autre clic comme une annulation de commande.

Lorsque la commande est complète, multifenêtre soumet une demande de service à l'expertise (c.-à-d. à l'Abstraction du niveau le plus haut) par l'intermédiaire du Contrôle du niveau le plus haut. Ce dernier active son mécanisme d'indirection : les références aux objets de l'interface sont remplacées par des références équivalentes aux concepts de l'expertise. Nous verrons au paragraphe suivant comment s'effectue la liaison.

La figure 9.39 traduit le cheminement des informations durant cette étape de l'interaction. Nous allons décrire le cheminement inverse avec le traitement de la réponse émise par l'Abstraction du niveau le plus haut. Celle-ci va nous permettre d'illustrer la création dynamique de liens entre concepts du domaine et objets de présentation.

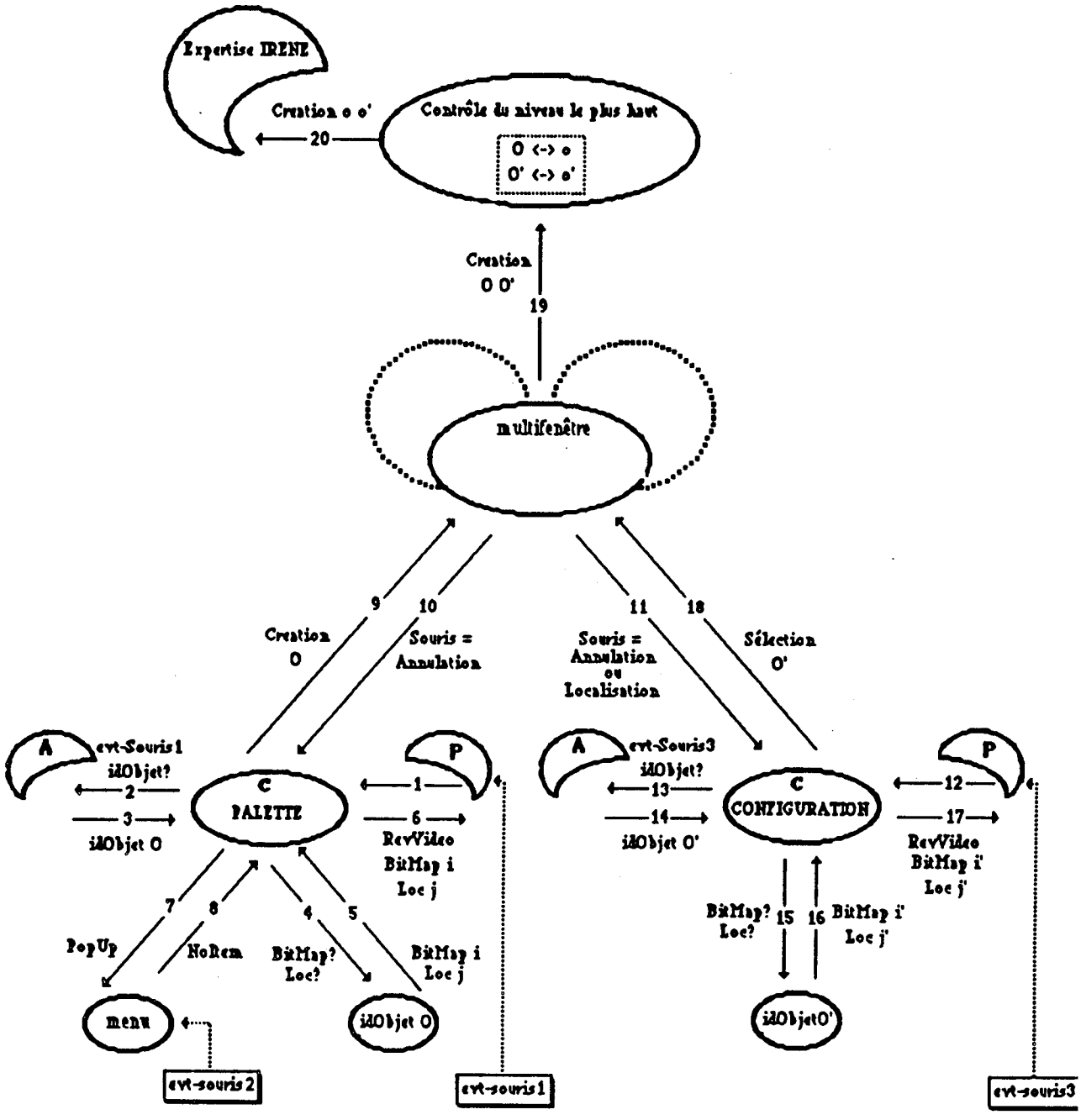


Figure 9.39 : Le cheminement des informations dans le sens interface-application pour une demande de création d'un exemplaire. Les 2 niveaux d'abstraction impliqués dans la saisie d'une commande : PALETTE/CONFIGURATION et MULTIFENETRE.

5.5. Création dynamique d'entités

La création dynamique d'entités est souvent une source de difficulté pour le réalisateur d'interface car elle exige un mécanisme de lien dynamique entre concepts du domaine et objets de présentation. La figure 9.40 montre le principe général de la mise en œuvre. Elle fait suite à la description de la figure précédente qui décrivait le cheminement aller des informations.

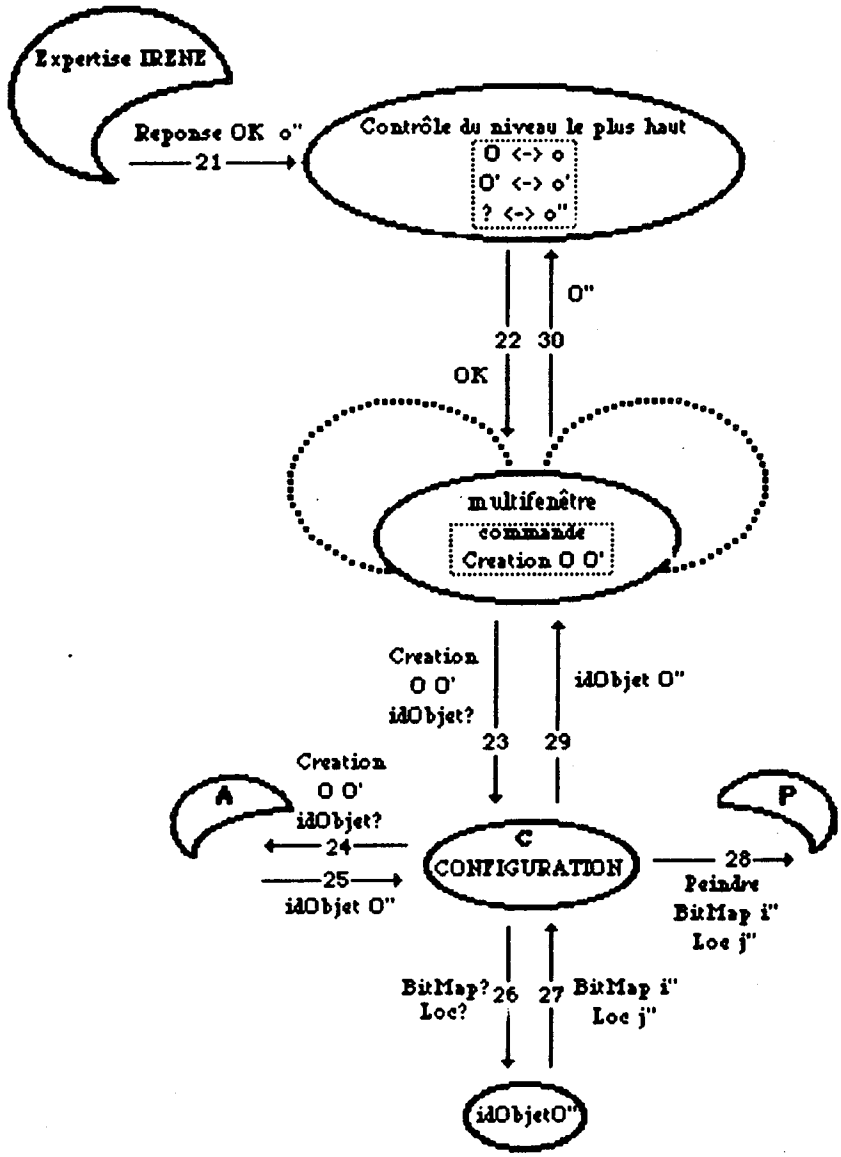


Figure 9.40 : Le cheminement des informations dans le sens application-interface pour une demande de création d'exemplaire. A l'étape 30, le lien entre le concept o" du domaine et l'objet de présentation O" sera effectif ([? <-> o] sera complété en [O" <-> o"]).

Nous nous plaçons dans la situation où le traitement de la demande est entré dans le domaine de l'expertise. Nous supposons que les vérifications sémantiques répondent favorablement à la requête. La réponse est une structure symbolique du domaine qui reflète l'effet sémantique de la commande ; elle précise notamment l'identification **idf-expertise** de l'exemplaire créé dans la mémoire de travail. La structure est réceptionnée par le Contrôle du niveau le plus haut. Ce dernier mémorise l'existence de **idf-expertise** en créant un lien partiellement résolu (l'objet de présentation est encore inconnu) ; il indique ensuite à multifenêtre que la réponse est favorable.

Au paragraphe précédent, nous avons décrit le rôle de multifenêtre comme analyseur des entrées Nous allons voir sa fonction de synthétiseur des sorties. multifenêtre sait relier la réponse symbolique à la commande de création maintenue dans sa mémoire de contrôle. Une réponse favorable à une telle commande se traduit par une demande de création d'un exemplaire d'objet graphique auprès de la fenêtre de configuration. Cette dernière, qui contrôle la géométrie de son contenu, évalue l'emplacement de l'objet à créer, ordonne la création et l'affichage. L'exemplaire de présentation se crée et s'identifie sous le nom d'**idf-interface**. Ce nom est ensuite retransmis jusqu'au Contrôle du niveau le plus haut qui complète dans sa mémoire de contrôle le lien d'indirection **idf-expertise/idf-interface.**

Le lien entre le concept du domaine et l'objet de présentation étant maintenant connu, le Contrôle du niveau le plus haut est en mesure de traduire la structure symbolique émise par l'expertise en réponse à la commande de l'utilisateur. Cette fonction fait l'objet du paragraphe suivant.

5.6. Génération des messages d'erreur sémantique

La qualité informative des messages d'erreur est un facteur déterminant dans les étapes d'évaluation et d'élaboration du plan de travail. Hélas, elle est souvent gâtée par une expression pauvre et rigide qui ne tient pas compte de la situation : chacun de nous a certainement apprécié le caractère informatif du fameux "SYNTAX ERROR"! A condition que l'architecture logicielle s'y prête, les maladresses qui relèvent de l'interface sont facilement réparables. La situation est plus grave lorsque les causes proviennent de l'application : un habillage syntaxique ne parvient jamais à couvrir le manque d'information sémantique. Dans le cas d'IRENE, qui a pour vocation l'aide à la conception, la richesse informative des expressions de sortie est essentielle. Il est clair que le laconisme expéditif de l'expression "commande refusée" est inacceptable.

Nous retrouvons le problème évoqué au paragraphe 5.3 sur la difficulté, au moment de la conception, de déterminer les informations abstraites utiles à l'opérateur humain. L'expérience forgée avec la réalisation d'IRENE suggère l'application systématique de la stratégie générale suivante : toute "modification sensible" de l'état de l'application doit se traduire par la production d'une expression symbolique. J'entends par "modification sensible", un changement d'état d'un concept connu de l'utilisateur. Une erreur est un concept! Elle intéresse au premier chef les utilisateurs mais les réalisateurs de logiciels, par souci simplificateur, l'écartent trop souvent du centre de leurs préoccupations. En conséquence, la notion d'erreur doit être explicitement représentée dans l'application.

Dans IRENE, toute modification d'élément sensible de la mémoire de travail se traduit par la production d'une expression symbolique. Sont sensibles les éléments correspondant aux constituants de la configuration et les éléments qui modélisent les anomalies. Une expression symbolique s'exprime dans les termes de l'expertise. En particulier, les constituants de la configuration sont désignés par des noms locaux à l'expertise (des **idf-expertise**). Grâce au mécanisme de liaison maintenu par le Contrôle du niveau le plus haut, les noms locaux à l'expertise sont remplacés par des noms locaux à l'interface (des **idf-interface**).

Une commande de l'utilisateur peut entraîner la modification de plusieurs éléments sensibles de la mémoire de travail. L'expression symbolique produite en réponse à une commande est donc une "compilation" des expressions élémentaires classées par catégories, en l'occurrence : les besoins, les incompatibilités, les anomalies heuristiques et les anomalies résolues. La classification des expressions de sortie en catégories devrait être assurée par toute application : elle permet de mettre en valeur la qualité sémantique des messages par des attributs de présentation. Par exemple, les expressions de la catégorie "danger" peuvent être présentées dans une fenêtre particulière assortie d'une icône évocatrice, voire d'un signal sonore. La classification proposée pour IRENE dépend clairement du domaine. Elle est utile à la présentation des messages par ordre de gravité.

Pour poursuivre notre exemple, l'expression symbolique produite par l'expertise est traduite par le Contrôle du niveau le plus haut en termes d'idf-interfaces, puis transmise à l'objet qui gère les commandes : multifenêtre. Selon la commande concernée par la réponse, l'affichage a lieu dans les fenêtres IRENE ou CHECK. Ces dernières sont des fenêtres textuelles. Une Présentation de fenêtre textuelle est capable d'effectuer les opérations usuelles de rafraîchissement. Elle décide également du passage à la ligne puisqu'elle connaît sa largeur. Le Contrôle est l'intermédiaire habituel entre la Présentation et l'Abstraction. L'Abstraction d'une fenêtre textuelle est un constructeur de phrases en langue anglaise (ou française) élaborées à partir des informations symboliques fournies par multifenêtre.

Dans le cas de la fenêtre IRENE, une phrase particulière est le séparateur entre les flots de phrases. Ce séparateur, en l'occurrence une ligne d'étoiles, permet à l'opérateur d'identifier plus facilement les messages de réponse à une commande donnée. Il existe deux autres phrases prédéfinies qui, comme le séparateur, sont utiles à l'étape d'évaluation : "Command cancelled" et "Command completed". La première est présentée pour confirmer l'annulation de la commande en cours de spécification ; la seconde est affichée lorsque l'expertise ne produit aucune expression : l'opérateur habitué à consulter la fenêtre IRENE peut s'assurer, en complément des effets visuels sur le reste de l'Image, que l'action est traitée avec succcès.

La technique de génération des messages est simple pour des résultats assez probants. Par exemple, le symbole qui exprime la disparition d'une anomalie se traduit par la chaîne : "You no longer need a" (ou "Vous n'avez plus besoin de"). A l'avantage de la simplicité, s'ajoute celui de la souplesse. Une formulation maladroite est remaniable : l'architecture qui concentre les fonctions de traduction dans des objets spécialisés, rend possibles les ajustements syntaxiques. Nous sommes loin du piège courant des messages statiques déclarés comme des chaînes de caractères dispersées au hasard des modules.

5.7. Délégation de "pouvoirs sémantiques" dans l'interface

La délégation de pouvoirs sémantiques consiste à placer, dans l'interface, des fonctions relevant du domaine. Cette technique présente deux avantages, la réparation sémantique et l'amélioration des performances, que nous allons illustrer avec deux exemples tirés d'IRENE : le STATUS-MAN et la commande **SET MAXIMA**.

5.7.1. Réparations sémantiques

Quel que soit le sérieux de l'étude, aucune conception ne peut prétendre définir de manière exhaustive l'ensemble des fonctions utiles à la tâche. Dans certains cas, les lacunes sont purement fonctionnelles : l'application envoie à l'interface des informations qui pourraient aussi servir de données à des fonctions dépendantes du domaine mais utiles à l'interface. Le STATUS-MAN d'IRENE illustre parfaitement la situation.

Dans notre conception initiale, nous n'avions pas perçu la nécessité de présenter une version synthétique de l'état de la configuration. L'expertise se contentait d'émettre des expressions symboliques à partir desquelles les fenêtres textuelles produisaient des messages en langue naturelle. La fonction de comptage des diverses catégories d'expressions, nécessaire à l'élaboration de la vue synthétique, faisait défaut. La réparation fonctionnelle pouvait s'envisager en pratiquant une extension dans l'expertise ou bien dans l'interface.

Modifier l'application est envisageable si l'on dispose des programmes source. C'est la solution choisie pour la réalisation du STATUS-MAN : une réponse symbolique ne comprend pas seulement les expressions élémentaires par catégorie mais aussi les deux entiers qui représentent respectivement le nombre d'erreurs tolérables et le nombre d'erreurs critiques. multifenêtre, qui connaît le monde qu'elle contrôle, répartit les informations symboliques entre les fenêtres textuelles (pour les messages) et la fenêtre de configuration (pour les deux compteurs). Cette dernière, responsable de la présentation de son contenu, choisit le symbole graphique approprié à la situation. (Le STATUS-MAN est un objet

graphique. Pour les raisons déjà évoquées, nous n'avons pas jugé utile de le réaliser en véritable entité PAC.)

L'extension fonctionnelle dans l'interface est possible si l'application lui fournit les paramètres d'entrée. Cette condition est satisfaite dans le cas de l'expertise d'IRENE : la réponse est une liste d'expressions typées. Il suffit que la fonction de comptage interprète cette liste pour reconstituer les valeurs des deux compteurs. Le modèle PAC, qui autorise la répartition des fonctions sémantiques, est parfaitement approprié à la réparation fonctionnelle. Dans le cas d'IRENE, il serait naturel de placer la fonction de comptage dans l'Abstraction de multifenêtre.

5.7.2. Amélioration des performances

La qualité sémantique du retour d'information peut exiger des aller-retour fréquents entre l'application et les couches les plus basses de l'interface (celles qui interprètent les événements). La conséquence de longues chaînes d'échanges peut être un temps de réponse incohérent avec l'expectative de l'utilisateur. La délégation de pouvoirs sémantiques dans l'interface garantit l'amélioration des temps de réponse en réduisant la chaîne des échanges. La commande **SET-MAXIMA** est une illustration de cette technique.

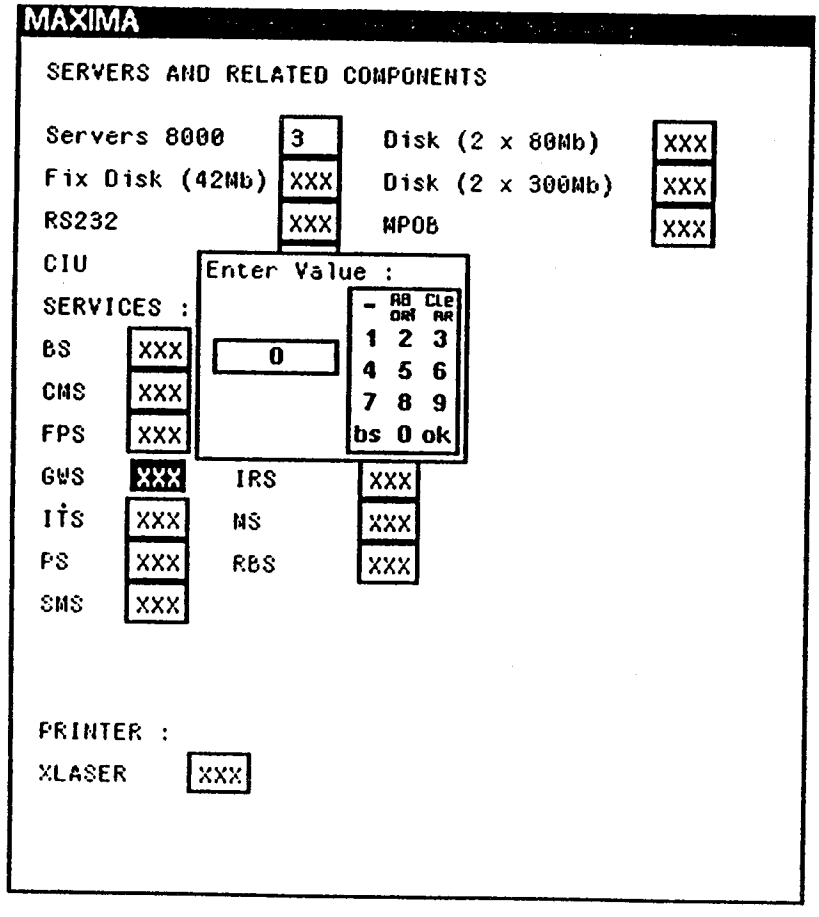


Figure 9.41 : un formulaire dont l'Abstraction a des pouvoirs sémantiques relevant du domaine.

La figure 9.41 montre le formulaire de saisie de la commande SET-MAXIMA. Il permet à l'opérateur de spécifier, pour chaque catégorie de constituants, une contrainte sur le nombre maximum d'exemplaires. Ces contraintes servent d'indicateurs complémentaires à la connaissance experte. Dans le cas présenté, la configuration doit au plus comporter trois serveurs. Les cases où figurent des XXX indiquent qu'il n'y a pas eu de maximum imposé.

La délégation de pouvoirs sémantiques est ici appliquée à la mise en pratique de deux règles ergonomiques : la concision et la qualité informative du retour d'information. Puisqu'il s'agit d'un formulaire, les champs doivent proposer des valeurs par défaut et la saisie doit autant que possible s'accompagner de vérifications sémantiques immédiates :

- la nature du formulaire MAXIMA suggère pour valeurs par défaut, celles des maxima spécifiés à la dernière intervention de l'utilisateur. Ces valeurs, qui concernent l'expertise, sont mémorisées dans l'application ; elles sont aussi modélisées dans l'Abstraction du formulaire afin de limiter les échanges. Cette technique, qui reproduit une part des informations abstraites est acceptable dans la limite de la capacité mémoire de l'environnement d'accueil.

- la vérification sémantique systématique à chaque saisie de champ n'est pas forcément souhaitable ni pour l'utilisateur ni pour les performances. Pour cette raison, il est fréquent que la vérification soit effectuée globalement en fin d'édition de formulaire. Dans d'autres circonstances, il est utile que l'utilisateur soit informé dès que possible d'une saisie erronée. En l'occurrence, si l'utilisateur spécifie un maximum de deux serveurs alors que la configuration en comporte déjà trois, il est utile de l'informer immédiatement de l'incohérence. Le nombre d'exemplaires de chaque catégorie présents dans la configuration est connu de l'application. Il se trouve que ces concepts du domaine ont une représentation dans l'interface. Le formulaire peut donc, sans interroger l'application, connaître la situation actuelle de la configuration et effectuer des vérifications sémantiques utiles et immédiates à chaque saisie de champ. Cette connaissance pourrait être mise à jour à chaque modification de la configuration. Pour des raisons de performance (la commande SET-MAXIMA est peu employée), elle est n'actualisée qu'à l'ouverture du formulaire : la sélection de la commande SET-MAXIMA est réceptionnée, puisqu'il s'agit d'une commande, par multifenêtre. Cette dernière interroge la fenêtre CONFIGURATION sur le nombre d'exemplaires par catégorie. La réponse est ensuite transmise par multifenêtre lorsqu'elle active la fenêtre MAXIMA.

Le formulaire MAXIMA, grâce à son savoir-faire sémantique, minimise les coûts de transfert avec l'application. Ses compétences syntaxiques peuvent être exploitées aux mêmes fins en ne transmettant que les éléments effectivement modifiés : un champ édité qui retrouve sa valeur initiale ne fait pas partie du transfert.

Ces exemples montrent l'adéquation du modèle PAC à la prise en compte de la délégation des fonctions sémantiques. La mise en œuvre de cette forme particulière de la répartition repose toutefois sur les ressources en mémoire du matériel d'accueil mais elle réduit sensiblement les échanges avec l'application. Cette réduction est très certainement souhaitable dans le cas, de plus en plus fréquent, où l'application s'exécute sur un calculateur distant.

EN RESUME :

L'expérience acquise selon les lignes directrices du modèle PAC peut se formuler en quelques règles utiles à la conception et à l'organisation des systèmes interactifs.

Règle 1 :

Dans un système interactif, il faut distinguer les fonctions internes des techniques de présentation, mais ces deux perspectives d'un même sujet, l'Abstraction et la Présentation, doivent être reliées par une passerelle logicielle explicite : le Contrôle. Cette organisation en composants de Présentation, d'Abstraction et de Contrôle est applicable récursivement à tous les niveaux d'abstraction d'un logiciel interactif.

Règle 2 :

Une Présentation et son Abstraction ne communiquent jamais directement mais toujours par l'intermédiaire de leur Contrôle. Toutes deux s'expriment dans les termes qui leur sont propres. Cette contrainte garantit l'indépendance de la réalisation entre les comportements interne et externe. Un Contrôle doit donc inclure un mécanisme de liaison et de traduction entre les perspectives qu'il sert.

Règle 3 : Corollaire des Règles 1 et 2

La séparation entre l'application et l'interface n'est pas un gouffre fonctionnel mais une frontière entre deux perspectives sur un même domaine. Cette frontière doit trouver son équivalent logiciel. PAC propose le "Contrôle du niveau le plus haut". L'application et ce Contrôle communiquent dans les termes de l'application. Cette dernière n'a aucune connaissance des objets de présentation utilisés dans l'interface : sa réalisation ne doit en aucun cas désigner l'un de ces objets.

Règle 4 :

Les propriétés du mécanisme de liaison doivent être définies en fonction des besoins du système et de l'utilisateur. Le mécanisme peut être statique ou dynamique, multiple et distribué. Dans un mécanisme de liaison statique, la correspondance entre abstractions et présentations est invariable. Un mécanisme de liaison dynamique autorise la redéfinition d'une association ; il permet aussi la création de liens. Cette fonction est indispensable dans un système où des abstractions peuvent être créées dynamiquement. La liaison multiple permet d'associer à une même abstraction plusieurs objets de présentation. Ce service est utile à la mise en œuvre de la représentation multiple d'un même concept. Dans un mécanisme de liaison distribué un concept structuré n'est pas contraint d'être présenté par un seul objet mais chacun de ses éléments peuvent être associés à des objets distincts. Ce mécanisme, qui évite des indirections inutiles, garantit une meilleure efficacité.

Règle 5 :

Un objet de présentation doit ignorer l'existence de ses voisins. Ceci signifie qu'une réalisation d'objet ne peut désigner explicitement un objet de statut équivalent. Toute référence doit s'effectuer par l'intermédiaire d'un objet père désigné de manière symbolique. Cette contrainte facilite les réparations de l'interface et garantit la réutilisation de logiciels.

Règle 6 : Corollaire de la Règle 5

La création d'un objet composé s'impose dès l'instant où des objets ont des relations. Celui-ci joue alors le rôle d'arbitre entre les compétiteurs ou bien de vérificateur permanent de contraintes. Il s'applique au cas de la cohabitation de plusieurs domaines, à l'allocation de l'unité centrale, à l'expression de contraintes spatiales, à l'assemblage graphique, au rafraîchissement et aux réparations dues aux superpositions, à l'analyse syntaxique distribuée dans l'Image (cimentation des actions réparties sur un ensemble d'objets de présentation), aux dialogues à plusieurs fils d'activité.

Règle 7 :

La délégation de pouvoirs sémantiques dans l'interface doit être pratiquée tant que l'autorisent les ressources en mémoire du matériel d'accueil. Cette technique, qui réduit les échanges avec l'application (éventuellement distante), permet d'améliorer les temps de réponse.

Règle 8 :

Les objets élémentaires ne sont pas nécessairement des agents PAC. C'est le cas lorsqu'ils sont hérités de l'environnement ou lorsque leur comportement graphique est essentiellement statique. Ils doivent alors être considérées comme des entités passives capables seulement de répondre à des questions (opérations de lecture) et d'obéir à des ordres (opérations d'écriture).

Règle 9 :

Tout automate local d'objet doit, pour chaque état, prévoir une possibilité d'échappement vers la terminaison. Cette précaution garantit l'évitement des blocages dans un mode.

Règle 10 :

L'application doit signifier à l'interface toute modification sensible de son état, c'est-à-dire tout changement d'état de tout concept connu de l'utilisateur. Une erreur est un concept!

Règle 11 : Corollaire de la Règle 10

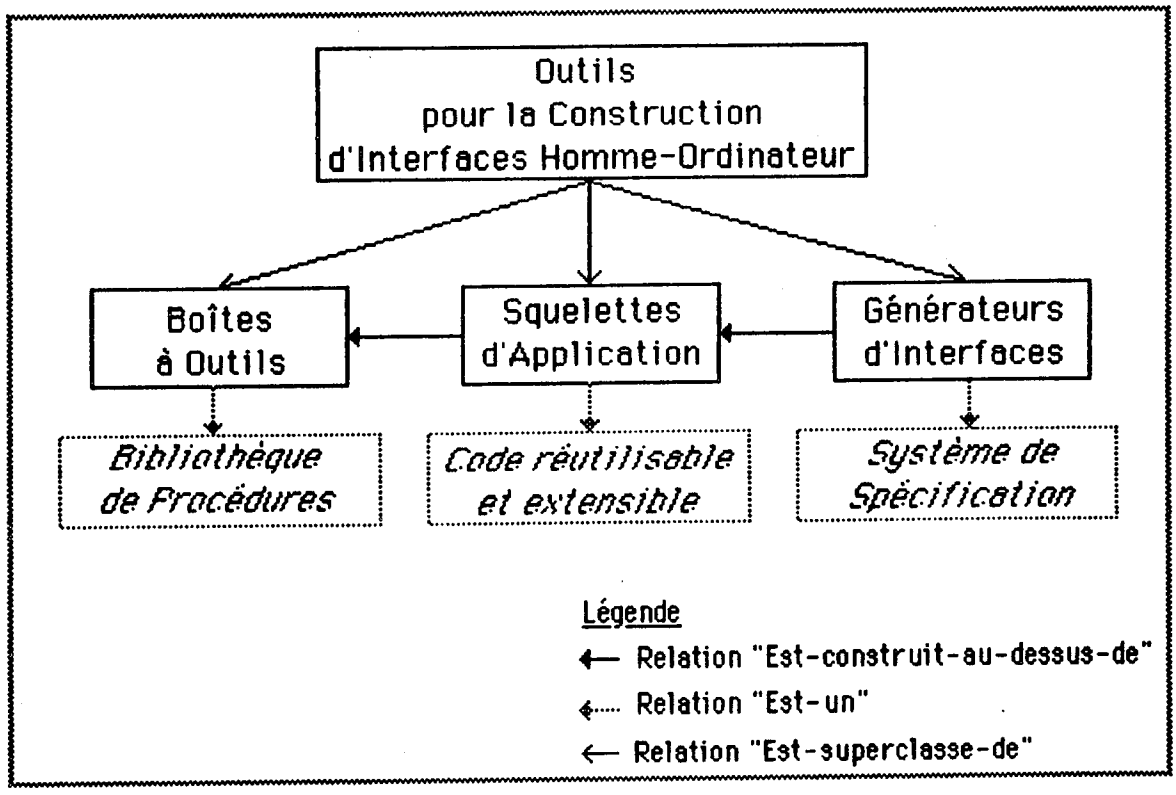
L'application doit impérativement modéliser le concept d'erreur et proposer une classification qui permette de mettre en valeur, par des attributs de présentation, le contenu sémantique du message. Ne pas tomber dans le piège des messages statiques déclarés comme des chaînes de caractères dispersées au hasard des modules. Un message, comme toute information émise par l'application, doit être exprimé sous une forme symbolique dégagée de tout attribut syntaxique et lexical. Un objet de présentation spécialisé dans la traduction se charge de produire la forme adaptée aux circonstances (classe d'utilisateur, langue naturelle, etc.).

Section III

Outils pour la Construction d'Interfaces Interactives

La section précédente nous a présenté la nature et l'organisation des abstractions spécifiques aux logiciels interactifs. Nous devons maintenant compléter ce cadre de principes directeurs par une vue plus concrète du domaine. Cette section est consacrée aux outils logiciels qui réalisent les abstractions identifiées précédemment et qui véhiculent les modèles d'architecture déjà cités.

Les outils pour la construction d'interfaces homme-ordinateur se présentent sous trois formes : les boîtes à outils, les squelettes d'application et les générateurs d'interfaces. Cette classification grossière est illustrée dans la figure suivante :



- Une boîte à outils est une bibliothèque de procédures. L'éventail des fonctions offertes va de la gestion des événements physiques tels que les clics souris et l'affichage de points, à la gestion d'entités de dialogue telles que les menus et les formulaires.
- Un squelette d'application réalise les fonctions usuelles de l'interface homme-machine sous la forme d'un logiciel réutilisable et extensible. La tâche du programmeur consiste à greffer sur le squelette les composants spécifiques à son application. Un squelette est construit au dessus d'une boîte à outils.
- Un générateur d'interfaces crée l'interface homme-ordinateur d'une application interactive à partir d'une spécification. Cette interface est reliée à un noyau d'éxécution qui s'apparente à un squelette d'application.

Le chapitre 10 présente le principe des boîtes à outils. Il propose une étude comparative des boîtes à outils existantes et identifie les avantages et limitations de ce type d'outil.

Le chapitre 11 présente les techniques de réalisation des machines à images abstraites. Une machine à images abstraites définit, quand elle existe, un niveau d'abstraction dans une boîte à outils. Ses avantages pour la restitution et l'acquisition d'information méritent d'être exposés.

Le chapitre 12 est consacré aux squelettes d'application. Il montre à l'aide d'un exemple comment cette classe de logiciel comble utilement les lacunes des boîtes à outils.

Le chapitre 13 développe les aspects essentiels de la génération automatique d'interfaces interactives. Il en indique les éléments prometteurs mais aussi les limitations actuelles.

Chapitre 10

Boîtes à Outils

1. Définition

1.1. Boîtes à outils et gestion du poste de travail

1.2. Boîtes à outils et gestion du dialogue

2. Eléments comparatifs entre boîtes à outils

2.1. Stratégie de contrôle
- 2.1.1. Protocole embarqué
- 2.1.2. Protocole non embarqué
- 2.1.3. Elément d'évaluation entre protocoles embarqués et non embarqués

2.2. Personnalisation par programme

2.3. Personnalisation par l'utilisateur

2.4. Dispositifs pour l'interactivité
- 2.4.1. Interactivité et manipulation directe
- 2.4.2. Interactivité et édition d'objets graphiques

2.5. Dispositif pour la distribution
- 2.5.1. Dialogue à distance : espace d'affichage distribué
- 2.5.2. Migration de l'information : "couper-coller"

2.6. Dispositifs pour l'information multimédia

3. Avantages et inconvénients

3.1. Les avantages
- 3.1.1. Portabilité des logiciels interactifs
- 3.1.2. Extensibilité des fonctions
- 3.1.3. Souplesse d'utilisation
- 3.1.4. Intégration de critères ergonomiques

3.2. Les inconvénients
- 3.2.1. Risque de mauvaise décomposition modulaire
- 3.2.2. Apprentissage difficile
- 3.2.3. Duplication d'effort

EN RESUME

1. Définition

Une boîte à outils est une bibliothèque de procédures adaptées à l'écriture d'interfaces homme-ordinateur. L'éventail des fonctions offertes définit différents niveaux d'abstraction. Pour simplifier, la figure 10.1 les organise en deux catégories : gestion du poste de travail et gestion du dialogue.

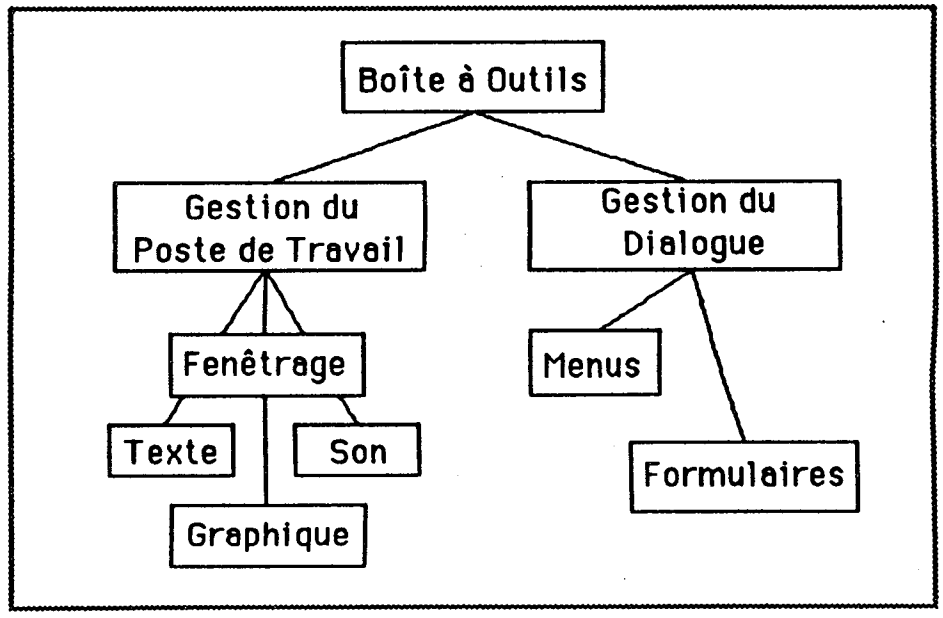


Figure 10.1 : Les fonctions usuelles d'une boîte à outils.

1.1. Boîte à outils et gestion du poste de travail

Les abstractions introduites pour la gestion du poste de travail définissent un terminal abstrait dont la fonction est de cacher le fonctionnement de l'appareil physique et d'en assurer le partage. Les abstractions usuelles, bien que variant d'une boîte à outils à l'autre, correspondent à des

- notions graphiques de base (surface d'affichage),
- notions graphiques construites (icône, curseur, fenêtre)
- notion d'affichage structuré (technique des boîtes, par exemple)
- notions textuelles (police de caractères),
- notion d'événement (en provenance des unités logiques),
- notion d'événement synthétisé (en provenance des programmes clients).

La nature et le rôle de ces abstractions ont été précisés au chapitre 6 sur les éléments fonctionnels des systèmes interactifs. Nous ne reviendrons pas sur ces aspects. Toutefois, il est bon d'insister sur trois critères déterminants de la qualité des services du niveau poste de travail : haut degré d'interactivité, distribution et support pour la communication d'information multimédia. Ces trois aspects seront développés au paragraphe 2 consacré aux éléments de comparaison entre boîtes à outils.

1.2. Boîte à outils et gestion du dialogue

Les services du niveau dialogue s'appuient sur les fonctions du poste de travail. On y trouve deux sortes d'entités :

- les entités simples comme les boutons,
- les entités composées comme les barres de défilement, les menus, les formulaires et les tableaux de boutons.

L'intérêt de ces entités de présentation prêtes à l'emploi est discuté au paragraphe 3 à propos des avantages des boîtes à outils. Les deux classes de services, gestion du poste de travail et gestion du dialogue sont, selon les constructeurs,

- réunies dans une bibliothèque unique : c'est le cas du Toolbox du Macintosh [Rose 86] ou d'Intuition de l'Amiga [Mical 86], ou bien
- interfacées dans des bibliothèques distinctes : c'est le cas de l'environnement X qui regroupe les primitives d'accès aux services de base dans la bibliothèque Xlib et les services de dialogue dans la bibliothèque X Toolkit.

Que l'interface de programmation soit définie par une ou plusieurs bibliothèques, le programmeur retrouve, d'un environnement à l'autre, des concepts similaires. Toutefois, les boîtes à outils présentent des différences qui influencent la réalisation du logiciel interactif.

2. Eléments de comparaison entre boîtes à outils

La décision d'utiliser une boîte à outils particulière peut résulter de contraintes conjoncturelles telles que la disponibilité. Elle peut aussi s'effectuer selon des critères plus "scientifiques" tels que l'efficacité, la facilité d'utilisation, l'étendue et le niveau d'abstraction des services offerts. Ici, nous retiendrons comme critère de choix la facilité d'utilisation et la nature des services. Pour le cas particulier d'une boîte à outils, j'ai identifié six composantes marquantes : la stratégie du contrôle qui dépend du mécanisme d'acquisition des événements, la possibilité de modifier ou non le comportement standard des entités d'interaction, la faculté de modifier ou non les détails syntaxiques

et lexicaux sans toucher au programme exécutable, la présence ou l'absence de mécanismes pour gérer les besoins de plus en plus exigeants de l'interactivité, de la distribution et du transfert d'information multimédia.

2.1. Stratégie de contrôle

Le protocole d'acquisition et de traitement des événements conditionne la structure de contrôle d'un système. Concernant les entités de dialogue, je distingue deux types de protocole : les protocoles embarqués et les protocoles non embarqués.

2.1.1. Protocole embarqué

Avec l'approche du protocole embarqué, toute entité d'interaction véhicule un mécanisme de traitement des événements. Ce mécanisme est automatiquement activé à chaque occurrence d'événement présentant un intérêt pour l'entité. L'intérêt qu'une entité porte à une classe d'événements est exprimé par l'entité elle-même et peut être modifié à tout instant.

Une entité de dialogue reçoit donc automatiquement les événements pour lesquels elle a manifesté de l'intérêt. Elle les traite et, en complément de traitement, provoquent l'activation des procédures fournies par le programme client. Pour chaque classe d'événement, le programme client peut spécifier une procédure. Si, pour une classe donnée, aucune procédure n'a été indiquée, le traitement complémentaire spécifique au programme client n'a pas lieu. Cela signifie que ce type d'événement n'a pas d'effet particulier sur le comportement du programme client.

Les "widgets" de X Toolkit et les "objects" de NeWS appliquent cette technique.

2.1.2. Protocole non embarqué

Avec l'approche du protocole non embarqué, le mécanisme de traitement n'est pas intégré à l'entité de dialogue : l'enchaînement des éléments qui constituent le traitement d'un événement est exprimé à l'extérieur de l'entité, c'est-à-dire par le programme client. Si l'on prend la vision de la programmation par objets, une entité de dialogue sans protocole embarqué est un objet sans méthode de traitement général des événements. Le programme client doit explicitement identifier la classe d'événement et appeler la méthode de l'entité qui répond à cette classe.

Toutes les entités de dialogue du Toolbox du Macintosh appliquent cette technique. L'exemple de la figure 10.2 illustre, en le simplifiant, le processus de traitement d'un événement. La difficulté pour

le programmeur commence dès la réception d'un événement (GetNextEvent) en provenance de la souris (theevent->what = mouseUp ou mouseDown). Connaissant l'adresse d'un point dans le système de coordonnées globale de l'écran (theevent->where), il faut maintenant déterminer la fenêtre propriétaire du point (FindWindow) puis la zone dans la fenêtre (inContent ou dans l'une des zones sensibles du cadre de la fenêtre tels inGrow, inDrag). Si l'événement s'est produit dans le contenu (InContent), alors la désignation du point doit être exprimée dans le système de coordonnées locales de la fenêtre (GlobalToLocal). Si l'événement ne concerne pas une entité de dialogue prédéfinie telle que les barres de défilement ou les boutons (FindControl), alors le programme client doit effectuer un traitement spécifique (objfind[1]).

```
While (go)
  if ( GetNextEvent(everyEvent, theevent) )
    switch ( theevent->what)
      case keyDown : ..... break;

      case mouseUp :
      case mouseDown :
          wheremouse = FindWindow (theevent->where,&whichwindow);
          switch (wheremouse)
            case inDesk : ..... break;
            case inMenuBar : ..... break;
            case inGrow : case inDrag : ..... break;

            case inContent :
              localwhere = &theevent->where;
              GlobalToLocal (&localwhere);
              whereincontrol = FindControl ( localwhere, whichwindow, ... );
              if ( whichcontrol == 0)
                 objfind ( whichwindow, localwhere, theevent);
              else .....
```

Figure 10.2 : Processus de traitement d'un événement dans le cas du Toolbox qui ne pratique pas le modèle du protocole embarqué. Les identificateurs qui comprennent à la fois des majuscules et des minuscules représentent des fonctions ou constantes prédéfinies dans le Toolbox du Macintosh.

1. objfind est une méthode de **OBJ**, superobjet contrôleur d'entités PAC (voir chapitre 9). Cette méthode est chargée de déterminer, s'il existe, le sous objet PAC propriétaire du point.

2.1.3. Elément d'évaluation entre protocoles embarqués et non embarqués

Ces deux protocoles se distinguent essentiellement par la localisation du contrôle et la difficulté d'apprentissage.

Dans l'approche du protocole embarqué, les procédures du programme client sont automatiquement appelées en complément de traitement alors que dans le cas du protocole non embarqué, le programme client décide des enchaînements. Dans le premier cas, le client est un serveur auquel la boîte à outils fait appel en dernier ressort ; dans le second cas, les rôles sont inversés et la boîte à outils se contente de répondre à des demandes de services. Si l'on considère la souplesse, il semble préférable de laisser le programme client maître du contrôle. Cet avantage a ses contreparties : une mauvaise architecture logicielle et la difficulté de trouver la colle.

Laisser le plein contrôle au programme client, c'est s'orienter vers une architecture à contrôle interne. Nous avons évoqué les dangers de ce type de contrôle au chapitre 6 à propos de la gestion du dialogue : dilution des composants de présentation dans les composants fonctionnels et emprisonnement potentiel de l'utilisateur dans des modes.

Le second désavantage du protocole non embarqué est de laisser le programmeur assembler la suite des procédures adéquates. Cet assemblage requiert une connaissance approfondie de la boîte à outils. Dans le cas du protocole embarqué, le programmeur n'a pas besoin de connaître le détail du fonctionnement interne de l'entité d'interaction. Il lui suffit de fournir des procédures qui répondent chacune à une situation précise.

Une solution acceptable est la présence, dans la boîte à outils, de deux niveaux d'abstraction pour le traitement des événements : un niveau élémentaire qui permet à l'application de prendre le contrôle en cas de besoin et un niveau conçu selon le modèle de la programmation par objets dans lequel chaque entité de dialogue dispose d'une méthode de traitement des événements avec appel automatique de procédures spécifiques au client. Si, pour une classe donnée, la méthode de traitement des événements ne convient pas, le programme client, grâce à l'approche par objets, pourra définir une nouvelle classe spécialisée. X Toolkit permet l'expression des deux types de protocole.

2.2. Personnalisation par programme

Au paragraphe 3, nous verrons que l'une des retombées utiles d'une boîte à outils est la définition d'un style de présentation. Si l'existence d'un style est souhaitable, le programmeur doit néanmoins pouvoir réaliser des ajustements. Les choix "normalisés" d'une boîte à outils sont supposés être les

mieux adaptés tant du point de vue ergonomique que du point de vue des besoins du programmeur mais il est illusoire de penser que ces choix conviennent à toutes les situations. Les boîtes à outils véritablement "orientées objet" permettent les ajustements cherchés sans compromettre le fonctionnement de l'existant. Nous avons rencontré un exemple de cette possibilité avec les fenêtres IRENE réalisées comme des spécialisations de fenêtre de l'environnement InterLisp.

Contrairement aux boîtes à outils réalisées dans un langage de programmation par objets, la plupart des boîtes qui se disent "orientées objet" mais réalisées dans des langages traditionnels, ne disposent pas du mécanisme naturel de réutilisation et d'extension. Elles fournissent, tel le Toolbox du Macintosh, des points d'ancrage, mais le protocole de programmation est suffisamment obscur pour décourager toute vélléité de modifications.

En liaison avec la modification par programme du comportement des entités de dialogue, nous devons aborder les possibilités de personnalisation des entités par l'utilisateur.

2.3. Personnalisation par l'utilisateur

La personnalisation par l'utilisateur suppose la possibilité de modifier l'interface sans aucune opération de programmation, de compilation ou d'édition de liens. Cette faculté repose sur l'existence d'une représentation externe de l'interface.

"Représentation externe" signifie ici représentation non câblée dans le système interactif et modélisée comme un ensemble de données permanentes. Une donnée permanente est une structure dont la durée de vie n'est pas liée à celle des processus où s'exécute le système interactif. Dans ces conditions, une représentation externe peut servir de donnée à plusieurs logiciels, en particulier, au système interactif pour lequel elle a été définie et à l'éditeur qui a permis de la construire et de la modifier.

La notion d'objets permanents utilisables par un programme comme données d'entrée n'est pas nouvelle. Dans les systèmes d'exploitation traditionnels, chaque utilisateur peut éditer un fichier qui définit le profil de son espace d'exécution. Dans le domaine des boîtes à outils, le Toolbox a été le premier à exploiter cette idée : la description des entités de dialogue d'une application est maintenue dans un fichier dit de ressources distinct du fichier contenant le code. Le fichier de ressources est une suite structurée de caractères ASCII qui, pour chaque entité de dialogue (appelée ressource), précise le type, l'identification et diverses informations liées à la présentation de la ressource. Cette description constitue une représentation pivot accessible à tous les outils. L'accès au fichier de ressources s'effectue grâce à des primitives de la boîte à outils. Ces primitives demandent comme informations

d'accès, l'identité du fichier de ressources (il peut y avoir plusieurs fichiers de ressources pa application), le type et l'identité de la ressource recherchée. L'entité de dialogue est alors créée e mémoire centrale à partir de la description ASCII.

La portée de la personnalisation ainsi permise se limite à des ajustements syntaxiques et lexicaux Les modifications du comportement logique interne ne sont pas accessibles à l'utilisateur. Elle doivent s'effectuer par programme. Même si la personnalisation n'affecte que des "attributs d surface", elle permet de réparer la terminologie ou l'arrangement spatial des entités de dialogue san faire intervenir l'implémenteur ni le concepteur.

Pour conclure, il est important de ne pas câbler les éléments syntaxiques et lexicaux dans u logiciel interactif. Ce principe ne s'applique pas seulement aux boîtes à outils. Il relève de l paramétrisation des logiciels. Depuis longtemps, on enseigne la nécessité de définir les bornes fixe d'un tableau par une constante ou mieux de déterminer sa valeur à partir d'une donnée externe Curieusement, dans les logiciels interactifs, domaine innovateur s'il en est, on régresse vers le pratiques antiques : on continue à utiliser des valeurs numériques à la place de symboles (ex CréerFenêtre à l'emplacement 10, 20) et des norias de chaînes de caractères diluées dans tout le code Tout message statique, tout libellé de formulaires, tout nom de boutons ou d'élément de menu devrai avoir une représentation externe!

2.4. Dispositifs pour l'interactivité

Le haut degré d'interactivité est la réponse logicielle aux règles ergonomiques du retou d'information immédiat et de la manipulation directe. Si la boîte à outils n'offre pas les service adaptés à ce type d'exigences, le réalisateur, confronté à des difficultés de programmation, sera tenté de contourner ces principes. Prenons deux exemples : la manipulation directe et l'édition d'objet graphiques.

2.4.1. Interactivité et manipulation directe

Dans la manipulation directe, les objets sont contraints à suivre les mouvements d'un dispositi physique de pointage (par exemple, la souris). La réalisation du suivi de la souris implique troi actions logicielles : effacer l'objet de son ancienne position, réparer la surface endommagée e redessiner l'objet au nouvel emplacement. Effacer et dessiner s'effectuent simplement : le premie avec une opération raster Ou-Exclusif de l'objet sur lui-même, ou encore en couvrant la surface occupée par l'ancien dessin avec un motif de nettoyage (en général, la couleur de la surface d restitution). La solution dépend du mécanisme de visualisation de l'objet selon qu'il existe une copie

du dessin dans une surface virtuelle ou que le dessin est créé à partir d'une liste d'affichage. La difficulté réelle est la réparation de la surface endommagée.

Nous avons évoqué le problème de la réparation au chapitre 6 à propos du fenêtrage et nous avons conclu de l'utilité des hiérarchies de fenêtres. Même si ces mécanismes de base existent, les services fournis dans les boîtes à outils ne répondent pas vraiment au cas précis de la manipulation directe. Afin de mesurer l'effet de ces lacunes sur l'effort de programmation, nous allons détailler la réalisation d'un déplacement d'objet. L'exemple choisi est tiré du système Thermo réalisé avec le Toolbox, boîte à outils réputée pour la qualité de ses services graphiques.

Le comportement du thermomètre de la figure 10.3 est réalisé sous forme d'un module C dont chaque procédure modélise un opérateur de l'entité PAC. Les attributs de chaque exemplaire de thermomètre sont regroupés dans une structure de données. Parmi ces attributs, nous retenons la region, **superglassregion**, macroinstruction de primitives graphiques dont l'interprétation produit un dessin de thermomètre. (Pour la définition d'une region, se reporter aux figures 6.6 et 6.7 du chapitre 6.) La figure 10.3 montre à droite la region d'un exemplaire rendue visible grâce à l'exécution de la primitive PaintRgn du Toolbox et à gauche lorsque DragGrayRgn est en cours d'exécution.

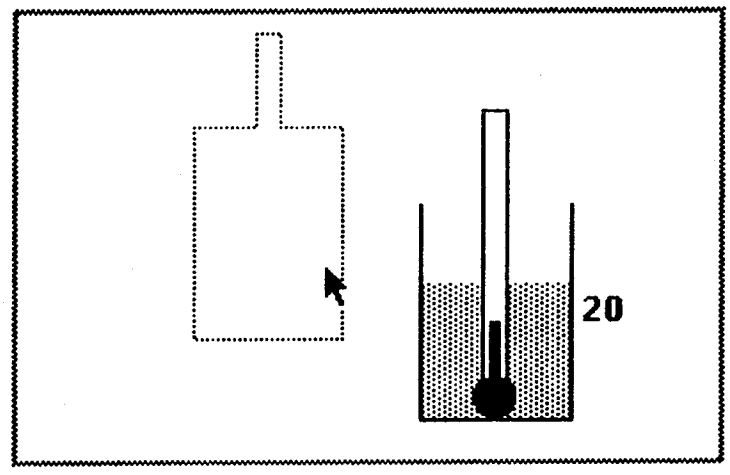


Figure 10.3 : Une region construite avec la boîte à outils du Macintosh ; vue de l'écran lorsque la region interprète l'opérateur DragGrayRgn.

DragGrayRgn lie le contour de la region aux déplacements de la souris. Dès que l'utilisateur en relâche le bouton, le contour disparaît automatiquement. Cette primitive du Toolbox réalise donc un retour d'information immédiat tout en restaurant dynamiquement les parties surchargées par le contour. Si l'utilisateur y trouve son compte (il peut aisément évaluer la position finale de l'objet), le programmeur n'est pas pour autant déchargé des trois actions logicielles (effacement, restauration et réaffichage) au relâchement du bouton de la souris. La figure 10.4 montre la programmation du mouvement d'un exemplaire de thermomètre avec la méthode **spgevent**. L'enchaînement des traitements

qui conduisent à l'appel de cette procédure sort de nos propos.

```
spgregion (...) .....;
spgwindow (...) .....;
.....
spgevent (whichinstance, theevent)
  switch (theevent->what)
    case mouseDown :
      dhdv = DragGrayRgn (spginstances[whichinstances].superglassrgn,...);
      spgerase (whichinstance);
      objgravity (SUPERGLASS, whichinstance, dhdv);
      spgdraw (whichinstance);

spgdraw (whichinstance)
    -- ici, le code qui dessine la region (voir figure 6.4) dans la surface
       virtuelle hors écran de l'exemplaire.
    myrect = rectangle occupé par la region dans la surface de restitution
    objsuperdraw (SUPERGLASS, whichinstance, myrect);
```

Figure 10.4 : Programmation du déplacement d'un objet. Les identificateurs constitués à la fois de majuscules et de minuscules représentent des fonctions ou constantes prédéfinies dans le Toolbox du Macintosh.

spgevent associe le contour de la region au mouvement de la souris (DragGrayRgn). DragGrayRgn rend le déplacement relatif de la souris par rapport à sa localisation au moment de l'appel. L'ancien dessin de l'exemplaire est effacé avec **spgerase**. L'effacement consiste à peindre le rectangle occupée par l'exemplaire avec la couleur du fond de la fenêtre. Le nouvel emplacement de l'exemplaire est calculé et contrôlé par la méthode **objgravity** de la classe OBJ qui règle les relations spatiales entre les objets (se reporter au chapitre 9 pour la définition du rôle de OBJ). L'exemplaire est redessiné avec **spgdraw** qui dessine l'exemplaire dans une surface virtuelle qui lui est propre. Elle signale le fait à l'arbitre OBJ grâce à la méthode **objsuperdraw**. Cette dernière, non détaillée ici, règle les problèmes de superposition : les exemplaires endommagés par l'ancienne position de l'exemplaire sont priés de se redessiner. L'ordre des messages de rafraîchissement est déterminé par l'agencement des superpositions des exemplaires, agencement connu et contrôlé par OBJ.

Cette succession de traitements qui décrit un comportement à reproduire dans chaque classe d'objets devrait être directement réutilisable depuis une boîte à outils mais, afin de répondre aux

besoins spécifiques de chaque présentation, elle devrait aussi être surchargeable par l'expression déclarative de relations spatiales. La superposition et le mouvement d'objets, qui vont de paire avec les interfaces de manipulation directe, ne sont à ce jour satisfaits simultanément par aucune boîte à outils commerciale. L'absence d'une classe de services n'est pas toujours sans incidence sur la qualité des produits résultants.

2.4.2. Interactivité et édition d'objets graphiques

Les normes graphiques, tels GKS et Core, répondent maladroitement aux besoins de l'interactivité et de l'édition d'objets graphiques parce qu'elles manipulent des unités graphiques fermées : la composition d'un segment GKS ne peut être modifiée dynamiquement par les programmes clients. La même remarque s'applique aux regions et aux pictures du Macintosh. Afin de fournir un retour d'information qui soit autre chose qu'un effet de transformations géométriques ou de coloriage, le programme client doit, à chaque modification, reconstruire un nouveau segment ou une nouvelle region. Dans ces conditions, la programmation de l'évolution dynamique des objets graphiques devient rapidement fastidieuse. Les structures de PHIGS présentées au chapitre suivant, simplifient cette tâche.

2.5. Dispositif pour la distribution

Les techniques de communication de bas niveau définies par la norme ISO sur les architectures interconnectées ouvertes sont aujourd'hui parfaitement maîtrisées. Des noyaux de systèmes répartis commencent à apparaître et à fonctionner correctement. On sait distribuer ou faire migrer sur un réseau local des entités logicielles : processus, fichiers [Jones 82, Morris 86] et autres structures complexes. Le logiciel pour l'écriture des interfaces interactives doit suivre une évolution identique. Dans ce domaine, la distribution revêt deux aspects complémentaires : le dialogue à distance mais aussi, la possibilité pour l'utilisateur de faire migrer ("Couper-Coller") de l'information entre applications.

2.5.1. Dialogue à distance : espace d'affichage distribué

Un dialogue utilise généralement une fenêtre comme support de restitution. En conséquence, un dialogue à distance exige que l'espace d'affichage du système de fenêtrage soit distribué.

Actuellement, X et NeWS sont les seuls candidats commercialement disponibles. Les boîtes à outils construites au dessus de ces systèmes et/ou qui leur servent d'interface de programmation ont le potentiel requis pour satisfaire les besoins de la distribution.

2.5.2. Migration de l'information : Couper-Coller

La mise en œuvre des fonctions Couper-Coller pose un double problème de choix : celui d mécanisme de transfert et celui du formalisme de représentation des données. Le transfert peut êtr asynchrone par l'intermédiaire d'un ou plusieurs tampons ou bien synchrone en établissant une sort de tuyau ("pipe") entre les clients producteur et consommateur. La solution asynchrone est plu simple et fait moins intervenir le système d'exploitation. Dans X, comme dans le Macintosh, o utilise la technique des tampons.

Le choix du formalisme de représentation des données peut à lui seul faire le sujet d'une recherch intéressante. En toute rigueur, le transfert des données impliquées dans une opération couper-colle relève d'une technique générale de conversion de types. Ainsi, une donnée source serai automatiquement transformée en une structure de type compatible avec ceux de l'applicatio destinataire. En l'absence d'une technique qui conserve les formalismes natifs, les données son transmises dans des formalismes proches des techniques graphiques : le pixel, l'octet, la list d'affichage ou l'image abstraite.

- Le pixel est une représentation commune à toutes les surfaces d'affichage. En conséquence, u client récepteur peut toujours accueillir des pixels en les recopiant directement dans ses surface d'affichage.
- Un liste d'affichage ou une image abstraite telle que celle que nous présentons au chapitr suivant, permettent de transmettre une information plus compacte et de communiquer à un plu haut niveau d'abstraction.

Lorsque le récepteur sait manipuler le formalisme du transfert, les structures reçues sont, en su de la restitution sur une surface d'affichage, intégrées dans l'espace de travail du récepteur L'utilisateur peut alors en manipuler les éléments. Inversement, lorsque le récepteur ne connaît pas l formalisme du transfert, les données reçues forment un corps étranger. Le récepteur accepte l'intru dans son espace de travail mais s'en protège en l'encapsulant d'un film logique qui rend le conten inaccessible à l'utilisateur. Ce changement de statut de l'information peut conduire aux incohérence évoquées au paragraphe 3.3 du chapitre 4.

En l'absence de formalisme de communication universel de haut niveau, le Toolbox du Macintosh offre la possibilité de définir de nouveaux protocoles. Il permet trois modes de transfert : texte, image et application. Dans le mode ASCII, l'information est une suite de caractères ; dans le mode graphique, l'information est une suite de primitives graphiques ; dans le dernier mode, le format est défini par le programmeur d'application. Cette dernière option ouvre la voie à la définition de protocole de plus haut niveau qui pourrait inclure comme pour les systèmes ouverts, une étape préalable d'entente statuant sur le ou les formalismes de transfert à employer.

2.6. Dispositifs pour l'information multimédia

Nous venons d'évoquer l'orientation des logiciels vers la communication. A travers les moyens de communication, toute forme d'informations doit pouvoir transiter : texte, graphique, son. Dans un avenir proche, les boîtes à outils devront étendre leur domaine de compétence. Jusqu'ici l'accent a surtout été placé sur les aspects graphique et textuel, la composante son ayant été quelque peu négligée. Bien que les résultats soient souvent spectaculaires, faire parler un Macintosh n'est pas une tâche de programmation simple!

3. Avantages et inconvénients des boîtes à outils

La notion de boîte à outils n'est pas nouvelle. Il y a longtemps que l'on développe des bibliothèques de procédures spécialisées. Comme toute bibliothèque, une boîte à outils répond à des besoins. Elle a donc ses avantages. L'expérience aidant, les besoins évoluent vers des exigences qui ne sont plus satisfaites et la boîte à outils présente des lacunes. Nous abordons dans ce paragraphe ces deux aspects : avantages et inconvénients.

3.1. Les avantages

Les avantages d'une boîte à outils se résument essentiellement à ceux d'une bibliothèque : définition d'un niveau de portabilité, extensibilité et souplesse d'utilisation. Le dernier avantage est spécifique au domaine de l'interaction homme-machine avec l'intégration, dans les abstractions, de principes ergonomiques.

3.1.1. Portabilité des logiciels interactifs

La portabilité des logiciels est un problème qui mobilise beaucoup d'énergie aussi bien dans le monde industriel que dans le milieu de la recherche. Ce problème est particulièrement aigu pour les systèmes interactifs dont l'interface homme-ordinateur représente, en volume de code, plus de la moitié du produit.

Une boîte à outils a un rôle à jouer dans la portabilité des logiciels interactifs car, en tant que bibliothèque, elle définit un niveau d'abstraction. Le niveau de portabilité d'une boîte à outils pour logiciels interactifs se situe en deux points stratégiques : au niveau du poste de travail ou au niveau

des entités de dialogue. La portabilité placée au niveau du poste de travail signifie la définition d'ur système de fenêtrage normalisé. La portabilité au niveau des entités de dialogue cache la diversité des systèmes de fenêtrage. Dans l'attente d'un système de fenêtrage normalisé, il est raisonnable de placer la frontière de portabilité au niveau des entités de dialogue. C'est le choix que nous avons effectué pour la boîte à outils PIXIA [Bernard 87b] disponible à la fois sur SunWindows et Apollo Domain.

3.1.2. Extensibilité des fonctions

Par nature, une boîte à outils peut toujours être enrichie de nouvelles fonctions ou de nouveaux objets. En particulier, si elle offre un mécanisme de construction d'objets interactifs, tel X Toolkit, le programmeur est encouragé à définir les objets adaptés à son domaine. Si en outre, la boîte à outils est intégrée à un environnement de programmation par objets, tels Interlisp-D ou Smalltalk, il est alors naturel de réutiliser des objets existants en les spécialisant.

L'extensibilité semble aller à l'encontre de la portabilité puisque chaque programmeur va potentiellement construire des boîtes à outils personnalisées. La solution à ce genre de situation devra nécessairement passer par des définitions de niveaux de portabilité comme on l'a pratiqué dans d'autres domaines.

3.1.3. Souplesse d'utilisation

Parce qu'elle offre des niveaux d'abstraction variés, la boîte à outils laisse au programmeur le choix entre le plein contrôle de l'appareil et l'utilisation de services évolués. Nous verrons que cette souplesse a sa contrepartie.

3.1.4. Intégration de critères ergonomiques

Le programmeur n'est pas forcément compétent en psychologie cognitive. L'un des attraits d'une boîte à outils est de lui proposer un éventail d'entités de présentation dont la nature et le comportement répondent à des critères ergonomiques. Concernant la nature, nous avons vu au chapitre 4 sur les principes ergonomiques que les menus, les formulaires, les boutons et autres entités de dialogue servaient avant tout de dispositifs d'extension de la mémoire à court terme.

Concernant le comportement, les choix syntaxiques et lexicaux répondent à des considérations ergonomiques les plus diverses. Par exemple, pour les boutons, leur décoration et leur forme permettent d'identifier aisément leur rôle : soit une activation de fonction, soit une spécification de paramètres désignés dans des listes de choix ; la visualisation en vidéo inverse d'un élément de menu facilite l'évaluation des actions physiques : elle confirme la localisation actuelle de la souris ; une

présentation en grisé clair informe de l'invalidité actuelle d'un concept du domaine : le grisé clair est moins visible que le corps gras. Il s'associe donc naturellement à l'état d'indisponibilité ; dans les menus, l'indication des raccourcis renseigne l'utilisateur sans imposer de surcharge cognitive : le renseignement encourage l'utilisateur à employer des méthodes plus performantes tout en respectant son rythme d'apprentissage.

Une seconde retombée de ces entités de présentation prêtes à l'emploi est qu'elles définissent un style de présentation. Les boutons, les barres de défilement, les formulaires et bien d'autres objets composés ont un comportement que l'utilisateur retrouve d'application en application. L'utilisation généralisée de ces services renforce donc la cohérence syntaxique entre applications. Cette cohérence répond aux observations ergonomiques selon lesquelles la similitude des présentations favorise le transfert des connaissances syntaxiques.

Avec des entités véhiculant des critères ergonomiques convenables, on peut espérer améliorer la qualité des interfaces. L'imitation généralisée du style Macintosh témoigne du succès de l'approche.

3.2. Les inconvénients

La souplesse de la boîte à outils a trois contreparties : risque de mauvaise décomposition modulaire, apprentissage difficile et duplication d'effort.

3.2.1. Risque de mauvaise décomposition modulaire

La boîte à outils, sans structuration, ouvre la voie à la définition d'architectures logicielles douteuses. En particulier, elle n'impose pas la séparation modulaire entre les fonctions spécifiques au domaine et les fonctions de présentation à l'utilisateur.

3.2.2. Apprentissage difficile

Le deuxième inconvénient d'une boîte à outils est la difficulté d'apprentissage. Comme le montre la figure 10.5, une boîte à outils se présente comme un grand sac de fonctions en vrac.

Les fonctions d'une boîte à outils se situent à différents niveaux d'abstraction et s'utilisent de manière mélangée. Si l'on se reporte au programme de la figure 10.2, on constate que la primitive de bas niveau, GetNextEvent, qui permet de retirer un événement de la file des événements, est suivie d'une fonction de plus haut niveau, FindWindow, fournie par le système de fenêtrage. FindWindow conduit à son tour à des entités de dialogue plus évoluées avec la primitive FindControl.

Le programmeur doit donc extraire les fonctions du grand sac et les utiliser dans un ordr convenable afin de parcourir correctement les niveaux d'abstraction. C'est ce que j'appelle détermin la colle d'assemblage. Le cheminement de l'exemple précédent est compréhensible parce qu' correspond à la méthode familière du traitement ascendant d'un problème. La difficulté pour l programmeur ne réside pas dans la compréhension du schéma général d'utilisation d'une boîte à outi mais dans la mise en place des détails. Par exemple, un point défini dans l'espace de coordonné globales doit être traduit dans l'espace de coordonnées de la fenêtre courante avant d'appeler l fonction FindControl (voir la primitive GlobalToLocal du programme de la figure 10.2).

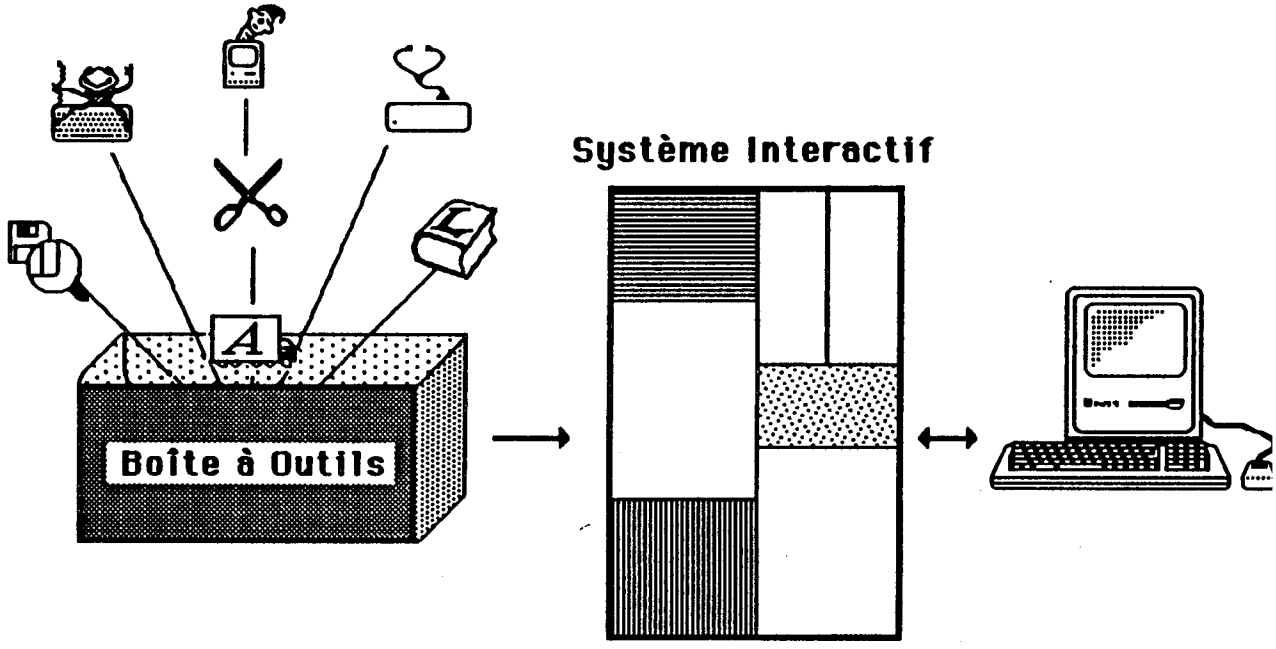


Figure 10.5 : Une boîte à outils est un grand sac de fonctions en vrac. Au programmeur de déterminer la colle.

La découverte des subtilités du mode d'emploi d'une boîte à outils est une tâche que j'ai évaluée trois mois de travail. Cette tâche est une véritable barrière cognitive qui s'amplifie avec la richesse et l diversité, pourtant souhaitables, des services et des concepts. Le succès des ouvrages expliquant l mode d'emploi des boîtes à outils avec des exemples programmés est symptomatique de la difficulté.

3.2.3. Duplication d'effort

Une fois le mode d'emploi acquis, le programmeur constate les faits suivants :

- certains traitements sont reproduits d'application en application et
- chaque application s'organise selon le même modèle de contrôle.

Ces observations suggèrent le partage ou, à défaut, la réutilisation du logiciel commun à toutes le applications. Partage et réutilisation constituent l'une des raisons d'être des squelettes d'applicatio présentés au chapitre 12.

EN RESUME :

Une boîte à outils est une bibliothèque de procédures adaptées à l'écriture d'interfaces homme-ordinateur. Elle inclut tous les niveaux fonctionnels intervenant dans un logiciel interactif : fenêtrage, affichage à différents niveaux d'abstractions, entités de dialogue. Toutes les boîtes à outils recouvrent ces aspects mais avec des différences. Les divergences se manifestent sous diverses formes :

- la stratégie de contrôle avec les notions de protocole embarqué et de protocole non embarqué. Retenons que le protocole embarqué a l'avantage de faciliter l'utilisation de la boîte à outils ;
- la possibilité de personnaliser le comportement des entités de dialogue. La personnalisation doit pouvoir s'effectuer à la fois par surcharge programmée et par l'intervention de l'utilisateur. La première condition est directement satisfaite par les boîtes à outils conçues selon le modèle de la programmation par objets ; la seconde repose sur l'existence d'une représentation externe permanente des entités de dialogue ;
- l'existence de mécanismes qui facilitent l'expression de l'interactivité, en particulier : la présence de dispositifs qui se chargent de gérer la superposition d'entités de dialogue (ceci afin de traiter les effets de bord de la manipulation directe) ; la présence d'outils qui permettent d'éditer par programme des structures graphiques (ceci afin d'alléger l'écriture des programmes à retour d'information immédiat et manipulant des objets graphiques structurés) ;
- la prise en compte de la distribution et de l'information multimédia afin de répondre à la demande d'une société qui s'oriente résolument vers la communication de toute forme d'informations.

Parmi les avantages d'une boîte à outils, nous retenons :

- l'extensibilité et la souplesse d'utilisation,
- un rôle à jouer dans la définition de niveaux de portabilité des logiciels interactifs,
- l'intégration de critères ergonomiques "prêts à l'emploi" et la définition d'un style d'interaction que l'utilisateur retrouve d'applications en applications, cohérence qui facilite le transfert de connaissance.

Parmi les inconvénients, nous notons :

- le risque d'une mauvaise décomposition modulaire qui entrave, voire rend impossible, la mise au point itérative d'un système interactif.
- un apprentissage difficile que j'évalue à trois mois avant de maîtriser les éléments essentiels : le programmeur doit trouver la bonne colle!
- la duplication d'efforts : le programmeur se voit recréer la colle pour chaque nouvelle application.

Chapitre 11

Machines à Images Abstraites

1. Introduction

Les fonctions générales de l'affichage et de l'acquisition des informations ont été présentées au chapitre 6. Nous avions alors indiqué que ces fonctions pouvaient s'effectuer à divers niveaux d'abstraction, distinguant les machines graphiques élémentaires, les machines à images simples et les machines à images abstraites. Les concepts et techniques des machines élémentaires et des machines à images simples sont aujourd'hui assez bien maîtrisés. Les machines à images abstraites font encore l'objet de recherche. L'objet de ce chapitre est d'en présenter les principes ainsi que ma contribution dans ce domaine.

Le paragraphe 2 introduit le problème fondamental : la distance entre la représentation des données internes et celle utilisée par le support de restitution. La différence entre deux représentations peut être atténuée par l'introduction d'une abstraction intermédiaire. Le paragraphe 3 présente l'application de ce procédé au cas de l'affichage. La définition d'une abstraction tient compte des propriétés dominantes des structures dont elle est l'intermédiaire. Dans le cas de l'affichage, nous observons deux tendances : l'une, tournée vers l'expression de relations structurelles, fait l'objet du paragraphe 4 ; l'autre, fondée sur l'expression de contraintes, est présentée au paragraphe 5.

2. Le problème

Les systèmes de fenêtrage permettent aux programmes clients d'effectuer des opérations de lecture et d'écriture indépendamment du fonctionnement de l'appareillage physique et sans risque de nuire à autrui. Les primitives de lecture transmettent des événements dont le niveau d'abstraction reste proche du matériel : un localisateur ou une clé, le changement de taille d'une fenêtre, etc. De même, les primitives d'écriture (ou requêtes graphiques) correspondent à un modèle d'image de bas niveau : tracé de droite ou de cercle, opérations raster, fonctions de manipulation des systèmes de coordonnées, etc. Tout au plus trouve-t-on l'encapsulation de requêtes graphiques sous forme de macro. Les notions de *path* de PostScript [Adobe 85], de *segment* dans GKS [ISO 85, Duce 87] et de *picture* dans Quickdraw [Rose 86] relèvent de cette technique.

Le niveau d'abstraction des événements, les notions de fenêtre, de droite, de cercle ou de segment conviennent à la présentation de structures internes simples. Ces abstractions, qui ne contiennent pas ou peu d'éléments de structuration, ne sont pas adaptées à la présentation de structures internes complexes comme celles que manipulent les constructeurs de documents. La difficulté de mise en correspondance concerne la restitution mais tout autant l'acquisition. Par exemple, la désignation d'un point sur l'écran peut signifier la sélection de tout un paragraphe. Par extension, toute information de

sortie doit pouvoir être utilisée comme donnée d'entrée.

L'exemple de la désignation d'un point montre que certaines actions de l'utilisateur ont des retombées sémantiques : l'utilisateur agit, par souris interposée, sur les structures logiques internes. D'autres actions, tels le défilement du contenu d'une fenêtre ou le changement de taille d'une fenêtre, sont purement syntaxiques : l'utilisateur ne cherche pas à agir sur les structures internes mais sur leur présentation. Dans le premier cas, le programme client doit effectuer la suite de transformations nécessaires à la détermination de l'entité interne. Dans le second cas, le programme client doit créer une partie de l'image afin de compléter le contenu de la fenêtre. Dans les deux cas, le client a la charge d'opérations qui ne concernent pas directement sa compétence. Cette constatation conduit naturellement à l'introduction d'une machine abstraite qui se substitue au client pour les tâches de transformations liées aux images. J'appelle cette machine, machine à images abstraites.

3. Principe de la solution : machine à images abstraites

La raison d'être d'une machine à images abstraites est de cacher, à l'aide d'abstractions, les propriétés du terminal abstrait, mais ce masquage n'est pas quelconque : afin d'être utilisables, les abstractions de la machine doivent se rapprocher des caractéristiques des structures internes à afficher. Ces abstractions dont la fonction première est l'affichage constituent une image abstraite.

3.1. Image abstraite

La figure 11.1 met en évidence le rôle de l'image abstraite. Une image abstraite est une représentation intermédiaire entre une structure interne et une image concrète. Une structure interne désigne, dans ce chapitre, la représentation d'une information à afficher et spécifique au programme client. Une image concrète est une représentation graphique pour une surface de restitution. Cette surface peut être une surface d'affichage virtuelle, une fenêtre ou, si l'on ne dispose pas de système de fenêtrage, un écran réel. L'image effectivement perçue par l'utilisateur résulte de la projection de la surface de restitution sur l'écran réel.

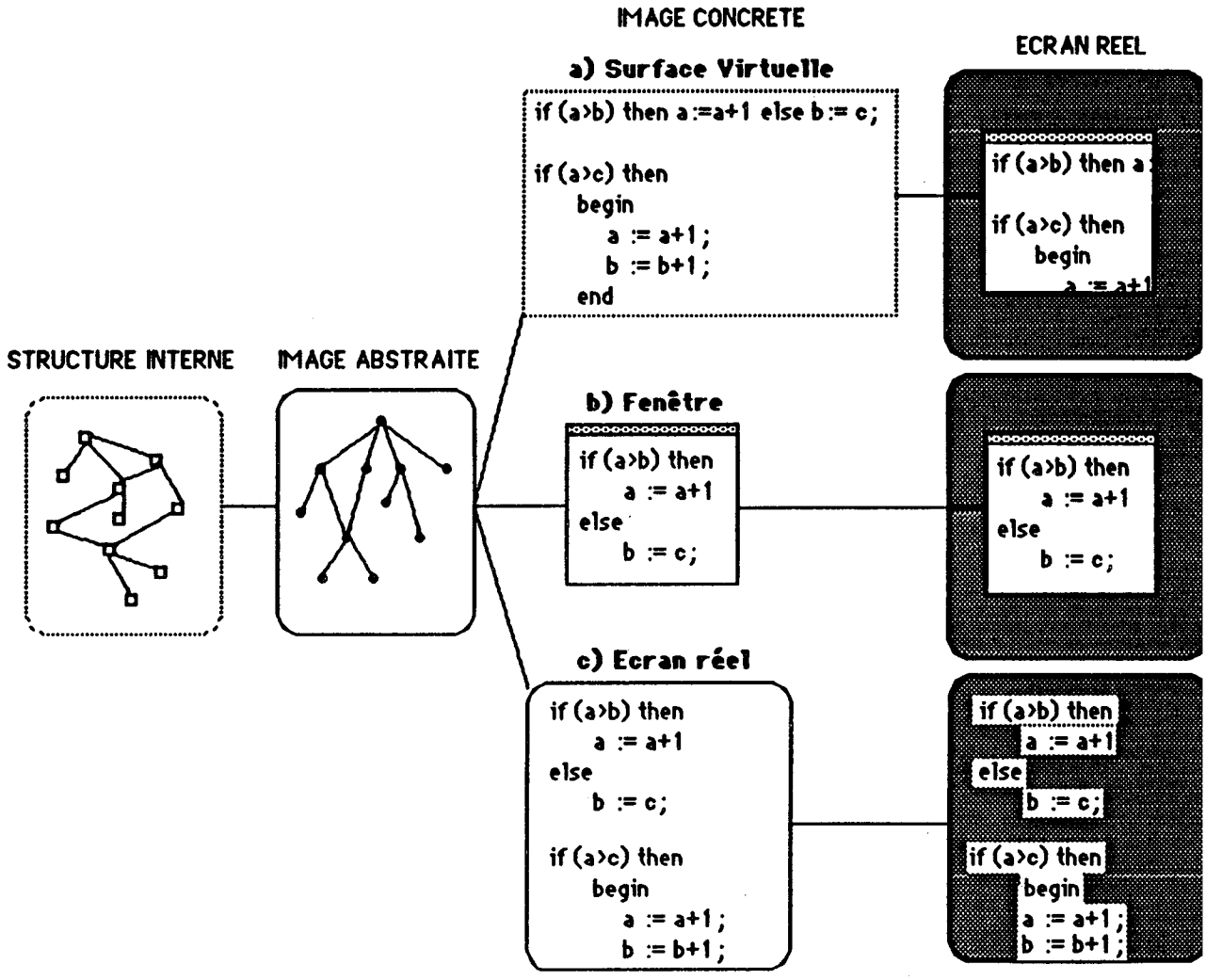


Figure 11.1 : De la structure interne à l'image concrète.

3.2. Image concrète et image perçue

Reprenons l'exemple présenté de la figure 11.1. Le programme client de la machine à image abstraites est l'éditeur syntaxique d'un langage de programmation. La structure interne est l'arbr abstrait d'un programme. L'image abstraite est produite par le client au moyen des primitives de l machine. Elle exprime la présentation du programme en termes abstraits, c'est-à-dire en terme indépendants des caractéristiques de la surface de restitution. Dans l'exemple, elle traduit le fait qu les composants d'une instruction "if" doivent être présentés sur une même ligne sauf si le support d restitution n'est pas assez large. La valeur de la caractéristique "largeur du support de restitution n'est pas câblée dans l'image abstraite mais est désignée de manière symbolique. Observons l'imag perçue par l'utilisateur selon la surface de restitution :

1. Une surface virtuelle est théoriquement de taille suffisante pour recevoir n'importe quelle imag concrète. La machine à images abstraites, interprétant la description de l'image abstraite produit une instruction "if" qui tient sur une ligne. Une part de la surface virtuelle, définie pa

la lucarne (voir chapitre 6), est ensuite projetée sur une fenêtre. En conséquence, dans l'image perçue, l'instruction "if" n'est que partiellement visible. L'utilisateur devra élargir la fenêtre ou bien pratiquer une opération de déroulement latéral pour observer la partie invisible. Notons que ces actions peuvent être traitées sans l'intervention du programme client.

2. Si la surface de restitution est une fenêtre de taille identique à celle du cas précédent, l'instruction "if" sera présentée sur plusieurs lignes. L'utilisateur n'aura pas besoin d'effectuer de déroulement horizontal. A l'inverse, le champ de vision vertical est réduit : la deuxième instruction "if", en partie visible précédemment, est maintenant masquée. Des opérations de déroulement ou de changement de taille de la fenêtre permettront de résoudre le problème.

3. Lorsque la surface de restitution est un écran réel, l'image perçue est identique à celle produite sur une fenêtre de taille constante égale à celle de l'écran.

Remarquons qu'en affectant à la surface virtuelle la largeur de la fenêtre de projection, l'utilisateur perçoit la même image dans les cas 1 et 2. Pour certaines applications de type WYSIWYG ("What You See Is What You Get" [Smith 82]), l'image concrète et l'image perçue doivent être identiques. Dans ces conditions, il faut envisager la modification dynamique de la taille de la surface virtuelle en fonction de celle de la fenêtre.

L'implémentation d'une surface virtuelle de taille variable s'effectue par spécialisation : une nouvelle classe de surface virtuelle hérite des propriétés de la surface virtuelle classique mais s'enrichit de la notion de taille logique. Cette taille définit à un instant donné, les dimensions de la surface de restitution. Elle a donc une incidence sur l'aspect de l'image concrète. Elle doit se distinguer de la notion de lucarne qui est une ouverture pratiquée sur une partie de la surface virtuelle et qui n'en modifie pas le contenu.

3.3. Relations dominantes

Du point de vue du programme client, le rôle d'une image abstraite est d'exprimer par des relations graphiques les relations logiques qui lient les éléments de la structure interne. Il ne s'agit pas de refléter toutes les relations logiques mais seulement celles qui doivent aider l'utilisateur à franchir les distances d'exécution et d'évaluation. La détermination de ces relations relève, encore une fois, de l'analyse de tâche. J'appelle relations dominantes, les relations logiques qui doivent être exprimées dans l'image abstraite.

Dans certains systèmes, tels les constructeurs de document, les relations dominantes sont de nature structurelle. Par exemple, un article est constitué de chapitres et un chapitre comprend plusieurs paragraphes, etc. Dans ce cas, l'image abstraite doit véhiculer l'idée de composition hiérarchique. Dans d'autres systèmes, tels les simulateurs, les relations dominantes sont de nature prédicative. Par exemple, la température de l'eau en ébullition est de 100°C. Dans ce cas, l'image abstraite doit

exprimer la présence de bulles lorsque la température est 100°C. Ces deux exemples montrent

- que le type d'expression demandée à l'image abstraite dépend des relations dominantes système et
- que ces relations sont trop dissemblables d'un système à l'autre pour être traduit convenablement par une même image abstraite.

L'image abstraite à usage universel étant difficile à appréhender, la définition d'images abstrait s'est naturellement tournée vers les besoins les plus pressants : l'expression des relations structurell et l'expression des contraintes. Ces deux classes sont successivement présentées dans les paragraph suivants.

4. Affichage et relations structurelles

On se sert de relations structurelles pour décrire l'assemblage d'entités entrant dans un composition. Une composition peut à son tour être un constituant dans une nouvelle constructio Cette organisation hiérarchique des concepts est l'une des façons les plus répandues de modéliser l connaissance. Il n'est donc pas surprenant que des machines à images abstraites aient été conçue dans le but de faciliter la présentation de structures internes hiérarchiques. Ces machines s répartissent en deux catégories : l'une s'appuie sur la notion de boîte, l'autre peut se voir comm l'extension des techniques graphiques présentées jusqu'ici. La première compte de nombreuse réalisations, la seconde est illustrée par la norme PHIGS [ISO 86a, Shuey 86, Shuey 87].

4.1. Images abstraites fondées sur le concept de boîte

Le concept de boîte a vu ses premiers succès avec TEX [Knuth 79], un formateur de documents Le problème à résoudre était de déterminer la position et la dimension de chacun des constituants d'u document sur le support de restitution. La solution adoptée par Knuth avec la notion de boîte, a ét reprise par d'autres éditeurs et formateurs de documents [Hibbard 84, Nanard 84, Quint 85, Quin 87]. Elle a aussi été exploitée dans le contexte général de l'affichage d'informations de environnements de programmation [Mikelsons 81, Coutaz 84b, SEMS 85, BULL 85, Alhers 86 Morcos 86a, Morcos 86b, Borras 87].

Chacune de ces réalisations utilise le même principe mais attribue à la boîte des propriété différentes nuancées par le domaine d'application. Dans ce qui suit, nous allons identifier le princip commun et l'utilité générale de la boîte. Ensuite, nous verrons un exemple d'utilisation : ma propr expérience de conception et de mise en œuvre de cette notion.

Une boîte peut se voir comme un récipient d'informations. Puisque les supports de restitution sont des espaces à deux dimensions, le récipient en question n'est pas un volume mais une surface. Cette surface est définie par un contour généralement invisible, un point de référence dans l'espace du support de restitution et un contenu. Le contenu est une information dépendante du programme client sur laquelle la boîte n'a aucun droit de regard : une chaîne de caractères, un dessin, une formule, etc.

Avec cette définition de la notion de boîte, une image concrète est obtenue en collant des boîtes sur un support de restitution. La figure 11.2 illustre le modèle. Nous observons, comme dans le cas des outils graphiques déjà présentés, une structure plate sans grand rapport avec la complexité des structures internes. Afin de traduire des relations de composition, une boîte doit pouvoir contenir d'autres boîtes. Avec cette nouvelle propriété, la boîte devient une structure hiérarchique.

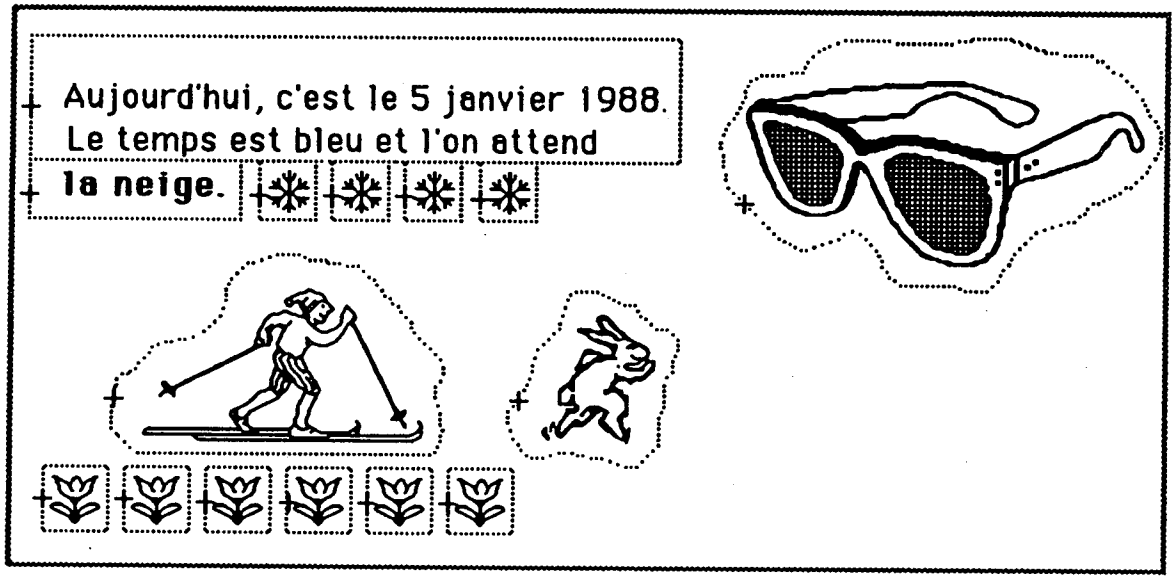


Figure 11.2 : Image concrète obtenue en collant des boîtes sur un support de restitution. En pointillé, le contour des boîtes ; le symbole + indique le point de référence de chaque boîte. On remarque que les boîtes "skieur" et "lapin" ont leur point de référence aligné.

Une représentation courante pour une structure hiérarchique est un graphe orienté sans cycle : un nœud représente une boîte composite ; une feuille est une boîte simple ; un nœud qui a plusieurs prédécesseurs traduit la présence d'une boîte dans plusieurs compositions. Une structure hiérarchique convient admirablement à la mise en place de règles d'héritage et de synthèse. Ces règles sont utiles au calcul des valeurs des attributs que l'on associe aux nœuds. Dans le cas des boîtes, les attributs servent à exprimer de manière abstraite des caractéristiques d'affichage : par exemple, une couleur, une mise en évidence ou encore une règle de placement d'une boîte par rapport à une autre. La couleur et la mise en évidence effectivement obtenues dépendront de l'appareil abstrait (c'est-à-dire du système de fenêtrage et en fin de compte du support physique) sur lequel l'image concrète sera produite.

La figure 11.3 montre l'effet de l'héritage et de la surcharge dans la détermination des attributs d boîtes. L'une des boîtes, D, a plusieurs prédécesseurs. Elle exprime la reproduction d'un information en des emplacements distincts du support de restitution. La figure 11.4 montre que l présentation finale de l'information en ces deux emplacements peut être différente.

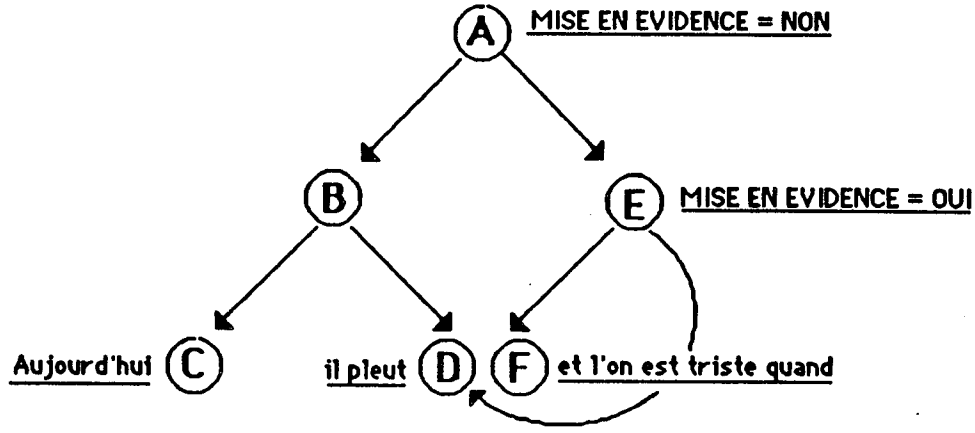


Figure 11.3 : Une structure de boîtes est un graphe orienté sans cycle ; application au calcul d'attributs de présentation.

Considérons la détermination de l'attribut "mise en évidence" des boîtes terminales sachant que l parcours du graphe s'effectue par profondeur et de gauche à droite. La boîte C est la première à êtr collée sur le support de restitution. Elle n'a pas d'attribut "mise en évidence" : elle en hérite de so prédécesseur B qui, à son tour, en a hérité de la racine. Le processus est identique pour D. L première occurrence de D hérite de B : la chaîne "il pleut" n'est pas mise en évidence. La second occurrence de D hérite de E : la chaîne "il pleut" est alors mise en évidence selon la technique d support de restitution (ici, l'utilisation de caractères gras). De la même façon, la valeur du point d référence, non précisée, est celle du précécesseur.

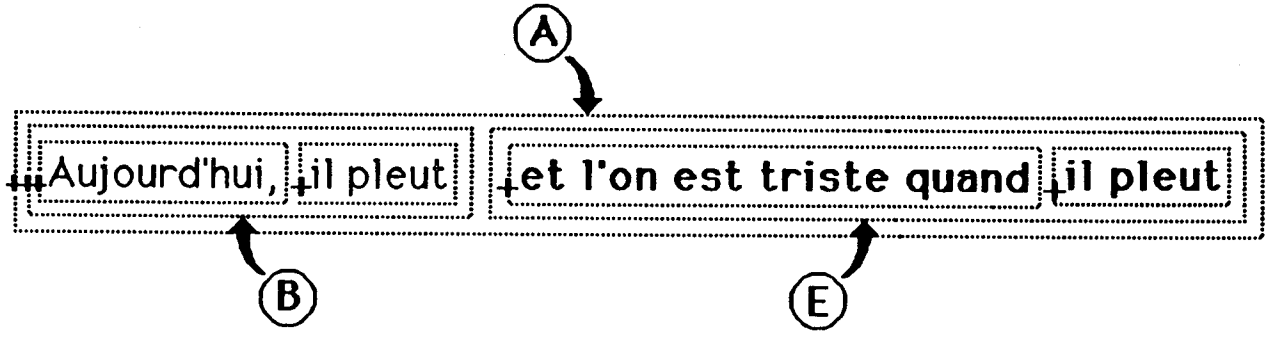


Figure 11.4 : Image concrète produite à partir de la structure de boîte de la figure 11.3. En pointillé les contours des boîtes. Les flèches désignent le contour des boîtes composites. Les points de référence des boîtes, représentés par le symbole +, sont alignés sur celui de leur prédécesseur.

Le principe général défini par un concept est souvent teinté de restrictions par sa mise en œuvre. Dans le cas du concept de boîte, les diverses réalisations citées plus haut imposent trois sortes de limitations : la forme des contours, le recouvrement de boîtes et le graphe de représentation.

1. Contrairement au principe général, le contour des boîtes n'est pas de forme quelconque mais souvent rectangulaire. L'avantage est double : les algorithmes de calcul de la surface occupée par une boîte sont plus simples et les opérations raster du support de restitution s'appliquent directement. Cependant, avec la généralisation des surfaces virtuelles de forme quelconque (voir les paths de PostScript), cet avantage s'estompe : on peut se servir de ces surfaces comme support de restitution du contenu des boîtes.

2. Par souci de simplification, le recouvrement de boîtes n'est pas toujours permis : lorsqu'une boîte change de position, les parties anciennement dissimulées doivent être repeintes. Au chapitre 6, nous avons analysé les implications logicielles de ce type de traitement : si le système de fenêtrage ne fournit pas la notion de sous-fenêtre ou de surface virtuelle, le traitement du recouvrement de boîte entraîne l'écriture d'un volume de code non négligeable.

3. Parfois, le graphe orienté de boîtes est un arbre. Cette restriction simplificatrice peut être acceptable pour certains types d'applications.

L'exemple qui suit applique ces trois restrictions : boîtes rectangulaires, pas de recouvrement et structure arborescente. Il constitue l'une des premières utilisations de la notion de boîte dans un atelier de développement de logiciel.

4.2. Un exemple d'application : les Boîtes d'Adèle

Ce travail a débuté avec le projet Adèle [Estublier 83], un atelier de développement de logiciels, effectué au Laboratoire de Génie Informatique de Grenoble. L'objectif était de permettre aux divers composants d'Adèle (l'éditeur syntaxique, le metteur au point, le gestionnaire de la base de programmes [Estubier 86] et le contrôleur du dialogue [Coutaz 84a, Coutaz 84b]) de construire des images abstraites en faisant usage du même mécanisme : la boîte.

Une représentation commune a l'avantage de servir de monnaie d'échange. Elle rend possible le transfert d'informations entre composants à condition qu'elle soit à la fois une unité d'écriture et de lecture (se reporter à la discussion sur les fonctions Couper-Coller du chapitre précédent). Les boîtes des formateurs de documents telles que celles de TEX ont été conçues pour l'affichage uniquement : ce sont des unités d'écriture. Les boîtes d'Adèle ont été conçues pour permettre à la fois les opérations d'affichage et de lecture.

Une première réalisation de ces boîtes a été effectuée en Pascal par Marc Herrmann [Herrman 84, Coutaz 84b] sur Multics pour des écrans alphanumériques. Elle a ensuite été portée, étendue et optimisée par moi-même sur un poste de travail équipé d'un écran à points (le Perq), avec Sapphire

comme système de fenêtrage [Coutaz 84a, Coutaz 85a]. Cette version a servi d'outil d'affichage pou prototyper la présentation d'un système d'aide interactif [Coutaz 84c] pour Mint [Hibbard 84], u éditeur de documents développé à l'Université de Carnegie-Mellon dans le cadre du projet Spice.

Avant de préciser la définition des boîtes d'Adèle, nous présentons l'architecture de la machine boîtes d'Adèle ainsi que ses relations opératoires avec les programmes clients. Nous verrons ensuit les propriétés marquantes de ces boîtes pour l'expression des relations spatiales, du comportemer syntaxique et des opérations de lecture. Enfin, nous verrons comment l'utilisateur peut obtenir de images concrètes multiples à partir d'une même arborescence de boîtes et comment il peut augmente son champ de vision sans agrandir le support de restitution.

4.2.1. Architecture et fonctionnement de la machine à boîtes d'Adèle

L'architecture de la machine à boîtes est illustrée par la figure 11.5. Le programme client émet se requêtes auprès du Compositeur. Celles-ci sont de deux sortes :

1. les opérations de modification de l'arborescence : par exemple, la création, la destruction, l copie, l'éclatement de boîtes ;
2. les demandes de lecture ou d'affectation d'attributs telles la couleur, la mise en évidence, l dimension, la position, etc.

Le Compositeur interprète les requêtes du client et consigne leurs effets dans l'arbre de boîtes Cette arborescence constitue une image abstraite dont l'interprétation par un Afficheur produit un image concrète sur un support de restitution. Dans la réalisation sur Sapphire, les supports d restitution sont des fenêtres et un même arbre de boîtes peut être interprété simultanément pa plusieurs afficheurs à raison d'un afficheur par fenêtre.

Un afficheur est un programme client du système de fenêtrage : il émet des requêtes graphique afin de produire ou de mettre à jour une image concrète et il reçoit les événements concernant l fenêtre, support de restitution de l'image concrète. La mise à jour de l'image concrète s'effectue dè que le compositeur signale une modification de l'arbre de boîtes ou dès qu'un événement signale une opération de rafraîchissement sur la fenêtre. L'afficheur doit donc être capable d'agir de manière incrémentale et immédiate : ne mettre à jour que ce qui est nécessaire et sans faire attendre l'utilisateur.

Les contraintes de la mise à jour incrémentale sont faciles à satisfaire lorsque la modification de l'arbre de boîtes n'entraîne pas de changement de dimension : par exemple, le passage en vidéo inverse ou le clignotement d'une boîte n'a aucune incidence sur sa taille. Connaissant la visibilité, le point de référence et la dimension de la boîte, une simple requête graphique suffit à la mise à jour de l'image concrète. A l'inverse, l'ajout ou le retrait de boîtes, le changement de police de caractères ou

de contenu remettent en cause le placement des informations dans l'image concrète. De même, lorsque l'utilisateur agrandit la fenêtre ou effectue une opération de défilement, une partie de l'image concrète doit être créée.

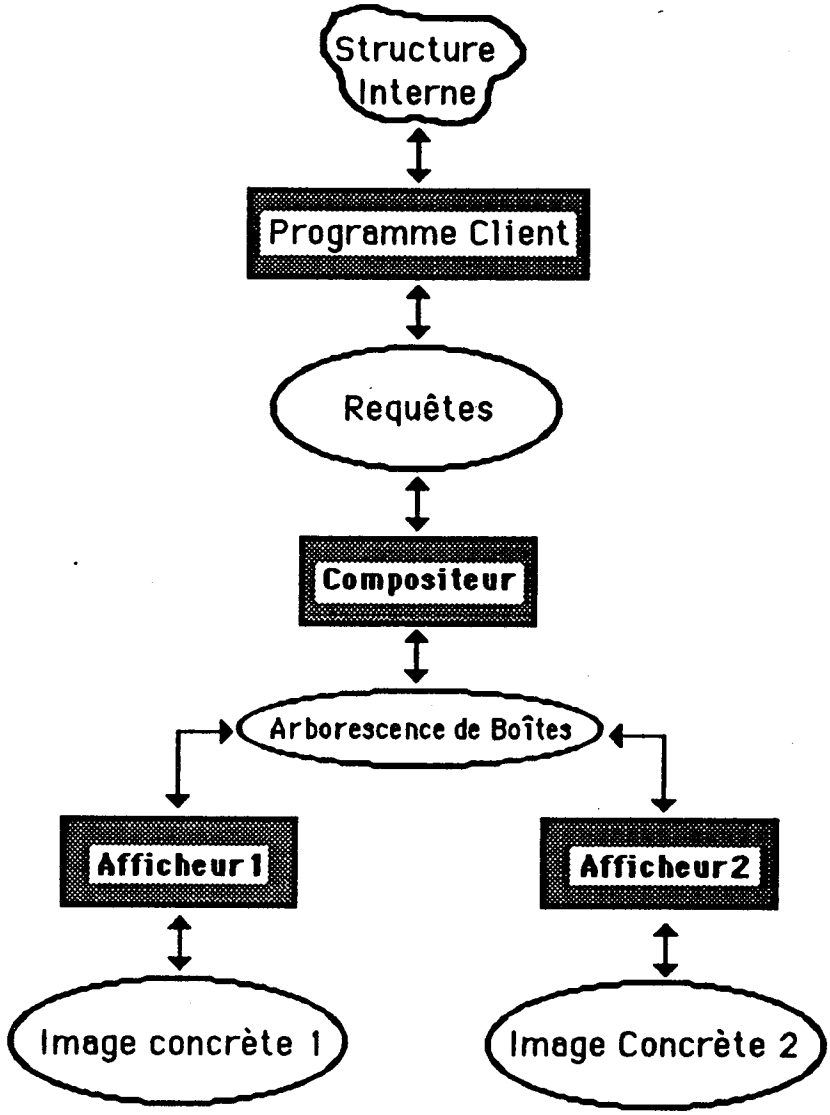


Figure 11.5 : Architecture de la machine à boîtes Adèle et ses relations avec un programme client. Les flèches indiquent l'existence d'échanges.

Il y a deux façons de résoudre ce problème : appliquer la méthode brutale ou définir une technique optimisée.

1. La méthode brutale consiste à réinterpréter l'arbre de boîtes dans sa totalité. Cette technique n'est acceptable qu'à deux conditions : l'arbre est petit et l'appareil de restitution fournit la notion de surface virtuelle.

   Un arbre petit comporte tout au plus une centaine de nœuds. Par exemple, la représentation d'un programme d'une trentaine de lignes dépasse largement ce seuil. Par contre, la représentation d'un formulaire ou celle d'un menu tient dans ces limites.

   Sans surface virtuelle, l'afficheur doit produire l'image concrète directement dans la fenêtre et l'utilisateur voit l'effet progressif des requêtes graphiques. Avec une surface virtuelle, la

construction est effectuée de manière invisible, puis projetée sur l'écran en une seule opératio raster.

2. La technique optimisée est celle que j'ai pratiquée. Son principe est indiqué dans la figure 11.6 Chaque boîte, qu'elle soit composite ou non, est repérée dans l'espace de l'image concrète pa un point de référence, une hauteur et une largeur. Ces trois informations sont calculées à parti des attributs associés aux nœuds de l'arborescence. Elles sont regroupées dans une structur appelée pavé. Ainsi, à l'arborescence de l'image abstraite correspond une structure de pavé imbriqués qui rapproche l'arbre de boîtes de l'image concrète. Cette structure permet en effet d déterminer le point d'ancrage du contenu des boîtes élémentaires sur la surface de restitution.

   L'adresse du point de référence d'un pavé est un couple (h, v) qui désigne un déplacemen horizontal et un déplacement vertical relatifs au point de référence de la racine. La partie visibl de l'ensemble des pavés est lui-même un pavé d'adresse relative (H,V) et de dimension égale à celle de la fenêtre. Ce pavé est une lucarne. Voyons maintenant l'utilité de ces structures d données pour la mise à jour incrémentale de l'image concrète.

   Toute modification d'attribut d'un nœud ayant une incidence sur la taille ou toute modificatio de la structure en un nœud entraîne le recalcul des pavés et sous-pavés dépendants. Pa exemple, dans la figure 11.6, le changement de police de caractères dans la boîte G a comm effet de bord le changement de taille du pavé de G, donc un changement de taille de prédécesseurs F et A et un changement du point de référence des successeurs H et I. Si la parti affectée est invisible, c'est-à-dire hors de la lucarne, la mise à jour de l'image concrète n'a pa lieu. Dans le cas contraire, seuls les pavés affectés sont rafraîchis. Afin de parfaire la qualit visuelle du rafraîchissement, les pavés sont recréés hors écran puis projetés sur l'écran à travers une zone de coupure égale à la partie recréée.

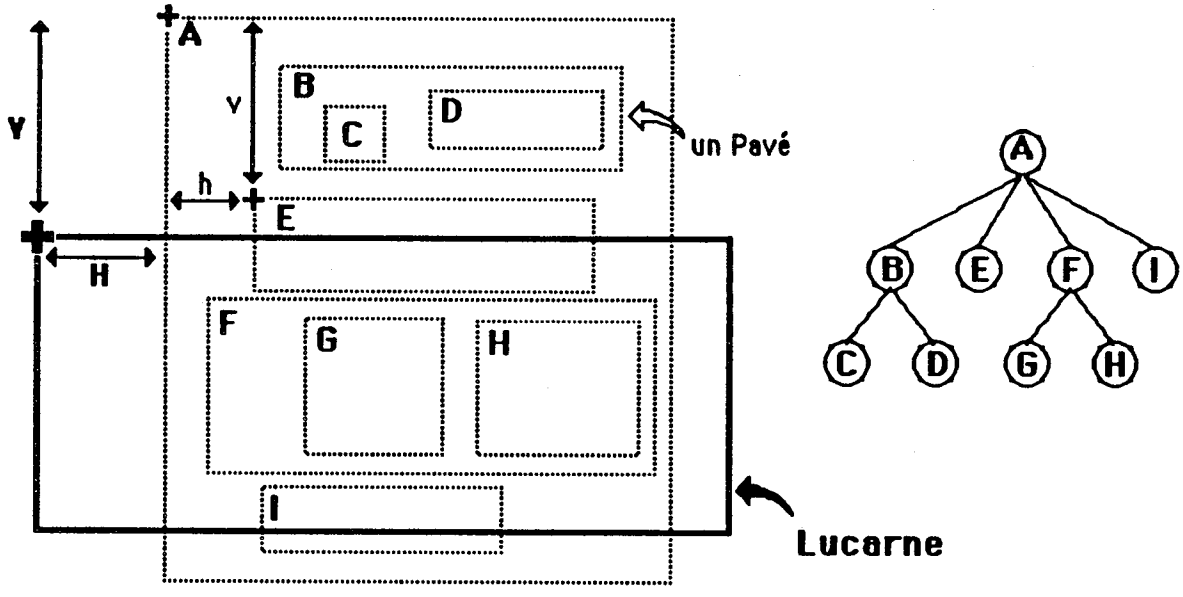


Figure 11.6 : Arbre de boîtes et sa structure de pavés correspondante. En pointillé, le contour des pavés. En trait épais, la lucarne actuelle. Le symbole + désigne un point de référence.

Cette technique de mise à jour incrémentale ne résoud pas tous les problèmes. En particulier, la double présence de l'arbre de boîtes et des structures de pavés peut conduire à un blocage par carence de mémoire centrale. L'économie peut s'effectuer à deux niveaux : ou bien l'afficheur ne conserve qu'une partie des pavés de l'arborescence de boîtes ou bien le client ne produit qu'une partie de l'image abstraite. La machine à boîtes d'Adèle ne fait aucune hypothèse sur le comportement du

programme client : pour chaque fenêtre elle maintient une partie des pavés mais signale au programme client le besoin en boîtes.

Les pavés mémorisés se comportent un peu à la manière des ensembles de travail (working sets) des systèmes de pagination : ils représentent les boîtes qui sont ou viennent d'être visualisées. A la différence des systèmes de pagination, les pavés mémorisés sont contigus et remplissent une surface dont la taille est un multiple de celle du support de restitution. Ce multiple est un paramètre de l'afficheur. Un ensemble de travail contigu est plus simple à réaliser : les algorithmes de gestion échappent au recollage des morceaux. Il se justifie aussi par son type d'utilisation : les pavés mémorisés sont utilisés pour boucher les trous qu'entraînent les opérations de défilement et d'agrandissement de fenêtre. Pour ces deux types d'opérations, les trous sont contigus à l'information actuellement visible.

Si un trou ne peut être rempli par un pavé de l'ensemble de travail, il y a défaut de pavé. Les pavés nécessaires sont créés à partir de l'arbre de boîtes. S'il n'y a pas de boîte (cas où l'utilisateur déroule la fenêtre déjà placée sur la fin du document), l'Afficheur en avertit le Compositeur qui signale le défaut de boîte au client. Ce dernier peut alors créer l'image abstraite manquante ou ignorer le message.

4.2.2. Définition d'un arbre de boîtes Adèle

La justification première d'une image abstraite hiérarchique est l'expression de relations structurelles entre les constituants de la structure interne. Par exemple, dans l'arbre abstrait d'un éditeur structuré, une instruction IF comprend une CONDITION et les parties THEN et ELSE. L'image abstraite correspondant à cette instruction doit pouvoir refléter cette composition. Avec cet objectif comme élément directeur, une arborescence de boîtes se définit ainsi :

- une feuille est une boîte simple. Elle contient une information typée (textuelle ou graphique) entourée d'un rectangle imaginaire. La dimension du rectangle est déterminée par la taille du contenu ou bien peut être contrainte par le programme client. Le remplissage du rectangle est effectué par un outil spécialisé dépendant du type du contenu ;

- un nœud est une boîte composite, résultat de la composition de boîtes filles. Un rectangle imaginaire englobe les rectangles de la descendance ;

- chaque boîte, composite ou non, est décorée d'attributs héritables ou synthétisés qui permettent l'expression de relations spatiales, l'affectation de propriétés syntaxiques, la réalisation d'opérations de lecture, l'obtention d'images concrètes multiples et l'optimisation de la surface de l'écran.

4.2.3. Expression des relations spatiales

Des attributs de formatage assurent l'expression des relations spatiales. Ce sont les attributs c composition, d'espacement, de retrait, et de contrainte sur les dimensions des rectangles.

L'attribut de composition joue un rôle privilégié. Il prend ses valeurs dans l'ensemble : **{H, V HouV, HetV}**.

- **H** et **V** concatènent respectivement horizontalement et verticalement les rectangles des boît filles. Avec la composition **H**, l'information qui s'épanche au delà de la lucarne existe mais e masquée.
- **HouV** tente une composition horizontale. Si le rectangle résultant dépasse le bord droit de lucarne, une composition verticale est appliquée.
- **HetV** fonctionne comme le passage à la ligne des traitements de texte : les boîtes filles so composées horizontalement jusqu'à violation du bord droit de la lucarne ; un repliement vertic est alors effectué et la composition **HetV** reprend.

L'exemple suivant montre comment un programme client peut tenir compte de la taille des fenêtr de manière abstraite et dynamique. La figure 11.7 présente l'arbre de boîtes d'une instruction I Selon la largeur de la fenêtre, un afficheur produit les images de la figure 11.8 ou de la figure 11.9. cette largeur est modifiée, le passage d'une image concrète à l'autre a lieu automatiquement sa même solliciter l'intervention du programme client : l'image abstraite filtre les actions syntaxiques d l'utilisateur qui ne relèvent pas des compétences du client.

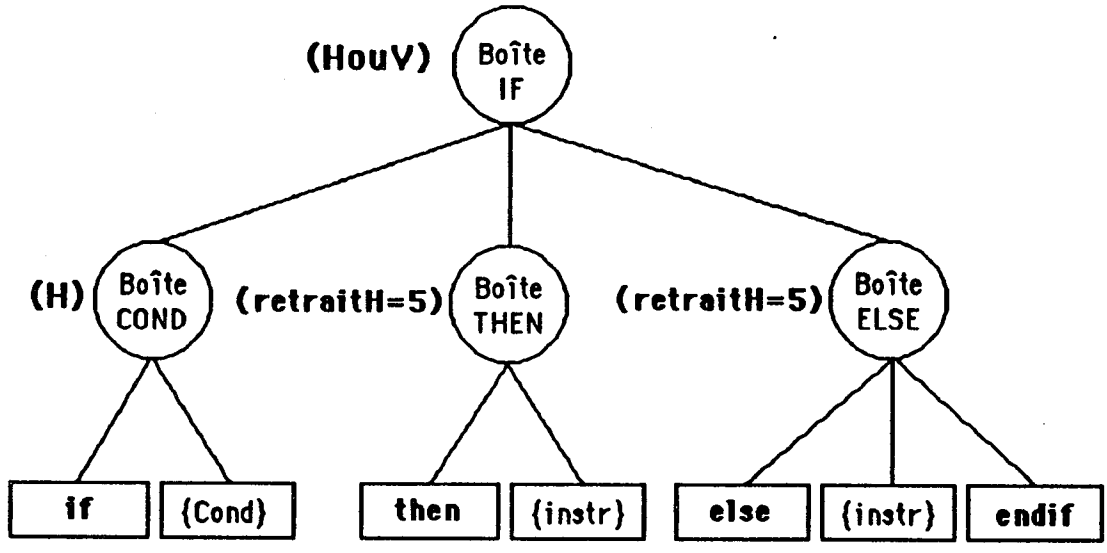


Figure 11.7 : Un arbre de boîtes pour présenter l'instruction IF.

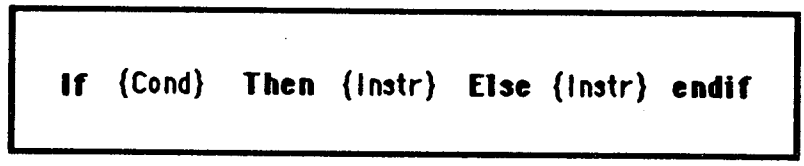


Figure 11.8 : L'image concrète produite lorsque la fenêtre est suffisamment large.

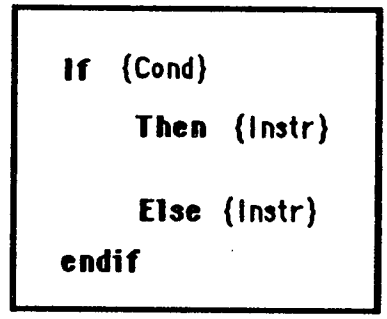


Figure 11.9 : L'image concrète produite lorsque la fenêtre de restitution est étroite.

L'expression des relations spatiales ne constitue qu'une partie des besoins. Les programmes clients cherchent aussi à fournir une forme de retour d'information en jouant sur des effets graphiques tels que le corps des caractères, le soulignement, le clignotement, la forme du curseur, la couleur, etc.

4.2.4. Expression des effets graphiques

Les effets graphiques sont véhiculés par les attributs dits "décoratifs" des boîtes. L'exemple des figures 11.10 et 11.11 montre comment. L'arbre de boîtes présenté dans la première figure est l'image abstraite d'un menu déroulant du système d'aide pour Mint [Coutaz 84c]. Dans ce système, un menu présente une liste de termes sur lesquels il est possible d'obtenir des renseignements. Pour chaque terme, les renseignements peuvent être détaillés ou au contraire succints avec des exemples typiques d'utilisation. Considérons maintenant la présentation de ces deux options. La difficulté est de signifier à l'utilisateur l'existence des deux formes de renseignements sans le coincer dans un mode implicite (mode détail et mode résumé) et sans lui imposer d'actions physiques supplémentaires (par exemple, la sélection d'un bouton d'options).

La solution choisie repose sur le changement de forme dynamique du curseur selon sa position : lorsque le curseur est sur la moitié gauche d'un menu, il prend automatiquement la forme du texte **Résumé** entouré d'un cadre fléché. Lorsqu'il est sur la moitié droite, le texte est remplacé par **Détails**. Cette idée est appliquée systématiquement à toutes les régions du menu afin d'expliciter la sémantique de chaque région par une présentation appropriée du curseur. Dans l'image concrète de la figure 11.11, on trouvera les formes possibles du curseur : une flèche verticale dirigée vers le haut pour la

barre de défilement vers le haut, une flèche orientée vers le bas pour la barre de défilement vers le b
les options Résumé et Détails. Grâce à la technique du changement de curseur, l'utilisateur sélection
en toute connaissance de cause, sans mode et en un seul clic, le sujet et la forme du renseignemen
clic à gauche pour des renseignements résumés et clic à droite pour des renseignements détaillés.

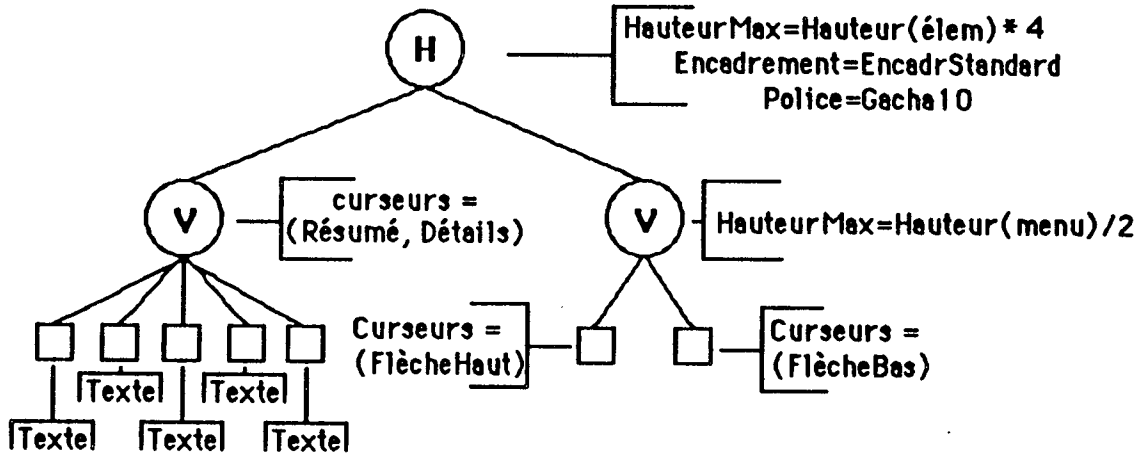


Figure 11.10 : Un arbre de boîtes spécifiant la présentation du menu de la figure 11.11.

Le comportement du menu décrit par l'arbre de boîtes de la figure 11.11 est géré par la machine
boîtes. Cet arbre indique que l'image concrète doit comprendre deux pavés concaténé
horizontalement : l'un pour contenir les éléments du menu, le second pour servir de barre d
défilement vertical. La valeur de l'attribut curseur du sous-arbre des éléments du menu est une list
désignant deux ressources de type curseur. Cet attribut est hérité par les boîtes terminales de typ
texte. La racine de l'arbre limite la taille du menu à quatre éléments, propose un encadrement standar
et la police Gacha10. Ces deux derniers attributs, non spécifiés dans les sous-arbres, sont hérités pa
toutes les boîtes.

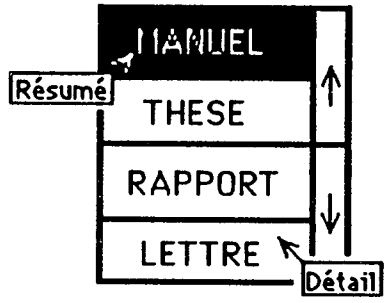


Figure 11.11 : L'image concrète de l'arbre de la figure 11.10 où sont présentées toutes les formes de curseur. En réalité, l'utilisateur est en train de sélectionner des renseignements résumés sur le format Mint d'un Manuel.

Lorsque l'utilisateur désigne un point d'une boîte, comment retrouve-t-on la structure intern
qu'elle présente?

4.2.5. Opération de lecture : d'un point de l'écran à la structure interne

Le passage d'un point de l'écran à une structure interne s'effectue par l'intermédiaire de l'attribut "lien". Comme pour les autres attributs, le lien doit être affecté par le programme client. A la réception d'une requête de la forme : "quel objet correspond à (x,y)?", un afficheur recherche la boîte feuille propriétaire du point. Ce point est un emplacement spécifié sous forme de coordonnées relatives à la fenêtre. On rappelle que la fenêtre est modélisée par une lucarne dont l'afficheur maintient la position relative par rapport au pavé racine. Connaissant cette position, celle du point et celles de chaque pavé, il est facile de déterminer le pavé puis la boîte feuille propriétaire du point.

L'afficheur évalue ensuite l'attribut lien de la feuille en appliquant les règles d'héritage : il remonte dans l'arborescence jusqu'à la rencontre d'un nœud dont l'attribut lien a été affecté d'une valeur. Ce nœud est la racine d'un sous-arbre représentant la structure interne recherchée. Le mécanisme laisse au programme client le droit d'ajuster dynamiquement la granularité des désignations. Par exemple, un éditeur syntaxique décidant que toute sélection de point dans une instruction IF a pour résultat la désignation de l'instruction toute entière met la boîte "if" de la figure 11.7 en correspondance avec le nœud "if" de l'arbre syntaxique sans spécifier d'attribut lien pour les autres sous-boîtes.

Jusqu'ici, nous avons surtout vu l'intérêt des boîtes en prenant le point de vue du programmeur d'application. Qu'en est-il de l'utilisateur? Les boîtes d'Adèle offrent la possibilité de vues multiples d'un même concept et proposent une façon originale d'optimiser l'espace écran.

4.2.6. Images concrètes multiples

L'utilisateur a parfois besoin d'observer un objet sous des angles différents. Ce désir dépend de la tâche, de l'intérêt porté à l'objet ou bien de la surface disponible sur l'écran (se reporter au chapitre 4). Il existe deux façons courantes d'obtenir des vues multiples. La première, l'élision, sert à contracter ou étendre la représentation d'un objet. Par exemple, l'utilisateur d'un éditeur de documents affiche le document sous la forme réduite d'une table des matières, puis demande l'expansion du chapitre qui l'intéresse. La seconde façon consiste à associer à l'objet plusieurs images abstraites. Par exemple, l'objet maison peut être vu comme le mot "maison", le dessin ou le plan d'une maison.

Le service "vues multiples" de la boîte repose sur le principe de l'élision [Mikelsons 81]. Selon ce principe, il est possible de produire des images concrètes distinctes à partir du même arbre de boîtes en ignorant les boîtes qui ne respectent pas des conditions précises. L'une des contraintes les plus fréquentes concerne la profondeur d'une boîte. En jouant sur l'expression de la contrainte et sur la

valeur d'un seuil, on obtient aisément des effets de zoom intéressants. Par exemple, pour obtenir le grandes lignes d'une composition, la contrainte sera de la forme "profondeur < seuil" ; à l'inverse pour supprimer les grandes lignes mais conserver, au cœur de la fenêtre, les détails qui constituent l centre d'intérêt de l'utilisateur, la contrainte sera de la forme "profondeur > seuil". Le client a l possibilité de proposer une boîte de remplacement pour les boîtes rejetées par l'afficheur. Pa exemple, un éditeur structuré peut proposer comme boîte de remplacement une boîte contenant de points de suspension.

4.2.7. Optimisation de la surface écran

Du point de vue de l'utilisateur, l'écran est toujours trop petit. Les gestionnaires de fenêtres (ave les recouvrements, les modifications de taille et de localisation de fenêtres) ne résolvent le problèm que partiellemment : ils ne manipulent que le support de l'information. L'information elle-même doi pouvoir contourner les limitations du matériel. Le défilement et l'élision constituent deux procédé usuels. Toutefois, le défilement ne convient pas aux tâches qui nécessitent l'affichage simultané d parties distantes d'un même objet. Par exemple, le programmeur qui écrit le corps d'une procédure, besoin d'avoir simultanément accès à la partie déclarative de la procédure et éventuellement à celle d module. Les va-et-vient entre diverses parties non simultanément visibles gênent la productivité.

Afin d'éviter les défilements, l'utilisateur peut éclater la représentation d'un objet dans plusieur fenêtres comme on le pratique couramment avec Emacs [Stallman 79]. Avec une structur arborescente comme celle des boîtes, cet éclatement est immédiat : il suffit d'associer les sous-arbre pertinents à des fenêtres distinctes. Cet éparpillement a cependant l'inconvénient de répartir sur l'écra des informations logiquement connectées. Pour éviter cela, les boîtes Adèle permettent de faire défile localement des sous-objets (par exemple, le corps d'une procédure) sans toucher à l'enveloppe d l'objet englobant (par exemple la procédure elle-même). Alors, l'utilisateur déroule le corps de l procédure tout en laissant la partie déclarative fixe au sommet de la fenêtre. Ce type de défilement es possible grâce à une hiérarchie de "Boîtes-Racines".

Une Boîte-Racine est la racine d'un arbre (ou sous-arbre) qui possède au moins un afficheu privé. Cet afficheur crée et gère l'image concrète à l'intérieur du rectangle imaginaire correspondant à l'arbre (ou au sous-arbre). Tout nœud peut, sur demande du programme client et à tout instant, êtr déclaré Boîte-Racine. Dans ces conditions, l'image concrète d'un arbre est produite et gérée par une hiérarchie d'Afficheurs. Chacun de ces Afficheurs a les propriétés déjà décrites et évolue indépendamment des autres.

La figure 11.12 illustre l'effet des Boîtes-Racines hiérarchiques. Dans cet exemple, la Boîte-Racine principale décrit un formulaire. Elle comprend trois Boîtes-Racines : la première (en hau

à gauche) contient l'information résultat de la composition de boîtes textuelles et graphiques ; la seconde (en haut à droite) définit un menu ; la dernière (en bas) représente un atelier comprenant trois champs numériques. Chaque champ est une Boîte-Racine décorée de l'attribut de composition V, d'une hauteur contrainte (en sorte qu'une seule valeur du sous-arbre soit visible à un instant donné), du suiveur **CliquezIci** et d'un ensemble dynamique de feuilles contenant les valeurs numériques. A tout instant et dans un ordre quelconque, l'utilisateur peut faire défiler le formulaire tout entier, faire défiler chacune des trois parties, désigner un champ (pour obtenir la valeur suivante) ou encore modifier la taille des rectangles des Boîtes-Racines (puisque les caractéristiques d'une boîte sont modifiables dynamiquement).

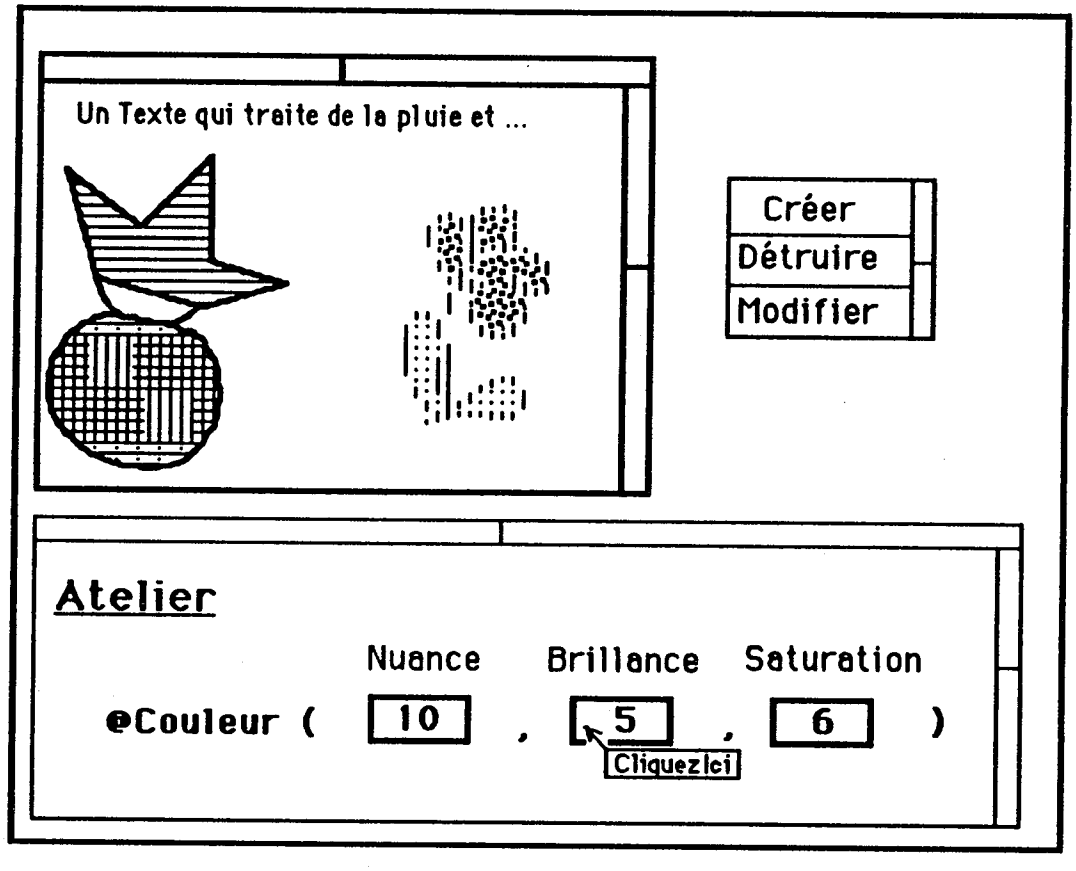


Figure 11.12 : Un formulaire défini par une hiérarchie de Boîtes-Racines.

Cet exemple montre que du point de vue de l'utilisateur, une hiérarchie d'afficheurs se comporte comme une hiérarchie de fenêtres, la lucarne d'un sous-afficheur correspondant à une sous-fenêtre. Du point de vue du programmeur, la notion d'afficheur hiérarchique permet de réaliser du fenêtrage hiérarchique avec en plus

- l'expression et la gestion automatique des relations spatiales et structurelles entre fenêtres,
- le rafraîchissement automatique des fenêtres (déroulement, agrandissement, changement d'exposition),

- l'attachement à chaque fenêtre d'une image abstraite au lieu d'une simple surface virtuelle d pixels.

Une hiérarchie d'afficheurs telle que celle présentée ici a un double avantage :

- elle répond à une classe de problèmes que l'on retrouve à plusieurs niveaux d'abstractions : a niveau des fenêtres et à tous les niveaux de structuration du contenu d'une fenêtre. Ce problèmes concernent tous les aspects de la gestion de surfaces d'affichage : relations d composition, déroulement, superposition, rafraîchissement, etc.

- elle ne coûte pas plus cher à mettre en œuvre qu'un afficheur "plat" : il suffit que la procédur d'interprétation d'un arbre de boîtes soit récursive.

Au moment où la notion de fenêtre hiérarchique commençait à faire une apparition timide, ce typ d'outil offrait un niveau d'abstraction intéressant.

4.2.8. Récapitulation

La boîte Adèle est une image abstraite récursive.

- Elle permet des échanges à un haut niveau d'abstraction : en entrée, elle fournit l'identité d'un structure interne ; en sortie, elle produit l'image concrète d'une structure interne.

- Elle filtre les actions de l'utilisateur qui ne relèvent pas de la compétence du programm d'application : changement de taille ou d'exposition d'une fenêtre, obtention de vues multiples déroulements à divers niveaux de granularité.

Du côté des insuffisances, nous avons cité la non superposition de boîtes, mais il manque auss un habillage du langage de la machine à boîtes. Pour l'instant, l'interface est un ensemble d primitives. Afin qu'un tel outil soit utilisable à moindre effort, il est indispensable de l'habiller d'ur langage adapté à l'expression de la composition d'images. Le langage P développé par Vincent Quin pour Grif [Quint 87] est un bon exemple ; mais il faut aller plus loin : remplacer les langages textuel non interactifs compilés par des langages graphiques interactifs interprétés (mais compilables). Dan ces conditions, le programmeur d'application décrit des schémas de présentation interactivement schémas qui se trouvent générer ensuite des arbres de boîtes.

4.3. PHIGS

PHIGS est un système graphique normalisé. Ses concepts s'inspirent largement de ceux de sor prédécesseur, GKS : similitude des requêtes et des attributs graphiques, notion identique de station de travail et modèle des entrées similaire. A l'organisation plate des segments GKS qui gêne la correspondance entre structures internes et images abstraites, PHIGS substitue des structures hiérarchiques. Aux segments statiques de GKS qui contrarient les besoins de l'interactivité, PHIGS

oppose des structures dynamiques.

4.3.1. Structure hiérarchique

Une *structure* PHIGS est constituée d'éléments. Un élément de structure peut être :

- une primitive graphique : par exemple, un tracé de droite ou l'affichage d'un texte ;
- une affectation d'attributs. Ceux-ci se répartissent en quatre catégories, parmi lesquelles :
  - les attributs géométriques qui contrôlent la forme et la dimension des primitives graphiques : par exemple, le style de droite, la hauteur des caractères, l'épaisseur du pinceau,
  - les attributs non géométriques qui affectent l'apparence des primitives graphiques : par exemple, un indice de couleur.
- une primitive qui exprime la composition de structures : "EXECUTE_STRUCTURE (nom-d-une-structure)".

Cette description des éléments d'une structure est incomplète mais suffit à saisir le fonctionnement de la machine PHIGS : pour qu'une structure soit visible à l'écran, le programme client doit émettre la requête "POST_STRUCTURE (nom-d-une-station-de-travail, nom-d-une-structure)". Toute structure a un nom défini par le client. La requête POST_STRUCTURE lance l'interprétation du contenu de "nom-d-une-structure". A la rencontre d'un élément "EXECUTE_STRUCTURE (nom-d-une-autre-structure)", la machine procède comme le fait une machine pascal pour un appel de procédure :

1. l'interprétation du contenu de la structure courante est suspendue,
2. les valeurs des attributs sont sauvegardées,
3. le contenu de "nom-d-une-autre-structure" est interprété en héritant les valeurs des attributs de la structure mère,
4. les valeurs des attributs de la structure mère sont restaurées, et
5. l'interprétation de la structure mère reprend.

L'élément EXECUTE_STRUCTURE exprime la relation de composition de structures. PHIGS ne limite pas la profondeur des compositions mais n'autorise pas de cycle. Comme le montre la figure 11.13, une structure PHIGS est un graphe orienté acyclique. La composition hiérarchique de structures n'est pas seulement gérée pour les sorties. Elle l'est également pour les entrées.

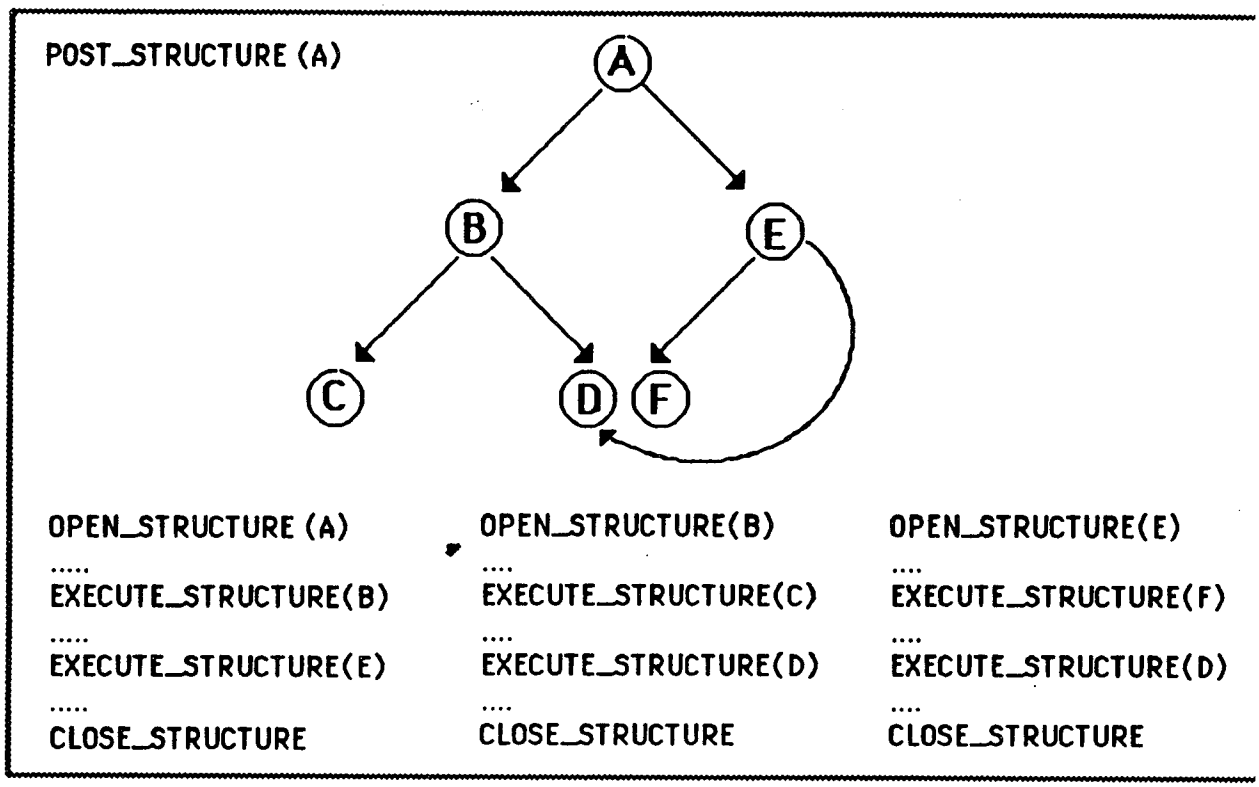


Figure 11.13 : Une structure PHIGS est un graphe orienté acyclique.

Pour les entrées, PHIGS reprend le modèle des unités logiques de GKS mais l'étend pour teni compte des structures hiérarchiques. Par exemple, un localisateur (PICK) rend un chemin qui désign de manière unique l'élément sélectionné. Ce chemin comprend, pour chaque niveau hiérarchique, l nom de la structure et le numéro de l'élément. (Un élément est repéré dans une structure par u numéro d'ordre.) A la demande du client, une valeur de type PICK exprime un chemin e commençant soit par le sommet de la hiérarchie (c'est-à-dire par le nom de la structure sur laquelle o a appliqué la requête POST_STRUCTURE), soit en commençant par le bas de la hiérarchi (c'est-à-dire par le numéro de l'élément désigné).

A elle seule, la composition ne suffit pas à exprimer toutes les formes de relations entre primitive graphiques. Pour satisfaire ce besoin de généralité, PHIGS utilise la notion d'ensemble. Un ensembl regroupe des primitives qui n'appartiennent pas forcément à la même structure mais qui sont reliée par simple décision du programme client. Son contenu évolue avec l'insertion ou le retrait d primitives. Ces ensembles sont référencés dans la spécification des filtres. Les filtres sont attachés à l station de travail. Ils sont modifiables dynamiquement et concernent la visibilité, la mise en évidenc et la détectabilité. Ainsi, tout ensemble E inclus dans le filtre de mise en évidence a comme effet l mise en évidence des primitives graphiques appartenant à E. De même, pour empêcher la sélection d toutes les fenêtres d'une maison, il suffit que les primitives graphiques qui correspondent aux fenêtre soient liées à un ensemble exclu du filtre de détectabilité.

4.3.2. Structure dynamique

PHIGS répond aux besoins de l'interactivité de deux façons complémentaires : en permettant l'édition des structures et en évaluant les attributs dynamiquement. Editer une structure c'est en modifier le contenu. Par exemple, le programme client peut ajouter, supprimer des requêtes graphiques ou encore changer la valeur des attributs. Comme pour les boîtes, toute modification d'un attribut hérité est dynamiquement propagée à la descendance.

L'édition d'une structure reprend le modèle de l'édition de texte ligne à ligne : à la notion de fichier correspond celle de structure et à la notion de ligne courante correspond celle d'élément courant. Comme dans les éditeurs de texte, il faut commencer par ouvrir le récipient : "OPEN_STRUCTURE (nom-d-une-structure)". A ce moment-là, la machine PHIGS place le pointeur sur le dernier élément et se met en mode d'insertion : les primitives graphiques qui suivent la requête OPEN_STRUCTURE sont ajoutées comme éléments en fin de structure. L'édition s'arrête à l'exécution d'une requête CLOSE-STRUCTURE ; mais une modification n'a pas toujours lieu en fin de structure.

Comme dans les éditeurs ligne à ligne, le pointeur peut être déplacé : SET-ELEMENT-POINTER (numéro-d-ordre-d-un-élément) place le pointeur sur l'élément ayant le numéro d'ordre spécifié. L'édition peut s'effectuer selon deux modes, insertion et remplacement, laissés au choix du programme client. La figure 11.14 donne un exemple d'édition d'une structure. Le nom de la structure et l'identité de l'élément nous viennent, par exemple, d'un chemin retourné par une unité logique PICK, à la suite d'une sélection de l'utilisateur. L'édition commence par supprimer l'élément graphique étiqueté MYWINDOW. (Un élément LABEL n'a pas d'incidence sur l'organisation de la structure. Il sert à repérer symboliquement un autre élément de structure ; ici, la primitive graphique qui dessine une fenêtre.) Puis, le pointeur d'éléments est placé sur l'élément MYDOOR qui est remplacé par d'autres éléments (non détaillés dans la figure).

L'édition des structures et l'évaluation dynamique des attributs facilitent la vie des programmes clients lorsqu'il s'agit de jouer finement sur le retour d'information. Pour en apprécier l'intérêt, procédons par comparaison avec GKS :

- une fois créé, le contenu d'un segment GKS ne peut être modifié. Seuls sont modifiables les attributs qui agissent sur la totalité du segment comme la visibilité, la mise en évidence, la détectabilité et la transformation géométrique globale. Dans PHIGS, tout composant est susceptible d'être modifié à tout instant.

- GKS évalue les attributs des primitives graphiques à la création du segment. En conséquence, si la couleur d'une primitive doit être modifiée, il faut le détruire puis le recréer avec l'indice de couleur voulue.

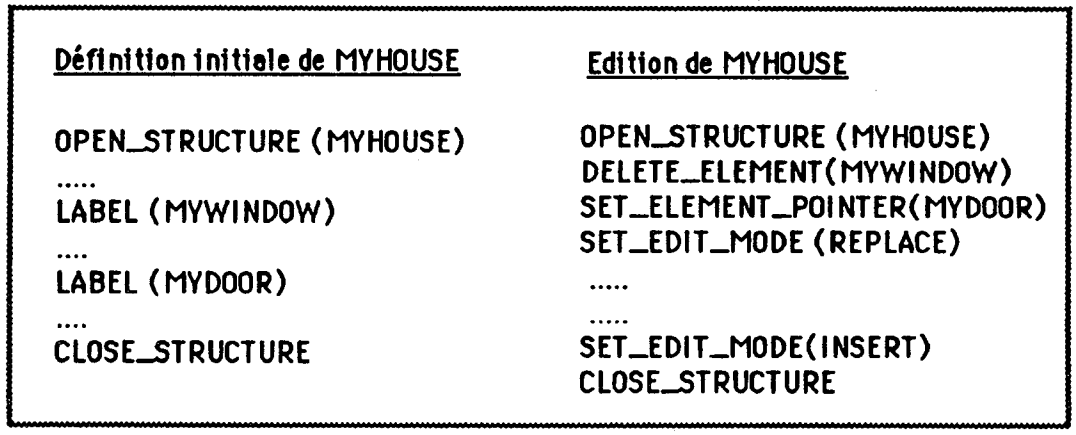


Figure 11.14 : Edition d'une structure PHIGS.

Le segment GKS est une structure plate et statique qui s'emploie comme un tout. La structure PHIGS est une organisation hiérarchique et dynamique, modifiable au niveau de l'élément. Le segment se comporte comme un symbole mort alors que la structure PHIGS est une matrice évolutive. Le premier correspond mal aux besoins dynamiques de l'interaction, le second, comme les boîtes, s'en rapproche.

En résumé, l'intérêt de PHIGS ne provient pas de la puissance ni de l'originalité de ses primitives mais de la façon dont elles sont organisées et de la façon dont le programme client peut les manipuler.

5. Affichage et Systèmes à Contraintes

Une contrainte décrit une relation qui doit être satisfaite en permanence. Les relations gérées par les machines à images abstraites du paragraphe précédent correspondent à l'expression de contraintes particulières : contraintes structurelles et spatiales, visibilité, mise en évidence et détectabilité. Cet ensemble de relations est fixé une fois pour toute. Si les besoins du programme client sortent de ce cadre, l'expression et la maintenance d'une nouvelle contrainte doivent être programmées.

5.1. Programmation d'une Contrainte

La programmation d'une contrainte et de ses conséquences n'est pas une activité simple. Elle requiert la mise en place d'un système de résolution qui détermine les contraintes à appliquer et qui décide comment les appliquer. Un système de résolution ad hoc comme celui des machines à images

abstraites citées plus haut, n'est pas trop complexe à réaliser mais ne permet aucune généralité. Par exemple, la machine à boîtes d'Adèle n'offre pas de primitive pour centrer deux boîtes l'une par rapport à l'autre. Pour ce faire, le programme client doit déterminer les décalages horizontaux des boîtes en fonction de leur taille respective. Ce calcul exprime la méthode qui satisfait la contrainte de centrage. Le problème ne provient pas du fait que le client spécifie une méthode mais du fait que le client a la charge d'en détecter les conditions d'activation (en particulier à chaque changement de taille ou de localisation de l'une des deux boîtes). Cette détection devrait revenir à la machine à boîtes. Pour ce faire, le programme client devrait pouvoir définir des méthodes particulières et les transmettre dynamiquement à la machine à boîtes. Celle-ci aurait alors la fonction d'un système de résolution général comme l'est ThingLab.

5.2. Un Exemple de Système à Contraintes : ThingLab

ThingLab [Borning 86a, Borning 86b] est un environnement de spécification de contraintes réalisé au dessus de Smalltalk-80. Cet environnement permet de construire de manière interactive et sous forme graphique de nouveaux objets en définissant des contraintes sur des objets existants. La nature des contraintes dépend du client, non de ThingLab. Une contrainte ThingLab comprend un prédicat et une ou plusieurs méthodes. Le prédicat permet de vérifier qu'une contrainte est satisfaite. Sa forme, une expression algébrique, garantit la généralité. Les méthodes modifient tout ou partie des entités référencées dans le prédicat pour que celui-ci soit vérifié. A partir de la spécification du prédicat, ThingLab est capable de créer automatiquement les méthodes en question.

Considérons l'exemple de la figure 11.15 : l'objectif est de réaliser un rectangle d'histogramme. Pour cela, la hauteur du rectangle doit être proportionnelle à la valeur à représenter. Cette contrainte s'exprime simplement avec l'expression algébrique :

$$\frac{n}{100} = \frac{h_1 y - h_2 y}{p_1 y - p_2 y}$$

où

- n est la valeur à représenter,
- h_1 et h_2 sont les deux coins opposés du rectangle de l'histogramme,
- $h_1 y$ et $h_2 y$ désignent respectivement l'ordonnée de h_1 et celle de h_2,
- p_1 et p_2 sont deux points tels que la longueur du segment $[p_1 p_2]$ détermine la taille du rectangle lorsque n=100.

Cette équation comprend des variables et des constantes. Lorsque n varie, le rectangle change d hauteur mais sa base reste fixe. En conséquence, h_1 et n sont des variables tandis que h_2 est un constante. Le segment $[p_1p_2]$ sert de référence : p_1 et p_2 sont donc des points fixes. La spécificatio de l'expression algébrique comprend donc l'écriture de l'équation et la désignation des constantes d cette équation. Le procédé qui permet à l'utilisateur d'effectuer cette spécification n'est pas décrite ic La figure 11.15 n'en reflète que le principe.

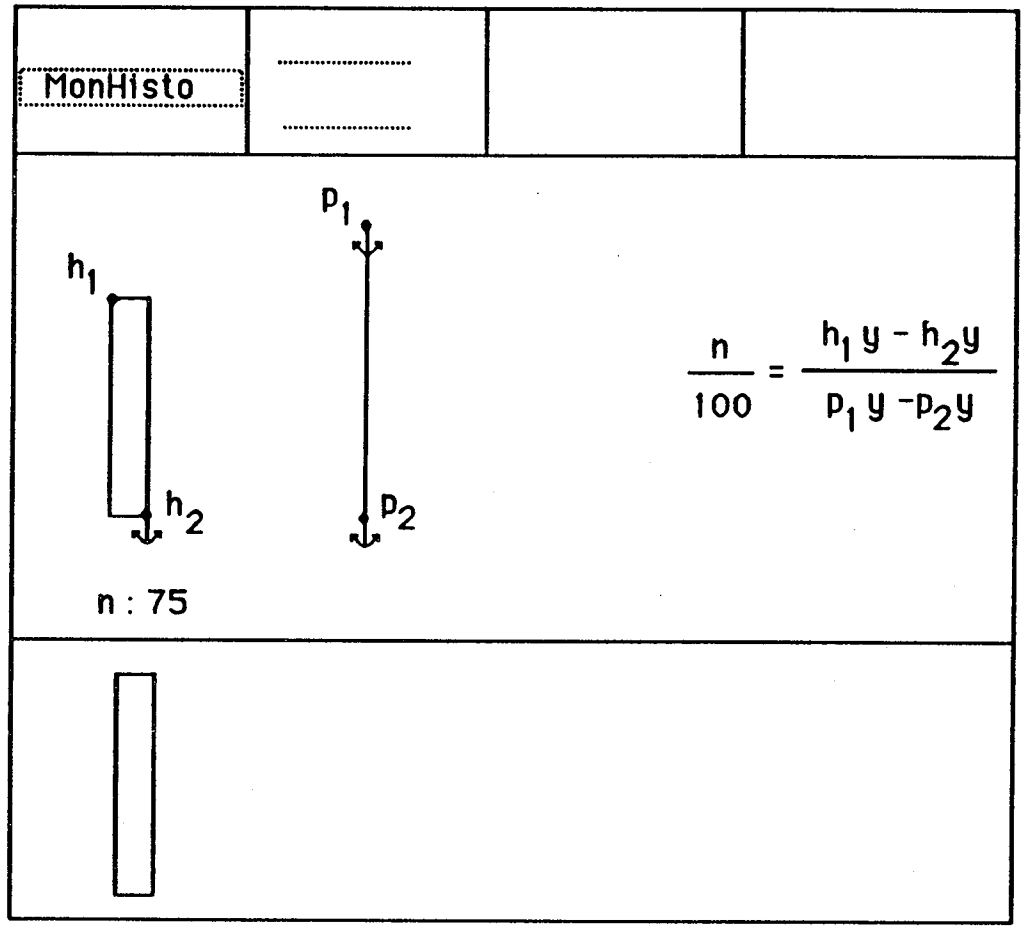


Figure 11.15 : Principe de la spécification d'une contrainte dans ThingLab.

Dans la fenêtre du bas de la figure 11.15, l'objet MonHisto tel qu'il apparaîtra lorsqu'il ser utilisé. Dans la fenêtre du centre, MonHisto en cours de construction avec le segment de référence l'équation et l'indication des variables et constantes. Le statut de constante est signalé par la présenc d'une ancre. Dans la fenêtre du haut, les menus du Browser de Smalltalk. Par l'intermédiaire de menus, l'utilisateur peut

- indiquer qu'il construit un nouvel objet à partir d'autres (dans notre cas, à partir de la class Rectangle),
- construire l'expression algébrique et indiquer les constantes de l'expression, etc.

L'intérêt de ThingLab est le mariage de la généralité et de la spécification interactive. La généralité est garantie par le pouvoir d'expression des équations algébriques ; la spécification interactive des contraintes ouvre la voie à la spécification interactive des interfaces homme-machine : construction de nouveaux objets de présentation et spécification d'images abstraites. Néanmoins, ThingLab ne peut tout faire. Il ne peut générer des méthodes qu'à partir de méthodes existantes. Dans l'exemple de la figure 11.15, ThingLab sait créer la méthode de mise à jour de la hauteur de MonHisto parce qu'elle peut s'exprimer à partir des méthodes de la classe Rectangle. Si, comme on l'a vu au chapitre 9, on veut qu'un liquide présente des bulles à partir d'une certaine température, la génération de cette méthode n'est possible que si un objet dans la base possède une méthode pour dessiner des bulles. Dans le cas contraire, la méthode doit être programmée selon les techniques traditionnelles. Nous reviendrons sur ce point à propos de la spécification interactive de dialogue.

Thinglab a servi de base à la réalisation de systèmes plus spécialisés : Animus [Duisberg 86] qui introduit la contrainte de temps et le Filter Browser [Ege 87] orienté vers la spécification de dialogue. Dans la même veine que ThingLab, nous trouvons Juno [Nelson 85] conçu pour la spécification interactive de contraintes géométriques. Le langage qui se cache derrière l'interface interactive de Juno est un transformateur de prédicats de la géométrie euclidienne. Juno n'a donc pas la généralité de ThingLab mais relève du même principe : à partir d'une contrainte spécifiée de manière interactive, créer (ou éditer) un programme qui maintient la contrainte. Cette technique mériterait d'être approfondie pour l'adapter aux besoins de la construction interactive d'interfaces homme-ordinateur. On trouve une timide application de ces idées dans X Toolkit avec la notion de *Geometry manager* [MIT 87]. Quelques travaux de recherche en cours de développement visent à intégrer l'expression de contraintes entre les objets de présentation d'une boîte à outils [Szekely 88].

EN RESUME :

L'affichage d'informations peut se voir comme une cascade de transformations qui, à partir d structures de données internes, produit une image sur un support de restitution. L'acquisition de informations implique une suite de transformations inverses. La difficulté de ces opérations est l distance entre la représentation des données internes et celle utilisée par les supports de restitutio Cette différence peut être atténuée par l'introduction d'une abstraction intermédiaire : l'image abstrait

Une image abstraite est une représentation intermédiaire entre une structure interne spécifique a programme client et une image concrète produite sur un support de restitution. Elle traduit en terme abstraits la présentation d'une structure interne, c'est-à-dire en termes indépendants de caractéristiques du support de restitution, fût-il virtuel.

On distingue deux classes de machines à images abstraites : l'une, tournée vers l'expression d relations structurelles, l'autre fondée sur l'expression de contraintes. La première utilise le techniques de représentation hiérarchique : graphe orienté acyclique et arbre décorés d'attributs d présentation. Les deux exemples illustratifs de cette approche sont les boîtes et PHIGS ; la seconde plus générale, fournit un système de résolution qui permet l'expression dynamique de contraintes Dans les deux cas, l'objectif recherché est répondre aux besoins de modification instantanée e incrémentale de l'interactivité.

Le rôle principal des boîtes est de dimensionner et de placer, les unes par rapport aux autres, le images des constituants d'une structure interne. Les boîtes n'ont aucun droit de regard sur leu contenu. Le rôle principal des structures de PHIGS est de regrouper des primitives graphiques dan des unités et de définir entre elles des relations hiérarchiques. A la différence d'une boîte, un structure PHIGS interprète son contenu. La boîte est un outil de composition d'images comme on l pratique en imprimerie alors que PHIGS est un outil graphique. Cette différence d'objectif ren possible leur combinaison, par exemple, en remplissant le contenu des boîtes avec des structure PHIGS.

Le rôle principal d'un système à contraintes est de permettre au programme client de spécifier de contraintes et d'en assurer la maintenance. La spécification d'une contrainte peut s'effectuer sous l forme d'un programme (une procédure ou une méthode qui indique comment satisfaire la contrainte) ou sous la forme d'un prédicat qui décrit la contrainte (dans ce cas, le système crée automatiquemen le programme). Un système à contraintes repose sur un mécanisme de résolution qui détermine le contraintes à appliquer et décide comment les appliquer. ThingLab est l'exemple marquant de cette catégorie d'outil.

Chapitre 12

Squelettes d'Application

1. Introduction

2. Principes directeurs

2.1. Rappel des objectifs et hypothèse de travail

2.2. Charpente réutilisable

2.3. Contrôleur réutilisable
- 2.3.1. Arbitre de la coopération
- 2.3.2. Transformateur de formalismes
- 2.3.3. Gérant de protocole de communication

3. APEX

3.1. Architecture

3.2. Le Contrôleur et ses fonctions d'arbitrage
- 3.2.1. Gestion du flux de l'information
- 3.2.2. Allocation de l'unité centrale aux entités actives du système

3.3. Le Contrôleur et le transfert de formalismes
- 3.3.1. Zone de communication entre l'application et le Contrôleur
- 3.3.2. Zone de communication entre la présentation et le Contrôleur

3.4. Le Contrôleur et le protocole de communication

3.5. Les services de présentation réutilisables

3.6. Les lessons de l'expérience APEX
- 3.6.1. Des principes directeurs
- 3.6.2. Des lacunes

1. Introduction

Nous avons relevé, à propos des boîtes à outils, la difficulté de l'apprentissage et la duplication d'efforts : le programmeur doit découvrir le mode d'emploi des services, les assembler correctement et reproduire une partie de cette construction pour chaque système interactif. Par exemple, on observe, à l'utilisation du Toolbox du Macintosh, que

1. certains traitements, comme les séquences d'initialisation et le rafraîchissement des fenêtres, sont systématiquement reproduits de système en système ;

2. chaque système s'organise généralement autour d'une boucle "tant que" de la forme :

```
tantque pas-fini faire
    acquérir-événement;
    traiter-événement;
fin-tant-que;
```

Cet exemple appelle deux remarques :

1. une boîte à outils ne fournit pas nécessairement le niveau de service désiré. Si une extension fonctionnelle convient à une classe de systèmes, alors sa réalisation doit être réutilisable.

2. une boîte à outils induit un modèle de contrôle conditionné par le mécanisme de réception des événements. Si l'architecture du système doit être centrée sur le traitement des événements, (n'oublions pas que les événements sont les manifestations logicielles des actions de l'utilisateur et que l'utilisateur est un client prioritaire), alors il doit être possible de réaliser un noyau réutilisable qui serve de charpente minimale à toute une classe de systèmes.

Un squelette d'application regroupe, sous la forme d'un logiciel réutilisable et extensible, les notions de noyau d'exécution et de services étendus. Lorsque cet assemblage est le lieu d'exécution d'un processus, on emploie également le terme "serveur d'interface". La tâche du programmeur consiste à remplir les trous du squelette et à redéfinir les parties prédéfinies inadaptées au cas particulier du système. La réutilisation, l'extensibilité et la surcharge de logiciel se traduisent directement dans les termes de la programmation par objets. Il n'est donc pas surprenant que les squelettes d'application actuellement disponibles aient été réalisés en exploitant les avantages de cette technique.

Avant de rentrer dans les détails d'exemples pratiques, nous allons présenter les principes directeurs des services d'un squelette. Une réalisation de squelette sera ensuite décrite avec APEX qui illustre mon expérience dans le domaine, puis on trouvera une étude comparative avec d'autres réalisations tels MacApp et Serpent. Le chapitre s'achève par une évaluation des avantages et des lacunes de cette classe d'outils.

2. Principes directeurs

Les principes directeurs à la conception de squelettes d'application sont en réalité mal définis. a plusieurs raisons à cela : le récent développement des boîtes à outils mais surtout l'absenc modèles d'architecture éprouvés. Le résultat de ces négligences est l'existence d'un savoir-faire formalisé, appliqué cas par cas et trop dépendant des outils logiciels. L'objet de ce paragraphe d'éclaircir la situation en s'appuyant sur les enseignements directifs de la section II consacrée modèles d'architecture.

Ce paragraphe est organisé ainsi : après un rappel sur les objectifs à atteindre et les hypothèse travail, un modèle de référence est présenté comme point de départ à la conception d'un squelette modèle met en évidence le rôle central d'un constituant, le Contrôleur. En raison de son importa ce dernier fait l'objet d'une description détaillée dans la dernière partie.

2.1. Rappel des objectifs et hypothèses de travail

L'objet d'un squelette d'application est de fournir une structure logicielle réutilisable qui assur plupart des fonctions de l'interaction avec l'utilisateur. Le corollaire est permettre au programm d'un système interactif de se consacrer essentiellement à la réalisation du logiciel dépendant domaine : l'application.

Décharger le programmeur d'application des techniques de présentation et de l'assemblage outils logiciels ne signifie pas ignorer l'utilisateur. Rappelons que les fonctions de l'applicat doivent répondre aux besoins conceptuels de la tâche, que l'utilisateur doit être immédiatem informé de toute "modification sensible" de l'état de l'application, et que l'application doit avoi possibilité de demander à l'utilisateur d'intervenir lorsque son fonctionnement l'exige. Nous reviendrons pas sur ces problèmes que nous supposons ici résolus. Nous nous plaçons dans situation où les concepts informatiques, procédures et structures de données, ont été défini organisés en fonction des objectifs et des variables psychologiques d'un utilisateur type.

Nous nous intéressons maintenant à la conception de la charpente réutilisable.

2.2. Charpente réutilisable

La conception de logiciel réutilisable rencontre une première difficulté avec l'identification de services qui soient utiles au plus grand nombre de clients [Tesler 86]. S'il existe des outils pratiques qui encouragent la décomposition modulaire, la façon d'organiser les niveaux d'abstraction et les relations entre les constituants d'un assemblage repose au contraire sur des méthodes purement empiriques. Une structure modulaire mais inadaptée aux clients compromet inévitablement la réutilisation. Dans le cas du squelette d'application, le recours à un modèle d'architecture est une stratégie raisonnable.

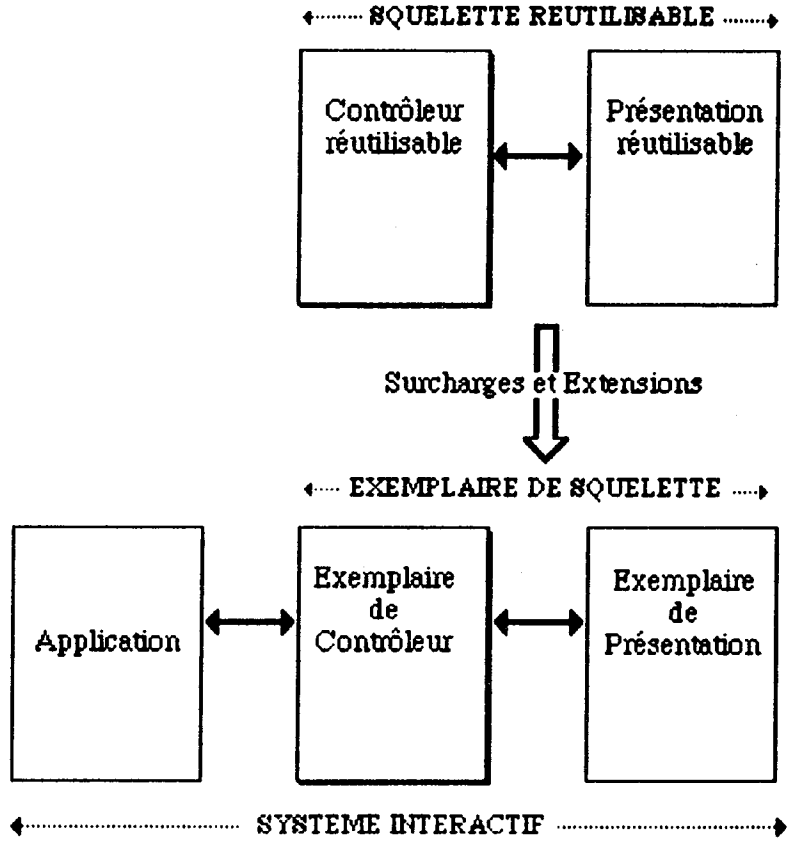


Figure 12.1 : Le modèle d'architecture servant de référence à la conception de squelettes d'application.

Les modèles d'architecture présentés dans la section II ne satisfont pas tous les "critères qualité" de l'interaction mais tous acceptent l'idée d'une séparation entre l'application et l'interface, et tous voient dans l'interface au moins deux niveaux de fonctions : pour certains, les aspects syntaxiques et lexicaux ; pour d'autres, la présentation et le pont avec l'application. A la vue langage qui modélise l'interaction comme un long flot régulier d'échanges traités en pipe-line, il faut préférer une approche

capable de répondre au comportement imprévisible de l'utilisateur. Nous avons vu aux chapitres 8 9 que l'organisation application-contrôle-présentation allait dans le sens de cette exigence.

Ces observations conduisent naturellement au choix du modèle PAC considéré au nivea d'abstraction le plus haut. Comme l'indique la figure 12.1, un système interactif comprend le logici de l'application relié à un exemplaire de squelette. Celui-ci est constitué d'un Contrôleur et d'ur présentation construits par extension et surcharge à partir du squelette de référence. Le Contrôleur c référence, qui sert d'intermédiaire entre l'application et sa présentation, est l'organe pivot de l charpente. Il justifie donc notre attention.

2.3. Contrôleur réutilisable

Le modèle de référence donne au Contrôleur un rôle de médiateur. Les deux mondes qu'il ser l'application et la présentation, vivent à des rythmes différents, ne parlent pas le même langage ma ont besoin de communiquer. En conséquence, une première fonction de base du Contrôleur e: l'arbitrage qui garantit la cohérence des traitements ; une seconde fonction est la transformation c formalismes ; une troisième est la réalisation d'un protocole adapté à la communication.

2.3.1. Arbitre de la coopération

Conceptuellement, l'application et la présentation sont le lieu d'activités parallèles. Chacune gèr la coopération des activités internes selon des critères qui conviennent à la situation locale, mais l fonctionnement global du système interactif nécessite un réglage des relations entre l'application et l présentation. La situation centrale du Contrôleur en fait l'arbitre naturel.

2.3.2. Transformateur de formalismes

L'application modélise la solution d'un problème au niveau le plus abstrait et communique ave l'environnement dans le formalisme qui convient à la situation. La présentation modélise ses relation avec l'environnement dans un formalisme adapté au type d'interaction. Elle inclut généralement le objets de dialogue d'une boîte à outils dont le caractère général est inadapté au cas précis d'un application donnée. L'application et la présentation définissant des perspectives différentes sur un même notion, utilisent nécessairement des types de représentation distincts. Une transformation d formalismes doit donc être opérée. Si l'on souhaite conserver l'indépendance entre la modélisatio

interne des concepts et leur présentation à l'utilisateur, cette transformation doit avoir lieu hors de l'application et de la présentation. La position centrale du Contrôleur le désigne naturellement à cette fonction.

Une transformation de formalisme inclut un mécanisme de conversion de type. Elle comprend également un mécanisme de transfert entre langages d'implémentation lorsque l'application et la présentation sont réalisées dans des langages distincts ;

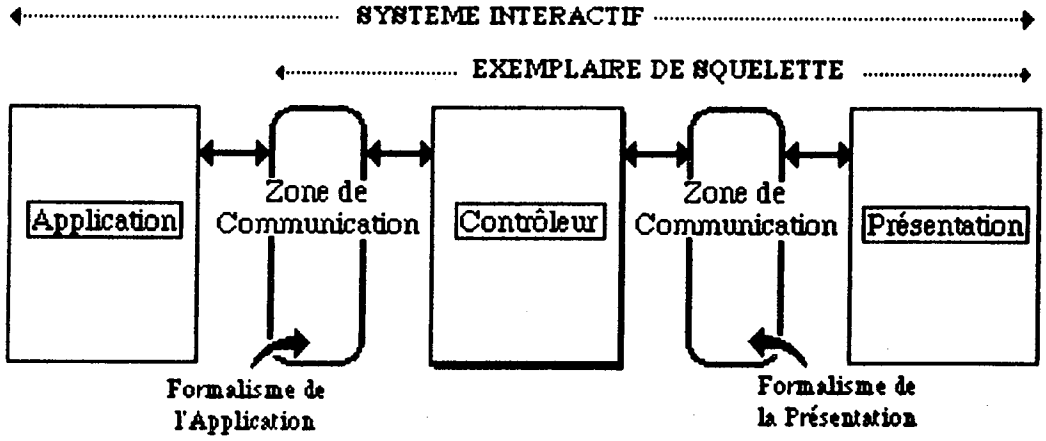


Figure 12.2 : Mécanisme de transfert entre formalismes assuré par le Contrôleur.

La figure 12.2 illustre le mécanisme de transfert entre les deux formalismes. Deux zones de communication encadrent le Contrôleur. L'une emploie le formalisme de l'application, l'autre celui de la présentation. Le Contrôleur traduit les éléments de l'une dans les éléments de l'autre. Ce transfert s'apparente à la transformation de types généralisée, domaine de recherche théorique dont l'avance est insuffisante pour répondre à nos besoins. Aussi, je suggère une approche plus technique en considérant ce transfert comme un mécanisme d'indirection assorti d'un traitement adéquat. L'indirection définit les associations entre les éléments des types de l'application et ceux de la présentation : le traitement effectue les adaptations éventuelles entre les valeurs sources et les valeurs destinataires.

La réalisation de la combinaison indirection-traitement peut s'envisager selon plusieurs techniques. Dans Serpent [Bass 88], les règles d'un système de productions expriment les conditions de transfert : les parties gauche référencent les éléments d'une zone de communication alors que les parties droite appellent des fonctions qui référencent des éléments de la seconde. Un autre procédé, comme dans APEX [Coutaz 87b], consiste à réaliser l'indirection par des tables de liaison et les traitements par des procédures de transformation associées aux entrées des tables. Une autre façon

relevée dans GWUIMS [Sibert 86], est la répartition des associations et des traitements dans u ensemble d'objets d'interface (les objets-I, médiateurs entre les objets-A qui représentent les concep de l'application, et les objets-G qui interviennent dans la définition de l'image du système). A lumière de ces exemples, nous observons que la technique choisie relève des outils d l'environnement mais que, dans tous les cas, l'objectif est identique.

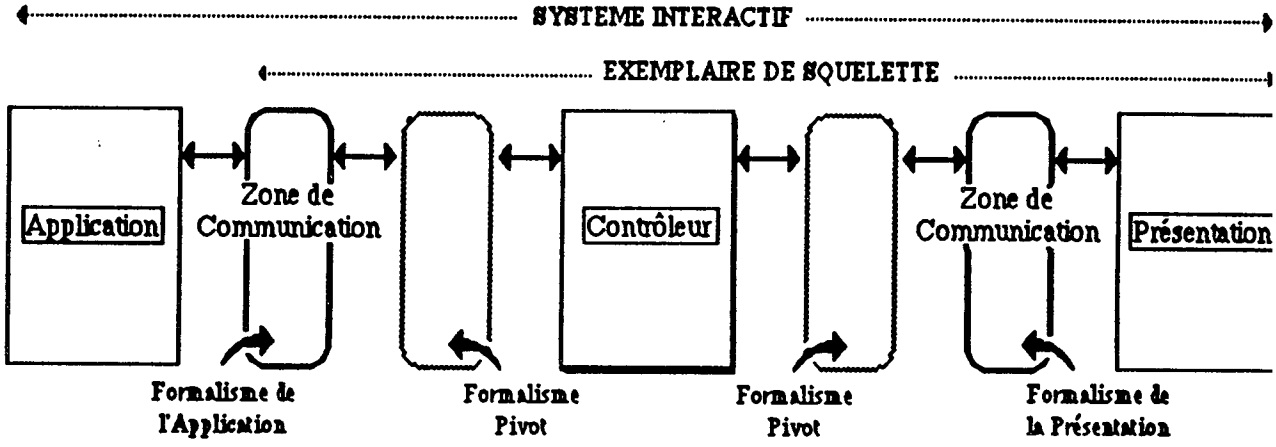


Figure 12.3 : Transfert entre formalismes par double indirection.

Le transfert entre formalismes peut aussi s'envisager par double indirection comme le montre l figure 12.3. L'inconvénient est l'accroissement des coûts de transformation. L'avantage est l définition d'un noyau véritablement indépendant des formalismes source et destinataire. Cett indépendance permet l'écriture de services généraux qui, utilisant la représentation pivot, ont l garantie d'être portables.

2.3.3. Protocole de communication

La dernière fonction à envisager pour le Contrôleur est la mise en place d'un protocole d communication. Un tel protocole a pour support un langage de description des conditions de transfer des informations. On y trouve généralement des primitives d'initialisation, d'écriture et de lecture. L type de synchronisation souhaitable dépend de l'environnement : dans un système monotâche, nou assistons à une communication synchrone alors que dans un environnement multitâche, il est possibl d'exploiter l'asynchronisme au profit de l'utilisateur (destruction brutale de processus). Le niveau d'abstraction des informations véhiculées constitue un second critère essentiel du protocole. C niveau doit être en rapport avec notre modèle de référence. En particulier, l'application doit émettre e recevoir des informations exprimées dans son formalisme. C'est ce que j'appelle un protocole de hau niveau.

Le type de service rendu par un protocole d'échange de haut niveau est à rapprocher des opérations d'entrée/sortie des langages de programmation. Ces opérations masquent le fonctionnement des appareils de restitution et assurent le passage entre une représentation interne binaire et une représentation externe de caractères imprimables. Aujourd'hui, avec l'évolution des exigences et la diversité des techniques, les chaînes de caractères ne sont plus les seules représentations cibles mais l'analogie reste valide : il faut que les mécanismes de traduction soient transparents au programmeur d'application.

Le langage du protocole une fois défini, il reste encore à préciser le mécanisme de transfert. Pour simplifier la diversité des techniques, nous dirons qu'il existe deux modèles essentiels : le modèle "base de données passives" et le modèle "base de données actives".

- Une base de données passives est une mémoire commune dans laquelle l'information est déposée par des producteurs et récupérée par les consommateurs intéressés. Appliqué à notre cas, le modèle suggère une base de données par zone de communication. Concernant la zone de communication entre l'application et le Contrôleur, la base contient des structures abstraites de l'application. Ces structures ne sont pas forcément identiques aux structures internes mais sont du même niveau d'abstraction. Par exemple, les structures de boîtes présentées au chapitre 11 pourraient servir de "formalisme d'exportation" de l'application. Une structure de boîtes est du même niveau d'abstraction que les concepts de l'application mais elle se rapproche des besoins de la présentation : elle exprime des règles d'affichage de manière symbolique et indépendamment de tout support de restitution. L'interprétation de l'arborescence correspond alors au mécanisme de traduction évoqué plus haut.

- Dans l'approche base de données actives, la communication ne s'effectue pas par l'édition d'une mémoire commune mais par l'envoi de messages directement interprétés par les objets destinataires. Le message contient la valeur modifiée. Cette technique convient parfaitement aux applications implémentées comme un ensemble de services où un service est une procédure (modèle type abstrait) alors que le modèle précédent convient davantage aux applications dirigées par les données (approche des structures globales partagées).

Après avoir présenté les principes directeurs à la conception de squelettes d'application, nous sommes en mesure d'étudier des exemples de réalisation.

3. APEX

APEX (APplication EXtensible) est le squelette d'application organisé selon le modèle PAC et réalisé en langage C au dessus du Toolbox du Macintosh [Coutaz 87a]. Une première version a servi de charpente à un éditeur graphique simple [Lermigeaux 86]. Avec l'expérience, de nouvelles

fonctions d'intérêt général ont été insérées [Bernard 87a]. Aujourd'hui, un même squelette est utili pour des applications aux domaines très différents, en particulier : l'application de thermodynamiq présentée au chapitre 9 dont l'interface suit les principes de la manipulation directe et un émulateur terminal graphique. Cet émulateur inclut le protocole VT100 et propose des extensions graphiqu simples tels les tracés de cercle, de droite, de polygone, etc. Il sert de frontal graphique à un application qui peut s'exécuter sur un calculateur distant relié à un Macintosh par une ligne série. Cet application peut être le shell d'Unix ou toute autre application répondant au protocole d communication de l'émulateur. La figure 12.4 montre une image de l'écran lorsque l'émulateur est frontal d'un simulateur de robot mobile [Crowley 86a, Crowley 86b] s'exécutant sur un VAX.

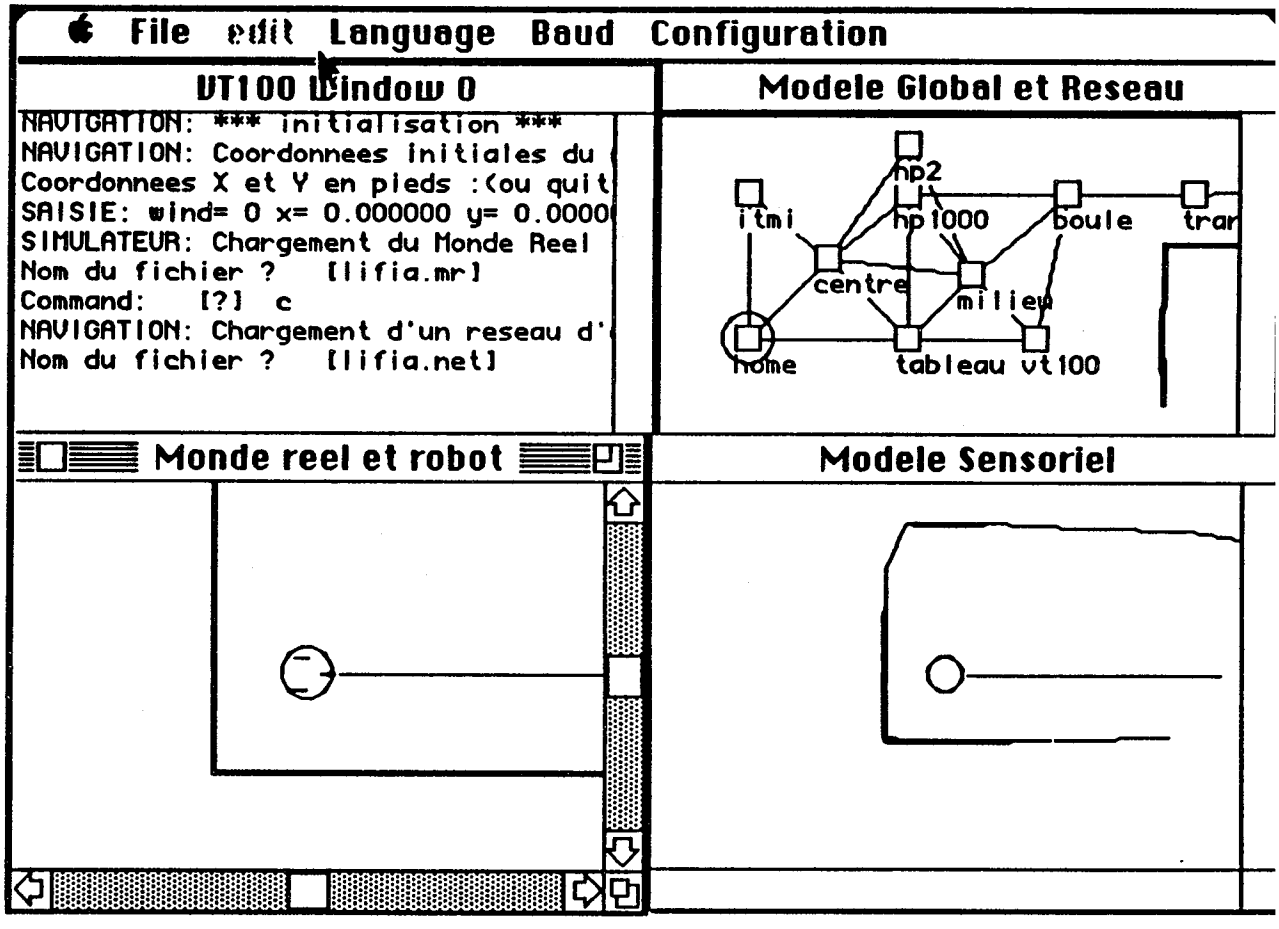


Figure 12.4 : Une vue de l'écran du Macintosh utilisé comme terminal graphique d'un simulateur de robot mobile. Le simulateur s'exécute sur un calculateur VAX relié au Macintosh par une ligne série.

Après ce bref exposé introductif sur les utilisations d'APEX, nous abordons les détails technique avec, en premier lieu, une présentation des principes de l'architecture logicielle, puis la descriptio des composants essentiels : le Contrôleur avec ses fonctions d'arbitrage, son mécanisme de transfe

entre formalismes et son protocole de communication enfin, les services de présentation réutilisables. La présentation sur APEX s'achève par une analyse de l'expérience acquise.

3.1. Architecture

Comme l'indique la figure 12.5, le squelette APEX est organisé selon le modèle PAC :

- la Présentation comprend des classes d'objets réutilisables dont la nature et l'intérêt seront précisés au paragraphe 3.5 ;
- l'Abstraction est structurée en deux niveaux de fonctions : une zone de communication et l'application. La zone de communication joue le rôle d'interface entre APEX et les traitements du domaine proprement dit. L'application est une machine abstraite sur laquelle APEX ne fait aucune hypothèse. Ce niveau est organisé en fonction des besoins de représentation et de manipulation des concepts du domaine.
- le Contrôle (ou Contrôleur) constitue le noyau d'exécution. Ses fonctions sont décrites aux paragraphes suivants.

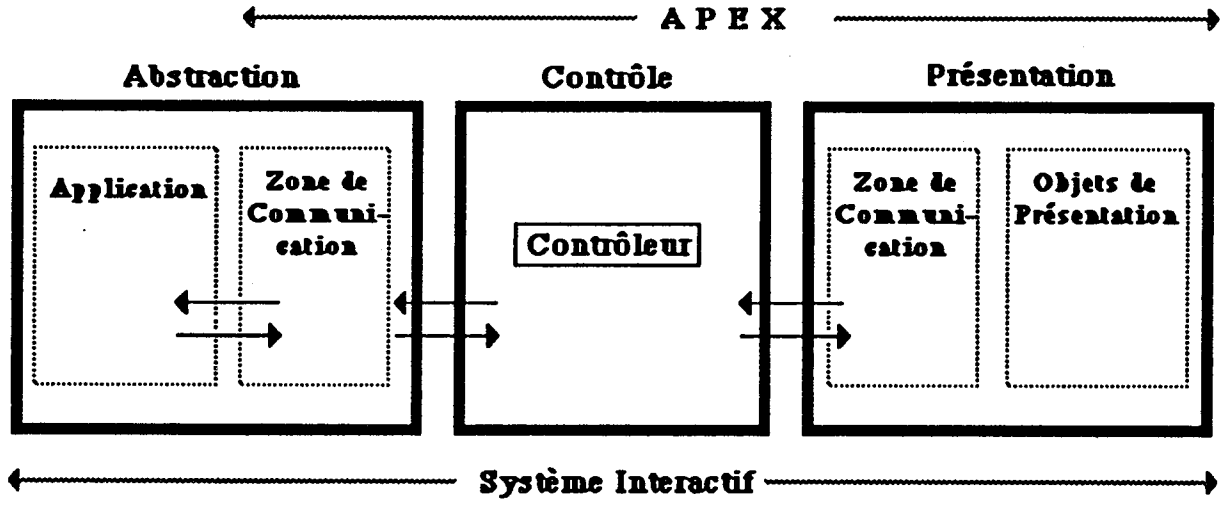


Figure 12.5 : L'architecture PAC du squelette APEX.

Considérant le point de vue du programmeur, le Contrôleur et les classes d'objets de présentation contiennent le code réutilisable ; la zone de communication, qui définit la vue qu'APEX a de l'application, est un logiciel surchargeable. Lorsque les objets de présentation ne conviennent pas aux besoins, le programmeur a la possibilité de définir de nouvelles classes. En aucun cas, le programmeur ne doit modifier le Contrôleur, organe central de la charpente logicielle.

3.2. Le Contrôleur et ses fonctions d'arbitrage

Le Contrôleur intervient au titre d'arbitre dans la gestion du flux de l'information et da l'allocation de l'unité centrale. Ces deux activités, bien que très liées au fonctionnement Macintosh, ont cependant l'intérêt de montrer comment parer aux lacunes d'un système d'accueil.

3.2.1. Gestion du flux de l'information

En règle générale, dans un système interactif, la gestion du flux de l'information est centrée su réception des événements. Ceci parce que la plupart des événements traduisent les actions physiq de l'utilisateur et que l'utilisateur est le centre d'attention prioritaire du système. Dans certai environnements, tels X Toolkit et Interlisp, un événement est automatiquement dirigé vers l'objet présentation concerné. Dans d'autres, telle la boîte à outils du Macintosh, la distribution doit ê programmée. Le Contrôleur d'APEX est donc organisé selon une boucle "tant-que" de réception et distribution des événements (on peut se reporter à la figure 10.2 du chapitre 10 pour une présentati détaillée de cette boucle).

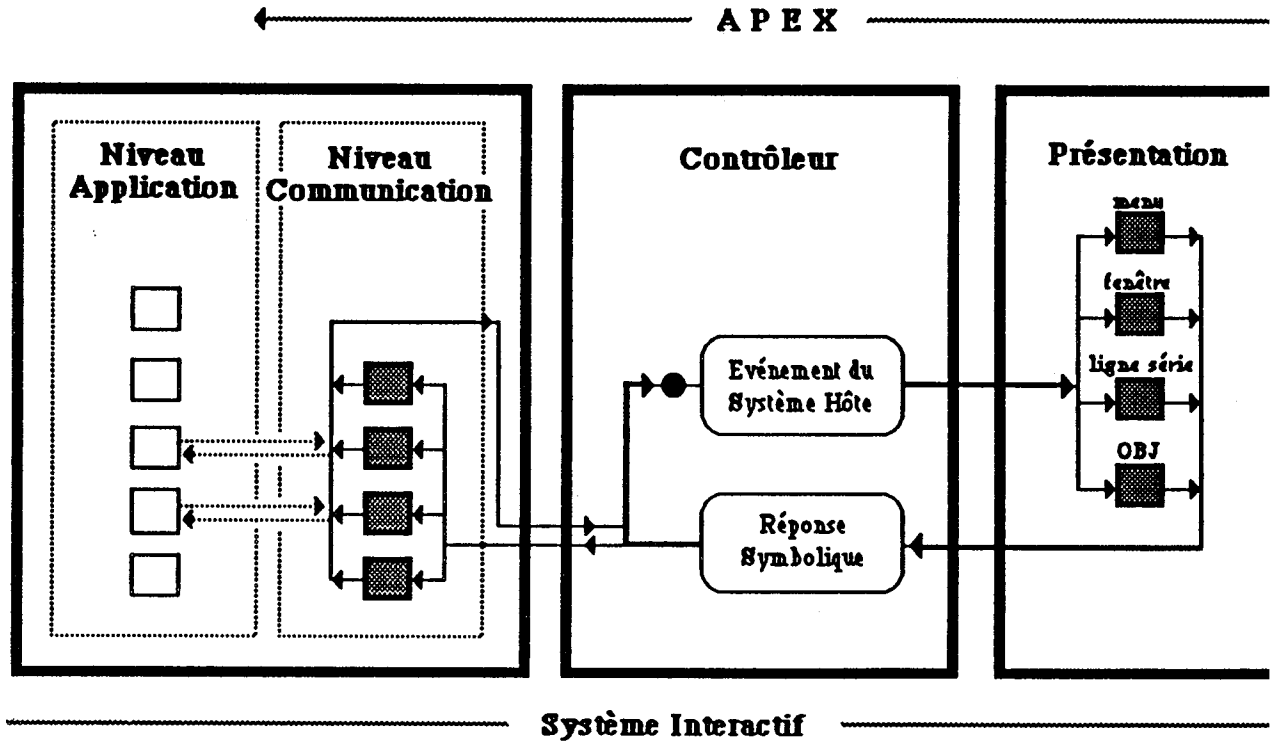


Figure 12.6 : Le Contrôleur est au centre du flux des informations.
(Les flèches concrétisent la "boucle" de contrôle du dialogue, les traits fins représentent des appels procéduraux. Le • indique le point d'entrée du cycle.)

Comme le montre la figure 12.6, le cycle du Contrôleur débute par une demande de réception d'événement. Dans un environnement monotâche comme celui du Macintosh, il est capital qu'une telle requête ne soit pas bloquante. En l'absence d'événement, le Contrôleur peut s'adonner à d'autres tâches, par exemple, gérer l'allocation de l'unité centrale à des activités locales, créant ainsi l'illusion du parallélisme. A la réception d'un événement, le Contrôleur détermine, parmi les exemplaires des classes prédéfinies, l'objet de présentation destinataire : fenêtre, menu, bouton, ligne série et OBJ, un représentant de toutes les classes d'objet répondant au modèle PAC (par exemple, des réchauds, des thermomètres, etc.). Nous avons démontré l'intérêt conceptuel de cet objet au chapitre 9. Sur le plan pratique, OBJ factorise, du point de vue du Contrôleur, la présence d'une multitude d'objets non standard construits pour des besoins précis de présentation.

Tout objet de présentation traite l'événement dans la limite de ses compétences. Il exprime au Contrôleur le résultat de son interprétation dans un formalisme qui lui est propre. Lorsque la réponse nécessite un complément de traitement, le Contrôleur fait appel aux services dépendants du domaine. A son tour, l'application entre dans une phase de traitement, exprime les modifications sensibles de son état grâce au protocole de communication, et requiert éventuellement l'intervention de l'utilisateur si la poursuite du service en dépend. Nous verrons, lors de la description du protocole, qu'une demande d'intervention est bloquante pour l'application, laissant le Contrôleur à nouveau maître du flux des informations. Au retour du traitement effectué par l'application, le Contrôleur entame le prochain cycle.

L'exemple suivant illustre le cheminement de l'information lorsque le Contrôleur reçoit un clic souris. Nous supposons que l'utilisateur a désigné le réchaud du système Thermo du chapitre 9. Le Contrôleur recherche le destinataire en s'appuyant sur les services de la boîte à outils du système d'accueil. En l'occurrence, les premiers candidats à la recherche sont les supports de restitution (les fenêtres ou les menus), puis, s'il s'agit d'une fenêtre, les objets prédéfinis de la boîte à outils affichables dans une fenêtre (les boutons, les barres de défilement). Nous nous plaçons dans la situation où le Contrôleur vient de déterminer la fenêtre concernée. Il lui adresse alors l'événement. (Rappelons que dans la plupart des systèmes de fenêtrage, l'identité de la fenêtre réceptrice de l'événement est transmise dans l'événement lui-même.)

Une fenêtre APEX est un objet de présentation. Elle a donc, à son tour, une Présentation :

- la Présentation de sortie hérite des attributs des fenêtres du système d'accueil. Elle comprend un cadre décoré d'icônes et une surface sur laquelle un programme client peut dessiner ou lire des informations. Le cadre est une zone qui n'a de sens que pour la fenêtre. La surface de dessin est partagée entre la fenêtre qui la perçoit comme un ensemble de pixels et le client qui peut en

donner une interprétation toute autre (se reporter aux techniques d'affichage structuré, pa exemple) ;

- la Présentation d'entrée consiste à interpréter les événements reçus. Par exemple, si le clic souri a lieu dans le cadre, l'événement concerne en premier lieu les compétences de la fenêtre e matière de cadre ; en supposant que l'utilisateur ait désigné l'icône de changement de taille, l fenêtre se charge d'acquérir interactivement la nouvelle taille, redessine automatiquement l cadre et reconstitue la surface de dessin à partir de la représentation qu'elle a de cette surfac (Nous reviendrons sur ce dernier point au paragraphe 3.5.) Si, en vertu de notre hypothès l'événement a lieu dans la surface de dessin mais hors d'un objet prédéfini de la boîte à outils, l fenêtre ne peut que constater le fait. Sa réponse au Contrôleur est l'événement initial enrichi d l'information "EVENEMENT DANS MA SURFACE DE DESSIN". Dans le cas d'u changement de taille, sa réponse aurait été "TAILLE MODIFIEE" accompagnée des nouvelle dimensions.

L'événement ayant eu lieu dans la surface de dessin de la fenêtre, le Contrôleur tente sa chanc auprès du représentant des objets non standard : OBJ. Connaissant les exemplaires présents dan l'image, OBJ est en mesure de les interroger à tour de rôle : "Est-ce que ce point appartient à votr image?".

- Lorsque la recherche est infructueuse, OBJ rend comme réponse "OBJET NON TROUVE". E dernier ressort, le Contrôleur signale l'événement à l'application qui peut alors décider d l'intérêt de l'information suivante : "Clic souris x-y dans la fenêtre f".

- Lorsqu'un objet de présentation PAC répond favorablement, OBJ lui demande de le traiter. L résultat de l'interprétation est toujours soumise au jugement de OBJ (voir au chapitre 9, l justification de ce contrôle). Selon le cas, OBJ signale au Contrôleur que l'événement a ét entièrement traité dans la Présentation, ou dans le cas contraire, active le mécanisme de transfer entre formalismes pour un traitement complémentaire dans l'application.

3.2.2. Allocation de l'unité centrale aux entités actives du système

La nature de l'événement courant détermine l'allocation de l'unité centrale aux entités susceptible de le traiter. Ce traitement chemine entre entités selon un mécanisme de communication synchrone qu accapare l'unité centrale jusqu'à ce que le traitement soit achevé. Or, il se peut que des entités no impliquées dans la chaîne de traitement aient besoin de s'exprimer. En particulier, tout objet anim requiert l'unité centrale pour traduire le mouvement visuel. Avec un noyau de processus légers, cett contrainte est levée de manière immédiate. Sans outil de multiprogrammation, il est nécessair d'installer un palliatif minimal. La solution adoptée dans APEX repose sur la technique de l'horlog programmée.

L'horloge programmée permet au Contrôleur de mettre en place une politique de temps partag entre les entités qui le demandent. Ces entités sont l'application ou des objets de présentation. Pou

qu'une entité obtienne l'unité centrale par le truchement de l'horloge, il lui faut en faire la demande. Cette demande indique la fréquence souhaitée des activations, fréquence que l'entité peut à tout instant modifier. Les activations ont lieu jusqu'à ce que l'entité en exprime la suspension. Les requêtes liées à l'horloge font partie du protocole de communication présenté au paragraphe 3.4.

L'horloge est constituée d'un ensemble de compteurs à raison d'un compteur par entité cliente. Chaque compteur est décrémenté de une unité à chaque cycle du Contrôleur. Lorsque le compteur d'une entité devient nul, il est réinitialisé avec la valeur de la fréquence d'appel et un service donné de l'entité est automatiquement lancé. L'horloge programmée permet ainsi

- à l'entité qui gère la ligne série d'émettre ou de recevoir des caractères,
- à l'application d'actualiser des valeurs (dans le cas de Thermo, celles de la température et de la chaleur),
- à des objets de présentation de réaliser des effets d'animation (dans le cas de Thermo, les effluves montent du réchaud, l'eau s'échappe du distributeur et l'eau bouillonne dans le vase).

Cette solution a l'avantage de la simplicité mais est elle ne convient qu'aux situations où la précision temporelle n'est pas essentielle et où les unités de traitement ne dépassent pas les 100 msec du cycle du processeur sensoriel. La base de temps de l'horloge programmée est le cycle du Contrôleur. Elle varie donc avec les traitements effectués au cours du cycle. Si, de surcroît, ces traitements durent au delà des 100 msec, l'effet d'animation des objets de présentation est compromis. Les classes d'applications envisagées avec APEX et le caractère expérimental de ce logiciel ne justifiaient pas une approche plus rigoureurse. Il est néanmoins essentiel qu'un squelette d'application soit conçu sans perdre de vue l'asynchronisme entre activités multiples.

3.3. Le Contrôleur et le transfert de formalismes

Le transfert entre formalismes s'effectue dans APEX selon les deux modèles, base de données actives et base de données passives, pour chacune des deux zones de communication (application et présentation).

3.3.1. Zone de communication entre l'application et le Contrôleur

La zone de communication entre l'application et le Contrôleur présente à la fois les caractéristiques de l'approche par objets et celles de l'indirection passive. Elle comprend un ensemble de points

d'entrée dont les noms sont imposés par APEX, et une table d'association entre les concepts d domaine et les entités de présentation dont les identifications sont au contraire laissées libres.

Un nom de point d'entrée correspond à une classe de réponses à effet de bord sémantique. U réponse relevant de cette catégorie est un signal adressé au Contrôleur lorsque la Présentation n'a p les compétences requises pour traiter à elle seule l'événement actuel. Il se trouve que des situatio d'incompétence se reproduisent de systèmes en systèmes. A ces causes parfaitement identifiées systématiques, il est naturel d'associer un nom de désignation permanente.

Le tableau de la figure 12.7 regroupe l'ensemble des noms définis dans APEX avec la descripti de la cause. A chacun d'entre eux correspond, dans la zone de communication de l'application, u procédure de même nom qui exprime la réaction du domaine à la situation. Par exemple, la procédu de nom **Task** peut relancer des calculs de valeurs du domaine et la procédure **InContent** peut donn un sens dépendant du domaine au clic souris survenu dans la surface de dessin d'une fenêtre.

Begin	l'application doit s'initialiser
End	l'application doit se terminer
InContent	un événement souris a eu lieu dans la surface de dessin d'une fenêtre
ResizedW	une fenêtre a changé de taille
ScrolledW	une opération de défilement a eu lieu sur une fenêtre
ClosedW	une fenêtre a été fermée
Key	une touche du clavier a été enfoncée
DoCommand	une fonction de l'application vient d'être désignée
SerialLine	un caractère vient d'arriver sur la ligne série
Task	c'est au tour de l'application de posséder l'unité centrale
Language	un changement de langue naturelle a eu lieu

Figure 12.7 : Les classes de situations symboliques concernant le savoir-faire d'une application.

En reprenant la terminologie et l'organisation logicielle de la programmation par objets, un no sert de sélecteur de message et la procédure de même nom est la méthode spécialisée dans le traiteme de ce type de message. APEX, avec cet ensemble prédéfini de noms, modélise l'application comm un objet véritable. Les fonctions dépendantes du domaine peuvent alors s'organiser en deux niveau d'abstraction : une machine qui réalise les opérateurs de noms imposés par APEX et une machine q réalise, sans aucune contrainte APEX, les fonctions du domaine. La première est surchargeable, l seconde doit être programmée à part entière. On trouvera en annexe 1 le squelette réutilisable de

machine surchargeable programmé en langage C.

Aux situations systématiques et prévisibles, s'oppose la liste infinie des cas spécifiques inadaptés à la vue statique des noms prédéfinis. A cette réalité imprévisible, APEX répond par une base de données passives. Cette base prend ici la forme toute simple d'une table d'association entre concepts du domaine et objets de présentation. Les principes de fonctionnement de cette liaison ont été présentés au paragraphe 3.3 du chapitre 9 : chaque fois que la base est modifiée par l'application, l'effet est immédiatement répercuté sur la Présentation. L'application exprime la modification dans le protocole de haut niveau présenté au paragraphe 3.4.

La base de données passives sert aux liaisons entre concepts spécifiques du domaine et objets de présentation spécifiques. Elles peut aussi être utile à la liaison entre des entités abstraites locales au Contrôleur et des objets de présentation. Ces entités correspondent à des variables psychologiques qui n'entrent pas dans le cadre de compétence défini pour l'application (se reporter aux notions de réparation et de répartition sémantiques évoquées au chapitre 9). Par exemple, dans le système Thermo, en supposant que l'utilisateur soit intéressé de connaître, sans avoir à compter, le nombre de façons dont la notion "température" est représentée à l'écran, en supposant aussi que ce comptage n'ait pas été prévu dans l'application, un compteur abstrait (un entier) peut être maintenu dans le Contrôleur et peut être présenté à l'écran par un objet PAC comme celui de la figure 12.8.

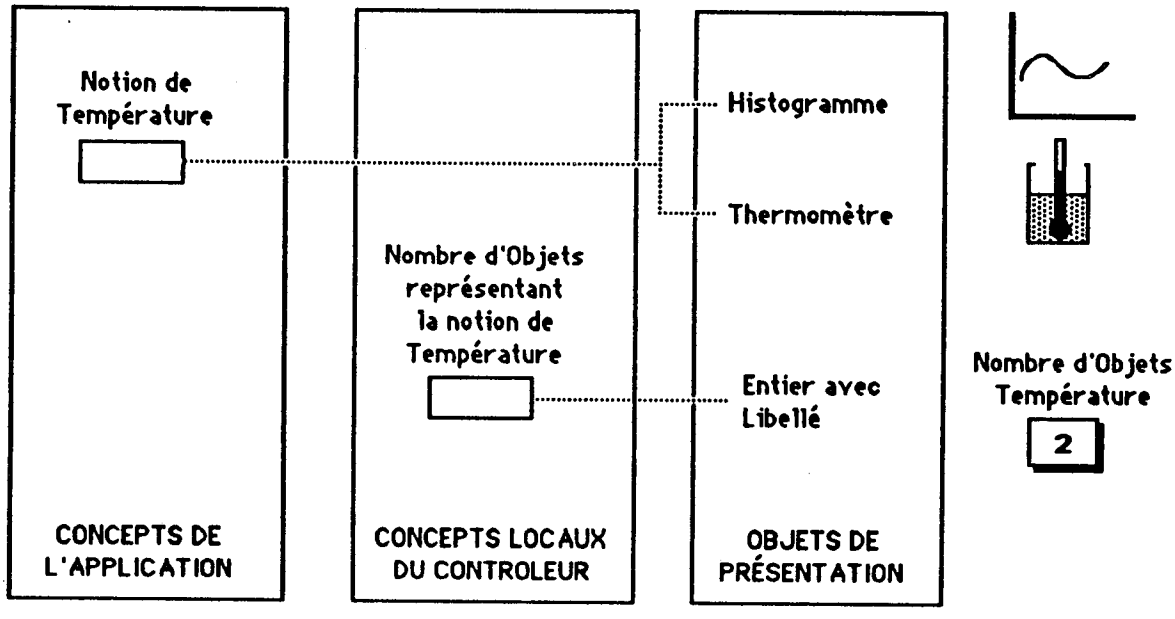


Figure 12.8 : Un exemple de notion abstraite locale au Contrôleur : le nombre d'objets représentant simultanément la notion de température de l'application.

3.3.2. Zone de communication entre la Présentation et le Contrôleur

Comme pour l'application, la zone de communication de la Présentation fait usage des deu formes de modèle.

La base des données actives a comme réalisation les entités prédéfinies : fenêtres, menu, lign série et OBJ. Ce dernier définit pour le Contrôleur une machine abstraite qui sait s'adresser au ca particulier de chacun des objets de présentation. Ses opérateurs permettent au Contrôleur de manipul les éléments de présentation sans les nommer directement. Par exemple, on se souvient dans l système Thermo que le concept de Température est lié à l'abstraction Temp d'un exemplaire d thermomètre. Cette association est représentée dans la base de données passives de l'application o Temp est désignée symboliquement par **"la constante TEMP de l'exemplaire j de la class THERMOMETRE"**. Le Contrôleur répercute la modification de Température en appelant l'opérateu de OBJ spécialisé dans cette tâche : **ObjSetVal(TEMP, j, THERMOMETRE, valeur)**. L'opérateur s charge de traduire la requête dans les termes de la classe concernée. La traduction s'appuie su l'hypothèse que les classes considérées sont des machines PAC.

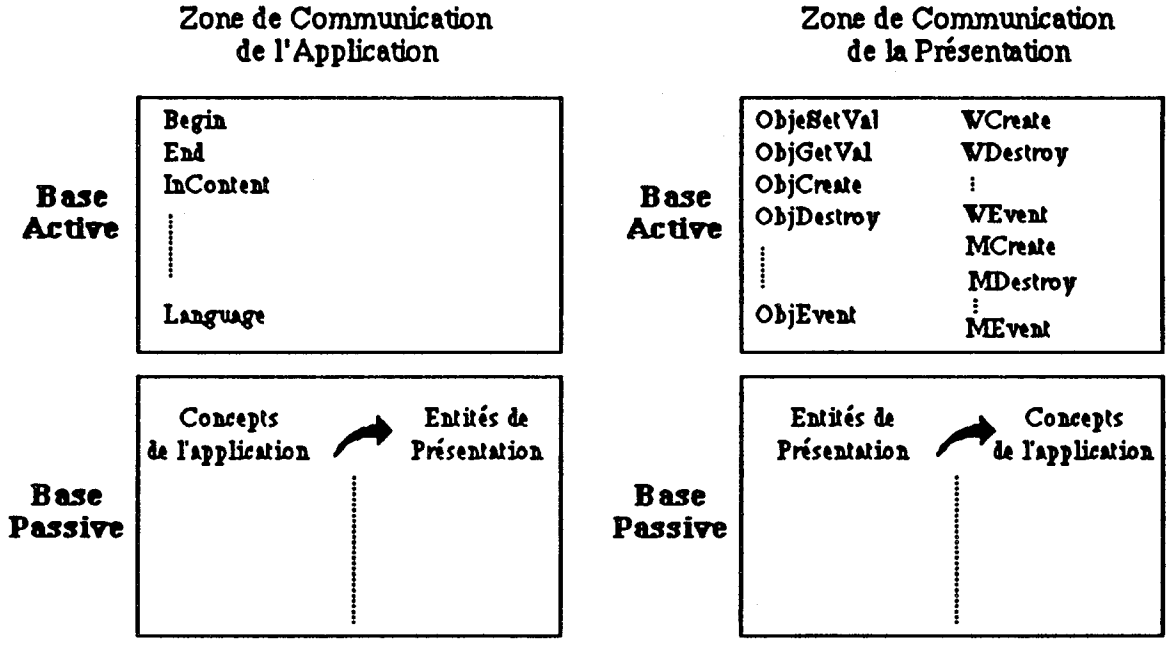


Figure 12.9 : Les deux zones de communication d'APEX utilisées pour le transfert entre formalismes.

La base des données passives de la Présentation est réalisée selon les mêmes principes que cell de l'application : un objet de présentation est lié à une abstraction de l'application. Les objets d présentation émettent deux types de réponses : les unes relèvent du cadre figé des noms prédéfinis, le

autres sortent de ces prévisions. La première catégorie a déjà été présentée : son interprétation par le Contrôleur donne lieu à un appel d'une procédure de la zone de communication active de l'application. Pour la seconde, à chaque classe de réponse symbolique est liée une abstraction de l'application (généralement une fonction paramétrée). OBJ, représentant unique de l'ensemble des objets de présentation PAC, et contrôleur ultime de leur comportement, effectue la traduction entre la réponse symbolique et le format attendu par l'application.

La figure 12.9 résume la situation. Les deux zones de communication actives, celle de l'application et celle de la Présentation, définissent pour le Contrôleur deux interlocuteurs qui masquent la diversité et le fonctionnement de deux autres mondes : celui de l'application et celui de la Présentation. Réalisées sous forme d'objet, elles conviennent à la communication de messages dont le type se retrouve de système en système. Pour les variations, les zones de communication passives offrent les moyens de lier entre elles des entités quelconques de chacun des deux mondes. L'association déclarative entre une classe de concepts et une ou plusieurs classes d'objets de présentation est une décision qui revient au concepteur de l'interface. Elle peut s'effectuer de deux manières complémentaires :

- soit en éditant la ou les table(s) qui concrétise(nt) les correspondances. Cette édition peut être effectuée directement à la main ou bien par l'intermédiaire d'un langage de spécification d'un générateur d'interface. Dans la réalisation actuelle, l'édition manuelle entraîne une recompilation et il n'existe pas d'outil de spécification ;

- soit par programme en exécutant une primitive du protocole de communication qui modifie la table d'association.

3.4. Le Contrôleur et le protocole de communication

Le protocole de communication entre l'application et le Contrôleur répond aux principes énoncés au chapitre 6 à propos de la gestion du dialogue :

- le contrôle est externe, c'est-à-dire maintenu hors de l'application mais il donne à l'application l'illusion d'acquérir le contrôle ;

- les échanges s'effectuent à un haut niveau d'abstraction, c'est-à-dire au niveau choisi par le programmeur de l'application.

Le contrôle externe est assuré par le Contrôleur, allocateur unique de l'unité centrale. L'application peut toutefois faire des demandes d'acquisition du contrôle grâce aux primitives Read et SetTask du protocole de communication : la première pour demander l'intervention de l'utilisateur, la

seconde pour pouvoir s'exprimer à intervalles réguliers.

Toutes les primitives du protocole véhiculent des informations du niveau d'abstraction d concepts de l'application. Ces informations sont typées de manière symbolique et toute primitive q inclut la désignation d'un concept inclut aussi le type du concept. Le tableau de la figure 12. répertorie les primitives essentielles.

Read	lecture d'une valeur d'une abstraction
Write	écriture de la valeur d'une abstraction
Bind	liaison d'une abstraction à une classe de présentation
UnBind	suppression d'une liaison entre une abstraction et un objet de présentation
Lock	invalidation des accès à une abstraction
UnLock	validation des accès à une abstraction
SetTask	affectation de la fréquence d'allocation de l'Unité Centrale

Figure 12.10 : Les primitives du protocole de communication utilisables par l'application.

Les primitives d'entrée/sortie sont synchrones. Un Read est bloquant jusqu'à ce que le concep paramètre du Read, ait reçu une valeur. Ce concept reçoit une valeur dès que l'utilisateur termine ur saisie sur l'objet qui lui sert de présentation. Par exemple, le service d'aide du système Therm exécute la primitive **Read(more)** pour demander à l'utilisateur la poursuite ou non des renseignement Le service d'aide est mis en attente jusqu'à ce que l'un des deux boutons qui lui servent d présentation soit activé par l'utilisateur. Afin de ne pas contraindre l'utilisateur à agir immédiatemer sur les objets de présentation, APEX inclut un noyau de coroutines qui permet d'exécuter les service qui effectuent des lectures comme des coroutines du Contrôleur.

Une opération d'écriture n'est pas bloquante. Elle transmet immédiatement son effet à l Présentation. Par exemple, chaque fois que Thermo souhaite notifier au monde extérieur la valeı actuelle de la variable entière **Temperature**, elle exécute l'instruction **Write(TypeInt, Temperature** L'interprétation de cette primitive par le Contrôleur inclut la recherche, dans la base de donnée passives, de la liaison qui lui correspond et diffuse la nouvelle valeur à la Présentation pa l'intermédiaire de OBJ (**ObjSetVal**).

Les primitives de contrôle des accès, **Lock** et **UnLock**, permettent à l'application d'exprimer un forme de sémantique dynamique. Selon l'état de l'application, certains concepts ne sont pa

accessibles ou au contraire deviennent à nouveau manipulables. La primitive **Lock** signale au Contrôleur qu'un concept est désormais inaccessible jusqu'au prochain **UnLock**. L'interprétation de ces primitives par le Contrôleur inclut la recherche, dans la base de données passives, de la liaison correspondant au concept et diffuse la nouvelle valeur à la Présentation par l'intermédiaire de OBJ (respectivement **ObjLock** et **ObjUnLock**).

Les primitives **Bind** et **UnBind** permettent de lier dynamiquement des concepts de l'application à des objets de présentation. C'est ainsi que dans le système Thermo, l'utilisateur peut dynamiquement demander à observer la température à la fois dans le thermomètre et dans l'afficheur de courbe, ou bien supprimer l'une des deux vues (ou encore les deux). Ces demandes s'expriment en l'occurrence par la sélection d'éléments de menu. L'interprétation de ces sélections inclut des Bind (ou UnBind) du concept de Temperature avec l'abstraction TEMP de l'exemplaire j de la classe THERMOMETRE ou avec l'abstraction TEMP de l'exemplaire k de la classe PLOT.

3.5. Les services de présentation réutilisables

Les services de présentation réutilisables regroupent des fonctions utiles à une classe au moins de systèmes interactifs. Dans le cadre des systèmes envisagés avec APEX, nous avons retenu les conséquences de la manipulation de fenêtre et plus généralement les contraintes logicielles imposées par les interfaces de manipulation directe. Une conséquence inévitable de la manipulation des fenêtres est la restauration des parties endommagées. Une conséquence inévitable des interfaces de manipulation directe est le mouvement et la superposition d'unités graphiques. Mouvement, superperposition et restauration de fenêtres et d'objets graphiques ne sont que partiellement assurés par le Toolbox mais sont intégrés aux services de présentation réutilisables d'APEX.

D'autres services APEX réutilisables, telles la gestion de la ligne série, la prise en compte des menus et l'installation automatique des ressources, dont l'intérêt technique est spécifique au Macintosh, ne seront pas décrits ici. La technique de restauration des dommages subis par les unités graphiques a déjà fait l'objet d'une description au chapitre 10 (paragraphe 2.4, Dispositifs pour l'interactivité). Seuls sont présentés ici, les principes de réalisation de fenêtres dotées d'un mécanisme de rafraîchissement automatique (mécanisme évoqué au chapitre 6 à propos du fenêtrage).

L'utilisateur effectue sur les fenêtres quatre actions entraînant systématiquement une restauration logicielle de fenêtre : l'ouverture, le déplacement, le changement de taille et le défilement. Ces actions

relèvent essentiellement de "tâches syntaxiques" car elles ne concernent généralement pas les tâcl conceptuelles du domaine ; elles sont les conséquences directes des conditions matérielles de présentation : fenêtres trop petites, écran restreint ou surchargé, etc. A ces tâches syntaxiques, qui concernent pas directement le domaine d'application, doit correspondre une réponse logicielle dans squelette. Dans le cas d'APEX, la solution choisie est la définition d'une nouvelle classe de fenêt qui hérite, surcharge et étend les fonctions de base fournies par le système de fenêtrage du Macinto

En tant qu'agrégat d'attributs, une fenêtre APEX est une fenêtre Macintosh assortie d'une surfa d'affichage virtuelle. La figure 12.11 montre les relations entre la surface virtuelle et la fenê Macintosh d'une fenêtre APEX. Le programme client peint sur la surface virtuelle en utilisant l primitives graphiques du Toolbox ; (celles-ci sont donc héritées par les fenêtres APEX.) La fenê Macintosh est une lucarne de taille variable qui se déplace dans l'espace de coordonnées de la surfa virtuelle au gré des opérations de défilements latéraux et verticaux. Le coloriage de tout ou partie de lucarne s'effectue par transfert de pixels depuis la surface virtuelle. Ce transfert a lieu à l'exécuti d'une méthode de rafraîchissement spécifique à la classe des fenêtres APEX.

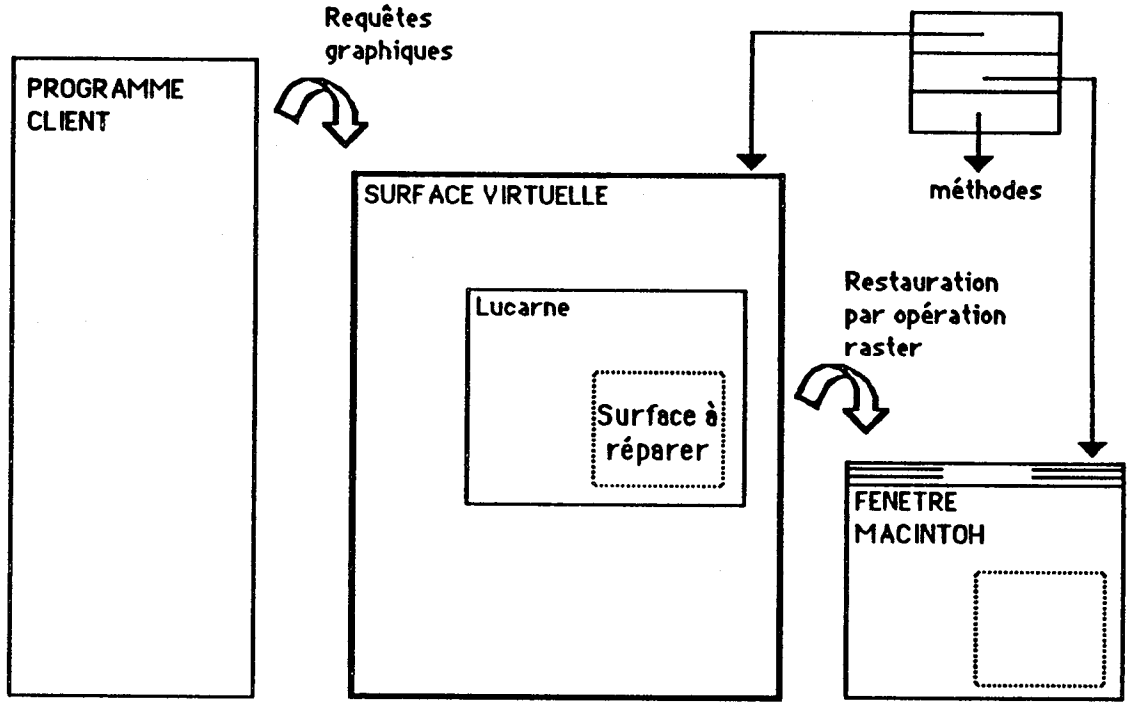


Figure 12.11 : Fenêtre APEX et fenêtre Macintosh.

Récepteur de toutes les requêtes graphiques, la surface virtuelle sert de cache de pixels. La taille de ce cache ne dépend pas de la fenêtre Macintosh visible ; elle est fixée par la seule entité qui soit capable de la déterminer de manière optimale : le programme client. Celui-ci précise la taille désirée à la création de la fenêtre APEX. Grâce à ce cache de pixels, une fenêtre APEX est en mesure de prendre en compte toutes les causes de rafraîchissement sans demander l'intervention du client. Il suffit d'exécuter la méthode de rafraîchissement qui, par simple opération raster de copie, répare la zone endommagée de la fenêtre Macintosh. Pour l'utilisateur, le rafraîchissement est immédiat et s'effectue sans provoquer de gêne visuelle.

Pour le programmeur de l'application, le détail du fonctionnement du système de fenêtrage devient transparent. Toutefois, une fenêtre APEX "n'avale" jamais l'événement de bas niveau à l'origine du rafraîchissement. Celui-ci est transmis au client (le Contrôleur) sous forme d'une réponse symbolique (se reporter au tableau de la figure 12.7 et y relever ResizedW et ScrolledW). Ce procédé a deux avantages :

- informer l'application des événements qui pourraient avoir une interprétation à un autre niveau d'abstraction. Par exemple, un changement de taille pourrait s'interpréter comme un changement du format de présentation. Si ce format est géré par l'application, celle-ci doit en être averti. Si ce format est géré par un objet de présentation spécialisé (par exemple, un système de boîtes), celui-ci doit aussi en être averti.

- permettre à l'application d'étendre les fonctions des fenêtres APEX. Par exemple, les opérations de défilement des fenêtres APEX sont bornées par l'espace de pixels de la surface virtuelle. Cette restriction est acceptable pour la plupart des applications graphiques mais doit être levée pour les applications manipulant des documents volumineux. La gestion du cache doit alors s'effectuer à plusieurs niveaux car il est prohibitif de maintenir un document de grosse taille dans sa totalité dans la surface virtuelle.

3.6. Les lessons de l'expérience APEX

APEX est l'une des premières expériences dans le domaine des squelettes d'application. De cette expérience se dégagent des principes directeurs de conception mais aussi des limitations à corriger.

3.6.1. Des principes directeurs

Les principes directeurs qui identifient les composants d'un squelette, leurs fonctions et leurs propriétés, ont été présentés au paragraphe 2. De ces composants, il faut retenir le rôle central du Contrôleur et les techniques de réalisation des zones de communication :

- les fonctions d'arbitrage du Contrôleur sont utiles à l'allocation de l'unité centrale aux activit du système et à la gestion du flux de l'information entre l'application et l'interface ; s protocole de haut niveau garantit l'indépendance logicielle de l'application vis-à-vis d techniques de présentation et ses zones de communication assurent le transfert entre l formalismes du domaine et ceux de l'interface ;

- les zones de communication peuvent être des objets de traduction aux opérateurs prédéfinis bien des bases de données agissant comme des vases communicants sans contenu imposé, encore, comme dans APEX, une combinaison des deux techniques. La première techniq exploite directement les propriétés de la programmation par objets, la seconde s'apparente l'édition de liens dynamique. La première, malgré ses noms prédéfinis, permet les extensio en jouant sur la spécialisation : il est toujours possible d'introduire de nouveaux nor (réponses symboliques) en définissant une sous-classe de la zone de communication existant

3.6.2. Des lacunes

Les lacunes concernent le protocole de communication et l'impossibilité pour le squelet d'imposer une frontière entre les formalismes de l'application et ceux qui relèvent de l'interface.

Les primitives du protocole ont l'avantage de véhiculer des informations à un haut nive d'abstraction mais elles appellent des extensions :

- dans la version actuelle, les types sont désignés par des symboles prédéfinis. Par exemple, concept de type entier a pour type symbolique la constante **TypeInt**. Le protocole devrait inclu les primitives de création et de destruction dynamique de symboles comme dans X Toolkit. serait alors possible de spécifier dynamiquement qu'un concept de type X doit être lié à d objets de présentation de classe Y ou Z ;

- de même, le protocole n'inclut pas la création ou la destruction d'exemplaires de concep L'absence de ces primitives est simplement une question de réalisation.

Si le protocole d'APEX est élémentaire, il démontre néanmoins la possibilité d'installer à peu frais un mécanisme de communication de haut niveau entre l'application et son environnement. I définition d'un protocole plus riche et général est à l'évidence un thème de recherche à entreprendre.

La difficulté pour le programmeur d'application de discerner la frontière entre l'abstrait et concret tient au fait que le squelette ne peut interdire la présence de primitives du Toolbox da l'application. Le programmeur peut directement produire le contenu des fenêtres sans passer par d objets de présentation PAC, prenant le risque d'enfouir dans l'application des caractéristiques releva de la présentation. L'exemple de la figure 12.4 est tout à fait parlant. Les messages qui apparaisse dans la fenêtre du haut et à gauche, sont produits par le simulateur et envoyés à l'émulateur graphiq par l'intermédiaire de la ligne série. A l'évidence, la formulation de ces messages est une affai d'interface. D'où vient l'erreur?

Le simulateur, qui s'exécute à distance, constitue la véritable expertise de l'ensemble ; l'émulateur graphique est, du point de vue d'APEX, la véritable application. En raison de ses fonctions, il convient que cette application ait accès aux primitives graphiques du Toolbox mais il convient aussi que le niveau d'abstraction du protocole graphique permette à l'expertise distante de s'exprimer de manière symbolique. Ce protocole manipule des notions graphiques de bas niveaux (texte, droite, cercle, rectangle, etc.) mais n'a aucune notion d'objet standard de présentation (message, menu, formulaire) et encore bien moins celle d'objet PAC. En raison du niveau de service proposé par le protocole, le simulateur envoie des messages textuels qui reproduisent dans la fenêtre VT100 l'effet d'un menu ; il envoie aux autres fenêtres les ordres graphiques "dessiner un cercle" et "quelle est la position du clic souris?" pour représenter la position actuelle du robot et lire la destination souhaitée par l'utilisateur.

Afin d'être conforme aux principes de la distinction entre les formalismes de l'application et de l'interface, le protocole de l'émulateur à destination du simulateur devrait relever son niveau d'abstraction. De même qu'il dispose de primitives de manipulation symbolique de fenêtres (dont l'interprétattion fait appel aux opérateurs des fenêtres APEX), de même, il devrait inclure des primitives de manipulation de menus dont les éléments seraient créés, comme pour les fenêtres, à partir de représentations externes ; il devrait inclure des primitives de manipulations symboliques de messages dont l'élaboration serait réalisée comme dans IRENE ; il devrait permettre des primitives de manipulation symbolique d'objets PAC. En l'occurence, on pourrait imaginer l'insertion, dans la Présentation d'APEX, d'un robot PAC conforme au squelette attendu par OBJ, chargé de se dessiner sur l'écran et d'interpréter les balayages de la souris quand l'utilisateur "le conduit par la main" vers une nouvelle destination. Le simulateur communiquerait avec l'émulateur non pas en termes de **"dessiner un cercle de tel diamètre à tel endroit"** mais en termes de **"le robot r est maintenant à l'emplacement x-y"** où r serait le concept lié au robot PAC. L'avantage serait d'une part, le remplacement d'une image de robot par une autre sans toucher au logiciel du simulateur et d'autre part, la suppression dans le simulateur de la traduction des informations qui relèvent véritablement du domaine en leur équivalent graphique.

L'exemple de l'émulateur graphique met clairement en évidence le piège permanent de la confusion entre niveaux de formalismes. Cet écueil n'est que plus dangereux en l'absence d'outils directifs ou, si l'on préfère, en présence d'outils trop permissifs! Nous allons voir que des dangers similaires se retrouvent dans l'usage d'autres squelettes.

4. Autres squelettes : GROW, Serpent, MacApp et EZWin

Les squelettes ici présentés sont tous très proches de la conception par objets et tous, à l'exceptio de MacApp, sont des prototypes de laboratoire : GROW, Serpent et EZWin.

4.1. GROW

GROW (GRaphical Object Workbench) [Barth 86] est un environnement d'objets de présentatio réalisé au dessus de Interlisp-D dans le langage à objets Strobe [Smith 83] sur les machines Xerox d la série 1100. Les éléments directeurs de GROW sont la conception par objets et la séparation entr l'application et son interface.

Les objets graphiques de GROW sont liés par trois formes de relations : l'héritage qui permet l partage de procédures et de structures ; la composition qui permet de considérer plusieurs objet comme une nouvelle unité ; et la dépendance des données qui permet d'exprimer les effets d'un modification d'attribut sur d'autres attributs :

- l'héritage est la relation par excellence des langages de programmation par objets. Elle perme l'extension fonctionnelle par spécialisation sans modifier le logiciel original. Cette propriété es importante puisque l'extension fonctionnelle est nécessaire aux ajustements logiciels d l'interface.

- la composition est une relation structurelle fondamentale. Dans GROW, elle permet de propage une action destinée à un objet composé à l'ensemble de ses constituants. Par exemple, u message de mouvement adressé à un objet composé se traduit automatiquement par l'envoi d'u message de mouvement à ses constituants.

- la dépendance des données est l'apport original de GROW dans le cadre d'un squelett d'application. Il s'inspire des principes de résolution de contraintes de ThingLab dont l'intérêt a été présenté au chapitre précédent. A la différence de ThingLab, GROW se limite au cas de relations acycliques.

La séparation entre l'application et l'interface est le second principe directeur de GROW. L liaison entre les deux mondes est assurée par une base de données passive sous forme d'une tabl d'indirection : lorsque l'application demande la création d'un exemplaire d'objet de présentation, ell précise une clé qui servira à désigner le concept du domaine.

Par rapport à APEX, réalisé au dessus du Toolbox du Macintosh, GROW bénéficie de l'environnement à objets du système hôte : le programmeur d'APEX construit un nouvel objet de présentation en éditant un squelette (donc du logiciel existant) tandis que le programmeur GROW procède par spécialisation. Alors que PAC, le modèle sur lequel APEX s'appuie, offre un cadre adapté à l'expression des relations de composition et de dépendance de données, APEX, contrairement à GROW, n'inclut pas de mécanisme pour l'expression déclarative de ces relations. En conséquence, OBJ, qui joue dans APEX le rôle de vérificateur de contraintes, n'a pas la généralité envisageable avec GROW.

Toutefois, dans APEX, le protocole d'échange entre l'application et l'interface garantit une meilleure distinction entre les formalismes. L'application APEX ne référence les objets de présentation qu'à la liaison d'un concept à des objets de présentation. Toutes les primitives du protocole autres que Bind désignent uniquement des concepts du domaine. A l'inverse, à chaque modification de valeur de concept, l'application GROW doit appeler la méthode correspondante des objets de présentation concernés. Ceci signifie que dans le sens application-interface, la traduction entre formalismes est effectuée à la main par l'application, mettant en cause son indépendance vis-à-vis des techniques de présentation. Contrairement à APEX, le mécanisme d'indirection mis en place par GROW n'est pas symétrique : il fonctionne seulement dans le sens interface-application.

4.2. Serpent

Serpent [Bass 88] est réalisé en langage C et OPS83 [Forgy 84] dans l'environnement Unix au dessus de X. Il est organisé selon le principe général d'architecture des squelettes d'application en s'inspirant du modèle multiagent. La Présentation comprend des objets élaborés avec X Toolkit. Le Contrôleur est un système de productions qui décrit les conditions de cheminement du dialogue. Les zones de communication s'inspirent directement du modèle des bases de données relationnelles. Par exemple, l'application déclare des schémas de relations et dépose dans la zone qui la relie au Contrôleur, des valeurs de n-uplets. Le Contrôleur, averti du dépôt, se charge du transfert de formalisme vers la Présentation. Ce transfert s'effectue par l'intermédiaire d'exemplaires de "Contrôleurs de Vue" (View Controllers).

Un Contrôleur de Vue est une matrice similaire à la notion de classe d'objets pour la création d'exemplaires :

- il définit les conditions de création et de destruction des exemplaires de la classe en fonction

d'événements survenant à une classe de n-uplets de la base de l'application (par exemple, dépôt d'un n-uplet),

- il précise les classes de widgets devant intervenir dans la présentation de la classe de n-up ainsi que les fonctions de transfert entre les valeurs du n-uplet et les attributs du widget,
- il définit les conditions de création ou de destruction d'exemplaires d'autres Contrôleurs de V en fonction des événements provenant de la base de Présentation (par exemple, à la récepti d'un message de sélection d'un bouton X Toolkit, créer un Contrôleur de Vue qui provoqu l'affichage d'un formulaire X Toolkit).

A l'exécution, le Contrôleur Serpent est donc un ensemble d'exemplaires de Contrôleurs de V simultanément actifs. Certains, ceux dont la création est effectuée par d'autres Contrôleur de Vues, trouvent avoir des parents. La structure hiérarchique ainsi produite rappelle l'organisation du mod PAC. Une seconde similitude avec PAC est le rôle central accordé aux Contrôleurs de Vue avec fonctions d'arbitrage, de transfert entre formalismes et de gestion d'activités multiples. Alors q PAC est un modèle qui ne préjuge pas des outils de réalisation, Serpent est une réalisation q s'appuie sur le modèle multiagent où chaque agent est un minisystème de productions.

Serpent, comme GROW et APEX, inclut des mécanismes d'extension : il est toujours possible définir de nouveaux Contrôleurs de Vue, comme il est possible de créer de nouvelles class graphiques GROW ou d'objets PAC. L'avantage de Serpent est la description déclarative à partir laquelle le noyau Serpent gère automatiquement les enchaînements du dialogue et ceci à n'impor quel niveau d'abstraction. Cette gestion automatique bénéficie avantageusement des mécanismes q moteur d'inférence d'OPS83. Dans GROW, l'aspect déclaratif est limité à l'expression de dépendance entre les données. Dans APEX, les éléments déclaratifs sont restreints aux liaisons. Da MacApp, il n'y a pas de description déclarative.

4.3. MacApp et EZWin

MacApp [Schmucker 86a] est réalisé en Object Pascal [Schmucker 86b] au dessus du Toolbox q Macintosh. Sa fonction première : fournir au programmeur la colle qui fait tant défaut au Toolbox. I l'assemblage se présente sous forme de classes d'objets qui, comme il se doit, sont surchargeables extensibles par spécialisation. Si MacApp fait un usage intensif des possibilités de la programmati par objet, son respect pour les principes d'architecture est pour le moins contestable.

On retrouve dans la classe TApplication, la fonction centrale d'un Contrôleur avec la bouc principale de contrôle ; les concepts du domaine peuvent se formaliser par des spécialisations q

TDocument et les présentations par des formes de TView. Si ces trois classes répondent à peu près au trois facettes d'un système interactif, MacApp n'inclut aucun mécanisme de transfert entre formalismes : les exemplaires des classes ci-dessus se référencent directement mettant en péril certain l'indépendance entre le logiciel qui manipule les concepts du domaine et le logiciel chargé de la présentation.

Très similaire à MacApp par l'esprit et la forme, EZWin [Lieberman 85] est réalisé avec les Flavors de ZetaLisp sur Symbolics 3600. Comme MacApp, EZWin s'en remet au modèle de la programmation par objets. Il définit un ensemble organisé de classes prêtes à l'emploi et surchargeables. On y retrouve les notions d'application et de commandes génériques mais EZWin explicite un peu mieux que MacApp la distinction entre objets de présentation et concepts du domaine.

5. Avantages et inconvénients

La notion de squelette d'application est une approche intéressante à la construction des systèmes interactifs mais elle a aussi ses inconvénients.

5.1. Les avantages

Le programmeur hérite automatiquement d'une architecture logicielle saine qui facilite et encourage la mise au point itérative de l'interface. Nous avons évoqué le cas trop fréquent où les fonctions de l'interface et celles de l'application s'entremêlent au point de bloquer l'évolution du système interactif. Un squelette d'application bien conçu évite cet écueil.

Le programmeur se voit déchargé de cette tâche longue et complexe qui consiste à déterminer un assemblage correct de briques logicielles. Le squelette, développé par des spécialistes, fournit en principe un assemblage optimal prêt à l'emploi.

Le programmeur reçoit, au delà d'une architecture, un ensemble de services dont le niveau d'abstraction se rapproche de ses besoins. Par exemple, si les concepteurs du système de fenêtrage ont jugé trop coûteux de gérer automatiquement le rafraîchissement des fenêtres et si, pour une classe d'applications, le confort fourni par ce type de service supplante les considérations de coût, alors le squelette est là pour intégrer une nouvelle abstraction. De même, la gestion automatique de la

superposition d'objets, un service utile à la mise en œuvre des interfaces de type manipulati directe, peut être greffée sur le squelette.

Conçu selon les techniques de la programmation par objets, le squelette est extensible et soupl Par exemple, le noyau d'APEX peut être enrichi par l'ajout de nouveaux objets de présentation ; et protocole d'échange entre le monde de l'application et le monde de l'interface peut être affiné par définition de nouveaux points d'entrée.

Un squelette d'application est une étape souhaitable en direction de la génération automatiq d'interfaces. Nous verrons au chapitre suivant qu'un générateur d'interfaces produit un systèn interactif à partir de spécifications de haut niveau. Le système résultant s'organise nécessaireme autour d'un noyau d'exécution prédéfini auquel sont automatiquement reliées :

- les fonctions spécifiques au domaine (produites éventuellement avec un outil de spécificati propre au type d'application), et
- les fonctions de l'interface créées à partir de la spécification de l'interface.

Un squelette d'application peut servir de noyau d'exécution aux systèmes interactifs ain assemblés.

A ces avantages s'opposent quelques inconvénients.

5.2. Les inconvénients

Les inconvénients tiennent essentiellement aux difficultés de la réutilisation de logiciel. Si réutilisation de logiciel est un remède à la duplication d'efforts de programmation, elle requie inévitablement une étape d'apprentissage. L'un des reproches que l'on entend souvent prononcer propos des environnements à objets, qui se font les champions de la réutilisation, concern l'identification des services (classes et méthodes) disponibles. Pour réutiliser à bon escient, il fa pouvoir évaluer les différences avec les données du problème à résoudre. Alors seulement, programmeur peut entreprendre les actions de surcharge et d'extension.

Les opérations de surcharge et d'extension sont des actions de programmation qui s'exprime dans les termes de l'environnement d'accueil. Le programmeur retombe inévitablement sur l services de la boîte à outils. S'il est vrai qu'un squelette prépare largement la construction d'u

système interactif, il est également vrai que toute extension nécessite de connaître la boîte à outils support. Les générateurs d'interfaces, avec leur mécanisme de spécification, tentent de remédier à ce niveau de programmation jugé trop bas.

Le dernier inconvénient de la réutilisation est la réutilisation malgré soi. La difficulté ici n'est pas d'ordre cognitif mais technique. Elle concerne la taille des systèmes interactifs construits autour d'un squelette. L'un des attraits du squelette est l'extensibilité. Sous la pression des besoins, la version originale s'enrichit de services "d'utilité publique". Progressivement, le squelette devient un tout énorme dont le programmeur hérite automatiquement sans possibilité de discernement. L'héritage imposé à une application d'une grande quantité de fonctions inutiles peut parfois être perçu comme une limitation sévère.

EN RESUME :

Un squelette d'application est un assemblage logiciel réutilisable et extensible qui assure une lar part des fonctions de l'interaction avec l'utilisateur. Quels que soient les techniques de mise en œuv et l'environnement d'accueil, un squelette doit satisfaire aux conditions d'une architecture saine q garantit l'indépendance entre l'application et les techniques de présentation. Pour cela, le squelette d s'organiser autour d'un composant central, le Contrôleur, qui assure les trois fonctions essentielles communication, de transfert et d'arbitrage entre l'application et la présentation.

Un "bon" Contrôleur est celui qui dispose d'un mécanisme de communication tel que les logicie de l'application et de la Présentation ne se référencent JAMAIS directement mais communique TOUJOURS par l'intermédiaire du Contrôleur. Cette communication s'effectue grâce à un protoco et à des zones de communication idoines. Un protocole adapté est celui qui permet à l'application et la Présentation de s'exprimer dans des termes compatibles avec leur niveau d'abstraction. Les zon de communication sont des combinaisons d'objets de transfert et de bases d'association. Dans s fonction d'arbitrage, le Contrôleur doit inclure le réglage de l'asynchronisme des activités parallèl lorsque ce service, dans le système hôte, est absent ou inadapté.

Pour répondre à sa vocation de logiciel réutilisable, un squelette doit aussi satisfaire les besoins e présentation d'une large classe de systèmes. Dans ce but, il peut inclure des objets graphiqu prédéfinis qui renforcent la cohérence des interfaces, mais il doit avant tout offrir des mécanism d'extension. L'approche par objets, qui autorise l'extension par spécialisation, est un mécanism général dont tout squelette devrait s'inspirer. Au cadre général du modèle à objets doivent s'ajout des mécanismes spécifiques à l'interaction et répondant aux besoins d'une large classe de systèm comme l'expression de contrainte, de dépendance.

Chapitre 13

Générateurs d'Interfaces Interactives

1. Introduction : le problème

Au chapitre précédent, nous avons vu que les squelettes d'application réduisaient sensiblement coûts de développement mais que leur adaptation au cas particulier d'une application impliq l'utilisation des services d'une boîte à outils. A leur tour, les boîtes à outils ont l'inconvénient d' difficiles à appréhender. L'objet des générateurs d'interfaces est de faciliter la tâche de programma en permettant la spécification de l'interface homme-ordinateur dans un langage adapté.

Ce chapitre présente l'état de l'art sur la génération automatique des interfaces homme-ordinat Il est organisé comme suit : un premier paragraphe décrit le principe de la génération d'interfaces prend comme point de départ la technique de la spécification ; un second paragraphe est consacré formalismes de spécification d'interfaces. Le chapitre s'achève sur une évaluation comparative techniques actuelles de génération automatique d'interfaces homme-machine.

Pour simplifier l'expression de l'exposé, j'emploie désormais le terme "générateur d'interfac pour désigner un "générateur d'interfaces homme-machine".

2. Principe de la solution

Un générateur d'interface crée un système interactif à partir d'une spécification. Comme l'indi la figure 13.1, il comprend :

- un analyseur du langage de spécification,
- un compilateur qui produit le code exécutable de l'interface,
- un noyau d'exécution au centre du contrôle du dialogue et
- un mécanisme de liaison qui assemble l'interface, le noyau d'exécution et l'application en système interactif exécutable.

Cette organisation s'appuie sur la distinction modulaire entre l'application et l'interfa L'application est exprimée dans un langage de programmation usuel. Le générateur suppose s existence et n'offre aucun outil pour sa construction. La spécification de l'interface définit présentation des concepts de l'application. Le paragraphe 3 précise la nature des formalisn possibles. Le noyau d'exécution se charge des aspects de l'interaction communs à la cla d'applications visée par le générateur. Ce noyau peut répondre à la définition du squele d'application présenté au chapitre précédent.

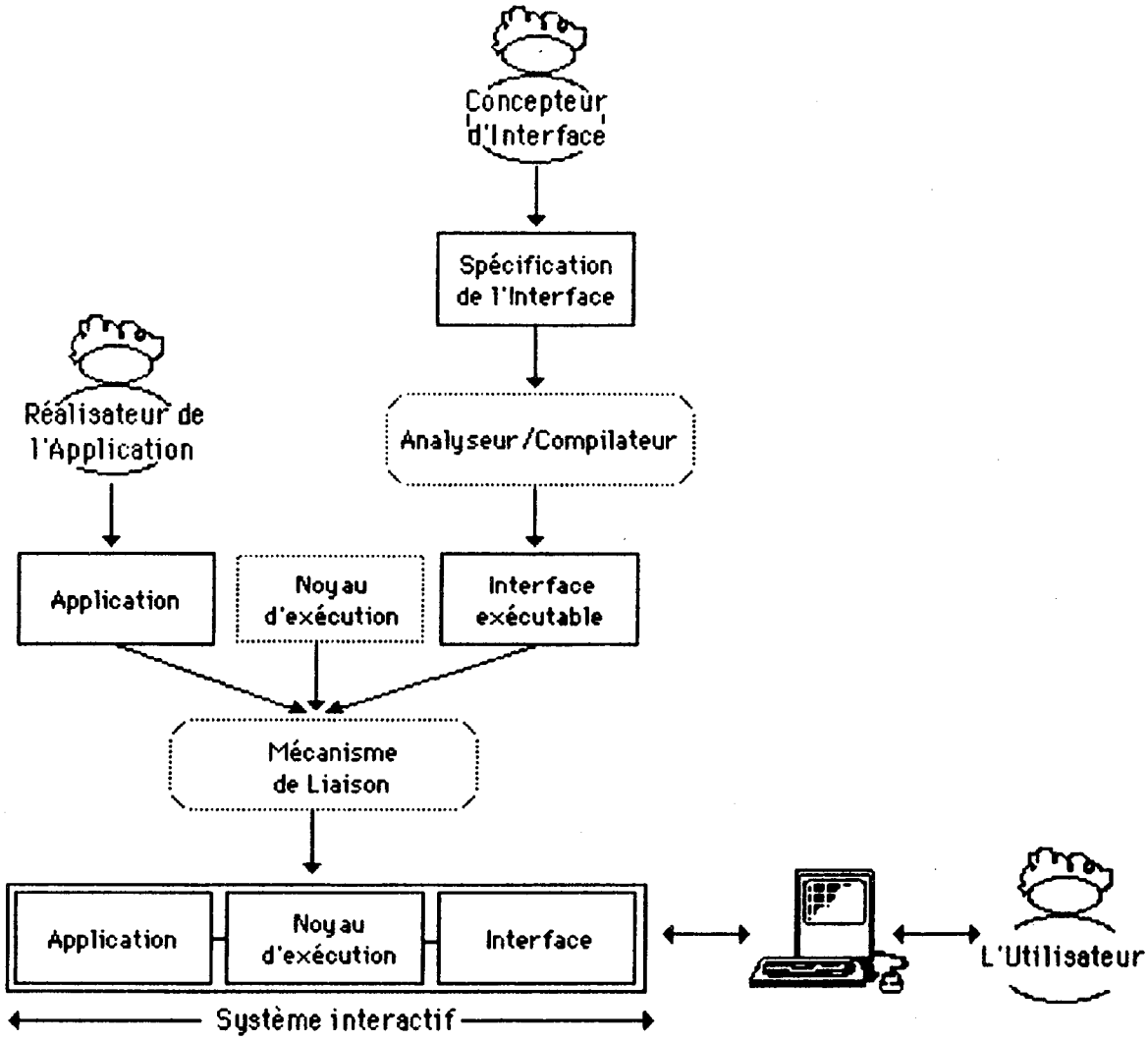


Figure 13.1 : La génération automatique d'interface. En pointillé, les constituants du générateur d'interface.

Le schéma de la figure 13.1 indique les composants de base d'un générateur d'interface procédant par compilation. Comme pour les langages de programmation, le compilateur peut être remplacé par un interpréteur. L'avantage de cette seconde technique est de permettre au concepteur d'évaluer les effets de la spécification au fur et à mesure de la construction de l'interface. L'interpréteur est accompagné d'un éditeur qui interagit avec le concepteur. On parle alors d'environnement de spécification d'interfaces. Comme pour les environnements de programmation usuels, les fonctions de base nécessaires à l'élaboration de la spécification d'interface peuvent être assorties d'un système de gestion de versions.

3. Les formalismes de spécification

Toute spécification pose le problème du choix d'un formalisme. Un formalisme se juge à sa puissance d'expression et à son adéquation à l'expression de la classe de problèmes considérés. La

puissance d'expression détermine le domaine représentable alors que l'adéquation traduit la facilité transcription du problème. La première s'évalue de façon rigoureuse grâce la théorie des langa formels tandis que la seconde s'estime de manière subjective. L'adéquation d'un formalisme cependant largement responsable du choix de la solution finale. Par exemple, dans le domaine l'interaction homme-ordinateur, si l'analyse conceptuelle suggère un type d'interface alors que formalisme n'est pas approprié à son expression, les choix conceptuels seront probablem détournés au profit d'une solution directement exprimable dans le formalisme mais certainem inadaptée à l'utilisateur.

Ce paragraphe passe en revue les formalismes employés dans le domaine de la spécificat d'interfaces en précisant les modèles sous-jacents. Un formalisme servant de véhicule communication d'un modèle, nous pourrons juger de son adéquation à l'expression des interfaces nous référant aux propriétés des modèles d'architecture présentés dans la section II. Les formalisn se répartissent essentiellement en trois catégories : les extensions de formalismes existants à base graphes ou de grammaires, les langages spécialisés et l'habillage graphique interactif.

3.1. Les graphes

Les graphes orientés ont largement été utilisés en spécification. Ils l'ont été également dans domaine de l'interface homme-machine. Retenons les diagrammes de transition et les réseaux Petri.

3.1.1. Les diagrammes de transition

Les diagrammes de transition appliqués à la description des interfaces prennent des form diverses. La version la plus simple consiste à étiqueter les arcs avec les éléments lexicaux d'enti (c.-à-d. les actions physiques de l'utilisateur) et à utiliser les nœuds pour représenter les ét possibles de l'interaction.

D'autres techniques, comme celle des diagrammes de transition récursifs, substituent à représentation plate des diagrammes simples, une organisation structurée qui permet de partitionnei dialogue en sous-unités logiques. Dans ce cas, certains arcs (ou certains nœuds) sont décorés noms de sous-graphes. Ces sous-graphes se comportent comme des procédures récursives : peuvent s'appeler et la traversée d'un arc étiqueté d'un nom de sous-graphe est complète lorsque parcours du sous-graphe est achevé ; lorsque les noms des sous-graphes sont associés aux nœuds, sous-graphe est exécuté dès que le système atteint l'un de ces nœuds. Afin de reconnaître c langages à contexte lié, les diagrammes de transition récursifs sont parfois étendus en associant a

arcs des registres et des fonctions booléennes. Ces registres servent de variables de travail et permettent de modéliser la notion de contexte. Les fonctions ont accès aux registres en lecture et en écriture. Un arc est traversé si l'élément lexical d'entrée est reçu et si la fonction booléenne retourne la valeur VRAI.

Quelle que soit la forme choisie, l'objectif est identique et l'évolution dans le graphe relève du même principe général : reconnaître le langage de l'interaction à l'aide d'une machine d'états finis et activer les actions sémantiques par des appels de procédures. Ces procédures sont supposées disponibles et écrites à la main dans un langage de programmation. Le démarrage d'une session est représenté par un état initial. Le passage d'un état à l'autre s'effectue le long des arcs étiquetés. Le système se trouvant dans un état donné et l'utilisateur effectuant une action, l'arc suivi est celui dont les conditions d'activation sont satisfaites. Si aucun arc ne peut être traversé, c'est que l'utilisateur vient de commettre une erreur. En général, chaque nœud comporte un arc qui conduit au traitement des anomalies.

Le système de Jacob [Jacob 84] et Rapid/Use de Wasserman [Wasserman 85] sont deux exemples de système de spécification à base de diagrammes de transition. Tous deux introduisent des extensions au schéma de base des diagrammes de transition récursifs : des variables simples, des conditions et des expressions de sortie textuelles attachées aux nœuds ou aux arcs. Tous deux se limitent à la description d'interfaces textuelles et écartent l'usage de la souris.

3.1.2. Les réseaux de Petri

Les réseaux de Petri adjoignent aux notions d'état et de transition, l'expression d'activités parallèles. Dans le domaine de l'interface homme-ordinateur, ils ont été utilisés comme outil de description. Les "diagrammes de technique d'interaction" (interaction technique diagrams) de Foley [Foley 81] sont des réseaux de Petri étendus qui permettent de décrire les aspects cognitifs, perceptuels et moteurs impliqués dans la réalisation de tâches. Foley introduit plusieurs catégories de nœuds : les étapes fonctionnelles de l'utilisateur, les étapes fonctionnelles du système, les points de décision de l'utilisateur et ceux du système. Les réseaux de Petri à structures de données de M.F. Barthet servent de support formel à l'analyse conceptuelle de tâche [Barthet 86]. Ils étendent la notion de jeton à celle de structures de données et introduisent la représentation du temps.

Alors que les réseaux de Petri ont été appliqués à la description de tâches informatisées, ils n'ont pas été utilisés pour la spécification d'interfaces en vue d'une génération automatique.

3.2. Les grammaires hors contexte

En spécification d'interfaces, les éléments terminaux d'une grammaire BNF servent à représent les actions physiques élémentaires de l'utilisateur ; les non-terminaux et les règles de productio définissent l'espace des commandes.

SYNGRAPH (SYNtax directed GRAPHics) est un exemple de générateur d'interface à base d grammaire [Olsen 83]. Contrairement à RAPID/USE et au générateur de Jacob, SYNGRAPH e capable de prendre en compte les dispositifs nécessaires aux applications graphiques. On y retrouv les notions d'icône et d'unités logiques (localisateur, choix, etc.) présentées au chapitre 6. Comm pour les générateurs à base de diagrammes de transition, une spécification SYNGRAPH explici l'activation des actions sémantiques sous forme d'appels de procédure écrites dans un langage d programmation usuel (en l'occurrence Pascal). Une spécification SYNGRAPH est traduite en u programme Pascal : chaque production donne naissance à une procédure Pascal dont les paramètr formels doivent être spécifiés par le concepteur d'interface en respectant la syntaxe Pascal.

3.3. Les systèmes de productions

Au delà du cas particulier des grammaires BNF, les systèmes de productions peuvent servir d base à la définition de langages parallèles. Toutes les règles dont la partie gauche est satisfaite p l'état actuel du système sont activables. Le non-déterminisme d'une telle situation est résolu par u contrôleur de stratégie. Ce type de fonctionnement est à la base des systèmes de productions utilisé en Intelligence Artificielle. Il l'est aussi pour ERS (Event Response System), un langage d spécification d'interfaces motivé par l'expression des actions simultanées de l'utilisateur sur le dispositifs de commande [Hill 87a, Hill 87b].

La partie gauche d'une règle ERS (c.-à-d. la condition) est une conjonction de valeurs qu comprend un nombre quelconque d'indicateurs booléens et éventuellement la désignation d'u événement. Les indicateurs jouent le rôle de mémoire d'état. Les événements servent à communique avec l'environnement (c'est-à-dire, l'application, les objets de présentation et les unité d'entrée/sortie). Ils sont mémorisés dans une file d'attente unique. Les événements et les indicateur sont des structures de données dont les champs peuvent être lus et modifiés. Une règle ERS e satisfaite lorsque tous les indicateurs de la condition sont positionnés et, si la condition désigne u événement, lorsque cet événement est en tête de la file. Toutes les règles satisfaites sont exécutées e parallèle. L'exécution d'une règle retire l'événement de la file, désactive les indicateurs de la conditio et exécute les actions de la partie droite. Ces actions incluent l'envoi de messages, le positionnemen d'indicateurs, et l'affectation de champs d'indicateurs et d'événements. Le détail du fonctionnemen d'un interpréteur ERS peut être consulté dans [Hill 87a, p. 45].

ERS est un automate qui permet de reconnaître les langages réguliers et d'exprimer aisément la composition de tels langages. Cette dernière propriété ouvre la voie à la spécification de dialogues parallèles.

3.4. Les langages déclaratifs spécialisés

Un langage déclaratif permet de décrire un phénomène alors qu'un langage procédural permet d'exprimer la suite des traitements qui produisent le phénomène. Dans le cadre de la spécification d'interfaces, le concepteur indique la nature des informations échangées entre l'application et l'interface et la façon dont ces informations se présentent à l'utilisateur. Cette description s'effectue dans un langage spécialisé. Elle est ensuite compilée puis reliée à un noyau d'exécution. Deux systèmes très voisins, Cousin [Hayes 83, Hayes 85] et Domain/Dialogue [Schulert 85], relèvent de cette technique. Le premier est un prototype de recherche, le second est distribué par Apollo Computer. Plus ambitieux mais relevant du même point de départ, il faut retenir IDL [Foley 87] dans le cadre du projet UIDE (The User Interface Design Environment) [Foley 88].

Dans Cousin, la présentation des concepts de l'application s'effectue au sein de formulaires. Ce choix est délibéré. L'originalité de Cousin réside dans ses tentatives de correction automatique des saisies textuelles erronnées. Dans Domain/Dialogue, l'interaction s'effectue essentiellement par l'intermédiaire de formulaires mais l'application peut aussi faire usage de surfaces graphiques. Dans Cousin comme dans Domain/Dialogue, les échanges entre l'application et l'interface s'effectuent à un haut niveau d'abstraction : les données, bien que non structurées (valeurs entières, booléennes, etc.), sont celles que manipule l'application. Le réalisateur de l'application n'a pas connaissance des techniques de présentation utilisées dans l'interface. Les figures 13.2 à 13.6 permettent d'apprécier le travail de spécification d'une interface dans Domain/Dialogue.

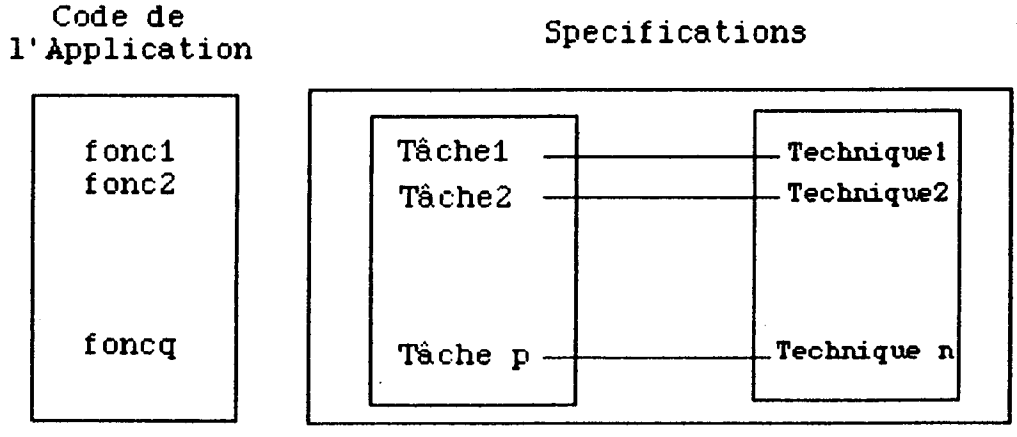


Figure 13.2 : Le modèle de Domain/Dialogue.

La figure 13.2 précise le modèle. A gauche de la figure, on relève un ensemble de fonctio servant de points d'entrée dans l'application. Ces fonctions sont écrites en Pascal, Fortran ou C. droite, les deux composantes de la spécification produites par le concepteur d'interface : les tâches les techniques. Les tâches servent d'intermédiaires entre les fonctions de l'application et l techniques de présentation. Elles sont typées et capables d'effectuer des traitements simples. L techniques de présentation sont des objets interactifs prédéfinis : icônes, champs booléen numériques ou textuels. Lorsque l'application dépose une valeur dans une tâche, celle-ci transm l'effet à la technique qui lui est associée. Inversement, lorsqu'une action de l'utilisateur a un effet s une technique, celle-ci en avertit la tâche qui lui correspond et la tâche répercute l'effet en appelant u fonction de l'application.

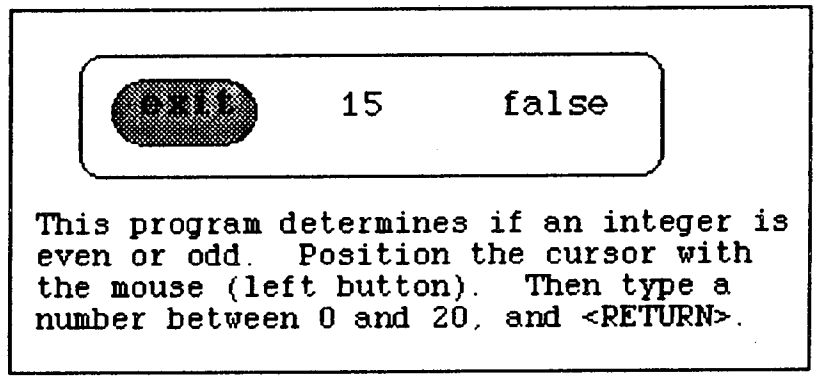


Figure 13.3 : Une interface produite avec Domain/Dialogue.

La figure 13.3 donne l'exemple d'une interface produite à partir des spécifications des figur 13.4 et 13.6 . La tâche exit-task de la figure 13.4 est de type NULL : elle ne rend pas de valeu Lorsqu'elle reçoit l'événement COMP, le contrôle est rendu en sorte que l'application se termine. L tâche number-task est de type INT : elle renferme une valeur entière. Lorsqu'elle reçoit l'événeme COMP, elle vérifie que la valeur transmise dans l'événement est comprise entre 0 et 20. Si c'est l cas, elle appelle la procédure odd-or-even de l'application, sinon, elle affiche un message d'erre dont le libellé revient entièrement à Dialogue.

La technique exit de la figure 13.5 est de type ICON. Elle se présente sous forme d'un rectang aux bords ROUNDED, de taille (100 350) pixels, rempli avec la couleur GREY et dans lequel e inscrite la chaîne de caractères "exit". Lorsque l'utilisateur la désigne avec la souris, elle produit u événement COMP à destination de la tâche exit-task qui lui est associée. La technique number est u champ de saisie de valeurs entières. La frappe d'un retour-charriot provoque l'envoi à la tâch number-task de la valeur saisie dans un événement COMP. Lorsque l'utilisateur enfonce la touch "help" du clavier, le message "you must give an integer from 0 to 20" apparaît dans une nouvell fenêtre. La technique row-bottom exprime la composition horizontale des techniques élémentair exit, number et true-false avec un encadrement et des valeurs d'espacement à respecter.

```
APPLICATION-INTERFACE example

   exit-task:=NULL:
     COMP  =>  <RETURN>

   number-task:=INT:
     COMP  =>  <CALL odd-or-even>
     MIN=0; MAX=20
     END

   true-false-task:=BOOL:
     COMP  => <>
     END

   message-task:=MSG:
     VALUE =
       "This program determines if an integer is even or odd."
       &"Position the cursor with the mouse (left button)."
       &"Then type a number between 0 and 20, and <RETURN>."
     END
```

Figure 13.4 : La spécification de l'ensemble des tâches pour l'interface de la figure 13.3 .

```
USER-INTERFACE example

   exit:=ICON:
     TASK = exit-task;
     BACKGROUND = GREY;
     SHAPE = ROUNDED;
     SIZE = (100 350) PIXELS;
     STRING = "exit"
     END

   number:=INT_FIELD:
     TASK = number-task;
     BACKGROUND = OFF
     SHAPE = ROUNDED
     HELP-TEXT = "you must give an integer from 0 to 20"
     END

   true-false:=BOOL-FIELD
     TASK = true-false-task;
     BACKGROUND = OFF;
     SHAPE = ROUNDED;
     HELP -TEXT = "true=even number" & "false = odd number"
     END

   row-bottom:=ROW
     BACKGROUND = ON;
     ORIENTATION = HORIZONTAL;
     BORDER-WIDTH = 10; DIVISION -WIDTH = 5;
     OUTLINE = ON; SHAPE = ROUNDED;
     CONTENTS = (exit number true-false)
     END
```

Figure 13.5 (début) : La spécification des techniques de présentation.

```
message:= DISPLAY TEXT
  TASK = message-task;
  SHAPE = ROUNDED
  END

row-all:= ROW
  BACKGROUND = ON;
  ORIENTATION = VERTICAL;
  BORDER-WIDTH = 10; DIVISION-WIDTH = 5;
  OUTLINE = ON; SHAPE = ROUNDED;
  CONTENTS = (row-bottom message)
  END

std-window:
  CONTENTS = row-all
  END
```

Figure 13.5 (suite): La spécification des techniques de présentation.

Dans le programme d'application de la figure 13.6, la primitive dp-$bool-set-value dépose valeur VRAI dans la tâche true-false-task. Celle-ci, reliée à la technique true-false, fera paraître le tex true à l'écran. La procédure odd-or-even (appelée par la tâche number-task lorsque celle-ci reçc l'événement COMP de la technique number), utilise la primitive dp-$int-get-value pour retirer valeur maintenue dans number-task. Elle effectue ensuite les tests voulus et dépose le résultat dans tâche true-false-task grâce à la primitive dp-$bool-set-value.

```
-- initialiser DIALOG
       dp-$...
-- initialiser les valeurs des tâches
       dp-$bool-set-value(true-false-task,true,status);
-- activer toutes les tâches
       dp-$task-activate (dp-$all-task-group,...);
-- attendre un événement
       dp-$event-wait (......);
-- sortir de DIALOG
       dp-$terminate (...);
```

Un MODULE: la procédure qui vérifie la parité

```
odd-or-even()
   int value-int, value-bool;
begin
   -- acquérir la valeur de la tâche
       dp-$int-get-value (number-task, value-int, status);

   -- vérifier la parité
       if ((value-int/2) == 0) then value-bool = true
       else value-bool = false;
   -- envoyer le résultat à la tâche
        dp-$bool-set-value (true-false-task, value-bool, status);
end
```

Figure 13.6 : Une partie du programme d'application.

Comme Cousin, IDL s'attache à la description du niveau conceptuel et sémantique de l'interaction. Le concepteur définit la hiérarchie des classes manipulées par le système, leurs attributs, et leurs opérateurs assortis de préconditions et de postconditions. Contrairement à l'utilisateur de Dialogue, le concepteur IDL n'est pas concerné par les techniques de présentation. Cet aspect est pris en charge par un autre composant de l'environnement de conception d'interfaces.

En raison de son haut niveau d'abstraction, une description IDL peut servir d'entrée à d'autres outils d'aide à la conception. En l'occurrence, la définition des opérateurs avec leurs conditions d'activation permet de produire automatiquement des messages d'aide (indiquer, par exemple, pourquoi une commande est invalide dans le contexte actuel). Une description IDL sert également d'entrée à un système d'aide à l'organisation de l'espace des commandes. Ce système procède par transformations sur la spécification initiale. Les algorithmes de transformation proposés reflètent des stratégies de conception que les auteurs d'IDL ont identifiées empiriquement. Une stratégie couramment pratiquée est la factorisation : observer qu'un paramètre est commun à plusieurs opérateurs et donc instaurer la notion de paramètre courant qui évitera à l'utilisateur de spécifier ce paramètre pour chaque commande. De même, on peut décider de la notion de commande courante, d'ensemble d'objets courants, etc.

3.5. Les systèmes de spécification interactive d'interfaces

L'objectif des outils de spécification interactive d'interfaces est la construction de prototypes d'interface sans programmation. Ils permettent au concepteur d'interface :

- de créer des tableaux de bord (ou formulaires) par assemblage d'objets de présentation prédéfinis,
- de décrire les enchaînements entre les tableaux de bord, et
- d'associer les éléments de l'interface aux concepts de l'application.

Contrairement aux systèmes à langages spécialisés, les activités d'assemblage, d'enchaînement et d'associations s'effectuent de manière interactive : au langage textuel compilé se substitue un langage graphique généralement interprété. Typiquement, l'interface d'un tel système est spécifiable à partir de lui-même.

Comme pour les langages spécialisés, cette classe d'outils s'appuie sur :

- le principe de la séparation modulaire entre l'application et l'interface et
- sur l'existence d'une base d'objets de présentation d'usage général tels que les boutons, les barres de défilement, les champs de saisie et les menus.

Menulay [Buxton 83] est une première tentative de spécification interactive d'interface. L concepteur place sur l'écran du texte, des potentiomètres, des icônes et des boutons là où l'utilisatei les verra à l'exécution. A chaque objet sensible aux dispositifs de commande, il associe une fonctio sémantique qui sera exécutée lorsque l'utilisateur du système final agira sur l'objet. Le principe d mise en œuvre de Menulay est simple : les associations entre les objets et les fonctions sémantiqu sont compilées sous forme de tables intégrées au noyau d'exécution. Panther [Helfman 87] appliqu la même technique pour associer des fonctions de comportement à des régions de l'écran organisée en hiérarchie.

Comme Menulay, GUIDE (Graphical User Interface Development Environment) [Granor 8(utilise la technique de la compilation mais il autorise un contrôle plus fin sur les conditions d'affichaj des objets de présentation et sur l'enchaînement entre les tableaux de bord.

Trillium [Henderson 86] a été conçu pour simuler des interfaces de photocopieurs. A la différenc de Menulay et de GUIDE, une spécification Trillium est interprétée (Trillium est réalisé e Interlisp-D). Le caractère immédiat de l'interprétation permet au concepteur de se placer directemei dans les conditions de l'utilisation effective. Trillium exploite les avantages de la séparation entre le comportements interne et externe en permettant au concepteur de modifier une présentation graphiqu tout en conservant le même comportement interne et ceci sans programmer. A la différence de GUID et en raison du domaine d'application visé, Trillium offre peu de possibilité d'enchaînement entre le tableaux.

Graffiti [Karsenty 87], SOS [Hullot 86] et le système de Cardelli [Cardelli 87] relèvent du mêm esprit que les outils précédents. Graffiti exploite explicitement le modèle à objets ; SOS et Cardel proposent en sus un langage d'images qui permet, à la façon des boîtes et des objets composés d Domain/Dialogue, de spécifier l'organisation spatiale des "interacteurs" d'une fenêtre. Ur organisation d'interacteurs peut ainsi donner naissance à une nouvelle classe de tableaux de bord doi seul le comportement géométrique est géré de manière générique par le système de spécification. L sémantique de l'objet nouvellement construit est à programmer dans l'application.

Peridot [Myers 87a, Myers 87b] tient son originalité du modèle de construction. A l'inverse de systèmes procéduraux cités jusqu'ici, Peridot découvre, au moyen de règles d'inférence, la clas: d'objet de présentation que le concepteur est en train de construire ; mais comme pour les autr systèmes de spécification interactive, la puissance de Peridot dépend étroitement de sa base c connaissances. Pour les systèmes classiques, la connaissance est celle des objets prédéfinis ; poi Peridot qui ne dispose pas de mécanisme d'apprentissage, la connaissance est l'ensemble des règl qui permettent de déterminer la classe de l'objet en cours d'élaboration.

4. Evaluation comparative

4.1. Les formalismes et les modèles sous-jacents

Les graphes modélisent l'interaction entre un utilisateur et un système par une machine d'états finis déterministe. Un réalisateur d'interfacc pense en termes d'états ; sa tâche consiste à exprimer le séquencement des états dans le temps en relation avec les actions de l'utilisateur.

Les grammaires hors contexte modélisent l'interaction sous forme d'un langage. Un réalisateur d'interface pense en termes de transformations d'éléments non-terminaux ; sa tâche consiste à définir la forme du langage.

Les systèmes de productions modélisent l'interaction comme un ensemble de machines élémentaires capables de répondre à des événements. Un réalisateur d'interface pense en termes d'événements et de processeurs de traitements ; sa tâche consiste à identifier les événements, à définir la nature des traitements, leur enchaînement et leurs points de synchronisation.

4.2. Les graphes et les Grammaires

L'intérêt premier de la spécification d'interfaces à base de graphe et de grammaire est le caractère immédiat de la mise en œuvre grâce à la technique bien maîtrisée de la génération automatique d'analyseurs/compilateurs. Le modèle à événements demande au contraire une réalisation spécifique.

La notion d'état des graphes convient à la représentation des systèmes dont l'interaction avec l'utilisateur fait largement usage de modes. A chaque mode correspond un contexte de travail (par exemple, l'ensemble des commandes valides) que l'on peut représenter par un sous-graphe. Dans les grammaires, la notion d'état est implicite et la représentation d'un mode n'est pas l'objet des grammaires hors contexte. Contrairement aux graphes et aux modèles à événements, les grammaires s'intéressent à la forme de l'interaction non pas au flux du contrôle.

Les diagrammes de transition et les grammaires conviennent à des interfaces dont la structure syntaxique est riche (c'est le cas des interfaces textuelles) et aux systèmes bien particuliers qui n'exigent pas, lors des saisies, une réaction permanente de la part du niveau sémantique. Par exemple, pendant la saisie d'une commande Unix, le noyau, qui est le détenteur des actions sémantiques, n'intervient pas. Seul se manifeste le niveau lexical de l'interpréteur des commandes qui assure l'écho des caractères. Pour les systèmes qui se caractérisent par des retours d'informations permanents, les formalismes développés jusqu'ici autour des graphes et des grammaires n'autorisent

pas une communication suffisante entre les niveaux sémantiques (l'application) et syntaxiqu (l'interface).

L'insuffisance des communications entre l'application et l'interface se manifeste sous de formes: l'exploitation restreinte voire impossible des valeurs de retour des fonctions sémantiques et pauvreté des expressions de sortie. L'appel de fonctions sémantiques pendant la saisie d'une phra (une commande est une phrase) permet d'actualiser les informations de retour lorsque celles dépendent des compétences de l'application. Parmi ces informations, il faut aussi compter les valeu par défaut. Ces besoins se manifestent très souvent dans les interfaces de manipulation directe m aussi dans les interfaces moins dynamiques comme celles des formulaires. Par exemple, dans l systèmes de base de données, la saisie d'un nom de personne inconnu doit faire l'objet d' avertissement immédiat afin que l'utilisateur ajuste au plus vite le plan mental d'exécution de la tâcl Les grammaires hors contexte rendent impossible ce type d'échange qui est de nature contextuel Par exemple, SYNGRAPH permet l'appel de fonctions sémantiques mais ne fournit aucun moy d'en tester les résultats. Dans l'exemple classique de la procédure du "login", la spécification l'interface peut transmettre au système l'identité de l'utilisateur et son mot de passe par l'appel d'u fonction sémantique. Sans la possibilité de tester la valeur de retour, le concepteur ne peut spécifier présentation du succès de la connexion ni préciser la nature de l'erreur, ni, dans ce dernier c proposer une nouvelle tentative!

En théorie, les diagrammes de transition augmentés permettent d'associer une fonction sémantiq à chaque action de l'utilisateur. En pratique, le niveau de détail d'une telle description conduit à d diagrammes volumineux difficiles à lire. En outre, les valeurs de retour des fonctions ne suffisent p à traduire la richesse des expressions de sortie. Nous touchons ici à la seconde composante de pauvreté des échanges entre l'application et l'interface.

Les formalismes de spécification à base de graphes et de grammaires mettent l'accent sur la forr des expressions d'entrée. Si la rigueur de ces formalismes peut être exploitée pour détect l'incohérence ou la complexité structurelle d'un langage de commande [Reisner 82, Foley 84 p.23 ces travaux, en dépit de leur intérêt, ne concernent que la moitié du dialogue. La spécification d expressions de sortie, quand elle est possible, relève du niveau lexical seulement (par exemple "prompt" ou un message d'erreur statique) ; elle est inapplicable au cas des expressions de sor issues du niveau sémantique ; et dans tous les cas, le découplage entre les langages d'entrée et sortie rend impossible la spécification de la réutilisation d'expressions de sortie comme expressio d'entrée.

Les "grammaires multipartie" ("Multiparty Grammars") de Shneiderman [Shneiderman 8 marquent un effort dans le bon sens sans fournir cependant de mécanismes adaptés à la gestion de

sémantique. En particulier, cette grammaire ne permet pas de spécifier une expression de sortie dont la forme dépend de l'expression d'entrée (nous retrouvons le problème général des grammaires hors contexte). Au chapitre 11, nous avons relevé l'existence d'outils de spécification d'images abstraites mais ces formalismes ne sont pas intégrés à de tels outils de spécification d'interfaces.

La dissymétrie entre la spécification des entrées et celle des sorties a comme conséquence inévitable la nécessité de programmer les expressions de sortie de l'application à l'aide des services d'une boîte à outils. Cette programmation qui revient au concepteur de l'application et pour laquelle les générateurs d'interface ne fournissent aucun guide, met en péril la séparation modulaire entre les fonctions du domaine et les techniques de présentation. Résultat : la moitié de l'interface échappe au contrôle du concepteur d'interface ; produite par l'application, elle rend impossible l'ajustement iteratif d'une large part de la présentation sans faire intervenir une nouvelle activité de programmation.

4.3. Les formalismes à événements

Les formalismes à événements visent à améliorer tous les aspects évoqués ici de la communication entre l'application et l'interface. L'événement, à la fois signal de phénomènes et véhicule d'informations, conjugué aux conditions contextuelles des traitements élargit l'éventail des échanges avec l'expression des activités parallèles. Le parallélisme est devenu une caractéristique déterminante des interfaces actuelles. Les grammaires usuelles et les diagrammes de transition ne sont pas prévus pour cela : il faudrait exprimer une composition de langages.

L'expression du parallélisme a conduit à la définition d'un bon nombre de langages tels CSP [Hoare 78], ESTEREL [Berry 86a, Berry 86b, Clement 88], les "State Charts" [Harel 87]. Tous ces langages sont trop généraux pour correspondre aux besoins précis de la spécification d'interfaces. Squeak [Cardelli 85] et ERS [Hill 87a, Hill 87b] sont les premiers exemples d'expression du parallélisme dans la spécification d'interfaces, encore que le niveau d'abstraction visé (usage simultané de plusieurs dispositifs de commande) soit insuffisant à la description de tous les aspects de l'interaction.

Enfin, un inconvénient qui concerne tous les langages d'expression du parallélisme est la difficulté pour le programmeur de produire un logiciel correct et compréhensible.

4.4. Les langages déclaratifs spécialisés

Les langages déclaratifs spécialisés, tels Cousin et Domain/Dialogue, ont l'avantage de libérer le

concepteur d'interface de la gestion de l'enchaînement des événements. Ces aspects son automatiquement pris en charge par le noyau d'exécution. Le concepteur se borne à spécifier la natur des informations échangées entre l'application et l'interface, leurs conditions de transfert et l présentation des concepts. Pour Cousin comme pour Domain/Dialogue, les échanges entr l'application et l'interface s'effectuent à un haut niveau d'abstraction, c'est-à-dire dans les termes d l'application. L'avantage est la réalisation de logiciels d'application indépendants de la présentation.

Les restrictions proviennent du caractère figé des techniques de présentation dont le comportemen ne peut être surchargé et dont l'ensemble ne peut être étendu. En conséquence, ce type d'outil n convient qu'à un ensemble restreint d'applications. Toutefois, par souci de généralité Domain/Dialogue offre une classe élémentaire de techniques de présentation : les surfaces graphique qui acceptent tous les opérateurs de tracés graphiques et textuels. Domain/Dialogue permet ains l'expression des besoins spécifiques mais il ouvre aussi la porte au danger permanent de l formulation de l'interface dans l'application. En tout état de cause, Cousin et Dialogue imposent a concepteur l'apprentissage d'un nouveau langage de programmation, celui de la spécification.

4.5. Les systèmes interactifs de spécification d'interfaces

Les outils de spécification interactive apportent une réponse au problème de l'apprentissage d'u langage par un habillage graphique de manipulation directe. L'avantage est triple : réduire la par d'effort de mise en œuvre, permettre au concepteur d'évaluer sur le champ l'effet de la spécification e donner à des non-informaticiens la possibilité de réaliser ou de personnaliser des interfaces. E réalité, la construction d'interface sans programmation n'est possible que pour les applications qu répondent au schéma prévu par l'outil. Comme pour Cousin et Dialogue, la spécification interactiv repose sur l'existence d'une base d'objets de présentation. Si celle-ci ne répond pas aux besoins d l'application, le concepteur doit avoir recours à la programmation soit pour construire de nouveau objets de base, soit pour élaborer des objets composés assortis d'un comportement sémantiqu personnalisé.

L'activité de programmation nécessaire à la personnalisation de l'interface s'effectue dans u langage de programmation traditionnel ; elle a lieu dans l'environnement du langage d programmation considéré, non pas dans celui de l'outil de spécification. Séparée de l'activité d spécification, elle est perçue comme une rupture, une déviation de l'objectif central. HyperCar [Harvey 88] dont la vocation initiale n'est pas la spécification d'interfaces mais la création de réseau d'idées, est le premier système à concilier la construction interactive avec la programmation.

Comme les outils de spécification interactive, HyperCard permet au concepteur d'assembler des objets prédéfinis mais il offre la possibilité d'en redéfinir le comportement, de créer des dessins quelconques et de leur insuffler une vie. La surcharge du comportement et la définition de nouveaux agents s'effectuent par programmation dans le langage HyperTalk [Shafer 88]. Chaque objet comprend une partie visible et un "script" exprimé en HyperTalk. Le script décrit les actions à entreprendre lorsque l'objet reçoit un événement (par exemple, un clic souris). L'édition du script s'effectue comme l'édition de l'apparence graphique. Si le modèle d'HyperCard et le langage HyperTalk ne cherchent pas à distinguer le comportement abstrait du comportement de présentation, le système est néanmoins intéressant pour trois raisons :

1. la modularité : une application est un réseau d'objets,
2. l'extensibilité sémantique : un comportement est toujours programmable,
3. l'intégration des différentes facettes de la construction : la réutilisation d'objets et la création constituent une même activité d'édition.

Modularité, extensibilité et édition interactive intégrée ont été nos principes directeurs dans la réalisation de CONCEPT [Geiben 88]. Ce système permet de construire de manière interactive des tableaux de bord à partir de classes d'objets prédéfinies. Ces classes sont les entités de la boîte à outils portable PIXIA réalisée pour l'environnement de programmation Delphia-Prolog. Les objets manipulables dans CONCEPT réutilisent l'apparence graphique des objets PIXIA mais leur comportement interne s'exprime dans un script éditable. Le langage de programmation d'un script d'objet CONCEPT est un habillage de Prolog, l'objectif étant la spécification d'interfaces pour des applications réalisées en Prolog. Contrairement à HyperCard, CONCEPT respecte la distinction modulaire entre les fonctions de l'application et celles de l'interface. CONCEPT est le résultat d'un projet d'étudiants de l'E.N.S.I.M.A.G. A ce titre, il est un prototype élémentaire qui justifie de nombreuses améliorations. Il montre néanmoins l'intérêt de l'approche qui intègre l'édition graphique par manipulation directe à l'édition du comportement par programmation.

Un outil comme CONCEPT donne accès à la programmation. Ce faisant, il vient contrarier l'objectif initial des outils de spécification interactive qui se veulent utilisables par des non-informaticiens. Si l'on considère comme essentielles l'extensibilité et la généralité, la tâche de programmation est inévitable. Si celle-ci est incontournable, il faut la rendre moins agressive, moins exigeante par la forme. Concernant la forme, la programmation visuelle est une voie qui mérite d'être étudiée. Elle consiste à représenter graphiquement les séquences de traitement en s'inspirant du modèle des flux de données. VIP [VIP 86] développé pour construire des applications au dessus du Toolbox du MacIntosh, reproduit la technique des organigrammes abandonnée depuis longtemps dans l'enseignement de la programmation. VIP propose un simple habillage graphique qui ne permet pas de contourner les difficultés inhérentes à la programmation.

Fabrik [Ingalls 88], réalisé en Smalltalk, répond davantage à la demande. Ce système s'inspire d la métaphore des jeux de construction appliquée au modèle des flux de données. L'environneme propose un ensemble de briques de base (des objets abstraits tels que l'addition) mais aussi des obje graphiques (tels que des rectangles) qu'il est possible de relier par des fils connecteurs. La liaisc s'effectue de manière interactive par manipulation directe. Fabrik vérifie la validité des types d informations en transition sur les fils afin qu'une sortie d'objet soit compatible avec l'entrée de l'obj qui lui est relié. Ce système propose également un mécanisme d'abstraction en sorte qu'un ensemb d'entités puisse être considéré comme une entité à part entière. Fabrik étant un environneme interprété, l'effet de la construction est immédiatement visible. Les exemples donnés par Ingalls da [Ingalls 88] veulent démontrer l'applicabilité de l'outil à la construction de nouveaux objets d présentation.

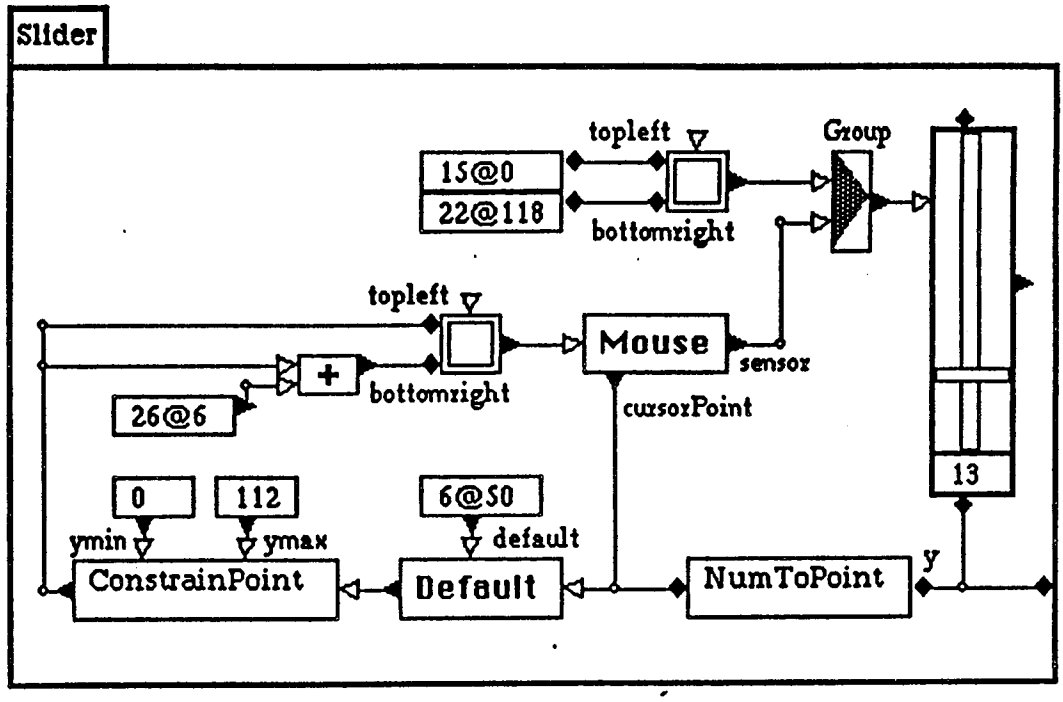


Figure 13.7 : Un programme Fabrik pour la construction d'une jauge.

La figure 13.7 représente le programme d'une jauge dont l'indicateur et la valeur évoluent avec l actions de balayage de la souris. Au vu de cet exemple, on peut s'interroger sur la complexité visuel des programmes autres que les simples cas d'école. D'une manière générale, le pouvoir d'expressio d'une technique de représentation graphique dépend de paramètres précis (qu'il n'est pas toujou facile de préciser). Parmi ces paramètres, le volume d'information semble être un critère déterminat de la complexité : une technique de visualisation qui convient pour une faible quantité d'information pour une tâche donnée devient inopérante en d'autres circonstances. Dans ce domaine, il reste enco beaucoup à inventer au delà des techniques holophrastiques usuelles.

EN RESUME

Un générateur d'interfaces crée un système interactif à partir d'une spécification. Cette spécification est une description textuelle exprimée dans un langage spécialisé (Cousin, Domain/Dialogue) ou dans une notation, extension de formalismes existants (graphes de transitions, grammaires formelles). La tendance actuelle se tourne vers la construction interactive d'interfaces (SOS, système de Cardelli, Grafitti, Peridot).

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

D'une façon générale, les outils développés jusqu'ici autour des automates d'états finis ne spécifient que la moitié du dialogue (les expressions d'entrée) et ne garantissent pas des échanges suffisants et permanents entre l'interface et l'application. De ce fait, ils laissent à l'application la charge de générer les expressions de sortie. La distinction modulaire entre l'application et l'interface, condition nécessaire à la mise au point itérative de l'interface, est donc remise en question ; et le gestionnaire d'interface, laissé à l'écart de la production des expressions de sortie, ne peut réutiliser les expressions produites par l'application comme expressions d'entrée. La pauvreté des échanges entre l'application et l'interface rend difficile, voire impossible, l'entrelacement de l'analyse d'expressions d'entrée partielles avec la production d'expressions de sortie. La qualité sémantique de l'écho et des valeurs par défaut est donc mise en péril.

Les formalismes à événements visent à améliorer la qualité des échanges tout en tenant compte de l'expression du parallélisme. Malheureusement, les outils disponibles dans cette catégorie concentrent l'attention sur les techniques d'interaction de bas niveau (les dispositifs de commande en particulier).

Les systèmes déclaratifs à langage spécialisé ont l'avantage de libérer le concepteur de la spécification des enchaînements des traitements. La contrepartie est une perte en généralité que limite une base non extensible d'objets de présentation.

Les systèmes de spécification interactive ont pour objectif de permettre la construction d'interfaces

sans programmation avec la possibilité d'évaluer immédiatement l'interface créée. Comme le systèmes à langage spécialisé, ces outils sont limités par la base d'objets de présentation. La généralit et l'extensibilité passent inévitablement par la programmation. Cette tâche doit alors être intégrée aus naturellement que possible à l'environnement de spécification. A cette fin, les langages d programmation visuels méritent d'être envisagés.

Si l'intégration des diverses fonctions d'un système de spécification doit être harmonisée, il e d'autres fondements qui devraient aujourd'hui être abordés. Jusqu'ici, aucun outil de spécification n exploré le cas des environnements multiapplication, aucun d'entre eux n'a traité le problème de l répartition (applications de type collaboration), et aucun n'a abordé le sujet délicat de l'adaptabilit automatique.

Conclusion

La contribution de la thèse.

Ce travail contribue au domaine de l'interaction homme-ordinateur sous deux formes : d'une part, il propose une organisation synthétique de la diversité des connaissances dans le domaine et d'autre part, il apporte un ensemble de réponses à quelques-uns des problèmes fondamentaux de la réalisation des systèmes interactifs.

Une Synthèse

La synthèse exprimée dans ce travail est organisée selon les deux axes essentiels de la recherche sur les interfaces homme-ordinateur : celui des sciences cognitives et celui de l'informatique. L'une et l'autre disciplines ont pendant trop longtemps évolué sur des chemins parallèles avec pour conséquence, l'existence de systèmes informatiques inadaptés, culpabilisants et traumatisants. Une première retombée de ce travail est un début de rapprochement entre les deux domaines. Le moyen choisi est sans doute bien modeste mais certainement accessible : enseigner aux lecteurs informaticiens les lignes de pensée marquantes des sciences cognitives et présenter les techniques informatiques qui permettent de les prendre en compte (au moins partiellement).

Les sciences cognitives procèdent selon deux démarches : la théorie et l'expérimentalisme. Les théories servent à prédire ou à expliquer le comportement général du sujet humain face à l'utilisation d'un système informatique alors que les principes pratiques de la démarche expérimentale suggèrent des solutions pour des cas bien cernés. Les premières sont soit trop informelles, soit plus techniques mais alors trop réductrices. Les seconds sont trop nombreux et exigent, pour les appliquer, un savoir-faire étranger à la formation de l'informaticien. En dépit de leur manque de rigueur et de leur diversité, les unes et les autres définissent des points de référence que ce travail s'est efforcé de dégager.

Dans l'état actuel des choses, on peut retenir comme référentiel simplificateur une approc "essentiellement" descendante de la conception. Par "essentiellement descendante", il faut entendre nécessité d'initialiser la conception par l'analyse de tâche. Les étapes restantes, la définition fonctionnement sémantique et de la présentation du système, peuvent s'effectuer selon un ordre mo strict.

- Une analyse de tâche identifie les objets et les opérateurs du domaine (appelés aussi variabl psychologiques) ; elle structure l'espace de travail ; elle définit les enchaînements types en les tâches ; elle repère les conditions d'activités parallèles et, si possible, les poir d'interruption. A ce niveau, la description définit l'utilité du système du point de vue l'utilisateur, non pas du point de vue de la réalisation informatique. Les tâches atomiques leurs opérandes sont cependant de bons indicateurs pour déterminer les fonctio souhaitables du système. Nous entrons alors dans l'étape de la description du fonctionnem sémantique.

- La définition du fonctionnement sémantique du système doit s'efforcer d'établir une doul correspondance : celle des variables psychologiques avec les concepts du système et celle la structure de l'espace de travail avec celui des fonctions. Ces fonctions et ces concepts so ceux que l'on rendra visibles dans l'interface. Ils constituent donc un sous-ensemble d fonctions et des concepts effectifs du système. Afin de raccourcir la distance sémantique, correspondance entre le monde psychologique et les concepts informatiques devra être aus directe que possible. La présentation des concepts par des variables physiques s'efforcera réduire la distance articulatoire.

- La définition de la présentation devrait être la dernière étape de la conception. Sans chercher être exhaustif, on fera intervenir à ce niveau de description quelques principes pratiqu simples sur la cohérence syntaxique et lexicale, sur le choix d'une métaphore d'interactio sur la forme des retours d'informations.

Il est souhaitable que les différentes descriptions reçoivent le support d'un ou plusieu formalismes. CLG [Moran 81], TAG [Payne 86], UML-GUSL [Green 85], les réseaux de Pé [Barthet 86], les graphes Et-OU, les statecharts de Harel [Harel 87] sont des bases possibl d'expression même si ces formalismes ne sont pas adaptés à l'expression de tous les phénomènes l'interaction, même s'ils ne véhiculent pas toujours les principes d'une solution ni de l'aide pour choix des compromis. En dépit de ces difficultés, il faut savoir que concevoir un logiciel sans support d'une analyse ergonomique préalable n'est pas plus admissible que programmer sans l'app d'une spécification.

Parallèlement aux sciences cognitives, s'est développé sous la pression anarchique des besoin un ensemble confus de techniques informatiques. Conformément à sa première mission de synthè pédagogique, ce travail propose une terminologie, identifie les services nécessaires à la constructio des systèmes interactifs et les organise en niveaux d'abstraction. On obtient ainsi un canevas génér

servant de grille d'accueil à la diversité des outils. Parmi ces outils, certains, tels les systèmes de fenêtrage et les boîtes à outils, constituent les logiciels de base de l'interaction alors que d'autres, tels les systèmes à images abstraites, les squelettes d'application et les générateurs d'interface font encore l'objet de recherches actives.

Bien que l'on sache construire des systèmes de fenêtrage et des boîtes à outils, il est des lacunes pas toujours clairement identifiées mais dont il est bon de connaître les effets sur la qualité des interfaces. Ce travail s'est efforcé de les présenter. En particulier,

- un système de fenêtrage doit distinguer les politiques de présentation des mécanismes de base afin que l'interface d'utilisation des fenêtres puisse être modifiée indépendamment des services de base ; il doit offrir de véritables surfaces virtuelles afin que les programmes clients soient encouragés à gérer convenablement le rafraîchissement avec, pour l'utilisateur, la garantie d'un meilleur confort visuel ; il doit prévoir des stratégies de repliement pour l'allocation des ressources afin d'accroître la robustesse des programmes clients et par voie de conséquence, la sécurité et la confiance de l'utilisateur ; il doit inclure les mécanismes de base pour la mise en place des fonctions couper-coller afin que l'utilisateur puisse, sans restriction et de manière cohérente, réutiliser des informations.

- une boîte à outils doit offrir la possibilité de personnaliser le comportement des entités de dialogue. La personnalisation doit pouvoir s'effectuer par surcharge programmée mais aussi, au moins pour les aspects lexicaux de l'interaction, par l'utilisateur. Cette dernière forme d'ajustement suppose l'existence d'une représentation externe permanente des objets de présentation. Afin de faciliter la mise en œuvre des interfaces de manipulation directe, une boîte à outils doit être capable de gérer la superposition d'objets et l'édition interactive d'objets graphiques structurés. On rejoint ici le problème aigu des techniques d'affichage structuré et à contraintes.

Si l'utilité des systèmes de fenêtrage et des boîtes à outils ne fait aucun doute, ces derniers laissent sans réponse des questions fondamentales. Parallèlement à l'exposé synthétique, ce travail s'efforce d'apporter les réponses attendues.

Un Ensemble de Réponses

Un problème fondamental et incontournable est celui de l'architecture logicielle des systèmes interactifs. Plusieurs modèles ont été proposés çà et là sans cependant s'inquiéter des aspects importants de l'interaction tels que le comportement opportuniste de l'utilisateur, les dialogues à plusieurs fils d'activité et la qualité informative des retours d'information. Le modèle PAC vise à la prise en compte de ces aspects.

A la vue restrictive centralisée de l'assemblage des deux blocs, application et interface, P substitue un continuum de structures qui se répartissent la gestion de l'interaction à différents nive d'abstraction. Un système interactif PAC est une organisation récursive d'agents qui regroup chacun trois formes de compétence : la Présentation, l'Abstraction et le Contrôle. Une Présentat définit un comportement perceptible de l'utilisateur ; une Abstraction définit des fonctions et attributs internes ; un Contrôle est un vérificateur de cohérence qui inclut des opérations de liaiso de traduction entre les formalismes de l'Abstraction et de la Présentation et des fonctions d'arbitr en cas de conflit. L'avantage d'un contrôle réparti, tel que PAC le présente, sur la vue centralisée modèle Seeheim est de

- rendre possibles les dialogues à plusieurs fils d'activité : au lieu d'un automate d'états fi unique qui contraint l'utilisateur à se comporter comme un fichier séquentiel, l'état l'interaction est géré par un ensemble de contrôles locaux coopérants,

- faciliter la mise au point itérative du système : au lieu d'entités liées par la connaissa explicite des autres, une Abstraction et une Présentation communiquent toujours l'intermédiaire d'un Contrôle ; ce mécanisme d'indirection qui permet de désigner destinataire sans connaître son nom véritable, garantit la réutilisation de logiciel e substitution (par exemple le remplacement d'une Présentation par une autre) sans mettre cause l'expression de l'autre,

- encourager la répartition des traitements sémantiques et syntaxiques. Cette propriété présen son tour plusieurs retombées intéressantes : les constituants de l'interface peuv communiquer au niveau d'abstraction voulu. En particulier l'application peut (et d s'exprimer dans les termes qui lui sont propres indépendamment de toute considération présentation ; l'existence d'un continuum sémantique élimine la notion de frontière stricte (ailleurs souvent difficile à déterminer) entre l'application et l'interface ; grâce au continuum est possible d'effectuer des réparations sémantiques sans mettre en péril l'architecture l'ensemble (par exemple, lorsque l'application est une boîte noire, une fonction relevant domaine mais oubliée à la conception, peut être insérée dans l'Abstraction d'un objet PA des domaines sémantiquement distincts peuvent cohabiter sans mettre en cause l indépendance fonctionnelle (par exemple, le domaine de la thermodynamique, objet l'application, et celui de la gravitation qui intervient dans le comportement d'objets présentation, tels le réchaud et le thermomètre, s'insèrent naturellement dans d Abstractions) ; enfin, la distribution de la sémantique permet de déporter dans l'interface fonctions de l'application. Quand elle est possible, cette délégation réduit les échan d'informations entre l'interface et l'application. Elle contribue donc à l'amélioration performances du système.

L'expérience acquise selon les lignes directrices du modèle PAC a permis de formuler des règ méthodologiques de construction des systèmes interactifs (voir le résumé du chapitre 9). Ces règ consignent un savoir-faire jamais explicité jusque-là. En l'absence d'outils automatiques d'aide à conception logicielle des systèmes interactifs, cet ensemble restreint de quelques principes pratiq peut servir de substitut.

Une fois le problème de base traité, ce travail s'est attaché à trois classes de problèmes : l'affichage structuré, la définition d'un squelette d'application réutilisable et la génération d'interfaces.

- A l'heure des premières expériences avec les boîtes du système Adèle, les pertes en performance dues à la gestion d'une structure de données supplémentaire étaient perçues comme une limitation sévère. Avec les progrès du matériel, l'affichage d'informations par l'intermédiaire d'images abstraites est aujourd'hui une technique largement admise.

- L'idée directrice de la notion de squelette d'application est de fournir aux réalisateurs une structure logicielle réutilisable qui assure la plus large part des fonctions de l'interface. Cette thèse propose une définition de cette notion encore mal cernée et en précise les implications. Un exemple de réalisation, APEX, en démontre les principes. Parallèlement à APEX, se sont développées quelques réalisations relevant de la même motivation. Contrairement à la plupart de ces travaux, APEX répond à un modèle d'architecture et fournit un protocole d'échange de haut niveau qui encourage l'indépendance du logiciel d'application vis-à-vis de l'interface. Une autre utilisation d'un tel squelette est celle de noyau d'exécution pour des générateurs d'interfaces.

- Les premiers générateurs d'interface ont été accueillis avec beaucoup d'enthousiasme. Aujourd'hui, avec l'élévation du niveau des exigences, on constate qu'ils conviennent tant bien que mal à des applications dont les concepts s'accomodent d'objets de présentation usuels (boutons, menus, chaînes de caractères, icônes). A la contrainte du nombre restreint de classes de présentation, vient s'ajouter une liste de limitations incompatibles avec les "critères qualité" d'une bonne interaction. En particulier, les générateurs ne sont pas capables de traiter des expressions d'entrée partielles entrelacées avec la production d'expressions en provenance de l'application. Cette incapacité écarte sans appel la mise en œuvre d'interfaces de manipulation directe. Les générateurs actuels ne permettent pas non plus de spécifier une expression d'entrée en réutilisant des expressions de sortie, pas plus qu'ils ne sont capables de gérer des dialogues à plusieurs fils d'activité.

  Ce tableau noir de restrictions mis au clair dans la thèse, peut néanmoins s'éclaircir avec l'ouverture par la programmation. Dans le cadre de la conception d'interfaces, programmer doit être une activité complémentaire à laquelle on a recours lorsque les options de l'éditeur d'interface ne répondent plus au besoin. Dans CONCEPT, nous avons appliqué ce principe d'ouverture en intégrant la programmation à la construction interactive d'interface. Dans sa version actuelle, cet outil est cependant loin de suffire à la tâche.

Comme nous venons de le voir, le travail consigné dans cette thèse a un double objectif : identifier les principes de conception et de réalisation des systèmes interactifs et relever les problèmes non résolus. Cette thèse apporte une réponse au problème fondamental de l'architecture, fondement nécessaire à toute forme de prolongements.

Prolongements et Perspectives.

Il y a deux formes de prolongements possibles. La première est une extension des trava présentés autour de PAC, d'APEX et de CONCEPT ; la seconde, plus ambitieuse, est dissémination d'une compétence grâce à une chaîne d'outils pour la conception des interfaces.

Le modèle PAC a été décrit de manière informelle. APEX, le squelette d'application, est son se support formel. En dépit de son intérêt, ce squelette n'intègre pas toute la puissance du modèle par qu'il est structuré en un nombre fixe de niveaux de raffinement PAC. Si une décomposition plus fi vient à s'imposer, celle-ci doit être programmée explicitement : APEX ne propose aucun disposi pour les opérations de raffinement et d'abstraction.

- Une première tâche consiste donc à intégrer au squelette un mécanisme d'abstraction et raffinement qui supporte le modèle PAC.
- Un second travail est d'extraire d'APEX les éléments indépendants de la boîte à outi sous-jacente par exemple, le mécanisme de liaison et le protocole d'échange de haut nivea L'objectif est de confronter les composants génériques à des environnements autres que cel du Macintosh, par exemple le système GUIDE [Krakowiak 87], afin d'identifer de nouvell extensions.
- Une troisième tâche est d'utiliser le nouveau squelette étendu comme noyau d'exécuti réutilisable d'un générateur d'interfaces. La spécification d'interfaces reste un thèn d'actualité qui mérite une attention particulière. Il y a encore beaucoup à imaginer pour intégr le dialogue à plusieurs fils d'activités et l'adaptation automatique à l'utilisateur, pour concili la construction interactive avec la souplesse d'une programmation sans douleur, pour aider concepteur à faire le "juste choix". Ce dernier point rejoint des idées de projets plus ambitieu

La mise en commun des compétences des ergonomes et des informaticiens pourrait aboutir à réalisation d'une "fabrique d'interfaces". La fonction fondamentale d'un tel environnement est l'aide toutes les étapes de la création d'une interface, depuis l'analyse de tâche jusqu'à sa liaison av l'application. La compétence des ergonomes intervient dans l'analyse de tâche et dans la déterminati de l'Image ; celle des informaticiens se manifeste au niveau des choix d'une architecture et de programmation de l'interface. L'objectif est de modéliser ces deux formes de compétence au sein de fabrique d'interfaces. Le mode de fonctionnement de l'ensemble rappelerait l'esprit de collaborati du système IRENE : lorsqu'il est compétent (ou déterminé), le concepteur prend l'initiative d décisions mais le système vérifie la validité des choix ; inversement, lorsque l'utilisateur igno comment procéder, le système peut, à la demande, proposer une solution. La réponse n'est p garantie "optimale" mais elle est utile comme point de départ à l'élaboration d'une solution mie

adpatée. Le domaine envisagé dans IRENE était bien cerné. Modéliser le savoir-faire des ergonomes, représenter la diversité des applications et des techniques de présentation relèvent d'un autre facteur d'échelle. Néanmoins, ce travail mérite d'être entrepris avec cette vision à long terme d'une chaîne d'outils intégrés.

Annexes

Annexe 1. UN EXEMPLE DE DESCRIPTION CLG

Annexe 2. LE SQUELLETE D'APPLICATION D'APEX

Annexe 1

Un exemple de description CLG

1. Rappel

La figure A.1 synthétise les niveaux de description CLG :

- les niveaux tâche et sémantique pour la composante conceptuelle,
- les niveaux syntaxe et interaction pour la communication,
- les niveaux présentation et unités d'E/S pour la composante physique.

Ces six niveaux, ordonnés par degré d'abstraction, depuis le niveau tâche jusqu'au nivea physique, forment ensemble une description CLG. La quasi-indépendance entre niveaux permet d projeter un niveau donné sous diverses formes dans les termes du niveau inférieur. En particulier, une sémantique donnée, peuvent être associées plusieurs syntaxes. Les opérations de raffinement (de projection successives se répètent jusqu'à ce que le système soit spécifié au niveau de détail voulu

Projection et raffinement s'effectuent selon des règles précises appliquées à des éléments préci Règles et éléments se répartissent essentiellement en quatre catégories :

1. les entités, objets manipulés dans le niveau,
2. les opérations élémentaires de manipulation des entités,
3. les procédures, compositions d'opérations élémentaires,
4. les méthodes, associations de procédures à des tâches exprimant les moyens d'accomplir le tâches.

Un exemple va nous permettre d'illustrer le rôle et les éléments des différents niveaux d description. Nous choisissons un système qui automatise l'émission des forfaits d'une station de ski

2. Un exemple : brève description

Dans la station de ski qui nous intéresse, les forfaits se répartissent en quatre catégories :

1. les forfaits "journée-miniréseau" valides le jour de leur émission et donnant accès à un nombr limité de remontées mécaniques,
2. les forfaits "séjour-miniréseau" valides sur une période donnée et donnant accès a mini-réseau seulement,
3. les forfaits "journée-maxiréseau" valides le jour de leur émission sur tout le réseau de l station,
4. les forfaits "séjour-maxiréseau".

Avec le système Appli-Forfait, le guichetier doit pouvoir produire chacun de ces types de forfait (déterminer la recette du jour. Le directeur a besoin d'une étude statistique sur les forfaits vendu durant la saison.

La description CLG d'Appli-Forfait débute avec la représentation du niveau tâche.

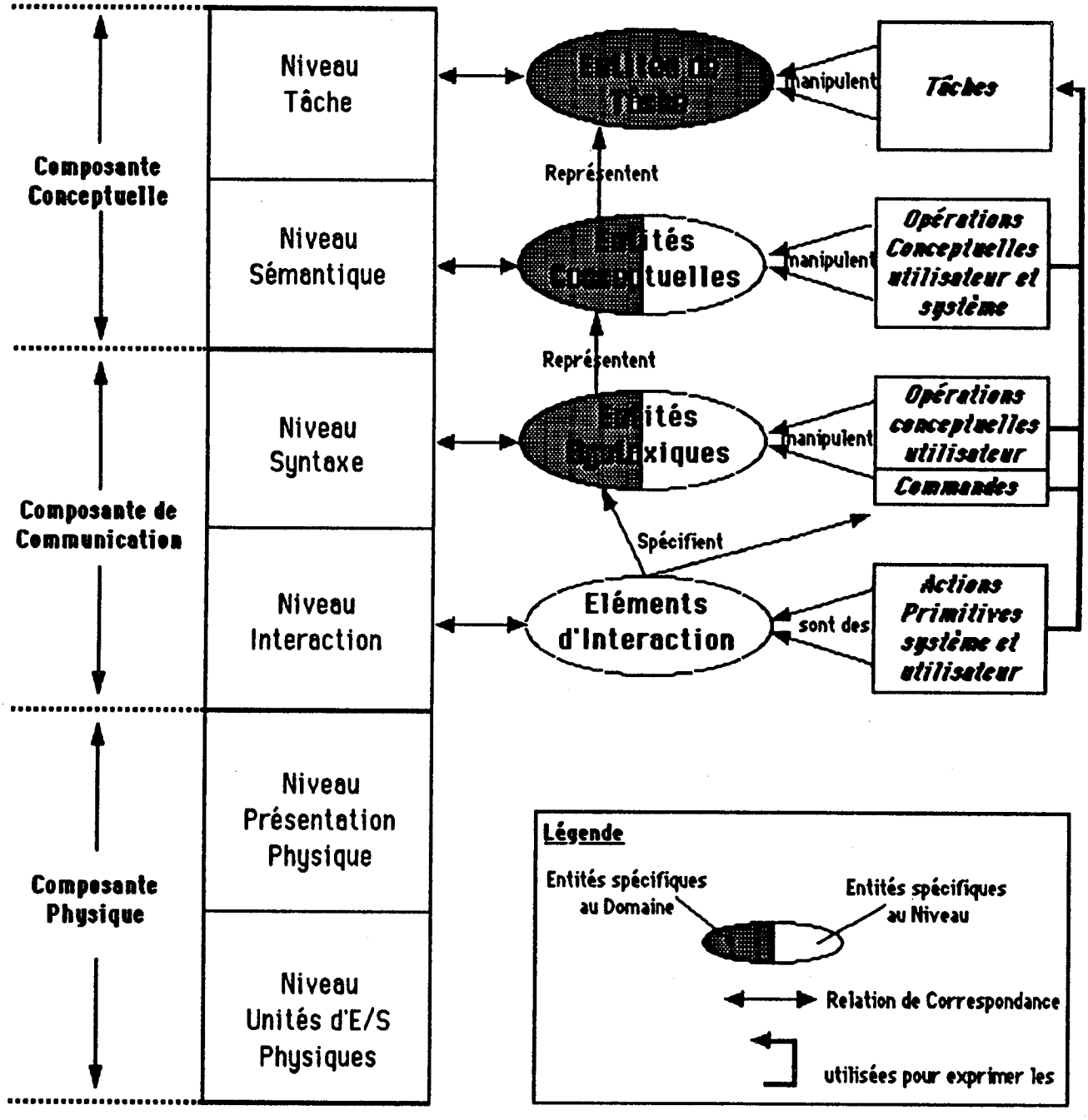


Figure A.1: Les niveaux d'une description CLG : entités et opérations associées.

3. Le niveau Tâche

Le niveau Tâche représente l'utilité du système, c'est-à-dire les besoins et les exigences de l'utilisateur. Il décrit les opérations que le système est supposé accomplir avec et pour l'utilisateur.

3.1 Les entités de tâche

Une entité de tâche est un objet conceptuel du domaine. Sa description réunit les propriétés importantes, les constituants et les relations de l'entité. Dans Appli-Forfait, les entités manipulées au cours des diverses opérations comprennent les notions de forfait, de recette, de mode de paiement,

d'argent reçu, d'argent à rendre et d'ensemble de forfaits vendus. La figure A.2 fournit la descripti des entités de tâche d'Appli-Forfait.

```
T-FORFAIT =
    (UNE ENTITE
      NOM = "forfait"
      (* un objet à produire avec Appli-Forfait.Si l'objet est
      forfait-journée, le code du jour y est imprimé en gros caractères.
      l'objet n'est pas un forfait-journée, il doit comporter le nom du clie
      et la durée de validité.)
    )


T-MODE-DE-PAIEMENT =
    (UNE ENTITE
      NOM = "mode de paiement"
      (* le mode de paiement utilisé par le client pour acheter son forfai
      L'achat peut s'effectuer en liquide ou avec un chèque.)
    )

T-ARGENT-RECU =
    (UNE ENTITE
      NOM = "argent RECU" (* somme d'argent reçu du client lors du réglement
    )

T-ARGENT-A-RENDRE =
    (UNE ENTITE
      NOM = "argent à rendre"
      (* une somme d'argent à rendre au client lors du réglement.))

T-FORFAITS-VENDUS =
    (UNE ENTITE
      NOM = "forfaits vendus"
    (* les forfaits vendus depuis le début de la saison.))

T-RECETTE =
    (UNE ENTITE
      NOM = "recette"(* la recette de la journée))
```

Figure A.2 : NIVEAU TACHE, description des entités de tâche d'Appli-Forfait.

3.2 Les tâches

Une tâche est un objectif que l'utilisateur se fixe et qu'il peut réaliser avec l'aide du système. El se caractérise par des paramètres d'entrée, un résultat, des conditions d'échec, une importance, u fréquence, etc. Dans Appli-Forfait, émettre un forfait, encaisser un forfait et vérifier la caisse sont d tâches fréquentes. Dans notre modélisation, nous ne prendrons pas en compte les tâch exceptionnelles comme l'annulation d'un forfait ou l'émission anticipée de forfaits journaliers.

3.3 Procédures et méthodes de tâche

Une procédure de tâche décrit la réalisation d'une tâche sous forme d'une hiérarchie sous-tâches : une tâche complexe est décomposée en tâches plus simples. Les tâches terminales de

hiérarchie sont les tâches élémentaires du domaine. Dans Appli-Forfait, illustré maintenant par la figure A.3, la réalisation de la tâche T-EMETTRE-FORFAIT implique l'accomplissement des trois sous-tâches élémentaires T-CHOISIR_FORFAIT, T-EDITER-FORFAIT et T-ENCAISSER-FORFAIT.

```
T-EMETTRE-FORFAIT =
    (UNE TACHE
      (* qui consiste à émettre un forfait)
      NOM = "émettre un forfait"
      RESULTAT = (UN T-FORFAIT)
      FAIRE
        (* le composant FAIRE définit formellement la procédure à
        suivre pour accomplir la tâche)
        (EN SEQUENCE:
           (T-CHOISIR-FORFAIT)
           (T-EDITER-FORFAIT  (LE RESULTAT DE T-CHOISIR-FORFAIT))
           (T-ENCAISSER-FORFAIT (LE RESULTAT DE T-EDITER-FORFAIT))
        ))

T-CHOISIR-FORFAIT =
    (UNE TACHE
      (* qui consiste à choisir la catégorie du forfait à délivrer)
      NOM = "choisir la catégorie du forfait"
      RESULTAT = (UN T-FORFAIT))

T-EDITER-FORFAIT =
    (UNE TACHE
      (* qui consiste à remplir un forfait.)

      NOM = "remplir un forfait"
      PARAMETRE = (UN T-FORFAIT)
      RESULTAT = (UN T-FORFAIT)
      (*Remarque : T-EDITER-FORFAIT est le nom choisi par le concepteur
      de dialogue pour désigner cette tâche. "remplir un forfait" est le
      nom par lequel l'utilisateur désigne cette même tâche.
      PARAMETRE est le T-FORFAIT initial que le guichetier va remplir pour
      produire un nouveau forfait.))

T-ENCAISSER-FORFAIT =
    (UNE TACHE
      (* consiste à encaisser un forfait en recevant de l'argent
      selon un mode de paiement et à rendre la monnaie)
      NOM = "encaisser le forfait"
      PARAMETRE = ((UN T-FORFAIT) (UN T-MODE-DE-PAIEMENT) (UN T-ARGENT-RECU))
      RESULTAT = (UN T-ARGENT-A-RENDRE))

T-VERIFIER-CAISSE =
    (UNE TACHE
      (* qui consiste à vérifier la caisse journalièrement)
      NOM = "Vérifier la caisse"
      PARAMETRE = (UN T-FORFAITS-VENDUS)
      RESULTAT = (UNE T-RECETTE))
```

Figure A.3 : NIVEAU TACHE, la description des tâches au niveau tâche.

L'ensemble des procédures de tâche sert à structurer l'espace de travail de l'utilisateur. La nat de cette structuration est essentielle puisqu'elle définit la forme générale de la collaboration entre système informatique et l'utilisateur. Structurer est une première difficulté. Décomposer en est u seconde. Pour la première, il n'y a pas de règle précise si ce n'est correspondre au mieux au mod conceptuel de l'utilisateur. Pour la seconde, Moran propose comme critère d'arrêt de décompositi la dépendance de la tâche sur les fonctions du système. Par exemple, la tâc T-ENCAISSER-FORFAIT est élémentaire dans la mesure où le système calcule automatiquemen prix du forfait et sait classer le forfait dans les ventes. Inversement, si les fonctions de calcul de p et de tri ne sont pas disponibles, la tâche doit être décomposée en sous-tâches de calcul et de tri.

Une méthode de tâche associe une tâche à une procédure de tâche. Elle explicite le lien entre le b à atteindre - la tâche, et un moyen de l'atteindre - une procédure. Pour une tâche donnée, il y a aut de méthodes que de procédures distinctes pour accomplir la tâche. La partie supérieure du schéma la figure A.4 illustre l'association d'une tâche i aux procédures de tâche qui permettent de l'accomp

Comme le montrent les figures A.2 et A.3, CLG organise une description de niveau tâche autc d'éléments de représentation donnés (les tâches, les entités, les procédures et les méthodes) mais description des éléments est essentiellement informelle. Par exemple, la sémantique d'une tâche, composants et le rôle d'une entité sont exprimés en langue naturelle. La notion de méthode de tâc n'est pas explicite dans le formalisme. Seule l'expression des procédures utilisent un formalisme pl rigoureux. Les niveaux inférieurs, tel le niveau sémantique, gagnent en précision.

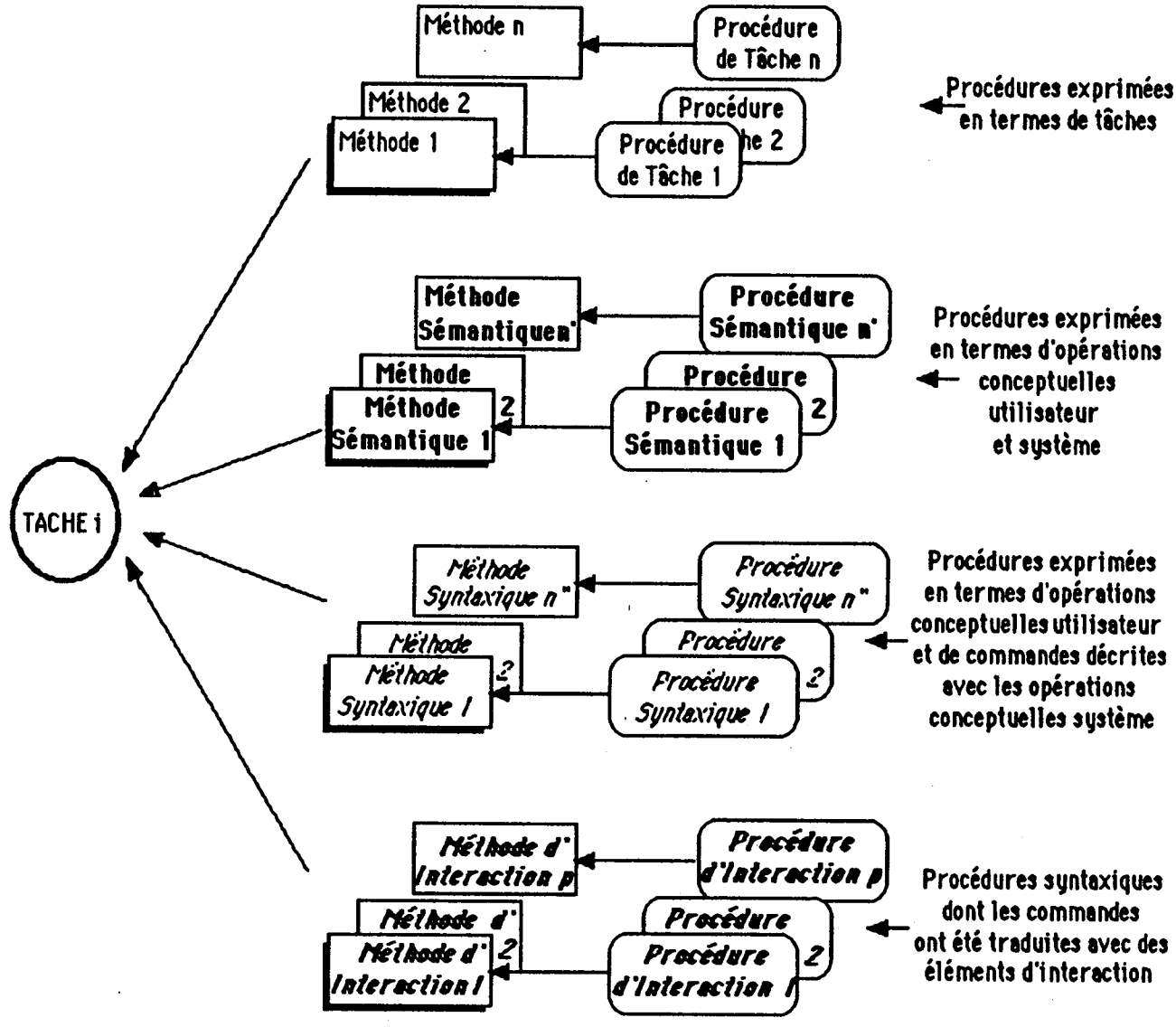


Figure A.4 : Correspondance entre une tâche et les méthodes pour l'accomplir.

4. Le niveau Sémantique

Le rôle du niveau sémantique est d'enrichir le modèle du niveayu tâche avec les concepts mis en jeu par le système. Ces concepts sont décrits avec des éléments de représentation : les entités et opérations conceptuelles, les procédures et méthodes sémantiques.

4.1 Les entités conceptuelles

Toute entité de tâche doit être représentée par une entité conceptuelle.

Dans la description sémantique de la figure A.5, Appli-Forfait est un système qui manipule douze entités conceptuelles : FORFAIT, CHAMP, RECETTE, CATEGORIE-DE-FORFAIT, REPERTOIRE-DES-CATEGORIES, MODE-DE-PAIEMENT, REPERTOIRE-DES-MODES-DE-PAIEMENT, ARGENT-RECU, ARGENT-A-RENDRE, REPERTOIRE-DES-VENTES, ECRAN et IMPRIMANTE. FORFAIT et REPERTOIRE-DES-VENTES représentent respectivement les entités de tâche T-FORFAIT et

T-FORFAITS-VENDUS. REPERTOIRE-DES-CATEGORIES, ECRAN et IMPRIMANTE sont
contraire des concepts spécifiques au système; ils ne représentent aucune entité de tâche.

```
APPLI-FORFAIT =
    (UN SYSTEME
      NOM = "application forfait"
      ENTITES =
         (ENSEMBLE:
            FORFAIT CHAMP RECETTE
            CATEGORIE-DE-FORFAIT REPERTOIRE-DES-CATEGORIES
            MODE-DE-PAIEMENT REPERTOIRE-DES-MODES-DE-PAIEMENT
            ARGENT-RECU  ARGENT-A-RENDRE REPERTOIRE-DES-VENTES
            ECRAN  IMPRIMANTE)
      OPERATIONS =
         (ENSEMBLE:
            EDITER RANGER TOTALISER AFFICHER
            IMPRIMER ENCAISSER))

FORFAIT =
    (UNE ENTITE
      REPRESENTE (UN T-FORFAIT)
      NOM = "forfait"
      PRIX = (UN ENTIER)
      NOMBRE-D-EXEMPLAIRES  = (UN CHAMP-ENTIER)
      CATEGORIE = (UNE CATEGORIE-DE-FORFAIT))

CATEGORIE-DE-FORFAIT =
    (UNE ENTITE
      TYPE-RESEAU = (UN-PARMI: MAXI MINI)
      TYPE-DUREE = (UN-PARMI: JOURNEE SEJOUR)
      (* distingue les catégories de forfait selon le type de réseau auque
      elles donnent droit et la durée de validité.))

JOURNEE =
    (UNE ENTITE
      CODE = (UN CHAMP-CHAINE)
      (* spécifie les attributs particuliers d'un forfait journée :
      un code spécifique au jour.))

SEJOUR =
    (UNE ENTITE
      DATE-DEBUT-VALIDITE = (UN CHAMP-DATE)
      DATE-FIN-VALIDITE = (UN CHAMP-DATE)
      NOMBRE-DE-JOURS = (UN CHAMP-ENTIER)
      NOM-CLIENT = (UN CHAMP-CHAINE-ALPHABETIQUE)
    (* spécifie les attributs spécifiques à un forfait séjour.))


CHAMP =
    (UNE ENTITE
      (COMPOSANT-DE (UN FORFAIT))
      A-EDITER = (UN-PARMI: OUI NON)
      (* le système indique qu'un champ est éditable; L'utilisateur
      décide si le champ est à éditer ou non))
```

```
CHAMP-DATE =
    (UN CHAMP
    VAL = (UNE DATE)
    (* DATE est une entité prédéfinie dans l'environnement CLG.))

CHAMP-CHAINE-ALPHABETIQUE =
    (UN CHAMP
    VAL = (UNE CHAINE-ALPHABETIQUE)
    (* CHAINE-ALPHABETIQUE est une entité prédéfinie dans l'environnement
    CLG.) )

CHAMP-ENTIER =
    (UN CHAMP
    VAL = (UN ENTIER)
    (* ENTIER est une entité prédéfinie dans l'environnement CLG.) )

REPERTOIRE-DES-CATEGORIES =
      NOM = "les catégories de forfait"
      (ENSEMBLE:
         (LA CATEGORIE-DE-FORFAIT TYPE-RESEAU=MAXI TYPE-DUREE=JOURNEE)
         (LA CATEGORIE-DE-FORFAIT TYPE-RESEAU=MAXI TYPE-DUREE=SEJOUR)
         (LA CATEGORIE-DE-FORFAIT TYPE-RESEAU=MINI TYPE-DUREE=JOURNEE)
         (LA CATEGORIE-DE-FORFAIT TYPE-RESEAU=MINI TYPE-DUREE=SEJOUR)
      (* les 4 sortes de forfaits possibles.))

MODE-DE-PAIEMENT =
    (UNE ENTITE
      NOM = "mode de paiement"
      REPRESENTE  (UN T-MODE-DE-PAIEMENT)
      VAL = (UN-PARMI: LIQUIDE CHEQUE)
      (* spécifie les modes de paiement possibles.))

REPERTOIRE-DES-MODES-DE-PAIEMENT =
      (ENSEMBLE: LIQUIDE CHEQUE)

ARGENT-RECU =
    (UNE ENTITE
      NOM = "montant de l'argent reçu du client lors du réglement"
      REPRESENTE  (UN T-ARGENT-RECU)
      VAL = (UN ENTIER))

ARGENT-A-RENDRE =
    (UNE ENTITE
      NOM = "montant de l'argent à rendre au client lors du réglement"
      REPRESENTE  (UN T-ARGENT-A-RENDRE)
      VAL = (UN ENTIER))


REPERTOIRE-DES-VENTES =
    (UNE LISTE
      NOM = "forfaits vendus"
      REPRESENTE (DES T-FORFAITS-VENDUS)
      DE (UN FORFAIT)
      (* cette liste contient l'ensemble des forfaits vendus depuis le
      début de la saison))
```

```
RECETTE =
   (UNE ENTITE
     REPRESENTE (UNE T-RECETTE)
   (* la recette de la journée))

ECRAN =
   (UNE ENTITE
     (* sert à afficher des informations sur un écran physique. I
     souci de simplification, nous supposons que l'ECRAN e
     suffisamment grand pour afficher l'image d'un forfait et c
     recettes.))

IMPRIMANTE =
   (UNE ENTITE
     (* sert à imprimer des informations sur une imprimante réelle.))
```

Figure A.5 : NIVEAU SEMANTIQUE, les entités conceptuelles d'Appli-Forfait.

4.2 Les opérations conceptuelles

Les opérations conceptuelles sont les actions élémentaires applicables aux entités conceptuell Elles se caractérisent par des paramètres d'entrée, un résultat, des conditions d'échec, etc. Elles répartissent en deux catégories :

- les opérations réalisées par le système telles que EDITER, RANGER, TOTALISE AFFICHER, IMPRIMER, ENCAISSER (se reporter aux figures A.1 et A.6),
- les opérations servant à modéliser les actions conceptuelles de l'utilisateur comme par exemp CHERCHER et LIRE (voir les figures A.1 et A.6). CHERCHER et LIRE traduisent le besc d'informations de l'utilisateur, besoin que le système se doit de satisfaire au moment précis il est exprimé dans une procédure.

```
EDITER =
   (UNE OPERATION-SYSTEME
     (* a pour paramètre un CHAMP OBJET1 dont la valeur est replac
     par une valeur OBJET2. le type de la valeur de OBJ2 doit être celui
     composant VAL de OBJET1. MEME-TYPE-QUE est une opération standard
     CLG.)
     OBJET1 = (UN PARAMETRE VALEUR = (UN CHAMP))
     OBJET2 = (UN PARAMETRE VALEUR = (UN (MEME-TYPE-QUE (VAL DE OBJET1))))
   )


RANGER =
   (UNE OPERATION-SYSTEME
     (* range l'ENTITE OBJET1 dans une LISTE)
     OBJET1 = (UN PARAMETRE VALEUR = (UNE ENTITE))
     DANS (UN PARAMETRE VALEUR = (UNE LISTE))
   )


TOTALISER =
   (UNE OPERATION-SYSTEME
     (* consiste à calculer la recette du jour à partir d'u
     REPERTOIRE-DES-VENTES)
```

```
      OBJET = (UN PARAMETRE VALEUR = (UN REPERTOIRE-DES-VENTES))
      RESULTAT = (UNE RECETTE))


ENCAISSER =
   (UNE OPERATION-SYSTEME
     (* consiste à encaisser un forfait selon un mode de paiement et, si
     besoin est, à rendre la monnaie.)
     FORFAIT = (UN PARAMETRE VALEUR = (UN FORFAIT))
     MODE = (UN PARAMETRE VALEUR = (UN MODE-DE-PAIEMENT))
     RECU = (UN PARAMETRE VALEUR = (UN ARGENT-RECU))
     RESULTAT = (UN ARGENT-A-RENDRE))


AFFICHER =
   (UNE OPERATION-SYSTEME
     (* affiche une entité OBJET en un EMPLACEMENT de l'ECRAN )
     OBJET = (UN PARAMETRE VALEUR = (UNE ENTITE))
     DANS (UN PARAMETRE VALEUR = (UN EMPLACEMENT DE ECRAN)))


IMPRIMER =
   (UNE OPERATION-SYSTEME
     (*imprime une entité OBJET sur l'IMPRIMANTE )
     OBJET = (UN PARAMETRE  VALEUR = (UNE ENTITE)))


CHERCHER  =
   (UNE OPERATION-UTILISATEUR
     (* représente le fait que l'utilisateur recherche une information
     répondant à un certain critère et appartenant à une entité donnée
     présentée sur l'ECRAN)
     OBJET = (UN PARAMETRE  VALEUR = ((UNE ENTITE) CRITERE))
     APPARTENANT-A (UN PARAMETRE  VALEUR = (UNE ENTITE))
     DANS (UN PARAMETRE  VALEUR = (UN EMPLACEMENT DE ECRAN))
     RESULTAT = (UNE ENTITE (* qui satisfait le critère de recherche)))


LIRE =
   (UNE OPERATION-UTILISATEUR
     (* qui représente le fait que l'utilisateur lit en un emplacement de
     l'écran une entité donnée, mais, du point de vue système, on ne connaît
     pas, ou ne se soucie pas, du résultat de la lecture)
     OBJET =  (UN PARAMETRE  VALEUR = (UNE ENTITE))
     DANS  (UN PARAMETRE VALEUR = (UN EMPLACEMENT DE ECRAN)))


RECEVOIR-ARGENT =
   (UNE OPERATION-UTILISATEUR
     (* consiste à recevoir de l'argent du client)
     RESULTAT = (UN ARGENT-RECU))
```

Figure A.6 : NIVEAU SEMANTIQUE, les opérations conceptuelles d'Appli-Forfait.

Les opérations conceptuelles, système et utilisateur, servent à l'expression des procédures sémantiques.

4.3 Procédures et méthodes sémantiques

Tout comme une procédure de tâche, une procédure sémantique décrit un moyen pour accomp une tâche ; et tout comme une méthode de tâche, une méthode sémantique associe une procédure à u tâche (voir la figure A.4). La description des méthodes sémantiques d'Appli-Forfait peut ê consultée dans la figure A.7. Seules sont indiquées les procédures associées à des tâch élémentaires. Une procédure de tâche composée s'obtient par composition et combinaison d procédures des tâches élémentaires.

```
SEM-METHOD1 =
   (UNE METHODE-SEMANTIQUE
     POUR   T-CHOISIR-FORFAIT
     FAIRE
       (* Le composant FAIRE décrit la procédure sémantique
       exécuter pour réaliser la tâche. Cette description utilise l
       opérations conceptuelles du niveau sémantique. La procédure consis
       à afficher le répertoire des catégories, chercher dans ce répertoi
       une catégorie de forfait qui répond aux critères du client, affich
       le forfait correspondant à la catégorie et éventuellement lire
       contenu du forfait.)
       CHOIX-DU-CLIENT = (UN PARAMETRE VALEUR = (UNE CATEGORIE-DE-FORFAIT))
       RESULTAT = (FORFAIT = (UN FORFAIT))
       (EN-SEQUENCE:
          (AFFICHER (LE REPERTOIRE-DES-CATEGORIES))
          (CHERCHER
               ((UNE CATEGORIE-DE-FORFAIT)(CORRESPONDANT-A CHOIX-DU-CLIENT)
                APPARTENANT-A  REPERTOIRE-DES-CATEGORIES)
          (AFFICHER (FORFAIT CATEGORIE = RESULTAT DE CHERCHER)
          (OPTIONNELLEMENT (LIRE (FORFAIT)))
          (* OPTIONNELLEMENT représente le non déterminisme
          comportement de l'utilisateur.)))

SEM-METHOD2 =
   (UNE METHODE-SEMANTIQUE
     POUR   T-EDITER-FORFAIT
     (* consiste à modifier les champs du forfait qui nécessitent
     l'être. La demande du client est un paramètre d'entrée de la procédur
     Elle est représentée par une liste de champs. L'utilisateur va édit
     les champs du forfait à partir des valeurs de champs de la demande
     client.)

     FAIRE
       CHOIX-DU-CLIENT = (UN PARAMETRE VALEUR = (UNE LISTE DE (UN CHAMP)))
       FORFAIT = (UN PARAMETRE VALEUR = (UN FORFAIT))
                 (* le forfait à éditer.)
       (POUR choix = (PREMIER-DE CHOIX-DU-CLIENT)
              JUSQUA (DERNIER-DE CHOIX-DU-CLIENT) FAIRE
          (EN SEQUENCE:
            (CHERCHER
              ((UN CHAMP)
                 (A-EDITER==OUI NOM==(NOM DE choix) VAL!=(VAL DE choix)))
                 APPARTENENT-A FORFAIT)
            (SI ((LE RESULTAT DE CHERCHER) != ECHEC) ALORS
                 (EDITER (LE RESULTAT DE CHERCHER) choix))
            (OPTIONNELLEMENT (LIRE (LE RESULTAT DE EDITER)))
          )))
```

```
SEM-METHOD3 =
   (UNE METHODE-SEMANTIQUE
     POUR   T-ENCAISSER-FORFAIT
     (*une fois le paiement reçu du client, lui donner effectivement son
     forfait. "Donner le forfait" s'exprime au niveau informatique par
     "imprimer le forfait".)
     FAIRE
       FORFAIT = (UN PARAMETRE VALEUR = (UN FORFAIT))
       CHOIX-MODE-DU-CLIENT = (UN PARAMETRE VALEUR = (UN MODE-DE-PAIEMENT))
       RESULTAT = (UN ARGENT-A-RENDRE)
       (EN-SEQUENCE:
          (AFFICHER (LE REPERTOIRE-DES-MODES-DE-PAIEMENT))
          (MODE=(CHERCHER
                  ((UN MODE-DE-PAIEMENT)(VAL==(VAL DE CHOIX-MODE-DU-CLIENT)))
                  APPARTENANT-A REPERTOIRE-DES-MODES-DE-PAIEMENT))
          (ENCAISSER (FORFAIT MODE (LE RESULTAT DE RECEVOIR-ARGENT)))
          (AFFICHER (LE RESULTAT DE ENCAISSER))
          (OPTIONNELLEMENT (LIRE (LE RESULTAT DE ENCAISSER)))
          (IMPRIMER FORFAIT)
          (RANGER FORFAIT DANS (LE REPERTOIRE-DES-VENTES))
       ))


SEM-METHOD4 =
   (UNE METHODE-SEMANTIQUE
     POUR   T-VERIFIER-CAISSE
     FAIRE
       (EN-SEQUENCE:
          (TOTALISER (LE REPERTOIRE-DES-VENTES))
          (AFFICHER (LE RESULTAT DE TOTALISER))
          (OPTIONNELLEMENT(IMPRIMER (LE RESULTAT DE TOTALISER)))
       ))
```

Figure A.7 : NIVEAU SEMANTIQUE : méthodes et procédures sémantiques d'Appli-Forfait.

4.4 Résumé sur le niveau sémantique

L'observation de l'exemple Appli-Forfait suggère quatre remarques :

1. les méthodes sémantiques et les entités conceptuelles concrétisent le lien entre le fonctionnement du système et le domaine d'application. Toutes deux représentent explicitement des éléments du niveau tâche.

2. une procédure sémantique Ps correspond à une procédure de tâche Pt. Ps et Pt décrivent le même procédé de réalisation d'une tâche donnée mais en des termes différents : chacune utilise les éléments de son niveau d'abstraction.

3. les procédures sémantiques décrivent non seulement la participation du système que l'utilisateur perçoit, mais aussi la participation de l'utilisateur lui-même : le passage du niveau tâche (orienté utilisateur) au niveau sémantique (orienté système), conserve l'activité de l'utilisateur.

4. comme au niveau tâche, la description sémantique fait appel au langage naturel. Toutefois, l éléments sémantiques vont servir de support à l'expression de la sémantique des éléments d niveaux inférieurs.

5. en faisant le lien avec les notions introduites au chapitre 3, la composante conceptuelle de CI (c.-à-d. les niveaux tâche et sémantique) formalise la connaissance sémantique de l'utilisate Cette connaissance, rappelons-le, concerne le domaine (tel que l'émission de forfaits) et système informatique (tel qu'Appli-Forfait).

Ici s'achèvent les descriptions de type conceptuel d'un système. Nous abordons maintena l'aspect communication qui, comme son nom l'indique, précise la présentation du systèn c'est-à-dire la façon dont l'utilisateur doit procéder pour obtenir un service et la façon dont le systè réagit aux demandes. La composante communication formalise la connaissance syntaxique évoqu au chapitre 3.

5. Le niveau syntaxique

Le niveau syntaxique définit la structure du langage de commande. Une description syntaxiq CLG utilise les éléments de représentation suivants : commandes, arguments, contextes, variabl d'état, canaux de sortie, procédures et méthodes syntaxiques.

5.1 Les commandes

Les commandes représentent les requêtes de l'utilisateur auprès du système. Leur définiti comprend la spécification des arguments et la description de leur effet. Le rôle des arguments e précisé plus loin. L'effet d'une commande est défini par une procédure d'interprétation. Cet procédure, exprimée avec les opérations conceptuelles du niveau sémantique, explicite correspondance entre la notion de commande du niveau syntaxique et la notion d'opérati conceptuelle du niveau sémantique. Cette correspondance définit l'action atomique du système point de vue de l'utilisateur.

5.2 Les arguments

Les arguments représentent les entités conceptuelles référencées dans une commande. U argument est défini par une forme, sorte de descripteur qui précise :

- un type de valeur, c'est-à-dire la classe de l'entité système que l'argument représente,
- une valeur par défaut pour le cas où l'utilisateur ne spécifie pas de valeur pour l'argument,
- la notation employée par l'utilisateur pour spécifier la valeur de l'argument, et
- une procédure d'interprétation utilisée par le système pour accéder à l'entité système depuis notation.

Par exemple, dans la figure A.13, la commande CHOISIR-FORFAIT a pour argument OBJET forme ID-CATEGORIE. La définition de ID-CATEGORIE dans la figure A.8 stipule que la notatic utilisée pour désigner l'argument est une icône et que cet argument est un FORFAIT. Le char FORME de ID-CATEGORIE est ici une notation simple. CLG admet également des form

composées.

```
   ID-CATEGORIE =
      (UN DESCRIPTEUR
        FORME = (UNE ICONE-DE-CATEGORIE)
        VALEUR = (UN FORFAIT)
        VALEUR-PAR-DEFAUT = (LE FORFAIT-ACTUEL)
        PROCEDURE-D-INTERPRETATION
           (SI (FORME (PAS (DANS LES-ICONES-DE-CATEGORIE)))
              ALORS (*notifier identification de catégorie invalide)
              SINON (RETOURNER (*le dernier forfait vendu de même catégorie)))
        (*ID-CATEGORIE dénote une entité conceptuelle de type FORFAIT.
        La procédure d'interprétation est chargée de déterminer l'ENTITE à
        partir de la FORME du descripteur. La VALEUR-PAR-DEFAUT est
        choisie lorsque l'utilisateur ne spécifie pas de FORME. La FORME de
        ID-CATEGORIE est ici une ICONE, entité syntaxique CLG.))


ICONE-DE-CATEGORIE =
      (UNE ICONE
        (UNE-PARMI:  ICONE-JOURNEE-MINI ICONE-SEJOUR-MINI
                     ICONE-JOURNEE-MAXI ICONE-SEJOUR-MAXI))

   LES-ICONES-DE-CATEGORIE =
      (ENSEMBLE:     ICONE-JOURNEE-MINI ICONE-SEJOUR-MINI
                     ICONE-JOURNEE-MAXI ICONE-SEJOUR-MAXI )

   ICONE-DE-PAIEMENT =
      (UNE ICONE
        (UNE-PARMI:  ICONE-LIQUIDE  ICONE-CHEQUE))

   LES-ICONES-DE-PAIEMENT =
      (ENSEMBLE: ICONE-LIQUIDE  ICONE-CHEQUE)

   ID-CHAMP =
      (UN DESCRIPTEUR
        FORME = (UN LABEL-DE-CHAMP)
        VALEUR = (UN CHAMP)
        VALEUR-PAR-DEFAUT = (LE CHAMP-ACTUEL)
        PROCEDURE-D-INTERPRETATION
           (SI (FORME (PAS (DANS LES-LABELS-DE-CHAMP)))
              ALORS
                 (*notifier identification de champ invalide)
              SINON
                 (RETOURNER(*le champ correspondant à la FORME de ID-CHAMP)))
        (* la valeur par défaut est celle du champ correspondant dans le
        forfait actuel. Cette valeur est automatiquement héritée puisque UN
        CHAMP est UN COMPOSANT-DE FORFAIT.)

   LABEL-DE-CHAMP =
      (UN LABEL
        (UN-PARMI:LABEL-CHAMP-NOM-CLIENT LABEL-CHAMP-NOMBRE-DE-JOURS
                  LABEL-CHAMP-DATE-DEBUT LABEL-CHAMP-DATE-FIN
```

```
                LABEL-CHAMP-CODE LABEL-CHAMP-NOMBRE-D-EXEMPLAIRES)
      (* LABEL est un type prédéfini dans CLG. Les composant
      LABEL-CHAMP-DATE-FIN et LABEL-CHAMP-NOMBRE-DE-JOURS semblent fai
      double-emploi. Ils correspondent en fait à deux méthodes d
      spécification de la durée de validité d'un forfait : soit le guichetie
      indique le nombre jours et le système met automatiquement à jour l
      date-de-fin, soit il spécifie directement la date-de-fin. Le choi
      entre les deux méthodes dépendra fortement du client selon que celui-c
      exprime sa requête en disant qu'il désire "skier 3 jours" ou "skie
      jusqu'au 28 janvier"))

LES-LABELS-DE-CHAMP =
   (ENSEMBLE:  LABEL-CHAMP-NOM-CLIENT LABEL-CHAMP-NOMBRE-DE-JOURS
               LABEL-CHAMP-DATE-DEBUT LABEL-CHAMP-DATE-FIN
               LABEL-CHAMP-CODE LABEL-CHAMP-NOMBRE-D-EXEMPLAIRES)

ID-MODE =
   (UN DESCRIPTEUR
     FORME = (UNE ICONE-DE-PAIEMENT)
     VALEUR = (UN MODE-DE-PAIEMENT)
     VALEUR-PAR-DEFAUT = (LE MODE-ACTUEL)
     PROCEDURE-D-INTERPRETATION
        (SI (FORME (PAS (DANS LES-ICONES-DE-PAIEMENT)))
           ALORS (*notifier identification de mode de paiement invalide)
           SINON (RETOURNER (*le mode de paiement.))))

ID-MONTANT =
   (UN DESCRIPTEUR
     FORME = (UN LABEL)
     VALEUR = (UN ENTIER)
     VALEUR-PAR-DEFAUT = (PRIX DE FORFAIT-ACTUEL)
     (* si le guichetier ne spécifie pas de forme, alors la valeur d
     montant sera celle du prix du forfait.)
     PROCEDURE-D-INTERPRETATION
        (SI (PAS (MEME-TYPE-QUE (VALEUR DE FORME) ENTIER))
           ALORS (*notifier valeur saisie invalide)))
```

Figure A.8 : NIVEAU-SYNTAXIQUE, les descripteurs d'arguments de Appli-Forfait.

5.3 Les contextes

Les contextes sont des éléments de structuration du dialogue. Ils n'ont aucune correspondanc avec des éléments du niveau sémantique. Nous pouvons consulter la figure A.9 pour no familiariser avec les contextes d'Appli-Forfait : le contexte normal de travail, le contex d'encaissement d'un forfait et le contexte relatif à la détermination de la recette du jour.

```
CONTEXTE-USUEL =
      ( UN CONTEXTE
        VARIABLES-D-ETAT =
          (ENSEMBLE:DATE-DU-JOUR  CODE-DU-JOUR  FORFAIT-ACTUEL  CHAMP-ACTUEL
                    DERNIER-FORFAIT-SEJOUR-MAXI DERNIER-FORFAIT-JOURNEE-MAXI
                    DERNIER-FORFAIT-SEJOUR-MINI DERNIER-FORFAIT-JOURNEE-MINI)
        ZONES-D-AFFICHAGE =
          (ENSEMBLE: ZONE-FORFAIT ZONE-CATEGORIES ZONE-TOTAL-JOURNEE)
        COMMANDES =
          (ENSEMBLE:QUITTER-APPLI-FORFAIT CHOISIR-FORFAIT EDITER-FORFAIT
                    SIGNALER-FIN-EDITION  TOTALISER-JOURNEE)
        COMMANDES-D-ENTREE-DANS-CE-CONTEXTE =
          (ENSEMBLE:LANCER-APPLI-FORFAIT)
        COMMANDES-DE-SORTIE-DE-CE-CONTEXTE =
      .   (ENSEMBLE: QUITTER-APPLI-FORFAIT  SIGNALER-FIN-EDITION
                     TOTALISER-JOURNEE )
        DESCRIPTEURS =
          (ENSEMBLE: ID-CATEGORIE ID-CHAMP))

   CONTEXTE-DE-PAIEMENT =
       ( UN CONTEXTE
        VARIABLES-D-ETAT =
          (ENSEMBLE:MODE-ACTUEL MONTANT-RECU-ACTUEL)
        ZONES-D-AFFICHAGE =
          (ENSEMBLE: ZONE-DE-PAIEMENT)
        COMMANDES =
          (ENSEMBLE:ENCAISSER-FORFAIT IMPRIMER-FORFAIT
                    CHOISIR-MODE-DE-PAIEMENT EDITER-MONTANT-RECU)
        COMMANDES-D-ENTREE-DANS-CE-CONTEXTE =
          (ENSEMBLE:SIGNALER-FIN-EDITION)
        COMMANDES-DE-SORTIE-DE-CE-CONTEXTE =
          (ENSEMBLE:IMPRIMER-FORFAIT (UNE COMMANDE-USUELLE)
          (* on sort de ce contexte en imprimant le forfait ou en émettant une
          commande du contexte usuel.))
        DESCRIPTEURS =
          (ENSEMBLE: ID-MODE ID-MONTANT))

   CONTEXTE-RECETTE =
       ( UN CONTEXTE
        VARIABLES-D-ETAT =
          (ENSEMBLE:RECETTE-ACTUELLE)
        ZONES-D-AFFICHAGE =
          (ENSEMBLE: ZONE-DE-RECETTE)
        COMMANDES =
          (ENSEMBLE: IMPRIMER-RECETTE)
        COMMANDES-D-ENTREE-DANS-CE-CONTEXTE =
          (ENSEMBLE:TOTALISER-JOURNEE)
        COMMANDES-DE-SORTIE-DE-CE-CONTEXTE =
          (ENSEMBLE:IMPRIMER-RECETTE (UNE COMMANDE-USUELLE))
          (* on sort de ce contexte en imprimant la recette ou en émettant une
          commande du contexte usuel.)))
```

Figure A.9 : NIVEAU SYNTAXIQUE, les contextes d'Appli-Forfait.

5.4 Les variables d'état

Une variable est liée à une classe de valeurs du niveau sémantique mais son utilité est limitée niveau syntaxique. Une opération standard CLG, LIER, permet de l'associer à une ent conceptuelle. Elle peut être référencée dans les procédures syntaxiques et sa portée est définie par contexte auquel elle appartient. Dans le contexte usuel d'Appli-Forfait, FORFAIT-ACTUEL DATE-DU-JOUR sont des variables d'état qui désignent respectivement le forfait actuel et la date jour. Elles servent de valeur par défaut aux arguments de type forfait et date dans le contexte usi (voir le descripteur ID-CATEGORIE de la figure A.8).

```
DATE-DU-JOUR =
    (UNE VARIABLE-D-ETAT
      CONTEXTE = CONTEXTE-USUEL
      VALEUR = (UNE DATE))


CODE-DU-JOUR =
    (UNE VARIABLE-D-ETAT
      CONTEXTE = CONTEXTE-USUEL
      VALEUR = (UNE CHAINE-ALPHANUMERIQUE))

FORFAIT-ACTUEL =
    (UNE VARIABLE-D-ETAT
      CONTEXTE = CONTEXTE-USUEL
      VALEUR = (UN FORFAIT))

DERNIER-FORFAIT-SEJOUR-MAXI =
    (UNE VARIABLE-D-ETAT
      CONTEXTE = CONTEXTE-USUEL
      VALEUR = (UN FORFAIT))

DERNIER-FORFAIT-JOURNEE-MAXI = (* comme pour DERNIER-FORFAIT-SEJOUR-MAXI)
DERNIER-FORFAIT-SEJOUR-MINI = (* comme pour DERNIER-FORFAIT-SEJOUR-MAXI)
DERNIER-FORFAIT-JOURNEE-MINI = (* comme pour DERNIER-FORFAIT-SEJOUR-MAXI)

CHAMP-ACTUEL =
    (UNE VARIABLE-D-ETAT
      CONTEXTE = CONTEXTE-USUEL
      VALEUR = (UN CHAMP))

MODE-ACTUEL =
    (UNE VARIABLE-D-ETAT
      CONTEXTE = CONTEXTE-DE-PAIEMENT
      VALEUR = (UN MODE-DE-PAIEMENT))

MONTANT-RECU-ACTUEL =
    (UNE VARIABLE-D-ETAT
      CONTEXTE = CONTEXTE-DE-PAIEMENT
      VALEUR = (UN ARGENT-RECU))
```

```
RECETTE-ACTUELLE =
   (UNE VARIABLE-D-ETAT
     CONTEXTE = CONTEXTE-RECETTE
     VALEUR = (UNE RECETTE))
```

Figure A.10: NIVEAU SYNTAXIQUE, les variables d'état des contextes d'Appli-Forfait.

5.5 Les canaux de sortie

Les canaux de sortie représentent la partie de l'information que le système destine à l'utilisateur dans le cadre d'un contexte. Ils donnent une idée grossière de la nature du matériel qui va servir de support à l'information. Dans le cas du contexte usuel d'Appli-forfait, les canaux de sortie utilisés sont des zones d'écran assimilées à des fenêtres (voir la figure A.11).

```
ZONE-D-AFFICHAGE-USUELLE =
   (UNE ZONE-D-AFFICHAGE
     SUR  ECRAN
     CONTEXTE  CONTEXTE-USUEL)

ZONE-FORFAIT =
   (UNE ZONE-D-AFFICHAGE-USUELLE
    ' NOM = "Fenêtre Forfait")

ZONE-CATEGORIES
   (UNE ZONE-D-AFFICHAGE-USUELLE
     NOM = "Fenêtre des Catégories de Forfait")

ZONE-TOTAL-JOURNEE =
   (UNE ZONE-D-AFFICHAGE-USUELLE
     NOM = "zone pour l'affichage de la commande totaliser")

ZONE-DE-PAIEMENT
   (UNE ZONE-D-AFFICHAGE
     SUR  ECRAN
     CONTEXTE  CONTEXTE-DE-PAIEMENT
     NOM = "Fenêtre pour l'encaissement d'un forfait")

ZONE-DE-RECETTE
   (UNE ZONE-D-AFFICHAGE
     SUR  ECRAN
     CONTEXTE  CONTEXTE-RECETTE
     NOM = "Fenêtre pour l'affichage des recettes.")
```

Figure A.11: NIVEAU SYNTAXIQUE, les canaux de sortie d'Appli-Forfait sont visuels.

Maintenant que nous avons passé en revue les éléments syntaxiques utiles à définition d'une commande, nous pouvons consulter la liste des commandes des figures A.12, A.13, A.14 et A.15. Certaines commandes, comme LANCER-APPLI-FORFAIT et CHOISIR-FORFAIT, ont un attribut

EFFET-DE-BORD. Cet attribut exprime les retombées de l'exécution de la commande sur le contex comme l'association des variables d'état à des entités conceptuelles (voir la date du jour et le forf actuel).

```
LANCER-APPLI-FORFAIT =
   (UNE COMMANDE
     CONTEXTE = CONTEXTE-SYSTEME-D-EXPLOITATION
     NOM = "lancer Appli-Forfait"
     EFFET
        (* l'attribut EFFET exprime la sémantique de la commande
        l'aide des opérations conceptuelles du système.)
        (EN SEQUENCE:
           (ENTRER DANS CONTEXTE-USUEL)
           (EDITER
             (CODE DE DERNIER-FORFAIT-JOURNEE-MAXI) code)
           (EDITER
             (CODE DE DERNIER-FORFAIT-JOURNEE-MINI) code)
           (EDITER
             (DATE-DEBUT DE DERNIER-FORFAIT-SEJOUR-MAXI) date)
           (EDITER
             (DATE-DEBUT DE DERNIER-FORFAIT-SEJOUR-MINI) date)
           (AFFICHER (DERNIER-FORFAIT-JOURNEE-MAXI) DANS ZONE-FORFAIT))
           (AFFICHER (LES-ICONES-DE-CATEGORIE) DANS ZONE-CATEGORIES)
           (AFFICHER (ICONE-TOTAL-JOURNEE) DANS ZONE-TOTAL-JOURNEE))
     EFFET-DE-BORD =
           ((LIER DATE-DU-JOUR date)
           (LIER CODE-DU-JOUR code)
           (LIER FORFAIT-ACTUEL DERNIER-FORFAIT-JOURNEE-MAXI)
           (LIER CHAMP-ACTUEL (NOM-CLIENT DE FORFAIT-ACTUEL))
     (* L'entrée dans le contexte usuel s'accompagne des initialisatior
     suivantes : mise à jour du code du jour et de la date dans le
     exemplaires standard, afficher les icônes identifiant les catégories c
     forfait, et un exemplaire de forfait le plus souvent vendu
     journalier, maxiréseau-. ENTRER est une commande standard de CLG.))
```

Figure A.12 : NIVEAU SYNTAXIQUE, la commande du contexte système qui permet de lancer Appli-Forfait.

```
COMMAMDE-USUELLE =
   (UNE COMMANDE
     CONTEXTE = CONTEXTE-USUEL)

QUITTER-APPLI-FORFAIT  =
   (UNE COMMANDE-USUELLE
     NOM = "Quitter"
     EFFET
        (SI (PAS (DANS CONTEXTE-RECETTE))
             ALORS (ENTRER DANS CONTEXTE-RECETTE))
        (TOTALISER (LE REPERTOIRE-DES-VENTES))
        (AFFICHER (LE RESULTAT DE TOTALISER) DANS ZONE-RECETTE)
        (ENTRER DANS CONTEXTE-SYSTEME-D-EXPLOITATION))
```

```
CHOISIR-FORFAIT
   (UNE COMMAMDE-USUELLE
     NOM = "Choisir un forfait"
     OBJET = (UN ARGUMENT FORME = (UNE ID-CATEGORIE))
     EFFET
        (EN SEQUENCE:
        (SI (PAS (DANS CONTEXTE-USUEL)) ALORS (ENTRER DANS CONTEXTE-USUEL))
        (AFFICHER OBJET DANS ZONE-FORFAIT))
     EFFET-DE-BORD =
        ((LIER  FORFAIT-ACTUEL  OBJET)
        (* Le FORFAIT-ACTUEL devient le dernier forfait vendu dans la
        catégorie choisie. Voir la définition de ID-CATEGORIE)))

EDITER-FORFAIT
   (UNE COMMAMDE-USUELLE
     NOM = "remplir un champ du forfait"
     OBJET1 = (UN ARGUMENT FORME = (UN ID-CHAMP))
     OBJET2 = (UN ARGUMENT VALEUR = (UN MEME-TYPE-QUE (VAL DE OBJET1)))
     EFFET
        (EN SEQUENCE:
        (SI (PAS (DANS CONTEXTE-USUEL)) ALORS (ENTRER DANS CONTEXTE-USUEL))
        (EDITER OBJET1 OBJET2)
        (SI ((EST-UN (FORME DE OBJET1) LABEL-CHAMP-NOMBRE-DE-JOURS)
             OU
             (EST-UN (FORME DE OBJET1) LABEL-CHAMP-DATE-DEBUT)
             OU
             (EST-UN (FORME DE OBJET1) LABEL-CHAMP-DATE-FIN)
             OU
             (EST-UN (FORME DE OBJET1) LABEL-NOMBRE-D-EXEMPLAIRES))
        ALORS (AFFICHER (PRIX DE FORFAIT-ACTUEL) DANS ZONE-FORFAIT))
        (* Le prix du forfait n'est pas le seul à exiger un affichage de
        feed-back : tout changement sur le nombre de jours doit aussi
        s'accompagner de la mise à jour de la date de début ou de fin. De la
        même façon tout changement de date doit se refléter sur le nombre de
        jours.)
     EFFET-DE-BORD =
        (LIER CHAMP-ACTUEL (SUIVANT-DE CHAMP-ACTUEL))
        (* Le SUIVANT-DE du dernier champ rend le premier champ))

SIGNALER-FIN-EDITION =
   (UNE COMMAMDE-USUELLE
     NOM = "fin d'édition"
     EFFET
        (EN SEQUENCE:
           (ENTRER DANS CONTEXTE-DE-PAIEMENT)
           (AFFICHER LES-ICONES-DE-PAIEMENT DANS ZONE-DE-PAIEMENT)
           (AFFICHER MONTANT-RECU-ACTUEL DANS ZONE-DE-PAIEMENT))
     EFFET-DE-BORD =
        (LIER MONTANT-RECU-ACTUEL (PRIX DE FORFAIT-ACTUEL)))
```

```
TOTALISER-JOURNEE =
    (UNE COMMAMDE-USUELLE
      NOM = "Vérifier la Caisse"
      EFFET
         (EN SEQUENCE:
            (ENTRER DANS CONTEXTE-RECETTE)
            (TOTALISER (LE REPERTOIRE-DES-VENTES))
            (AFFICHER (LE RESULTAT DE TOTALISER) DANS ZONE-RECETTE))
      EFFET-DE-BORD =
         (LIER RECETTE-ACTUELLE (LE RESULTAT DE TOTALISER)))
```

Figure A.13 : NIVEAU SYNTAXIQUE, les commandes du contexte usuel d'Appli-Forfait.

```
COMMAMDE-DE-PAIEMENT =
    ( UNE COMMANDE
      CONTEXTE = CONTEXTE-DE-PAIEMENT)

ENCAISSER-FORFAIT =
    (UNE COMMAMDE-DE-PAIEMENT
      EFFET
         (EN SEQUENCE:
            (ENCAISSER FORFAIT-ACTUEL MODE-ACTUEL MONTANT-RECU-ACTUEL)
            (AFFICHER (LE RESULTAT DE ENCAISSER) DANS ZONE-DE-PAIEMENT)))

IMPRIMER-FORFAIT
    (UNE COMMAMDE-DE-PAIEMENT
      NOM = "Imprimer un forfait"
      EFFET
         (EN SEQUENCE:
         (IMPRIMER FORFAIT-ACTUEL)
         (RANGER FORFAIT-ACTUEL DANS (LE REPERTOIRE-DES-VENTES))
         (SI (* catégorie du forfait actuel est "SEJOUR-MAXI")
            ALORS ((LIER DERNIER-FORFAIT-SEJOUR-MAXI FORFAIT-ACTUEL)))
         (* idem pour les autres catégories de façon à mémoriser le dernie
         forfait vendu dans sa catégorie.)
         (ENTRER DANS CONTEXTE-USUEL)))

CHOISIR-MODE-DE-PAIEMENT
    (UNE COMMAMDE-DE-PAIEMENT
      NOM = "choisir un mode de paiement"
      MODE = (UN ARGUMENT FORME = (UN ID-MODE))
      EFFET-DE-BORD=
         (LIER MODE-ACTUEL MODE))

EDITER-MONTANT-RECU
    (UNE COMMAMDE-DE-PAIEMENT
      NOM = "changer le montant reçu"
      RECU = (UN ARGUMENT FORME = (UN ID-MONTANT))
      EFFET
         ((EDITER MONTANT-RECU-ACTUEL RECU)))
```

Figure A.14 : NIVEAU SYNTAXIQUE, les commandes du contexte de paiement d'Appli-Forfait.

```
COMMAMDE-DE-RECETTE =
    ( UNE COMMANDE
      CONTEXTE = CONTEXTE-RECETTE)

IMPRIMER-RECETTE
    (UNE COMMAMDE-DE-RECETTE
      NOM = "Imprimer la recette du jour"
      EFFET
         ((IMPRIMER RECETTE-ACTUELLE)
         (ENTRER DANS CONTEXTE-USUEL)))
```

Figure A.15 : NIVEAU SYNTAXIQUE, les commandes du contexte recette d'Appli-Forfait.

5.6 Méthodes et Procédures syntaxiques

Une méthode syntaxique associe une procédure syntaxique à une tâche. Une procédure syntaxique est constituée d'appels de commandes et d'opérations conceptuelles utilisateur comme le montre la figure A.16.

```
SYNT-METHOD1 =
    (UNE METHODE-SYNTAXIQUE
      POUR    T-CHOISIR-FORFAIT
      FAIRE
         CHOIX-DU-CLIENT = (UN PARAMETRE VALEUR=(UNE CATEGORIE-DE-FORFAIT))
         (EN-SEQUENCE:
            (OPTIONNELLEMENT (LIRE (CATEGORIE DE FORFAIT-ACTUEL)
                                   DANS ZONE-FORFAIT))
            (SI (CHOIX-DU-CLIENT != (CATEGORIE DE FORFAIT-ACTUEL)) ALORS
              (EN SEQUENCE:
                 (CHERCHER
                    (UNE ID-CATEGORIE) (CORRESPONDANT-A CHOIX_DU_CLIENT))
                    APPARTENANT-A LES-ICONES-DE-CATEGORIE
                    DANS ZONE-CATEGORIE)
                 (CHOISIR-FORFAIT (LE RESULTAT DE CHERCHER))))
            (OPTIONNELLEMENT (LIRE (LE FORFAIT-ACTUEL) DANS ZONE-FORFAIT)))
      (* Remarques : la méthode ne contient pas l'affichage des
      catégories de forfait effectué maintenant par la commande
      LANCER-APPLI-FORFAIT. La fonction CHERCHER est ici plus précise que son
      équivalent dans SEM-METHOD1 en ce qui concerne l'objet recherché par
      l'utilisateur. Il se peut que le guichetier ne se souvienne pas de la
      catégorie actuelle et fasse une recherche sur l'écran. Ce fait est
      modélisé ici par (OPTIONNELLEMENT (LIRE (CATEGORIE DE FORFAIT-ACTUEL)
      DANS ZONE-FORFAIT))))

SYNT-METHOD2 =
    (UNE METHODE-SYNTAXIQUE
      POUR    T-EDITER-FORFAIT
      FAIRE
         CHOIX-DU-CLIENT = (UN PARAMETRE VALEUR = (UNE LISTE DE (UN CHAMP)))
         (EN SEQUENCE:
```

```
            (POUR (choix = (PREMIER-DE CHOIX-DU-CLIENT))
                JUSQUA (DERNIER-DE CHOIX-DU-CLIENT) FAIRE
              (EN SEQUENCE:
                 (CHERCHER
                      ((UN CHAMP)
                      (A-EDITER=OUI NOM=(NOM DE choix) VAL!= (VAL DE choix)))
                      APPARTENENT-A FORFAIT DANS ZONE-FORFAIT)
                 (SI (LE RESULTAT DE CHERCHER != ECHEC) ALORS
                      (EDITER-FORFAIT (LE RESULTAT DE CHERCHER) choix))
                 (OPTIONNELLEMENT (LIRE (LE RESULTAT DE EDITER-FORFAIT)))
            ))
            (SIGNALER-FIN-EDITION)))


SYNT-METHOD3 =
      (UNE METHODE-SYNTAXIQUE
        POUR   T-ENCAISSER-FORFAIT
        FAIRE
          CHOIX-MODE-DU-CLIENT = (UN PARAMETRE VALEUR = (UN MODE))
          RESULTAT = (UN ARGENT-A-RENDRE)
          (EN-SEQUENCE:
          (* éventuellement choisir un nouveau mode de paiement si le mo
          actuel n'est pas celui du client et éditer le montant d'argent reçu
          ce choix et cette édition s'effectuent dans un ordre quelconque
          puis imprimer le forfait actuel.Le fait que l'utilisateur ait
          oublier le mode actuel n'est pas modélisé ici (Rajouter un LIRE).)
             (SI ((VAL DE MODE-ACTUEL) != (VAL DE MODE-CLIENT)) ALORS
               (EN SEQUENCE
                    (CHERCHER (UN ID-MODE (*qui correspond au choix du client
                      APPARTENANT-A (LES-ICONES-DE-PAIEMENT)
                      DANS ZONE-DE-PAIEMENT)
                    (CHOISIR-MODE-DE-PAIEMENT (LE RESULTAT DE CHERCHER))))
             (CHERCHER (UN ID-PAIEMENT) DANS ZONE-DE-PAIEMENT)
             (SI ((LE RESULTAT DE CHERCHER)!= (LE RESULTAT DE RECEVOIR-ARGENT)
              ALORS
                 (EDITER-MONTANT-RECU (LE RESULTAT DE RECEVOIR-ARGENT)))
             (ENCAISSER-FORFAIT)
             (IMPRIMER-FORFAIT)))

   SYNT-METHOD4 =
      (UNE METHODE-SYNTAXIQUE
        POUR   T-VERIFIER-CAISSE
        FAIRE
          (EN SEQUENCE:
             (TOTALISER-JOURNEE)
             (LIRE (LE RESULTAT DE TOTALISER))
             (OPTIONNELLEMENT
               (IMPRIMER (LE RESULTAT DE TOTALISER)))))
```

Figure A.16 : NIVEAU-SYNTAXIQUE : Les méthodes syntaxiques d'Appli-Forfait.

5.7 Récapitulatif sur le niveau syntaxique

Pour résumer, faisons le lien entre les éléments conceptuels des niveaux précédents et les éléments syntaxiques :

- l'association commandes-contexte identifie les opérations conceptuelles système permises à un instant donné;
- l'association commande-arguments indique les entités conceptuelles à désigner pour spécifier une commande ainsi que la forme générale des désignations telles que l'utilisateur les perçoit;
- l'association canaux-contexte précise l'information disponible à l'utilisateur dans ce contexte ainsi que le support de présentation;
- l'association contextes-variables-d'état sert à mémoriser les informations utiles aux diverses étapes du dialogue. Cette mémorisation peut être vue comme l'extension électronique de la mémoire à court terme de l'utilisateur : si le système maintient des informations comme la date du jour, l'utilisateur est évidemment libéré de cette charge;
- une procédure syntaxique Psynt correspond à une procédure sémantique Ps qui correspond à une procédure de tâche Pt. Psy, Ps et Pt décrivent le même procédé de réalisation d'une tâche donnée mais en des termes différents : chacune utilise les éléments de son niveau d'abstraction.
- comme les procédures sémantiques, les procédures syntaxiques conservent les activités conceptuelles de l'utilisateur.

Avec les notions de contexte et de commande, le niveau syntaxique définit la structure générale de l'interaction entre l'utilisateur et le système, mais il ne spécifie pas les actions physiques élémentaires nécessaires à la spécification des commandes. Cette spécification apparaît dans le dernier niveau de la composante communication : le niveau interaction.

6. Le niveau Interaction

Le niveau interaction décrit en termes d'actions physiques comment l'utilisateur et le système interagissent pour aboutir à la spécification des éléments syntaxiques ; il indique quand le système doit exécuter les procédures d'interprétation syntaxiques et vérifie que l'utilisateur respecte l'ordre d'occurence des éléments syntaxiques.

6.1 L'ordre de spécification

L'ordre de spécification des éléments syntaxiques se déduit de la description syntaxique : une commande d'entrée dans un contexte conduit à ce contexte ; un contexte conduit aux commandes permises ; chaque commande conduit à ses arguments et chaque argument conduit à ses descripteurs (un argument peut être spécifié selon plusieurs formes) ; enfin, un descripteur conduit à ses sous-descripteurs. La figure A.17 montre la hiérarchie syntaxique d'Appli-Forfait.

La suite d'actions physiques mises en jeu pour spécifier un élément syntaxique représente l'interaction entre l'utilisateur et le système pour spécifier cet élément. Cette suite est décrite par une arborescence d'éléments d'interaction.

6.2 Les éléments d'interaction

La figure A.18 est une représentation générale de l'arborescence associée à une commande. l racine de l'arborescence, appelée Spécification et notée S dans la figure, a pour successeurs possibl des éléments d'interaction terminaux et des éléments d'interaction non terminaux.

```
LANCER-APPLI-FORFAIT
    . CONTEXTE-USUEL
    .   . CHOISIR-FORFAIT
    .   .     . ARGUMENT DE CHOISIR-FORFAIT
    .   .     .     . ID-CATEGORIE
    .   .     .     .     . ICONE
    .   . EDITER-FORFAIT
    .   .     . ARGUMENT DE EDITER-FORFAIT
    .   .     .     . ID-CHAMP
    .   .     .     .     . LABEL
    .   .     .     . VALEUR-DU-CHAMP
    .   . SIGNALER-FIN-EDITION
    .   .     . CONTEXTE-DE-PAIEMENT
    .   .     .     . ENCAISSER-FORFAIT
    .   .     .     . CHOISIR-MODE-DE-PAIEMENT
    .   .     .     .     . ARGUMENT DE CHOISIR-MODE-DE-PAIEMENT
    .   .     .     .     .     . ID-MODE
    .   .     .     .     .     .     . ICONE
    .   .     .     . EDITER-MONTANT-RECU
    .   .     .     .     . ARGUMENT DE EDITER-MONTANT-RECU
    .   .     .     .     .     . ID-MONTANT
    .   .     .     .     .     .     . ENTIER
    .   .     .     . IMPRIMER-FORFAIT
    .   .     .     . (LES COMMANDES DE CONTEXTE-USUEL)
    .   . TOTALISER-JOURNEE
    .   .     . CONTEXTE-RECETTE
    .   .     .     . IMPRIMER-RECETTE
    .   .     .     . (LES COMMANDES DE CONTEXTE-USUEL)
    .   . QUITTER-APPLI-FORFAIT
```

Figure A.17 : NIVEAU INTERACTION, la hiérarchie syntaxique d'Appli-Forfait.

6.2.1 Les éléments d'interaction terminaux.

Un élément d'interaction terminal est soit une action produite par le système, soit une action physique de l'utilisateur, soit une date.

- le système peut produire l'une des actions suivantes :

  - un Prompt (noté **P** dans la figure A.18) pour inviter l'utilisateur à prendre l'initiative d'une action ; par exemple, le message "entrez une commande" sert souvent de prompt de commande.

  - une Réponse ou Accusé de réception (noté **R**) pour indiquer à l'utilisateur que la spécification d'un élément a été reçue ; par exemple la mise en vidéo-inverse de l'élément spécifié sert souvent de réponse dans les systèmes graphiques.

- l'utilisateur produit des actions élémentaires (notées **A**) telles que la frappe de touches du clavier et le déplacement de la souris.

- une Date (notée **Q**) sert à définir une relation d'ordre lorsque CLG n'en propose pas implicitement. CLG n'impose pas d'ordre sur la spécification des commandes d'un contexte ni sur les arguments d'une commande.

6.2.2 Les éléments d'interaction non terminaux

Un élément d'interaction non terminal est soit un Corps de spécification, soit la terminaison d'une spécification, soit une procédure d'interprétation d'une spécification.

- un Corps (noté **C**) dépend du type d'élément syntaxique : pour une commande, il est constitué d'une Désignation de commande (qui indique de quelle commande il s'agit parmi toutes les commandes possibles dans le contexte). Pour un argument, il comprend la Désignation de l'argument (qui indique de quel argument il s'agit parmi tous les arguments de la commande) et une Forme (qui décrit l'interaction qui doit conduire à la saisie de la valeur de l'argument).

- une Terminaison (notée **T**) décrit l'interaction qui signale la fin de spécification d'un élément syntaxique (typiquement, un retour charriot pour une fin de saisie d'un argument de type chaîne).

- une Interprétation (notée **I**) désigne la procédure syntaxique d'interprétation associée à un élément syntaxique et précise quand cette procédure doit être exécutée. **I** permet, par exemple, d'exprimer que l'interprétation d'un argument doit avoir lieu dès sa saisie ou bien différée jusqu'à ce que la commande soit entièrement spécifiée.

Les éléments d'interaction qui viennent d'être énumérés ne sont pas nécessairement tous présents dans une spécification. Par exemple, pour une procédure à un seul argument, l'élément "désignation" de l'argument n'est pas forcément indispensable. De même, nous allons voir que la spécification S-CHOISIR-FORFAIT de la commande CHOISIR-FORFAIT ne comprend qu'un prompt et un corps.

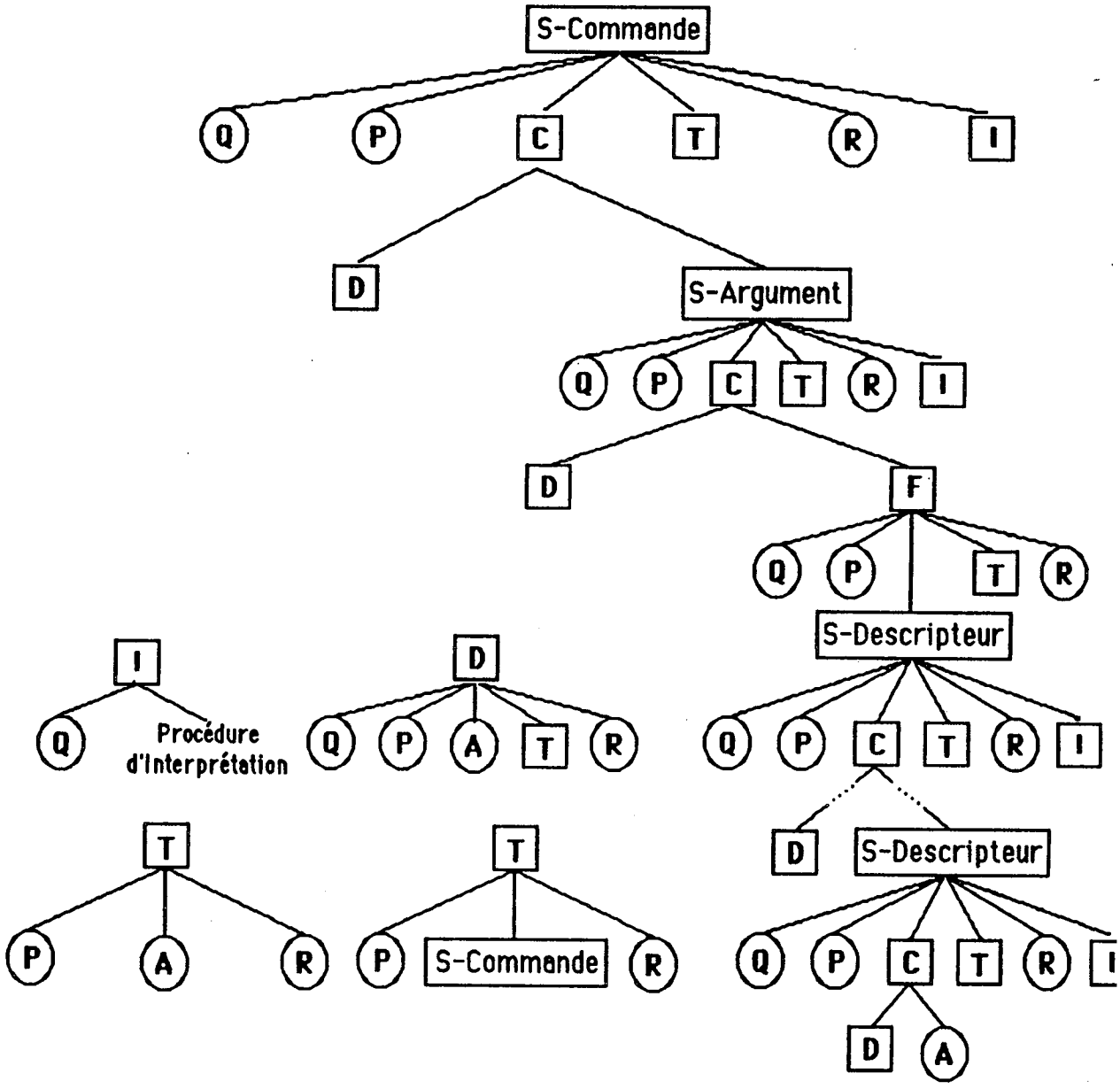


Figure A.18 : NIVEAU INTERACTION, arborescence des éléments d'interaction intervenant dans la spécification d'une commande. Les noeuds encadrés désignent des éléments non terminaux. Les noeuds encerclés correspondent à des éléments terminaux. La commande n'a, ici, qu'un argument. La spécification de cet argument est décrite par le sous-arb S-Argument. Noter qu'une forme peut etre une structure composée de descripteurs.

6.2.3 Les éléments d'interaction de l'exemple

L'arborescence de la figure A.19 indique que CHOISIR-FORFAIT est accessible dès que le mer des catégories de forfait est validé. Ce menu est une liste de boutons alignés au bas de l'écran comm le montre la figure A.20 . Le corps de la spécification de CHOISIR-FORFAIT est la spécification son unique argument : la catégorie de forfait présentée au guichetier sous forme d'un bouton-icôn Le sous-arbre S-Argument définit la spécification de l'argument, à savoir la sélection d bouton-icône. Afin de ne pas alourdir le schéma, la forme alternative de spécification qui consiste

frapper *s, n'a pas été représentée. On remarquera que S-CHOISIR-FORFAIT ne comprend pas d'élément Réponse, remplacé ici par l'élément Réponse de l'argument.

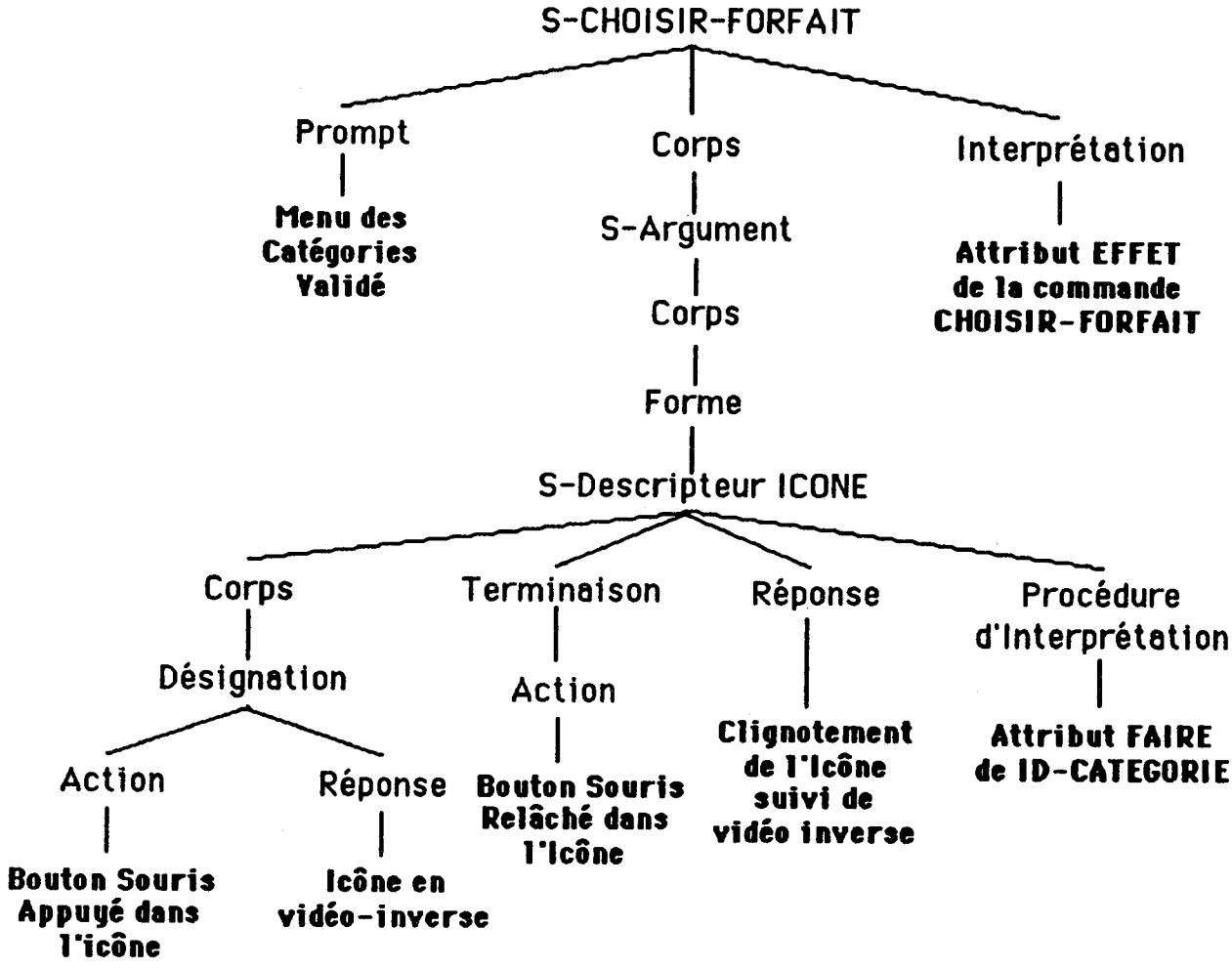


Figure A.19 : NIVEAU INTERACTION : arborescence d'éléments d'interaction décrivant la spécification de la commande CHOISIR-FORFAIT. Les éléments Date ont été omis afin de ne pas alourdir le dessin.

L'édition des champs d'un forfait constitue un second exemple intéressant (voir la figure A.21). Le Prompt de S-EDITER-FORFAIT met en évidence le champ éditable actuel. Il indique à l'utilisateur qu'il peut éditer le forfait. EDITER-FORFAIT a deux arguments : Argument1 désigne le champ à éditer et Argument2 représente la nouvelle valeur du champ. Le champ à éditer est, par défaut, le champ actuel maintenu par le système, ou bien le champ choisi par le guichetier lorsqu'il sélectionne la région occupée par le champ sur l'écran. Argument2 est un entier ou une chaîne alphabétique ou une date selon la nature d'Argument1. Les éléments Date ont été omis dans le schéma. Il est clair ici que Argument1 doit être spécifié avant Argument2.

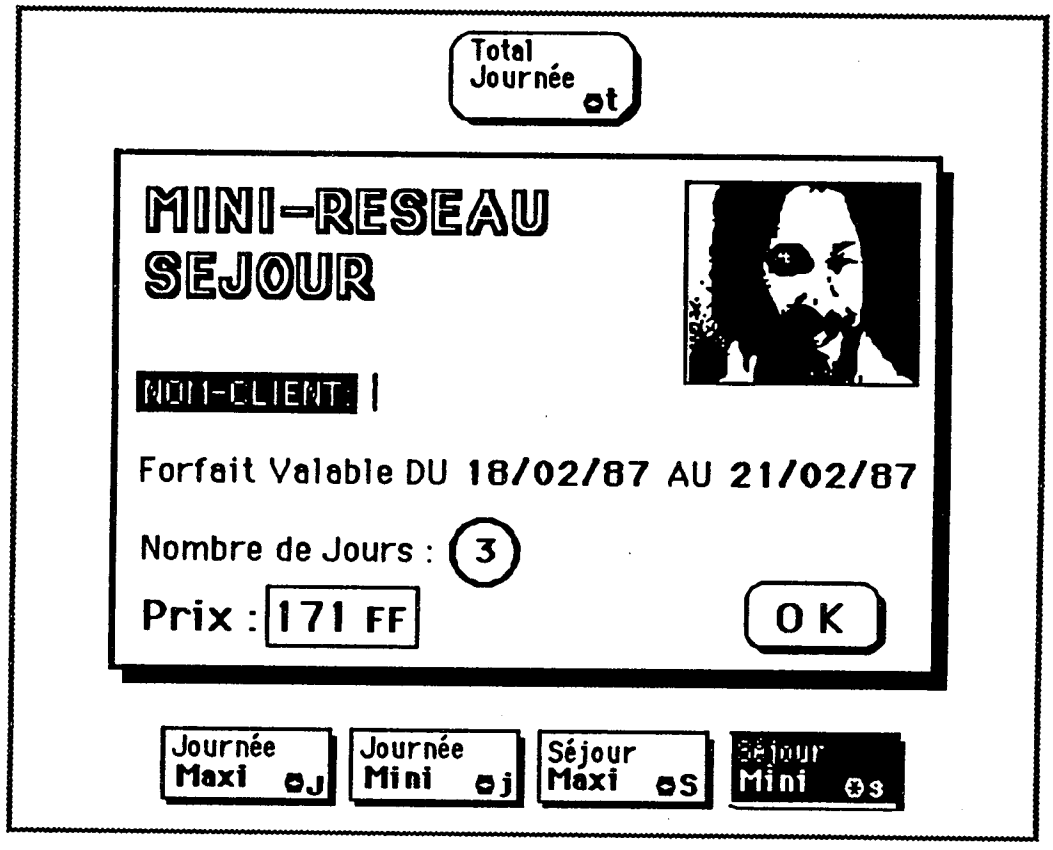


Figure A.20 : NIVEAU INTERACTION : une vue d'Appli-Forfait dès que le guichetier a appuyé sur le bouton "séjour mini" ou a tapé *s. Le forfait présenté est le dernier forfait "séjour-mini" émis dont seul le nom du client n'a pas de valeur par défaut. Les boutons de catégorie sont à tout instant validés. En conséquence, le guichetier peut éditer plusieurs forfaits simultanément dans des fenêtres qui se superposent. A un instant donné, il y a un seul forfait effectivement manipulé : celui qui se trouve au sommet de la pile de fenêtres.

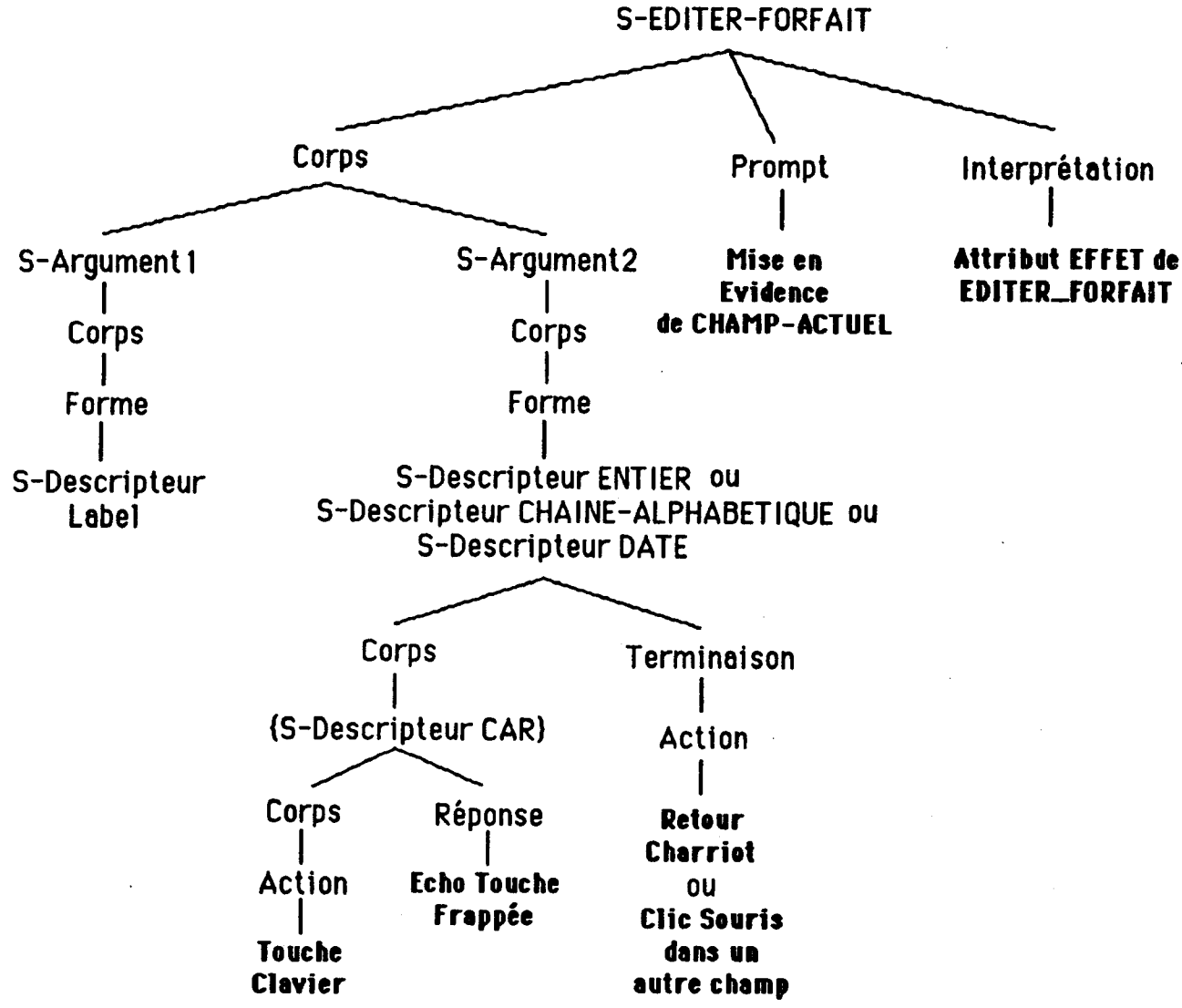


Figure A.21 : NIVEAU INTERACTION : arborescence d'éléments d'interaction décrivant la spécification de la commande EDITER-FORFAIT. Les éléments Date ont été omis afin de ne pas alourdir le dessin. Le sous-arbre "S-Descripteur Label" est similaire au "S-Descripteur Icône" de la figure A.19. Les accolades, comme dans{S-Descripteur CAR}, dénotent la répétition d'un sous-arbre. En réalité, la description ci-dessus ne prend pas en compte tous les aspects de la saisie d'une chaîne de caractères (effacement, couper-coller, etc.).

Nous venons de voir sous forme d'arborescences, le rôle des éléments d'interaction et leur association aux éléments syntaxiques. En réalité, cette association et la description des éléments d'interaction s'expriment à l'aide de règles d'interaction.

6.3 Les règles d'interaction

CLG distingue deux types de règles : les règles temporelles et les règles d'action.

- les règles temporelles servent à préciser l'ordre de spécification des éléments syntaxiques e indiquer quand le système doit interpréter la spécification d'un élément syntaxique;
- les règles d'action servent à définir
  - un Prompt ou une Réponse sous forme d'actions élémentaires système comme *Montr "entrez commande :" dans la zone Commande* ,
  - une Action sous forme d'actions physiques utilisateur comme *Frappe de la touche M* .

La figure A.22 donne quelques exemples de règles pour Appli-Forfait.

```
R-COMMANDE1 =
   (REGLE POUR
     (QUAND-SPECIFIER (UNE COMMANDE-USUELLE)
     ->
     (N-IMPORTE-QUAND DANS TOUT CONTEXTE))

R-COMMANDE2 =
   (REGLE POUR
     (ACTION-DE-DESIGNATION-DE (SIGNALER-FIN-EDITION))
     ->
     (SOURIS: placer-souris-sur (* BOUTON-OK DE ZONE-FORFAIT)
          appuyer-bouton-souris relâcher-bouton-souris))

R-COMMANDE3 =
   (REGLE POUR
     (ACTION-DE-DESIGNATION-DE (TOTALISER-JOURNEE))
     ->
     (SOURIS: placer-souris-sur (* BOUTON-TOTAL DE ZONE-RECETTE)
          appuyer-bouton-souris relâcher-bouton-souris))

R-COMMANDE4 =
   (REGLE POUR
     (ACTION-DE-DESIGNATION-DE (TOTALISER-JOURNEE))
     ->
     (CLAVIER : "⌘ t")
     (* il y a plusieurs façon de désigner la commande totaliser-journée))

R-COMMANDE5 =
   (REGLE POUR
     (REPONSE-A-LA-SPECIFICATION-DE
       (ARGUMENT DE CHOISIR-FORFAIT)
     ->
     ( MONTRER (*Clignotement-de-l-élément-de-menu
                video-inverse-de-l-élément-de-menu)))
```

```
R-COMMANDE6 =
    (REGLE POUR
      (REPONSE-A-LA-SPECIFICATION-DE (EDITER-FORFAIT))
      ->
      ( MONTRER (*CHAMP-ACTUEL-en-vidéo-inverse)))

R-COMMANDE7 =
    (REGLE POUR
      (QUAND-INTERPRETER SPECIFICATION-DE
        (UNE COMMANDE-USUELLE))
      ->
      (APRES (SPECIFICATION (COMMANDE-USUELLE))))

R-ARGUMENT1 =
    (REGLE POUR
      (QUAND-SPECIFIER
        (UN ARGUMENT DE (COMMANDE-USUELLE)))
      ->
      (EN-PREMIER-DANS (SPECIFICATION (COMMANDE-USUELLE))))

R-ARGUMENT2 =
    (REGLE POUR
      (ACTION-DE-DESIGNER
        (UN ARGUMENT DE (CHOISIR-FORFAIT)))
      ->
      ( SOURIS: placer-souris-sur (* élément de menu des catégories)
           appuyer-bouton-souris ))

R-ARGUMENT20 =
    (REGLE POUR
      (ACTION-DE-DESIGNER
        (UN ARGUMENT DE (CHOISIR-FORFAIT)))
      ->
      ( CLAVIER:   touche-contrôle +
                   touche-initiale-de-la-catégorie +
                   (si MAXI-RESEAU) touche-majuscule)

R-ARGUMENT3 =
    (REGLE POUR
      (ACTION-DE-TERMINER-SPECIFICATION-DE
        (UN ARGUMENT DE (CHOISIR-FORFAIT)))
      ->
      ( SOURIS: relâcher bouton dans un élément de menu des catégories))

R-ARGUMENT30 =
    (REGLE POUR
      (ACTION-DE-TERMINER-SPECIFICATION-DE
        (UN ARGUMENT DE (CHOISIR-FORFAIT)))
      ->
      ( CLAVIER: relâcher touches (*appuyées selon la règle R20)))
```

```
R-ARGUMENT4 =
      (REGLE POUR
        (QUAND-INTERPRETER-SPECIFICATION-DE
          (UN ARGUMENT DE (CHOISIR-FORFAIT)))
        ->
        ( APRES SPECIFICATION (ARGUMENT) DE (CHOISIR-FORFAIT)))

   ... et d'autres règles pour décrire la spécification des arguments
   EDITER-FORFAIT mises en évidence dans l'arbre.
```

Figure A.22 : NIVEAU INTERACTION , quelques règles temporelles et quelques règles d'action pour Appli-Forfa

La représentation du niveau interaction doit être complétée avec la spécification des procédures méthodes d'interaction.

6.4 Procédures et méthodes d'interaction

Une procédure d'interaction est la traduction d'une procédure syntaxique dans laquelle les app aux commandes sont remplacés par les actions physiques introduites par les règles d'interacti Enfin, une méthode d'interaction associe une procédure d'interaction à une tâche. Elle indiq concrètement comment effectuer une tâche définie initialement dans le composant conceptuel. l figure A.23 montre les méthodes d'interaction d'Appli-Forfait. On remarquera qu'il y a plusie méthodes d'interaction pour choisir un forfait.

```
INTERACTION-METHOD1.a =
   (UNE METHODE-D-INTERACTION
     POUR   T-CHOISIR-FORFAIT
     FAIRE
       CHOIX-DU-CLIENT = (UN PARAMETRE VALEUR = (UNE CATEGORIE-DE-FORFAIT))
       (EN-SEQUENCE:
          (OPTIONNELLEMENT (LIRE (CATEGORIE DE FORFAIT-ACTUEL)
                                 DANS ZONE-FORFAIT))
          (SI (CHOIX-DU-CLIENT != (CATEGORIE DE FORFAIT-ACTUEL)) ALORS
            (EN SEQUENCE:
              (CHERCHER
                 (UNE ID-CATEGORIE) (CORRESPONDANT-A CHOIX_DU_CLIENT))
                 APPARTENANT-A LES-ICONES-DE-CATEGORIE
                 DANS ZONE-CATEGORIE)
              (SOURIS: "placer-souris-sur" (LE RESULTAT DE CHERCHER)
                 "appuyer-bouton-souris" "relâcher-bouton-souris")
              (* application des règles R-Argument2 et R-argument3.)
          ))
          (OPTIONNELLEMENT (LIRE (LE FORFAIT-ACTUEL) DANS ZONE-FORFAIT))))

INTERACTION-METHOD1.b =
   (UNE METHODE-D-INTERACTION
     POUR   T-CHOISIR-FORFAIT
     FAIRE
       CHOIX-DU-CLIENT = (UN PARAMETRE VALEUR = (UNE CATEGORIE-DE-FORFAIT))
       (EN SEQUENCE:
          (OPTIONNELLEMENT (LIRE (CATEGORIE DE FORFAIT-ACTUEL)
```

```
                        DANS ZONE-FORFAIT))
        (SI (CHOIX-DU-CLIENT != (CATEGORIE DE FORFAIT-ACTUEL)) ALORS
            (CLAVIER : appuyer simultanément
                • la touche contrôle,
                • la touche correspondant à l'initiale de la catégorie du
                  forfait, c.-à-d. "j" pour journée et 's" pour séjour et,
                • la touche majuscule pour un forfait maxi-réseau)
        ))
        (OPTIONNELLEMENT (LIRE (LE FORFAIT-ACTUEL) DANS ZONE-FORFAIT)))
    (* Cette méthode applique les règles R-Argument20 et R-Argument30 de la
      figure A.22. Dans cette méthode, l'opération conceptuelle
      utilisateur de recherche de l'icône a disparu. Les doigts de
      l'utilisateur sont supposés placés sur le clavier et l'utilisateur
      est supposés être un expert. En toute rigueur, ces aspects de
      recherche devraient être exprimés dans la représentation.))


INTERACTION-METHOD2 =
   (UNE METHODE-D-INTERACTION
     POUR   T-EDITER-FORFAIT
     FAIRE
       CHOIX-DU-CLIENT = (UN PARAMETRE VALEUR = (UNE LISTE DE (UN CHAMP)))
       (EN SEQUENCE:
          (POUR (choix = (PREMIER-DE CHOIX-DU-CLIENT))
              JUSQUA (DERNIER-DE CHOIX-DU-CLIENT) FAIRE
            (EN SEQUENCE:
              (CHERCHER
                   ((UN CHAMP) (A-EDITER=OUI NOM=(NOM DE choix)
                                VAL != (VAL DE choix))))
                   APPARTENENT-A FORFAIT DANS ZONE-FORFAIT)
              (SI ((LE RESULTAT DE CHERCHER) != ECHEC) ALORS
                 (EN SEQUENCE:
                   (* Désigner le champ avec la souris et saisir le choix
                   demandé au clavier.)
                   ((SOURIS: "placer-souris-sur" (LE RESULTAT DE CHERCHER)
                   "appuyer-bouton" "relâcher-bouton"))
                   (CLAVIER: (UN VAL DE CHAMP) "Retour-Charriot")))
                   (OPTIONNELLEMENT (LIRE (LE CHAMP-ACTUEL)
                                          DANS ZONE-FORFAIT)))
          ))
         (SOURIS: "placer-souris-sur" (BOUTON-OK DE ZONE-FORFAIT)
                "appuyer-bouton-souris" "relâcher-bouton-souris"))

         (* Remarques : CLAVIER: (UN VAL DE CHAMP) représente l'interaction
         nécessaire à la saisie de la valeur d'un CHAMP. "Retour-Charriot"
         indique la condition de fin de cette saisie. Une alternative, non
         indiquée ici, est un clic souris ailleurs que sur le champ.Le
         signal de fin d'édition est ici spécifié par la sélection du
         bouton OK de la ZONE-FORFAIT.) )
```

```
INTERACTION-METHOD3 =
   (UNE METHODE-D-INTERACTION
     POUR   T-ENCAISSER-FORFAIT
     FAIRE
       CHOIX-MODE-DU-CLIENT = (UN PARAMETRE VALEUR = (UN MODE))
       RESULTAT = (UN ARGENT-A-RENDRE)
       (EN-SEQUENCE:
       (* éventuellement choisir un nouveau mode de paiement si le mo
       actuel n'est pas celui du client et éditer le montant d'argent reçu
       ce choix et cette édition s'effectue dans un ordre quelconque ; pu
       imprimer le forfait actuel.)
          (SI ((VAL DE MODE-ACTUEL) != (VAL DE MODE-CLIENT))
            ALORS
            (EN SEQUENCE:
                 (CHERCHER (UN ID-MODE (*qui correspond au choix du client
                   APPARTENANT-A (LES-ICONES-DE-PAIEMENT)
                   DANS ZONE-DE-PAIEMENT)
                 (SOURIS: "placer-souris-sur" (LE RESULTAT DE CHERCHER)
                 "appuyer-bouton-souris" "relâcher-bouton-souris")))
          (CHERCHER (UN ID-PAIEMENT) DANS ZONE-DE-PAIEMENT)
          (SI ((LE RESULTAT DE CHERCHER)!=(LE RESULTAT DE RECEVOIR-ARGENT))
           ALORS
              (CLAVIER:(UN ID-MONTANT)))
          (SOURIS: "placer-souris-sur" (BOUTON-OK DE ZONE-DE-PAIEMENT)
                 "appuyer-bouton-souris" "relâcher-bouton-souris"))
          (* CLAVIER:(UN ID-MONTANT) désigne la saisie du montant reçu et
          l'action de la souris sur le bouton OK de la zone de paieme
          lance l'impression du forfait.))


INTERACTION-METHOD4 =
   (UNE METHODE-D-INTERACTION
     POUR   T-VERIFIER-CAISSE
     FAIRE
       (EN SEQUENCE:
          (CHERCHER (BOUTON-TOTAL-JOURNEE) DANS ZONE-TOTAL-JOURNEE)
          (SOURIS: "placer-souris-sur" (LE RESULTAT DE CHERCHER)
                   "appuyer-bouton" "relâcher-bouton")
          (LIRE (LE RESULTAT DE TOTALISER) DANS ZONE-DE-RECETTE)
          (OPTIONNELLEMENT( (* Imprimer le résultat des ventes)
            (CHERCHER (ICONE-IMPRIMER) DANS ZONE-DE-RECETTE)
            (SOURIS: "placer-souris-sur" (LE RESULTAT DE CHERCHER)
                   "appuyer-bouton" "relâcher-bouton")))
          (* Remarque : comme pour les autres méthodes d'interactio
          il faudrait décrire les cas où la spécification d
          commandes se fait à partir de touches du clavier)))
```

Figure A.23 : NIVEAU INTERACTION : les méthodes d'interaction pour réaliser les tâches d'Appli-Forfait.

La description de la composante communication qui comprend, on le rappelle, les niveaux syntaxe et interaction, est maintenant complète. Reste, au bas de la hiérarchie, la composante physique qui décrit les éléments physiques en contact avec l'utilisateur. Ces éléments précisent la façon dont l'information est affichée sur l'écran ou fournie par l'intermédiaire d'autres dispositifs de sortie : c'est le niveau présentation ; ils concernent également les caractéristiques physiques des unités d'entrée/sortie : c'est le niveau unité physique. CLG ne propose aucun formalisme pour ces deux derniers niveaux. La description d'Appli-Forfait ne sera donc pas détaillée plus avant.

Annexe 2

La Partie Surchargeable du Squelette d'Application d'APEX

```
/* ***************************** module APPLI.C ***********************************

    This module is the interface between the higher semantic level of an
    interactive system (ie, the domain dependent code, the application)
    and its top level controller.
    (The top level controller is implemented by the module topcontrol.c;
    It is the interface between the various objects which implement the
    presentation and appli.c; it is responsible for acquiring low level
    events and dispatching them to the appropriate objects)

    This module is made of standard entry points called by the top level
    controller. These entry points are prefixed with "ap" such as aptask,
    apbegin, apend, etc. They are called when semantic actions need to be
    performed. In turn, they may call domain dependent procedures.
    These domain dependent procedures may be written by specialists in the
    domain such as thermodynamics as it is the case of the interactive
    system Thermo (see module thermo.c)

    EXPORTED PROCEDURES:
       apbegin       Called to initialize the application.
       apdocommand   Called when the user has specified a command
       apend         Called when the application has to shut down.
       apincontent   C.when a mouse click happened in the content region of a wind
       apresizedw    C. to tell the application that a window has been resized.
       apscrolledw   C. to tell the application that a window has been scrolled.
       apclosew      Called when the goaway region of a window has been clicked.
       aptask        C. to give the application a chance to get some CPU cycles.
       apabout       C. when the ABOUT item in the apple menu has been selected.
       apkey         called when the user has pressed a key that has no
                     correspondance with any menu item.

    WRITTEN by Joelle Coutaz summer 86.

    MODIFIED by Sophie Bernard winter 87

    NOTATION USED FOR IDENTIFIERS:

       lower case identifiers are defined in this program.
       upper case identifiers are constants in this program.
       mixed case identifiers are identifiers imported from the Mac Toolbox.

*****************************************************************************/
```

```
#include ":.h:goodc.h"       /* to use our own syntax */
#include ":.h:win.h"         /* to use the extended window package */
#include ":.h:appli.h"

#include ":.h:debug.h"       /* to use the debug package */
#include ":.h:help.h"
#include ":.h:apresource.h" /* to access the pointers of the windows created
                             through resources */


/* ---------------------------- GLOBAL VARIABLES ---------------------------- *
/* ---------------------------- GLOBAL VARIABLES ---------------------------- *
/* ---------------------------- GLOBAL VARIABLES ---------------------------- *

    /*
    here, declare all of the coroutine id's needed by the application. Beware of
    the permitted number of coroutines. Declare them in appli.h as external
    variables because they are possibly needed in other modules (cf apmenu.c).
    */

/* ---------------------------- EXPORTED OPERATORS ---------------------------- *
/* ---------------------------- EXPORTED OPERATORS ---------------------------- *
/* ---------------------------- EXPORTED OPERATORS ---------------------------- *


/* ********************************* apbegin ************************************

    initialization of the application. Is called by topcontrol.c at start time.

-------------------------------------------------------------------------------
*/

apbegin ()
{
int myself;

/* tell the coroutine kernel that the main program is the main coroutine */

      myself   = cocoroutine ((long)0);

/* here, create all of the other coroutines with the "cocoroutine" function whic
   returns the id of the created coroutine. EX:
      coroutid = cocoroutine (&the_function_to_be_executed_by_the_coroutine); *


/* set the frequence at which aptask should be called */
      setaptasktimes(10);

/* call the initialization of all of the data in resources with the chosen
   language as parameter */
      apresinstall(FRENCH);

/* here call a function to initialize the domain dependent code */

}
```

```
/*
************************************ apdocommand ********************************

       Calls the  function corresponding to the user's choice. For example,
       a menu selection.

       IN :      funcptr   <pointer to an application routine >

-------------------------------------------------------------------------------
*/

apdocommand (funcptr)
int    (**funcptr)();
{
/* if you want to do some processing before or after the call to the function
   set it here */


/* the function call per se */
    if (*funcptr != 0) (**funcptr)();
}

/*
********************************** apend ************************************

    Called when the application has to shut down.

-------------------------------------------------------------------------------
*/

apend()
{
int macsbug;    /* to have access to the function into macsbug */

/* here, call the domain dependent function when the application has to shut down
*/

}

/*
********************************** apincontent ******************************

    application dependent stuffs when the window has been made active by
    a mouse click down in the content region.
    It acts as a dispatcher to the appropriate routine according to the
    clicked window.

    IN : whichwindow  the window where the click happened
         theevent     the type of the event

-------------------------------------------------------------------------------
*/

apincontent(whichwindow, theevent)
WindowPtr whichwindow;
```

```
EventRecord *theevent;

{
/* the user has clicked in the content of a window. If it is the about window we
must hide it */
       if (whichwindow == aboutwindow)
                begin
                HideWindow(aboutwindow);
                end

/* here, call the domain dependent function when a mouse click down
   has a meaning for the domain and has not been processed by a presentation
    objetc */


}

/*
************************************ apresizedw ********************************

   apresized window tells the application that a window has been resized

--------------------------------------------------------------------------------
*/

apresizedw(whichwindow)
WindowPtr whichwindow;
{

/* here, call the domain dependent function when resizing a window has a meaning
   for the domain dependent code */
}

/*
************************************ apscrolledw *******************************

   apscrolledw window tells the application that a window has been scrolled

--------------------------------------------------------------------------------
*/

apscrolledw(whichwindow)
WindowPtr whichwindow;
{

/* here, call the domain dependent function when scrolling a window has a meanin
   for the domain dependent code */

}

/*
************************************ apclosew **********************************

    application dependent stuffs when the goaway region of a window
    has been selected by the user.
```

```
-------------------------------------------------------------------------------
*/

apclosew(wid)
int wid;
{
/* if you use window's help as in the example, keep the two lines below */

/* in the case of a second window help, we must enable the item of the menu in
     order to permit another access to this item */

       if (wid == HTHERM2ID) EnableItem(GetMHandle(HELPMENU),1);

       if (wid == HBURN2ID) EnableItem(GetMHandle(HELPMENU),2);

/* here, call the domain dependent function when closing a window has a meaning
   for the domain dependent code */
}

/*
****************************** aptask ****************************************

    Called by the main loop in topcontrol.c on each cycle.
    Gives the application a chance to have some cycles.

-------------------------------------------------------------------------------
*/

aptask()
{
int macsbug;    /* to have access to the function into macsbug */

/* here, call the domain dependent function to be executed cyclicly */

}

/*
******************************** apabout *************************************

   is called when the user has selected the ABOUT item in the apple menu.

-------------------------------------------------------------------------------
*/
apabout()

 {
 int notmouseup;
 EventRecord myevent;
 GrafPtr saveport;

    ShowWindow(aboutwindow);
    SelectWindow(aboutwindow);

}
```

```
/*
********************************** apkey ***************************************

   is called when the user has pressed a key that has no correspondance
   with any menu item.

--------------------------------------------------------------------------------
*/
int apkey(hitkey, modifiers)
char hitkey;
int modifiers;

{

/* Here, call the function which interprets keystrokes */

}
```

Reférences Bibliographiques

[Adobe 85]
Adobe: PostScript Language Reference Manual, Addison Wesley, Reading, Mass., 1985.

[Acetta 86]
M. Acetta, R. Baron, W. Bolowsky, D. Golub, R. Rashid, A. Tevanian, M. Young : Mach, A New Kernel Foundation for UNIX Development ; Carnegie-Mellon University Technical Report, May, 1986.

[Ahlers 86]
K.L. Ahlers, A. Dwelly : OUTILS: Towards a User Interface Management System for Graphical Interaction ; Technical Report ECRC (European Computer-Industry Research Centre), October, 1986.

[Alletru 88]
J.C. Alletru, F. Fournier, O. Roussel, A. Vial : IRENE, Système expert d'aide à la configuration de réseau XNS, Architecture logicielle, Spécifications externes, Manuel utilisateur ; rapport de fin d'étude, DESS-Génie informatique, Université Joseph Fourier, juin, 1988.

[Anderson 83]
J.R. Anderson: The Architecture of Cognition ; Harvard University Press, Cambridge, Massachusetts, 1983.

[Balkovich 85]
E. Balkovich, S. Lerman, R.P. Parmelee : Computing in Higher Education: The Athena Experience ; Commun. ACM, 28(11), 1985, 1214-1224.

[Barnard 81]
P.J. Barnard, N.V. Hammond, J. Morton, J.B. Long, I.A. Clark : Consistency and Compatibility in Human-Computer Dialogue ; International Journal of Man-Machine studies, 15, 1981, 87-134.

[Barnard 86]
P.J. Barnard : Cognitive Resources and the Learning of Human-Computer Dialogue ; MRC Applied Psychology Unit, 15 Chaucer Road, Cambridge, England, March, 1986.

[Barnard 87]
P.J. Barnard, M. Wilson, A. MacLean : Approximate Modelling of Cognitive Activity: Towards an Expert System Design Aid ; CHI+GI Conference Proceedings (Toronto, April 5-9). ACM, New York,1987, pp. 21-26.

[Barth 86]
P. S. Barth : An Object-Oriented Approach to Graphical Interfaces ; ACM Transactions c Graphics 5(2), April, 1986.

[Barthet 86]
M. F. Barthet, C. Sibertin-Blanc : La modélisation d'applications interactives adaptées au utilisateurs par des réseaux de Petri à structure de donnée ; Actes du Troisièn Colloque-Exposition de Génie Logiciel, Versailles, mai, 1986, pp. 117-136.

[Bass 88]
L. Bass, E. Hardy, K. Hoyt, R. Little, R. Seacord : The Serpent run time architecture ar dialogue model ; Carnegie Mellon Technical Report, CMU/SEI-88-TR-6, January, 1988.

[Bernard 87a]
S. Bernard : Projet MOUSE : Outils pour la Construction d'Interfaces Homme-Machine, Manu d'Utilisation du gestionnaire de Dialogue ; Rapport de stage DESS-GI Université Joseph Fouri de Grenoble, Septembre, 1987.

[Bernard 87b]
S. Bernard, L. Lys, P. Marchal, C. Po : Projet MOUSE : Outils pour la Constructic d'Interfaces Homme-Machine, Spécifications externes, Architecture logicielle ; Rapport de proj DESS-GI Université Joseph Fourier de Grenoble, Septembre, 1987.

[Berry 86a]
G. Berry, F. Boussinot, P. Couronné, G. Gonthier : ESTEREL V2.2 System Manuals Rapports techniques, Ecole des Mines, Sophia Antipolis, 1986.

[Berry 86b]
G. Berry, P. Couronné, G. Gonthier : Synchronous Programming of Reactive systems: a introduction to ESTEREL ; Proceedings of the First France-Japan Symposium on Artific Intelligence and Computer Science, Tokyo, October 1986, North Holland.

[Bly 86]
S. Bly, J.K. Jarret : A Comparison of Tiled and Overlapping Windows ; Computer Huma Interaction Conference Proceedings (Boston, April 13-17). ACM, New York,1986, p 101-106.

[Bobrow 83]
D.G. Bobrow, M. Stefik : The Loops Manual ; Tech. report KB-VLSI-81-13, Knowledg Systems Area, Xerox, Palo Alto Research Center, 1981.

[Borning 86a]
A.H. Borning : Graphically Defining New Building Blocks in ThingLab ; Human Comput Interaction, 2(4), 1986, pp. 269-295.

[Borning 86b]
A.H. Borning : Defining Constraints Graphically ; Computer Human Interaction Conferen Proceedings (Boston, April 13-17). ACM, New York,1986, pp. 137-143.

[Borras 87]
P. Borras, D. Clément, T. Despeyroux, J. Incerpi, G. Kahn, B. Lang, V. Pascual : CENTAUI the system ; Rapport de Recherche INRIA no 777, Domaine de Voluceau, BP 105, 78153 I Chesnay Cedex, décembre 1987.

[Brownston 85]
L. Brownston, R. Farrell, E. Kant, N. Martin : Programming Expert Systems in OPS5, A

Introduction to Rule-Based Programming ; Addison Wesley Publ., 1985.

[Buchanan 84]
B.G. Buchanan, E.H. Shortliffe : Rule-Based Expert Systems, The MYCIN Experiments of the Stanford Heuristic Programming Project ; Addison Wesley Publ., 1984.

[BULL 85]
Introduction générale à EMERAUDE ; Société Bull, Route de Versailles, Louveciennes, 1985.

[Buxton 82]
W. Buxton : An Informal Study of Selection Positionning Tasks ; Graphics Interface'82, 1982, pp. 323-328.

[Buxton 83]
W. Buxton, M.R. Lamb, D. Sherman, K.C. Smith : Towards a Comprehensive User Interface Management System ; Computer Graphics 17(3), July 1983, pp. 35-42.

[Card 83]
S. Card, T. Moran, A. Newell : The Psychology of Human-Computer Interaction, ISBN 0-89859-243-7, Lawrence Erlbaum Associates, Publish., 1983.

[Cardelli 85]
L. Cardelli, R. Pike : Squeak: a Language for Communicating with Mice ; Computer Graphics, SIGGRAPH'85 Conference Proceedings, San Francisco, ACM, 19(3), 1985, pp. 199-204.

[Cardelli 87]
L. Cardelli : Building User Interfaces by Direct Manipulation ; Digital Systems Research Center, Technical Report, 22, October, 1987.

[Carroll 83]
J.M. Carroll : Presentation and Form in User Interface Architecture ; Byte 8(12), December, 1983, pp. 113-122.

[Carroll 84]
J.M. Carroll, C. Carrithers : Training Wheels in a User Interface ; Communication of the ACM, 27(8), August, 1984, pp. 800-807.

[Carroll 85a]
J.M. Carroll, D.S. Kay : Prompting, Feedback and Error Correction in the Design of a Scenario Machine ; Proceedings of the CHI'85 Conference, The Association for Computing Machinery Publ., April 1985, pp. 149-153.

[Carroll 85b]
J.M. Carroll, R.L. Mack : Metaphor, Computing Systems, and Active Learning ; International Journal of Man-Machine Studies, 22(1), January, 1985, pp. 39-57.

[Ciccarelli 84]
E.C. Ciccarelli : Presentation Based User Interfaces, Technical Report 794, Artificial Intelligence Laboratory, Massachusetts Intelligence Laboratory, August 1984.

[Clement 87]
D. Clément, J. Incerpi : Specifying Behavior of Graphical Objects Using Esterel ; Rapport de Recherche INRIA no 836, Avril, 1988.

[Cohen 86]
E.S. Cohen, E.T. Smith, L.A. Iverson : Constraint-Based Tiled Windows ; IEEE Computer

Graphics and Applications, 6(5), May, 1986, pp. 35-45.

[Collet 87]
C. Collet : Les Formulaires Complexes dans les Bases de Données Multimédia ; Thèse Doctorat de l'Université Scientifique Technonologique et Médicale de Grenoble, novembre 19

[Conchon 86]
A. Conchon, J. Camacho, A.M. Rasser : Une session sur le poste de travail Concerto ; Actes 3ème Colloque-Exposition de Génie Logiciel, Versailles, AFCET, mai, 1986, pp. 57.68.

[Conklin 87]
J. Conklin : Hypertext: An Introduction and Survey ; IEEE Computer, 20(9), September 19 pp. 17-41.

[Coutaz 84a]
J. Coutaz : Towards Friendly Systems, Design Concepts for Interactive Interface ; Proceedin of the ACM SouthEast Regional Conference, Atlanta, April, 1984, pp. 56-61.

[Coutaz 84b]
J. Coutaz, M. Herrmann : Adèle et le Médiateur-Compositeur ou Comment rendre u Application Interactive indépendante de l'Interface Usager ; Actes du deuxième colloque de Gé Logiciel, Nice, Juin, 1984, pp. 195-212.

[Coutaz 84c]
J. Coutaz : A Proposal for a Help System for Spice ; Carnegie-Mellon University, private not December, 1984.

[Coutaz 85a]
J. Coutaz : A Layout Abstraction for User System Design ; ACM SIGCHI, January 1985, p 18-24. Paru également dans Carnegie-Mellon University Technical Report, CMU-CS-84-1 December, 1984.

[Coutaz 85b]
J. Coutaz : Abstractions for User Interface Design ; IEEE Computer, 18(9), September, 19 pp. 21-34.

[Coutaz 86a]
J. Coutaz : Abstractions for User Interface Toolkits, IFIP WG2.7 Working Conference on t Future of Command Languages, Rome, Septembre, 1985 ; paru également dans Foundation f Human-Computer Communication, éditeurs K. Hopper et I.A. Newman, North Holland, 19 pp. 335-354.

[Coutaz 86b]
J. Coutaz : Abstractions pour la construction d'interfaces Homme-Machine, Techniques Science Informatiques, vol 5(4), juillet-août 1986, pp. 239-250.

[Coutaz 86c]
J. Coutaz : La Construction d'Interfaces Homme-Machine, Rapport IMAG RR 635-I, novemb 1986.

[Coutaz 87 a]
J. Coutaz, F. Berthier : The construction of User Interfaces, IFIP WG8.4 Working Conferen on Methods and Tools for Office Systems, Pisa, octobre 1986, pp. 57-64 ; paru égalemene dans Office Systems: Methods and Tools, éditeurs G. Bracchi, D. Tsichritzis, North Hollar 1987, pp. 59-66.

[Coutaz 87b]
J. Coutaz : The Construction of User Interface and the Object Oriented Paradigm ; ECOOP'87, The European Conference on Object Oriented Programming, Paris, June, 1987, pp. 135-144.

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

[Coutaz 87d]
J. Coutaz : PAC, an Object Oriented Model for Implementing User Interfaces ; CHI+GI '87 Poster session Papers, ACM SIGCHI Bulletin, 19(2), October 1987, pp.37-41.

[Coutaz 88a]
J. Coutaz : De l'ergonome à l'informaticien : pour une méthode de conception et de réalisation des interfaces homme-machine ; Conférence invitée, Actes du colloque européen ERGO-IA'88, Ergonomie et Intelligence Artificielle, Biarritz, 4-6 octobre, 1988.

[Coutaz 88b]
J. Coutaz, L. Bass : Ergonomics and Software Principles for the Construction of Interactive Software ; Rapport de Recherche LGI (IMAG) RR 732-I, BP 53X, 38042 Grenoble Cedex, septembre, 1988.

[Crowley 86a]
J. Crowley, J. Coutaz : Navigation et Modélisation pour un Robot Mobile ; Rapport de Recherche LIFIA RR 28, IMAG RR 545, juin, 1986.

[Crowley 86b]
J. Crowley, J. Coutaz : Navigation et Modélisation pour un Robot Mobile ; Techniques et Science Informatiques, vol 6(4), septembre-octobre 1986.

[Doaré 87]
C. Doaré, V. Sabete, D. Salomon, A. Valette : Noyau de système Expert en Loops, Architecture logicielle, Spécifications externes, Manuel utilisateur, Commandes usuelles ; Rapport de fin d'étude, DESS-Génie informatique, Université Joseph Fourier, juin, 1987.

[Duce 87]
D.A. Duce, F.R.A. Hopgood : The Graphical Kernel System ; Computer-Aided Design, 19(8), October, 1987, pp. 396-409.

[Duisberg 86]
R.A. Duisberg : Animated graphical interfaces using temporal constraints ; Computer Human Interaction Conference Proceedings (Boston, April 13-17). ACM, New York, 1986, pp. 131-136.

[Ege 87]
R.K. Ege, D. Maier, A. Borning : The Filter Browser Defining Interfaces Graphically ; ECOOP'87, The European Conference on Object Oriented Programming, Paris, June, 1987, pp. 155-165.

[Estubier 83]
J. Estublier, S. Krakowiak, J. Mossière, Y. Rouzaud : Design Principles of the Adèle Programming Environment ; International Computing Symposium (ACM), March, 1983, pp. 149-156.

[Estublier 86]
J. Estublier, N. Beltakir : Experience with a Data Base of Programs ; Proceedings of the Second

Symposium on Practical Software Environments, Palo Alto, Sigplan/Sigsoft, December, 198

[Foley 81]
J.D. Foley, W.L. Wallace, P. Chan : The Human Factors of Graphic Interaction Tasks Techniques ; Technical Report GWU-IIST-81-3, Institute for Information Science Technology, The George Washington University, 1981.

[Foley 84]
J. D. Foley, A. Van Dam : Fundamentals of Interactive Computer Graphics ; Addison Wesl Publ., 1984.

[Foley 87]
J.D. Foley, W.C. Kim, C.A. Gibbs : Algorithms to Transform the Formal Specification User-Computer Interface ; Conference Proceedings of Human-Computer Interaction Interact H.J. Bullinger, B.Shackel ed., North Holland, 1987, pp. 1001-1006.

[Foley 88]
J.D. Foley, W.C. Kim, S. Kovacevic, K. Murray : The User Interface Design Environm Technical Report GWU-IIST-88-04, Department of Electrical Engineering and Compu Science, The George Washington University, Washington D.C. 20052, January, 1988.

[Forgy 82]
C.L. Forgy : Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Proble Artificial Intelligence, 19(1), North Holland Publishing Company, September, 1982, pp. 17-

[Forgy 84]
C.L. Forgy : The OPS83 report ; Tech. Report CMU-CS-84-113, Carnegie-Mellon Univers Computer Science, 1984.

[Geiben 88]
C. Geiben, P.A. Tourtier : Projet Concept ; Projet de fin d'études, Institut Natio Polytechnique de Grenoble E.N.S.I.M.A.G, juin, 1988.

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

[Gosling 86a]
J. Gosling : Partitioning of Functions in Window Systems ; in Methodology of Wind Management, Hopgood ed., Springer Verlag, 1986, pp. 101-106.

[Gosling 86b]
J. Gosling : SunDew: A distributed and Extensible Window System ; Proceedings of the Wir 1986 USENIX Conference, January, 1986, pp. 98-103.

[Gosling 86c]
J. Gosling, D. Rosenthal : A Window Manager for BitMapped Displays and Unix ; Methodology of Window Management, Hopgood ed., Springer Verlag, 1986, pp. 115-128.

[Granor 86]
T.E. Granor : A User Interface Management System Generator ; PhD Thesis, Department Computer and Information Science, University of Pennsylvania, Philadelphia, PA. Techni Report MS-CIS-86-42 Graphics Lab 12, May, 1986.

[Green 85]
M.W. Green : The Design of Graphical Interfaces ; PhD Thesis, Tech. Report CSRI-1

Computer Systems Research Institute, University of Toronto, Canada, M5S 1A1, April, 1985.

[GSPC 79]
Status Report of the Graphics Standards Comittee ; Computer Graphics, 13(3), August, 1979.

[Halasz 87]
F.G. Halasz, T.P. Moran, R.H. Trigg : NoteCards in a Nutshell ; CHI+GI Conference Proceedings (Toronto, April 5-9). ACM, New York, 1987, pp. 45-53 .

[Harel 87]
D. Harel : Statecharts: A visual formalism for complex systems ; Science of Computer Programming, 8(3), June, 1987, pp. 231-274.

[Harvey 88]
G. Harvey : Understanding HyperCard for Version 1.1 ; Sybex Books Publishers, 1988.

[Hayes 83]
P.J. Hayes, P. Szekely : Graceful Interaction through the Cousin Command Interface ; International Journal of Man Machine Studies 19(3), September, 1983, pp. 285-305.

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

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

[Helfman 87]
J.I. Helfman : Panther: A Specification System for Graphical Controls ; CHI+GI Conference Proceedings (Toronto, April 5-9). ACM, New York, 1987, pp. 279-284.

[Henderson 86]
A. Henderson : The Trillium User Interface Design Environment ; Proceedings SIGCHI'86 : Human Factors in Computing Systems. Boston MA. April 13-17, 1986, pp. 221-227.

[Herrmann 84]
M. Herrmann : Interface Usager-Application dans un Atelier de Génie Logiciel ; Thèse Doctorat de Troisième Cycle, Université Scientifique Technologique et Médicale de Grenoble, Octobre, 1984.

[Hibbard 84]
P. Hibbard : Mint User's Manual ; Carnegie-Mellon University, 1984.

[Hill 87a]
R.D. Hill : Supporting Concurrency, Communication and Synchronization in Human-Computer Interaction ; PhD Thesis, Department of Computer Science, University of Toronto, January, 1987 ; voir aussi : Supporting Concurrency, Communication and Synchronization dans Human-Computer Interaction-The Sassafras UIMS ; ACM Transactions on Graphics 5(2), April, 1986, pp. 179-210.

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

[Hoare 78]
C.A.R. Hoare : Communicating Sequential Processes ; Communications of the ACM, 21(1978, pp. 666-678.

[Hullot 86]
J.M. Hullot : SOS Interface, un Générateur d'Interfaces Homme-Machine, Actes des Journ Afcet-Informatique sur les Langages Orientés Objet, Bigre+Globule, 48, Publ. IRISA, Camp de Beaulieu, 35042 Rennes, janvier, 1986, pp. 69-78.

[Hutchins 86]
E. L. Hutchins, J. D. Hollan, D. A. Norman : Direct Manipulation Interfaces ; User Cente System Design ; Lawrence Erlbaum Associates, Publishers, 1986, pp. 87-124.

[Ingalls 88]
D.H. Ingalls, S. Wallace, Y. Chow, F. Ludolph, K. Doyle : Fabrik, A visual Programmi Environment ; OOPSLA'88 proceedings, 1988, pp. 176-190.

[ISO 85]
International Organization for Standardization : Information processing systems - Compu graphics - Graphical Kernel System (GKS) functional description ; ISO IS 7942, July, 1985.

[ISO 86a]
International Organization for Standardization : Information processing systems - Compu graphics - Programmer's Hierarchical Interface to Graphics (PHIGS) functional descriptio ISO DP 9592, October, 1986.

[ISO 86b]
International Organization for Standardization : Information processing systems - Compu graphics - Techniques for interfacing graphical devices (CGI) functional description ; ISO 9636, December, 1986.

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

[John 85]
B.E. John, P.S. Rosemblom : A theory of Stimulus-Response Compatibility Applied to Hun Computer Interaction ; CHI'85 Conference Proceedings (San Francisco, April 14-18). AC New York, 1985, pp. 213-220.

[John 87]
B.E. John, A. Newell : Predicting the Time Recall Computer Command Abbreviations ; CHI+ Conference Proceedings (Toronto, April 5-9). ACM, New York, 1987, pp. 33-40.

[Jones 82]
M.B. Jones, R.F. Rashid, M. Thompson : Sesame: The Spice File System ; Tech. Repo Carnegie-Mellon University, October, 1982.

[Karsenty 87]
S. Karsenty : Graffiti : un outil interactif et graphique pour la construction d'interfa homme-machine adaptables ; Thèse de doctorat de 3ème cycle Informatique, Université Paris-Sud, Centre d'Orsay, décembre, 1987.

[Kieras 85]
D. Kieras, P.G. Polson : An Approach to the Formal Analysis of User Complexity ; International Journal of Man-Machine Studies, 22, 1985, pp. 365-394.

[Kiger 84]
J.I. Kiger : the depth/breadth trade-off in the design of menu-driven user interfaces ; International Journal of Man-Machine Studies, 20, 1984, pp. 201-213.

[Knuth 79]
D.E. Knuth : TEX and Metafont : New Directions in Typesetting ; Digital Press, 1979.

[Krakowiak 87]
S. Krakowiak, M. Meysembourg, M. Riveill, C. Roisin : Modèle d'Objet et Langage du Système Guide ; Rapport Guide R2, Laboratoire de Génie Informatique (IMAG), Novembre, 1987.

[Lantz 86]
K.A. Lantz : On User Interface Reference Models ; ACM SIGCHI Bulletin, 18(2), pp. 36-44.

[Lermigeaux 86]
F. Lermigeaux : Un Modèle d'Architecture pour Application Interactive sur Macintosh ; Rapport de Stage de DESS Génie Informatique, Centre de Recherche Bull IMAG, Septembre, 1986.

[Lieberman 85]
H. Lieberman : There's More to Menu Systems than Meets the Screen ; SIGGRAPH'85, Computer Graphics, 19(3), 1985, pp. 181-189.

[Lieberman 87]
H. Lieberman : Reversible Object-Oriented Interpreters, ECOOP'87, The European Conference on Object Oriented Programming, Paris, june, 1987, pp. 13-22.

[Linton 86]
M. Linton, C. Dunwoody : Partitioning User Interfaces with Interactive Views ; Computer Systems Laboratory, Stanford University, Stanford, CA 94305-2192, communication privée, April, 1986.

[Lopez 83]
M. Lopez, J. Palazzo, F. Velez : The Tigre Data Model ; Rapport de Recherche IMAG/Centre de Recherche Bull RR-Tigre no 2, novembre, 1983.

[Lunati 88]
J.M. Lunati, V. Normand : Un Serveur d'Interaction Centré Objet ; Projet de fin d'études, Institut National Polytechnique de Grenoble, E.N.S.I.M.A.G, juin, 1988.

[Lynch 86]
G. Lynch, J. Meads : In Search of a User Interface Reference Model : Report on the SIGCHI workshop on User Interface Reference Models ; SIGCHI Bulletin, 18(2), October, 1986, pp. 25-33.

[Meyer 87]
B. Meyer : Reusability : The Case for Object-Oriented design ; IEEE Software, March, 1987, pp. 50-59.

[Michard 82]
A. Michard : Graphical presentation of boolean expressions in a data base query language: design notes and an ergonomic evaluation ; Behaviour and Information Technology, 1(3), 1982, pp.

279-288.

[Mikelsons 81]
M. Mikelsons : Prettyprinting in an Interactive Programming Environment ; Proceedings of ACM Sigplan SIGOA Symposium on Text Manipulation, June, 1981.

[Miller 75]
G.A. Miller : The Psychology of Communication ; Basic Books Publ., New York, sec edition, 1975.

[Miller 81]
D.P. Miller : The depth/breadth trade-off in hierarchical computer menus ; Proceedings of 25th Annual Meeting of the Human Factors Society, 1981, pp. 296-300.

[MIT 87]
X Toolkit Library - C Language interface, X protocol Version 11, 1987.

[Moran 81]
T. Moran : The Command Language Grammar: a representation for the user interface interactive computer systems ; International Journal of Man-Machine Studies, 15, 1981, 3-50.

[Moran 83]
T.P. Moran : Getting into a System: External-Internal Task Mapping Analysis ; Compu Human Interaction'83, Boston, ACM SIGCHI Bulletin special issue, 1983, pp. 45-49.

[Morcos 86a]
E. Morcos-Chounet, M.J. Brossard, A. Conchon : Affichage Interactif d'Arbres Abstrai Troisième Colloque-Exposition de Génie Logiciel, Versailles, AFCET, mai, 1986, pp. 137-1

[Morcos 86b]
E. Morcos-Chounet, A. Conchon : PPML, A general formalism to specify pretty-printir Proceedings of IFIP Congress 1986, Dublin, North Holland, 1986.

[Morris 86]
J.H. Morris, M. Satyanarayanan, M.H. Conner, J.H. Howard, D.S.H. Rosenthal, F.D. Sm Andrew: A Distributed Personal Computing Environment ; Communications of the ACM 29(March, 1986, pp. 184-201.

[Myers 84]
B.A. Myers : The User Interface of Sapphire ; IEEE Computer Graphics and Applications 4(1 December, 1984, pp. 13-23.

[Myers 87a]
B.A. Myers : Creating dynamic interaction techniques by demonstration ; Proceedings of A(CHI+GI'87 Conference Human Factors in Computing Systems and Graphics Interfa Toronto, 5-9 April, 1987, pp. 271-278.

[Myers 87b]
B.A. Myers : Creating User Interfaces by Demonstration ; PhD thesis, Department of Compu Science, University of Toronto, Technical Report CSRI-196, Computer Systems Resea Institute, May, 1987.

[Nanard 84]
J. et M. Nanard : Manipulation Interactive de Documents ; Techniques et Science Informatiqu 3(6), 1984, pp. 443-451.

[Neal 87]
L. R. Neal : Cognition-Sensitive Design and User Modelling for Syntax-Directed Editors ; Proceedings of ACM CHI+GI'87 Conference Human Factors in Computing Systems and Graphics Interface, Toronto, 5-9 April, 1987, pp. 99-102.

[Nelson 85]
G. Nelson : Juno, a Constraint-Based Graphics System ; Computer Graphics : SIGGRAPH'85 Conference Proceedings, San Francisco, 19(3), July 1985, pp. 235-243.

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

[Norman 86]
D. A. Norman, S. W. Draper : User Centered System Design ; Lawrence Erlbaum Associates, Publishers, 1986.

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

[Pato 84]
J.N. Pato, S.P. Reiss, M.H. Brown : The Brown University Environment ; Technical Report CS-84-03, Department of Computer Science, Brown University, Providence, RI 02912, USA, january, 1984.

[Payne 86]
S.J. Payne, T.R.G. Green : Task-Action Grammars: A model of the mental representation of task languages ; Human Computer Interaction, 2, 1986, pp. 93-133.

[Pfaff 82]
G. Pfaff, H. Kuhlman, H. Hanusa : Constructing User Interfaces Based on Logical Input Devices ; Computer, IEEE, 15(11), November, 1982, pp. 62-68.

[Pfaff 85]
User Interface Management Systems ; G. E. Pfaff ed., Eurographics Seminars, Springer-Verlag, 1985.

[Polson 85]
P.G. Polson, D.E. Kieras : A Quantitative Model of the Learning and Performance of Text Editing Knowledge ; CHI'85 Conference Proceedings (San Francisco, April 14-18). ACM, New York, 1985, pp. 207-212.

[Price 83]
L.A. Price, C.A. Cordova : Use of Mouse Buttons ; CHI'83 Conference Proceedings (Boston, December 12-15) ; ACM, New York, 1983, pp. 263-266.

[Quint 83]
V. Quint : Un système interactif pour la Production des Documents Mathématiques ; Techniques et Science Informatiques, 2(3), 1983, pp. 179-190.

[Quint 85]
V. Quint, I. Vatton : Grif : un Editeur Interactif Structuré ; Rapport de Recherche TIGRE no 27, IMAG-Laboratoire de Génie Informatique, BP 53 X, 38042 Grenoble Cedex.

[Quint 87]
V. Quint : Une Approche de l'Edition Structurée des Documents ; Thèse de doctorat d'I Université Scientifique Technologique et Médicale de Grenoble, 1987.

[Robertson 81]
C.K. Robertson, D.L. McCracken, A. Newell : Experimental Evaluation of the ZOG Fr Editor ; Technical report CMU TR 81-112, Carnegie-Mellon University, 1981.
G.D Robertson, D.L. McCracken, A. Newell : The ZOG approach to Man-Macl Communication ; International Journal of Man-Machine Studies, 14(4), 1981, pp. 461-488.

[Rose 86]
C. Rose et al.: Inside Macintosh ; Addison Wesley Publ., 1986.

[Rosson 83]
M.B. Rosson : Patterns of Experience in Text Editing ; Proceedings of CHI'83 conferenc Human Factors in Computer Systems, The Association for Computing Machinery Pu December, 1983, pp. 177-183.

[Sacerdoti 74]
E.D. Sacerdoti : Planning in a Hierarchy of Abstraction Spaces. Artificial Intelligence, 5, 1 pp. 115-135.

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

[Schank 77]
R.C. Schank, R.P. Abelson : Scripts, Plans, Goals, and Understanding: An Inquiry into Hur Knowledge Structures. Hillsdale N.J. , Erlbaum Associates, 1977.

[Scheifler 86]
R.W. Scheifler, J. Gettys : The X Window System ; ACM Transactions on Graphics 5 April, 1986, pp. 79-109.

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

[Schmucker 86b]
K. Schmucker : Object-Oriented Languages for the Macintosh ; Byte 11(8), 1986, 177-188.

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

[SEMS 85]
SPS7, Système Graphique : Ensemble des primitives pour l'écriture d'applications graphiqu Bull-Sems, 20 893 759 REF 01/FR, juin, 1985.

[Shafer 88]
D. Shafer : HyperTalk Programming ; MacIntosh Library, Hayden Books, Indianapc Indiana, USA.

[Shneiderman 82]
B. Shneiderman : Multiparty Grammars and Related Features for Defining Interactive Syste

IEEE Transactions on Systems, Man and Cybernetics, SMC-12, no 2, 1982, pp. 148-154.

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

[Shuey 86]
D. Shuey, D. Bailey, T.P. Morrissey : PHIGS: A Standard, Dynamic, Interactive Graphics Interface ; IEEE Computer Graphics and Application, August, 1986, pp. 50-57.

[Shuey 87]
D. Shuey : PHIGS: a graphics platform for CAD application development ; Computer-Aided Design, 19(8), october, 1987, pp. 410-417.

[Sibert 86]
J.L. Sibert, W.D. Hurley, T.W. Bleser : An Object Oriented User Interface Management System; SIGGRAPH'86, 20(4), 1986, pp. 259-268.

[Simon 84]
H. A. Simon : The Sciences of the Artificial ; The MIT Press, third edition, 1984.

[Sisson 86]
N. Sisson : Dialogue Management Reference Model ; ACM SIGCHI, 18(2), October 1986, pp. 34-35.

[Smith 82]
D.C. Smith, C. Irby, R. Kimball, B. Verplank : Designing the Star User Interface ; Byte, 7(4), April 1982, pp. 242-282.

[Smith 83]
R.G. Smith : Strobe: Support for structured object knowledge representation ; Proceedings of the 8th Joint Conference on Artificial Intelligence, Karlsruhe, William Kaufman ed., 1983, pp. 855-858.

[Smith 87]
R.B. Smith : Experiences with the Alternate Reality Kit, an Exemple of the Tension between Literalism and Magic ; Computer Human Interaction Conference Proceedings (Toronto, April 5-9). ACM, New York, 1987, pp. 61-67.

[Stallman 79]
R.M. Stalman : Emacs : The Extensible Customizable Self Documenting Display Editor ; MIT Artificial Intelligence Laboratory, AI Memo 519, Cambridge, Mass., 1979.

[Stefik 87]
M. Stefik, G. Foster, D.G. Bobrow, K. Hahn, S. Lanning, L. Suchman : Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings ; Communications of the ACM, 30(1), January, 1987, pp. 32-47.

[SUN 85]
SUN Microsystems Inc. : Programmer's Reference Manual for the Sun Window System ; SUN Microsystems Inc., 2250 Garcia Avenue, Mountain View, CA 94043.

[SUN 87]
SUN Microsystems Inc. : NeWS Manual ; SUN Microsystems Inc., 2250 Garcia Avenue, Mountain View, CA 94043.

[Sweetman 86]
D. Sweetman : A modular Window System for Unix ; dans Methodology of Wind Management, Hopgood ed., Springer Verlag, 1986, pp. 73-79.

[Szekely 88]
P.A. Szekely, B.A. Myers : A User Interface Toolkit Based on Graphical Objects ; Constraints ; OOPSLA'88 proceedings ; ACM Conference on Object-Oriented Programm Systems and Applications, San Diego, California, 25-30 septembre, 1988, pp. 36-45.

[Tanner 83]
P. Tanner, W. Buxton : Some Issues in Future User Interface Management Systems (UIM Development. IFIP Working Group 5.2 Workshop on User Interface Management, Seehe West Germany, November, 1983.

[Tesler 86]
L. Tesler : Programming Experiences ; Byte, 11(8), August, 1986, pp. 195-206.

[Trigg 87]
R.H. Trigg, T.P. Moran, F.G. Halasz : Adaptability and Tailorability in NoteCards ; Confere Proceedings of Human-Computer Interaction Interact'87, H.J. Bullinger, B.Shackel ed., Nc Holland, 1987, pp. 723-728.

[Tufte 83]
E.R. Tufte : The Visual Display of Quantitative Information ; Graphics Press, Box 4 Cheshire, Connecticut 06410, 1983.

[Tullis 85]
T.S. Tullis : Designing a Menu-Based Interface to an Operating System ; CHI'85 Confere Proceedings (San Francisco, April 14-18), 1985, pp. 79-84.

[Uebbing 86]
J. Uebbing, C. Young : User Interface Performance Issues ; Byte 11(8), 1986, pp. 176-176.

[VIP 86]
Visual Interactive Programming ; distrib. EMDAY, 46 av. Tervieren, B-1040 Bruxelles.

[Warnock 82]
J. Warnock, D.K. Wyatt : A Device Independent Graphics Imaging Model for Use with Ras Devices ; Computer Graphics, 16(3), July, 1982, pp. 313-319.

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

TABLE DES MATIERES

SECTION II : ARCHITECTURE LOGICIELLE DES SYSTEMES INTERACTIFS

AUTORISATION DE SOUTENANCE

DOCTORAT D'ETAT

Vu les dispositions de l'Article 5 de l'Arrêté du 16 avril 1974,

Vu les rapports de Monsieur....Gilbert..Kahn.....

Monsieur...Patrick..Baudelaire.....

Monsieur...Sacha..Kakovkiak.....

Madame...Joëlle.....COUTAZ..... est autorisé à présenter une thèse en vue de l'obtention du grade de **DOCTEUR D'ETAT ES SCIENCES.**

Fait à Grenoble, le 12 DEC. 1988

Le Président de l'U.S.T.M.G.

J.J PAYAN