Interface usager - application dans un atelier de génie logiciel

Marc Herrmann

THESE

ntée à

l' Université Scientifique et Médicale de Grenoble

pour obtenir le grade de
DOCTEUR DE 3ème CYCLE
«Informatique»

par

Marc HERRMANN

INTERFACE USAGER–APPLICATION DANS UN ATELIER DE GENIE LOGICIEL.

Thèse soutenue le 26 octobre 1984 devant la commission d'examen.

S. KRAKOWIAK	Président
V. JOLOBOFF	
F. MARTINEZ	Examinateurs
J. MOSSIERE	

UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE

Année universitaire 1982-1983

Président de l'Université : M. TANCHE

MEMBRES DU CORPS ENSEIGNANT DE L'U.S.M.G.

(RANG A)

SAUF ENSEIGNANTS EN MEDECINE ET PHARMACIE

PROFESSEURS DE 1ère CLASSE

ARNAUD Paul	Chimie organique
ARVIEU Robert	Physique nucléaire I.S.N.
AUBERT Guy	Physique C.N.R.S.
AYANT Yves	Physique approfondie
BARBIER Marie-Jeanne	Electrochimie
BARBIER Jean-Claude	Physique expérimentale C.N.R.S. (labo de magnétisme)
BARJON Robert	Physique nucléaire I.S.N.
BARNOUD Fernand	Biosynthèse de la cellulose-Biologie
BARRA Jean-René	Statistiques - Mathématiques appliquées
BELORISKY Elie	Physique
BENZAKEN Claude (M.)	Mathématiques pures
BERNARD Alain	Mathématiques pures
BERTRANDIAS Françoise	Mathématiques pures
BERTRANDIAS Jean-Paul	Mathématiques pures
BILLET Jean	Géographie
BONNIER Jean-Marie	Chimie générale
BOUCHEZ Robert	Physique nucléaire I.S.N.
BRAVARD Yves	Géographie
CARLIER Georges	Biologie végétale
CAUQUIS Georges	Chimie organique
CHIBON Pierre	Biologie animale
COLIN DE VERDIERE Yves	Mathématiques pures
CRABBE Pierre (détaché)	C.E.R.M.O.
CYROT Michel	Physique du solide
DAUMAS Max	Géographie
DEBELMAS Jacques	Géologie générale
DEGRANGE Charles	Zoologie
DELOBEL Claude (M.)	M.I.A.G. Mathématiques appliquées
DEPORTES Charles	Chimie minérale
DESRE Pierre	Electrochimie
DOLIQUE Jean-Michel	Physique des plasmas
DUCROS Pierre	Cristallographie
FONTAINE Jean-Marc	Mathématiques pures
GAGNAIRE Didier	Chimie physique

.../...

GASTINEL Noël	Analyse numérique - Mathématiques appliquées
GERBER Robert	Mathématiques pures
GERMAIN Jean-Pierre	Mécanique
GIRAUD Pierre	Géologie
IDELMAN Simon	Physiologie animale
JANIN Bernard	Géographie
JOLY Jean-René	Mathématiques pures
JULLIEN Pierre	Mathématiques appliquées
KAHANE André (détaché DAFCO)	Physique
KAHANE Josette	Physique
KOSZUL Jean-Louis	Mathématiques pures
KRAKOWIAK Sacha	Mathématiques appliquées
KUPTA Yvon	Mathématiques pures
LACAZE Albert	Thermodynamique
LAJZEROWICZ Jeannine	Physique
LAJZEROWICZ Joseph	Physique
LAURENT Pierre	Mathématiques appliquées
DE LEIRIS Joël	Biologie
LLIBOUTRY Louis	Géophysique
LOISEAUX Jean-Marie	Sciences nucléaires I.S.N.
LOUP Jean	Géographie
MACHE Régis	Physiologie végétale
MAYNARD Roger	Physique du solide
MICHEL Robert	Minéralogie et pétrographie (géologie)
MOZIERES Philippe	Spectrométrie - Physique
OMONT Alain	Astrophysique
OZENDA Paul	Botanique (biologie végétale)
PAYAN Jean-Jacques (détaché)	Mathématiques pures
PEBAY PEYROULA Jean-Claude	Physique
PERRIAUX Jacques	Géologie
PERRIER Guy	Géophysique
PIERRARD Jean-Marie	Mécanique
RASSAT André	Chimie systématique
RENARD Michel	Thermodynamique
RICHARD Lucien	Biologie végétale
RINAUDO Marguerite	Chimie CERMAV
SENGEL Philippe	Biologie animale
SERGERAERT Francis	Mathématiques pures
SOUTIF Michel	Physique
VAILLANT François	Zoologie
VALENTIN Jacques	Physique nucléaire I.S.N.
VAN CUTSEN Bernard	Mathématiques appliquées
VAUQUOIS Bernard	Mathématiques appliquées
VIALON Pierre	Géologie

PROFESSEURS DE 2ème CLASSE

ADIBA Michel	Mathématiques pures
ARMAND Gilbert	Géographie

.../...

AURIAULT Jean-Louis	Mécanique
BEGUIN Claude (M.)	Chimie organique
BOEHLER Jean-Paul	Mécanique
BOITET Christian	Mathématiques appliquées
BORNAREL Jean	Physique
BRUN Gilbert	Biologie
CASTAING Bernard	Physique
CHARDON Michel	Géographie
COHENADDAD Jean-Pierre	Physique
DENEUVILLE Alain	Physique
DEPASSEL Roger	Mécanique des fluides
DOUCE Roland	Physiologie végétale
DUFRESNOY Alain	Mathématiques pures
GASPARD François	Physique
GAUTRON René	Chimie
GIDON Maurice	Géologie
GIGNOUX Claude (M.)	Sciences nucléaires I.S.N.
GUITTON Jacques	Chimie
HACQUES Gérard	Mathématiques appliquées
HERBIN Jacky	Géographie
HICTER Pierre	Chimie
JOSELEAU Jean-Paul	Biochimie
KERCKOVE Claude (M.)	Géologie
LE BRETON Alain	Mathématiques appliquées
LONGEQUEUE Nicole	Sciences nucléaires I.S.N.
LUCAS Robert	Physiques
LUNA Domingo	Mathématiques pures
MASCLE Georges	Géologie
NEMOZ Alain	Thermodynamique (CNRS - CRTBT)
OUDET Bruno	Mathématiques appliquées
PELMONT Jean	Biochimie
PERRIN Claude (M.)	Sciences nucléaires I.S.N.
PFISTER Jean-Claude (détaché)	Physique du solide
PIBOULE Michel	Géologie
PIERRE Jean-Louis	Chimie organique
RAYNAUD Hervé	Mathématiques appliquées
ROBERT Gilles	Mathématiques pures
ROBERT Jean-Bernard	Chimie physique
ROSSI André	Physiologie végétale
SAKAROVITCH Michel	Mathématiques appliquées
SARROT REYNAUD Jean	Géologie
SAXOD Raymond	Biologie animale
SOUTIF Jeanne	Physique
SCHOOL Pierre-Claude	Mathématiques appliquées
STUTZ Pierre	Mécanique
SUBRA Robert	Chimie
VIDAL Michel	Chimie organique
VIVIAN Robert	Géographie

à Brigitte

Je tiens à remercier:

Sacha Krakowiak, Professeur à l'Université de Grenoble, de m'avoir accueilli dans son équipe et de l'honneur qu'il m'a fait en acceptant la présidence du jury de cette thèse.

Jacques Mossière, Directeur du Laboratoire de Génie Informatique, d'avoir dirigé la conduite de ma thèse et d'avoir su trouver le temps nécessaire aux relectures détaillées. Cette thèse lui doit beaucoup tant sur la forme que sur le fond.

Francis Martinez, Responsable de l'Equipe Communication Graphique de l'E.N.S.I.M.A.G, d'avoir accepté de participer à ce jury. Je considère, pour ma part, que ses travaux sur les systèmes graphiques interactifs ont eu une forte influence sur la définition du Compositeur.

Vania Joloboff, Ingénieur de Recherche au Centre de Recherche Bull de Grenoble, pour l'intérêt porté à ce travail et le temps consacré à la lecture et à la critique du texte initial.

Les membres de l'équipe Adèle qui tous, de par leurs idées, leurs critiques et leurs conseils ont contribués à l'élaboration de ce travail.

Joelle Coutaz pour avoir initialisé cet axe de recherches et d'y avoir guidé mes premiers pas.

Daniel Iglésias et son équipe de reprographie, pour le soin apporté à la présentation matérielle de cette thèse.

Qu'il me soit permis d'associer à ces remerciements Roland Balter et toute l'équipe du Centre de Recherche Bull de Grenoble pour l'intérêt porté à ce travail et la possibilité qu'ils m'ont offerte de le poursuivre dans le cadre du projet TIGRE.

TABLE DES MATIERES

-ooo-

Introduction

1. Environnement de l'étude

L'étude présentée dans cette thèse s'inscrit dans le cadre du projet Adèle (Atelier de DEveloppement de LogiciEl). (1) Cet atelier a pour objectif de définir un environnement de programmation qui facilite la construction et la mise au point de programmes Pascal.

Ses principaux constituants sont:

- un système d'archivage <Ghoul 83> qui assure la gestion des versions multiples d'un logiciel;

- un éditeur syntaxique <Rouzaud 84> et un interpréteur <Lenne 84> qui réalisent le cycle d'écriture mise au point d'un programme;

- un générateur de code multi-cible <Santana 83>; (2)

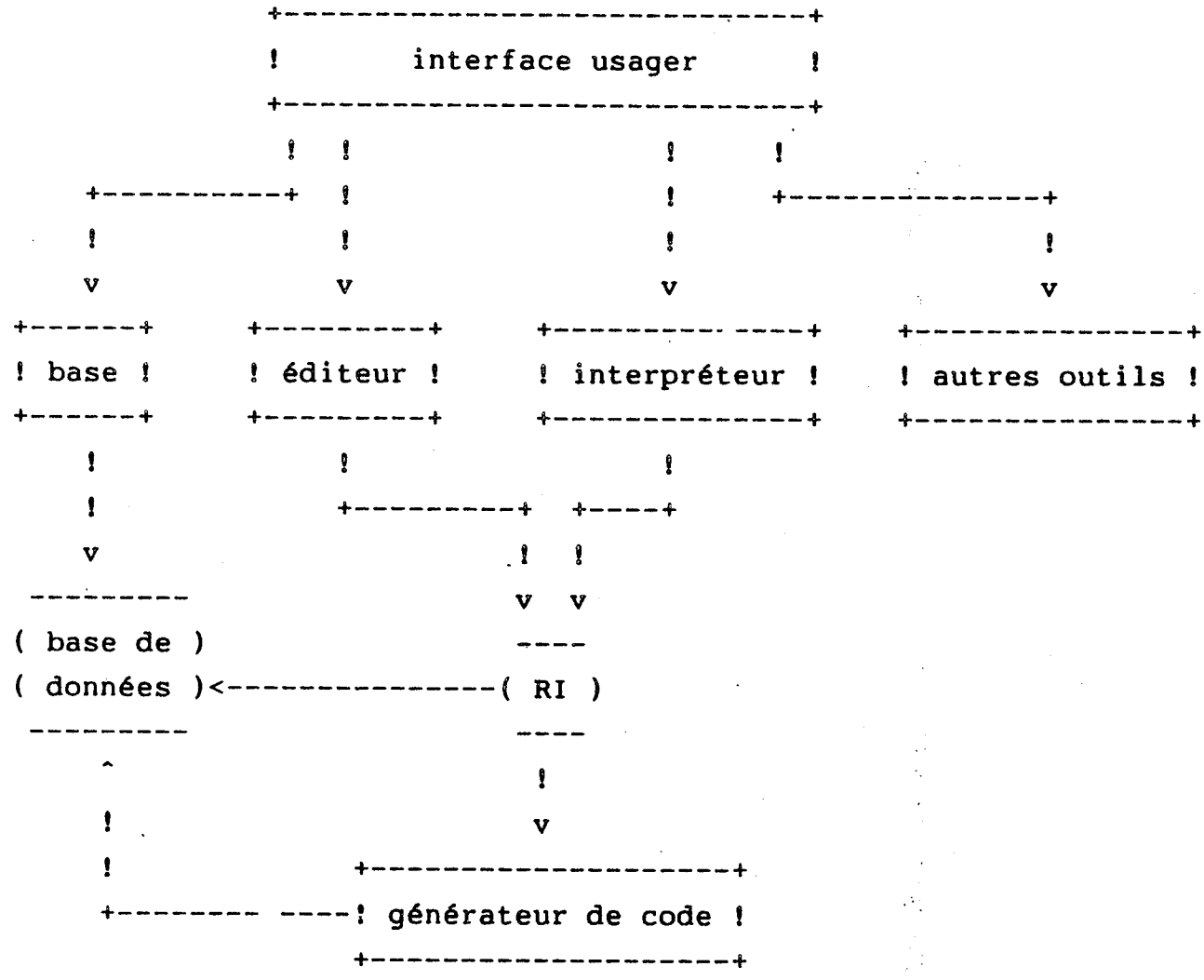
---

(1) Ce travail a été partiellement financé, d'une part, par l'ADI et d'autre part, par le CNET Lannion dans le cadre du projet CONCERTO.

(2) L'éditeur syntaxique, l'interpréteur et le générateur de

- une interface usager qui assure la médiation entre les différents outils précédemment cités et l'usager.

L'atelier Adèle est le résultat de l'intégration de l'ensemble de ces composantes. Son architecture globale est la suivante:



---

code opèrent sur une représentation unique des programmes qui est un arbre abstrait.

2. Motivations et objectifs

L'atelier Adèle est un ensemble intégré d'outils interactifs de manipulation de programmes. Tous ces outils communiquent avec l'usager et doivent par conséquent posséder certaines qualités ergonomiques.

La solution qui consiste à réaliser une interface usager particulière à chaque outil aboutit nécessairement à multiplier les efforts sans pour autant garantir une cohérence globale d'utilisation au niveau de l'atelier.

L'approche que nous avons retenue consiste à définir un module d'interface unique, utilisé par l'ensemble des outils de l'atelier pour communiquer avec l'usager.

L'objectif de cette interface usager <RR 299> est de:

- définir une interface cohérente et souple utilisable par toutes les applications du poste de travail;

- proposer des modes de dialogues adaptés et adaptables par l'usager;

- permettre la vision simultanée de différentes parties d'un programme et d'assurer la cohérence entre toutes les occurences visibles;

- donner à l'usager la possibilité de passer aisément d'une activité à une autre et plus généralement lui donner le contrôle prioritaire du déroulement parallèle de ces activités.

De l'analyse de ce "cahier des charges" on peut extraire trois grandes classes de fonctions:

- la gestion du dialogue qui a pour finalité d'acquérir les commandes émises par l'usager. Le Médiateur remplit cette

fonction;

- la visualisation et la gestion des informations visibles, tâches qui incombent au Compositeur;

- le partage de l'écran entre les différents outils de l'atelier et le changement d'activités sont assurés par le gestionnaire de fenêtres.

Ce que nous appelons interface usager est un logiciel qui possède un rôle de médiation entre une application et un usager <Pilote 83>.

Toutes les composantes logicielles qui font habituellement partie intégrante des applications et qui interviennent dans la relation homme-machine sont regroupées dans l'interface usager <Hayes 79, Ball 82a, Olsen 83b, Shaw 83>; ceci couvre l'acquisition des informations, la visualisation, etc.

Le but de ce regroupement est de faciliter et de réduire les coûts de conception et de réalisation des applications interactives en factorisant les composantes qui leurs sont communes.

L'interface usager est une entité indépendante qui ne peut donc contenir aucune information spécifique à une application. Pour prendre en compte les particularités de chacune d'elles (commandes, forme des informations visualisées, etc), le moyen généralement retenu est d'opérer à partir d'une description de l'application <Ball 80>.

Les ergonomes définissent l'interface usager d'une application comme étant la perception que possède l'usager de l'application qu'il utilise. Cette définition englobe tous les éléments avec lesquels l'usager est en contact <Moran 81>; à savoir les concepts autour desquels l'application est définie, la communication usager-application et les périphériques employés.

Nous n'intégrons dans l'interface usager que les aspects communication et matériel; le terme "interface usager" est donc un abus de langage. L'emploi "d'interface de communication" serait plus judicieux et permettrait de lever l'ambiguité.

3. Plan de la thèse

Dans le chapitre 2 nous allons préciser les objectifs, les fonctions et l'architecture d'une interface usager. Pour ce faire, nous commencerons par mettre en évidence les problèmes liés à la conception d'une application interactive qui tienne compte de certains des critères ergonomiques actuels; nous nous placerons dans le cadre d'une réalisation traditionnelle où le concepteur est aussi amené à définir l'interface de communication.

Puis, nous expliciterons les difficultés que rencontrent les usagers lors de l'utilisation d'une application interactive. (1)

Les aspects liés à la réalisation des applications interactives et à l'ergonomie du logiciel seront suivis d'un survol de l'état actuel de l'art dans les différentes techniques nécessaires à la réalisation d'une interface usager. Nous terminerons ce chapitre par une proposition d'architecture.

La présentation générale de l'architecture de l'interface usager d'Adèle fera l'objet du chapitre 3. Nous détaillerons les caractéristiques du Médiateur, du Compositeur et du gestionnaire de fenêtres. Nous nous attacherons à mettre en évidence les relations qui existent entre ces différents composants.

---

(1) Nous présentons dans ces deux parties certaines des conclusions parues dans l'abondante littérature consacrée à l'ergonomie des systèmes informatiques. Nous espérons vous livrer ces conclusions sans avoir déformé la pensée des auteurs cités, nos compétences en ce domaine étant limitées.

Le chapitre 4 fera l'objet d'une étude plus détaillée des techniques et mécanismes mis en oeuvre dans le Compositeur pour permettre la gestion des information visibles.

Nous terminerons (chapitre 5) sur les problèmes rencontrés, les apports et perspectives de l'interface que nous présentons.

L'interface usager

1. Application interactive

Une application interactive est un programme qui nécessite une intervention humaine pour pouvoir s'exécuter. Une telle application modélise une activité de nature incrémentale pour laquelle on ne peut, ou on ne veut, constituer a priori la succession des actions élémentaires (c'est à dire un programme) qui permette d'obtenir le résultat escompté.

L'homme dirige l'exécution de ce type de programme en émettant des commandes. Toute commande active l'exécution de la fonction qui lui est associée. Cette commande fournit un résultat au vu duquel l'usager réagit de manière à se rapprocher du but qu'il s'est assigné.

Les média de cette communication sont traditionnellement l'écran et le clavier; viennent s'y adjoindre des périphériques permettant la désignation directe des informations affichées sur l'écran.

L'application modélise:

- les objets que les personnes emploient dans leur tâche. Les objets sont définis par leurs propriétés, leur structure et les relations qui peuvent exister entre eux. Cette modélisation des objets permet d'obtenir une représentation informatique que nous appelons REPRESENTATION INTERNE ou RI. La RI est un ensemble choisi de données issues du monde réel; de cet ensemble considéré comme significatif du problème à traiter peuvent être dérivés les résultats escomptés <Wirth 76>;

- les actions des usagers sur les objets en définissant les opérateurs qui s'appliquent sur la RI. Ces opérateurs définissent l'ensemble des actions élémentaires qu'un usager peut employer pour manipuler la représentation électronique des objets. Un opérateur est défini par ses paramètres d'entrée, ses résultats et messages d'erreurs éventuels; ainsi que par la relation qui existe entre les paramètres et les résultats (ce qui constitue le traitement).

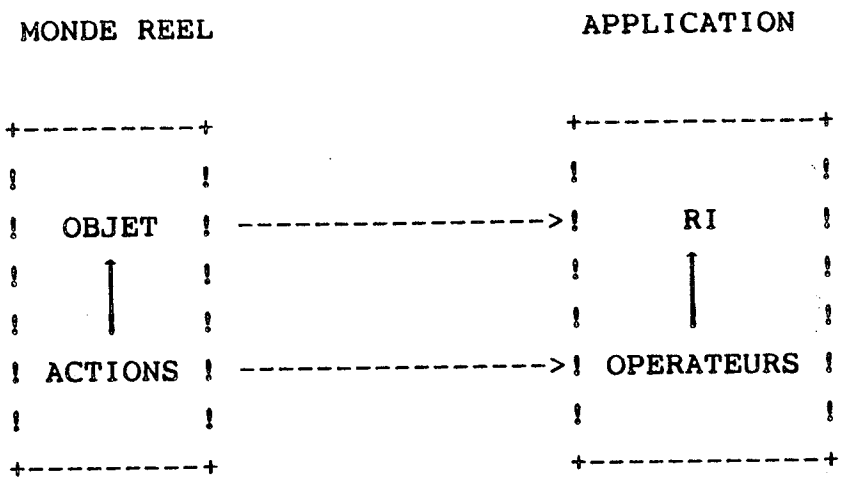
REMARQUE:

Nous savons que le temps de réponse influe sur le comportement de l'usager <Shneiderman 79>. Le fait d'en tenir compte a des incidences sur le choix de la représentation des objets et des techniques algorithmiques employées pour implanter les opérateurs, et donc sur la définition du modèle.

Ainsi, dans le projet Adèle <RR 299> un programme est modélisé par un arbre. Le temps de réponse minimum et la détection immédiate des erreurs sont obtenus en utilisant des techniques d'analyse syntaxique et sémantique incrémentales <Rouzaud 84, Notkin 84>.

La modélisation inclut de fait la modélisation du comportement des individus. Les concepteurs se définissent en quelque sorte un usager type qui possède un comportement moyen, représentatif de l'ensemble des utilisateurs potentiels. Ils supposent que les futurs usagers sont conformes à l'usager type: pour réaliser leur tâche ils n'auront besoin que des opérateurs définis dans le modèle; leur vision de l'objet et de ses propriétés est celle définie par la spécification de l'application.

La relation "monde réel"-application peut être schématisée comme suit:



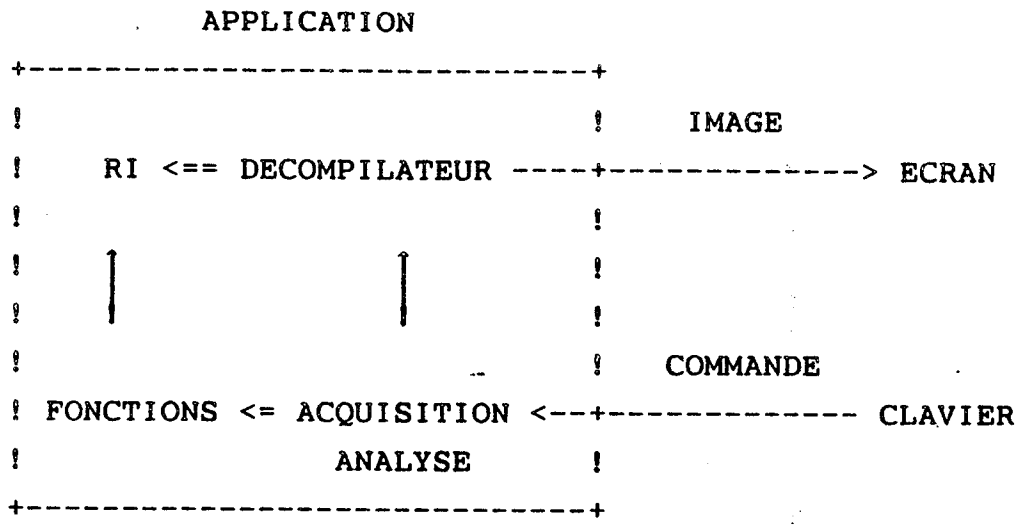
L'usager dirige l'application en lui soumettant des commandes, il est alors nécessaire d'établir le lien qui existe entre des objets et les opérateurs du modèle et les moyens qui permettront à l'usager de les désigner.

Ceci consiste à préciser comment l'usager va dialoguer avec l'application et à définir les mécanismes nécessaires à ce dialogue. A cette fin on intègre généralement dans l'application des fonctions d'acquisition et d'analyse des commandes.

La communication entre l'usager et l'application utilise un support de visualisation qui matérialise le résultat des actions qui sont entreprises. Il faut alors définir l'organisation et la nature des informations qui sont visualisées et préciser la forme sous laquelle la RI des objets se matérialise sur l'écran.

A cette fin, on définit dans l'application des fonctions qui visualisent et qui gèrent les informations visibles produites à partir de la RI. Elles peuvent fournir à l'usager un certain nombre de facilités qui lui permettent d'influer sur l'organisation et la nature des informations visibles. Appelons l'ensemble de ces fonctions DECOMPILATEUR.

Nous obtenons alors un schéma qui décrit les composants essentiels de toute application interactive.



1.1. La désignation des commandes

Définir le dialogue entre l'usager et l'application revient à préciser les méthodes de communication utilisées pour manipuler les entités du modèle, et à établir comment ces entités sont présentées à l'usager.

Dans ce but, on associe aux opérateurs des symboles (noms ou icones) qui sont employés dans des phrases possédant une structure syntaxique.

En associant des symboles aux différents opérateurs de l'application, les concepteurs désirent transmettre un modèle implicite de celle-ci (ensemble de relations) en symbolisant les opérateurs de manière à refléter ce modèle <Rosenberg 82>. Un tel symbole doit:

- suggérer directement ce que fait l'opérateur associé au symbole,

- suggérer la relation qui existe entre cet opérateur et l'ensemble des opérateurs de l'application: similaire, opposé, sans relation.

Les concepteurs sont alors amenés à choisir le type des symboles qui représentent les opérateurs: soit les icônes, soit les commandes nommées.

1.1.1. Les icones

S'ils choisissent les icônes, ils ont de fait opté pour une technique de communication unique et particulièrement adaptée aux usagers débutants ou occasionnels: le menu.

Les icones présentent un double avantage:

- on les reconnait facilement <Lodding 83>;

- les graphismes véhiculent une quantité d'information plus importante que le texte seul <Hemenway 82>.

Comme on ne sait pas définir avec exactitude si une image particulière véhicule bien le message spécifique que l'on désire lui associer <Lodding 83>, l'utilisation des icones présente certaines difficultés.

Une image est en effet facile à comprendre si elle ressemble à un objet connu. Dans le cas inverse, un texte explicatif et un contexte doivent lui être adjointes pour qu'elle puisse être comprise.

Il sera généralement nécessaire de réunir trois éléments:

- l'image

  elle représente un objet ou un concept que l'usager peut identifier soit parce qu'il en a préalablement pris connaissance, soit parce qu'elle lui est connue;

- la légende

elle permet d'expliciter le message véhiculé par l'image en fournissant les indications supplémentaires qui peuvent se révéler être nécessaires;

- le contexte

il est donné par l'environnement (l'état de l'application) nécessaire à l'interprétation de l'image et de la légende.

Les icones ne peuvent alors être considérées comme un choix de définition de l'interface de dialogue que si l'on se restreint à l'utilisation de l'image seule; nous venons de constater les difficultés qui en résultent pour l'usager.

La tentation de procéder de la sorte reste néanmoins grande compte tenu des difficultés de réaliser une interface qui propose plusieurs modes de communication et du désir d'exploiter au mieux les possibilités d'un terminal graphique.

1.1.2. Les commandes nommées

La seconde approche possible est d'employer des noms tirés du vocabulaire usuel. Les usagers se trouvent alors confrontés au problème d'apprendre, de comprendre, et de se souvenir du nouveau sens véhiculé par ces mots.

Lorsqu'on lit la littérature consacrée à ce sujet, il apparait que le choix des termes est important <Barnard 82, Black 82, Scapin 82, Rosenberg 82>. Ces études montrent en effet que le choix des noms influe sur la facilité d'utilisation et d'apprentissage des commandes.

Il apparait ainsi que l'on a à choisir entre les termes génériques et les termes spécifiques. Les noms génériques possèdent une sémantique qui leur permet de couvrir un large domaine d'utilisation. Ils peuvent alors être employés dans des applications complexes dans lesquelles des opérateurs similaires sont définis sur des objets distincts. Ils permettent de regrouper des commandes; regroupement dont l'importance est montrée dans <Scapin 82>.

Les termes spécifiques possèdent quant à eux une sémantique très précise. Celle-ci doit être explicitée pour chaque application dans laquelle ce terme est employé.

En tentant de déterminer si le vocabulaire défini par les usagers était "meilleur" que celui défini par les concepteurs, une première étude <Black 82> montre les difficultés qu'éprouvent les usagers à nommer un jeu de commandes.

On constate en outre que les termes retenus par les usagers tendent à être des mots dont l'emploi est fréquent dans le langage courant. De plus, ces termes sont très (pour ne pas dire trop) généraux. Dans cette étude, les termes peu fréquents sont décrits comme étant les plus "performants" (apprentissage, mémoire et utilisation).

Les résultats parus dans <Barnard 82> montrent que le temps global mis par des néophytes pour résoudre un ensemble d'exercices est plus important (bien que l'écart ne soit pas statistiquement significatif) lorsque des personnes emploient des noms génériques (c'est à dire un jeu de commandes pour lequel il existe une règle de structuration) que lorsqu'ils utilisent des noms spécifiques et que cet écart diminue lorsque les utilisateurs gagnent en expérience.

Cette étude montre par ailleurs que les stratégies d'apprentissage sont distinctes: ceux qui emploient les commandes génériques ont besoin de plus d'informations décrivant les

commandes, les consultent plus souvent, et ont recours plus rapidement à une aide disponible sur le site en cas de difficulté que ceux qui utilisent les commandes spécifiques.

Ces derniers mettent par contre davantage de temps à émettre une commande. Ils hésitent à consulter une documentation en cas d'incertitude; ils prennent aussi un temps de décision plus élevé après une telle consultation.

Enfin, il s'avère que les usagers se souviennent plus facilement de l'objectif d'une commande spécifique que de celui d'une commande générique.

Indépendamment des fréquences d'apparition des noms de commandes dans le langage usuel, de leur spécificité ou de leur généralité, c'est l'aspect discriminant des termes utilisés pour définir l'ensemble des commandes d'une application qui conditionne la facilité d'utilisation de celle-ci <Barnard 82, Black 82, Rosenberg 82, Scapin 82>.

Cette constatation implique de fait l'exclusion, dans un même jeu de commandes, de deux termes aux significations proches dans le langage courant, même si leur sont associées deux fonctions similaires. C'est en fait l'existence d'opérateurs similaires qui est à proscrire.

Par contre, l'existence d'une certaine redondance dans le schéma de nomination des commandes par l'inclusion de termes complémentaires (paires opposées telles que créer-détruire) en facilite l'utilisation et l'apprentissage <Barnard 82>.

REMARQUE:

S'il est évident qu'il faille tenir compte de telles considérations lors de la conception d'une interface de dialogue, il nous parait cependant indispensable de prendre quelque recul

par rapport aux conclusions que nous pourrions être amenés à formuler si nous étions dans la situation d'un concepteur d'application.

Nous pensons en effet qu'une conclusion amenant à un choix d'alternatives ne peut s'effectuer in abstracto, en ne tenant compte que des performances induites par ces choix. Dans la conception d'une interface usager nous devons nous poser le problème de la cohérence inter-application, problème pour lequel une solution possible est l'utilisation de termes génériques.

Cette cohérence ne peut pas reposer sur des considérations d'ordre linguistiques et sur une discipline de réalisation: nous devons aussi penser à faciliter la construction des applications interactives.

Notre approche du problème nous amène a vouloir inclure des commandes réellement génériques dans l'interface usager et donc à définir une architecture adaptée à cet objectif.

1.2. Le dialogue

Les critères précédemment cités ayant guidés les concepteurs dans le choix des symboles, il reste à décider du mode de communication qui sera proposé aux usagers.

Cette décision sera prise en tenant compte du type des usagers que l'on désire privilégier. Aucune technique de dialogue n'est en effet parfaitement adaptée à la fois aux usagers novices et aux experts.

Le choix se fera en tenant compte du matériel disponible et en se liant éventuellement à ce matériel (présence d'une souris par exemple). Un autre critère du choix sera le rapport simplicité de réalisation-satisfaction du type d'usager privilégié.

Ce dernier est souvent l'usager débutant. Ceci est compréhensible compte tenu de l'un des arguments actuel de vente: la facilité d'utilisation (voir certaines publicités: "le système que l'on peut utiliser sans rien connaitre à l'informatique"). Les techniques les plus couramment employées sont donc celles du menu ou du formulaire.

Ces choix effectués, il faut définir la syntaxe abstraite des commandes et réaliser un mécanisme d'acquisition et d'analyse des commandes.

Ceci fait, il reste à décider de la manière dont l'écran va être employé pour le dialogue entre l'application et l'usager.

1.3. Visualisation des objets de l'application

La communication entre l'application et l'usager utilise un support de visualisation dont les concepteurs décident de l'utilisation. Ils définissent l'organisation et la nature des informations qui y seront visualisées, ainsi que les facilités qui sont offertes à l'usager.

L'affichage de ces informations les amène à préciser la forme sous laquelle ils visualisent la représentation interne des objets de l'application et à se donner les moyens nécessaires à la gestion de ces informations sur le support de visualisation.

Se pose alors le problème de l'utilisation des moyens de communication disponibles (écran, organe de désignation,...), sans pour autant se lier de manière définitive à un type particulier de terminal.

Ceci passe par la définition d'un terminal virtuel qui assure l'indépendance syntaxique face à une classe de terminaux <Lantz 79>.

La gestion de l'espace écran est nécessaire si l'on veut donner à l'usager une vision dite pleine page des objets qu'il manipule. Remarquons que la gestion de l'information visible est un problème plus complexe que celui de la visualisation.

Pour permettre la désignation directe des informations visibles, il faut assurer la liaison entre la représentation interne et la représentation visuelle de l'objet.

Pour résoudre ces problèmes, on associe à l'application un décompilateur dont la tâche est d'effectuer la transformation RI-représentation visuelle (RV). Le décompilateur opère directement sur la représentation interne pour générer un ensemble d'ordres d'affichage à destination d'un terminal virtuel.

Sont à sa charge la gestion de l'écran, l'exploitation des possibilités du terminal, et le maintien de la cohérence entre la RI et la RV. Pour permettre la désignation directe <Naffah 79>, il faut assurer la liaison entre la représentation visuelle et la représentation interne de l'objet; c'est à dire qu'il faut être capable d'effectuer la liaison entre les coordonnées d'un point désigné sur l'écran et un objet, ou une partie de l'objet de l'application.

Il est possible de donner à l'usager un certain nombre de facilités qui lui permettent par exemple:

- de voir simultanément plusieurs parties d'un même objet, éventuellement avec des niveaux de détails différents <Milkelson,81, Teitelbaum,81>,

- d'organiser les informations sur l'écran.

La conséquence de ceci est que l'application devient un programme très complexe dont la tâche essentielle est de dialoguer avec l'usager, nous en voulons pour preuve la part importante dédiée à la communication dans des logiciels de gestion

<Sutton 78>.

De par les difficultés et les coûts de réalisation de la partie de l'application qui gère la relation homme-machine, les concepteurs optent souvent pour des solutions simples qui leur permettent de "cibler" au mieux le type des usagers auquel le produit est destiné.

Pour l'usager final, ceci se traduit par des formes de dialogue rigides et pas toujours très agréables qui peuvent se révéler être inadéquates durant la phase d'apprentissage, ou après une période d'utilisation prolongée.

2. L'usager

Pour réaliser son travail, l'usager utilise un environnement qui lui donne accès à un ensemble d'outils. Chacun d'eux est une application au sens du chapitre précédent.

Dans cet environnement, il veut pouvoir effectuer plusieurs tâches éventuellement distinctes en suivant un mode opératoire qui lui est propre et aussi proche que possible de son environnememt de travail habituel. Dans ce dernier, il possédait un contrôle (presque) absolu sur l'organisation de son travail, et ce contrôle lui est nécessaire lorsqu'il utilise un système informatique <Shneiderman 79>.

Nous avons vu qu'une application ne permettait de restituer que les caractéristiques essentielles des objets et des opérateurs tels qu'ils sont perçus par un ensemble de personnes. Elle ne constitue donc pas un modèle parfaitement fidèle d'une activité telle qu'un individu peut la réaliser.

L'application définit pour l'usager une vision nouvelle de l'activité. Les moyens matériels ne sont pas identiques aux précédents, les objets ne se matérialisent plus de la même

manière, ne possèdent plus tout à fait les même propriétés, et leur manipulation ne suit plus le mode opératoire traditionnel.

L'usager doit "apprendre" à "collaborer" avec l'application. Il doit se plier aux contraintes qu'elle lui impose. Sa première tâche est de comprendre le modèle défini par l'application et d'apprendre à l'utiliser et à communiquer avec elle.

De plus, les besoins de l'usager évoluent au cours du temps. Ainsi, l'usager qui débute dans l'utilisation d'une application désire que celle-ci soit "facile à apprendre", c'est à dire qu'il puisse se familiariser au plus vite avec son utilisation. Il nous faut cependant remarquer qu'il n'est en fait intéressé que par les aspects qui lui seront utiles, et ne souhaite nullement devoir en posséder une connaissance approfondie.

Son désir est de pouvoir utiliser rapidement la machine, sans avoir à consulter une abondante documentation technique.

Une fois que l'usager s'est familiarisé avec son application, qu'il a pris confiance en ses possibilités, il veut pouvoir réduire la quantité d'informations qu'il doit soumettre à l'application pour effectuer sa tâche. Cet élément est une manifestation d'une tendance plus générale de la part de l'usager qui, en évoluant, porte un regard plus critique sur les conditions de travail qui lui sont imposées par la machine.

Il veut un "système facile à utiliser".

Dominant parfaitement certains aspects de la machine, il désirera très probablement, par nécessité ou par curiosité, connaitre ceux des aspects qui lui sont encore inconnus et qui pourraient lui faciliter la tâche. Il devient alors plus exigeant quant aux services rendus, et cherche à personnaliser l'application afin de l'adapter à son mode de travail.

Il désire alors un système "qui s'adapte facilement à ses besoins".

2.1. Le contrôle

Dans son environnement de travail habituel, l'usager entreprend généralement plusieurs activités dans une même journée et l'ordre dans lequel elles se succèdent est déterminé par des contraintes provenant de l'environnement (contraintes exprimées par la hiérarchie, coup de téléphone obligeant notre personnage à changer d'activité, etc), par un ordonnancement lié aux différentes activités (telle tâche ne peut être effectuée avant telle autre parce qu'elle en utilise les résultats), soit par une organisation des tâches qui est propre à l'individu.

Il organise son bureau de manière à pouvoir accéder facilement à ce qui matérialise ses activités et le mode d'organisation est dépendant du but recherché <Malone 82>.

Pour restituer à l'usager le contrôle qu'il possède dans un environnement non informatisé, les propositions actuelles <Meyrowitz 81, Lantz 79> définissent les notions d'activité, de gestionnaire de fenêtres, et de fenêtre.

Une activité correspond à l'utilisation d'une application par un usager. Toute activité se matérialise sur l'écran par une fenêtre: portion rectangulaire de la surface de l'écran qui est le lieu de visualisation et de dialogue avec l'application.

Sur l'ensemble des fenêtres présentes sur l'écran est définie la notion de fenêtre courante. Elle représente le centre d'intérêt de l'usager et toutes les informations émises par celui-ci au clavier sont dirigées vers cette fenêtre, et donc vers l'application correspondante.

Dans ce modèle, le changement d'activité correspond à un changement de fenêtre. Celui-ci peut être exprimé soit à l'aide de commandes émises au clavier, soit par l'utilisation d'un organe de désignation à l'aide duquel on déplace le réticule qui lui est asservi vers la fenêtre que l'on désire sélectionner. Celle-ci devient alors la nouvelle fenêtre courante.

Conformément aux possibilités d'organisation des dossiers sur sa table de travail, l'usager peut décider de l'organisation des fenêtres sur l'écran, de leur taille et de leur disposition. Le gestionnaire de fenêtres lui fournit les commandes qui permettent ces manipulations.

Remarquons qu'il est possible d'utiliser le principe des fenêtres pour représenter:

- l'ensemble des tâches réalisées par l'usager à un instant donné,

- les différentes vues sur un objet dans une même application.

EXEMPLE

Dans une activité de type édition de programme, il est intéressant d'avoir sous les yeux la partie déclaration des types, les déclarations de données et la procédure que l'on écrit. C'est de cette manière que travaille habituellement un programmeur sur son bureau: il découpe les différentes parties de son listing.

Il serait donc souhaitable de permettre ce type d'organisation dans une application "édition de programme".

Si le mécanisme de base (la fenêtre) employé est le même dans ces deux cas, la fonction remplie est cependant différente. Dans le dernier cas, plusieurs fenêtres peuvent représenter une même application, et l'association fenêtre-application est moins

schématique que la description que nous en avions donnée.

Remarquons que dans la majorité des systèmes, les fonctions de contrôle ne sont disponibles qu'au niveau des commandes système où il peut:

- lancer une activité représentée par un processus auquel il donne un objet (fichier) en paramètre. L'usager entre alors dans un mode de travail dans lequel les seules commandes reconnues sont celles de l'application.

  La seule possibilité qui lui reste est d'arrêter l'exécution de l'application. Cette fonction est souvent représentée par le symbole "Break". Le sens associé à break peut être défini par: quel que soit le traitement en cours, rendre la main à l'usager au niveau des commandes système.

  Ceci entraine soit la perte de la totalité du travail déjà effectué, soit l'empilement du contexte du processeur et le retour au système (cf Multics). L'usager peut, dans ce dernier cas, gérer ses activités en pile. Cette possibilité n'est cependant pas adaptée au schéma de travail que nous avons décrit.

- l'arrêt d'une activité est exprimée à l'aide de la commande "FIN" de l'application. Celle-ci est la seule qui permette de rendre la main au système alors que l'application est dans un état connu.

Le problème fondamental qui se pose à l'usager est que les commandes système ne lui sont pas accessible à tout instant, et que la majorité des systèmes n'ont pas été conçus de manière à ce qu'une même personne puisse contrôler simultanément plusieurs activités.

Le gestionnaire de fenêtres apporte des solutions à ce type de problème.

2.2. L'apprentissage

L'apprentissage a pour but de faire connaitre à l'usager les concepts définis par une application, ainsi que les manipulations qu'il peut effectuer à l'aide de celle-ci. Les mécanismes utilisés à cette fin sont la mémorisation et la compréhension.

Traditionnellement, la phase d'apprentissage débute par la compilation d'une documentation quelquefois volumineuse, explicitant le sujet. Puis le futur usager passe par une phase d'essais durant laquelle il exploite les connaissances nouvellement acquises afin de vérifier si elles ont été bien assimilées.

Dans les faits, ceci se traduit bien souvent par l'émission d'une commande ayant pour résultat l'affichage d'un message d'erreur plus ou moins explicite. L'usager est alors amené à consulter sa documentation pour déterminer la nature et la cause précise de l'erreur.

Or, le premier contact d'un usager avec une application possède une grande importance <Giboin 83>. Deux des critères concernant l'ergonomie des systèmes informatiques sont la facilité et la rapidité d'apprentissage <Giboin 83, Turoff 82>.

L'objectif est donc d'amener l'usager à utiliser rapidement une application sans avoir à:

- connaitre toute l'application;

- accéder à une documentation (ensemble de brochures);

- prendre en compte des éléments non significatifs relativement au but recherché.

L'application doit alors de se "présenter" et faire connaitre à l'usager les possibilités qui lui sont offertes. Cette présentation ne saurait être une visualisation unique et exhaustive de toutes les fonctions de l'application et de leurs modes d'utilisation, mais une présentation progressive de celles nécessaires à l'usager, donc de celles qu'il utilise effectivement <Turoff 82>.

A cette fin, deux approches complémentaires peuvent être envisagées:

- les techniques d'E.A.O décrivant l'emploi de l'application à l'aide de textes explicatifs et d'exemples auxquels on peut associer un ensemble d'exercices;

- l'assistance à l'usager pendant que celui-ci utilise le nouvel outil <Sidner 82>.

Pour cette dernière approche, la mémorisation peut être facilitée par la visualisation répétée d'une quantité volontairement limitée d'informations en des moments où leur connaissance est vraiment nécessaire; c'est à dire, lorsqu'un choix s'impose, lorsqu'une erreur est commise, ou sur demande de l'usager.

Dans cette optique, la technique du "menu" associée à une documentation sur le site <Savage 82> semble bien adaptée. Son principe est d'afficher les noms de commandes ou d'objets (ou bien les symboles qui les représentent). L'usager doit alors sélectionner l'un de ces symboles à l'aide d'un organe de désignation (touche, souris,..). Ce processus se réitère pour déterminer les options éventuelles, les paramètres, etc.

Le menu a pour avantages de porter à la connaissance de l'usager les commandes disponibles (il doit les reconnaitre, non pas les mémoriser), de limiter à chaque étape les choix que celui-ci peut effectuer (menus hiérarchisés), et lui interdire ainsi toute erreur syntaxique.

Ce type de dialogue, assez rigide semble bien adapté aux novices (<Savage 82> discussion dans <Benbassat 81>). Dans cette approche, la mémorisation des informations utiles se fait par répétition.

L'apprentissage ne se limitant pas à un effort de mémorisation, il est aussi nécessaire que l'individu puisse aisément comprendre l'application.

REMARQUES:

Le mécanisme de la compréhension semble être dominé par le raisonnement par analogie <Carbonell 81>. Certains préconisent l'emploi d'une métaphore pour définir une application (principe du Star qui exploite une métaphore de bureau). La métaphore est en effet le résultat dans le langage de ce type de raisonnement <Carbonell 81>.

Giboin et Halasz <Giboin 83, Halasz 82> font cependant remarquer qu'il est préférable, pour le raisonnement, de définir des modèles conceptuels abstraits et réservent les analogies et métaphores pour l'apprentissage de ces modèles.

Bien que l'on ne sache définir ce qu'est un bon modèle <Fischer 82>, on s'accorde cependant à dire qu'une application devrait être aisément compréhensible et utilisable si les modèles conceptuels de l'usager et du concepteur sont proches <Moran 81, Liebermann 82>.

Dans l'approche que nous décrivons, l'expérimentation joue un rôle clef: elle permet à la fois d'acquérir les connaissances nécessaires et de vérifier leur compréhension tout en étant guidé

dans l'utilisation de l'application. Dans cette optique, soulignons le rôle joué par le support de visualisation; celui-ci est triple:

- il a un rôle de mémoire cache prenant le relais de la mémoire à court terme <Shneiderman 79>, qui évite à l'usager de mémoriser toutes les informations décrivant l'action en cours;

- Il a un rôle d'aide mémoire permettant de rappeler à l'usager les actions qui restent à entreprendre pour atteindre son but;

- Il est enfin le reflet de l'état courant de l'application: toute action de l'application doit se traduire par une modification du support de visualisation. L'usager effectue instinctivement l'analogie entre ce qu'il voit et ce qui s'est passé, les caractéristiques des objets manipulés sont assimilées à celles qui sont visibles, la finalité des actions étant déduites de leurs effets visuels sur l'objet <Smith 82>.

Toute expérimentation amène l'individu à commettre des erreurs. Comme cette expérimentation représente pour l'usager un moyen d'apprendre à connaitre une application, il est important que les conséquences d'une erreur soient aussi minimes que possible. L'usager doit , par exemple, pouvoir revenir à tout instant à un stade antérieur <Giboin 83, Good 81>. La stratégie générale qui est définie consiste à ne pas pénaliser l'erreur.

Un accent particulier doit être mis sur l'aspect informatif du message d'erreur. Celui-ci doit permettre à l'usager de répondre à trois questions:

- Où ? (localisation précise de l'erreur)

- Pourquoi ? (cause exacte de l'erreur)

- Comment ? (moyens de rectification et de récupération)

Dans le cas où un message ne semble pas suffisamment explicite à l'usager, celui-ci doit pouvoir utiliser la documentation disponible en ligne. Celle-ci doit intégrer une aide syntaxique définissant la syntaxe des commandes et des paramètres, ainsi qu'une aide "sémantique" définissant les objectifs des commandes et les propriétés des paramètres.

Par l'intermédiaire de cette phase d'apprentissage, l'usager évolue vers une utilisation de moins en moins assistée par le système. Ayant pris une confiance suffisante en ses possibilités propres, il aura tendance à rejeter un mode de dialogue rigide, et désirera un contrôle plus important sur la machine <Shneiderman 79>; le système devra donc être apte à fournir une réponse à cette modification du comportement de l'usager.

2.3. L'utilisation

Dans cette partie, nous désirons mettre en évidence certains des éléments qui influent sur la facilité d'utilisation d'une application et, les méthodes proposées par les ergonomes pour atteindre ce but. (1)

Nous aborderons alors deux aspects de la relation homme machine, à savoir la communication et le traitement des erreurs.

---

(1) Les concepts définis par l'application influent fortement sur la facilité d'utilisation <Moran 81, Liebermann 82>, et vouloir résoudre tous les problèmes par la définition d'un interface de communication même magistralement imaginée serait illusoire <Reenskang 81>.

Ne sachant définir ce qu'est un bon modèle (voir la remarque précédente), nous nous contentons de souligner et de constater son influence.

2.3.1. La communication

Pour un individu donné, communiquer facilement c'est pouvoir utiliser un moyen de communication qui lui est adapté. Dans le chapitre précédent, nous avons préconisé l'utilisation du menu comme moyen de communication privilégié entre un novice et une application; or cette technique présente plusieurs inconvénients pour un usager évolué:

- la lenteur d'utilisation: la cadence de sélection des options est d'une parmi dix toutes les cinq secondes. Ceci est à rapprocher avec la cadence normale de frappe qui est d'un mot par seconde;

- L'apparition des options et du texte explicatif éventuellement associé peut être ressentie comme une gêne par les personnes n'ayant plus à être assistées;

- L'obligation qui est faite d'utiliser un mode de dialogue peu efficace, et très rigide.

D'autres formes de dialogue existent: le formulaire, les langages de commandes, les abréviations et touches synonymes, la langue naturelle; chacune de ces formes ayant ses partisans et détracteurs...

Le formulaire <Hayes 83, Hayes 84> permet de définir un type de dialogue moins rigide et moins prolixe que celui du menu. Un formulaire informatique utilise le même principe qu'un formulaire administratif, une fiche d'état civil par exemple.

Celle-ci est contituée d'un ensemble de rubriques, chacune d'elles possédant un en-tête: petit texte explicatif précisant la nature de l'information que la personne doit apporter pour la compléter.

Nom: DURAND
Prenom: Jean
Age: 29

Remarquons que l'ordre dans lequel ces rubriques sont complétées est indifférent et qu'une rubrique déjà complétée peut être modifiée pour peu que l'on dispose d'un moyen d'effacement. Cette modification n'a aucune incidence sur les autres rubriques déjà complétées (ceci n'est pas le cas pour le menu, où l'ordre est figé, et où un retour arrière "supprime" les informations apportées).

Pour utiliser cette forme de communication, il est nécessaire que l'usager connaisse la réponse correcte à apporter à chaque rubrique, ainsi que son format d'entrée. Une aide pourra être demandée par l'usager en entrant dans la rubrique concernée un "?" par exemple.

Le langage de commande représente la forme de communication la plus classique, et sans doute la plus utilisée. Contrairement aux techniques précédentes, celle-ci n'apporte aucune assistance à l'usager. Celui-ci doit connaitre les noms des commandes et des objets de l'application, ainsi que la syntaxe à respecter pour pouvoir soumettre des commandes. L'avantage de cette technique par rapport aux précédentes est sa concision, et donc la rapidité du dialogue.

Les abréviations et touches synonymes sont les résultats de la recherche de formes de dialogue très concises dont les usagers dits confirmés sont friands. Les termes employés étant rarement explicites, quelques caractères au plus (cf Emacs), ils sont difficiles à mémoriser.

L'une des caractéristiques des modes de communication cités jusqu'ici est leur forte formalisation: seuls un certain nombre de termes sont reconnus, toute phrase acceptée par le système possède une syntaxe rigide, tout terme possède une signification bien

précise. Il ne peut (ou du moins, ne doit) exister aucune ambiguité.

Le but des interfaces en langue naturelle est de supprimer ces inconvénients; la rigueur disparait donc, et l'usager peut exprimer ses requêtes en ses propres termes, éventuellement sans connaitre ni les commandes ni la syntaxe. Le système doit être à même déliminer les ambiguités et de déterminer ce que l'usager a l'intention de faire <St Dizier 83>. Quelques remarques sont cependant formulées dans <St Dizier 83>:

- La formulation d'une requête est lourde, et les temps de réponse sont actuellement "élevés", de l'ordre de cinq secondes.

- Il est nécessaire de paraphraser les requêtes afin d'obtenir une confirmation sur les intentions de l'usager. Les reformulations sont nécessaires pour lever les ambiguités.

- Le manque de concision pour cette forme de dialogue. L'entrée vocale permettrait de lever cette dernière remarque.

A l'heure actuelle, le coût d'utilisation d'un système de dialogue en langue naturelle, et l'état de la technique en limite l'emploi. (1)

Une autre forme de communication possible est la manipulation des informations visibles dont l'un des aspects est la désignation directe.

La désignation directe permet de donner à l'usager la possibilité d'expliciter au "système" l'objet qui constitue son

---

(1) Il semble cependant que le traitement automatique de la langue naturelle puisse devenir assez rapidement un outil réellement exploitable dans le cadre de la communication homme-machine <St Dizier 83>.

centre d'intérêt en le désignant à l'aide de l'interface matérielle appropriée. Pour ce faire, il opère directement sur la représentation de l'objet, matérialisée par une image sur un support de visualisation, en déplaçant sur ce support un réticule asservi aux déplacements appliqués par l'usager à cette interface.

Grâce à ces déplacements, l'usager positionne le réticule sur l'objet et émet un signal qui est interprété par le système comme étant la désignation de l'objet sur lequel il désire opérer. Aucune direction de déplacement n'étant privilégiée, tout se passe pour l'usager comme s'il n'opérait que sur l'image qu'il a devant les yeux. La structure de l'objet telle qu'elle est modélisée dans l'application lui reste cachée.

Ceci est à rapprocher des éditeurs pour lesquels les déplacements peuvent être qualifiés de structurels: les déplacements autorisés tiennent compte de la structure de donnée sous-jacente, sur laquelle on plaque les notions d'élément courant, d'éléments précédent et suivant.

L'usager dispose ainsi de commandes telles que: mot suivant, phrase suivante, paragraphe suivant ou sortir d'un mode de déplacement "phrase" pour entrer dans le mode de déplacement "paragraphe". Passer d'un élément courant à un autre, revient à se déplacer à l'intérieur d'une structure de donnée qui est explicitement connue.

Le mécanisme de la désignation directe permet de se déplacer de manière simple et rapide vers un élément en positionnant directement le réticule sur une portion de l'image correspondante. On est alors à même de déterminer quel est le plus petit élément de l'objet qui est désigné.

Le problème qui se pose est de savoir ce qui est désigné: est-ce l'élément atomique, ou est-ce l'élément englobant dont l'élément désigné n'est qu'une composante.

Ainsi, je peux pointer mon doigt vers le combiné d'un téléphone pour désigner le téléphone dans sa totalité. Dans la vie courante, une telle ambiguité est levée par le contexte défini lors d'une discussion préalable, ou par un dialogue durant lequel on précise la portée de la désignation.

Dans un système, des mécanismes analogues devront être mis en place pour lever toute l'ambiguité contenue dans une désignation. La portée d'une désignation telle qu'elle a été interprétée par le système devra être explicitée à l'usager de façon à ce que ce dernier puisse, par un dialogue, préciser ce qu'il a voulu désigner.

Si l'intérêt de la désignation directe est actuellement reconnue, il n'en est pas de même pour les autres formes de manipulations directes généralement réservées aux seul éditeurs dits "pleine page". De tels mécanismes permettraient cependant d'offrir à l'usager de réelles facilités.

EXEMPLE

Prenons pour exemple un système de gestion de fichiers. Dans un tel système des fonctions de création, de destruction et de renommage de fichiers existent toujours. Ainsi, la fonction "renommer" <nom_initial> <nouveau_nom> peut être remplacée par l'édition du nom du fichier. Par une combinaison astucieuse de la désignation directe et de la modification des informations visualisées, un tel objectif peut être atteint.

De même, la destruction d'un fichier peut être exprimée comme la destruction d'un nom dans une liste de fichiers et, la création comme l'insertion d'un nouveau nom dans cette même liste.

Nous pouvons donc constater que des fonctions d'édition de texte permettent de remplacer ou d'exprimer de manière peut-être plus intuitive, des commandes traditionnellement incluses dans

l'application.

Une généralisation de cette approche, nous amène à vouloir manipuler directement la représentation visuelle des objets d'une application; de manière à ce que ces modifications se traduisent implicitement par des modifications de la structure de données de l'application.

L'éditeur étant accessible à l'usager quelque soit l'application employée, les commandes de l'éditeur permettent alors de réaliser des commandes génériques. On aboutit ainsi à une homogénéisation de certaines tâches élémentaires.

2.3.2. Le traitement des erreurs

Dans le chapitre précédent, nous avons mis l'accent sur l'aspect informatif du message d'erreur durant la phase d'apprentissage; celui-ci doit très explicite. Ceci a pour conséquence l'affichage d'un texte important, inutile pour les usagers connaissant parfaitement l'application. Il faut donc que le contenu de ce type de message puisse être adapté aux compétences de l'usager, et dépende du type d'erreur rencontrée: dans certains cas, une simple indication sonore suffit, dans d'autres, une information plus substantielle doit être fournie (lors d'une erreur de syntaxe, présence d'un paramètre éronné...). Ceci implique d'une part une analyse plus fine de la source de l'erreur, éventuellement replacée dans son contexte, ainsi qu'une meilleure connaissance de l'individu utilisant l'application (compétence, taux d'utilisation, date de dernière utilisation).

Le second aspect du traitement des erreurs concerne les possibilités de récupération et de continuation. En effet, dans bien des systèmes, le simple fait de commettre une erreur oblige l'usager à reformuler sa commande. Si cette solution semble satisfaire certains concepteurs d'applications, elle se révèle

être source d'énervement pour l'utilisateur <Hayes 81> qui ne peut modifier le texte initial de sa requête.

La récupération concerne essentiellement les erreurs de manipulation de l'usager. Une erreur de manipulation, bien souvent, ne représente pas une erreur au sens de l'application (commande ou paramètre incorrect). Elle constitue une faute non détectable par le système, qui apparait dans la suite logique des requêtes émises par l'usager et remet en cause l'obtention du résultat recherché.

Nous pouvons faire entrer dans cette catégorie d'erreurs le lapsus (émission d'une requête à la place d'une autre), l'erreur de frappe dont le résultat reste néanmoins une requête ayant un sens pour l'application, l'oubli etc...

La caractéristique de ces erreurs est qu'elles sont susceptibles d'être commises par tous, que leurs conséquences sont quelquefois fâcheuses, et qu'elles sont souvent détectées au moment même où elles sont émises. Donner à l'usager le moyen de défaire ou d'arrêter l'exécution de la commande permet de réduire les sources d'énervement inutiles.

Il se peut qu'il soit techniquement impossible de défaire certaines commandes, on a alors recours à une demande de confirmation explicite: l'application signale à l'usager les conséquences de la commande en cours ainsi que son caractère irréversible, et lui demande de confirmer son choix initial. L'expérience montre qu'un recours systématique à cette technique lui enlève toute efficacité: l'usager confirme immédiatement sa commande sans même attendre le message explicatif.

2.4. L'adaptation

Comme le font remarquer les concepteurs du Star <Smith 82>, il est impossible de satisfaire les exigences de tous les usagers

potentiels d'un système, quelle que soit sa puissance et son degré de généralité.

La seule solution est de concevoir le système en lui intégrant des possibilités d'adaptation utilisables par les usagers pour personnaliser leur environnement.

Plusieurs niveaux de personnalisation seront nécessaires:

- la personnalisation du poste de travail (adaptation de la forme);

- l'ajout de nouveaux opérateurs, en s'appuyant sur ceux qui préexistent (extension);

- l'adaptation du modèle par l'usager suivant ses besoins propre.

La manifestation la plus élémentaire de la personnalisation du poste de travail est la redéfinition des noms et icônes représentant les commandes ou objets de l'application, on donne alors à l'usager la possibilité d'employer des termes qui lui sont propres, avec les avantages vus au chapitre précédent. De même, on pourra lui permettre de préciser les options par défaut utilisées dans le système (objets, entêtes de fenêtre, niveau de détail dans les messages d'erreurs et d'aide). Toujours dans le cadre de l'adaptation de la forme, on doit y inclure l'organisation des informations sur le support de visualisation.

Laisser à l'usager le libre choix dans sa manière de travailler, c'est lui permettre par exemple de mener de front plusieurs activités, sans lui imposer une priorité arbitraire dans l'ordonnancement de ses tâches. Cette forme d'adaptation est inexistante sur la quasi totalité des systèmes, à cause de l'impossibilité de passer de manière simple d'une activité à une autre.

L'extension est une possibilité offerte par la majorité des systèmes existants (cf les exec_com de Multics). Dans tous les cas, il s'agit de fournir à l'usager un niveau de programmation supplémentaire. On peut se demander si celà résoud effectivement les problèmes d'adaptation pour des usagers non informaticiens, usagers qui représenteront sous peu la part la plus importante des utilisateurs de systèmes informatiques.

L'adaptabilité d'un modèle pose quand à lui des problèmes différents de par leurs natures, deux cas extrêmes pouvant se présenter:

- Soit l'application est paramétrée, et il se peut que l'adaptation soit réalisable par l'usager lui même, moyennant un travail qui reste non négligeable (détermination des paramètres permettant d'obtenir le modèle désiré);

- l'application n'est pas paramétrée, et soit l'usager doit recourir aux services d'un concepteur; soit il possède les connaissances informatiques nécessaires à la définition de sa propre application. Dans ce dernier cas, il se trouvera dans la position d'un concepteur, avec un problème en moins: il sait très exactement ce qu'il désire obtenir.

A cette occasion, nous pouvons remarquer que si l'adaptabilité d'une application est souhaitable, les techniques nécessaires à son obtention sont cependant souvent difficiles à mettre en oeuvre. On procède généralement cas par cas.

Dans la définition d'une interface de communication nous devons néanmoins nous poser ce problème. Une certaine adaptabilité peut être obtenue en définissant pour chaque composante de notre interface un ensemble limité d'options entre lesquelles l'usager peut choisir.

Si l'on fait correspondre à chacune de ces options un choix de conception possible (faut t-il par exemple afficher

horizontalement ou verticalement les éléments d'un menu ?) on peut espérer pouvoir satisfaire les "goûts" d'un maximum de personnes. Remarquons qu'une telle démarche peut impliquer la réalisation d'une interface très complexe où un grand nombre d'options sont présentes.

Si l'on procède de la sorte, le nombre des configurations possibles est une combinaison du nombre d'options définies dans chaque couche et, croit donc rapidement.

Si de plus, toute configuration peut être modifiée dynamiquement, nous pouvons espérer donner à l'usager quelques moyens qui lui permettent adapter l'application à ses besoins. Il reste évident que seul l'aspect externe de l'application pourra réellement être adaptée. C'est ce que nous appelons une adaptation de forme.

La tâche de l'interface sera de rendre ces adaptations de formes transparentes à l'application.

3. Architecture d'une interface usager

Ce qui précède nous a permis de préciser les besoins des usagers et des concepteurs d'applications. Résumons ceux des éléments qui nous seront nécessaires.

Pour les usagers, les moyens de communication avec l'application doivent être adaptés à leurs besoins. Ces besoins, nous l'avons vu, évoluent au cours du temps. Il est alors nécessaire de pouvoir leur proposer plusieurs formes de dialogues. La communication avec l'application ne se restreignant pas à l'émission de commandes, des possibilités de manipulation directe d'informations visibles doivent être disponibles.

Il leur est nécessaire de contrôler l'évolution des activités et pouvoir passer de manière simple d'une activité à une autre.

Chaque application doit pouvoir proposer simultanément plusieurs vues, éventuellement distinctes, des objets qu'elle manipule.

Pour faciliter l'apprentissage et l'utilisation des différentes applications du poste de travail, il faut assurer une certaine cohérence inter-application. Cette cohérence ne peut reposer que sur une discipline de réalisation, mais elle peut être obtenue par la définition de fonctions que l'usager peut employer quelle que soit l'application avec laquelle il travaille (actions génériques) et intégrées à l'interface usager.

Cette dernière devra enfin répondre aux besoins d'adaptation qu'expriment les usagers. A cette fin, l'interface devra renfermer les mécanismes nécessaires à la personnalisation des applications.

Nous avons vu que seule la forme pouvait être adaptée, et que nous devions laisser à l'usager le soin de choisir celles des options qui lui conviennent le mieux.

On imagine assez facilement la difficulté et le coût de la réalisation d'une application qui intègre l'ensemble de ces contraintes. Il est donc nécessaire que l'interface usager, de par les abstractions qu'elle propose au concepteur, en facilite la réalisation.

Dans ce but, il faut rendre l'application totalement indépendante de la forme effective du dialogue avec l'usager et traduire les manipulations des objets visualisés dans des termes compréhensibles par l'application: activation d'une de ses fonctions sur un objet qui lui est connu.

Pour faciliter la réalisation de ce que nous avions appelé le décompilateur, il nous faut lui proposer des outils qui lui permettent de gérer les informations qu'elle visualise et assurer la cohérence de toutes les occurences visibles.

L'écran étant partagé entre plusieurs applications, il faut définir pour chaque application la notion de terminal virtuel et lui assurer la transparence des moyens matériels qu'elle utilise.

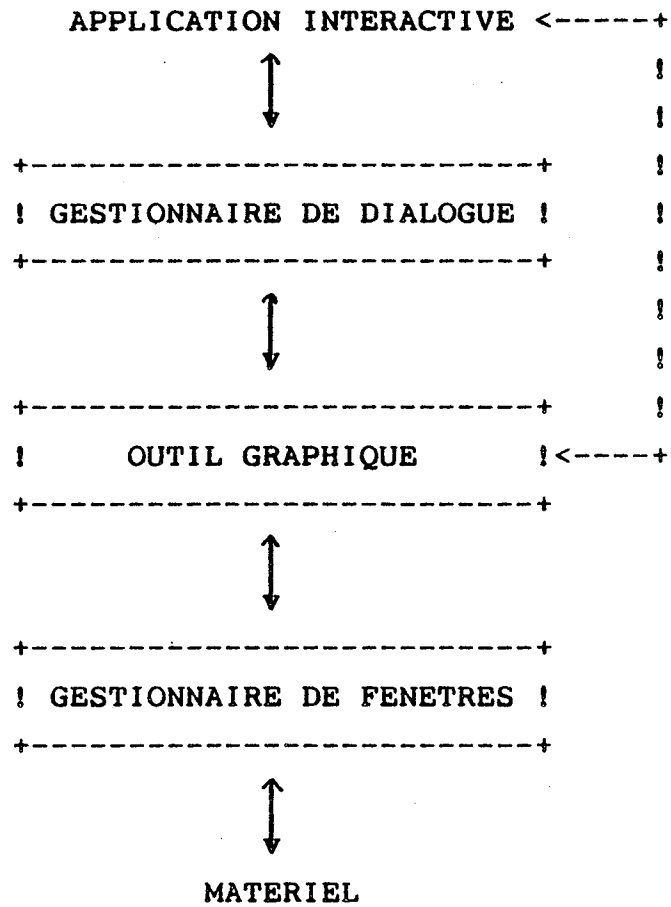
Pour résoudre l'ensemble des problèmes qui se posent, nous sommes amenés à décomposer l'interface usager en un ensemble de couches fonctionnelles. Trois classes de problèmes sont à résoudre:

- la gestion du dialogue;

- la visualisation d'informations et leur gestion;

- le partage des ressources physiques et le changement d'activité.

Le schéma fonctionnel qui en résulte est le suivant:



Le gestionnaire de dialogue réalise l'acquisition des commandes. La forme du dialogue est choisie par l'usager parmi trois options essentielles: le menu, le formulaire et le langage de commandes traditionnel. Il doit lui être possible de définir ses abréviations ou, plus généralement, de renommer ses commandes.

Pour en vérifier la validité, le gestionnaire de dialogue possède une description des commandes de l'application. Cette description lui permet de vérifier la validité syntaxique et de s'assurer de la cohérence sémantique (syntaxe contextuelle) en utilisant à cette fin les fonctions de vérification de l'application.

Pour réaliser la communication avec l'usager, le gestionnaire de dialogue s'appuie sur l'outil graphique.

L'outil graphique permet aux logiciels qui l'utilisent de décrire les informations qu'ils désirent visualiser. Il effectue la visualisation des informations à partir de leur description en projetant l'image qui en résulte dans une ou plusieurs fenêtres. Toute modification qui est apportée à cette description se traduit par la mise à jour des occurences visibles.

Vu sous cet aspect, l'outil graphique réalise l'interface entre le gestionnaire de fenêtres et les applications.

Son rôle ne peut cependant se limiter à la visualisation. Il doit en effet proposer des mécanismes d'acquisition d'informations. A cette fin, ce que nous avons appelé "description" aura les propriétés d'une structure d'échange: elle est accessible à la fois en "écriture" et en "lecture". Ainsi, toute information décrite pourra être consultée. (1) Il nous semble en effet impératif de simplifier la réalisation des

---

(1) Ce qui n'est pas le cas dans des logiciels tels que GKS ou Core

logiciels en unifiant entrée et sortie d'informations.

De plus, devront être intégrés dans le logiciel graphique des fonctions d'édition (elles constitueront une partie des actions génériques). Elles permettront d'obtenir une cohérence pour les tâches élémentaires d'édition.

Le gestionnaire de fenêtres gère le partage des ressources physiques (écran, clavier, moyen de désignation). Il définit la notion de fenêtre: portion rectangulaire de la surface de l'écran caractérisées par ses dimensions et sa position sur l'écran. Une fenêtre définit pour le logiciel graphique un terminal virtuel constitué d'une surface de visualisation, d'un clavier de touches de fonction, etc.

le gestionnaire de fenêtres, de par sa relation avec le système et l'usager, réalise le changement d'activité. A cette fin, il reconnait et interprète certaines interactions avec l'usager.

Plus généralement, il propose à ce dernier un ensemble de fonctions qui lui permettent de modifier la disposition et les dimensions des fenêtres. Ces fonctions sont les moyens que ce gestionnaire met à la disposition des usagers pour adapter leur poste de travail. L'une de ses fonctions essentielles est de réaliser l'interface entre le matériel et le logiciel graphique.

On pourrait faire remarquer que la décomposition fonctionnelle que nous venons de décrire ne présente aucun caractère original.

En effet, dans presque tous les systèmes il existe un "gestionnaire de dialogue" qui se charge de l'acquisition et de l'analyse des commandes. Certains permettent de réaliser des menus (cf. BCPL <Knox 78>), des formulaires (cf. Cousin <Hayes 83>) et de définir des abréviations (cf. Multics).

Les logiciels graphiques (cf. Core et GKS) permettent à une application de décrire des images et de réaliser l'interaction avec l'usager. Tous les problèmes de visualisation et d'indépendance par rapport aux terminaux étant résolus par ces logiciels.

La notion de gestionnaire de fenêtre enfin n'est pas nouvelle, la réalisation de VTMS <Lantz 79> date de 1979.

Les constituants sont donc connus, tout comme les abstractions qu'ils définissent.

Paradoxalement, c'est là que réside l'un des problèmes. Les abstractions sont connues, mais ne s'adaptent pas toujours de manière simple les unes aux autres. Leur multiplicité les rend difficilement utilisables par les applications, qui doivent réaliser l'interface entre ces abstractions.

EXEMPLE

BCPL <Knox 78> est un logiciel qui permet de décrire, d'afficher et de gérer (mise en évidence d'un élément de menu par exemple) des menus. Lorsque l'application désire interagir avec l'usager, elle appelle une fonction de BCPL qui lui retourne le numéro de l'élément désigné par l'usager. Cette fonction se charge d'effectuer la conversion coordonnées d'un point désigné-numéro d'un élément de menu.

Mais il est nécessaire, pour que ce logiciel puisse fonctionner, que l'application précise où afficher le menu sur l'écran, où positionner chaque élément de menu et définisse les coordonnées des coins du rectangle incluant chaque élément.

Supposons maintenant que la même application utilise les services d'un gestionnaire de fenêtres. Toutes coordonnées sont alors exprimées relativement à cette fenêtre et non par rapport à l'écran.

Problème: comment utiliser les services de BCPL pour la gestion des menus et les services du gestionnaire de fenêtres pour définir une fenêtre de dialogue?

L'application doit réaliser l'interface entre les deux utilitaires!

Il est donc nécessaire de réaliser non pas trois logiciels indépendants spécialisés chacun dans son domaine mais, un ensemble intégré et cohérent d'outils qui ne nécessitent aucune intervention de l'application pour réaliser leur intégration. Celle-ci doit être réalisée dans l'interface.

Afin que l'usager puisse, en toute occasion, garder un contrôle prioritaire, il faut éviter que le dialogue entre l'application et l'utilisateur puisse aboutir à des points de blocage dûs par exemple à l'existence de commandes qui définissent des modes de traitement.

Ce problème est habituellement illustré sur l'exemple de l'éditeur pour lequel on indique explicitement que l'on désire ajouter de l'information au texte, la fin de cette action étant délimitée par un symbole conventionnel ("\f" sous Ted, deux retour chariot sous d'autres éditeurs).

S'il n'y prend garde, l'usager peut omettre le symbole conventionnel et émettre ce qui représente pour lui des commandes, alors qu'il est toujours dans le mode "ajout d'informations". Celà modifie bien évidement le texte (effet non recherché) sans activer les fonctions souhaitées.

L'application en simplifiant à l'extrême les techniques d'acquisition d'information peut créer des modes de fonctionnement. Le contrôle explicite des interactions par l'application (on dit alors que le contrôle est interne) est la cause de ce type de situation.

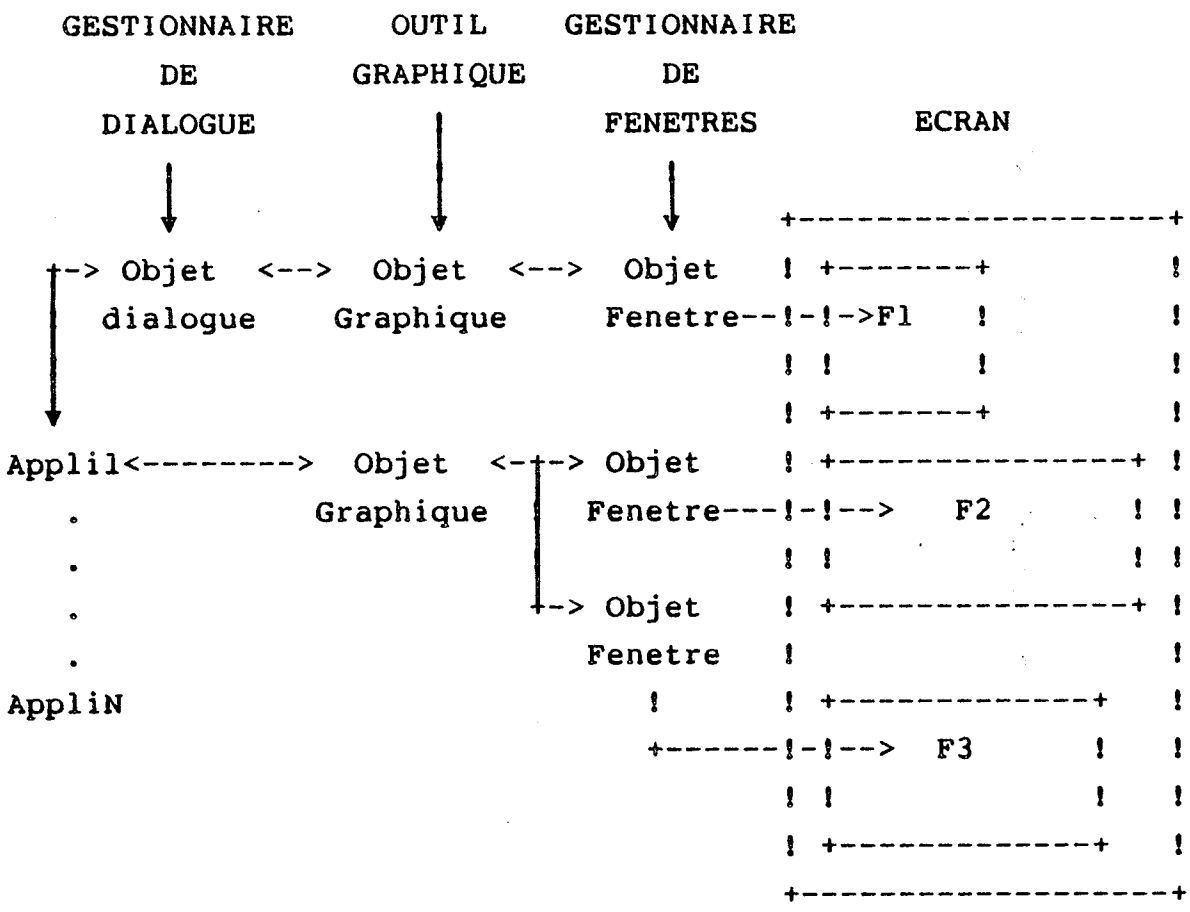
Il est alors nécessaire de sortir le flot de contrôle de l'application en lui interdisant toute acquisition d'information effectuée sous son contrôle direct (pas de "read" dans son code). Pour éviter que le même problème ne se pose dans l'interface usager, nous allons considérer que toute interaction avec l'usager est un évènement asynchrone.

Pour faciliter la réalisation des applications et donner à l'interface une certaine cohérence au niveau des concepts, on est amené à offrir des abstractions qui sont du niveau syntaxique pour le gestionnaire de dialogue et de niveau structurel pour l'outil graphique. Ce dernier point est une condition nécessaire pour résoudre en dehors de l'application le problème de l'écho des caractères (par exemple) dans des cas non triviaux. Cette condition satisfaite, nous pouvons alors résoudre le problème de l'acquisition des informations en introduisant la notion d'éditeur dans l'outil graphique.

Pour décrire l'architecture permettant de résoudre ces principaux problèmes, nous sommes amenés à faire correspondre à chaque constituant de la décomposition fonctionnelle une entité (code + données) que l'on peut appeler objet.

Pour résoudre les problèmes liés à l'asynchronisme des évènements et à l'adaptation entre couches, une communication bi-directionnelle devra exister entre deux objets appartenant à des constituants voisins de la décomposition fonctionnelle.

L'architecture qui en résulte peut être schématisée de la façon suivante:



Dans ce qui suit, nous commencerons par mettre en évidence les caractéristiques principales des périphériques les plus couramment employés aujourd'hui. Sont exclus de cet exposé la synthèse et la reconnaissance de la parole trop peu répandus actuellement. Puis, nous étudierons les différentes composantes de l'interface usager en commençant par les couches les plus proches du matériel.

3.1. Le matériel

Comme nous avons pû le constater, l'échange des informations qui a lieu entre une application et un usager lors d'un dialogue, ne possède pas un caractère séquentiel. Sur l'écran, les informations se présentent de manière organisée; l'usager les consulte en demandant éventuellement de revoir les informations qui ne sont plus visibles et leur apporte des modifications locales.

Dans ce type de communication, la visualisation ne peut plus être considérée l'envoi d'un flot d'informations à destination d'un support séquentiel.

Une information présente sur l'écran est la matérialisation visuelle de la structure logique de l'information manipulée par une application. Pour satisfaire à ce point de vue, les supports de visualisation doivent posséder certaines propriétés:

- ils doivent accepter des débits d'informations élevés. Selon notre expérience, ce débit devrait être d'au moins 4800 bauds pour des terminaux alpha-numériques;

- permettre l'adressage sélectif des informations visualisées;

- permettre la mise en évidence;

- posséder une forte définition pour une technologie de type matrice de points.

Pour accéder aux informations, les désigner et les manipuler, il est nécessaire de disposer de périphériques appropriés dont les qualités principales sont:

- la vitesse avec laquelle une information peut être atteinte;

- la précision avec laquelle une information peut être désignée.

La configuration standard d'un poste de travail (bureautique par exemple) est constitué d'un écran, d'un clavier et d'un organe de désignation. (1)

---

(1) Chacun des éléments employés satisfait à des contraintes ergonomiques qui ont été largement étudiées; nous ne les aborderons donc pas.

Dans ce qui suit nous allons mettre en évidence les principales caractéristiques des constituants matériels.

Nous distinguerons à cet effet les terminaux alpha-numériques des terminaux graphiques qui se différencient par l'unité d'information traitée. Nous limiterons notre exposé aux technologies les plus couramment utilisées.

Enfin, nous passerons en revue les principaux moyens de désignation.

3.1.1. Les écrans alphanumériques

Les écrans alpha-numériques manipulent l'unité d'information qui est le caractère. Ils emploient un générateur de caractères pour transformer un code ASCII ou EBCDIC en sa représentation visuelle (habituellement une matrice de points). Du nombre de générateurs de caractères dépend le nombre de polices de caractères alpha-numériques ou semi-graphiques disponibles.

L'écran est d'une technologie est identique à celle d'un tube de télévision. La quantité d'information qu'on peut présenter sur ces écrans varie entre 24 lignes de 80 colonnes et 16 lignes de 64 colonnes. Remarquons que certains terminaux permettent la définition de lignes de 132 caractères.

La tendance actuelle est d'intégrer dans ces terminaux des fonctions qui doivent faciliter la réalisation de logiciels de gestion d'écran. Pour accéder à ces fonctions, le logiciel envoie des séquences particulières de caractères qui sont interprétés par le terminal.

Ces fonctions permettent:

- l'adressage d'une partie quelconque de l'écran.

Ils permettent ainsi l'écriture sélective par positionnement du curseur.

- la mise en évidence d'informations.

Celle-ci est obtenue par association d'un ensemble d'attributs de visualisation (inversion vidéo, clignotement, soulignement, surbrillance) à la chaine de caractères à mettre en évidence.

Pour modifier la valeur de l'un de ces attributs, il sera nécessaire de réécrire la chaine de caractères en lui associant des valeurs d'attributs différentes; (1)

- l'insertion de lignes et parfois de caractères;

- l'effacement sélectif.

Celui-ci se limite à des actions élémentaires qui permettent l'effacement d'une ligne ou d'un écran dans sa totalité et l'effacement du début de la ligne à la position courante du curseur, ou de cette position jusqu'à la fin de la ligne.

Si l'on veut opérer de façon plus fine, il faut surcharger le texte initial avec le caractère espace, ce qui est à la fois plus long (temps de réponse) et plus coûteux en temps de calcul;

- le déplacement du curseur sur l'écran.

---

(1) Remarquons qu'il est alors nécessaire de connaitre la valeur des attributs de visualisation et la chaine de caractères. Il faut donc mémoriser au niveau du logiciel les informations qui sont visibles.

L'usager emploie des touches du clavier qui sont spécialisées dans le déplacement du curseur. Leur interprétation peut rester locale au terminal. Soulignons que ce déplacement est lent et malaisé si ces touches ne sont pas à répétition automatique.

Le positionnement effectué, l'usager le fait savoir à l'application. Pour connaitre la position du curseur sur l'écran, l'application doit envoyer une requête au terminal qui lui retourne alors cette position. Il reste alors à interpréter ces coordonnées ... (1)

- le partitionnement de l'écran pour la réalisation du fenêtrage.

Dans certains terminaux (VT100 par exemple) est définie la notion de partition courante: suite de lignes adjacentes de l'écran qui se comportent comme un terminal, mais de taille restreinte. Les fenêtres ne sont pas connues du terminal, qui ne connait que la "fenêtre" courante.

D'autres (OPTI 940 par exemple) gèrent effectivement toutes les fenêtres; celles-ci n'occupent pas forcément toute la largeur de l'écran. Mais, il faut calculer les dimensions de toutes les zones qui sont implicitement créées lors de la définition d'une fenêtre: la création d'une zone rectangulaire sur un écran peut impliquer le calcul des dimensions de quatre zones rectangulaires.

A l'usage, les fonctions intégrées aux terminaux (quand elles existent) se révèlent être trop élémentaires pour réellement faciliter la réalisation d'un multi-fenêtrage: le nombre des fenêtres est limité, la superposition est interdite, la définition

---

(1) A cette fin, il faut maintenir dans le logiciel une association position-information.

d'une fenêtre est quelquefois une opération complexe.

Notre expérience dans ce domaine montre l'inadéquation de ces fonctions à une réalisation aisée de la gestion des informations affichées sur un écran. Celle-ci reste à la charge du calculateur hôte.

Ceci a pour conséquence un coût de réalisation élevé, et la nécessité de disposer d'une puissance de calcul importante pour gérer l'écran, quelle que soit "l'intelligence" actuellement intégrée à ces terminaux.

3.1.2. Les écrans graphiques

Sont regroupés sous cette dénomination l'ensemble des écrans qui permettent la visualisation d'informations élémentaires autres que des caractères, c'est à dire des points ou des segments. Ces périphériques utilisent différents technologies (tubes à mémoire, à balayage cavalier, à balayage de trame, terminaux à plasma, etc). La plus répandue actuellement utilise un tube à balayage de trame.

Cette technique emploie des tubes de télévision, et bénéficie donc des progrès réalisés en ce domaine. Rappelons que l'image est obtenue sur un téléviseur par un déplacement ligne à ligne du faisceau électronique. Ce balayage se répète une trentaine de fois par seconde. Le rafraichissement nécessite une mémoire de trame (bitmap) qui décrit l'image point par point.

Le coût d'un tel périphérique est fonction de la définition de l'image. Celle-ci varie de 512x512 (considérée comme "faible") à 1024x1024 (qui est une définition courante).

La mémoire de trame est directement accessible par le processeur qui construit l'image. Cette technique possède donc l'avantage de ne poser aucun problème d'adressage sélectif.

Ce type d'écran nécessite néanmoins un investissement important dans la définition des fonctions de bases, nécessaires à son utilisation. En effet, si les fonctions intégrées aux terminaux alpha-numériques ne sont pas parfaitement adaptées à nos besoins, elles présentent toutefois l'avantage d'exister et, de ne nécessiter aucune puissance de calcul autre que celle intégrée au terminal.

Dans ce type de technologie, il en va tout autrement. (1) Il faut définir et réaliser les fonctions élémentaires qui prennent en en charge:

- le tracé des formes géométriques élémentaires (segment, cercles, rectangles, etc);

- les opérations de déplacement de surfaces rectangulaires dans la mémoire de trame et, de manière plus générale, de gestion de cette mémoire.

Ces fonctions nécessitent une forte puissance de calcul, aussi sont-elles soit micro-programmées (solution retenue pour le Perq), soit exécutées par un processeur dédié à la gestion de la mémoire de points (c'est ce qui avait été prévu lors de la définition du buroviseur, et qui est en prévision pour les SM 90 haut de gamme).

La forte définition des écrans graphique permet de réaliser des logiciels élaborés qui, par exemple, permettent de créer des documents dans lesquels texte et graphique se mélangent et où plusieurs polices de caractères sont employées.

---

(1) Il existe, il est vrai des terminaux à balayage de trame qui proposent des primitives graphiques du niveau GKS (cf 3.3). Ces terminaux renferment un processeur qui se charge de la gestion de la mémoire de trame. Ce sont des terminaux aux possibilités plus étendues que les terminaux alpha-numériques mais posent fondamentalement les mêmes problèmes.

La surface de ces écrans étant plus importante que celle des écrans alpha-numériques, la quantité d'information qui peut y être visualisée est elle aussi plus importante, ce qui permet d'assurer plus de confort à l'usager.

3.1.3. Les organes de désignation

Nous regroupons sous cette appellation un ensemble de périphériques habituellement associés à des terminaux graphiques. Ils permettent à l'usager de désigner un endroit précis sur l'écran. Le résultat fourni par ces périphériques permet au système de déterminer les coordonnées du point ainsi désigné.

Parmi ceux-ci, on peut citer:

- les tablettes à digitaliser;

- le réticule que l'on déplace à l'aide de deux molettes intégrées au terminal;

- le photostyle, qui constitue un moyen de désignation peu précis;

- l'écran sensitif, qui ne nécessite aucun instrument additionnel: on touche de son doigt l'écran à l'endroit où l'on veut désigner;

- la souris.

Dans ce chapitre nous ne détaillerons pas l'intérêt qu'il y a d'utiliser ces périphériques, ni l'utilisation que l'on peut en faire (cf 2.3.1). Les problèmes liés à l'exploitation du résultat d'une désignation ont été abordés en 2.1.

3.1.4. Conclusion

Nous retiendrons de ce tour d'horizon des matériels dont nous pouvons disposer que si ceux-ci répondent effectivement aux besoins que nous avons exprimés, ils nécessitent cependant un investissement important dans la réalisation de logiciels qui permettent d'en exploiter les possibilités.

En effet, si les problèmes liés à la visualisation sont généralement résolus, les fonctions nécessaires à la gestion des informations visibles restent à la charge du logiciel qui les utilise.

3.2. Le gestionnaire de fenêtres

Le gestionnaire de fenêtres donne à l'usager les fonctions nécessaires pour partitionner l'écran. L'unité de partition est la fenêtre. A une activité du poste de travail les gestionnaires actuels associent une fenêtre: lieu de visualisation des données de l'application et d'écho des interactions avec l'usager (écho des caractères frappés au clavier par exemple). Une activité correspond à l'utilisation d'une application par un usager.

Sur l'ensemble des activités du poste est définie la notion d'activité courante: c'est l'activité propriétaire de la fenêtre courante. La fenêtre courante est la fenêtre vers laquelle sont dirigées toutes les informations qui résultent des interactions avec l'usager.

Le gestionnaire de fenêtres a pour tâche de déterminer, lors de toute interaction avec l'usager, à quelle fenêtre l'information est destinée et assure la redirection des entrées.

La fenêtre assure aux applications l'indépendance syntaxique par rapport aux terminaux réellement employés. Elle définit un terminal virtuel qui s'appuie sur les fonctions intégrées au

matériel (terminaux intelligents) et sur le logiciel élémentaire qui gère la mémoire de trame (écrans graphiques). De même elle résoud les problèmes liés au recouvrement des fenêtres (si le recouvrement est permis).

Pour assurer aux applications la transparence des recouvrements, la technique habituelle consiste à adjoindre à chaque fenêtre une mémoire qui possède le même rôle que la mémoire d'écran intégrée au terminal réel.

Cette mémoire est utilisée lorsque la "repeinture" de la fenêtre est nécessaire c'est à dire lorsqu'une modification de l'organisation des fenêtres sur l'écran découvre l'une d'elles. Les techniques employées à cette fin <Pike 83, Meyrowitz 81> dépassent le cadre de notre propos et ne seront pas explicitées ici.

En sortie, la fenêtre se comporte comme un terminal virtuel qui aiguille les informations à afficher vers la surface qui matérialise la fenêtre sur l'écran.

3.2.1. Etat de l'art

Les gestionnaires décrits dans la littérature sont conformes à la description très générale que nous venons d'en donner. Aussi ne reviendrons nous pas sur les aspects fonctionnels.

Beaucoup de gestionnaires de fenêtres ont été réalisés pour des terminaux alpha-numériques. Parmis eux nous pouvons citer VTMS <Lantz 79>, Bruwin <Meyrowitz 81>, Wm <Robert 84>, BBN <Mankins 83>. D'autres réalisations sont plus spécifiques à une machine tels ceux du Perq <Perq 82> ou de la Lisp Machine <Symbolics 81> qui tirent profit d'un écran à point.

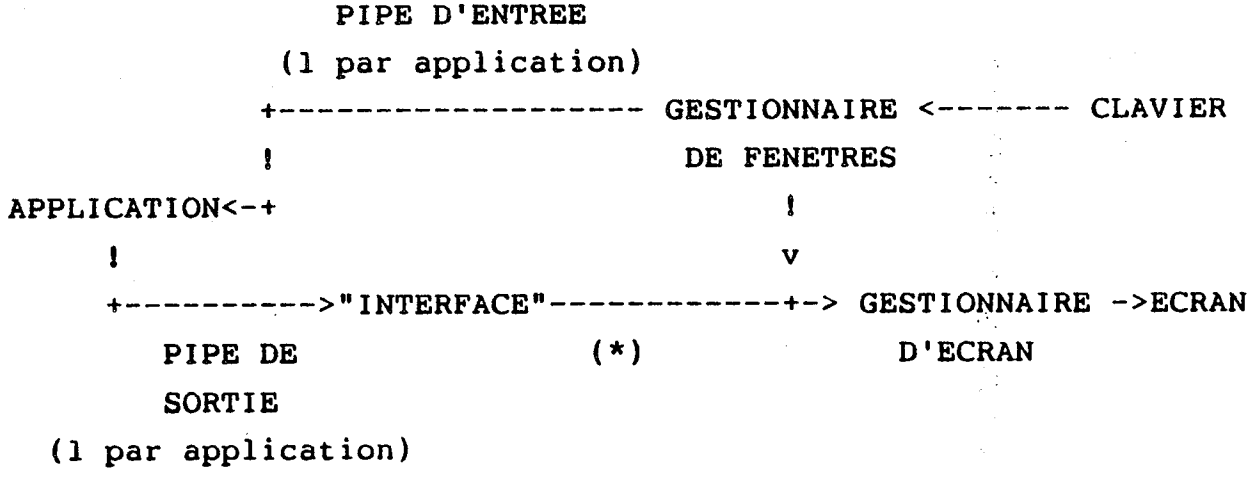
Dans ce qui suit nous allons détailler deux réalisations.

La première Wm nous permettra de mettre en évidence les caractéristiques principales de tous les gestionnaires de fenêtres ainsi que leur liaison avec le système (Unix pour Wm).

La seconde, VTMS présente l'intérêt de définir une relation entre la fenêtre et la partie du logiciel qui opère sur une structure de donnée représentant l'image et qui la projette dans une fenêtre. Appelons celà une "fonction de projection". Ceci nous permettra de faire le lien entre la fenêtre et le logiciel graphique.

Wm est un gestionnaire réalisé pour le système Unix, qui permet à l'usager de contrôler plusieurs applications. Chacune d'elles est un processus qui communique avec Wm via des "pipes" Unix. Le nombre total de processus nécessaires à l'exécution simultanée de N applications utilisant Wm est 2N+2.

L'architecture de ce gestionnaire est la suivante:



Le gestionnaire de fenêtre effectue la lecture des caractères entrés au clavier et en fonction de la fenêtre courante, il détermine sur quel pipe d'entrée il faut transmettre le caractère (il y a un pipe d'entrée par application). Le fait d'envoyer un caractère sur le pipe d'entrée d'un processus bloqué en lecture sur ce pipe, a pour conséquence la réactivation de ce processus.

Le changement d'activité utilise ce principe.

Il reconnait et interprète un ensemble de commandes (destruction et création de fenêtre, modification des dimensions et déplacement) qu'il transmet au gestionnaire d'écran via le pipe noté (*) qu'il partage avec les "interfaces" (1 par application).

L'"interface" réalise l'adaptation entre les informations envoyées par l'application (écriture d'une chaine de caractères) et le gestionnaire d'écran en préfixant cette chaine avec l'identification de la fenêtre à laquelle elle est destinée.

Sur ce pipe transitent donc des commandes destinées au gestionnaire d'écran et des informations.

Le gestionnaire d'écran réalise les fonctions de création, destruction et modification de fenêtre et effectue l'affichage des informations sur l'écran.

Etant le plus proche du matériel, il assure l'indépendance syntaxique des applications et du gestionnaire de fenêtres par rapport au terminal par l'emploi du "termcap": fichier décrivant chaque type de terminal.

L'architecture de VTMS est différente même s'il assure globalement les mêmes fonctions. La première différence avec Wm se situe au niveau de la définition de la notion de terminal virtuel.

VTMS définit un terminal virtuel, non pas en terme de signaux élémentaires (caractère, coordonnées fenêtre, ...), mais en terme d'entités logiques que sont:

- la ligne; elle représente l'unité logique d'entrée d'informations. Sur la ligne est défini un éditeur;

- le "Pad": c'est une structure de donnée sur disque permettant la mémorisation des informations affichées par l'application. Un Pad est associé a chaque terminal virtuel;

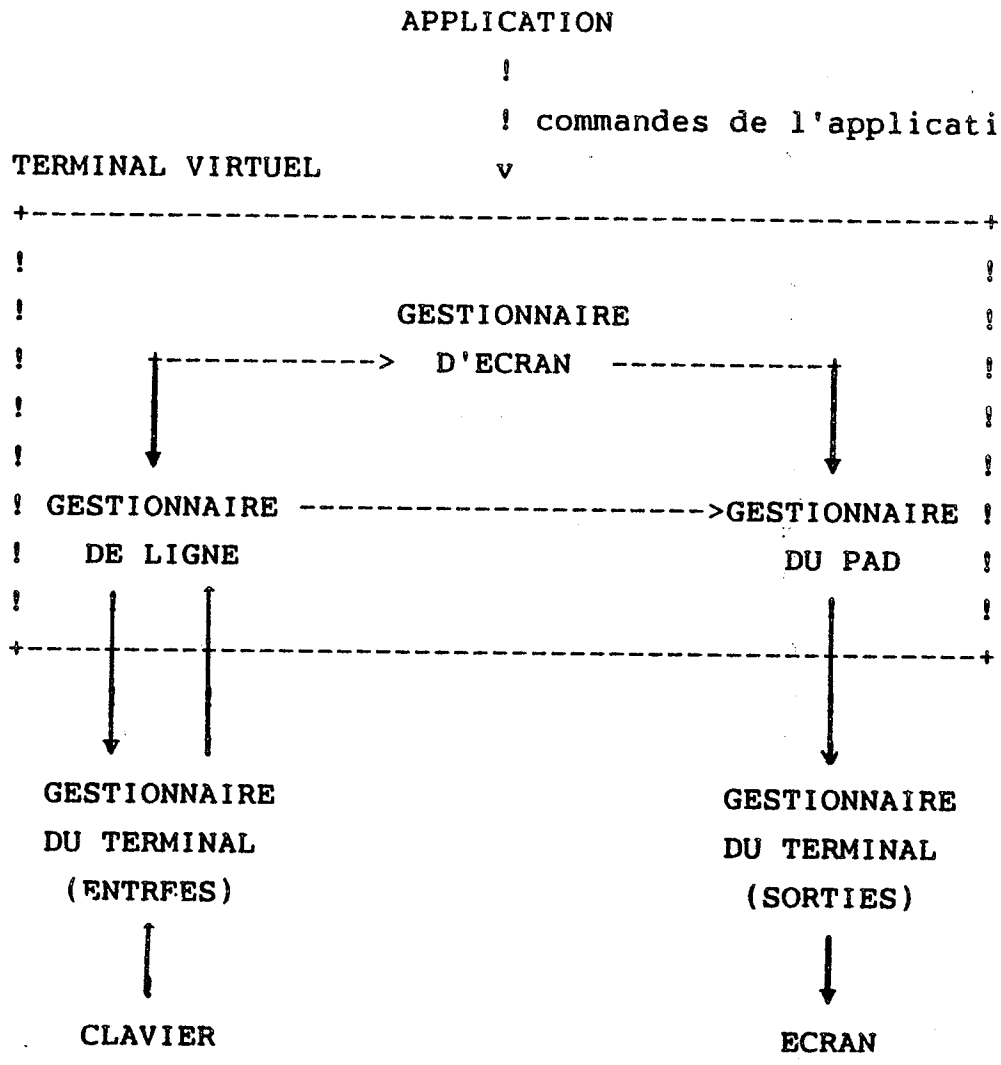
Sur ce Pad sont définies entre autres fonctions, la mémorisation des lignes qui, sous l'effet des défilements ont disparu de l'écran, la possibilité d'utiliser ces lignes comme nouvelles entrées, le maintien de la cohérence entre le contenu du Pad et sa partie visible sur l'écran. Le Pad est la description de l'image qui apparait dans une fenêtre.

- la gestion de l'écran: elle est effectuée par un gestionnaire de fenêtres qui utilise une décomposition hiérarchique pour gérer l'écran sous la forme d'une mosaique de fenêtres, mais autorise le recouvrement total d'une mosaique par une autre.

A chaque activité est associé un ensemble de fenêtres regroupées dans une "Super-fenêtre". Plusieurs des ces Super-fenêtres peuvent être présentes simultanément sur l'écran (configuration définie par l'utilisateur), elles constituent une "Image".

Plusieurs Images peuvent exister à un instant donné dans le système, mais une seule d'entre elles est visible (recouvrement de mosaiques). L'usager commute d'une Image à une autre par l'envoi d'une commande au gestionnaire.

L'architecture de VTMS repose sur 5 processus COOPERANTS. Le schéma de l'architecture est donnée ci-dessous.



Le gestionnaire d'écran est chargé de gérer la partition de l'écran en fenêtres. Il envoie les demandes d'interaction exprimées par l'application au gestionnaire de ligne.

Le gestionnaire de ligne, lorsqu'il reçoit un caractère du clavier, transmet cette information au gestionnaire du Pad qui modifie la structure de données et fait l'écho. Certaines touches correspondent à des fonctions d'édition. Le gestionnaire de ligne active les fonctions correspondantes du gestionnaire de Pad.

Lorsque le texte est modifié par l'application ou l'usager, ce gestionnaire répercute les modifications sur la structure de données et sur l'écran en se servant des fonctions du gestionnaire du terminal.

La définition de VTMS nous amène à remarquer la présence d'un ensemble de fonctions de manipulation de données visuelles et, d'une forme (élémentaire) de structuration (le Pad est une suite de lignes). Enfin, le gestionnaire du Pad réalise la projection d'une structure de donnée dans une fenêtre. L'image correspondant à la structure occupe éventuellement une surface supérieure à celle de la fenêtre dans laquelle elle est projetée.

Le gestionnaire du Pad est une forme simple de fonction de projection. VTMS établit le lien entre une description d'image et son support d'affichage.

Ces aspects sont inexistant dans les autres gestionnaires de fenêtres cités.

3.2.2. Les limitations

Les principales limitations (VTMS excepté) proviennent de l'inexistence de liaison entre la fenêtre et la fonction qui y projette les informations. Comment prendre par exemple en compte l'augmentation de la taille d'une fenêtre sans cette liaison?

EXEMPLE

L'usager modifie ces dimensions pour adapter la quantité d'information visible à ses besoins actuels. Pour limiter la quantité d'information, il peut diminuer les dimensions de la fenêtre.

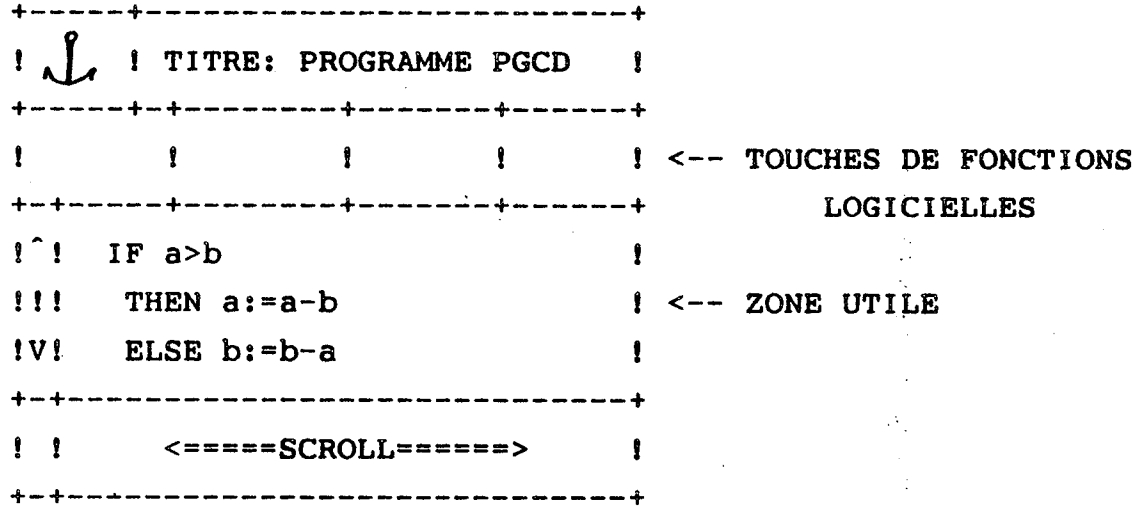
Par l'utilisation de la mémoire associée à la fenêtre, on peut cacher cette modification à l'application en s'en servant pour mémoriser les parties qui ne sont plus visibles. Remarquons qu'il faudra alors définir au niveau de la fenêtre la notion de défilement.

Dans le cas d'une augmentation de la taille, la fenêtre ne peut plus masquer ce changement à l'application. Pour répondre aux besoins de l'usager, il faut que l'application (ou l'outil graphique) fournisse les informations supplémentaires pour compléter la surface libre ainsi créée.

La liaison entre la fonction de projection et la fenêtre est donc nécessaire.

Une autre des limitations est que la notion même de fenêtre est trop élémentaires dans des gestionnaires tels que Wm, Bruwin ou BBN.

Sur des écrans plus élaborés, une fenêtre peut se présenter sous la forme d'une surface rectangulaire qui se matérialise sur l'écran par un cadre dans lequel s'inscrivent un ensemble de zones aux propriétés distinctes:



- une zone d'ancrage (représentée sur le schéma par une ancre) dans laquelle l'usager positionne le réticule pour asservir la fenêtre à ses déplacements. Toute désignation de cette zone est interprétée localement au gestionnaire de fenêtre.

Suite à cette action, le gestionnaire peut détecter la réapparition de certaines fenêtres précédemment masquées. Des

actions de repeinture sont alors nécéssaires; au gestionnaire de déterminer l'ordre optimal dans lequel celles-ci doivent être effectuées;

- la zone dans laquelle s'affiche le titre de la fenêtre. Ce titre est un identificateur auquel l'usager peut faire référence s'il veut, par exemple, faire réapparaitre une fenêtre entièrement recouverte;

- la zone où sont matérialisées les touches de fonction logicielles. Chacune de ces touches est représentée par une icône. La désignation de l'une d'entre elles provoque l'envoi vers l'application d'un signal identique à celui de l'appui sur une touche de fonction clavier.

Cette zone, contrairement aux précédentes, ne s'adresse pas au gestionnaire de fenêtres, mais à la fenêtre elle même.

- les deux zones notées "scroll" définissent des barres de défilement <Perq 82> dans lesquelles l'usager doit pointer s'il veut faire défiler le contenu de la zone utile. Ces deux zones doivent être gérées par la fenêtre: elles s'adressent à la fonction de projection;

- la zone utile dans laquelle l'application visualise ses résultats, et vers laquelle sont dirigés les caractères et les autres informations en provenance de l'usager lorsque la fenêtre est fenêtre courante. Vue de l'application, la fenêtre se réduit à la zone utile.

Pour ce type de fenêtres, les informations qui transitent sont plus riches donc plus complexes. Le nombre de touches de fonctions logicielles peut dépendre de l'application qu'elle représente, un ensemble de fenêtres distinctes peuvent représenter une même application.

3.2.3. Propositions

La fenêtre réalise pour l'application un terminal virtuel qui:

- en sortie se réduit à la zone utile. Cette zone définit un espace de coordonnées équivalent aux coordonnées écran d'un terminal réel. Cet espace de coordonnées est explicitement connu par la fonction de projection du logiciel graphique.

Sur cette zone utile est défini un ensemble de commandes (affichage de caractères, tracé de figures géométriques, changement de fonte) qui s'expriment dans l'espace de coordonnées précédement défini. La mise en correspondance avec le terminal réel est réalisée par la fenêtre;

- en entrée interprète ou transforme dans des signaux standards les désignations dans certaines zones. Toute désignation effectuée dans la zone utile se traduit par une normalisation des coordonnées. Le résultat de cette transformation est transmis vers la fonction de projection.

Tous les autres signaux à destination de la zone utile, et qui ne concernent ni le gestionnaire de fenêtres ni la fenêtre, ne subissent aucune transformation (frappe des caractères par exemple).

La relation entre la fenêtre et le logiciel qui l'utilise repose sur une coopération. Ainsi, toute interaction sera transmise de manière asynchrone vers le logiciel utilisateur. Certaines actions sur la fenêtre peuvent se traduire par une demande de traitement émise par la fenêtre vers le logiciel utilisateur.

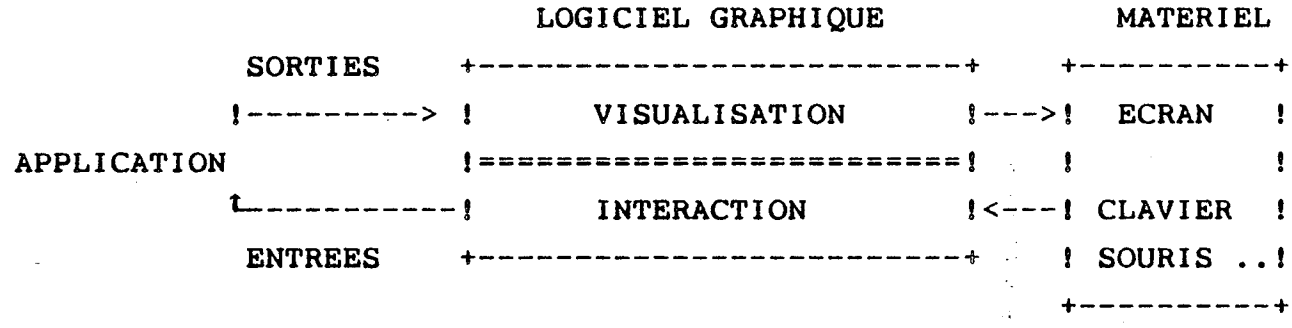
Le même type de relation sera nécessaire entre la fenêtre et le gestionnaire. Il est en effet possible que le logiciel graphique demande à une fenêtre de se détruire; la connaissance de cet évènement est impérative au gestionnaire de fenêtres.

Celui-ci reconnait un ensemble de commandes qu'il traduit comme des actions sur des fenêtres (comme par exemple déplacer une fenêtre). Certaines actions seront effectuées par le gestionnaire seul (changement d'activité), ou nécessiteront un traitement effectué par une fenêtre (modification de dimensions).

Il possède la charge de la gestion de l'écran et de la redirection les informations vers la fenêtre courante.

3.3. Le logiciel graphique

La définition des logiciels graphiques résulte de la nécessité de simplifier la réalisation des applications qui produisent des images et de les rendre indépendantes des terminaux employés. Le logiciel graphique joue le rôle d'une interface entre l'application qui le contrôle et le matériel.



Les logiciels ainsi définis, comme Core <Graphics 77> et GKS <GKS 83>, résolvent surtout les problèmes de visualisation et d'acquisition d'informations. (1)

---

(1) Core et GKS tendent par ailleurs à s'imposer comme des standards. Ils sont donc employés dans la définition d'interfaces usagers bien que leur utilisation soulève quelques problèmes liés au bas niveau des abstractions qui y sont définies (cf 3.3.1).

Ces logiciels définissent la notion de terminal logique ou "Workstation" (WS) qui représente une collection de périphériques (UN écran, et un nombre arbitraire de claviers, de tablettes, de souris, etc). A chaque WS est associée une surface de visualisation qui possède un système de coordonnées défini par le constructeur. Tous les périphériques de désignation associés à une WS expriment leurs résultats dans ce système.

L'application décrit une image à l'aide de primitives graphiques telles que tracer un cercle, tracer un segment de droite, etc. Cette image est décrite dans un système de coordonnées propre à l'application que nous appellerons le référentiel application (description d'une pièce de 3x5x2,5 mètres par exemple).

Cette image est visualisée sur un écran (de dimensions plus faibles ...) par une fonction de projection qui effectue la visualisation sur la WS. Le logiciel graphique maintient (sous certaines conditions) la correspondance entre la description et l'image en utilisant les fonctions du terminal graphique.

Afin de permettre à des applications de visualiser plusieurs parties d'une même image (une vue globale de la pièce et un gros plan sur le bureau qui s'y trouve par exemple), ces logiciels définissent les notions de "Window" (WD) et de "Viewport" (VP).

Une "Window" ne possède pas les mêmes caractéristiques qu'une fenêtre telle que nous l'avons définie en 3.2. Aussi utiliserons nous les termes anglais ou leurs abréviations pour éviter toute confusion.

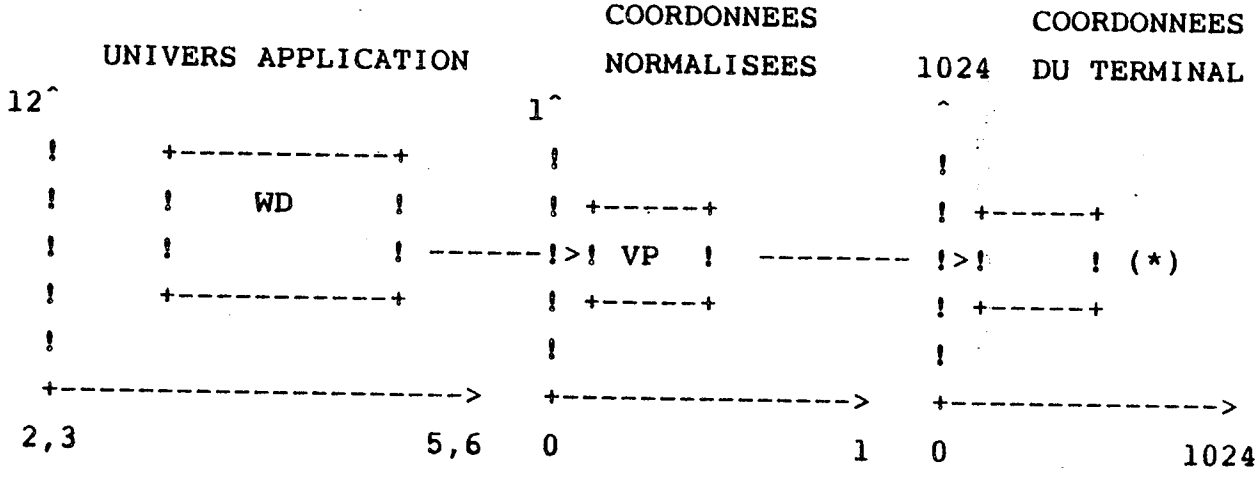
Une WD est définie comme une surface rectangulaire de l'univers du programme d'application. Ses dimensions et sa position sont exprimées par l'application dans le référentiel application.

Une WD est représentée sur l'écran dans un VP. Le VP est une surface rectangulaire de l'écran dont la position et les

dimensions sont exprimées dans un système de coordonnées normalisées. L'utilisation des coordonnées normalisée est obligatoire si l'on veut exprimer les paramètres de la VP indépendamment du système de coordonnées du terminal.

Une VP se projette finalement sur la workstation.

EXEMPLE



Où (*) représente ce qui est effectivement visible sur l'écran.

Ces notions permettent à l'application de gérer plusieurs images présentes sur un même écran. Remarquons que toute WD et VP créées le sont sous le contrôle explicite de l'application et que la correspondance WD-partie d'une image est gérée par celle-ci.

En entrée, le logiciel graphique met à la disposition des applications des périphériques logiques qui leurs permettent d'acquérir les informations; lecture d'une chaine de caractères, acquisition des coordonnées d'un point, résultat d'une désignation, etc.

Des limitations de ce type de logiciels (cf 3.3.1), que nous regrouperons sous le terme de logiciels de visualisation, est issue une nouvelle tendance qui amène le logiciel graphique à répondre aux besoins des applications de composer et de gérer des

images, nous les appellerons des logiciels de composition (cf 3.3.3).

Pour ceux-ci, la visualisation n'est plus une fin en soi, mais une conséquence de la manipulation de la structure d'une image.

Ces logiciels graphiques réalisent la synthèse d'images (images réalistes, animation, etc) et, aspect intéressant, résolvent les problèmes liés à la gestion des images. C'est sous ce dernier point de vue que nous les aborderons.

Dans ce qui suit, nous distinguerons l'entrée des informations de la description des images. Cette distinction peut paraitre arbitraire, mais se justifie par le fait que la plupart des logiciels graphiques sont conçus sur le principe de la séparation totale entre entrée et sortie d'informations. Cette absence de liaison s'exprime tant au niveau des concepts qu'au niveau des primitives.

Dans la suite de ce chapitre, nous mettons en évidence les principales caractéristiques des deux classes de logiciels graphiques (logiciel de visualisation et de composition) en fonction des besoins et des problèmes rencontrés dans la réalisation d'applications interactives; puis, nous préciserons la notion d'interaction.

3.3.1. Les logiciels de visualisation

Tous les logiciels graphiques reposent sur le principe de la séparation entre modélisation (description d'un objet) et visualisation (génération d'une image à partir de sa description). La distinction entre logiciel de visualisation et logiciel de composition d'image repose sur la différence du niveau des abstractions qui permettent cette description.

Dans les logiciels de visualisation, une image est décrite par l'application comme une succession (temporelle) d'appels à des primitives graphiques. Ces primitives permettent la description des éléments graphiques de base: lignes brisées, polygones, grilles, formes géométriques (cercle, ellipse, ...) et texte.

Elles réalisent le tracé de ces éléments en tenant compte des attributs qui leurs sont associés. Ceux-ci définissent l'apparence des éléments graphiques en précisant par exemple le type du trait (plein, pointillé), la forme des caractères (fonte, hauteur) ou la couleur du tracé.

Si l'application veut permettre à l'usager de désigner des éléments d'image sur l'écran, il faut qu'elle exprime l'association à une primitive graphique d'un identificateur qui le désigne. A cette fin, elle associe un attribut d'identification à une primitive graphique.

Cet attribut d'identification ou "pick_id" est défini par l'application et peut être adjoint à toute primitive. (1)

EXEMPLE

```
(* description d'un triangle dont  *)
(* chaque côté peut être désigné *)

identifier (1)        (* 1= identificateur du premier côté *)
décrire_droite ((0,0),(5,4)) (* coordonnées du segment "1"  *)
identifier (2)
décrire_droite ((5,4),(5,2))
```

---

(1) Remarquons que la correspondance entre l'identificateur et la nature de la primitive à laquelle il est associé doit être gérée par l'application. Un tel identificateur ne peut en effet être employé par l'application pour prendre connaissance de la nature de l'élément graphique qu'il identifie.

```
identifier (3)
décrire_droite ((5,2),(0,0))

(* fin de description du triangle *)
```

La notion de "Segment" a été introduite afin de permettre une certaine structuration de l'image il représente l'unité de modification. (1) Il peut être défini comme le regroupement d'une succession de primitives graphiques sous une même identification . Cet identificateur est employé par l'application pour manipuler le segment et, il lui est retourné par le logiciel de visualisation (en plus de l'identificateur de primitive graphique) en cas de désignation.

Exemple de définition de segment:

```
/* CREATION D'UN SEGMENT */

ouvrir_segment (1)          /* (1)= nom du segment créé     */
 decrire_cercle (4,5,2)     /* coordonnées et rayon         */
 decrire_droite (0,7,10,7) /* droite tangeante au cercle */
fermer_segment              /* fin de définition du segment */
```

Les manipulations se résument à:

- la création (un seul segment ouvert à un instant donné),

- la fermeture (qui indique la cloture de la liste de primitives),

- la destruction,

---

(1) Les manipulations autorisées sont adaptées aux possibilités des terminaux à balayage cavalier. <Rosenthal 82>.

- l'insertion d'un segment dans le segment courant ouvert (il s'agit en fait d'une copie des primitives contenues dans le segment référencé),

- et enfin la modification.

La modification est restreinte à l'ensemble des attributs dynamiquement modifiables c'est à dire:

- la transformation (rotation, translation, ...),

- la visibilité,

- la priorité et la détectabilité (en cas de désignation).

L'application décrit l'image à l'aide des primitives précédement citées en utilisant son propre système de coordonnées. Elle définit ce système en fonction de critères qui lui sont propres.

La transformation du système de coordonnées application vers le système de coordonnées du terminal est effectuée par le logiciel graphique. Deux étapes sont nécessaires:

```
COORDONNEES      (T1)      COORDONNEES    (T2)      COORDONNEES
APPLICATION  ------------>    ECRAN     ---------->     DU
                           NORMALISEES                TERMINAL
```

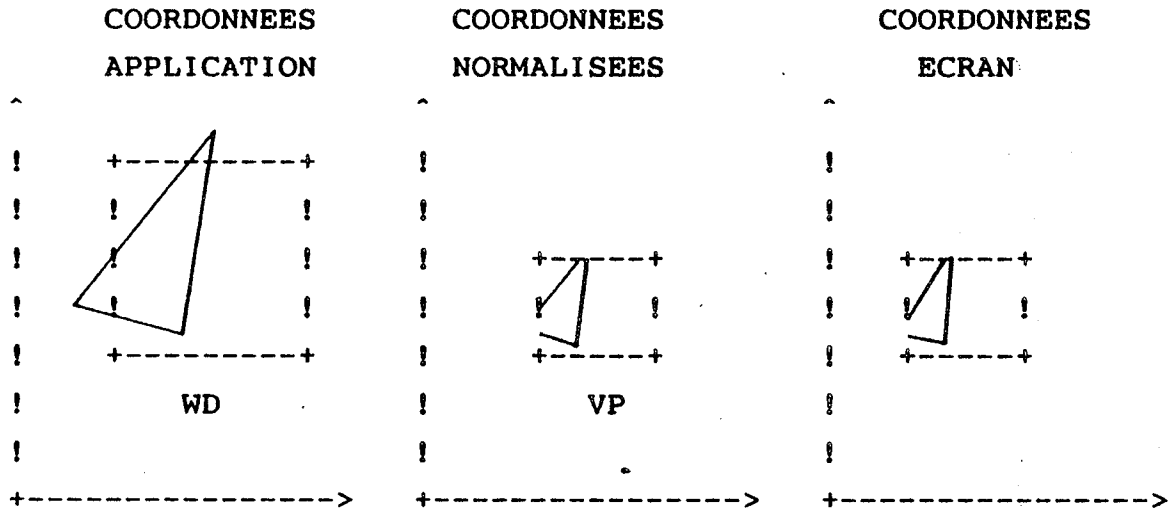
où T1 est la transformation WD -> VP et T2 la transformation VP -> coordonnées du terminal. (1)

---

(1) Remarquons que le résultat d'une désignation subit une transformation inverse ayant pour objectif le passage des coordonnées terminal au système de coordonnées application. C'est le résultat de cette transformation qui est transmis à l'application.

Toute entité graphique, définie par le programme d'application, qui s'inscrit dans sa totalité dans la WD ne subira aucune altération lorsqu'elle sera visualisée dans un VP. Par contre, toutes celles qui ne s'y inscrivent pas totalement, sont tronquées de manière à ce que n'apparaissent dans la VP que les parties de ces entités qui s'inscrivent dans la WD.

De manière très schématique, la visualisation du contenu d'une WD dans un VP est effectuée par une fonction de projection (Foley 82, Encarnacio 80 pour plus de détails) définie dans le logiciel graphique.



Cette fonction de projection effectue les troncations nécessaires en balayant la liste des objets graphiques d'une WS, et en leurs faisant subir les deux étapes de transformations (dans GKS du moins) nécessaires à la visualisation.

Enfin, le rafraichissement de l'écran, configurable par l'application, utilise la liste des segments (et non la liste de toutes les primitives) définis. Suivant la configuration, le rafraichissement est effectué lors de chaque appel à une primitive graphique, avant toute interaction ou sur demande explicite de l'application.

Remarquons que le défilement de l'information dans une VP peut être réalisé par le programme d'application, en déplaçant la WD qui lui est associée sur la description de l'image.

De même, les effets de zoom peuvent être obtenus par modification du rapport de dimensions entre WD et VP (diminution ou augmentation des dimensions de WD autour d'un point de focalisation).

Pour conclure partiellement sur ces logiciels, nous pouvons retenir qu'ils résolvent surtout les problèmes de visualisation des informations et que leur objectif premier est d'assurer aux applications l'indépendance syntaxique par rapport aux terminaux réellement employés.

Leur conception repose sur le principe de la séparation entre la modélisation (du ressort de l'application) et la visualisation (réalisée par le logiciel graphique).

Ils permettent la structuration de l'image sur deux niveaux: la primitive graphique et le segment. Cette structuration est essentiellement employée pour résoudre certains des problèmes liés à la désignation directe.

Une image est décrite comme une succession temporelle d'appels à des primitives graphiques. Celles-ci peuvent être regroupées dans des segments.

Le segment est l'unité de modification d'une image; son contenu n'est pas consultable ni modifiable. La modification d'un segment ne peut être que globale.

De manière plus générale, tous les problèmes liés à la gestion d'une image doivent être résolus par l'application. Les limitations de ces logiciels sont la cause de leur inadéquation aux besoins des applications interactives <Rosenthal 81, Cahn 83>. Ils ne peuvent donc être employés tels quels comme composante

d'une interface usager.

3.3.2. Les logiciels de composition d'image

Nous regroupons sous cette dénomination les logiciels graphiques qui permettent la description de la structure visuelle d'une image. Dans ces logiciels, une image est définie à partir de ses composantes, elles même définies à partir de constituants plus élémentaires jusqu'à ce que le niveau atomique soit atteint <Foley 82>.

Les atomes constituent les éléments prédéfinis du logiciel de composition; ils sont analogues aux primitives graphiques des logiciels de visualisation.

La distinction entre logiciel de visualisation et logiciel de composition réside dans le fait que le logiciel de composition possède une connaissance explicite des relations entre les éléments graphiques primitifs qui constituent une image. Il peut alors tirer partie de cette connaissance pour offrir des primitives de manipulation élaborées aux applications interactives pour en simplifier la réalisation.

De par leur définition, des logiciels tels que CLOVIS <Martinez 82>, NGS <Cahn 83>, et le logiciel graphique du Star <Lipkie 82> amènent l'application à structurer une image sous la forme d'un arbre.

Ils fournissent des primitives de manipulation de haut niveau qui permettent à l'application de structurer l'image et de tirer partie de cette structuration.

Ainsi, lors de certaines manipulations telles que la destruction ou la modification d'attributs de visualisation (changement de couleur, rotation de l'image correspondant à un sous-arbre), l'application n'a plus à connaitre la complexité

effective de la structure manipulée. Ceci lui donne une puissance d'expression accrue.

EXEMPLE

La destruction d'un sous-arbre réalise l'équivalent de la destruction de toutes les primitives graphiques que l'on peut atteindre à partir de sa racine. Ceci, dans le cas des logiciels de visualisation précédemment évoqués, aurait nécessité l'intervention de l'application, la seule à connaitre les liens entre les primitives graphiques.

L'image est décrite par une structure de donnée indépendante du matériel. Elle peut donc être consultée par les applications pour prendre connaissance de la nature des informations qui y sont présentes. La manipulation d'image s'exprime alors comme la manipulation de la structure de donnée associée.

Pour assurer l'orthogonalité du jeu de primitives proposé aux concepteurs d'applications, l'ensemble des opérateurs doit s'appliquer à tous les niveaux de l'arbre. (1)

En fonction de la structure du sous-arbre auquel ils s'appliquent, les attributs (transformations géométriques, définition de l'aspect, ...) peuvent être composés suivant des règles telles que l'héritage, la synthèse ou des mécanismes plus complexes de combinaison de valeurs d'attributs <Mallgren 82>.

---

(1) Ceci les différencie des logiciels de visualisation dans lesquels, par exemple, seul un segment peut être détruit. Il est en effet impossible de détruire une primitive graphique non incluse dans un segment.

De plus, il est impossible de détruire une primitive graphique contenue dans un segment: il faut détruire tout le segment et, le reconstituer sans la primitive que l'on désire supprimer.

Le logiciel graphique peut, quant à lui, tirer profit de cette structuration pour optimiser les traitements liés à la visualisation <Martinez 82>, et recalculer de manière incrémentale les résultats d'une modification d'image (cf. chapitre suivant).

Pour conclure partiellement sur ce type de logiciels, nous pouvons retenir que ceux-ci sont davantage adaptés à nos besoins dans la mesure où ils résolvent certains problèmes liés à la gestion d'image (ils ne se limitent pas à résoudre les problèmes de visualisation). Leur puissance d'expression s'explique par l'homogénéité et la nature des opérations. L'image possède une structure et ces logiciels l'utilisent pour optimiser leurs traitements.

3.3.3. L'interaction

Dans les logiciels graphiques est définie la notion de périphérique logique d'entrée (logical input device). Ces périphériques sont employés par l'application pour réaliser l'acquisition des informations de manière indépendante des particularités de chaque périphérique physique.

Cette notion correspond à un terminal virtuel restreint à l'entrée d'informations.

Cinq classes de périphériques logiques sont définies. A chacune d'elles correspondent des périphériques réels ou simulés. (1) Chaque classe est caractérisée par le type de la valeur (coordonnées, chaines de caractères, points) que retourne le périphérique logique correspondant.

---

(1) Remarquons qu'un même périphérique physique peut appartenir à plusieurs classes. Ainsi la souris appartient aux classes "Locator" et "Pick".

Parmis les types de valeurs on peut distinguer:

- la localisation (locator):

tout périphérique de cette classe retourne comme valeur les coordonnées du point désigné par l'usager à l'aide d'un périphérique physique tel que la souris ou la tablette à digitaliser. Une application emploie ce type de périphérique pour faire l'acquisition du centre et du rayon d'un cercle par exemple.

Dans GKS, les coordonnées du point sont exprimées dans le système de coordonnées de l'application: le logiciel graphique effectue la succession des transformations nécessaires au passage des coordonnées écran à celles de l'application.

Core n'effectue que la transformation coordonnées écran-coordonnées normalisées. C'est l'application qui réalise les transformations nécessaires à l'obtention de coordonnées exprimées dans son référentiel propre. (1)

- l'identification (Pick):

un périphérique de cette classe (une souris par exemple) est employée par l'application lorsqu'elle veut connaitre l'identification de l'objet désigné par l'usager.

Dans les logiciels de visualisation, la valeur transmise à l'application est constituée du nom du segment (si la primitive désignée appartient à un segment) et de l'identificateur (s'il existe) de la primitive graphique qui y est incluse.

---

(1) Pour plus de détails sur les raisons de ces différences consulter <Rosenthal 82, Graphics 77 et GKS 83>.

Dans les logiciels de composition, l'information fournie par un périphérique de cette classe peut être beaucoup plus riche. Ils peuvent en effet transmettre à l'application tout le chemin d'accès qui mène de la racine à l'élément atomique. (1)

- une valeur réelle (Valuator):

l'application emploie ce type de périphérique lorsqu'elle désire que l'usager lui définisse une valeur réelle.

- suite de caractères (String de GKS, Keyboard de Core):

La valeur qui est retournée à l'application est une chaine de caractères dont la taille maximum et la valeur initiale (en standard dans GKS, dépendant du système hôte dans Core) sont précisées par l'application.

Dans Core, des fonctions d'édition simples (effacement du dernier caractère, destruction de la suite des caractères déjà frappés, ...) sont proposées; dans GKS, l'édition débute à partir d'une position initiale précisée par l'application, aucune fonction d'édition autre que la surcharge n'est définie.

- la sélection (Choice de GKS et Button de Core):

---

(1) D'autres possibilités plus élaborées permettent de déterminer la portée de la désignation et lever toute ambiguité (cf 2.3.1). On introduit alors des fonctions de "dialogues" qui exploitent judicieusement la description de la structure de l'image. De telles fonctions étaient jusque là réalisées par les applications;

Ce type de périphérique rend une valeur entière, expression de la sélection de l'usager d'un élément qui appartient à une énumération. Cette classe de périphérique permet l'émulation des menus.

Les périphériques logiques opérent dans des modes. Le mode est le reflet de la méthode employée par l'application pour prendre connaissance de l'information qui résulte d'une interaction avec l'usager.

Le mode détermine à quel moment l'usager peut employer un périphérique. Trois modes de configuration sont possibles:

- Sample: dans ce mode, l'application échantillonne les valeurs instantanées d'un périphérique. L'usager peut employer à tout instant le périphérique configuré dans ce mode. Il n'est cependant pas certain que la valeur déterminée par une telle interaction soit prise en compte par l'application;

- Request: dans ce mode, l'application appelle une fonction de lecture qui rend une valeur du type spécifié par la classe à laquelle le périphérique appartient.

Cette fonction oblige l'usager à employer un périphérique de cette classe et à définir une valeur du type spécifié. La fonction de lecture ne rend de valeur à l'application (et le contrôle) qu'après que l'usager ne lui ait signalé la fin ou l'abandon de l'échange.

Cette configuration correspond au mode synchrone, implicitement employé par la fonction "readln" de Pascal;

- Event: dans ce mode de fonctionnement, le périphérique logique "génère" un "évènement" lors de chaque interaction avec l'usager.

Un évènement est constitué d'une identification de la classe du périphérique employé par l'usager, et de la valeur de l'information déterminée par l'interaction. Cet évènement est mis dans une file d'attente que l'application consulte suivant un rythme qui lui propre.

Ce type de fonctionnement correspond au mode asynchrone.

Ces trois modes de configurations ne sont pas disponibles sur tous les logiciels graphiques: ainsi Core ne définit t-il que les modes Sample et Event, Canvas <Ball 82b> se restreignant au mode Event.

En plus du mode de configuration, tout périphérique est caractérisé par un ensemble d'attributs. Ceux-ci sont fixés lors de l'implantation, ou spécifiés par l'application lors de l'initialisation du périphérique. Ces différents attributs déterminent:

- le message qui sollicite l'usager afin qu'il emploie un périphérique physique (prompt);

- l'écho est le reflet de la valeur entrée par l'opérateur à l'aide du périphérique. Un "commutateur" précise si l'écho doit être effectué par le logiciel graphique ou par l'application. (1)

- la validation: signal particulier que l'usager doit émettre pour signifier au logiciel graphique la fin de l'interaction.

---

(1) Sa présence se justifie par l'existence d'applications (des éditeurs par exemple) qui doivent gérer la relation entre ce qui est visible (sorties) et les informations entrées par l'usager; cette relation n'étant pas prise en compte par les logiciels de visualisation.

Un tel signal ne doit être émis que si le périphérique est configuré en mode Request ou Event;

- la valeur initiale que l'application associe à un périphérique logique;

3.3.4. Les limitations

Nous avons vu l'inadéquations des logiciels de visualisation à nos besoins: ils définissent essentiellement la notion de terminal virtuel, notion que nous intégrons dans le gestionnaire de fenêtres. De plus, ils ne sont pas adaptés à une gestion aisée des informations visibles; ce type de problème est résolu par les logiciels de composition. Ces derniers sont donc plus aptes à répondre à nos besoins.

Le principe de base sur lequel reposent ces logiciels, est la distinction entre modélisation (but de l'application) et visualisation (réalisée par le logiciel graphique).

Il résulte des habitudes de conception des logiciels que la relation entre application et logiciel graphique s'exprime en terme d'utilisation: l'application utilise les services du logiciel graphique qui est alors asservi aux "ordres" de l'application. L'application contrôle tout y compris l'acquisition des informations.

Des fonctions telles que le zoom et le défilement des informations dans une "window" doivent alors être réalisées par l'application en modifiant le rapport WD-VP ou en déplaçant la WD sur la description de l'image.

De par les principes mêmes qui ont présidés à la réalisation de ces logiciels, l'application joue et DOIT jouer un rôle centralisateur: toute facilité donnée à l'usager doit être donnée par l'application. Tout est sous son contrôle et elle doit tout

contrôler.

Si nous voulons donner des facilités à l'usager sans pour autant compliquer la réalisation des applications il nous faudra appliquer des principes différents!

Les logiciels de composition tels que Phigs <Cahn 84>, Ngs <Cahn 83> définissent des concepts différents pour l'entrée et la sortie d'informations en reprenant le concept de périphérique logique.

L'acquisition de l'information ne peut alors être réalisée que par l'application. Ceci a pour conséquence la centralisation du flot de contrôle dans l'application... ce que nous voulons éviter.

La solution à ce problème revient à inclure au niveau du logiciel graphique la notion d'éditeur; il faut alors réaliser l'écho des interactions avec l'usager au niveau du logiciel graphique. La connaissance de la structure visuelle des informations et des relations qui les lient, est une condition nécessaire à la résolution de ce problème.

Une alternative intéressante au problème de l'édition est proposée par le logiciel graphique du Star <LipKie 82> qui exploite, nous l'avons vu, une définition arborescente de l'image. L'originalité de cette réalisation réside dans l'existence d'un éditeur intégré et implicitement activé sur la structure de données sous-jacente à l'image. Les commandes reconnues par cet éditeur sont génériques: elles sont disponibles à tout instant quelle que soit l'application qui s'interface avec ce logiciel.

Une approche similaire est employée par GTX <Nanard 83>. GTX est un éditeur de document: objet électronique qui contient simultanément des zones de texte et de figures (graphique). GTX, qui repose sur le principe des boites <Knuth 79>, est un éditeur structurel qui n'a pas été défini par ses concepteurs comme un

logiciel de composition, bien qu'il en possède certaines caractéristiques.

Ces deux approches montrent qu'il est possible de dépasser le concept d'interaction et de périphérique logique pour proposer celui de résultat d'une édition: résultat d'un ensemble complexe d'interactions gérées par un éditeur; seul ce résultat ayant une importance pour l'application.

Cette approche permet d'établir la liaison entre les informations visualisées et les modification apportée par l'usager à l'image. De plus, elles permettent d'unifier les concepts de périphérique logique <Rosenthal 81>.

Résultat plus intéressant, cette approche permet d'unifier les concepts d'entrée et de sortie des informations: la description de l'image est en effet accessible en "lecture" et en "écriture". Connaitre la valeur qui résulte d'une interaction avec l'usager revient à consulter la nouvelle valeur du sous-arbre qui a été modifié par l'usager.

La "virtualisation" des entrées est alors possible: l'application n'a plus à contrôler l'usage des périphériques. L'existence de l'éditeur étant implicite, les moyens que celui-ci met en oeuvre peuvent être ignorés de l'application.

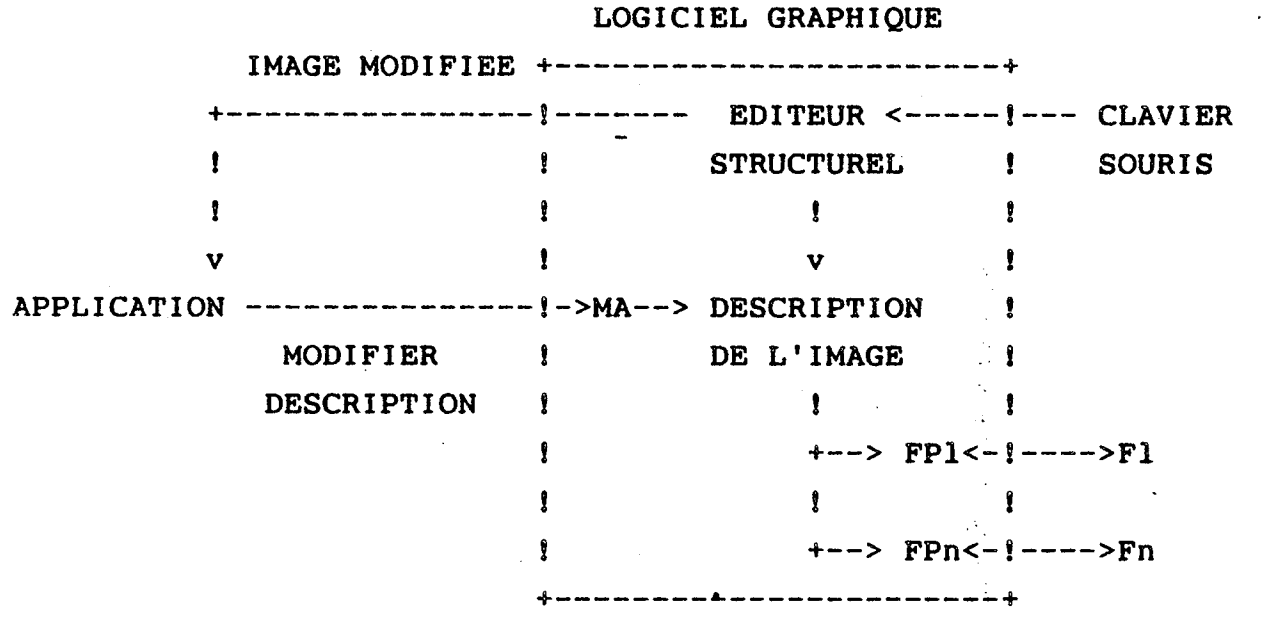
Ceci nous permet à la fois de sortir le flot de contrôle de l'application et de proposer aux usagers un ensemble d'actions réellement génériques (celles de l'éditeur). Dans cette approche nous simplifions aussi la tâche des concepteurs d'applications qui n'aurons plus à réaliser l'acquisition des informations, ni à se poser le problème de l'acquisition des informations.

3.3.5. Propositions

Elles découlent tout naturellement de ce qui précède: le logiciel graphique doit être un logiciel de composition qui intégre un éditeur pour l'acquisition des informations.

Toute fonction de projection possède l'autonomie nécessaire pour réaliser des actions telles que le zoom ou le défilement des informations dans une fenêtre sur demande de l'usager. De telles actions ne modifient pas la description de l'image, elles restent donc masquées à l'application.

Il nous faut alors définir la relation entre le logiciel graphique et l'application. Celle-ci peut être schématisée de la façon suivante:



L'application pour élaborer la description d'une image utilise la méthode d'accès notée MA qui lui cache l'implantation réelle de la structure de données qui décrit l'image. Cette structure de données, arborescente, peut être modifiée et consultée par l'application. Les modifications que cette dernière effectue sont répercutées par les fonctions de projection notées FPi dans les fenêtres Fi qui leurs correspondent.

Plusieurs fonctions de projection peuvent être actives simultanément sur la même description, les paramètres de projections pouvant être différentes pour chacune d'elles.

Pour masquer à l'application la notion de fenêtre, le logiciel graphique ne lui permet d'exprimer que la nécessité de visualiser une partie ou la totalité de l'image. Cette expression induit la création d'une association fonction de projection-fenêtre.

Par la suite l'application peut détruire une telle association, cette destruction se limitant aux associations qu'elle a elle même créées.

Comme l'application, l'usager peut demander la création d'une fenêtre dans laquelle sera visualisée une partie (ou la totalité) de la description de l'image définie par l'application. Pour ce faire, il utilise le moyen de désignation à sa disposition et délimite la partie de la description (sous-arbre) à visualiser. Cette action entraine la création d'une fenêtre et de la fonction de projection associée. Cette création reste masquée à l'application.

Un éditeur structurel est implicitement activé sur la structure de données sous-jacente à toute l'image produite par la fonction de projection. Cet éditeur fournit un ensemble standard de fonctions d'édition.

Pour limiter les actions de l'usager, il est nécessaire de donner à l'application les moyens de protéger certaines informations (le nom d'un champ de formulaire par exemple). A cette fin, un ensemble d'attributs de contrôle est mis à la disposition de l'application.

Chaque fois que l'usager modifie la description de l'image (insertion, destruction, modification d'un élément atomique), l'éditeur génère un évènement qui signale à l'application la modification qui est intervenue. En lui transmettant

l'identification du plus petit sous-arbre modifié, il lui permet de mettre à jour sa représentation interne.

L'éditeur structurel reçoit les informations en provenance des périphériques. Ces informations sont des évènements asynchrones qui lui sont transmis par la fenêtre courante.

Comme nous l'avons déjà dit, la fonction de projection réalise la visualisation d'un sous-arbre. Pour adapter le résultat de cette visualisation, la fonction de projection propose à l'usager des actions telles que le zoom, le défilement vertical et horizontal. En exploitant certaines informations mises par l'application dans la description, il lui est possible de présenter l'image qui y correspond avec différents niveaux de détails.

3.4. Le gestionnaire de dialogue

La notion de gestionnaire de dialogue est issue de la nécessité de proposer à l'usager des moyens de communication adaptés à ses besoins et de faciliter la réalisation des applications interactives en les déchargeant de l'acquisition des commandes.

L'approche initiale a consisté en la définition d'un ensemble d'outils d'acquisition spécialisés chacun dans une technique d'interaction. (1) Ils gèrent <Perq 82> ou permettent à une application de gérer <Knox 78> les périphériques nécessaires à la réalisation d'une succession d'interaction avec l'usager.

---

(1) Une technique d'interaction est définie par Foley <Foley 81> comme étant une méthode d'utilisation d'un périphérique pour entrer un certain type d'information, couplée à la forme la plus simple d'écho.

EXEMPLE

L'utilitaire "PopMenu" du Perq gère un menu. Pour réaliser une interaction, l'application fait appel à l'utilitaire qui lui retourne pour résultat le numéro de l'élément désigné par l'usager.

Dans l'architecture qui résulte de ce type de démarche, le flot de contrôle réside dans l'application <Rosenthal 81>: la succession des interactions est déterminée lors de la spécification de l'application et les appels aux outils d'interaction résident dans le corps de l'application.

La conséquence immédiate en est le contrôle explicite de la forme de la communication par le logiciel; ceci se traduit pour l'usager par une certaine rigidité dans le dialogue <Kasik 83>.

Pour palier ce problème, une nouvelle approche est nécessaire. Ses principaux objectifs sont:

- sortir le contrôle de l'application: celle-ci ne contrôle plus la succession des interactions et elle est indépendante de la forme effective de la communication;

- donner à l'usager une certaine souplesse dans la communication;

- réaliser l'acquisition des informations de l'application;

- effectuer l'analyse syntaxique et sémantique (syntaxe contextuelle) des commandes.

3.4.1. Les approches

Les propositions actuelles <Hanau 80, Borufka 82, Kasik 83, Kamran 83, Olsen 83a> tendent à séparer la partie algorithmique

d'une application de la partie qui gère le dialogue avec l'usager. Appelons ce module un "Gestionnaire de Dialogue".

Dans les architectures qui résultent de ces propositions, le contrôle est dit externe à l'application: la succession des interactions nécessaires à la constitution d'une commande est hors du contrôle de l'application. Cette succession est décrite dans un formalisme ad-hoc qui sert au gestionnaire de dialogue à gérer les interactions avec l'usager et à effectuer des vérifications syntaxiques et quelquefois sémantiques.

Le séquencement des traitements est induit par le gestionnaire de dialogue; il se déroule en trois étapes:

- une phase d'acquisition des informations sous la forme d'une succession d'interactions déterminée par le gestionnaire; les vérification syntaxique et sémantique sont faites conjointement à l'acquisition.

  Lorsque la commande est considérée comme valide par le gestionnaire de dialogue, c'est à dire quand toutes les vérifications qu'il a effectuées se révèlent positives, les deux étapes suivantes sont réalisées.

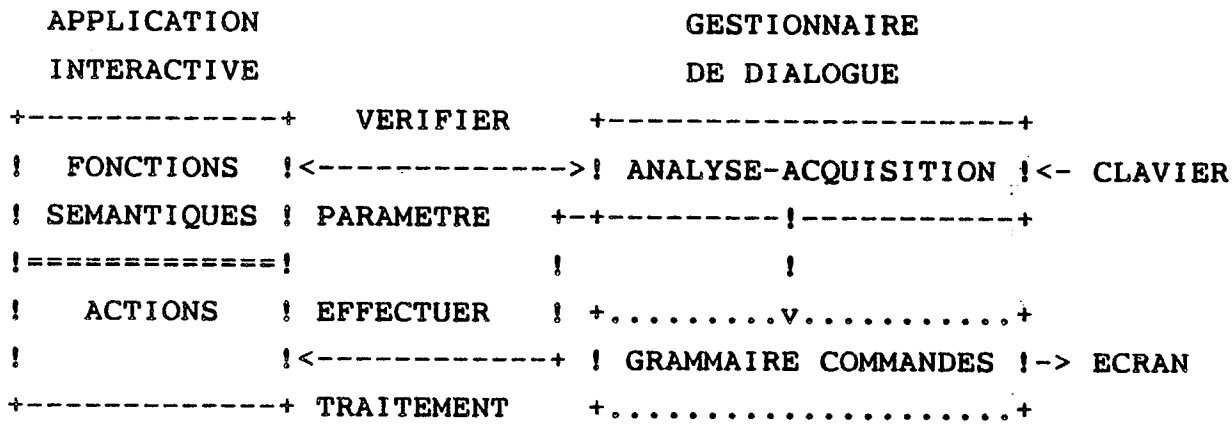
  Dans le cas contraire, le gestionnaire fait savoir à l'usager que sa commande est erronée, et lui demande de la modifier. Remarquons que cette méthode assure aux applications que toute commande activée puisse s'exécuter sans erreur dûe à l'usager;

- une phase d'exécution de la commande par une fonction de l'application: le gestionnaire de dialogue active cette fonction et lui passe les paramètres nécessaires à son exécution;

- une phase de visualisation des informations qui résultent de la phase précédente. L'application répercute sur la description

de l'image les modifications qu'elle a pu apporter à sa représentation interne.

L'architecture qui résulte de ces propositions peut être schématisée de la façon suivante:



L'idée de base est d'employer une approche grammaticale pour contrôler le dialogue avec l'usager <Hanau 80>. Ceci permet au concepteur d'application de définir un dialogue comme une combinaison de "sous-dialogues".

L'analogie avec la définition syntaxique d'un langage de commande est immédiate: tout non terminal d'une grammaire s'exprime comme une combinaison de règles plus élémentaires, jusqu'à ce que le niveau lexical soit atteint.

A tout non terminal de la grammaire on fait correspondre un dialogue élaboré défini par un ensemble d'interactions; à un élément lexical on fait correspondre une interaction au sens défini précédement.

A partir de la spécification d'une syntaxe, l'analyseur du langage correspondant peut être généré de manière automatique <Boullier 80> et des méthodes d'analyses peuvent être employées pour accepter une syntaxe "floue" <Hayes 80, Hayes 81>.

La liaison avec les vérifications sémantiques (syntaxe contextuelle) est obtenue en précisant dans la spécification syntaxique les fonctions à activer dans chaque règle.

Dans ce schéma, l'application se résume à un ensemble d'actions et de fonctions sémantiques qui sont appelées par l'analyseur syntaxique au fur et à mesure de sa progression dans l'analyse d'une commande.

Nous pouvons distinguer les fonctions sémantiques dont l'objectif est de vérifier si les valeurs des paramètres appartiennent au domaine des valeurs spécifiés par l'application, des actions sémantiques qui représentent l'implantation des algorithmes réalisant les traitements correspondant aux commandes de l'application.

L'intérêt de cette démarche est donc évident: en définissant une interface de dialogue suffisament générale il est possible de générer automatiquement de telles interfaces et réduire ainsi les coûts de production des logiciels interactifs.

3.4.2. Les limitations et leurs causes

Compte tenu des outils sur lesquels les analyseurs s'appuyent (des logiciels de visualisation), la spécification du dialogue comporte des indications sur les périphériques logiques que l'usager doit employer. Ainsi, ce qui correspond aux éléments lexicaux dans une définition de langage a pour équivalent, dans une spécification de dialogue, des noms de périphériques ou des éléments lexicaux dont le type détermine le périphérique logique que l'usager doit employer.

Il résulte des techniques employées que les demandes d'interaction sont effectuées conjointement au parcours des règles qui gouvernent le dialogue, c'est à dire pendant l'analyse syntaxique.

Pour l'usager le dialogue se réduit alors à une succession d'interactions dont l'ordre est déterminé par la grammaire. Le flot de contrôle, bien qu'externe à l'application est alors inclu dans le gestionnaire de dialogue.

Notre problème est d'offrir une certaine souplesse et des formes de dialogues adaptées à plusieurs "catégories" d'usager. Pour ces différentes formes, acquisition et analyse ne sont pas indépendantes. Leur "couplage" dépend de la forme de la communication.

EXEMPLES

- Dans une technique de communication telle que le "menu", le couplage fort entre l'analyse et l'acquisition est nécessaire.

En effet, les différents éléments d'un menu hiérarchisé, qui définissent les choix possibles à un instant donné, sont déterminés par l'état courant de l'automate d'analyse dans la grammaire du dialogue en cours. Le nombre des choix étant restreint, et le périphérique que l'usager doit employer étant connu d'avance (pick ou choice), la nécessité de découpler l'analyse de l'acquisition ne se fait pas sentir: la succession des interactions est connue et l'aspect séquentiel est recherché pour aider l'usager. Syngraph <Olsen 83a>, Ticcl <Kasik 83> proposent une technique de communication unique de type "menu".

- Dans le cas du formulaire, technique que nous avions décrite en 2.3.2 comme offrant plus de souplesse à l'usager, la succession des interactions ne doit être connue a priori.

A partir de la grammaire du langage, il nous faut alors déterminer l'ENSEMBLE et le TYPE des interactions nécessaires au remplissage du formulaire; l'ordre dans lequel les différents champs sont complétés dépend de l'usager <Hayes 84>.

Dans un formulaire, il peut en effet exister des champs dont l'existence est conditionnée par la valeur d'un ou plusieurs autres champs. Par exemple, dans un formulaire de type administratif, le champ "nom de jeune fille" ne peut être complété que si les champs "marié" et "sexe" ont respectivement pour valeur "oui" et "féminin". Le champ "nom de jeune fille" ne doit apparaitre qu'à partir du moment où les deux champs ont les valeurs précédement spécifiées.

Le couplage entre acquisition et analyse doit alors être faible, mais néanmoins exister. Pour obtenir la souplesse recherchée, un analyseur incrémental est nécessaire. Cette nécessité a été clairement ressentie lors de la définition d'éditeurs syntaxiques <Rouzaud 84, Notkin 84> qui utilisent par ailleurs une technique d'édition très proche de celle du formulaire.

Le mécanisme nécessaire à ce découplage, et qui donne à l'usager une certaine souplesse est inexistant dans les gestionnaires de dialogue précédement cités.

- Dans un langage de commande classique, l'usager détermine une commande par une succession d'interactions qui doivent suivre une certaine syntaxe.

C'est au moment où la "phrase" est considérée comme complète que l'usager envoit un signal de validation; la phrase est alors analysée.

Dans ce dernier cas, le découplage entre l'analyse syntaxique et l'acquision des informations est total.

La difficulté éprouvée par les concepteurs des interfaces usager examinées est dûe au fait que toute souplesse donnée à l'usager, c'est à dire toutes les transitions admissibles, doivent être décrites dans la grammaire du langage de dialogue <Olsen 83a>. Comme l'analyse est séquentielle, il faut prévoir au

niveau de l'analyseur des possibilités de retour arrière si l'analyseur est piloté par une table <Hanau 80, Buxton 83, Olsen 83b>, ou décrire l'ensemble des transitions admissibles si la syntaxe est décrite par un réseau de transitions <Kamran 83>.

Pour résoudre ce type de problèmes des méthodes d'analyses plus complexes sont alors nécessaires ("un automate d'états récursif" pour <Kamran 83>, technique dont les limitations sont montrées dans <Hayes 84>).

3.4.3. Propositions

Pour résoudre les problèmes du gestionnaire de dialogue, nos propositions s'appuient sur l'existence du logiciel graphique (avec l'éditeur intégré) précédemment défini.

Nous souscrivons totalement à une approche grammaticale pour l'ANALYSE syntaxique et sémantique des commandes.

Le couplage entre l'acquisition (éditeur du logiciel graphique) et l'analyseur est dépendant de la forme de la communication entre l'usager et l'application. Il est cependant possible que des outils de dialogue bien étudiés ne nécessitent pas ce "couplage variable". La syntaxe d'un langage de commandes est en effet souvent très simple et nous pensons qu'il est possible de tolérer que des erreurs ne soient pas immédiatement détectées si l'usager peut modifier le texte initial de sa commande. (1)

La relation entre le gestionnaire de dialogues et l'application est calquée sur celles que nous avons vues précédemment. Des extensions devront néanmoins êtres prévues pour pouvoir offrir à

---

(1) Dans le cas du menu, la situation est différente: un menu bien étudié ne permet pas de construire des phrases syntaxiquement incorrectes.

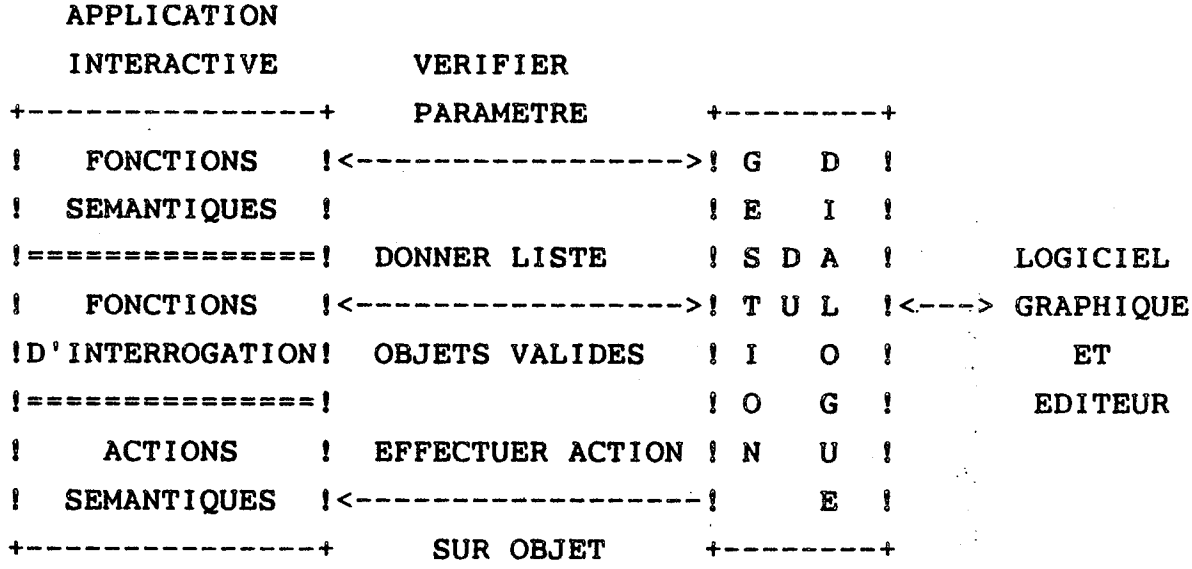
l'usager des modes de dialogues cohérents.

EXEMPLE

Prenons un système de gestion de fichier. Si le dialogue est de type menu, il faut qe l'usager puisse effectuer une action sur un fichier en le sélectionnant dans un menu. La liste des fichiers existants n'est pas descriptible statiquement; il est alors nécessaire que le gestionnaire de dialogue puisse demander cette information à l'application.

Celle-ci doit alors lui fournir cette liste dans un "format" exploitable par le gestionnaire de dialogue.

L'architecture résultante peut être schématisée de la manière suivante:



3.5. Conclusion

L'architecture que nous proposons repose sur les composants traditionnels des interfaces usager: le gestionnaire de fenêtres, le logiciel graphique et le gestionnaire de dialogues.

Chacun de ces composants résoud des problèmes spécifiques, propose les mécanismes d'adaptation nécessaires à l'usager et possède une certaine autonomie (compétence).

Le gestionnaire de fenêtres gère la partition de l'écran en fenêtres. Chacune d'elles est un terminal virtuel qui assure au logiciel graphique l'indépendance syntaxique par rapport aux terminaux réellement employés.

La fenêtre est un objet manipulable par l'usager, il peut la déplacer, la créer, la détruire et en modifier les dimensions.

Le logiciel graphique résoud les problèmes de gestion des informations visibles, de projection de la description de l'image dans plusieurs fenêtres et d'acquisition des informations. La structure de l'image est décrite par un arbre. Les relations entre les différents constituants de l'image sont connues au niveau du logiciel graphique.

La connaissance de la structure et des relations est nécessaire pour effectuer l'écho des informations qui résultent des interactions avec l'usager. Nous pouvons alors intégrer au niveau du logiciel graphique des fonctions d'édition élaborées. Ces fonctions sont présentes quelque soit l'application; elles permettent de définir des actions génériques.

De par la relation qui existe entre le logiciel graphique et l'application, correspondent à ces actions d'édition les fonctions de l'application qui permettent le maintien de la cohérence entre la représentation interne d'un objet et l'image qui lui est associée. Le protocole de communication entre le logiciel graphique et l'application force l'existence de telles fonctions.

Le logiciel graphique réalise l'interface entre l'application ou le gestionnaire de dialogue et le gestionnaire de fenêtres. Il masque la notion même de fenêtre aux couches qui lui sont hiérarchiquement supérieures. La relation entre le logiciel

graphique et le gestionnaire de fenêtres repose sur une coopération.

Le gestionnaire de dialogue effectue l'acquisition et l'analyse des commandes de l'application. Il propose à l'usager trois formes de communications et assure l'indépendance de l'application par rapport à la forme effective.

Sa réalisation s'appuie sur l'existence d'un logiciel graphique élaboré qui lui masque les problèmes d'acquisition des informations. Le couplage entre l'acquisition des informations et l'analyse est éventuellement fonction du mode de communication avec l'usager (une phase d'expérimentation sera nécessaire pour déterminer la faisabilité et l'utilité du couplage variable).

L'indépendance de l'application face à certaines actions de l'usager et la résolution du problème de la centralisation du contrôle par l'application a été obtenue en laissant à chaque couche une autonomie propre non contrôlée par l'application; chacune d'elles donnant à l'usager des fonctions pour manipuler les abstractions qu'elle définit.

L'interface usager dans Adèle

Le principal problème qui s'est posé lorsque nous avons débuté la spécification de cette interface était la multiplicité des contraintes et des techniques que l'étude de l'état de l'art nous engageait à prendre en considération.

L'analyse des études ergonomiques nous amenait quelquefois à vouloir respecter des contraintes contradictoires. Ainsi, par exemple, guider l'usager pas à pas dans l'utilisation d'une application mais aussi lui offrir un maximum de souplèsse dans son emploi.

L'étude des différentes techniques de communication nous a montré que chacune d'elles présente des avantages ou des inconvénients suivant le type de l'usager concerné. Ce qui peut être considéré par un usager comme un avantage à un instant donné, peut être perçu comme trop contraignant quelque temps après.

Des considérations de cette nature nous ont amené à reconsidérer le problème et à énoncer les principes suivants:

- il n'existe a priori aucun "cocktail" de contraintes ergonomiques permettant d'obtenir une interface reconnue comme "bonne" par une grande diversité de personnes,

- l'usager est une personne qui possède une individualité et qui, en conséquence, possède ses propres critères pour définir ce qu'est une application facilement utilisable. L'interface usager doit être un logiciel adaptable par l'utilisateur mais qui doit respecter certaines contraintes ergonomiques élémentaires,

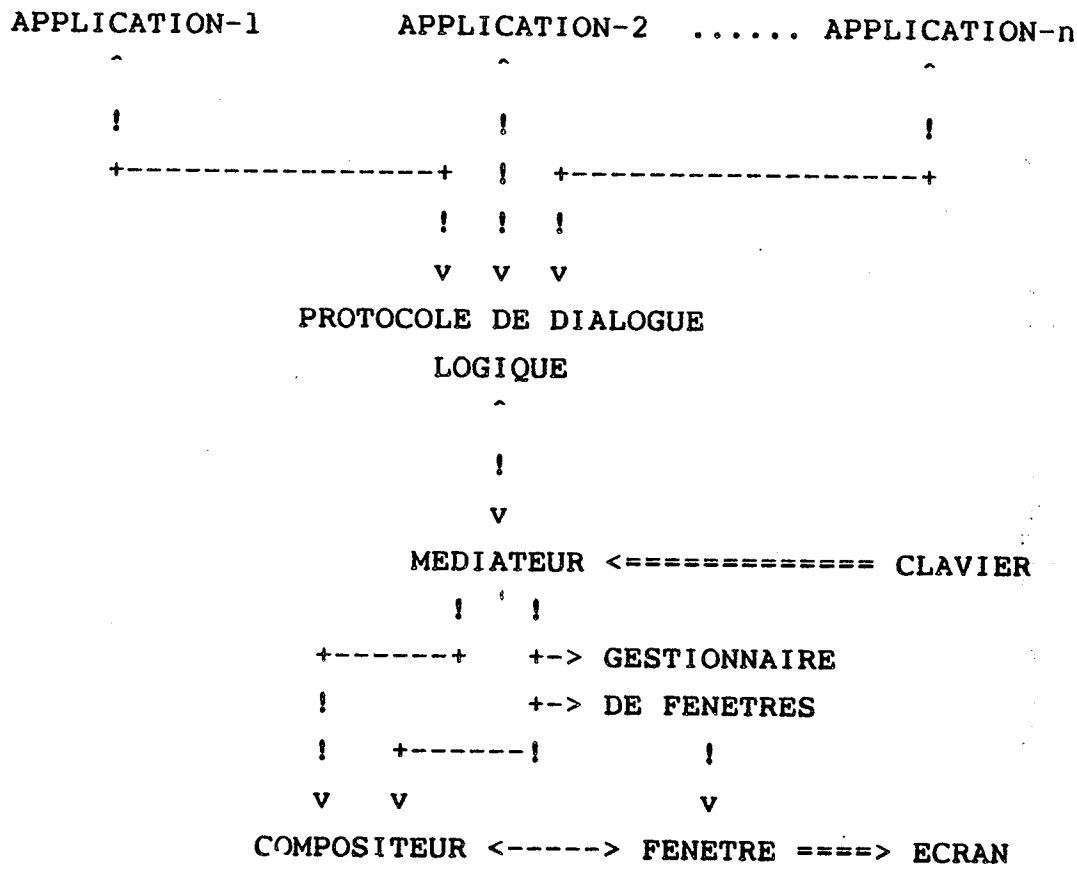
- les mécanismes d'adaptation, autres que ceux de la composante

conceptuelle de l'application, doivent rester transparents à cette dernière,

- les abstractions définies par l'interface usager constituent un protocole de communication entre l'application et l'interface qui est une représentation virtualisée du dialogue effectif entre l'interface et l'usager. L'interface usager doit adapter ce dialogue simplifié à une forme de dialogue élaborée, répondant aux besoins de l'usager,

- les techniques et outils permettant la réalisation d'une interface usager existent. Les interfaces actuelles visent à satisfaire une classe d'usager donnée, sans tenir compte du fait que l'usager évolue dans le mode d'emploi d'un outil informatique. Elles sont figées et exploitent de manière insuffisante les potentialités des techniques connues. Ceci est dû au manque de cohésion dans l'utilisation de ces techniques et au bas niveau des abstractions qu'elles offrent dans certains cas.

L'interface usager définie dans le cadre du projet Adèle <Coutaz 84> suit une approche basée sur ces principes. Elles renferme les composants traditionnels nécessaires à la définition de toute interface usager: un gestionnaire de dialogue appelé Médiateur, un outil graphique nommé Compositeur et un gestionnaire de fenêtres. Son architecture est décrite par le schéma ci-dessous.



Le "protocole de dialogue logique" (cf. 1) modélise l'échange des informations entre l'application et l'interface usager. Pour le définir, nous avons dû émettre quelques hypothèses sur le fonctionnement interne d'une application.

Le MEDIATEUR (cf. 2) offre à l'usager des moyens de communication cohérents avec les différentes applications du poste de travail. Il définit un modèle d'interaction souple et adaptable, tout en assurant aux applications l'indépendance face au modèle d'interaction effectif.

Le Médiateur remplit trois fonctions <Coutaz 84>:

- c'est un gestionnaire de dialogues. Comme tel, il traduit les interactions avec l'usager dans les termes du protocole de dialogue logique et vérifie la validité syntaxique des commandes,

- c'est un gestionnaire d'activités. Chaque activité est une application que l'usager peut sélectionner en déplaçant par exemple son curseur dans la fenêtre où elle visualise ses résultats. Le Médiateur interprète une telle interaction comme un changement d'activité,

- enfin, il centralise toutes les interactions et les aiguille vers les autres composantes de l'interface.

Remplissant à la fois le rôle de gestionnaire de dialogue, de gestionnaire d'activité et d'aiguilleur d'informations, il possède dans notre système un statut particulier. Les raisons de cette particularisation sont évoqués en 2.4.

Le COMPOSITEUR (cf. 4) est l'outil graphique de l'interface. Il permet aux applications de décrire à un niveau logique la structure et l'aspect visuel des informations. Il exploite les possibilités du multi-fenêtrage en autorisant par exemple la visualisation de la description d'une image sous des formes distinctes dans différentes fenêtres.

Il définit une structure d'échange unique qui est un arbre dont les noeuds sont appelés BOITES. Cette structure d'échange ou RE est constituée et modifiée sous le contrôle de l'application "propriétaire" par l'utilisation d'un ensemble de fonctions associées aux boites.

Le maintien de la cohérence entre la RE et le(s) support(s) de visualisation est réalisé par un module d'affichage qui résoud les problèmes liés à la visualisation et au partage d'une RE entre plusieurs fenêtres. Ce module, ignoré des applications n'est accessible qu'au Compositeur.

Sur les atomes de la RE (du texte dans l'état actuel de notre réalisation) sont définies des fonctions d'édition élémentaires. Elles permettent à l'usager d'opérer directement sur les informations qu'il voit apparaitre dans une fenêtre.

Le GESTIONNAIRE DE FENETRES (cf. 3) gère la partition de l'écran en fenêtres. Il définit l'ensemble des fonctions qui permettent à l'usager d'organiser les informations sur l'écran.

La FENETRE est la réalisation d'un terminal virtuel qui assure au module d'affichage l'indépendance syntaxique face à une certaine classe de terminaux, alpha-numériques dans notre réalisation. Elle définit les fonctions standards pour ce type de périphériques.

REMARQUES:

L'architecture de l'interface usager que nous avons définie est différente de celle que nous proposions dans le chapitre précédent. Cette différence est dûe au fait que la nécessité de formuler le problème à l'aide de processus coopérants n'est devenue "évidente" qu'après l'analyse des problèmes rencontrés et des techniques mises en oeuvre.

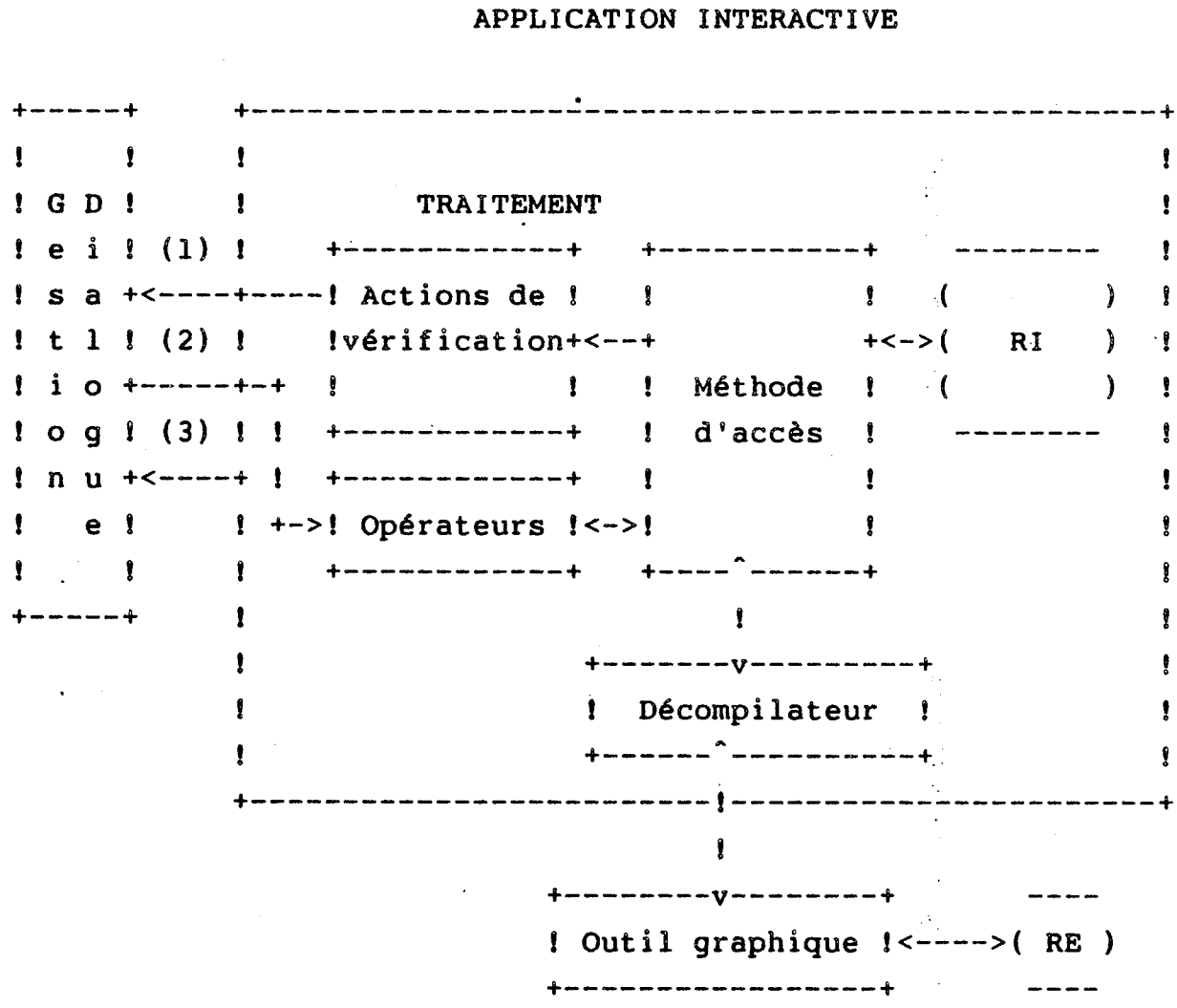
D'autre part, le système (Multics) sur lequel l'interface a été développée associe un processus et un seul à chaque utilisateur. Il est alors impératif que l'ensemble application-interface s'exécute comme un processus unique; ceci nous contraignait de toute façon à une réalisation séquentielle.

Dans ce qui suit, nous présentons l'ensemble des composants de l'interface réalisée. Nous commençons par définir la relation application-interface usager et présentons les caractéristiques du Médiateur. Puis, nous expliciterons le gestionnaire de fenêtres, et enfin les principes du Compositeur.

1. La relation application-interface usager

1.1. La modélisation de l'application

Dans notre approche, l'application doit être constituée de trois parties opératoires et d'une structure de données.



(1) représente l'appel aux actions de vérification,
(2) l'activation des opérateurs
(3) l'expression des changements d'état de l'application.

La RI représente l'objet sur lequel opère l'application. Cette structure est constituée d'éléments prédéfinis, que nous appelons des modèles. Ceux-ci sont instanciés par l'usager pour créer un

objet (un programme par exemple).

Les parties opératoires sont celles que nous appelons "traitement", "méthode d'accès" et "décompilateur". Certaines d'entre elles ("traitement" et "décompilateur") sont en relation directe avec les couches "gestion dialogue" et "outil graphique" de l'interface usager.

Le "traitement" correspond à la composante sémantique de l'application interactive. Elle est représentée par des actions sémantiques qui appartiennent à l'un des deux types suivants:

- les actions de vérification, permettant de vérifier la validité des paramètres des opérateurs (syntaxe contextuelle),

- les opérateurs qui représentent les traitements que l'usager peut effectuer.

La relation entre la composante "traitement" et l'interface usager repose sur l'existence d'un schéma exécutif inclus dans l'interface usager (cf 1.2). Lorsqu'une commande émise par l'usager est valide, le gestionnaire du dialogue active l'opérateur qui lui correspond.

La "méthode d'accès" permet aux opérateurs d'accéder à la représentation interne d'un objet, sans avoir à connaitre sa représentation effective. Elle définit l'ensemble des fonctions de consultation et de modification de la RI.

Les opérations de modification possèdent deux objectifs distincts. Le premier est d'opérer des transformations sur la représentation interne et le second est d'assurer le maintien de la cohérence entre la RI et la description de l'image correspondante (RE).

Les opérations de modification sont chargées en exclusivité du maintien de la cohérence de la RI; ce sont elles par conséquent

qui communiquent au décompilateur les changements intervenus pour qu'il réalise les modifications correspondantes sur la RE.

Le "décompilateur" possède donc la connaissance de la RI par le biais des opérations de consultation et de la RE via les opérateurs du logiciel graphique. Le "décompilateur" peut être vu comme un traducteur incrémental RI-RE.

Le logiciel graphique peut, pour sa part, être vu comme une méthode d'accès à la RE. Il cache en effet la structure de données qui décrit une image et les problèmes liés à la visualisation des informations.

Remarquons que dans la définition du décompilateur, nous avons implicitement supposé que la RE était la description de l'ensemble de la RI. N'ayant pas à prendre en compte la notion de visibilité de l'objet, la réalisation du décompilateur s'en trouve simplifiée.

Quand une commande arrive au terme de son exécution, l'application peut changer d'état: l'ensemble des commandes autorisées peut être différent de celui de l'état qu'elle vient de quitter. Certaines commandes sont invalidées tandis que d'autres deviennent disponibles. Dans l'implantation courante de notre interface ce changement d'état est exprimé par l'application.

La solution alternative est de décrire statiquement l'ensemble des états et des transitions de l'application. Cette description permettrait à l'interface de déduire les changements d'état de l'application, en fonction des commandes activées par l'usager.

1.2. L'usager logique et le protocole de dialogue logique

L'usager logique est un représentant simplifié et non exigeant de l'usager réel. Il définit en quelque sorte un "usager virtuel". Le dialogue que l'application entretient avec l'usager logique se

matérialise par ce que nous appelons le "protocole de dialogue logique". Il constitue une forme canonique du dialogue réel entre l'interface usager et un utilisateur.

Ce protocole de dialogue repose sur les hypothèses suivantes:

- l'application manipule des objets et reconnait un ensemble d'opérateurs,

- l'application distingue les modèles des instances d'objets. L'instruction IF est un exemple de modèle tandis que "IF a>b THEN a:=b" est, pour l'application "éditeur syntaxique", une instance de ce modèle,

- les opérateurs ont au plus un argument. Les opérateurs qui nécessitent plusieurs opérandes sont considérés comme des opérateurs sans argument. Ils demandent interactivement à l'usager logique de spécifier les opérandes,

REMARQUE:

Ce procédé a pour conséquence d'inclure une partie du flot de contrôle dans l'application; ce que nous voulions éviter (cf. chapitre 1).

- l'application possède des opérateurs spécifiques et des opérateurs génériques. La technique retenue dans l'interface Adèle est de proposer un ensemble d'actions a priori communes à l'ensemble des applications du poste de travail. Les noms et la sémantique de ces commandes ont été figés. Nous avons suggéré aux concepteurs des applications de définir les opérateurs correspondants. Il est cependant possible à une application d'invalider les opérateurs génériques qu'elle ne reconnait pas.

Le protocole de dialogue logique repose sur le principe que le gestionnaire de dialogue (le Médiateur) connait à tout instant les objets et les opérateurs valides dans l'état courant de

l'application. Ainsi, l'application en cours de session indique ses changements d'état en validant, invalidant, créant ou détruisant des objets et des opérateurs.

Le protocole de dialogue logique est composé de quatre requêtes principales à destination de l'application:

- "Initialisation" correspond à l'activation d'une application par le Médiateur. Elle demande à l'application d'initialiser ses structures de données et de déclarer à l'interface usager l'ensemble des modèles et des opérateurs disponibles. A cette occasion, l'application "hérite" des opérateurs génériques et ordonne l'invalidation de ceux qu'elle ne reconnait pas.

- "Fin de session" indique à l'application la fin de son utilisation. Celle-ci effectue alors les sauvegardes nécessaires.

- "Quel modèle?" permet au Médiateur d'interroger l'application sur le type ou modèle d'un objet donné. De la connaissance du modèle, le Médiateur en déduit les opérateurs que l'usager peut employer.

- "Analyse" est la requête adressée à l'application pour qu'elle effectue une certaine opération sur un objet donné. La réponse de l'application reflète son nouvel état. Elle y précise le message à transmettre à l'usager (s'il y a lieu), la création éventuelle de nouvelles fenêtres, les informations qui doivent y être visualisées et enfin, s'il est nécessaire de lancer une nouvelle activité pour le compte de l'application courante.

Le Médiateur définit le modèle d'interaction entre l'usager et l'interface. Il traduit le dialogue réel mené par l'usager en termes du protocole de dialogue logique et réciproquement.

2. Le Médiateur

Le Médiateur propose un modèle d'interaction qui imite l'environnement et les actions que l'usager connait dans le monde réel. Il définit des mécanismes d'adaptation qui permettent à l'usager de personnaliser son environnement de travail et il construit des objets en utilisant les concepts de fenêtre et de boite.

2.1. Le modèle d'interaction

Le modèle d'interaction proposé par le Médiateur est évolutif. En effet l'usager, informaticien dans le cadre de notre projet, passera rapidement du statut de "novice" à celui d'"expert". Aussi le Médiateur définit plusieurs formes de dialogues simultanément accessibles à l'usager. S'il sélectionne une forme privilégiée, il pourra néanmoins passer aisément à un autre mode de dialogue.

Toute commande émise par l'usager obéit systématiquement au paradigme:

Désignation d'un objet, PUIS désignation d'un opérateur.

Dans un souci de cohérence, la désignation d'un objet ou d'un opérateur s'effectue de la même façon. Trois procédés sont disponibles: la sélection d'une représentation visuelle, la frappe d'un texte et l'utilisation de touches synonymes.

Illustrons ces procédés sur l'exemple de la destruction d'un objet "IF":

- sélection d'une représentation visuelle:

  l'usager désigne l'objet IF présent sur l'écran, puis le texte "détruire" qui apparait dans la fenêtre menu des opérateurs.

Remarquons que si la fenêtre dialogue "Formulaire" est présente sur l'écran, la suite des interactions précédentes a pour résultat de compléter le formulaire qui y apparait. Ainsi verra t-on le champ OBJET prendre pour valeur "IF", et le champ OPERATEUR prendre la valeur "détruire";

- frappe d'un texte équivalent:

l'usager se positionne dans la fenêtre dialogue et complète le champ OBJET en lui affectant la valeur "IF". IF désignant un modèle, le Médiateur demande à l'usager quelle est l'instance de modèle qui est concerné.

Cette ambiguité levée, l'usager complète le champ OPERATEUR à l'aide du clavier et lui affecte la valeur "détruire";

- utilisation de touches synonymes:

le clavier est partitionné en trois "sous-claviers". Nous distinguons:

- le clavier des objets qui comprend l'ensemble des touches logiques qui désignent les objet-modèles. Nous avons fait correspondre à ce clavier les séquences de caractères préfixées par le caractère "escape". Ainsi, le modèle "IF" est désigné par la séquence "Esc if",

- le clavier des opérateurs, constitué de l'ensemble des touches logiques qui désignent des opérateurs. Nous lui avons fait correspondre une séquence de caractères de contrôle du clavier. L'opérateur détruire est désigné par "Ctrl d",

- le clavier standard qui correspond aux touches ordinaires employées pour la frappe de texte.

Remarquons que ces trois modes de dialogues constituent des

formes de base que l'usager peut employer simultanément.

Ainsi, il peut désigner l'objet à l'aide d'une souris, puis frapper "Ctrl d" qui désigne l'opérateur détruire. Le résultat aurait été identique s'il s'était positionné dans le champs OPERATEUR du formulaire en y entrant "détruire" ou ... "Ctrl d".

Il y a donc redondance dans les moyens de désignation. Bien que n'accroissant pas les fonctionnalités, celle-ci offre à l'usager la souplesse bien souvent absente dans les interfaces usager traditionnelles.

L'utilisateur ayant défini sa commande, le Médiateur l'analyse immédiatement et détecte les erreurs de nature syntaxique et certaines des erreurs contextuelles (application d'un opérateur sur un objet de type non compatible au paramètre par exemple).

En cas d'erreur, l'usager modifie la commande qu'il a émise en changeant par exemple d'opérateur. Il se positionne sur le champ OPERATEUR et en modifie la valeur. L'autre solution est de reformuler toute la commande. Cette possibilité bien que souvent perçue comme contraignante lui reste cependant acquise.

Lorsque la commande est considérée comme complète et correcte, le Médiateur la traduit dans les termes du protocole de dialogue logique en employant les requêtes décrites en 1.2. Pour l'application tout se passe comme si l'usager avait émis une commande en respectant une succession d'interactions parfaitement figée.

Dans ce sens, le Médiateur effectue l'adaptation entre le protocole de dialogue logique très rigide qui simule une succession d'interactions pré-déterminée et un dialogue réel pour lequel ni la forme, ni la succession effective des interactions n'est connue a priori.

Ne sont définies statiquement que les formes élémentaires du

dialogue avec l'usager. La facilité d'utilisation est obtenue en lui permettant de passer implicitement d'une forme de dialogue à une autre, en changeant simplement la formulation des requêtes.

Ce premier mécanisme permet à l'usager d'adapter la forme du dialogue à ses besoins. Le dialogue ne constitue qu'un aspect de l'interface usager; d'autres mécanismes d'adaptation sont alors nécessaires pour modifier l'environnement de travail.

2.2. L'adaptation

L'environnement que le Médiateur réalise pour l'usager est constitué d'un ensemble d'objets et d'opérateurs. Toute action se traduit par l'application d'un opérateur sur un objet. Dans le Médiateur, tout objet peut être considéré selon trois points de vue: comme une entité globale, comme un contenu et selon ses propriétés intrinsèques.

- une icone (dessin évocateur ou à défaut un texte court) représente l'objet dans sa totalité;

- l'opérateur universel "ouvrir" donne accès au contenu de l'objet. L'opérateur générique "fermer" réduit l'objet et son contenu à l'état d'icone;

- l'opérateur générique "montrer propriétés" fait apparaitre dans une fenêtre la liste des propriétés d'un objet.

Les objets tout comme les opérateurs possèdent des listes de propriétés. Celles-ci sont matérialisées par des attributs qui peuvent appartenir à l'un des deux types suivants:

- les attributs spécifiques à l'application, leur sémantique n'est connue que de celle-ci. Chaque attribut est défini par l'application comme une structure de données visualisable et manipulable par l'usager.

Elle peut par exemple se définir des attributs "documentation" sur lesquels elle interdit toute modification, mais qui peuvent être consultés (opérateur "ouvrir" sur l'icone qui représente cet attribut); ou des attributs "adaptation" dont les valeurs peuvent être modifiées dynamiquement par l'usager pour personnaliser l'application qu'il utilise. La modification d'un tel attribut est immédiatement reflétée à l'application qui adapte alors son comportement;

- les attributs définis et exploités par le Médiateur. Ces attributs concernent l'ensemble des objets (fenêtres, module de visualisation, gestionnaire de dialogue) gérés par le Médiateur. Parmis ces attributs on peut citer le niveau de visibilité des informations affichées à l'écran, la forme du dialogue, les dimensions d'une fenêtre, etc.

La liste des propriétés associée à un opérateur indique le type d'interaction courant (novice ou expert, avec ou sans confirmation), l'ouverture ou non d'une nouvelle fenêtre lors de l'activation de l'opérateur.

L'utilisateur peut par exemple imposer que toute destruction d'un objet de type module exige une confirmation. Il sélectionne alors la propriété "confirmation" de la liste des propriétés de l'opérateur "détruire" de l'objet modèle "module" et lui affecte la valeur "confirmer".

Dès ce moment, lorsque l'usager applique l'opérateur détruire sur l'icone d'un objet-instance de type module, le Médiateur demandera à l'usager de confirmer cette action.

Ces attributs permettent à l'usager de personnaliser son environnement en fonction de ses besoins et des connaissances qu'il a déjà acquises. Cette adaptation peut se réaliser au niveau de l'opérateur; ainsi un opérateur peu employé pourra être défini en mode novice et un opérateur connu en mode expert.

La personnalisation du dialogue peut aussi s'effectuer à un niveau global. Ceci conditionne la présence des fenêtres menu, la forme de la fenêtre de dialogue (présence ou non du formulaire) et l'apparition de certaines informations en cours de dialogue.

2.3. Les objets définis par le Médiateur

Pour répondre aux besoins de dialogue, de contrôle de l'évolution des activités et d'adaptation du poste de travail, le Médiateur définit pour l'usager les objets "activité", "dialogue" (menu, formulaire, langage de commande) et "attributs".

Pour matérialiser ces objets, le Médiateur utilise les services définis par le gestionnaire de fenêtres et le Compositeur.

Une "activité" se matérialise sur l'écran par un ensemble de fenêtres dans lesquelles sont visualisées les données d'une application. A une fenêtre correspond une application propriétaire; tout changement d'activité s'exprime par un changement de fenêtre courante.

Le Médiateur gère la correspondance entre fenêtre et activité. Pour ce faire, il filtre toutes les opérations sur les fenêtres et sous-traite leur réalisation au gestionnaire de fenêtres.

Le Médiateur est un traducteur de dialogue. Comme tel, il manipule des objets "dialogues" qui se matérialisent dans des fenêtres. L'ensemble de ces fenêtres représente à un instant donné le dialogue avec l'application courante. Lors du changement d'activité, ces fenêtres sont modifiées de manière à refléter l'état dans lequel se trouvait le dialogue lorsque l'usager a quitté l'application sélectionnée.

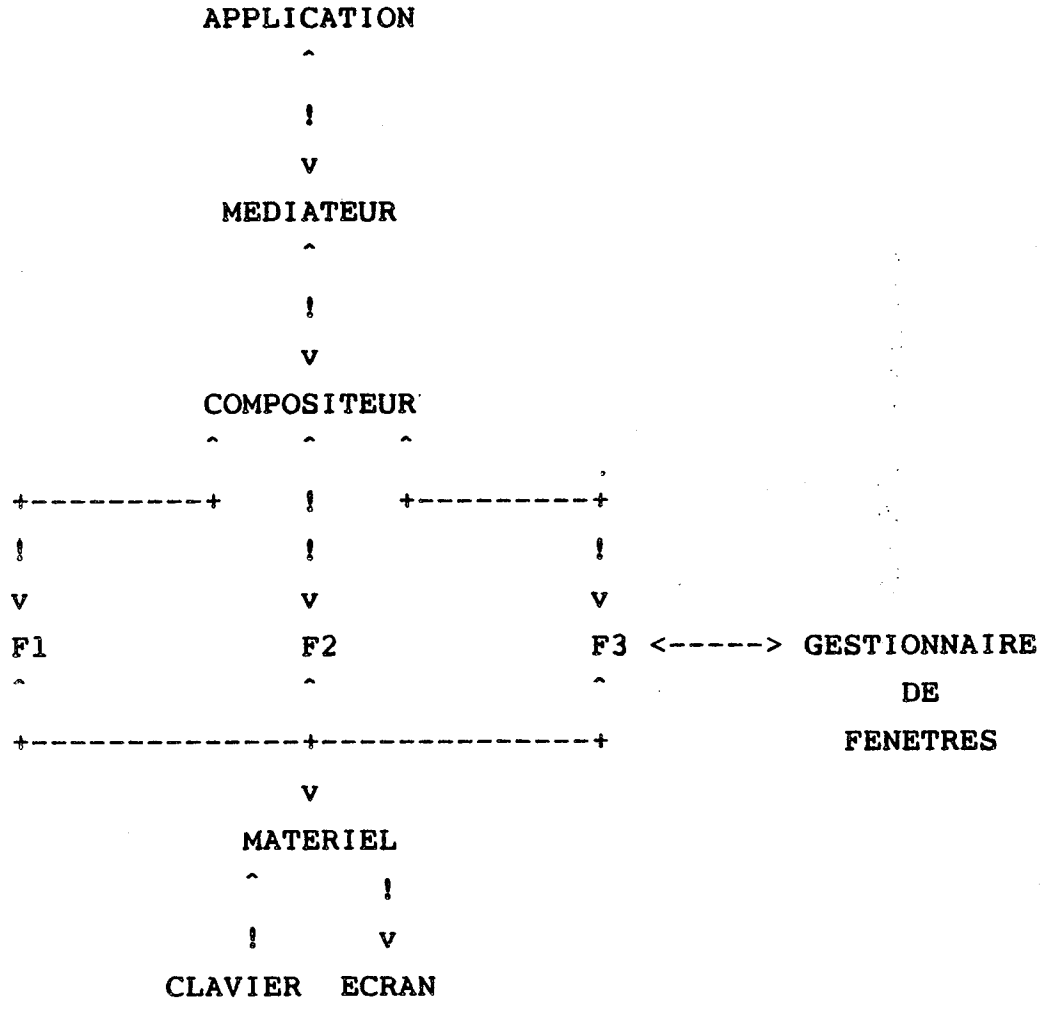
Les objets de dialogue (menus, formulaires, etc) et les propriétés associés aux modèles et aux opérateurs sont implémentés

en termes des abstractions élémentaires que sont les boites et la fenêtre.

Les menus sont décrits par des RE sur lesquelles seule l'opération de désignation est autorisée. Le formulaire est représenté par une RE sur laquelle les opérations d'édition sont permises.

2.4. L'intégration des composants de l'interface et ses limites

L'architecture que nous proposions au chapitre précédent est définie à l'aide d'entités coopérantes. Modélisée sous cette forme, l'architecture de notre interface serait la suivante:



L'intérêt de cette démarche est que toute composante réalise l'interface entre les composantes avec lesquelles elle est en relation; composantes appartenant à des couches fonctionnelles distinctes.

Ainsi, le Compositeur réaliserait l'interface entre la fenêtre, le gestionnaire de fenêtres et le Médiateur. Ce dernier n'aurait alors pas à connaitre le protocole de communication entre le Compositeur et le gestionnaire de fenêtres, ni les propriétés de ces deux entités. La nature des informations échangées et les transformations successives qu'elles subissent échapperaient totalement au Médiateur.

Dans cette approche, l'architecture adoptée et le type de conception qu'elle implique, résoud le problème de l'intégration des composants interactifs d'un logiciel.

Notre réalisation ne suit pas cette démarche et il nous a néanmoins fallu résoudre ce problème d'intégration dans l'interface usager, afin d'éviter que les applications n'aient, elles, à le résoudre.

2.4.1. L'intégration des composants

La solution que nous avons retenue consiste en la centralisation des interactions par le Médiateur. Les informations résultantes doivent alors être aiguillées vers les couches logicielles concernées.

EXEMPLE:

Supposons qu'après avoir positionné le curseur sur une information visible à l'écran, l'usager frappe une touche du clavier standard. Cette interaction est interprétée par le

Médiateur comme le désir d'éditer cette information.

La première action qu'il va entreprendre, est de déterminer quel est l'objet concerné. Connaissant la fenêtre courante, il demande au gestionnaire de fenêtres de déterminer l'identification de l'éditeur et de l'objet concerné. Ensuite, il va s'adresser à cet éditeur et l'activer sur l'objet modifié.

Il interprète donc les caractères du clavier standard comme des envois d'informations à l'éditeur et certaines interactions comme des appels aux fonctions de l'éditeur: destruction de caractère, de mot, de ligne et fonctions de déplacement dans l'objet édité. L'éditeur gère l'objet sous contrôle du Médiateur.

Le Médiateur réalise l'interface entre le logiciel graphique et le gestionnaire de fenêtres: il induit la succession des traitements nécessaires pour que les applications et les usagers puissent avoir une vision uniforme et simple de l'interface usager.

EXEMPLE:

Lors de la frappe du premier caractère dans l'exemple précédent, il a fallu passer d'une notion de très bas niveau (position du curseur sur le terminal physique), à la position du curseur dans une fenêtre.

La première action fut donc de déterminer quelle était la fenêtre courante, et donc éventuellement d'opérer un changement d'activité.

Connaissant la fenêtre courante, il a pû déterminer l'identification de l'éditeur qui est associé à la fenêtre et celle de l'objet concerné. Connaissant statiquement les propriétés de l'objet "éditeur", il a induit l'édition.

Il en va de même lorsqu'une application effectue une association entre une fenêtre et une RE, celle-ci est filtrée par le Médiateur qui évalue la surface de l'image occupée par la RE et demande l'allocation d'une fenêtre dont les dimensions optimales sont déterminées par le gestionnaire de fenêtres.

Le gestionnaire de fenêtres effectue cette allocation en fonction de critères qui lui sont propres (cf. 3.) et se met en relation avec le Compositeur pour que la RE puisse être visualisée.

L'application ne connait pas explicitement la notion de fenêtre: elle n'en fixe ni les dimensions, ni la position. Lors d'une association, elle n'exprime que la nécessité de montrer une image. Par la suite, la connaissance de la fenêtre créée ne lui est pas nécessaire, la relation entre une RE et les fenêtres où elle apparait est gérée par le module de visualisation du Compositeur.

Intégrant les concepts définis dans le gestionnaire de fenêtre et le Compositeur, le Médiateur gère les entités que ceux-ci définissent comme des objets.

Il possède par exemple la connaissance de l'objet "fenêtre" par la méthode d'accès "gestionnaire de fenêtre". Il manipule cet objet lorsqu'une application ou l'usager désire créer, détruire ou modifier les dimensions d'une fenêtre. Pour ce faire, le Médiateur connait statiquement l'ensemble des opérateurs définis sur une fenêtre.

De même, il possède la connaissance de certaines des caractéristiques du logiciel graphique et des entités qui assurent la projection de la RE. Il connait la notion de niveau de visibilité, d'éditeur ainsi que l'ensemble des opérations définies dans le logiciel graphique et directement accessibles à l'usager.

2.4.2. Limitations

Si le mécanisme mis en place permet d'assurer effectivement l'intégration des composants, il implique également que le Médiateur connaisse statiquement les caractéristiques de tous les objets logiciels qu'il utilise. La première conséquence en est la difficulté de réaliser le Médiateur qui possède tous les rôles, et qui doit résoudre le problème de l'intégration.

La seconde est de devoir le modifier toutes les fois qu'une modification majeure est effectuée sur l'une des abstractions qu'il utilise.

Ainsi, dans l'interface que nous décrivons, des entités telles que l'éditeur sont amenées à évoluer (cf. chap 4). Des possibilités supplémentaires doivent leurs être ajoutées.

Ces logiciels auront à terme un comportement d'applications (certes un peu particulières) et comme tels, nécessiteront un gestionnaire de dialogue adapté à leurs propres besoins. La gestion de l'ensemble des interactions ne pourra plus être réalisée par u.. gestionnaire unique.

Nous avons d'autre part vus au chapitre précédent les difficultés liées à la conception des logiciels. Nous avons conclus que la démarche employée procédait nécessairement par essais-erreurs. Dans ce type de démarche il est souhaitable que les modifications effectuées dans une couche n'induise pas de modification dans les couches de niveau supérieur (effet recherché dans l'approche modulaire).

Souhaitable dans la réalisation de logiciels traditionnels, le cloisonnement entre couches fonctionnelles est impératif dans le domaine des logiciels interactifs et plus particulièrement dans la définition des interfaces usager.

D'où l'intérêt de la démarche que nous proposions dans le chapitre précédent. Toutes les relations entre couches reposent sur un protocole qui, s'il a été soigneusement étudié pourra assurer la transparence totale des modifications de fonctionnalités apportées à une couche de bas niveau. Seules les compétences d'une couche doivent, en principe, évoluer et donc nécessiter la création de nouvelles possibilités d'interactions.

Ceci dans le cas d'une extension des compétences de l'éditeur vers celles d'un éditeur structurel ne saurait remettre en cause le protocole dont nous avions fourni une ébauche. Il permet en effet l'édition d'un élément atomique de la RE, il fournit comme résultat l'identification de l'élément modifié.

Lorsque l'on passe à une édition de type structurel, l'élément modifié peut être de nature composite. Mais malgré tout, le résultat d'une édition restera l'identification d'un élément.

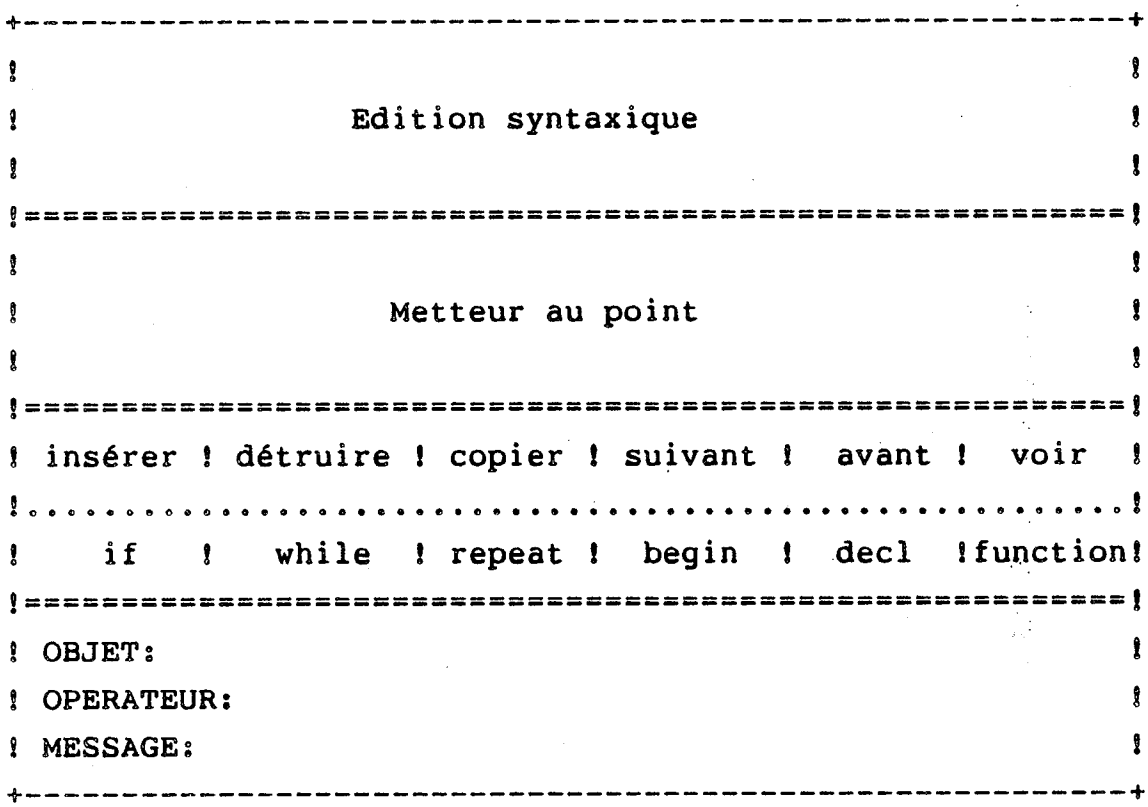
Comme la RE est une structure de donnée arborescente (ce qui permet de définir la portée d'une modification comme le plus petit sous-arbre incluant la ou les modifications) accessible en lecture, le principe de l'acquisition des résultats d'une modification ne change pas. Il faut que l'application parcoure un sous-arbre. Ce sous arbre, dans le cas particulier de notre éditeur, se restreint à un élément atomique.

La nécessité d'exprimer la relation entre les composants de l'interface usager en terme de coopérations est l'une des conséquences des difficultés rencontrées dans ce projet. L'expérience que nous en avons retirée nous a permis d'aboutir aux conclusions du chapitre précédent.

3. Le gestionnaire de fenêtres

Comme nous l'avons vu au chapitre précédent, le gestionnaire de fenêtres définit un ensemble d'opération sur les fenêtres. Chacune d'elles est un terminal virtuel qui se matérialise sur l'écran par un cadre rectangulaire. Dans ce cadre sont visualisées les informations que Compositeur y projette.

L'écran du poste de travail, tel qu'il est vu par l'usager, se présente comme une mosaique de fenêtres. Certaines sont le lieu d'exécution des outils de l'atelier Adèle, d'autres sont réservées au dialogue entre l'usager et l'activité courante. Une configuration type de ce poste est présentée par la figure ci-dessous.



Les deux premières fenêtres sont le lieu d'exécution des applications "éditeur syntaxique" et "metteur au point" de l'atelier.

Apparaissent ensuite deux fenêtres de menu. Le premier menu est celui qui matérialise les commandes de l'application courante (l'éditeur syntaxique dans notre cas); le second visualise les objets-modèles de cette même application.

Enfin, la dernière fenêtre matérialise le dialogue entre l'usager et l'application. Dans notre exemple, l'usager emploie un dialogue de type formulaire. Les menus situés au dessus de la fenêtre de dialogue lui présentent une partie des options qu'il peut choisir.

Le Médiateur utilise les fonctions du gestionnaire de dialogue lorsque l'usager émet une commande pour organiser les informations sur l'écran et lorsque l'application ou le médiateur demandent la création ou la destruction d'une fenêtre. A partir de telles modifications, le gestionnaire en déduit une nouvelle organisation des informations sur l'écran.

3.1. La partition de l'écran

Le Médiateur décrit le partage de l'écran en exprimant les relations qui existent entre les différentes fenêtres qui y apparaissent. Dans le gestionnaire que nous avons défini, les relations possibles sont telles qu'à tout instant il apparait à l'écran une mosaique de rectangles. (1)

Les opérations définies dans le gestionnaire permettent:

- la création d'une fenêtre:

cet opérateur a pour paramètre les dimensions souhaitées

---

(1) Les principes du gestionnaire, brièvement présentés ici, sont développés dans <Coutaz 82>.

pour cette nouvelle fenêtre. Le gestionnaire alloue alors une fenêtre dont la position est déterminée par la relation entre cette fenêtre et les fenêtres environnantes.

Il détermine à partir de ces relations les dimensions optimales de la fenêtre nouvellement créée, en tenant bien évidemment compte des souhaits exprimés.

EXEMPLE

Si lors de l'allocation de la première fenêtre, le logiciel demandeur exprime le souhait d'obtenir une fenêtre qui occupe tout l'écran, une fenêtre de taille maximum lui sera allouée.

Lors d'une seconde allocation, le gestionnaire réalisera un partage équitable entre les deux fenêtres. Les dimensions de la première seront diminuées, tandis que la nouvelle fenêtre occupera le reste de l'écran.

Le choix de partage que le gestionnaire de fenêtres a effectué peut être remis en cause par l'usager. Celui-ci peut en effet lui imposer les dimensions de son choix.

- la composition verticale de fenêtres:

le logiciel demande au gestionnaire de juxtaposer verticalement une fenêtre et un groupement de fenêtres préexistantes (éventuellement restreint à l'unité). Ceci réalise une répartition des dimensions tel que la somme des largeurs des rectangles ainsi créés soit égale à la largeur du groupement de fenêtres initiales. Aucune contrainte n'est appliquée sur les hauteurs.

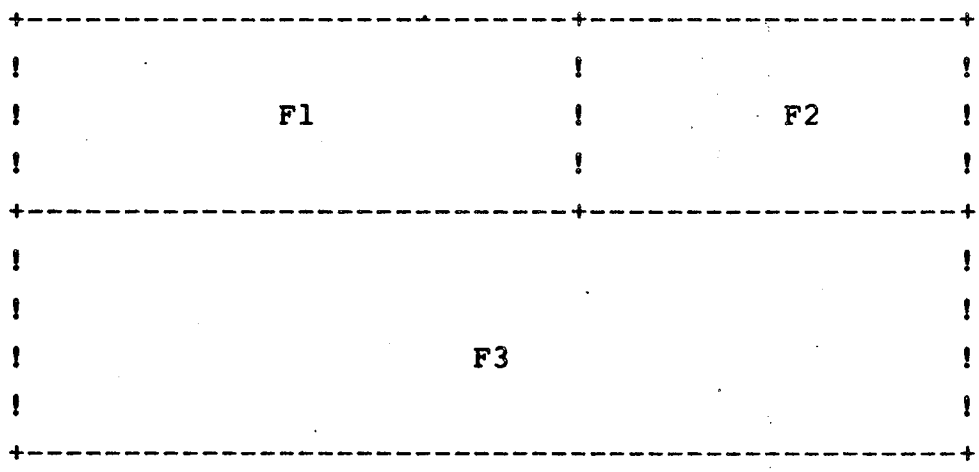
- la composition horizontale de fenêtres:

cette relation réalise la juxtaposition horizontale d'une

fenêtre et d'un groupement de fenêtres. La répartition des dimensions est telle que la somme des hauteurs des fenêtres résultantes est égale à la hauteur occupée par le groupement.

Si ! représente l'opération de composition verticale et - celui de composition horizontale, alors l'expression suivante a pour résultat:

((F1 ! F2) - F3)



Le gestionnaire que nous avons réalisé est une adaptation des principes énoncés aux possibilités limitées (cf. chapitre précédent) des terminaux alpha-numériques. Aussi la seule relation possible entre deux fenêtres est la composition verticale.

D'après les observations que nous avons pû effectuer, la stratégie de partition mise en oeuvre par le gestionnaire donne des résultats satisfaisants si le nombre de fenêtres est petit. Dans le cas contraire, celles-ci deviennent de dimensions trop faibles pour que les informations qui y sont visualisées puissent constituer un contexte de travail réellement exploitable.

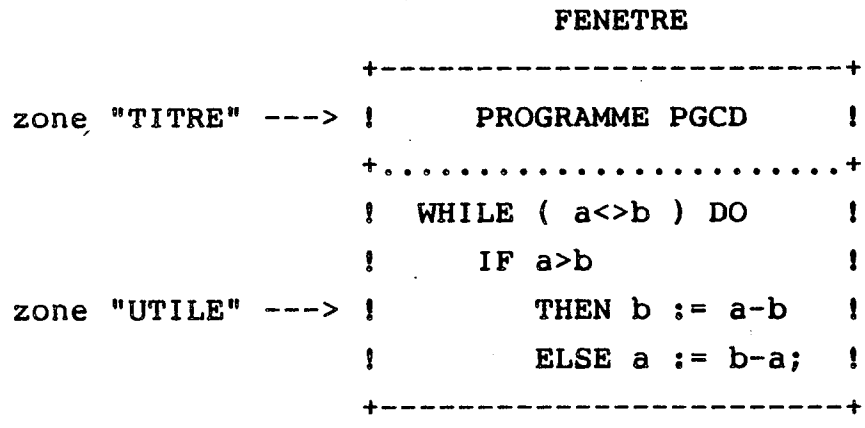
La création d'un nombre inconsidéré de fenêtres, quelle que soit la stratégie mise en oeuvre, aboutit toujours à des situations problématiques. L'expérience montre en effet qu'il est nécessaire de réorganiser de temps à autre la surface de travail (un bureau par exemple).

Le fond du problème est, et reste, la quantité relativement limitée des informations que l'on peut visualiser sur un écran. Plutôt que de rechercher une stratégie optimale, il faudrait donner à l'usager la possibilité de ranger celles des activités qui le gènent momentanément. Il faudra alors lui permettre de retrouver ultérieurement l'organisation qu'il avait quittée.

Dans un tel système, l'usager devra pouvoir exprimer les dépendances entre fenêtres et entre activités (c'est un peu le principe de VTMS) à l'aide de ce que l'on pourrait appeler un éditeur de fenêtres. L'approche décrite dans <Coutaz 82> mériterait donc d'être développée.

3.2. La fenêtre

Comme nous l'avons déjà précisé, la fenêtre est matérialisée sur l'écran par un cadre rectangulaire qui, dans notre réalisation se départage en deux zones distinctes.



Dans la première zone est matérialisé le "titre" de la fenêtre. Il permet à l'usager d'identifier l'activité qui s'y déroule. Le module d'affichage visualise dans la seconde zone les informations de l'application. Nous appelons cette zone la zone "utile"; c'est la seule explicitement connue du logiciel graphique.

En interne, la fenêtre peut être définie comme un type abstrait.

C'est une structure de données qui décrit la position effective de la matérialisation de la fenêtre sur l'écran, ses dimensions, la position de chaque zone, etc.

A cette structure de données sont associés les opérateurs de création, de destruction et de modification employés par les gestionnaire de fenêtres et le Médiateur.

Un autre ensemble de fonctions définit le protocole de communication entre la fenêtre et le logiciel graphique. Cette communication est bi-directionnelle, nous distinguons donc deux classes de fonctions:

- celles qui permettent les transferts d'informations du module d'affichage vers la fenêtre. Elles définissent les fonctions de tracé et de mise en évidence . Elles réalisent la notion de terminal virtuel. Une particularité de notre interface réside dans le fait qu'il est possible au compositeur de demander à une fenêtre de se détruire (lors de la destruction de la RE qui y est projetée par exemple);

- celles qui permettent à la fenêtre de répercuter vers le module d'affichage les modifications qu'elle subit. Ce mécanisme permet de prendre en compte la variations des dimensions d'une fenêtre, sa destruction etc.

Dans notre réalisation, le terminal virtuel est adapté aux possibilités des terminaux alpha-numériques. Ainsi, toute fenêtre

occupe systématiquement toute la largeur de l'écran et seuls les caractères sont affichables.

Les mises en évidences permises sont celles définies en standard dans ce type de périphérique (inverse-vidéo, clignotement, soulignement).

Sous Multics, l'indépendance syntaxique par rapport au terminal a été obtenue en définissant un jeu de procédures dont les noms et la sémantique sont normalisés et en utilisant le mécanisme d'édition de lien dynamique disponible sur ce système.

4. Le compositeur - vue externe

Le Compositeur est un outil général, utilisé par les applications du projet Adèle pour la description logique de la structure et de l'aspect visuel des informations à afficher. Le Compositeur est un logiciel de composition qui permet d'exploiter les possibilités du multi-fenêtrage.

Le mécanisme proposé repose sur l'existence d'une structure d'échange unique (définie indépendamment de toute application) qui est un arbre dont les noeuds sont appelés BOITES. (1)

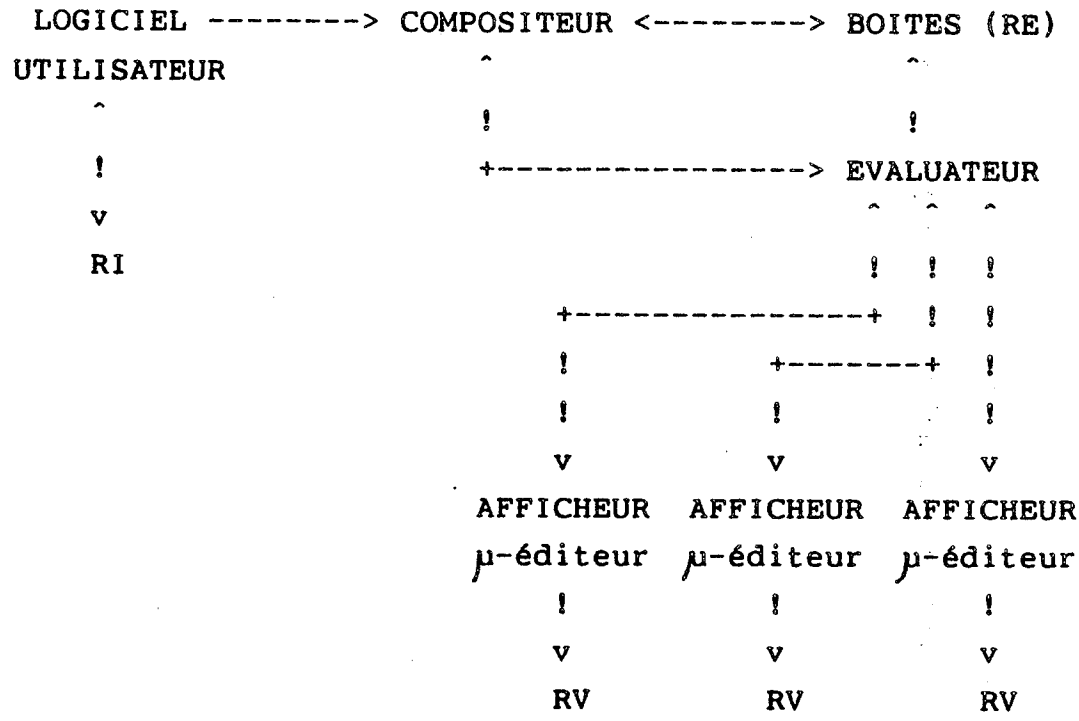
L'architecture globale du Compositeur peut être schématisée de

---

(1) La notion de Boite est aussi utilisée dans des éditeurs de documents tels que TEX <Knuth 79>, Pen <Allen 81>, Etude <Good 81, Hammer 81>, ED3 <Stromfors 81>, Grafmath <Joloboff 84>, GTX <Nanard 83> et Janus <Chamberlin 81>. La vocation de ces éditeurs est la restitution de documents sur des supports papiers.

Nos objectifs sont différents; nous résolvons les problèmes de description et de gestion d'informations visibles sur un support non séquentiel et non ceux de photo-composition <Bratley 83>.

la façon suivante:



L'application constitue et modifie cette structure d'échange, ou RE, à l'aide des opérateurs définis par le Compositeur (c'est ainsi que nous appelons cette méthode d'accès aux boites). Les modifications de la RE se traduisent par la modification de toutes ses occurences visibles, ou RV, qui apparaissent à l'écran. Chacune de ces occurences est le résultat de la projection de la RE dans une fenêtre.

La visualisation de la RE dans une fenêtre est réalisée par l'AFFICHEUR associé à chaque fenêtre. Celui-ci utilise les fonctions du terminal virtuel et simule celles des fonctions non réalisées par le matériel. Il assure aux applications une indépendance totale (pas seulement syntaxique) par rapport au terminal.

L'afficheur accède à la RE en utilisant les services d'un module de parcours d'arbre appelé EVALUATEUR qui regroupe les fonctions de linéarisation d'arbre et d'évaluation incrémentale

des incidences d'une modification de la RE.

L'évaluateur maintient aussi la cohérence entre une RE et les différentes RV qui la représentent. Ainsi, les fonctions de modification du Compositeur font appel à ce module pour évaluer les incidences visuelles d'une modification. C'est ce qui permet la répercussion immédiate et transparente à l'application des manipulations que cette dernière effectue sur la RE.

Le couple évaluateur-afficheur constitue ce que nous avons appelé le module d'affichage.

A chaque afficheur est associé un micro-éditeur qui définit les fonctions nécessaires à l'édition des éléments atomiques de la RE. Il permet à l'usager de manipuler directement les informations visibles.

Dans ce qui suit, nous mettons en évidence les principales caractéristiques de la RE, du Compositeur, de l'évaluateur et de l'afficheur μ-éditeur. Nous n'y aborderons pas les aspects techniques qui font l'objet du chapitre suivant.

4.1. Les boites: une structure d'échange

La RE doit renfermer les potentialités nécessaires pour qu'une application puisse décrire et structurer une image à un niveau logique.

Il faut donc qu'elle puisse:

- manipuler l'image à un niveau logique, indépendamment de tout problème de visualisation;

- en décrire l'aspect (couleur, fontes, effets visuels);

- décrire l'organisation des informations visualisées,

indépendamment des contraintes matérielles (largeur d'un terminal par exemple);

- décrire statiquement les différentes représentations visuelles qui peuvent être dérivées d'une même RI;

- consulter cette RE.

Dans notre interface usager, la BOITE est l'élément de description de la RE. C'est une structure de données arborescente dont les principes généraux peuvent se résumer ainsi:

- toute valeur (texte, graphisme élémentaire, etc) est associée à une boite feuille. La boite feuille est l'élément atomique de la RE,

- la valeur d'une boite composite (donc non atomique) est définie par la valeur de ses filles (synthèse). Lors de l'affichage ceci se traduit par l'inclusion des valeurs des boites composantes dans un rectangle imaginaire par lequel on peut représenter une boite composite,

- un ensemble d'informations appelées ATTRIBUTS peuvent être associées à toute boite. Ces attributs permettent de définir l'aspect (fonte des caractères, couleur du fond, etc) de la RV, les différentes formes qu'elle peut prendre (élision). Ils permettent aussi à l'application de contrôler l'accès de l'utilisateur à la RE (édition et désignation),

- l'aspect que prend la valeur d'une feuille lors de l'affichage est déterminé par les attributs qui lui sont associés, ainsi que par ceux associés à ses ancêtres (héritage),

- l'affichage de la RE repose sur une interprétation de l'arbre de boites.

Pour définir l'aspect de l'information affichée dans une fenêtre, nous disposons de cinq classes d'attributs:

- Ceux qui définissent le matériel typographique utilisé pour l'affichage;

- Ceux qui définissent les règles de composition;

- Celui qui permet de définir la relation RI-RE;

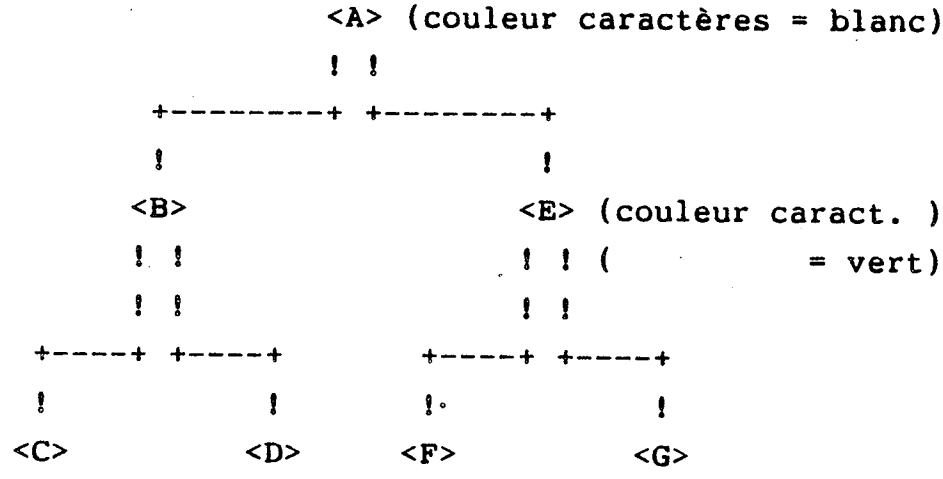
- Ceux qui permettent de contrôler l'accès de l'utilisateur à cette structure en désignation et en édition;

- Ceux qui permettent d'obtenir à partir d'une même description plusieurs représentations visuelles .

4.1.1. Portée des attributs

Le principe retenu pour la quasi totalité des attributs, est que la portée d'un attribut associé à une boite composite (racine d'un sous-arbre) est le sous-arbre dont cette boite est la racine. Cette portée pourra cependant être limitée par la redéfinition de la valeur de cet attribut dans une ou plusieurs branches de ce sous-arbre.

EXEMPLE



La couleur des caractères associée à l'information de type texte attachée à l'arbre de racine "A" est "blanc", sauf pour le sous-arbre de racine "E", pour lequel les caractères seront de couleur verte.

Lorsqu'un attribut est associé à une boite feuille, sa valeur se rapporte aux informations qui se rattachent à cette boite feuille.

Rappelons pour mémoire que seules les boites feuilles contiennent une information effectivement affichable (texte, dessin, etc); les autres boites définissent la structure et l'aspect visuel de l'information.

4.1.2. Les attributs définissant le matériel typographique

La FONTE définit la police de caractères ou le pinceau choisi pour l'impression de la valeur associée à une boite.

La couleur des caractères.

La couleur du fond définit la couleur de la surface sur laquelle est visualisée la valeur de la boite.

Les effets "spéciaux" comprennent toute forme de mise en évidence de la valeur de la boite: clignotement, soulignement, inverse vidéo, surbrillance, etc.

4.1.3. Les attributs de composition

Ils définissent la disposition des surfaces associées aux boites les unes par rapport aux autres.

La surface d'une boite feuille contenant une valeur de type texte est définie comme un rectangle dont:

- la largeur est celle de la ligne la plus large,
- la longueur est donnée par le nombre de lignes de ce texte.

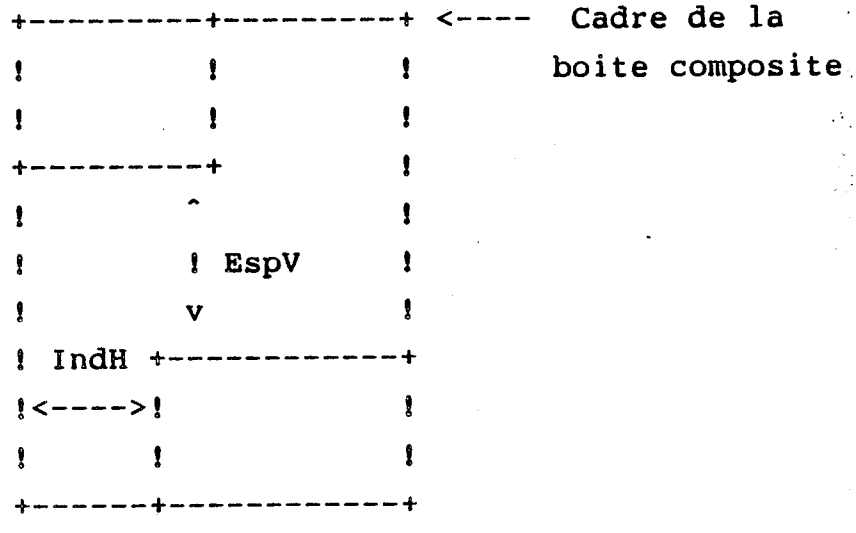
La largeur maximale d'une ligne peut être contrainte par l'application en associant à une boite un attribut qui la définit. Cette valeur est exprimée en nombre de caractères.

Lorsqu'une boite feuille contient une information de type texte dont le nombre de caractères est supérieur à cette valeur, ce texte est replié lors de l'affichage.

L'information qui défini la composition est donnée au niveau de la boite mère. Elle précise la disposition relative des surfaces définies par ses filles. De même, les espaces à laisser entre les boites filles (Espacement) sont définis au niveau de la boite mère.

Quatre formes de compositions sont définies:

1- Composition verticale.



Deux surfaces peuvent être disposées l'une en dessous de l'autre. C'est ce qui correspond à une composition verticale notée V.

Deux informations supplémentaires doivent alors être fournies:

- l'espacement entre les bords bas et hauts des deux surfaces, que nous appelons espacement vertical (noté EspV). Cette valeur est donnée en nombre de lignes;

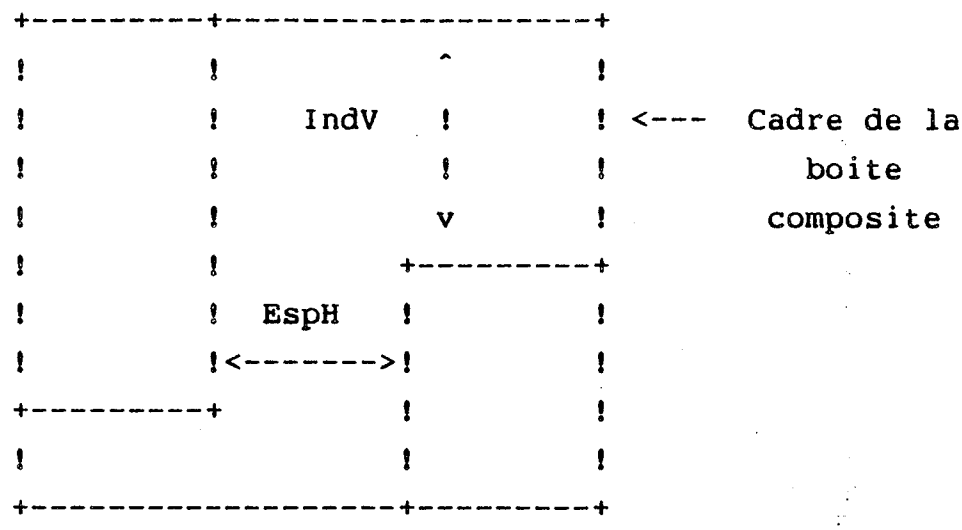
-le décalage horizontal par rapport au bord gauche du cadre de la mère, que nous appelons indentation horizontale (notée IndH). Elle est donnée en nombre de caractères.

2- Composition horizontale

Deux surfaces peuvent être disposées l'une à coté de l'autre; c'est ce que nous appellerons une composition horizontale (notée H). Les informations supplémentaires qui sont alors nécéssaires sont:

- l'espacement entre les bords droits et gauche des deux surfaces consécutives, appelé Espacement horizontal (noté EspH);

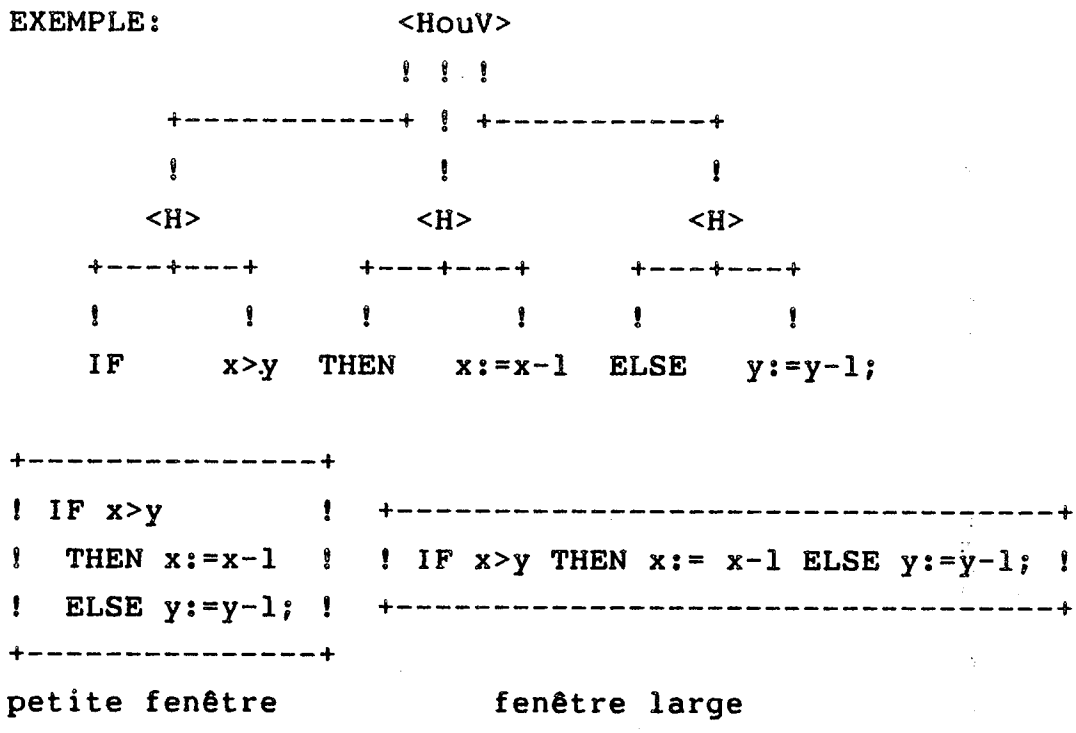
- l'indentation verticale (notée IndV) indique un décalage vertical exprimé en nombre de lignes par rapport au bord haut du cadre de la boite mère.



De ces deux types de compositions, nous pouvons en dériver d'autres qui s'expriment à partir des deux précédentes. Elles doivent permettre de tenir compte de contraintes telle que la largeur de la fenêtre où ces informations apparaissent.

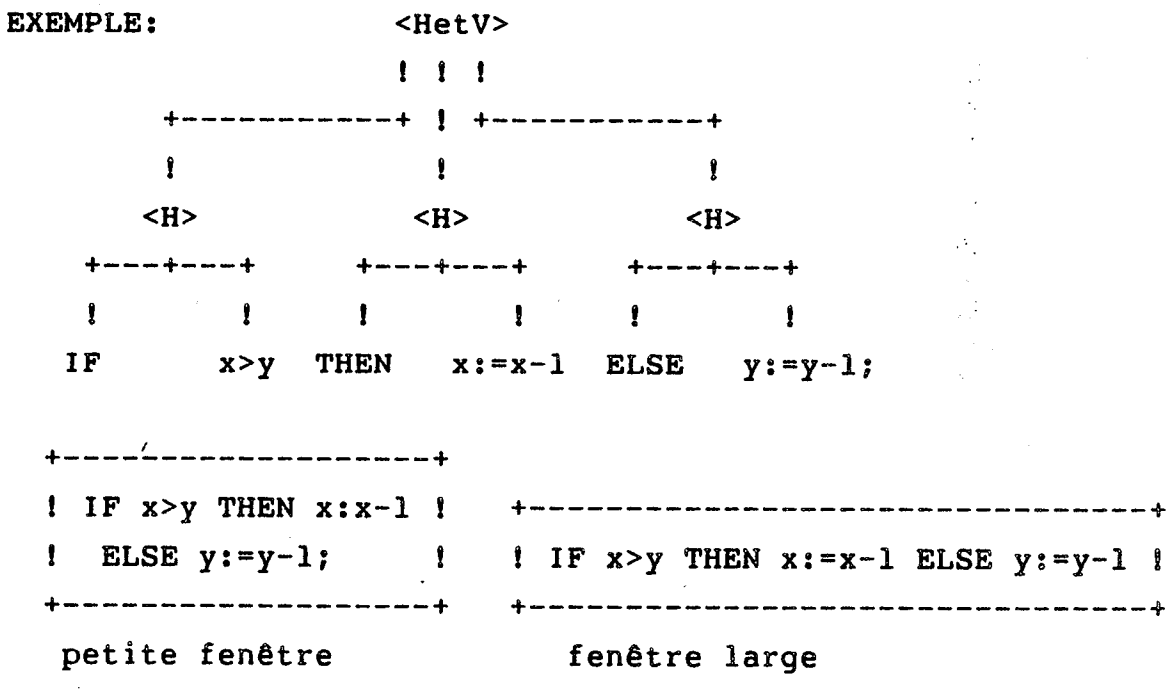
3. Composition horizontale ou verticale (HouV)

Ce type de composition a pour but de ne composer horizontalement des surfaces que si la largeur occupée par celles-ci est inférieure ou égale à la largeur de la fenêtre. Dans le cas contraire, la composition effective sera verticale.



4. Composition horizontale et verticale (HetV)

Ce type de composition a pour but de disposer les surfaces les unes à coté des autres tant qu'il y a de la place dans la fenêtre. Sinon il faut effectuer un repliement.



4.1.4. Attribut définissant la relation RI-RE

La sémantique associée aux boites est connue seulement de l'application et la granularité des informations (portée d'une désignation par l'usager d'un objet affiché à l'écran), est définie par celle-ci. Pour ce faire, elle tient compte du fait que la plus petite information désignable est la valeur d'une boite feuille.

La mise en correspondance entre une boite et un objet appartenant à l'application, s'effectue en associant l'attribut "lien" à cette boite. Un même lien pourra être associé à plusieurs boites, alors que certaines n'auront aucun lien.

Toute manipulation de la structure d'échange s'effectue en utilisant les primitives du compositeur et toute boite est désignée par son nom (appelé "nom de boite"). Le résultat d'une désignation, tel qu'il est transmis à l'application, est alors le nom de la boite désignée et non pas la valeur de cette boite.

La valeur que prend l'attribut lien pour la boite désignée peut être une valeur héritée. Dans l'implantation courante du Compositeur, cette valeur est demandée explicitement par l'application.

4.1.5. Attributs de contrôle d'accès

Ces attributs permettent à l'application de contrôler les actions que l'utilisateur peut effectuer sur la représentation visuelle. Ces actions sont au nombre de deux.

L'usager peut vouloir désigner à l'aide d'une souris ou d'un photo-style. Mais il se peut que la désignation d'une partie de la représentation visuelle n'ait pas de sens pour l'application (Un exemple en est la désignation d'un texte d'aide à l'usager).

L'attribut "contrôle_désignation" permet à l'application de valider ou d'invalider la désignation. Cet attribut peut prendre pour valeurs "désignable" ou "non_désignable". Sa valeur par défaut est "désignable".

L'usager peut vouloir modifier une partie de la structure d'échange à l'aide de l'éditeur. Afin que l'application puisse préserver la RE de toute modification intempestive, l'attribut "Contrôle_édition" peut prendre pour valeurs "éditable" et "non_éditable". La valeur par défaut pour cet attribut est "éditable".

La portée de ces d'attributs suivent le schéma général d'héritage. Leur portée est le sous-arbre dont la racine est la boite à laquelle cet attribut est associé; sauf redéfinition à un niveau inférieur.

4.1.6. Attributs permettant d'exprimer la multi-représentation

Ces attributs permettent à l'application de définir la ou les différentes représentations visuelles d'une RE, sans modifier la structure de l'arbre de boites.

Le principe de base est celui de l'élision <Milkelson 81, Teitelbaum 81, Teitelbaum 81.a>: on ne laisse voir à l'usager que certaines parties d'un objet, ce qui revient à rendre le reste invisible.

L'attribut "invisible" permet de rendre invisible le sous-arbre dont la racine est la boite à laquelle cet attribut est associé. Au niveau visuel, tout se passe comme si ce sous-arbre n'existait pas dans la RE.

Cet attribut permet à l'application d'effectuer des élisions

sous son propre contrôle. L'inconvénient majeur en est l'impossibilité de dériver à partir d'un tel arbre plusieurs représentations visuelles.

Pour définir plusieurs représentations visuelles simultanées d'une RI, l'application est alors amenée à gérer plusieurs arbres.

Une généralisation en est l'attribut "finesse" dont les raisons d'être sont les suivantes:

- il permet à l'application de définir statiquement les différentes représentations visuelles possibles d'un arbre de boites;

- le niveau de visibilité, qui définit la finesse de l'information visible à un instant donné, est un paramètre de l'afficheur associé à une fenêtre. Le niveau de visibilité est une valeur que l'usager peut modifier dynamiquement.

L'intérêt de cet attribut est que la gestion des différentes vues déduites d'une même descrition est gérée par le Compositeur et que les éventuelles modifications du niveau de visibilité restent transparentes à l'application.

La finesse d'une boite est définie comme suit:

- la finesse d'une boite à laquelle l'attribut finesse n'est pas associé est égale à la finesse de la boite mère (valeur héritée);

- la finesse d'une boite à laquelle cet attribut est associé, est égale à la finesse de sa boite mère augmenté de K, où K est n'importe quelle fonction qui rend une valeur strictement positive. Dans l'implantation actuelle, K est une constante ayant pour valeur 1;

- La finesse "héritée" par la boite racine de l'arbre

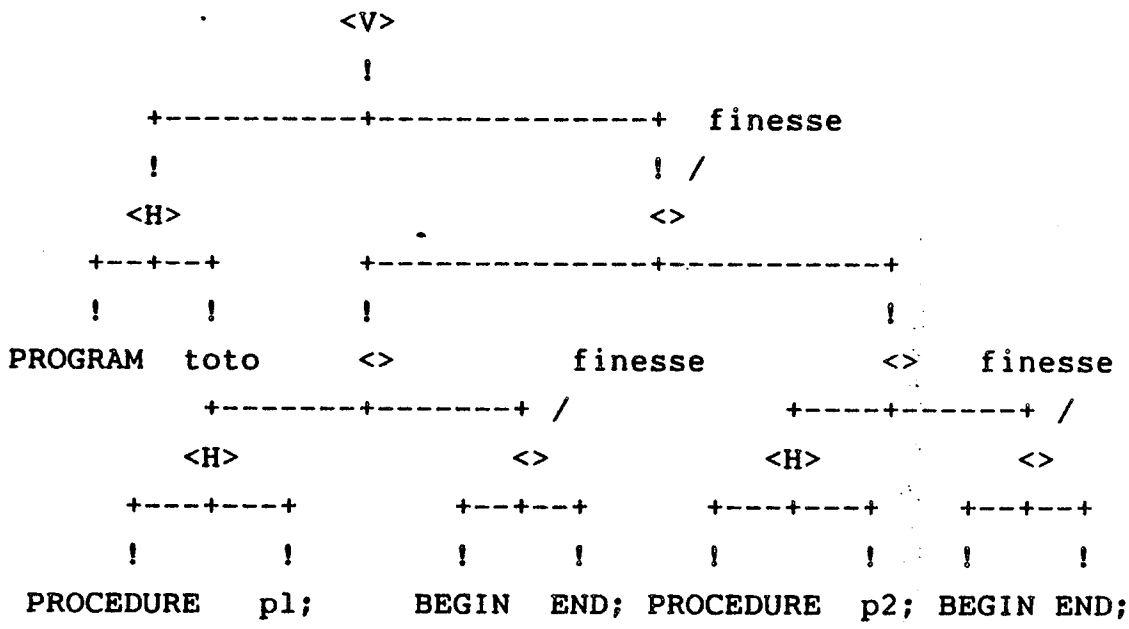
d'affichage est égale à 0.

La visualisation des informations dans une fenêtre nécessite l'interprétation de l'arbre de boites. L'un des paramètres de cette interprétation est le niveau de visibilité de l'afficheur pour lequel elle est effectuée.

Une boite n'est alors visualisée dans une fenêtre que si sa finesse est inférieure ou égale au niveau de visibilité associé à l'afficheur qui visualise cette boite dans cette fenêtre.

Remarquons que l'attribut finesse permet de décrire une visualisation en fonction de la profondeur de l'arbre <Hansen 71, Huet 77>, mais que ses possibilités sont plus étendues.

EXEMPLE



Ce qui donne les résultats suivants:

- Niveau de visibilité = 0

```
PROGRAM toto

PROCEDURE p1;

PROCEDURE p2;
```

- Niveau de visibilité = 1

```
PROGRAM toto

 PROCEDURE p1;
  BEGIN
  END;

 PROCEDURE p2;
  BEGIN
  END;
```

- Niveau de visibilité = 2

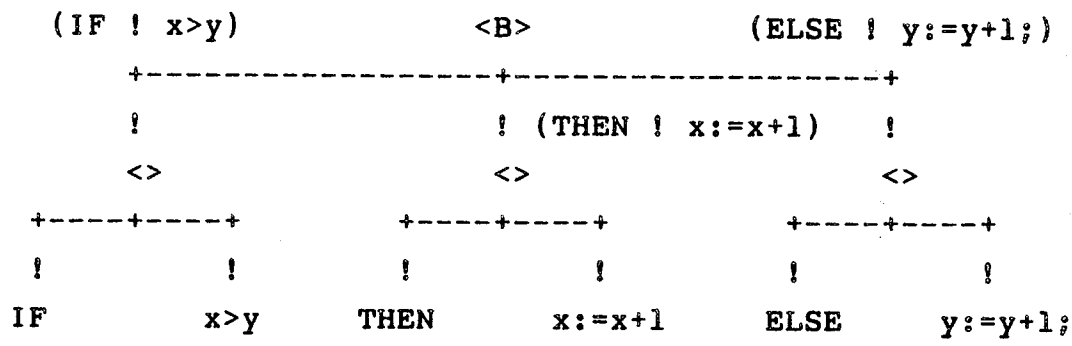
La puissance de ce mécanisme est totalement exploitée lorsque l'usager voit simultanément plusieurs vues distinctes de la même RE dans des fenêtres qu'il a lui même créées, et ceci sans que l'application ne gère ces vues, ni ne connaisse l'existence de plusieurs fenêtres.

4.1.7. Valeur d'une boite

Comme nous l'avons déjà précisé, les boites feuilles constituent les éléments atomiques de notre représentation externe. Les valeurs des boites feuilles sont, dans l'implantation actuelle du Compositeur, restreintes aux chaines de caractères.

La valeur d'une boite composite, est le sous-arbre dont la racine est désignée par le nom de la boite composite. L'ensemble des éléments atomiques que l'on peut atteindre à partir de cette racine résulte de la synthèse des textes associés aux boites feuilles de ce sous-arbre. Cette synthèse est le résultat du parcours post-fixé du sous-arbre.

EXEMPLE



La boite B a pour valeur "IF ! x>y ! THEN ! x:=x+1 ! ELSE ! y:=y+1;". Les traits verticaux marquent la séparation entre des textes appartenant à deux boites feuilles consécutives.

4.2. Le Compositeur: une méthode d'accès

Le Compositeur est la "méthode d'accès" à la RE. (1) Il définit les fonctions nécessaires à l'application pour manipuler la RE.

La liaison entre l'application et le compositeur s'effectue à l'aide d'un nom unique (appelé "nom boite") que ce dernier associe à chaque boite lors de sa création. (2) L'application utilise ce nom pour désigner la boite sur laquelle une action doit être effectuée.

Quatre types de primitives sont définies dans le Compositeur. Elles permettent:

---

(1) Ce module a donné son nom au logiciel "graphique" de l'interface usager que nous présentons.

(2) Nous le distinguons de l'attribut "lien" qui peut être associé à une boite, et qui n'a de signification que pour l'application.

- de construire et de parcourir l'arborescence;

- d'associer des valeurs d'attributs aux boites;

- d'évaluer la valeur des attributs d'une boite, ainsi que la valeur de cette boite;

- de contrôler la visibilité des informations.

4.2.1. Primitives de construction et de parcours

Nous regroupons sous cette appellation les primitives permettant:

- la création d'une boite d'un type donnée (boite composite, boite feuille texte), le résultat de cette fonction est le nom de la boite ainsi créée;

- l'insertion d'une boite dans l'arbre;

- l'éclatement d'un sous-arbre, c'est à dire la création d'une forêt constituée par les descendants de la racine de ce sous-arbre;

- l'extraction d'un sous-arbre: suppression des liens de parenté entre la racine du sous-arbre, sa mère et ses soeurs éventuelles;

- la destruction d'un sous-arbre.

Les primitives de parcours que nous avons définies, autorisent les déplacements vers la mère, les soeurs et vers la descendance d'une boite donnée. Ces fonctions ont pour paramètre et pour résultat un nom de boite.

4.2.2. Opérations sur les attributs d'une boite

Deux opérations sont définies sur ces attributs:

- l'affectation: elle permet d'associer un attribut dont le type et la valeur sont définies à la boite dont le nom est passé en paramètre;

- la suppression qui détruit l'association d'un attribut à la boite. La valeur de cet attribut pour cette boite sera donc une valeur héritée.

4.2.3. Les primitives d'évaluation

Elles sont au nombre de trois:

- la première permet de déterminer l'existence d'une association d'un attribut à une boite;

- la seconde recherche la première boite où est défini un attribut donné dans la suite constituée par la boite et ses ascendants.

- la dernière évalue la valeur d'un attribut pour une boite donnée (valeur associée ou héritée).

REMARQUE:

Des primitives similaires ont été définies pour évaluer la valeur d'une boite (restreinte à du texte pour l'instant), en tenant compte du fait que le résultat est obtenu par une synthèse des valeurs composant la boite.

4.2.4. Les primitives de contrôle

La première d'entre elles permet d'associer un arbre ou un sous-arbre à un support d'affichage (une fenêtre par exemple). Cette association a pour conséquence la création d'un afficheur visualisant cet arbre dans la fenêtre créée par le logiciel graphique.

Comme cette liaison a été explicitée au compositeur, celui-ci est donc à même de déterminer l'ensemble des fenêtres dans lesquelles une boite est visualisée. Nous évitons ainsi à l'application de gérer le partage d'une information dans plusieurs fenêtres. S'il y a partage, celui-ci reste totalement transparent à l'application (seule la RE lui est explicitement connue).

La seconde primitive permet à l'application de focaliser l'attention de l'usager sur une partie de la représentation visuelle de l'objet. Cette fonction permet de forcer l'apparition de cette portion d'image dans l'ensemble des fenêtres dans lesquelles cette information apparait.

4.2.5. Le module d'affichage

Le module d'affichage regroupe deux entités, appelées évaluateur et afficheur, qui ne sont connues que du Compositeur. Leur coopération permet de libérer les applications des problèmes de visualisation et donne à l'usager un ensemble de possibilités de manipulation des informations visibles.

Ce module réalise la projection d'un arbre de boites dans plusieurs fenêtres. Il assure la cohérence entre la RE et les différentes occurrences visibles.

Ces deux entités sont fonctionnellement distinctes:

- l'évaluateur définit toutes les fonctions nécessaires à

l'interprétation d'un arbre de boites. Il n'opère que sur la structure de l'arbre d'affichage et sur les attributs qui lui sont associés.

Il est appelé par le Compositeur lorsqu'un opérateur de modification de l'arbre est employé par l'application. Il déduit des modifications de structure et d'aspect les conséquences sur les différentes occurences visibles.

L'évaluateur définit pour l'ensemble des afficheurs qui opèrent sur une RE des fonctions de parcours d'arbre qui sont paramétrées par le niveau de visibilité de chacun d'eux.

- L'afficheur opère sur un "espace d'affichage": description de toutes les informations qui sont visibles dans UNE fenêtre.

Il définit pour l'usager des fonctions de défilement horizontal et vertical, de modification du niveau de visibilité, etc. Il réalise l'affichage effectif des informations dans une fenêtre, connait la position exacte des informations sur l'écran et les caractéristiques de la fenêtre (taille, primitives de tracé, etc).

L'afficheur simule celle des fonctions qui lui sont nécessaires et qui ne sont pas réalisées par le matériel. Il assure ainsi aux applications une indépendance totale par rapport aux possibilités du matériel.

A chaque afficheur est associé un Micro-éditeur qui est employé par l'usager lorsqu'il veut éditer une boite feuille. Cet éditeur élémentaire est implicitement activé sur les informations visibles qui sont définies par l'application comme étant modifiables.

Dans la mesure où l'application et dans une certaine mesure, l'utilisateur (édition), peuvent modifier la structure d'échange afin de supprimer, rajouter des informations ou bien changer

l'aspect de certaines d'entre elles, la solution qui consiste à produire un fichier affichable obtenu par une évaluation unique de la RE n'est pas envisageable (coût prohibitif lors d'une modification).

Le principe que nous avons retenu pour ce module repose sur une évaluation partielle et incrémentale de l'arbre de boites. Nous ne conservons sous une forme interprétée qu'un minimum d'informations.

Une représentation visuelle est le résultat d'une évaluation qui nécessite un parcours de l'arbre d'affichage. Le but de ce parcours est double: il permet de déterminer l'aspect visuel de chaque boite feuille et leurs dispositions relatives les unes par rapport aux autres. En effet, la technique que nous avons développée consiste à ne pas mémoriser de coordonnées dans les boites, seules leurs dimensions sont connues (cf. chapitre suivant).

Aucune restriction de principe n'étant imposée quant aux dimensions effectives de la surface que l'information visualisable peut occuper, il est prévisible que celle-ci, dans bien des cas, ne pourra être visualisée d'un seul tenant sur un écran standard.

Des possibilités de défilement horizontal et vertical sont donc définies afin que l'usager puisse avoir accès à toute l'information potentiellement visible.

A tout défilement d'informations sur l'écran correspond alors une évaluation partielle de l'arbre de boites. A cette fin, l'évaluateur propose aux afficheurs des fonctions qui déterminent les informations qui apparaissent dans une fenêtre lors d'un défilement. Ces fonctions sont paramétrées par le niveau de visibilité que l'usager à associé à l'afficheur.

Comme les boites feuilles seules portent des informations effectivement visualisables et que la structure d'arbre n'est pas

très adaptée à une exploitation directe en vue d'un affichage, nous effectuons lors de l'évaluation une transformation qui nous fait passer d'une structure arborescente vers une structure linéaire.

Cette structure linéaire est matérialisée par une liste de descripteurs de boites feuilles. Chacur d'eux contient l'information qui définit l'aspect, la disposition relative des informations les unes par rapport aux autres et la valeur de la feuille. Toutes les informations d'ordre structurelles sont éliminées.

Le descripteur de boite feuille est la structure de donnée élémentaire sur laquelle opère l'afficheur. Celui-ci est totalement indépendant de la structure de donnée sur laquelle opère l'évaluateur.

L'afficheur gère l'espace d'affichage, c'est à dire l'ensemble des descripteurs de boites feuilles qui sont visibles à un instant donné; il réalise la conversion position (coordonnées fenêtre) nom de boite lors d'une désignation. Sont disponibles à ce niveau un ensemble de fonctions de défilement horizontal et vertical; y est résolu le découpage en lignes et en colonnes du texte à afficher: l'afficheur réalise la troncation des informations qu'il projète en fonction des dimensions variables de la fenêtre.

Le module d'affichage résoud les problèmes liés à la répercussion des modifications apportées au niveau de l'arbre d'affichage (passage d'un sous-arbre en vidéo-inverse par exemple) sur les espaces d'affichage, donc sur l'écran.

Le principe que nous avons retenu est que toute modification apportée à l'arbre de boites est immédiatement répercutée, de manière totalement transparente à l'application, sur l'ensemble des fenêtres où l'information apparait. Par modification, nous entendons ajoût ou suppression d'un attribut (modification d'aspect) ou d'un sous-arbre (modification de structure).

Dans ce but, toutes les fonctions du Compositeur qui permettent de modifier la RE, activent l'évaluateur afin que celui-ci détermine les incidences sur les espaces d'affichages qui sont concernés.

Les techniques que nous mettons en oeuvre (cf chapitre suivant) nous permettent d'optimiser le rafraichissement des fenêtres en ne considérant que les parties de l'arbre qui sont réellement visibles. L'évaluateur effectue de telles optimisations sur la RE.

L'afficheur optimise le rafraichissement de l'écran en ne procédant qu'à une repeinture partielle des fenêtres concernées. Dans l'implantatin actuelle, il optimise le nombre de caractères envoyés vers le terminal alpha-numérique.

CHAPITRE 4

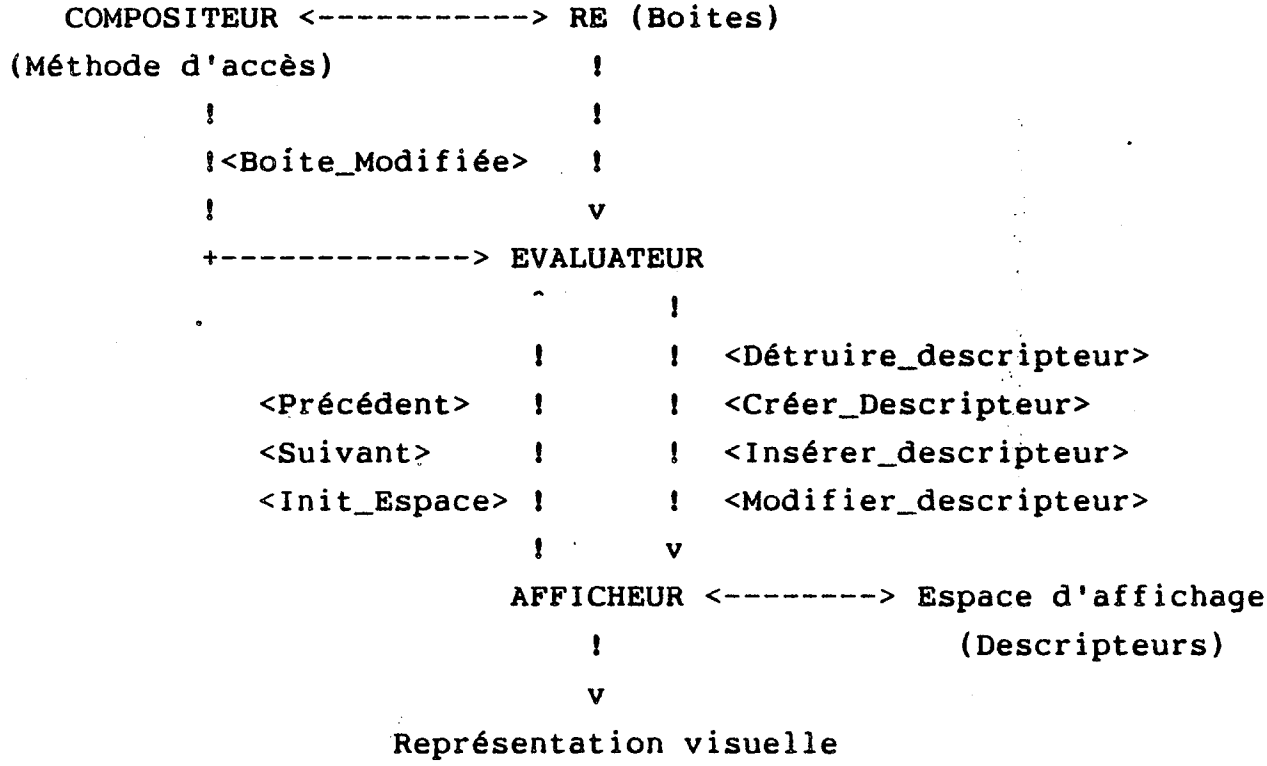
Aspects techniques de la visualisation des Boites

Le Compositeur est constitué d'un ensemble d'entités communicantes, spécialisées chacune dans un certain type de transformation:

- L'évaluateur transforme de manière incrémentale et partielle la RE en Descripteurs de boites feuilles;

- L'afficheur transforme un ensemble de descripteurs de boites feuilles en une représentation visuelle.

L'architecture du Compositeur est la suivante:



Le Compositeur définit un ensemble de primitives de manipulation de l'arbre de boites. Pour maintenir les images projetées dans les fenêtres consistantes avec la description, toutes les primitives de modification du Compositeur signalent à l'évaluateur l'identification de la boite modifiée et la nature de l'opération (primitive Boite_Modifiée).

L'évaluateur détermine les incidences d'une modification en les exprimant en terme de modification de descripteurs ou d'espace d'affichage (primitive Modifier_Descripteur, Créer_Descripteur, Détruire_Descripteur et Insérer_Descripteur).

L'évaluateur est un traducteur incrémental RE-espace d'affichage qui transforme un arbre en des listes de descripteurs de boites feuilles: ce que nous appelons un espace d'affichage.

Chaque descripteur est un atome de la représentation graphique. Dans l'implantation actuelle, seules des informations de type texte sont prises en compte. L'ensemble des descripteurs qui constituent un espace d'affichage est géré par un afficheur qui visualise ces informations interprétées.

Pour gérer cet espace, l'afficheur coopère avec l'évaluateur. Il émet lors de sa création la requête Init_Espace qui crée une première liste de descripteurs. La constitution de l'espace complet s'effectue sous le contrôle de l'afficheur à l'aide de la primitive Suivant qui est appelée jusqu'à ce que l'espace d'affichage contienne les informations suffisantes pour "remplir" la fenêtre.

Les primitives Suivant et Précédent sont employées par l'afficheur lorsque l'usager effectue des défilements verticaux d'informations dans la fenêtre.

Dans ce qui suit, nous présentons les principes généraux de l'afficheur, puis, nous explicitons les caractéristiques essentielles de l'évaluateur.

1. L'AFFICHEUR

L'afficheur gère les informations qui apparaissent dans la zone utile d'une fenêtre. Ces informations sont représentées par une structure de données, que nous appelons "espace d'affichage", qui décrit les informations visibles.

Cette structure de données est constituée de "descripteurs" de boites feuilles (dans l'implantation actuelle). Chacun d'eux contient l'information nécessaire à la visualisation d'une feuille (son aspect, ses dimensions, le texte à visualiser, son positionnement par rapport aux descripteurs voisins).

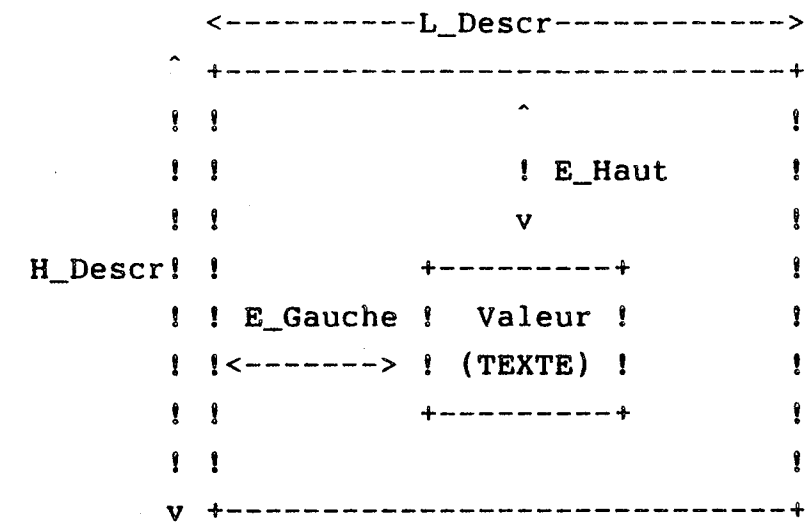
Organisés en listes, ces descripteurs permettent à l'afficheur de réaliser la visualisation des informations sur l'écran.

Pour ce faire, l'afficheur effectue les troncations qui sont nécessaires et il réalise le découpage en lignes et en colonnes.

1.1. Notion de descripteur

Chaque descripteur décrit une surface rectangulaire dans laquelle est projetée la valeur d'une boite; dans l'implantation actuelle, cette valeur est de type texte. Chaque descripteur est lié à la boite feuille qu'il représente et chaque feuille contient l'information qui permet d'accéder au descripteur, s'il existe (référence croisée).

La forme générale d'un descripteur est la suivante:



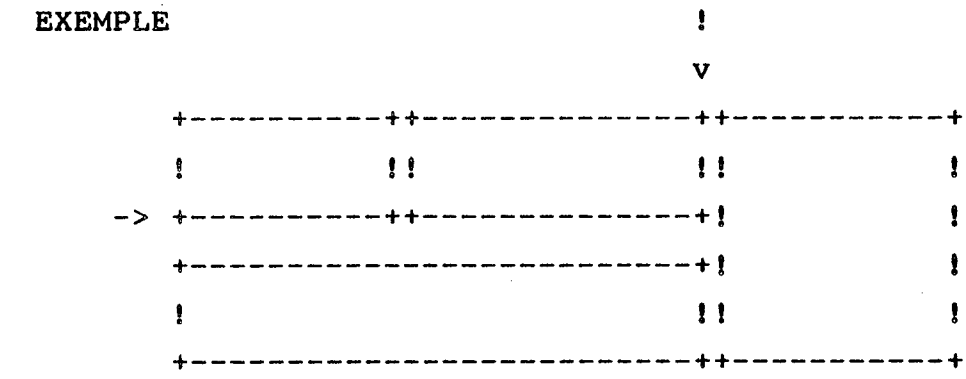
Le descripteur définit une surface rectangulaire de dimensions (L_descr, H_descr) qui sont déterminées de manière à ce que l'information qui y est inscrite (le texte) soit positionnée conformément aux directives portée par les attributs d'espacement et d'indentation.

Le texte définit une surface rectangulaire qui s'incrit dans le rectangle précédent. Il est positionné relativement aux bords haut et gauche du descripteur à une distance E_haut (Ecart par rapport au bord haut) et E_gauche respectivement.

La position de tout descripteur est exprimée par rapport à ses voisins immédiats ou par rapport au bord gauche de la fenêtre.

Ainsi, une liste horizontale de descripteurs ayant le même "voisin bas" ont leurs bords bas alignés. Une liste verticale de descripteurs ayant le même voisin droit ont leurs bords droit alignés.

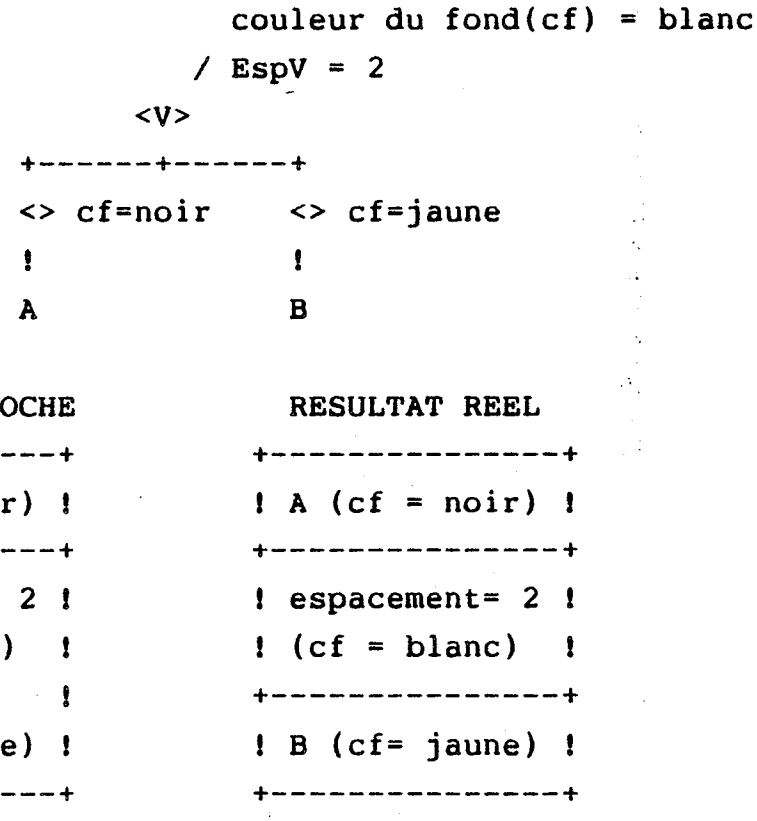
EXEMPLE



Nous ne mémorisons donc aucune coordonnée de descripteur. Ceci nous permet d'optimiser les traitements liés au défilement des informations en évitant de recalculer les coordonnées de chaque descripteur lors d'une telle action.

Le module d'affichage, dans sa forme actuelle, ne permet de visualiser qu'une approximation de l'image spécifiée par une RE. En effet, si le positionnement des informations est conforme à la spécification, leur aspect ne l'est pas.

EXEMPLE

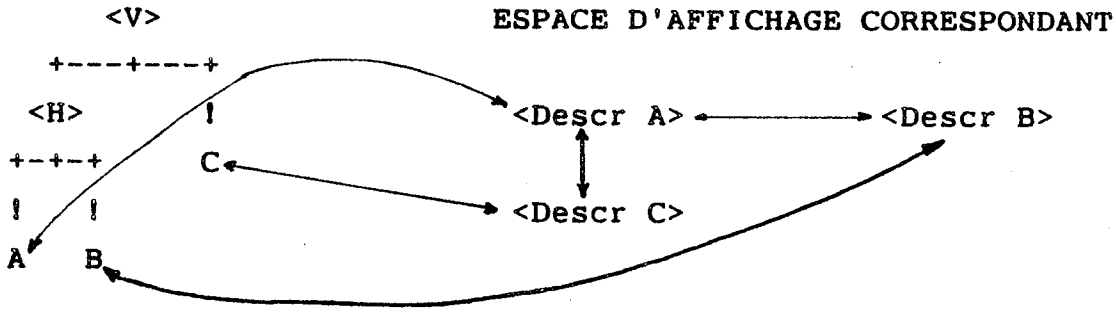


Cette approximation nous a néanmoins permis de définir le principe des techniques que nous pourrons employer dans une version ultérieure.

1.2. La gestion de l'espace d'affichage

Les descripteurs sont disposés relativement les uns par rapport aux autres. Ils sont liés par des relations de voisinage (au-dessus, en-dessous, à droite, à gauche). Un exemple d'espace d'affichage d'une RE complète est donnée ci-dessous.

EXEMPLE



Une RE décrit généralement une image dont les dimensions sont nettement supérieures à celles de la fenêtre où elle est projetée. Mais, on ne conserve sous une forme interprétée que les descripteurs qui décrivent l'image comprise entre les bords haut et bas de la fenêtre. Un espace d'affichage ne représente donc qu'une partie de l'image définie par une RE.

Les défilements verticaux nécessitent alors une coopération entre l'afficheur et l'évaluateur. L'afficheur détruit les descripteurs qui disparaissent de la fenêtre et demande à l'évaluateur ceux qui deviennent visibles.

On ne maintient donc sous une forme interprétée qu'un minimum de descripteurs, ce qui permet d'optimiser les calculs nécessaires à la détermination des incidences visuelles d'une modification en ne les effectuant que pour les boites visibles.

Nous définissons la notion de boite visible comme suit:

- Une boite feuille est visible s'il existe un espace d'affichage qui contient un descripteur de cette feuille.

- Une boite composite est visible si l'une au moins de ses filles est visible.

Remarquons que cette information est synthétisée. Il faut donc la mémoriser dans chaque boite si l'on veut réellement optimiser les temps de réponse.

Pour prendre en compte les modifications de la RE, l'afficheur définit les primitives qui permettent de modifier l'aspect ou les dimensions d'un descripteur (Modifier_descripteur), d'en créer et d'en détruire.

Une primitive (Insérer_descripteurs) permet à l'évaluateur d'insérer une liste de descripteurs dans un espace d'affichage en précisant les relations de voisinage avec les descripteurs déjà existants.

L'ensemble des afficheurs qui projètent une RE sont gérés par un gestionnaire d'afficheurs. Lorsqu'un descripteur est modifié, l'afficheur concerné est inséré dans la liste des afficheurs modifiés.

Cette liste sera consultés par le gestionnaire des afficheurs lorsque l'évaluateur lui aura signalé que le calcul des incidences visuelles induites par une modification est fini.

Il active alors successivement (dans l'implantation actuelle) les afficheurs concernés qui rectifient alors les parties de l'image ayant subi une modification. Ceci permet de limiter les effets visuels désagréables et d'optimiser les actions de repeinture de l'écran.

1.3. Visualisation des descripteurs

Les descripteurs qui constituent l'espace d'affichage décrivent une surface dont la largeur et la hauteur sont généralement supérieures aux dimensions de la fenêtre dans laquelle les informations sont visualisée. L'afficheur doit alors réaliser la troncation (clipping) des descripteurs.

Celle-ci est réalisée conjointement au parcours des listes de descripteurs. Elle s'effectue en deux étapes:

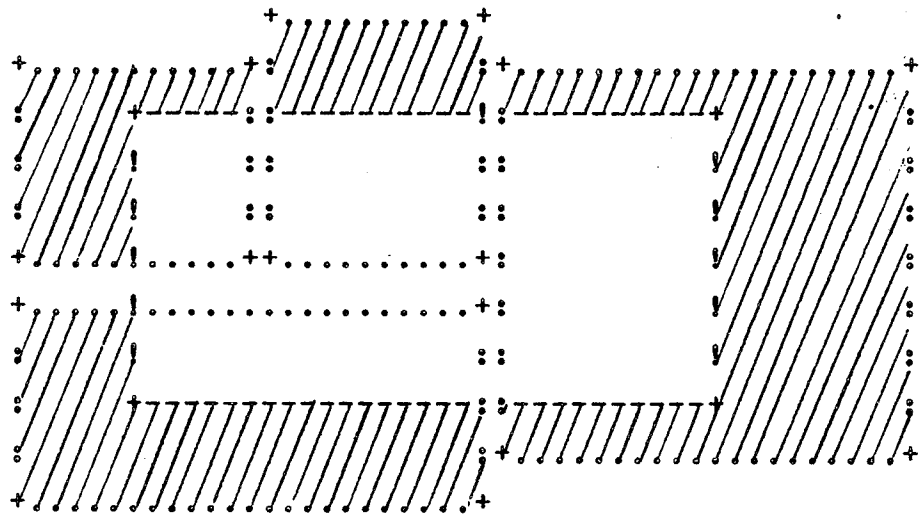
- détermination de la visibilité globale du descripteur.

  On calcule la surface rectangulaire (éventuellement nulle) du descripteur qui apparait dans la fenêtre;

- détermination des lignes de texte visibles.

  Connaissant la partie du descripteur qui est visible (calculée dans l'étape précédente), on réalise le découpage en lignes et en colonnes en utilisant les fonctions du formateur de texte.

EXEMPLE: Espace d'affichage



Remarque: les bords des descripteurs sont en pointillés, les bords de la fenêtre sont en traits pleins.

Apparaissent en hachuré les parties non visibles des descripteurs.

Pour chaque descripteur qui coupe l'un des bords horizontaux de la fenêtre on mémorise le nombre de lignes visibles et le nombre de lignes cachées. Lors d'un défilement vertical ces informations ne sont mises à jour que pour la liste des descripteurs qui coupent l'un de ces bords. On fait disparaitre une ligne sur l'un des bords et apparaitre une nouvelle sur l'autre bord.

Le formateur de texte, qui est intégré au niveau de l'afficheur, définit les notions de ligne "précédente" et "suivante". Il est capable d'afficher une ligne connaissant la position du premier et du dernier caractère visible.

Lors d'un défilement horizontal, il est nécessaire de balayer la liste des descripteurs pour déterminer et afficher ceux des descripteurs qui sont visibles après le décalage. Pour une telle opération, la repeinture de toute la fenêtre est nécessaire si l'on utilise des terminaux alpha-numériques (ce qui est notre cas).

Sur des matériels plus sophistiqués qui possèdent des opérations de déplacement de zone (raster-op), on peut effectuer une repeinture en deux étapes: décalage puis affichage des informations que le défilement horizontal a fait apparaitre dans la fenêtre.

Lors d'une modification de la RE, l'évaluateur calcule ses incidences sur l'ensemble des descripteurs liés aux feuilles. Il traduit les modifications apportées à l'arbre en terme de modification sur des descripteurs.

Il a à sa disposition quatre primitives essentielles:

- Modifier_Descripteur.

- Créer_Descripteur.

- Insérer_Descripteur.

- Détruire descripteur.

Lorsque l'espace d'affichage est modifié par l'évaluateur, tous les descripteurs modifiés sont "marqués" en vue du réaffichage. Dans le cas d'une modification de dimensions ou lors de l'insertion ou de la destruction d'un descripteur, il faut en outre réafficher tous les descripteurs situés à droite de la modification (dans l'implantation actuelle).

1.4. La désignation

Lorsque l'usager désigne une information dans une fenêtre, les coordonnées du point désigné sont transmises à l'afficheur qui effectue la correspondance coordonnées-nom de boite feuille.

2. L'EVALUATEUR

Nous abordons successivement les points suivants:

- les parcours d'arbre nécessaires au défilement des informations;

- la détermination des dimensions d'une boite et le positionnement relatif des descripteurs;

- le partage de la RE entre plusieurs afficheurs et l'incidence de ce partage sur la notion de dimension d'une boite;

- la détermination des conséquences d'une modification sur les espaces d'affichage.

2.1. Défilements et parcours correspondants

Le découpage des informations en lignes et en colonnes est résolu par l'afficheur. Les parcours ne tiennent donc compte que de la structure de l'arbre et des informations de composition qui y sont présentes.

L'avantage en est l'indépendance des algorithmes par rapport aux dimensions et à la nature des informations manipulées.

Comme nous l'avons déjà vu, seuls les défilements verticaux nécessitent une collaboration entre l'afficheur et l'évaluateur. Deux fonctions sont définies; celles que nous avons appelées "précédent" et "suivant".

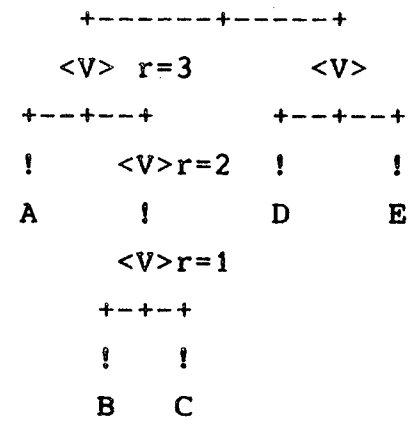
La première étape vers la définition des algorithmes nous amène à présenter ces fonctions pour des arbres où seule la composition verticale est employée.

L'étape suivante nous amène à définir la notion de "liste horizontale" et à montrer comment on l'obtient.

Enfin, nous traiterons le cas d'un arbre où les deux compositions de base sont présentes. Pour celà nous utiliserons les deux algorithmes précédemment définis.

2.1.1. Composition verticale de boites

Soit l'arbre suivant



liste des feuilles liées par les relations précédent et suivant

A
B
C
D
E

Soit C la boite dont on cherche à déterminer le précédent et le suivant.

Pour trouver le "suivant" de C, à savoir D, il faut effectuer le parcours suivant:

1- trouver un ancêtre de rang n (4 dans notre cas) de composition verticale, l'ancêtre de rang (n-1) possèdant une boite soeur droite. (1)

Cette soeur droite est la racine du sous-arbre qui, s'il n'est pas vide (s'il contient une feuille au moins), contient la boite feuille recherchée;

2- Cette dernière est la boite feuille située la plus à gauche dans ce sous-arbre.

3- Si le sous-arbre déterminé en -1- est vide, on réitère l'étape -1- en prenant comme point de départ la racine du

---

(1) L'ancêtre de rang 0 d'une boite X est égal à X

sous-arbre vide.

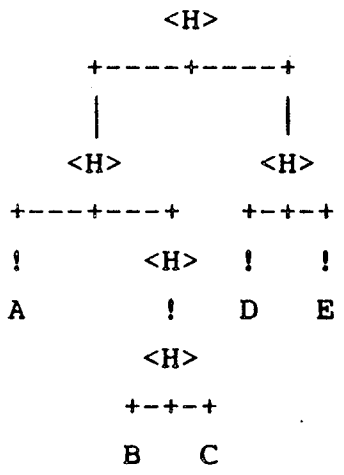
4- Lorsque l'ancêtre de rang (n) est la racine de l'arbre et si la racine précédente n'a pas de soeur droite, alors il n'existe pas de Suivant.

De manière similaire, nous pouvons définir le "précédent" d'une boite feuille comme étant la boite qui apparait immédiatement au-dessus de la boite "courante": précédent(C)=B.

Trouver le précédent d'une boite feuille consiste à appliquer l'algorithme précédent en remplaçant respectivement gauche par droite et droite par gauche (les deux parcours, Précédent et Suivant, sont symétriques).

2.1.2. Notion de "liste horizontale"

Soit l'arbre suivant:



La liste des feuilles qui apparaissent sur une même horizontale est la suivante

A B C D E

La notion de liste horizontale n'est définie que dans un sous-arbre dont la racine est de composition horizontale. La liste (A, B, C, D, E) en est un exemple.

Elle est obtenue en recherchant dans un arbre dont la racine est de composition horizontale l'ensemble des feuilles liées par la relation "sur une même horizontale" que l'on peut atteindre à

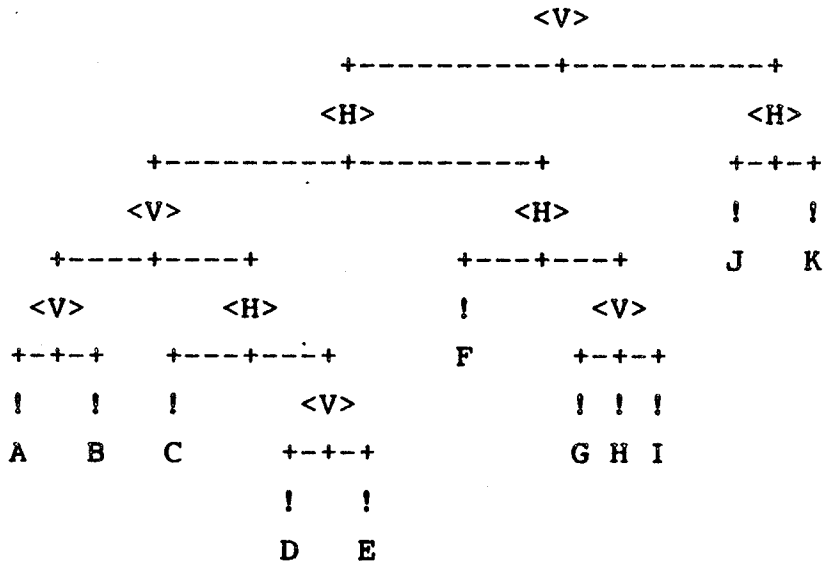
partir d'elle.

Cette liste est obtenue en parcourant les sous-arbres issus de la racine, de la gauche vers la droite. Si c'est une feuille, elle appartient alors à cette liste. On réitère ce processus toutes les fois que l'on atteint une boite composite.

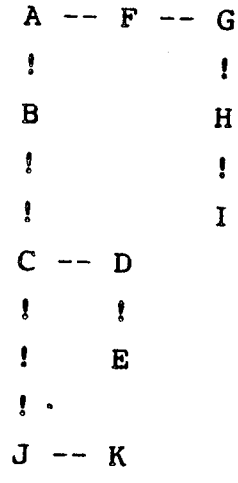
Le sens du parcours à son importance puisqu'il permet d'obtenir une liste déjà ordonnée.

2.1.3. Composition mixte

Soit l'arbre suivant:



Les dispositions correspondantes sont les suivantes:



Dans ce qui suit, nous appellerons "liste des suivants" (liste des précédents) d'une feuille, la liste horizontale des descripteurs de feuilles qui apparaissent immédiatement en-dessous (au-dessus) de la feuille considérée.

Trouver la liste des suivants d'une feuille c'est:

1- trouver l'ancêtre de rang (n), de composition verticale, l'ancêtre de rang (n-1) possédant une soeur droite.

Cette soeur droite est la racine d'un sous-arbre contenant une liste horizontale éventuellement vide.

2- Si la composition associée à la boite soeur est verticale alors il faut rechercher la boite feuille la plus à gauche dans le sous-arbre.

Si la composition est horizontale, il faut appliquer l'étape -2- à chacune des filles du sous-arbre, en effectuant le parcours de la gauche vers la droite (phase de détermination de la liste horizontale).

Déterminer la liste des précédents, consiste à appliquer l'algorithme que nous venons de décrire en remplaçant respectivement gauche par droite et droite par gauche dans les

deux étapes, sauf dans la phase de détermination de la liste horizontale (la symétrie est conservée).

EXEMPLES:

Liste_des_suivants (C) = (J, K)
Liste_des_précédents (C) = (B)

Rappelons qu'appartiennent à l'espace d'affichageé tous les descripteurs nécessaires à la visualisation de l'image sur TOUTE sa largeur. Ainsi, lorsque les descripteurs J et K ne sont pas visibles, les descripteurs A ou B ou C et (D ou E), F et (G ou H ou I) appartiennent à l'espace d'affichage.

Ceci nous permet de déterminer de manière incrémentale les informations qui "entrent" dans l'espace d'affichage lors d'un défilement vertical et nous permet de retrouver l'ensemble des listes horizontales qui décrivent l'image sur toute sa largeur.

REMARQUES:

- Nous avons réduit le problème posé par la composition horizontale de boites en considérant les boites filles d'une telle composition comme autant de compositions verticales.

- Les algorithmes de parcours sont indépendants des dimensions des boites et de la nature des informations qu'elles contiennent. Nous pouvons donc à terme visualiser du texte et du graphique.

- Les informations de nature structurelle ont disparu. Ne restent plus que les informations essentielles: les feuilles et leurs relations de disposition relatives.

Le positionnement exact des informations les unes par rapport aux autres fait l'objet de ce qui suit.

2.2. Dimensions d'une boite et positionnement des informations

L'une des originalités du module d'affichage réside dans le fait que nous ne mémorisons pas les coordonnées des boites. Nous déduisons le positionnement des informations connaissant les dimensions des boites (hauteur et largeur).

Quel est l'intérêt de ce procédé?

Si l'on mémorise les coordonnées dans chaque boite il faut les recalculer lors de chaque modification de dimensions qui intervient dans l'arbre (ajout ou suppression d'une boite). Dans le cas le plus défavorable, il faut parcourir tout l'arbre (agrandissement d'une information qui apparaitrait en haut d'une fenêtre de dimensions infinies).

Dans notre cas, il suffit de répercuter une telle variation de dimensions (remontée dans l'arbre) puis recalculer le positionnement relatif des descripteurs uniquement pour les boites visibles.

Dans le meilleur des cas le traitement se résume à une remontée dans l'arbre (le plus petit sous-arbre qui subit une variation de dimensions n'est pas visible). Dans le cas le plus défavorable, nous parcourons tout l'arbre.

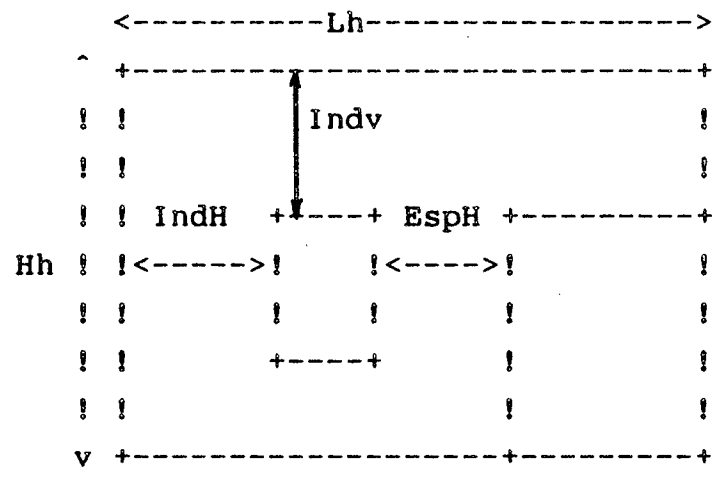
Remarquons que l'arbre est alors suffisamment petit pour que l'image qu'il décrit tienne dans la fenêtre (25 lignes, 80 colonnes au maximum).

2.2.1. Calcul des dimensions d'une boite

Nous évaluons le couple largeur, hauteur (L, H) d'une boite composite B, connaissant le couple (li, hi) des boites b1, ..,bi, ..,bn constituant la boite B. Ce calcul s'effectue à partir des attributs de composition (H, V) et des attributs EspV, EspH, IndV

et IndH associés ou hérités par B.

a. Cas d'une composition horizontale



Les relations suivantes définissent les dimensions de la boite B:

$$Lh = IndH + \sum_{i=1}^{n} li + (n-1).EspH$$

$$Hh = IndV + \max_{i \in (1,n)} hi$$

b. Cas d'une composition verticale

Les expressions suivantes définissent les dimensions d'une boite composite à laquelle une composition verticale est associée (les expressions sont symétriques):

$$Lv = IndH + \max_{i \in (1,n)} li$$

$$Hv = IndV + \sum_{i=1}^{n} hi + (n-1).EspV$$

c. Dimensions d'une boite feuille de type texte

Ces dimensions dépendent évidemment de l'algorithme de formatage employé et sont fonction de la valeur de la contrainte qui est imposée sur la largeur d'une boite feuille (valeur associée ou héritée).

Dans l'implantation actuelle, nous replions tout simplement les lignes; dans ce cas, les dimensions sont données par les relations suivantes:

L = Min (nombre de caractères de la boite, largeur maximale)

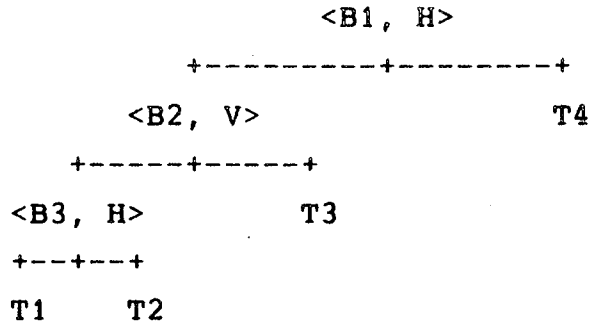
H = nombre de caractères / largeur maximale
si le reste de la division est nul, augmenté de 1 sinon

2.2.2. Positionnement des informations

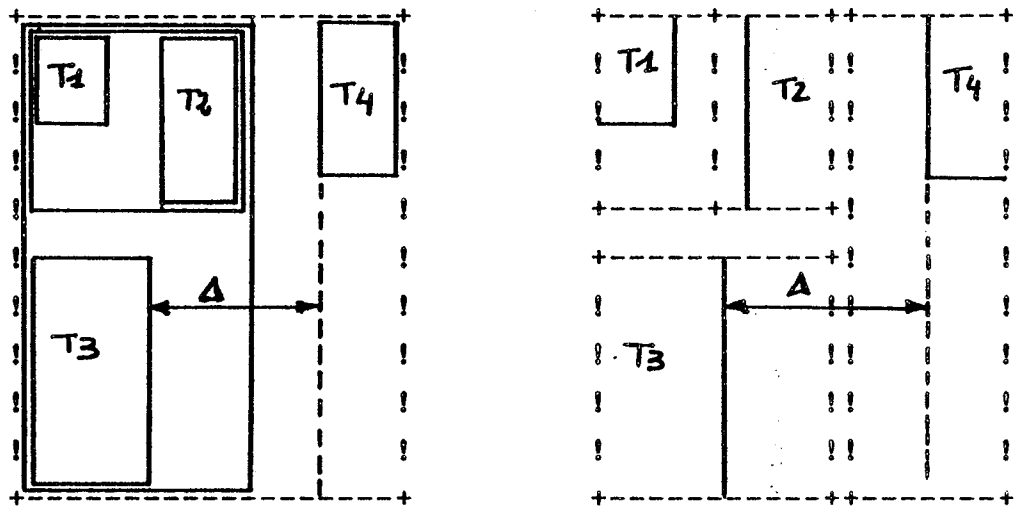
Ces informations sont calculées à l'aide de relations récursives qui nécessitent la connaissance des dimensions des boites, des attributs de composition, d'espacement et d'indentation.

Les équations qui permettent d'aboutir au résultat ne présentant pas un grand intérêt, nous ne les exposons donc pas. Le principe en sera néanmoins expliqué sur l'exemple ci-dessous.

Soit l'arbre suivant:



Les deux schémas qui suivent décrivent (celui de gauche) les imbrications et le positionnement réels des boites et (celui de droite) l'espace d'affichage qui lui correspond qui n'est qu'une approximation de l'image réelle (cf 1.1).



La distance que nous avons notée Δ est obtenue en tenant compte des différences de largeurs des boites T3 et B2:

$$\text{Largeur (B2)} = \text{Max (Largeur(B3), Largeur(T3))}$$

et en tenant compte des éventuelles indentations et espacements horizontaux hérités ou associés à la boite T4. La valeur de Δ est donnée par la relation générale suivante:

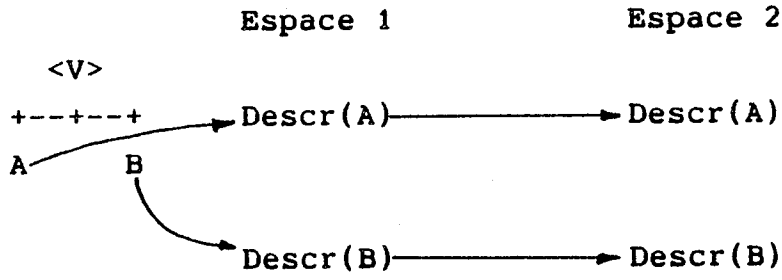
$$\Delta = \text{Largeur(B2)} - \text{Largeur(T3)} + \text{EspH(B1)} + \text{IndH(T4)}$$

2.2.3. Partage d'une RE entre plusieurs afficheurs

Nous avons vu dans ce qui précède qu'une boite feuille était représentée par un descripteur dans l'espace d'affichage associé à un afficheur.

Partager une RE entre plusieurs afficheurs consiste à représenter une même feuille par plusieurs descripteurs et, à maintenir la cohérence entre les descripteurs et la RE.

EXEMPLE



Il suffit alors de répercuter toute modification qui affecte une feuille vers les différents espaces d'affichages où cette information est représentée; on active la primitive "modifier_descripteur" sur la liste des descripteurs associés à la boite feuille.

Nous avions dit qu'à chaque afficheur était associé un niveau de visibilité et que les algorithmes de parcours étaient paramétrés par cette information.

S'il est simple de détecter qu'une information est invisible (il suffit de calculer la visibilité d'une boite et de la comparer au niveau de visibilité), le problème qui se pose est de savoir comment positionner convenablement les informations les unes par rapport aux autres.

Nous avons vu que le positionnement relatif était calculé à partir des dimensions des boites. Or, lorsqu'une boite n'est pas visible par un afficheur, tout se passe comme si ses dimensions

étaient nulles.

D'autre part, ces dimensions sont des valeurs de synthèse. Pour obtenir des temps de réponse acceptables, nous avons été amenés à mémoriser ces informations dans chaque boite. Pour deux afficheurs qui possédent des niveaux de visibilités distincts, les dimensions des boites apparaissent comme différentes.

La technique que nous employons pour résoudre ce problème, consiste à associer à chaque boite une liste de dimensions. Chaque élément de cette liste donne les dimensions de la boite pour un niveau de visibilité donné. Pour une boite ne sont calculées que les dimensions effectivement nécessaires.

EXEMPLE

Si une boite apparait dans trois espaces d'affichage possédant des niveaux de visibilités distincts, les trois dimensions précalculées correspondent chacune à un niveau de visibilité.

Si deux des trois espaces d'affichage possèdent le même niveau de visibilité, seules deux dimensions sont calculées.

Le calcul des dimensions des boites a lieu lors de l'association entre un sous-arbre et un afficheur en tenant compte du niveau de visibilité de celui-ci (on détermine les dimensions de tout le sous-arbre, si ce n'est déjà fait).

2.2.4. Incidences d'une modification de la RE

Deux types de modification sont possibles:

- les modifications d'aspect (changement de couleur, passage d'un sous-arbre en inverse vidéo, etc);

- les modifications de dimensions et plus généralement les modifications de structure.

a. Les modifications d'aspect

Une telle modification résulte du changement de la valeur d'un attribut hérité; affectation de l'attribut inverse vidéo à un sous-arbre par exemple.

Si la racine de ce sous-arbre n'est pas visible (cf 1.1) nous n'effectuons aucun traitement. Si elle l'est, nous descendons vers les feuilles visibles du sous-arbre et modifions l'aspect de la liste des descripteurs accessibles à partir de chaque feuille (utilisation de la référence croisée).

b. Les modifications de dimensions

Elles résultent d'une modification de la valeur d'une boite feuille ou de celle d'un attribut d'espacement ou d'indentation.

Dans ce cas, nous sommes amenés à recalculer les dimensions du sous-arbre, puis à répercuter la variation de dimensions résultante vers les ancêtres de la boite racine du sous-arbre (les dimensions sont des valeurs de synthèses).

A partir de la boite racine du plus petit sous-arbre affecté par cette variation de dimensions, nous devons recalculer, pour tous les afficheurs concernés, le positionnement des informations. Pour se faire, nous redescendons vers les feuilles visibles et déterminons les grandeurs nécessaires au positionnement de chaque descripteur.

c. Les modifications de structure

Modifier la structure d'une RE consiste à détruire ou à insérer une boite (éventuellement composite). Remarquons qu'une telle modification s'accompagne généralement d'une variation de dimensions.

Dans le cas d'une destruction, nous détruisons tous le sous-arbre et tous les descripteurs accessibles à partir de celui-ci. Puis nous effectuons le traitement précédemment décrit pour déterminer et répercuter les variations de dimensions résultantes.

Dans le cas d'une insertion, le traitement est plus complexe car il faut déterminer si l'information insérée doit apparaitre dans un espace d'affichage.

Nous commençons alors par répercuter vers les espaces d'affichage concernés les effets induits par les variations de dimensions, puis nous déterminons les prédécesseurs ou suivants éventuels pour chaque afficheur concerné. Ceci est suivi de l'insertion de descripteurs dans les espaces d'affichage concernés.

CHAPITRE 5

Conclusion

L'approche que nous avons développée consiste à séparer la partie sémantique des applications de leur composante communication qui est intégrée dans un complexe logiciel que nous appelons interface usager.

L'objectif de cette interface est de faciliter la réalisation des applications interactives en proposant à leurs concepteurs des abstractions qui permettent de découpler le logiciel du dialogue avec l'usager.

La relation entre l'interface usager et l'application repose sur l'existence d'un protocole de dialogue logique qui est une représentation canonique du dialogue avec l'usager.

L'acquisition des commandes est réalisée par le Médiateur qui traduit le résultat d'un dialogue dans les termes de ce protocole. Le Médiateur propose à l'usager trois techniques de communication standards, disponibles simultanément: le menu, le formulaire et le langage de commande. L'usager peut privilégier l'une d'elles, tout en conservant la possibilité de changer de mode de dialogue lorsqu'il en éprouve le besoin.

Le Compositeur met à la disposition des applications des mécanismes qui masquent les problèmes liés à la visualisation et à la gestion des informations visuelles. Il permet à l'usager de manipuler ce qu'il voit (désignation, édition) et d'organiser à sa guise les informations sur l'écran (création de fenêtres qui visualisent des portions d'images sélectionnées par exemple).

Un temps de réponse faible est obtenu en employant des techniques incrémentales d'évaluation d'images et en optimisant le nombre des caractères envoyés vers le terminal (dans l'implantation actuelle). La limitation la plus importante est actuellement le faible débit de la ligne (1200 et 4800 bauds).

Enfin, le gestionnaire de fenêtres définit les fonctions nécessaires à la partition de l'écran et donne à l'usager un contrôle prioritaire sur l'évolution des différentes activités parallèles.

Remarquons que l'ensemble des possibilités d'adaptation et de personnalisation définies dans notre interface ne dépendent pas de l'application interfacée. La décentralisation des traitements habituellement réalisés par l'application nous permet d'obtenir une interface usager qui, vue de l'utilisateur, est à la fois cohérente, souple et adaptable (dans certaines limites).

Les facilités offertes aux concepteurs d'applications et aux usagers imposent en contrepartie l'emploi d'un calculateur puissant, tant en capacité mémoire (un à deux Moctets) qu'en puissance de calcul (un Mips). De telles capacités ne sont actuellement disponibles que sur des machines haut de gamme telles que le Perq, l'Appolo et la SM 90.

1. LES APPORTS

L'interface usager que nous avons définie est un ensemble intégré d'outils dont ni la nature, ni les concepts ne sont fondamentalement nouveaux, bien que certaines des techniques employées possèdent des caractéristiques originales.

L'étude présentée montre l'importance de l'architecture d'une interface usager. Sa définition en terme d'entités coopérantes rend possible la décentralisation du flot de contrôle, permet de définir des actions génériques et résoud le problème de

l'intégration des composants.

Le transfert dans l'interface de certains traitements (édition, acquisition des commandes, analyse syntaxique) et informations (relations structurelles et dépendances entre les informations visualisées) habituellement inclus dans les applications, permet de réduire les problèmes liés à leur conception.

Cette "décentralisation des compétences" permet de réaliser des logiciels qui donnent à l'usager la possibilité d'adapter la forme du dialogue ou du poste de travail indépendamment de l'application utilisée.

2. LES LIMITATIONS

Dans son implantation actuelle, l'interface usager que nous avons définie ne peut raisonnablement être utilisée que pour des applications dont la syntaxe du langage de commande est de la forme <opérateur> <opérande>.

En effet, le Médiateur définit la syntaxe des commandes sans donner aux concepteurs la possibilité de la modifier. La solution proposée au chapitre 3 n'est pas satisfaisante dans la mesure où elle conduit à inclure dans l'application:

- une partie du flot de contrôle, ce que nous voulons éviter;

- une partie de la vérification syntaxique des commandes, ce qui rend notre interface non homogène et ne facilite pas la réalisation des applications.

Le Compositeur ne permet de définir que des relations hiérarchiques entre les constituants d'une image et les seules contraintes que l'application puisse exprimer sont celles de positionnement des informations (composition) et de largeur

maximum d'une boite feuille.

Si ces possibilités se révèlent être suffisantes pour l'affichage de programmes, de formulaires ou de documents très simples, elles sont nettement insuffisantes dans un cadre plus général.

Dans l'état actuel, il est par exemple impossible de définir des tableaux: la relation entre les éléments ne suit pas le modèle hiérarchique. En effet, les dimensions de chacun d'eux sont déterminées à partir de la largeur de la colonne et de la hauteur de la ligne auxquelles ils appartiennent.

Pour résoudre ce problème, nous prévoyons de superposer au modèle hiérarchique un modèle relationnel qui permettra d'exprimer des contraintes des dimensions entre éléments distants. Nous serons alors confrontés au problème de l'évaluation incrémentale d'un arbre comportant des relations lointaines <Rouzaud 84>.

Un tel mécanisme permettra par exemple de réaliser des tableaux sans définir la notion de boite tableau. Si de plus on introduit des boites feuilles portant des informations de nature graphique, il nous sera possible de faciliter par exemple la réalisation d'éditeurs de formules mathématiques ou de documents.

L'idée de base du Compositeur est que l'image et les relations entre ses composantes sont décrites par l'application. De cette description et des relations qui sont exprimée nous pouvons générer une image.

Ceci est l'une de ses originalités et très certainement sa principale source de limitation. Le problème posé est celui des limites de cette approche; question pour laquelle nous ne possédons pas de réponse.

3. LES PERSPECTIVES

Les perspectives à court terme sont:

- le développement d'un gestionnaire de dialogue qui, contrairement au Médiateur soit paramétré par une description de la syntaxe. Cette description est employée pour l'analyse des commandes émises par l'usager, elle ne pourra très certainement pas être employée pour générer automatiquement les différentes formes de dialogue que nous voulons offrir à l'usager; un "éditeur de dialogue" devra être défini à cette fin.

- l'extension des possibilités de description du Compositeur en y incluant des boites feuilles graphiques et en utilisant le modèle précédemment défini (hiérarchique + relations distantes). Nous désirons enrichir le noyau d'édition de texte en y incluant l'édition structurelle de la RE et répondre ainsi aux besoins des éditeurs syntaxiques <Mélèze 84, Meyer 84>.

A plus long terme, nous voulons faciliter la réalisation des éditeurs dirigés par la syntaxe <Mélèze 84, Meyer 84> en redéfinissant leur relation avec l'interface usager.

Dans l'état actuel, ces éditeurs définissent un ensemble de commandes qui permettent à l'utilisateur d'éditer un programme en employant des schémas prédéfinis. A partir de ces commandes, l'éditeur modifie sa RI puis génère l'image qui lui correspond (décompilation incrémentale).

En intégrant la notion de modèle dans l'éditeur structurel inclus dans l'interface usager, nous pouvons inverser ce processus: à partir d'une modification de la RE, l'éditeur syntaxique met à jour sa RI. Les modifications de la RI étant déduites de celles de la RE, la décompilation ne sera plus nécessaire.

Minimiser la taille de la mémoire centrale requise par le Compositeur est l'un des problèmes qu'il nous faudra résoudre. Nous pensons réaliser dans ce but une mémoire cache qui ne contiendra que les portions de RE utiles à un instant donné: celles qui sont vues, plus celles qui permettent d'anticiper les actions de l'usager.

Pour gérer cette mémoire cache, nous développerons un processus qui utilisera une heuristique basée sur la notion de visibilité; les portions de la RE qui ne sont pas dans cette mémoire cache seront conservées sur disque.

L'idée de définir une commande capable de défaire toutes les commandes qu'un usager peut émettre (commande "Undo" ou "Défaire") est largement développée dans la littérature. Nous pensons définir une telle commande pour les modifications non désirées d'informations visibles.

L'idée est simple: la relation entre la RI et la RE repose sur le principe de l'équivalence de ces deux représentations. Si l'on impose que l'application soit capable de générer une représentation interne équivalente à une partie de RE, il est alors possible d'apporter une solution externe à l'application au problème de la commande "défaire": il suffit de mémoriser (sur disque par exemple) toutes les informations qui permettent de régénérer une RE ayant subi une modification non désirée et de simuler l'envoi vers l'application des commandes qui permettent de retrouver l'état initial de la RE.

Ceci nous amène à étudier et à définir plus précisément la relation entre les applications et l'interface usager. De manière plus générale, il nous faudra préciser les relations entre les différents constituants (processus) de l'interface et affiner la notion de coopération.

Nous devrons alors définir un réel protocole de communication inter-processus qui tienne compte des priorités des informations

qui transitent et des problèmes liés à la synchronisation des processus (il nous faudra par exemple assurer l'exclusion mutuelle des accès à la RE entre l'usager et l'application).

Nombre de problèmes évoqués dans cette thèse restent ouverts; nous sommes persuadés que leur résolution n'est possible qu'en effectuant une synthèse des techniques employées aujourd'hui dans des domaines tels que les systèmes répartis, le graphique, les systèmes orientés objets et l'intelligence artificielle.

bibliographie

(Allen 81)
Allen T., Nix P., Perlis A., Pen: A Hierarchical Document Editor, ACM Sigplan Notices, vol.16 (juin 1981)

(Ball 80)
Ball E., Hayes P., Representation of Task-Specific Knowledge in a Gracefully Interacting User Interface, Rapport de recherche CMU (avril 1980)

(Ball 82a)
Ball E., Hayes P., A Test-Bed for User Interface Design, Human Factors in Computer Systems, Gaitherburg, Maryland (mars 1982)

(Ball 82b)
Ball J.E, Canvas: the Spice Graphics Package, Spice Document S108 CMU (juillet 1982)

(Barnard 82)
Barnard P., Hammond N., Maclean A., Morton J., Learning and Remembering Interactive Command, Human Factors in Computer Systems, Gaitherburg, Maryland (mars 1982)

(Benbassat 81)
Benbassat Izak, Dexler Albert S., An Experimental Study of the Human/Computer interface, ACM, vol.24, 11 (novembre 1981)

(Black 82)
Black J. B., Moran T. P., Learning and Remembering Command Names, Human Factors in Computer Systems, Gaitersburg, Maryland (Mars 1982)

(Borufka 82)
Borufka H.G, Kuhlmann H.W, Dialogue Cells: A Method for Defining Interactions, IEEE CG&A (Juillet 1982)

(Boullier 80)
Boullier P, Génération automatique d'analyseurs syntaxiques avec rattrapage d'erreur, Journées Francophones "production assistée de logiciel", Genève (janvier 1980)

(Bratley 83)
Bratley P., Coray G., Tiphane G., Compo: un langage de description de textes, Actes provisoires des journées Manipulation de documents (4-6 mai 1983)

(Buxton 83)
Buxton W., LAmb M.R, Sherman D., Smith K.C, Towards a Comprehensive User Interface Management System, Computer Graphics, vol.17, 3 (juillet 1983)

(Chamberlin 81)
Chamberlin D., King J., Slutz D., Todd S., Wede B., JANUS: An Interactive System for Document Composition, ACM Sigplan Notices, vol.16, 6 (juin 1981)

(Coutaz 82)
Coutaz J., Gandalf Window Package Proposal, Draft C.M.U (Octobre 1982)

(Coutaz 84)
Coutaz J., Herrmann M., Adèle et le Médiateur-Compositeur ou comment rendre une application interactive de l'interface usager, congrès AFCET, 2eme Colloque Génie Logiciel, Nice 4-6 Juin, (1984)

(Ghoul 83)
Ghoul S., Base de données et gestion de configurations dans un atelier de génie logiciel, Thèse de docteur-ingénieur, Grenoble, décembre, (1983)

(Giboin 83)
Giboin, Ergonomie cognitive des systèmes de manipulation de documents, Journées manipulation de documents, Rennes (1983)

(Good 81a)
Good M., Etude and the Folklore of User Interface, ACM Sigplan Notices, vol.16, 6 (juin 1981)

(Good 81b)
Good Michael, Etude and the Folklore of User Interface Design, Symposium on Text Manipulation (ACM SIGPLAN SIGOA), Portland, Oregon (juin 1981)

(Graphics 77)
Graphics, Status Report of the Graphic Standard Planning Commitee of ACM/SIGGRAPH, Computer Graphics, vol.11, 3 (juillet 1977)

(Halasz 82)
Halasz F., Moran T., Analogy Considered Harmful, Human Factors in Computer Systems, Gaitherburg, Maryland (mars 1982)

(Hammer 81)
Hammer M., Ilson R., Anderson T., Gilbert E., Good M., Niamir B., Rosenstein L., Schoichet S., The Implementation of Etude, an Integrated and Interactive Document Production System, ACM Sigplan Notices, vol.16, 6 (juin 1981)

(Hanau 80)
Hanau P.R, Lenorovitz D.R, Prototyping and Simulation Tools For User/Computer Dialogue Design, Computer Graphics, vol.14, 3 (juillet 1980)

(Hansen 71)
Hansen W.J, Creation of Hierarchy Text with a Computer Display, PhD Thesis, Stanford University (octobre 1971)

(Hayes 81)
Hayes P., Carbonell J.G, Multi-Strategy Parsing And its Role in Robust Man-Machine Communication, Rapport de Recherche CMU-CS-81-118 (mai 1981)

(Hayes 80)
Hayes P., Mouradian G., Flexible Parsing, Rapport de recherche CMU-CS-80-122 (Mai 1980)

(Hayes 79)
Hayes P., Reddy R., An Anatomy of Graceful Interaction in Spoken and Written Man-Machine Communication, Rapport de recherche CMU (septembre 1979)

(Hayes 83)
Hayes P., Szekely P., Graceful Interaction through the Cousin Command Interface, Rapport de recherche CMU-CS-83-102, (1983)

(Hayes 84)
Hayes P.J, Executable Interface Definitions Using Form-Based Interface Abstractions, Rapport de Recherche CMU-CS-84-110 (mars 1984)

(Hemenway 82)
Hemenway K., Psychological Issues in Command Menus, Human Factors in Computer Systems, Gaitersburg, Maryland (Mars 1982)

(Huet 77)
Huet G., Kahn G., Maurice P., Environnement de programmation Pascal, Sesori Laboria (novembre 1977)

(Joloboff 84)
Joloboff V., Quint V., Two-dimensional editing, rapport TIGRE (Juin 1984)

(Kamran 83)
Kamran A., Feldman M.B, Graphics computing Independant of Interaction Techniques and Styles, Computer Graphics (janvier 1983)

(Kasik 83)
Kasik D.J, A User Interface Management System, Computer Graphics, vol.16, 3 (juillet 1983)

(Knox 78)
Knox K., BCPL Menu Package Description, Xerox Corporation (Juillet 1978)

(Knuth 79)
Knuth D.E, TEX and Metafont: new directions in typesetting, Digital Press, (1979)

(Lantz 79)
Lantz K.A., Rashid R.F, VTMS: A Virtual Terminal Management for RIG, Rapport de recherche Tr 44, Computer Science Department, University of Rochester (mai 1979)

(Lenne 84)
Lenne C., Interprétation et mise au point de programmes dans un atelier de génie logiciel, Thèse de 3ème Cycle, Grenoble (1984 à paraitre)

(Lieberman 82)
Lieberman H., Designing Intercative Systems from de User's Viewpoint, ECICS, Stresa, Italie (Septembre 1982)

(Lipkie 82)
Lipkie D.E, Evans S.R, Newlin J.K, Weissman R.L, Star Graphics: An Object-Oriented Implementation, Computer Graphics, vol.16, 3 (juillet 1982)

(Lodding 83)
Lodding K. N., Iconic Interfacing, IEEE CG&A (Mars/Avril 1983)

(Mallgren 82)
Mallgren W.R, Formal Specification of Graphic Data Types, ACM Transactions on Programming Languages and Systems, vol.4, 4 (octobre 1982)

(Malone 82)
Malone T. W., How do People Organize their Desks? Implications for the Design of the Design of Office Information Systems, ECICS, Stresa, Italie (Septembre 1982)

(Mankins 83)
Mankins D, Franklin D, A Simple Window Management Facility for the Unix Timesharing System, Summer 83 Toronto Conference Proceedings, Toronto (1983)

(Martinez 82)
Martinez F., Vers une approche systematique de la synthèse d'image. Aspects logiciel et matériel, These d'état INPG (novembre 1982)

(Meyer 84)
Meyer B., Nerson J.M, CEPAGE: Un Editeur Structurel Pleine Page, 2ème Colloque de Génie Logiciel (AFCET), Nice (juin 1984)

(Meyrowitz 81)
Meyrowitz N., Moser M., Bruwin: An Adaptable Design Strategy for Window Manager / Virtual Terminal Systems, Communication ACM (decembre 1981)

(Mikelson 81)
Mikelson M., Pretty Printing in an Interactive Programming Environment, ACM Sigplan Notices, vol.16, 6 (juin 1981)

(Moran 81)
Moran T. P., The Command Language Grammar: a representation for the user interface of interactive computer systems, Int. J. Man-Machine Studies, vol.15 (1981)

(Mélèze 84)
Mélèze B., Edition Structurée-édition non structurée. Coopération et complémentarité, 2ème Colloque de Génie Logiciel (AFCET), Nice (juin 1984)

(Naffah 79)
Naffah, Exemple d'un poste de travail burotique à interface universelle, INRIA, Projet Pilote Kayak, MEV.2.503 (juin 1979)

(Nanard 83)
Nanard M., Nanard J., GTX: un système interactif de manipulation directe de la structure de documents illustrés, Actes des journées sur la manipulation de documents (INRIA), Rennes (mai 1983)

(Notkin 84)
Notkin D., Interactive Structure-Oriented Computing, rapport de recherche CMU-CS-84-103 (fevrier 1984)

(Olsen 83a)
Olsen D.R, Dempsey E.P, Syngraph: A Graphical User Interface Generator, Computer Graphics, vol.17, 3 (juillet 1983)

(Olsen 83b)
Olsen D.R., Dempsey P., Syntax Directed Graphical Interaction, Sigplan Notices, vol.18, 6 (juin 1983)

(PERQ 82)
PERQ, ICL Perq, System Software Reference, International Computers Limited, (1982)

(Pilote 83)
Pilote M., A Data Modeling Approach to Simplify the Design of User Interfaces, 9th Int. Conf. on VLDB, Florence, Italy (octobre 1983)

(RR 299)
RR 299, Adèle un atelier de développement de logiciel, rapport de recherche IMAG , 299 (avril 1982)

(Reenskang 81)
Reenskang T., User Oriented Description of Smalltalk Systems, Byte, vol.6, 8 (aout 1981)

(Robert 84)
Robert J.K Jacob, User-level Window Managers for Unix, Proceedings of the uniform conference on Unix (janvier 1984)

(Rosenberg 82)
Rosenberg J., Evaluating the Suggestiveness of Command Names, Human Factors in Computer Systems, Gaitersburg, Maryland (Mars 1982)

(Rosenthal 81)
Rosenthal D.S.H, "Methodology in Computer Graphics" Re-Examined, Computer Graphics, vol.15, 2 (juillet 1981)

(Rosenthal 82)
Rosenthal D.S.H, Managing Graphical Resources, Workshop on Graphics Input Interaction at Seattle (ACM-SIGGRAPH), Washington (juin 1982)

(Rouzaud 84)
Rouzaud Y., représentation et manipulation de programmes dans un atelier de génie logiciel, Thèse de docteur-ingénieur, Grenoble (1984)

(Santana 83)
Santana M., Un système de production automatique de générateurs de code, Thèse de 3ème Cycle, Grenoble (Décembre 1983)

(Savage 82)
Savage Ricky E., Habinek James K., Barnhart Thomas W., The Design, Simulation and Evaluation of a menu driven User Interface, Human Factors in Computer Systems, Gaitersburg (mars 1982)

(Scapin 82)
Scapin D., Computer Commands Labelled by Users Versus Imposed Commandss and the Effect of Structuring Rules on Recall, Human Factors in Computer Systems, Gaitersburg, Maryland (Mars 1982)

(Shaw 83)
Shaw M., Borisson E., Horowitz M., Lane T., Nichos D., Pausch R., Descartes: A Programming-Language Approach to Interactive Display Interfaces, Sigplan Notices, vol.18, 6 (juin 1983)

(Shneiderman 79)
Shneiderman B., Software Psychology, Winthrop Publishers Inc , (1979)

(Sidner 82)
Sidner C.L, Vittal J.J, Knowledge Representation Tools for the Design, Training and Use of Information Systems, ECICS, Stresa, Italie (septembre 1982)

(Smith 82)
Smith D., Irby C., Kimball R., Verplank B., Harslem E., Designing the Star User Interface , ECICS, Stresa, Italie (septembre 1982)

(St-Dizier 83)
St-Dizier P., Analyse et conduite du dialogue en langue naturelle dans une interface homme-machine, actes des journées BIGRE, Cap D'Agde (octobre 1983)

(Stromfers 81)
Stromfers O., Jonesjo L., The Implementation and Experiences of a Structure-Oriented Text Editor, ACM Sigplan Notices, vol.16, 6 (juin 1981)

(Sutton 78)
Sutton J. Sprague R., Raster Graphics for Interactive Programming Environments, CSL-79-6, Xerox Palo Alto Research Center (1978)

(Symbolics 82)
Symbolics, Lisp Machine Summary, , (1982)

(Teitelbaum 81)
Teitelbaum T., Reps T., The Cornell Program Synthetizer: a Syntax-Directed Programming Environment, Communications ACM, vol.24, 10 (octobre 1981)

(Turoff 82)
Turoff Murray, Hiltz Starr R., Kerr Elaine B., Controversies in the Design of Computer-Mediated Communication Systems. A Delphi Study, Human Factors in Computer Systems, Gaitersburg (mars 1982)

(Wirth 76)
Wirth N., Algorithms + Data Structures = Programs, Prentice-Hall Inc, (1976)

DERNIERE PAGE D'UNE THESE

3È CYCLE, DOCTEUR INGÉNIEUR OU UNIVERSITÉ

Vu les dispositions de l'arrêté du 16 avril 1974,

Vu les rapports de M. J. Nassière.....

M. .....

. Herrmann Marc..... est autorisé

a présenter une thèse en vue de l'obtention du grade de DOCTEUR de 3e cycle.....

Informatique.....

Grenoble, le 17 JUIL. 1984

Le Président de l'Université Scientifique et Médicale

M. TANCHE