PLANEX : système d'aide à la planification

Alain Michard, Alain Giboin, Chantal Mais, Philippe Verdret

INRIA

UNITÉ DE RECHERCHE
INRIA-SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
BP 105
78153 Le Chesnay Cedex
France
Tél. (1) 39 63 55 11

Rapports Techniques

N° 95

PLANEX : SYSTEME D'AIDE A LA PLANIFICATION

Alain MICHARD
Alain GIBOIN
Chantal MAÏS
Philippe VERDRET

MAI 1988

RT_095

PLANEX : SYSTÈME D'AIDE À LA PLANIFICATION[1]

PLANEX: A GOAL-DRIVEN PLANNING HELP SYSTEM

Alain MICHARD
Alain GIBOIN
Chantal MAÏS
Philippe VERDRET

INRIA Sophia Antipolis
2004 route des Lucioles
06565 VALBONNE
FRANCE

[1]Cette recherche a pu être réalisée grâce au soutien financier de la société BULL-MTS (Direction des Etudes avancées) : convention de recherche BT/DT 85 051 589.

Résumé

Ce rapport présente PLANEX, *un système d'aide à la planification des activités de l'utilisateur de logiciels interactifs, et plus précisément son application aux activités bureautiques. Dans une première partie est exposée la raison pour laquelle l'aide à la planification a été choisie. La seconde partie est consacrée à la présentation proprement dite du système : méthodologie, architecture et fonctionnement. En conclusion sont abordés les avantages et les limites des systèmes d'aide à la planification, ainsi que les voies possibles qui conduisent à la construction de systèmes d'aide aux capacités plus étendues.*

Abstract

PLANEX *is a goal-driven planning help system offering to office application end-users informations about the best possible procedure to reach a given state with the available software tools. The rationale of the choice of a planning help system in office environments is first explained. The second part of the report describes the prototype system, its architecture and behaviour. Special emphasis is put on the user interface implementation. A few possible research directions are indicated to overdraw the present limitations of the prototype system.*

Table des matières

Liste des Figures

Introduction

Ce rapport présente PLANEX, un système d'aide à la planification des activités de l'utilisateur de logiciels interactifs. On décrit ici son application aux activités bureautiques.

Dans une première partie est exposée, à partir d'un modèle de l'activité de l'utilisateur, la raison pour laquelle l'aide à la planification a été choisie : importance de l'étape de planification dans l'activité de l'utilisateur, limitation des systèmes d'aide "classiques", qui assistent principalement l'utilisateur dans l'étape d'exécution.

La seconde partie est consacrée à la présentation proprement dite du système : architecture, implantation du prototype, bases de connaissance, générateur de plans, interface utilisateur, fonctionnement et évaluation.

En conclusion sont abordés les avantages et les limites des systèmes d'aide à la planification, ainsi que les voies possibles qui conduisent à la construction de systèmes d'aide aux capacités plus étendues.

Chapitre 1

L'aide à l'utilisateur

Quelle *aide* apporter à l'utilisateur d'applications interactives, quel *système d'aide* lui fournir ? Répondre à cette question suppose d'avoir répondu auparavant à ces deux autres questions : quel utilisateur ? quelle application ?

On constate dans les faits que les applications interactives deviennent de plus en plus complexes, en même temps qu'elles s'adressent à un public de plus en plus large. Plus précisément, on remarque que :

- les utilisateurs sont souvent des professionnels ayant une compétence importante dans leur spécialité mais faible en technique informatique;
- le temps qu'ils sont prêts à consacrer à l'apprentissage spécifique d'un nouvel outil de travail est très limité aussi bien pour des raisons institutionnelles que personnelles;
- l'utilisateur "naïf" tend à disparaître (surtout dans le domaine de la bureautique). Les utilisateurs ont en général déjà travaillé avec des logiciels plus ou moins voisins de celui qui leur est proposé et cherchent à transférer leur savoir-faire acquis dans les situations de travail passées, à la situation actuelle;
- enfin, les interfaces homme-machine (IHM) évoluent rapidement et ceci n'est pas sans influencer les modes d'apprentissage privilégiés par les utilisateurs : l'émergence des systèmes à "manipulation" directe faisant un emploi massif de représentations iconiques des objets manipulables, favorise l'apprentissage par la découverte, exploration libre des fonctionnalités du système guidée par des hypothèses liées aux expériences acquises avec d'autres systèmes plus ou moins similaires. Cet apprentissage par la découverte se faisant par définition sans un recours systématique à un manuel, suppose, pour qu'il soit efficace, une assistance en ligne particulièrement bien conçue.

Pour définir l'aide à apporter, le système à fournir, il est donc important de tenir compte de ces constats sur l'utilisateur et l'application qu'il utilise. Cela n'est cependant pas suffisant. Il est également esssentiel de se référer à un modèle général de l'activité des utilisateurs en situation de résolution de problème assistée par ordinateur.

1.1 Modèle de l'activité de l'utilisateur

Différents modèles existent. Pour illustrer notre propos, nous nous contenterons d'une adaptation schématique du modèle proposé par Norman (1984, 1986). Ce modèle rend compte de l'activité de l'utilisateur sous deux aspects : *a)* les représentations mentales qui sous-tendent cette activité; *b)* la structure de cette activité.

1.1.1 Représentations mentales de l'utilisateur

L'utilisateur dispose de connaissances sur son environnement de travail.

- **Modèle mental et modèle de conception.**— L'utilisateur a des connaissances sur les objets informatiques ainsi que sur les autres objets qu'il utilise dans son travail. Ces derniers ont un, voire plusieurs, rôle(s) fonctionnel(s) et des propriétés, qui sont modifiables ou non. Les objets ont entre eux des relations de dépendance sémantique variées, et il est possible de faire avec ou sur eux des opérations. Enfin les objets se regroupent en classes d'objets de même nature : les catégories. Par exemple, un utilisateur connaît, dans son organisation de travail, le rôle et les propriétés d'un rapport d'activité, d'un chrono courrier arrivée, d'une lettre de rappel, d'un état comptable, etc.

  Par ailleurs il connaît aussi, et découvre progressivement, les objets informatiques tels qu'ils lui apparaissent à travers les interfaces utilisateur des logiciels qu'il emploie[1]. Il peut être ainsi amené à découvrir les propriétés d'un document, d'une feuille de calcul, etc. D'une façon générale, on dit que l'utilisateur se construit un *modèle mental* du système informatique. Ce modèle constitue en particulier une représentation du comportement du système. Derrière le système, ou ce qu'en voit l'utilisateur par l'intermédiaire de l'interface, il y a le *modèle de conception* (voir figure 1.1).

  La construction du modèle mental implique en particulier de la part de l'utilisateur qu'il établisse des correspondances entre les objets informatiques et les autres : les propriétés des uns et des autres étant sensiblement différentes, la mise en correspondance nécessite de détecter ces différences et les limites de l'analogie entre les deux classes d'objets (*cf.* flèche 3 de la figure 1.1).

- **Mode opératoire et mode d'emploi.**— D'autre part, l'utilisateur connaît et met en œuvre des procédures de travail, qu'on peut regrouper sous le terme de *mode opératoire*. L'utilisateur connaît par exemple des procédures pour créer un rapport d'activité à l'aide d'outils autres qu'informatiques. Il peut transférer ces procédures de travail à l'outil informatique. Mais souvent, ce nouvel outil, ou ce qu'il est convenu d'appeler les "contraintes du dispositif", lui impose de modifier ses procédures habituelles, d'acquérir un nouveau mode opératoire (*cf.* flèche 6 de la

[1] Un "listing" sur papier fait également partie de cette interface utilisateur.

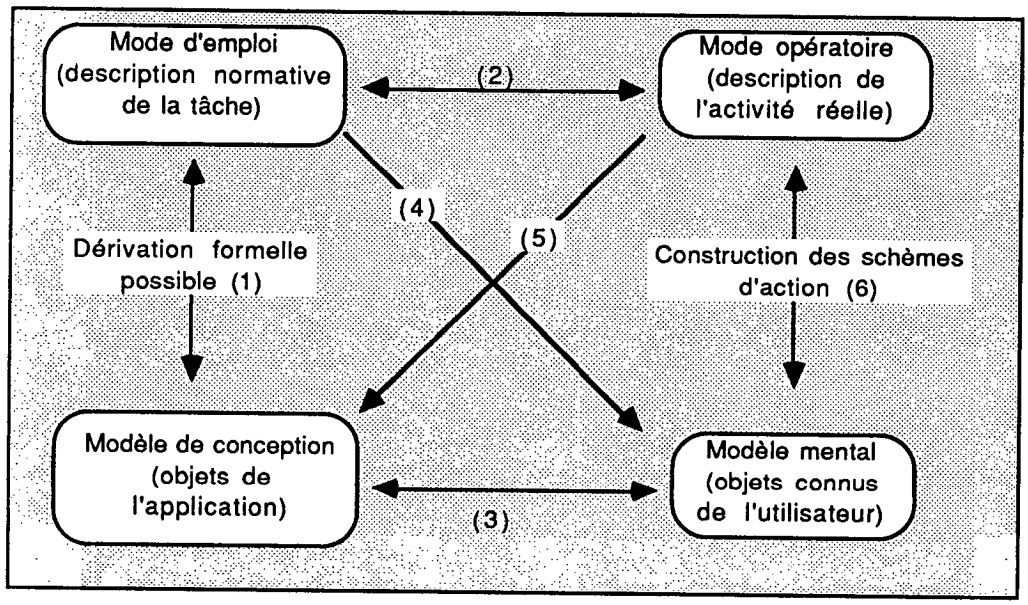

Figure 1.1: Relations entre modèle de conception et représentation mentale

figure 1.1). Cette acquisition implique en particulier de mettre en relation le *mode d'emploi* du système avec les procédures de travail (*cf.* flèche 2 de la figure 1.1). Le système propose une généralisation de la tâche parce que les objets du domaine informatique sont plus abstraits que ceux du domaine de la tâche dont le concepteur du système s'est inspiré. La tâche connue de l'utilisateur pourra ne constituer qu'une sous-tâche, l'ensemble des autres tâches proposées constituant une généralisation de la tâche connue.

1.1.2 Structure de l'activité de l'utilisateur

Ces représentations, l'utilisateur les met en oeuvre au cours de son activité. Cette dernière peut être décomposée en trois phases principales :

- **Formation de l'intention.**— L'utilisateur définit un objectif à atteindre, il se donne une tâche à réaliser. Se donner des objectifs adéquats nécessite principalement une bonne connaissance du domaine d'action, mais peut faire également intervenir la connaissance de l'outil informatique, certains sous-objectifs pouvant être impossibles à satisfaire ou inutiles dans l'environnement technique disponible.

- **Planification de l'action.**— L'utilisateur définit la façon dont il va atteindre cet objectif, il se donne une méthode. Il définit la suite des transformations qu'il va devoir mettre en œuvre et les séquences d'action correspondantes (les commandes). La planification dépend en particulier de l'outil disponible, de la représentation que l'utilisateur en a.

  Il y a possibilité de transférer un plan d'un système connu vers le nouveau système. Il arrive cependant que celui-ci se révèle incompatible avec le fonctionnement du système utilisé. Pour construire un plan compatible avec le système, il est nécessaire, en théorie, de disposer d'une bonne connaissance du fonctionnement du système (certaines actions ne sont pas nécessaires sur certains systèmes), de connaître les commandes correspondant aux actions à combiner et leurs effets.

- **Exécution du plan d'action.**— L'utilisateur réalise effectivement les manipulations (suite de commandes, contrôle de leurs effets) nécessaires. Lorsqu'une tâche devient familière à un utilisateur et que sa réalisation "s'automatise", seule subsiste cette phase de mise en œuvre. Il n'y a plus alors résolution de problème mais travail de routine.

Bien évidemment, cette décomposition est récursive, l'utilisateur oscillant entre une démarche de décomposition descendante du problème en sous-problèmes et une démarche ascendante permettant de remettre en cause des choix antérieurs en fonction des résultats partiels déjà obtenus. Un ensemble complet et satisfaisant de dispositifs d'aide et de documentation devrait prendre en considération ces différentes étapes de l'activité qui posent chacune des problèmes différents aux utilisateurs et des représentations que ces utilisateurs mettent en jeu lors de ces étapes.

1.2 Les différents types d'aide

L'aide que l'on peut apporter à l'utilisateur (ou le système d'aide qu'on peut lui fournir) peut être définie ou évaluée dans les termes du modèle de l'activité. Par exemple, on peut définir l'aide (ou le système d'aide) en fonction des étapes de l'activité sur lesquelles elle peut porter. Ainsi parlera-t-on d'aide (ou de système d'aide) à la formation de l'intention, d'aide (ou de système d'aide) à la planification et d'aide (ou de système d'aide) à l'exécution. En outre, à l'intérieur de chacun de ces types, on peut définir des sous-types correspondant aux représentations mentales mises en jeu; on parlera alors d'aide à la construction ou à la mise en œuvre du modèle mental ou du mode opératoire. Nous limiterons notre exposé aux aides à l'exécution et à la planification.

1.2.1 Aide à l'exécution

Les dispositifs d'aide en ligne traditionnels apportent principalement une aide à l'exécution. Ils offrent en général une description de la syntaxe d'une commande donnée en argument, ainsi qu'une description plus ou moins laconique de sa sémantique.

Divers modes d'obtention de l'information syntaxique aidant à la mise en œuvre de la commande sont proposés : commandes du type *man* ou *help* paramétrées par le nom d'une commande; complément automatique d'une commande incomplète; messages d'erreurs donnant la syntaxe autorisée lorsque des paramètres ou arguments ont été omis par l'utilisateur ou sont erronés, etc. Dans les systèmes à manipulation directe cette assistance peut prendre des formes différentes : proposition de menus n'offrant que les commandes ou paramètres valides dans un état donné; estompage des choix non valides; mode *aide* dans lequel une action de désignation (choix d'une commande, d'un objet-argument ou d'un paramètre) entraîne l'apparition de l'information correspondante.

Ces systèmes de documentation et d'aide en ligne ont évolué considérablement ces dernières années, principalement dans quatre directions :

1. *Hiérarchisation de l'information.*— Il s'agit de n'offrir par défaut qu'une information relativement laconique (syntaxe autorisée par exemple) et de donner à l'utilisateur la possibilité d'obtenir très simplement des détails et informations complémentaires à l'information de base proposée. Les systèmes HYPERTEXT (Conklin, 1987) facilitent considérablement la création et la mise à jour d'une base d'informations hiérarchiques et apportent à l'utilisateur final un "confort" dans la consultation du manuel en ligne tout à fait appréciable, en comparaison à la situation offerte par les systèmes plus anciens dont le paradigme reste le *man* d'UNIX.

2. *Mécanisme de renvois.*— Apparu avec les systèmes multi-fenêtres, il permet de traiter en zones sensibles à la désignation tous les mots-clés techniques figurant dans le corps d'un texte et d'associer à la désignation d'un de ces mots l'affichage du fichier documentaire correspondant dans une fenêtre en léger décalage (paquet de cartes) avec la précédente. Là encore les systèmes HYPERTEXT rendent aisément disponibles ces mécanismes de renvois.

3. *Formulation différenciée.*— Il s'agit de proposer des formulations différentes des messages d'erreurs et des informations proposées en réponse aux demandes d'aide, selon le "niveau" de l'utilisateur, ce niveau étant défini sur une échelle ordinale généralement à trois ou quatre valeurs. Le niveau est, dans les réalisations connues, indiqué par une variable propre à l'environnement de l'utilisateur et conservée dans un fichier personnel (*.profile*) consulté en début de session. Il n'existe malheureusement pas de méthode fiable et simple pour modifier dynamiquement la valeur de cette variable en fonction des activités, erreurs de l'utilisateur. Une telle interprétation nécessiterait un modèle de l'activité satisfaisant (Senach, 1987) qui reste du domaine de la recherche cognitive à long terme.

4. *Dépendance au contexte.*— Le contenu de l'aide apportée ne dépend pas seulement de la commande sélectionnée, mais aussi d'un certain nombre de caractéristiques pertinentes de l'environnement de l'utilisateur : ressources matérielles installées (les paramètres utiles d'une commande d'impression peuvent être différents selon le type de l'imprimante installée), mode activé, fichiers existants ou non, etc. Des exemples spectaculaires de dépendance au contexte bien conçue sont décrits dans (Danlos *et al.*; Guez, 1987) à propos du système INTERIX.

1.2.2 Aide à la planification

Second type d'aide, l'aide à la planification. La planification de l'action, ou pour employer le langage de l'ergonomie, la définition d'un mode opératoire compatible avec les contraintes du dispositif, constitue comme le montre plusieurs travaux en ergonomie du logiciel (Hoc, 1982; Senach,1987), la principale difficulté que doit surmonter un utilisateur confronté à un outil informatique qu'il ne domine pas parfaitement. Les plans élaborés par les utilisateurs sont souvent incomplets ou erronés (ils ne permettent pas d'atteindre l'objectif visé). Les dispositifs d'aide à l'exécution présentés ci-dessus sont incapables de fournir une aide à l'utilisateur dans ces cas. Pour pouvoir construire un plan susceptible d'atteindre un objectif, l'utilisateur doit combiner différentes commandes en tenant compte des effets de chacune d'elles.

Aide directe à la planification.— Pour cela, il faut connaître l'existence de la commande, connaître ses effets, mais aussi pouvoir juger de sa pertinence dans le contexte formé par la tâche en cours. Pour aider l'utilisateur dans cette phase d'élaboration de son plan de travail, un logiciel d'aide doit pouvoir répondre à des questions du type *"Comment faire pour arriver à* x *?"*, x étant n'importe quel objectif que l'utilisateur peut raisonnablement espérer atteindre à l'aide de son ou de ses logiciels d'application.

Plusieurs logiciels ont été proposés pour satisfaire ce besoin de l'utilisateur final. Le premier d'entre eux fut UNIX CONSULTANT (de Wilensky *et al.*, 1984) qui, comme son nom l'indique, fournissait des conseils aux utilisateurs du système d'exploitation UNIX. Ces systèmes répondent en général à des objectifs pouvant être atteints par une commande.

PLANEX, le système que l'on présente ici, appartient à la même catégorie générale, même s'il est fondé sur une méthodologie et sur une architecture logicielle différentes de celles proposées par son prédécesseur. Il vise l'atteinte d'objectifs plus complexes, qui nécessitent la combinaison d'actions différentes.

Dans ce domaine des aides à la planification, deux grandes options sont théoriquement possibles.

- La première, retenue aussi bien pour UC que pour PLANEX, consiste à n'apporter une aide qu'à la demande de l'utilisateur. Ce dernier "active" volontairement le système d'aide, pose sa question, puis retourne à sa tâche. On parle dans cas de système d'aide ***non intrusif***.

- Certains auteurs (Fischer, Lemke & Schwab, 1984; Zissos & Witten, 1985) ont proposé de construire des systèmes d'aide ***intrusifs***, le principe consistant à "surveiller" l'utilisateur pendant l'exécution de sa tâche, de tenter de détecter les erreurs, modes opératoires non optimaux, catachrèses, etc. et d'interrompre l'utilisateur pour lui proposer un ou des plans d'action plus satisfaisants, lorsque nécessaire. Ce type d'aide est donc fondé sur la reconnaissance automatique des objectifs de l'utilisateur à partir de l'historique de son interaction. Il est clair qu'un tel système ne serait acceptable que si les intrusions ne se faisaient toujours qu'à bon escient. Malheureusement la reconnaissance de plans d'action pour des tâches complexes se heurte à des difficultés théoriques (*cf.* Jackson & Lefrere, 1984), qui ne seront vraisemblablement pas surmontées à court terme.

Aide indirecte à la construction du modèle mental.— Au-delà de son rôle premier qui consiste à fournir un plan d'action, un système d'aide à la planification facilite la résolution du problème de la flèche 2 de la figure 1, et par effet secondaire la solution du problème de la flèche 3 de la figure 1. Il aide à la construction d'un modèle mental satisfaisant. Le plan à suivre pour exécuter une tâche est très lié au fonctionnement du système et le plan élaboré par l'utilisateur est très lié à sa représentation de la tâche sur le système. Lui fournir le plan à suivre, c'est en quelque sorte permettre indirectement à l'utilisateur de construire une représentation du système compatible avec le modèle de conception. Lui expliquer le plan et les contraintes que justifient ce plan permet plus directement à l'utilisateur de se construire une représentation compatible avec le modèle de conception.

L'acquisition d'un modèle mental opératif peut être facilité par le recours aux analogies. Le choix de la représentation doit être fondé sur une analogie forte et ouverte avec un domaine supposé bien connu de l'utilisateur. Nous ne décrirons pas ici plus avant cette technique d'interaction homme-machine, largement popularisée par la métaphore du "bureau" offerte en tant que représentation manipulable des objets d'un système d'exploitation. Cette méthode permet de rendre explicite la relation de type 3 (voir figure 1). Les représentations métaphoriques, pour utiles qu'elles puissent être, rencontrent très vite leurs limites dès que le système informatique propose des objets et des procédures de travail originaux par rapport aux objets et procédures habituelles. Le risque est alors pour le concepteur de s'enfermer dans une métaphore restrictive et naïve, assimilant un

traitement de textes à une machine à écrire, ou une messagerie à la distribution postale, empêchant du même coup l'opérateur de percevoir la richesse et l'originalité des objets informatiques.

Nous reviendrons en conclusion sur cette aide indirecte (et sur l'intérêt de la représentation explicite du bureau "réel"). Pour l'instant, nous allons décrire PLANEX et voir ainsi comment ce système aide à la planification des activités de l'utilisateur.

Chapitre 2

Le système PLANEX

2.1 Présentation générale

PLANEX est un système d'aide à la planification n'exploitant que des connaissances propres aux applications cibles. A chaque commande d'une application cible correspond un opérateur élémentaire permettant de modifier :

— soit l'état d'un ou de plusieurs objets manipulés par l'application ;

— soit le contexte (mode) définissant la sémantique courante d'une commande.

Etant donné un objectif de travail réalisable à l'aide de cet ensemble de fonctionnalités et un état courant du système cible (état des objets existants, contextes actifs), PLANEX peut générer la suite des transformations à appliquer pour parvenir à l'objectif. Les transformations élémentaires ("feuilles" du plan) correspondent donc à des commandes des logiciels cibles. PLANEX est un système d'aide à la planification individuelle : il n'y a pas de prise en considération des contraintes organisationnelles de réalisation d'une tâche, ni par conséquent des nécessités de coordination entre plusieurs agents pour la réalisation de tâches collectives.

De même, PLANEX n'a aucune connaissance sur la sémantique externe des objets que manipulent les applications : par exemple, l'aide portera sur la réalisation d'un document générique (objet du traitement de texte) et non sur la réalisation d'un rapport d'avancement de projet (objet de l'organisation).

PLANEX est donc un système s'inscrivant dans la lignée de UC ou d'INTERIX. Son intérêt spécifique est de permettre d'étudier sur un exemple "en vraie grandeur" les problèmes posés par la réalisation d'un prototype pré-industriel d'un tel système d'aide. En particulier nous nous attacherons à montrer comment ont été traités les problèmes de la modularité des connaissances et de la généricité de l'interface homme-machine, en vue de créer un système ouvert permettant l'intégration ultérieure de nouvelles applications cibles.

Le second objectif de cette réalisation est de disposer d'un prototype permettant une évaluation ergonomique d'un tel système d'aide. En effet très peu de données ont été

publiées sur l'aide effective apportée aux utilisateurs par de tels systèmes d'aide et les quelques informations disponibles dans ce domaine ne concernent pas le contexte bureautique, mais uniquement l'aide à l'utilisation d'un système d'exploitation.

2.2 Architecture

L'architecture générale du système PLANEX est illustrée par la figure 2.1.

Interface Entrées.— Le module d'interface utilisateur permet d'exprimer une requête sous forme d'un état final (un objectif) à satisfaire.Deux versions de cette interface ont été successivement étudiées. La première était basée sur un analyseur de langue naturelle exploitant une grammaire avec gestion limitée du contexte (voir Michard, 1985). Cette version d'interface utilisateur a été rapidement abandonnée pour des raisons qui seront explicitées à la page 20.

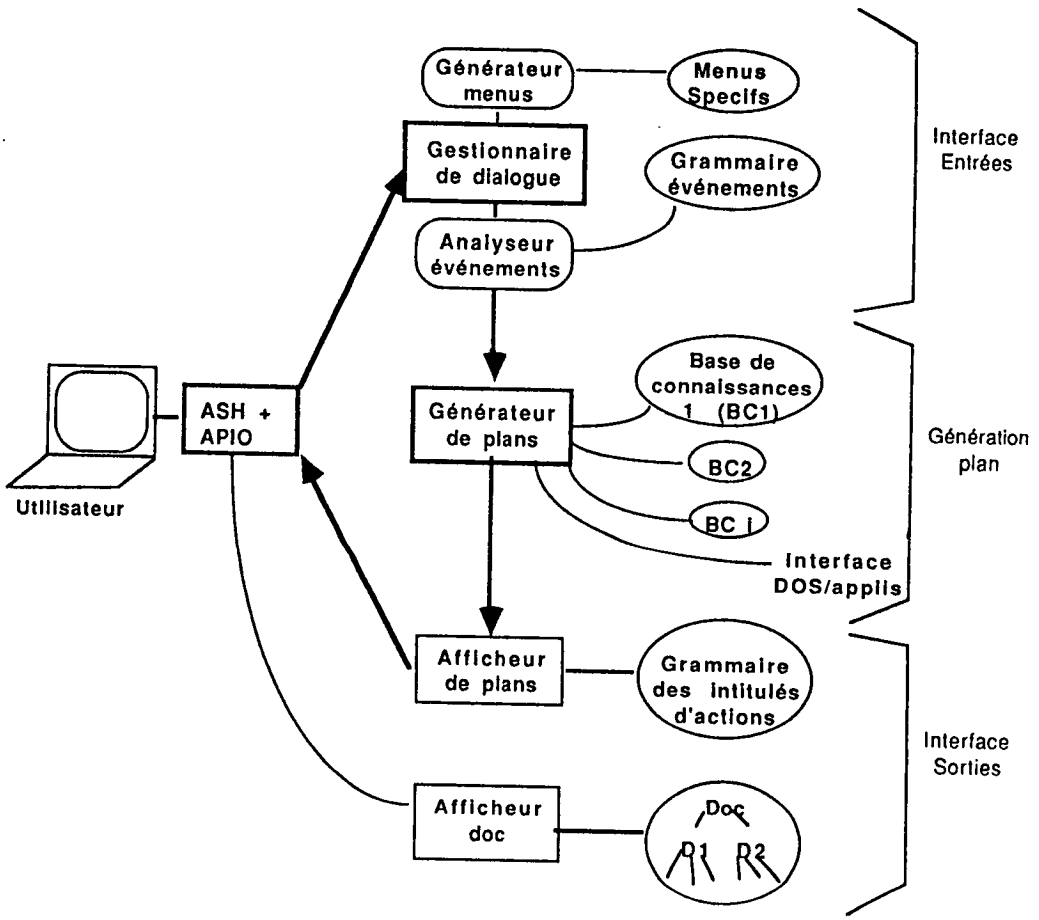

Figure 2.1: Architecture générale de PLANEX

La version actuelle est une interface à menus permettant à l'utilisateur de constituer une phrase-requête par sélection et assemblage de constituants prédéfinis. La requête ainsi constituée est interprétée à l'aide d'une grammaire pour fournir une représentation interne directement exploitable par le générateur de plans.

Génération plan.— Le générateur de plans est un moteur d'inférence fonctionnant en chaînage arrière, capable, étant donné un état final (E_f) à atteindre, et connaissant un état courant de départ (E_o), de générer la suite des opérations à appliquer pour passer de E_o à E_f. Comme pour tout générateur de plans, un opérateur n'est appliqué que si tous ses prérequis ont été au préalable satisfaits. L'algorithme utilisé est décrit à la section 18.

Pour générer un plan, le moteur d'inférence accède à des bases de connaissance (une par application cible et une commune à toutes les applications). Celles-ci décrivent toutes les opérations légales dans l'environnement informatique cible, ainsi que les prérequis de ces opérations. Le formalisme utilisé dans ces bases de connaissances est à base de règles et de propositions. Il est décrit à la page 15.

Interface Sorties.— Enfin un afficheur de plans complète l'interface utilisateur. Exploitant là encore une grammaire, il transforme le plan fourni par le générateur en une suite d'énoncés en langue naturelle. Ces énoncés sont ensuite affichés et l'utilisateur peut obtenir des explications détaillées sur une rubrique (une opération) donnée, par simple désignation. Ces explications détaillées sont enregistrées dans des fichiers dont l'ensemble forme une documentation en ligne éventuellement utilisable par un système d'aide plus traditionnel.

2.3 Langage de programmation du prototype

Pour privilégier la rapidité de développement et de modification, au détriment sans doute de l'efficacité et de la robustesse, le prototype est programmé en Prolog. Nous avons choisi l'interprète Prolog de l'Université d'Edimbourg, en raison de sa grande disponibilité, du fait que sa syntaxe s'est imposée comme une norme de fait et du fait que nous disposions de son code source, indispensable pour créer les extensions et interfaces mentionnées ci-dessous.

Le gestionnaire de station de travail (fenêtres et événements) est le *Brown Workstation Environment* (BWE), le seul dont nous disposions lors de ce développement (voir Pato, Reiss & Brown, 1984). Nous avons réalisé l'interfaçage entre l'interprète Prolog et ce gestionnaire de fenêtres. L'outil résultant, connu sous le nom de ASH_PROLOG, est un Prolog étendu d'une centaine de prédicats prédéfinis qui permettent toutes les manipulations nécessaires au niveau de l'interface homme-machine. Ces prédicats sont de relativement bas niveau: création et manipulation de fenêtres, des zones sensibles aux désignations et primitives graphiques simples (vecteurs, arcs, caractères) et opérations élémentaires sur rectangles de bits, dans les fenêtres. Un émulateur de terminal en mode VT100 est également fourni par accès aux primitives "VT" de BWE. ASH_PROLOG est décrit de façon détaillée dans (Michard & Monceyron, 1986).

A partir de cet outil, nous avons empiriquement défini une architecture logicielle pour PLANEX, décrite de façon détaillée dans les sections ci-dessous.

A posteriori il apparaît que cette organisation logicielle aurait été sans doute différente au niveau des modules de l'interface utilisateur, dans le sens d'une plus grande concision, élégance et efficacité, si nous avions disposé d'un gestionnaire de station de travail muni d'une boîte à outils de haut niveau telle que par exemple le *"Toolkit"* de *X_Windows* version 11 (Brunoff *et al.*, 1987). Une leçon (marginale pour notre propos) que l'on peut en effet tirer de cet effort, est que Prolog se prête assez mal à la programmation d'une telle boîte à outils, mais se prête de façon très élégante et pratique au prototypage d'applications exploitant les ressources de ladite boîte.

2.4 Les bases de connaissance

2.4.1 Nature de la connaissance

Le système dispose d'informations :

— sur la sémantique des fonctionnalités offertes par les différentes applications cibles. Par sémantique nous entendons une description des effets que chaque fonction est susceptible d'avoir soit directement soit par effet de bord, sur tous les objets existants dans la ou les applications;

— sur les propriétés des objets manipulés par ces applications: attributs modifiables ou non;

— sur les prérequis d'application des différentes fonctions: existence et états des objets et modes actifs.

— sur l'existence de modes opératoires "standard" permettant de réaliser un objectif donné partant d'un contexte donné. Ces plans pré-définis sont relativement peu nombreux: ils décrivent de façon laconique, donc éventuellement incomplète, des méthodes (plans d'actions) qui permettent de réaliser des tâches supposées fréquentes pour les futurs utilisateurs des logiciels cibles. Il correspondent *grosso modo* à des situations dans lesquelles l'utilisateur aimerait sans doute disposer d'une macro-commande. L'objectif poursuivi en représentant cette connaissance est purement un gain d'efficacité: il paraît peu raisonnable d'avoir à recalculer complètement des plans qui risquent d'être très souvent demandés par de nombreux utilisateurs en situation d'apprentissage. Il s'agit là d'une adaptation de l'idée des "plans squelettes" proposés par Friedland (1979).

Toutes ces connaissances, dans le prototype réalisé, décrivent deux applications bureautiques disponibles sur le poste de travail *Questar 400* de Bull, le système de messagerie *Starpost* et le système d'archivage *Starthèque*. Ces deux applications ont été choisies, en accord avec notre partenaire industriel, comme à la fois typiques de tout environnement de bureau moderne et en même temps suffisamment simples pour que la réalisation des bases de connaissance ne représente pas un effort démesuré par rapport à nos objectifs.

Ces bases de connaissance ont été obtenues par une analyse systématique des manuels de l'utilisateur et des manuels de référence de ces applications (voir Bull, 1985).

Par ailleurs, le système d'aide dispose également d'informations sur l'état courant du système cible: objets existants, modes actifs, etc. Dans le prototype actuel, développé sous UNIX, PLANEX accède simplement aux valeurs des variables d'environnement du processus *Shell* courant et peut lire la sortie standard de toute commande du système d'exploitation (fonctionnalités offertes par l'interprète Prolog utilisé). Le développement d'une version plus industrielle, mieux intégrée à son environnement bureautique, nécessiterait de prévoir ou d'utiliser des mécanismes plus efficaces de communication entre applications, PLANEX lui-même n'étant vu que comme une application particulière.

2.4.2 Formalisme utilisé

Selon la classification souvent utilisée en génération de plans, nos bases de connaissances forment un monde "STRIPS" (Fikes & Nilsson, 1971) : les connaissances sont représentées par des formules de la logique propositionnelle du premier ordre : propositions ou prédicats.

Prédicats relatifs à l'état initial.— Un premier groupe de propositions décrit l'état initial (E_o) du système :

- **always(X).** : X est toujours vrai, dans tout état du monde;
- **given(0, Y).** : Y est vrai dans E_o mais sa valeur de vérité pourra être modifiée;
- **Z.** : l'état représenté par la formule Z est vrai dans l'état courant du monde (état non daté). Remarquons au passage qu'une formule élémentaire Z peut contenir un nombre arbitraire de variables instanciées ou non;
- **imposs(Ek).** : l'état ou sous-état E_k du monde ne peut jamais exister. Un sous-état du monde est un ensemble de propositions (E & F ... & K) simultanément vraies.

Prédicats relatifs aux opérateurs de transition.— La description de chaque opérateur de transition consiste en sa **add-list** (la liste des formules qui doivent être ajoutées à l'état courant du monde lorsque l'opérateur est appliqué), de sa **del-list** (liste des formules à retirer de l'état du monde) et de ses préconditions.

- **add(A, Oj).** : lorsque l'opérateur O_j est appliqué, l'état A devient vrai et la formule correspondante est ajoutée à l'état du monde courant. L' opérateur O_j peut être paramétré par un nombre arbitraire de variables instanciées ou non;
- **del(B, Oj).** : l'état B devient faux et sa formule est retirée de l'état du monde;
- **can(Oj, C & D & ... & N).** : l'application de l'opérateur O_j n'est possible que si les formules $C...N$ sont présentes dans l'état du monde.

Prédicats ad hoc.— A ces prédicats de base qui suffisent à définir un "monde STRIPS", nous avons ajouté dans un souci d'efficacité les deux prédicats suivants:

- **forbidden(En, "nom_de_fichier").** : le sous-état E_n est grammaticalement bien formé, mais s'il est demandé en tant qu'objectif à satisfaire il n'y aura pas de tentative de génération de plan mais affichage du message contenu dans le fichier **"nom_de_fichier"**;

- **ppd(Ef, [Oq, Or,...Oz]).** : définit un plan incomplet à "expanser" pour satisfaire l'état E_f. Des opérations intermédiaires (O_s ... O_v) pourront si nécessaire être insérées dans le plan pour satisfaire les prérequis de ces opérateurs.

2.4.3 Grammaire de description des requêtes

Outre ce formalisme général de représentation des connaissances, la création des bases spécifiques représentant les applications cibles qui nous intéressent est guidée par une grammaire définissant l'ensemble des opérations et des objets existants. La prise en considération de nouvelles applications nécessiterait d'étendre cette grammaire pour décrire les objets et opérations qui lui sont spécifiques. Le lecteur trouvera en annexe les grammaires correspondant à *Starpost* et à *Starthèque*.

2.4.4 Modules

L'ensemble de la base de connaissance est réparti en *n+1* fichiers, où *n* est le nombre d'applications cibles documentées. Il existe un fichier de connaissances pour chaque application cible. Nous verrons lors de la description de l'interface que l'utilisateur est souvent en mesure d'indiquer dès les premières phases de spécification de sa requête quelle est l'application cible concernée par sa tâche. La base correspondante est alors chargée en mémoire centrale. Si la requête concerne plusieurs applications, les fichiers correspondants seront tous consultés (au sens de Prolog), mais il sera bien rare qu'une même requête concerne l'utilisation de plus de deux applications.

Le fichier supplémentaire définit le modèle conceptuel du système d'exploitation et plus particulièrement les opérations permettant les échanges d'objets entre applications ("couper-coller"). Tous ces fichiers sont contenus dans un même répertoire du système d'exploitation.

Par ailleurs, un autre répertoire contient tous les fichiers de messages d'erreurs et un troisième contient les fichiers de documentation des applications (manuel en ligne), utilisés pour les explications sur le plan généré (voir illustration à la page 27).

2.5 Le générateur de plans

Deux types d'objectifs peuvent être spécifiés au générateur de plan, le premier n'étant qu'un cas particulier simplifié du second.

Objectifs "simultanés".— Le premier type d'objectif se rencontre le plus fréquemment dans la pratique. Il s'exprime sous la forme:

- **traiter(A & B... & X).** : dans lequel *(A & B... & X)* est un ensemble de formules devant être simultanément présentes dans l'état final recherché.

Appelons Ω l'ensemble des formules *(A, B, .. X)*. L'algorithme de calcul de l'état final est le suivant:

1. *Rechercher si un quelconque sous-ensemble de* Ω *n'est pas incohérent (prédicat* `impossible`*), ou interdit (prédicat* `forbidden`*).*
2. *Résoudre* A*, c'est-à-dire tenter de créer un état dans lequel* A *est présent. Pour ce faire tenter d'appliquer l'une des méthodes suivantes :*
   - 2.1. *Vérifier si* A *n'est pas toujours vrai (*`always(A)`*).*
   - 2.2. *Vérifier si* A *n'est pas déjà vrai dans l'état courant.*
   - 2.3. *Vérifier s'il existe un plan prédéfini pour construire* A *et si oui "expanser" ce plan prédéfini, c'est-à-dire tenter d'appliquer les opérateurs de ce plan en vérifiant leurs prérequis.*
   - 2.4. *Construire* A *en cherchant un opérateur tel que* A *figure dans sa* `add-list`*, et tenter d'appliquer cet opérateur : résolution des prérequis, vérification que cette résolution ne supprime pas une formule élément de* Ω*.*
3. *Résoudre* B *en s'interdisant durant cette résolution de détruire* A*; en cas de blocage, tenter d'insérer la résolution de* B *avant celle de* A*.*
4. *Continuer de la même façon pour les autres éléments de* Ω*.*

Objectifs "successifs".— Le second type d'objectif se présente comme un enchaînement d'états (Ω_1, Ω_2,...) qui devront être successivement construits. Sa forme est la suivante :

- **traiter(Ω_i then Ω_j then...).** : avec $\Omega_k = A \ \& \ B \ \& ... Z$ (suite de formules représentant des états simultanément vrais), la règle étant que toute formule appartenant à Ω_i est conservée en Ω_j $(j > i)$, sauf si la résolution d'une formule de Ω_j nécessite sa destruction (principe d'économie).

Ce générateur de plans est une extension de l'algorithme de Warren (1974), modifié pour pouvoir traiter les plans prédéfinis incomplets et les plans à phases successives.

Le lecteur trouvera en annexe le programme Prolog commenté correspondant. Cet algorithme est moyennement efficace. Il ne permet pas de traiter le cas des plans conditionnels et itératifs, ni de la planification multi-agents. Enfin il n'y a pas de représentation et de traitement du temps: la sémantique d'un opérateur est stable quel que soit son moment d'application. Ces limitations sont acceptables pour créer un système d'aide tel que PLANEX, puisque ce logiciel n'exploite pas de représentations des événements et actions externes aux applications cibles. Les fonctionnalités d'un logiciel d'application sont indépendantes du temps réel et en règle générale ne prévoient pas (ou ne nécessitent pas) une utilisation collective coordonnée [1].

[1]Même les fonctionnalités d'un système de messagerie electronique peuvent être décrites sans faire référence au temps: les arrivées de messages, asynchrones par rapport aux actions de l'utilisateur, ne font qu'activer des états: on peut dire que la sémantique d'une commande du type *"read-mail"* dépend de l'état de l'indicateur booléen *"new-mail"*. Etant donnée une valeur pour cet indicateur, la sémantique de la commande est la même qu'elle soit utilisée une minute ou huit jours après la date du dernier changement d'état. On peut donc faire l'économie d'une véritable représentation du temps.

2.6 L'interface utilisateur

2.6.1 Présentation générale de l'interface

Nous décrivons dans cette section le modèle conceptuel externe de l'interface utilisateur. Les aspects "implémentation" sont traités à la page 30.

La première maquette de PLANEX était centrée, au niveau de l'interface d'expression des requêtes, sur l'utilisation de la langue naturelle. Grâce à l'obligeance de P. Saint Dizier, nous avions créé une version "adaptée" de son analyseur de langue naturelle ESOPE (Saint-Dizier, 1984). Ceci avait donné lieu à la construction d'une interface en langue naturelle restreinte (LNR) (Michard, 1985). Cette expérience nous a conduit à abandonner dans la version actuelle ce mode de dialogue au profit d'un système de menus décrit ci-dessous. Les raisons de cet abandon méritent d'être rappelées.

-a- Toute grammaire (aussi riche soit-elle) de la langue française écrite suppose et exprime un certain consensus sur les formes syntaxiques autorisées. Or les utilisateurs posant des questions spontanément au cours de leur travail utilisent volontiers des tournures elliptiques ambiguës, des abréviations, des anaphores, qui nécessitent pour être convenablement interprétées des grammaires d'un degré de complexité inacceptable. L'interface LNR impose donc une restriction arbitraire au vocabulaire et surtout à la syntaxe permise à l'utilisateur. Le problème est que ces restrictions ne sont pas "visibles" a priori : l'usager ne les découvre que par des refus occasionnels de ses requêtes, refus dont il perçoit mal les raisons profondes, n'étant pas grammairien.

-b- La frappe d'une question "en bon français" et si possible en évitant les fautes d'orthographe est longue et fastidieuse, même pour des questions sémantiquement simples : *"Comment imprimer en deux exemplaires tous les messages reçus postérieurement au 15/05/86 ?"* comporte douze mots et quatre-vingt-dix signes et espaces, alors que nous verrons que la même question peut s'exprimer en quatre désignations et huit caractères (ceux de la date).

-c- L'analyseur syntaxique et sémantique est un programme très lourd, très difficile à modifier, fort peu extensible en réalité : l'ajout de nouvelles règles de grammaire ayant souvent des effets de bord imprévisibles.

-d- Enfin l'efficacité des meilleurs analyseurs diponibles reste modeste : il faut compter plusieurs secondes (quatre à douze environ) pour analyser une requête telle que celle donnée en exemple ci-dessus, avec un bon interpréteur Prolog sur une machine de la classe SPS7. Ceci est toutefois susceptible d'évoluer rapidement avec l'apparition des compilateurs Prolog.

2.6.1.1 Expression des requêtes

Principe général.— Le principe de l'interface d'expression de requêtes est de conduire l'utilisateur à spécifier les champs variables d'une question dont la forme générale [2] est : *"Dans l'application X, comment faire l'opération* y *(paramétrée par les conditions* Ci,..,Cj*) sur l'objet (ou la classe d'objets)* Z *(paramétré par les attributs* Pk,.. Pn*)."*

Par exemple, pour exprimer la question *"Dans l'application messagerie, comment imprimer en trois exemplaires les messages reçus depuis le 30 Avril"*, l'utilisateur sera amené à indiquer :

— l'application : *Starpost*;

— l'objet : *messages*;

— les paramètres de l'objet, à savoir la date : > *30/04*, et le lieu de rangement : *chrono arrivée*;

— l'opération : *imprimer*; et

— les paramètres de l'opération : *3 exemplaires.*

L'utilisateur peut choisir à son gré d'indiquer d'abord l'objet ou l'opération.

Procédure.— L'utilisateur active le système d'aide PLANEX à partir de tout écran d'une application cible à l'aide de la touche fonction **SOS-PLAN** (voir figure 2.2).

L'écran de base de PLANEX apparaît alors (voir figure 2.2) avec d'une part une fenêtre de dialogue l'invitant à préciser l'application "principale" concernée par son problème et d'autre part un menu contenant la liste des applications documentées. PLANEX lui-même apparaît comme une application de cette liste, permettant d'obtenir une aide à son utilisation. Il est également possible de ne pas spécifier d'application principale (réponse ????).

Admettons que l'utilisateur ait sélectionné *Starthèque* (voir figure 2.3) c'est-à-dire l'application "archivage" et gestion de fichiers dans l'environnement bureautique considéré. PLANEX lui propose alors de sélectionner l'objet sur lequel il souhaite faire une opération (voir figure 2.4).

Le menu proposé contient la liste de toutes les classes d'objets existant dans l'application *Starthèque*. En désignant OPÉRATIONS, l'utilisateur pourrait obtenir la liste de tous

[2] Seules les requêtes concernant PLANEX lui-même ne suivent pas ce principe général.

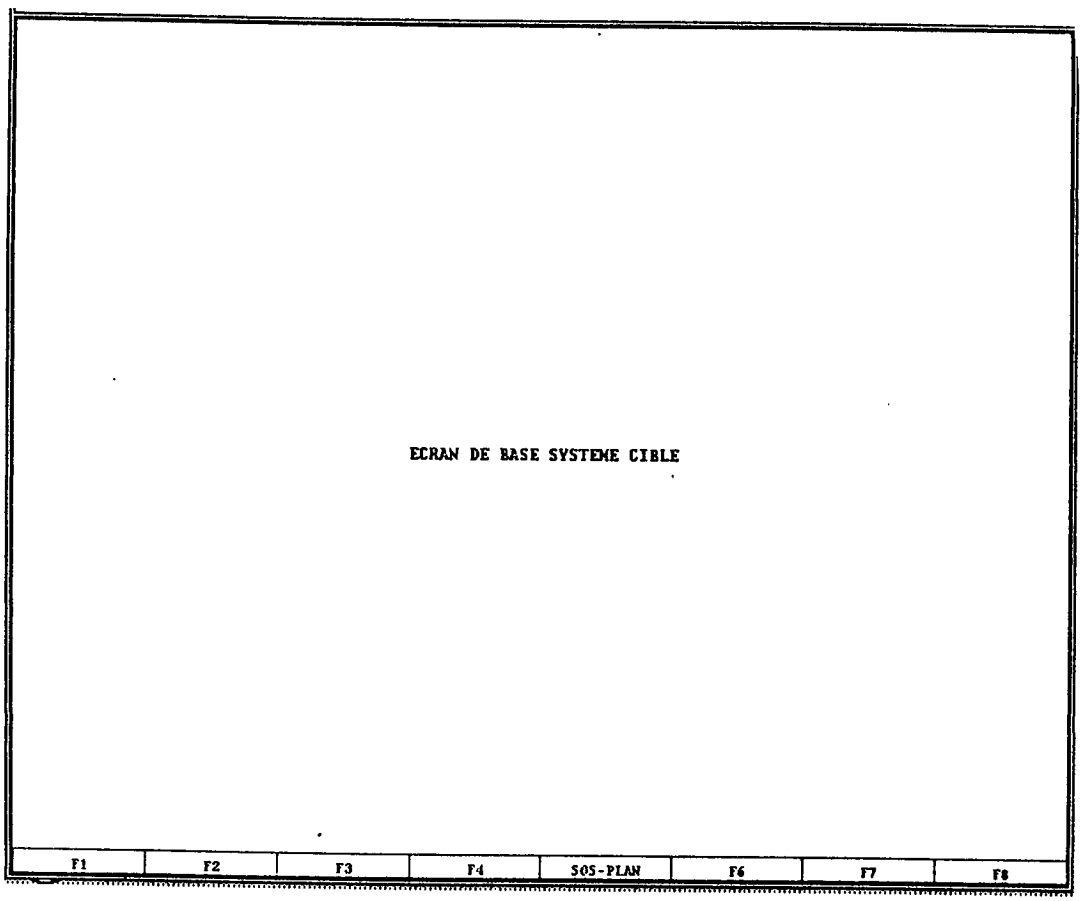

Figure 2.2: Ecran de base du système cible

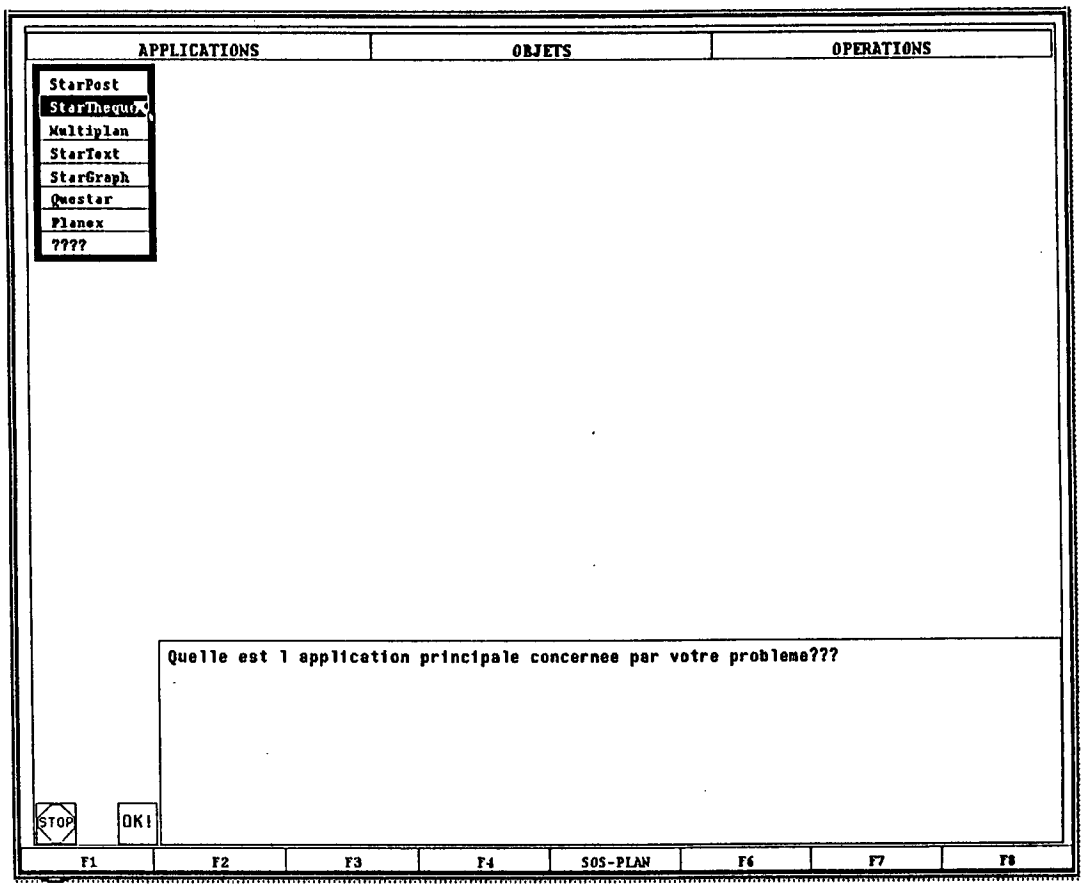

Figure 2.3: Ecran de base de PLANÉX : sélection de l'application *Starthèque*

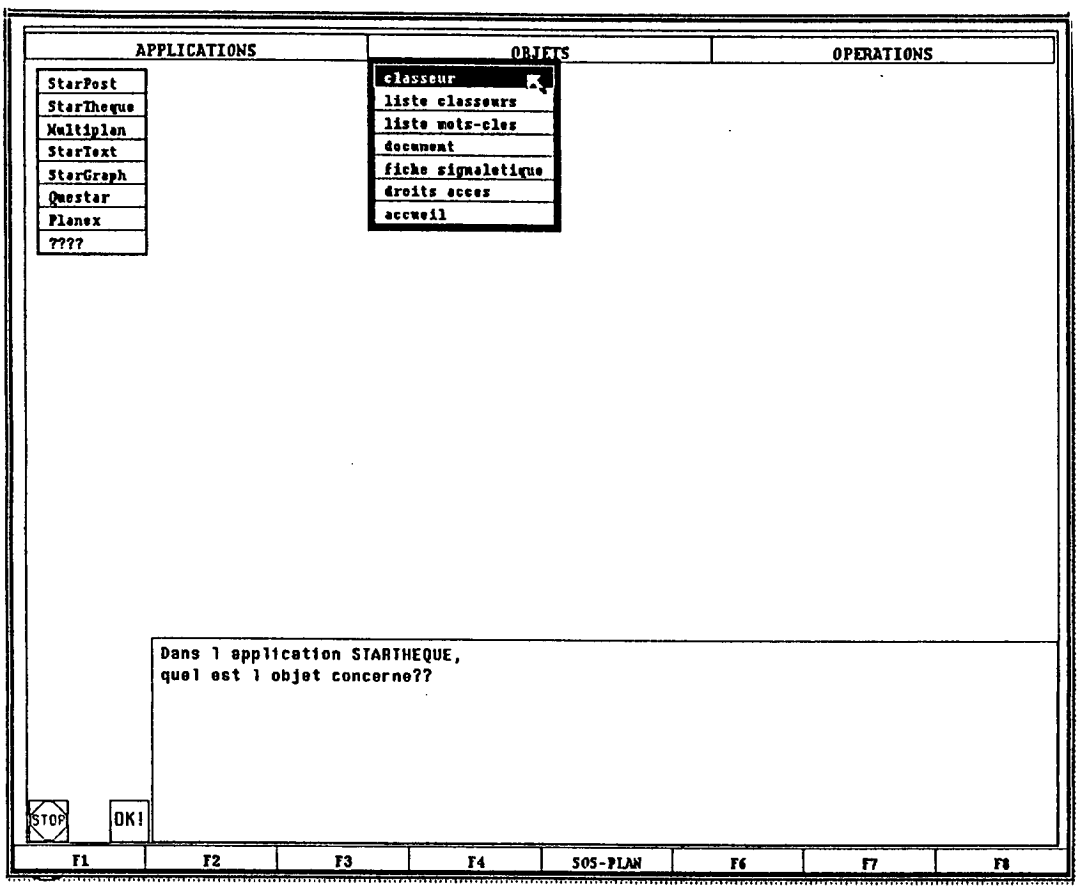

Figure 2.4: Sélection de l'objet *classeur*

les opérateurs disponibles et commencer par le choix de l'un d'entre eux. Admettons que notre utilisateur suive la démarche qui lui est suggérée et sélectionne l'objet [classeur]. Il obtient un menu (voir figure 2.5) contenant la liste des opérations valides sur un classeur, plus un item [DESCRIPTION] permettant d'obtenir une documentation sur cette classe d'objets (il n'y aurait pas alors de génération de plan).

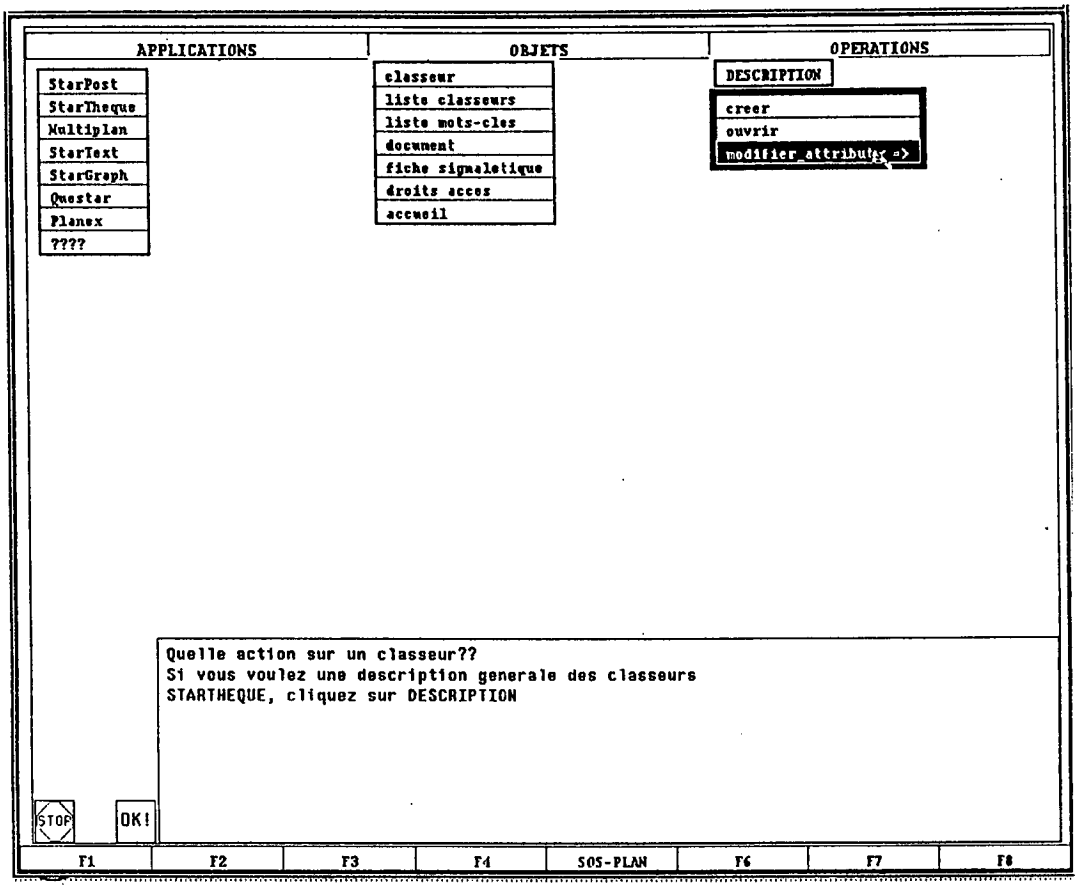

Figure 2.5: Choix de l'opération *modifier_ attributs*

L'opération [modifier_attributs] peut être précisée à l'aide d'un sous-menu donnant la liste des attributs modifiables (voir figure 2.6). Cette structure de menus est compatible avec le modèle conceptuel de l'application cible que l'on veut faire découvrir à l'utilisateur: la notion d'attributs d'un objet justifie la syntaxe du langage de commande de Starthèque, puisqu'une même commande permet d'accéder à la double fonctionnalité "changer le nom" ([nom]) ou "modifier les droits d'accès" ([droits_accès]) d'un classeur.

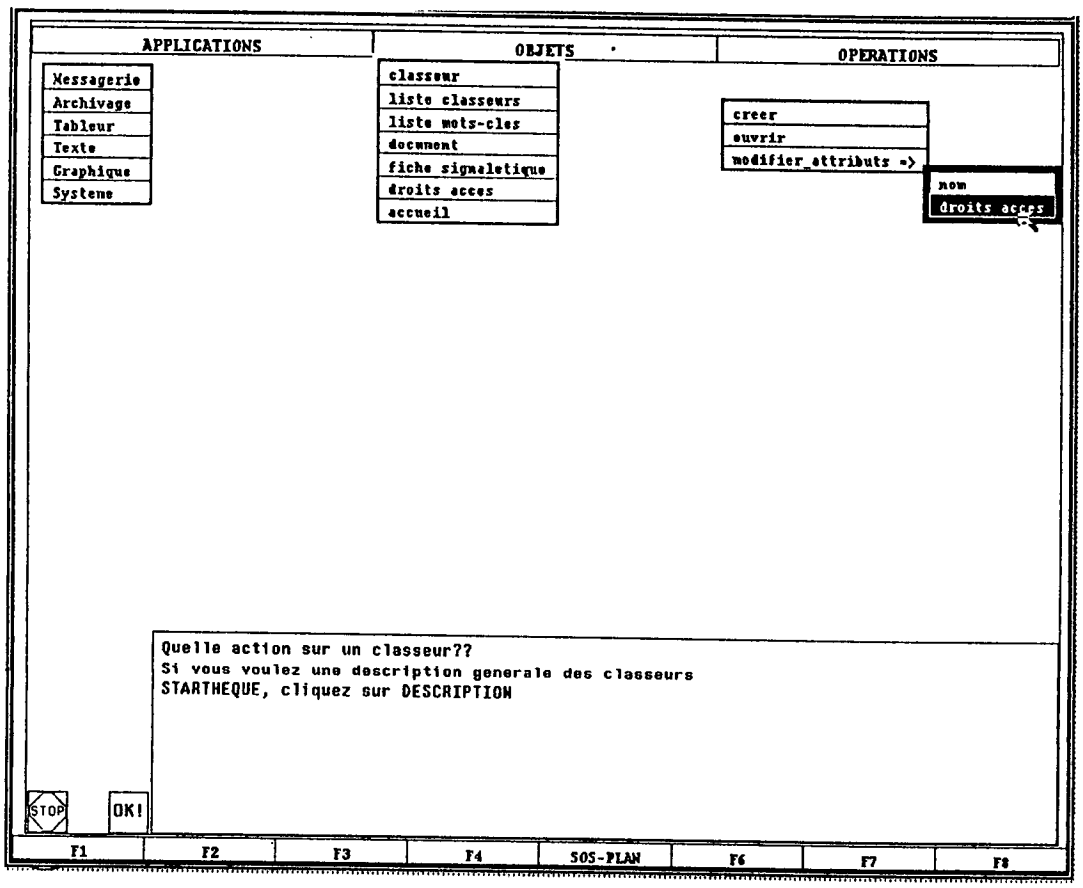

Figure 2.6: Sélection de l'attribut à modifier : *droits_accès*

La requête ainsi spécifiée peut ensuite être validée par désignation de l'icône OK !. A tout instant lors de la construction de cette requête, l'utilisateur peut revenir sur un choix antérieur et choisir un autre objet (ce qui modifierait le menu des opérations), voire même une autre application (ce qui modifierait aussi le menu des objets et annulerait donc une éventuelle sélection déjà faite dans cette rubrique).

2.6.1.2 Affichage du plan et des informations complémentaires

Le plan généré est visualisé sous forme d'une liste d'opérations décrites par un bref énoncé en langue naturelle (voir figure 2.7).

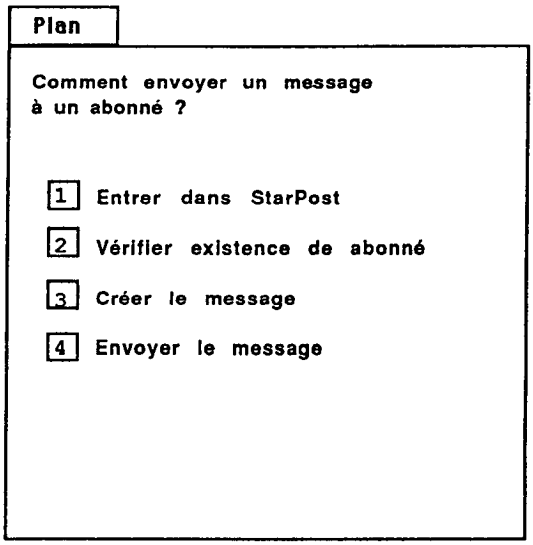

Figure 2.7: Plan fourni par PLANEX en réponse à une requête de l'utilisateur

Chacune des boîtes précédant les lignes de ce plan est sensible aux désignations et l'utilisateur peut, en "cliquant" sur l'une d'entre elles, obtenir des informations détaillées sur la méthode (syntaxe) permettant sa réalisation (voir figures 2.8 et 2.9). Cette explication est contenue dans un fichier, l'ensemble de ces fichiers constituant le manuel en ligne de l'environnement bureautique. Dans le prototype seuls quelques fichiers ont été créés afin de simuler cette fonctionnalité.

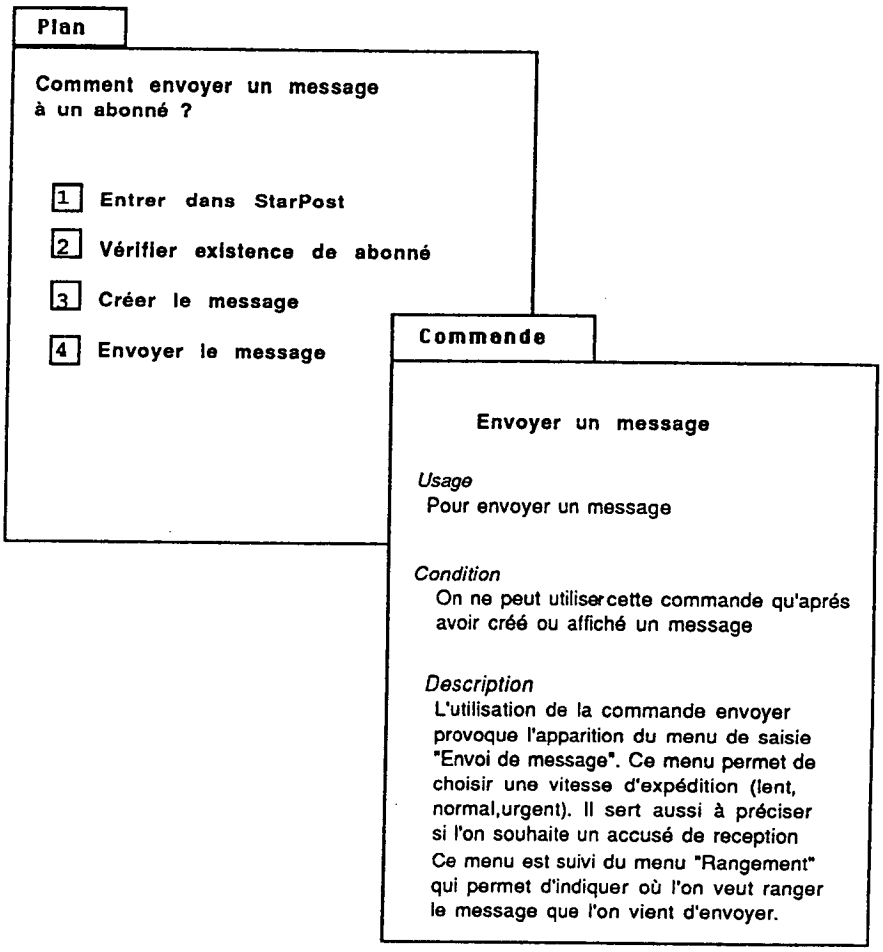

Figure 2.8: Description d'une sous-action du plan : ***envoyer un message***

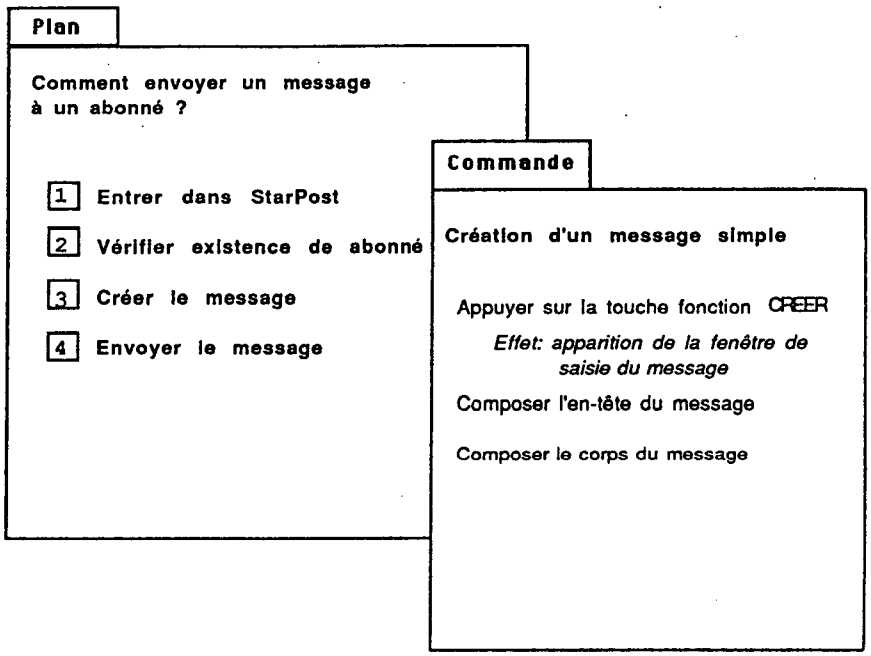

Figure 2.9: Description d'une autre sous-action du plan : *créer un message*

2.6.2 Implémentation de l'interface

L'interface utilisateur de PLANEX est composée des sept modules suivants, écrits en ASH_PROLOG:

— une boîte à outils minimale permettant de créer des fenêtres, des menus, des boîtes de dialogue, des boîtes d'alerte et de gérer l'affichage de ces objets dans des fenêtres ;

— une description déclarative des objets "statiques", c'est à dire indépendants des applications cibles;

— une description déclarative des relations existantes pour chaque application cible entre les objets (de ladite application) et les opérations qui sont possibles sur ces objets; ces relations sont utilisées pour générer les menus dynamiques dont la liste des items dépend de l'application cible sélectionnée ;

— un automate à états gérant les affichages en fonction des actions de l'utilisateur: chaque évènement généré par l'utilisateur provoque une transition d'état et l'effet correspondant au niveau de l'affichage;

— une grammaire permettant l'analyse du flux des évènements utilisateurs pour reconstituer la requête qui sera transmise au générateur de plan;

— une grammaire traduisant la représentation interne du plan généré en une suite d'intitulés en langue naturelle; dans la version actuelle, les énoncés synthétisés ne sont pas toujours très élégants et cette grammaire devrait être améliorée dans l'avenir;

— un "programme principal", en fait quelques clauses permettant l'initialisation du système, la lecture des fichiers et le contrôle des différents modules.

Les sections suivantes donnent quelques détails sur ces différents modules.

2.6.2.1 Boîte à outils

Notre boîte à outils "minimale" écrite en ASH_PROLOG permet de créer et de gérer l'affichage (coordonnées, visibilité) de fenêtres graphiques, de fenêtres d'édition, de menus verticaux ou horizontaux, de formulaires, de boîtes de messages, de boîtes de dialogue avec éditeur de ligne. Les menus peuvent être gérés en "statique", en "évanescent" (*pop-up*), en mono-choix (la sélection d'un item fait disparaître le menu), ou en multi-choix (possibilité de sélectionner plusieurs items dans un même menu). Le principe de notre boîte à outils est que tous ces objets du dialogue sont décrits (géométrie et mode d'affichage) dans un fichier (*cf. infra* "Objets statiques"). Un ensemble de fonctions de la boîte à outils permet d'initialiser ces objets (c'est-à-dire de créer les représentations internes correspondantes) et d'autres fonctions permettent d'en gérer l'affichage (gestion de la visibilité). Du fait qu'une interface homme-machine peut-être créée automatiquement à partir de spécifications déclaratives, cette boîte à outils est en fait un UIMS (*User Interface Management System*, voir Pfaff, 1983), mais un UIMS fermé, c'est-à-dire limité à certaines

catégories d'objets. Ainsi que nous l'avons déjà mentionné, cet UIMS aurait été plus concis, plus efficace et plus général s'il avait reposé sur une boîte à outils écrite en C telle que celles développées au dessus de *X_Windows*. Tel quel, notre UIMS constitue cependant un outil parfaitement utilisable pour prototyper très rapidement des interfaces homme-machine relativement "standard".

2.6.2.2 Objets statiques

La géométrie d'un objet du dialogue est définie à l'aide de quelques prédicats dont nous donnons ici quelques exemples.

Exemple : le menu vertical évanescent.— Considérons par exemple la définition d'un menu vertical évanescent:

```
window_mother(menu_exemple, base_window).
menu_struct(menu_exemple, vertic_1c).
menu_type(menu_exemple, popup).
menu_pos(menu_exemple, 10,10).
menu_items(menu_exemple, ['Item_1', 'Item_2', 'Item_3']).
menu_font(menu_exemple, 'cour.b.10').
```

Et voyons la signification de chacun des prédicats composant cette définition. Le premier prédicat (`window_mother`) permet d'indiquer quelle sera la fenêtre mère du menu `menu_exemple` (ici `base_window` est le nom symbolique prédéfini du plein écran).

Le second prédicat (`menu_struct`) définit la géométrie de base du menu, ici vertical à une colonne. Sont également prédéfinis les menus verticaux bi-colonnes et horizontaux.

Le troisième prédicat (`menu_type`) définit la méthode standard de gestion de l'affichage qui sera utilisée pour ce menu. Le choix offert est :

```
popup,
static,
one_choice,
multi_choice.
```

Le prédicat `menu_pos` permet d'indiquer la position du menu dans sa fenêtre mère lors de sa création (il pourra évidemment être déplacé).

Le cinquième prédicat (`menu_items`) permet d'indiquer la liste des items. Ces noms seront utilisés à la fois en tant qu'étiquettes dans le menu et en tant que noms symboliques des évènements générés lors de la sélection.

Enfin, le prédicat `menu_font` permet de choisir une police de caractères pour les étiquettes des items. Lorsque ce menu est ainsi défini, l'évaluation du prédicat `create_menu(menu_exemple)` crée les structures de données représentant cet objet, qui pourra être utilisé par l'application grâce en particulier au prédicat `show_menu(Menu,Event,Button)` qui provoquera son affichage à l'écran et après désignation (dans ce cas relâchement du bouton de la souris sur un item) instanciera `Event` au nom de l'item sélectionné et fera disparaître le menu de l'écran. `Button` donnera le numéro du bouton utilisé.

Création et modification d'objets.— Tous les objets du dialogue utilisés par PLANEX sont ainsi spécifiés dans un fichier unique. Il est donc très facile de se doter de nouveaux objets ou de modifier ceux existants. On remarquera en particulier que ces définitions n'utilisent pas un formalisme dédié et fermé mais le langage Prolog dans toute sa généralité. Si le concepteur de l'interface voulait par exemple faire dépendre la position du menu d'un certain nombre de paramètres externes (dimensions de la fenêtre mère par exemple) il serait donc parfaitement licite d'écrire quelque chose comme :

```
menu_pos(menu_exemple,X,Y) :- compute_position(menu_exemple,X,Y).
```

dans lequel `compute_position` serait une procédure Prolog aussi compliquée que nécessaire, accédant aux caractéristiques géométriques des autres objets et en particulier de la fenêtre mère de `menu_exemple`. Ces possibilités sont largement employées dans l'interface de PLANEX. Par exemple, en définissant une fois pour toutes :

```
font_for_menus('cour.b.10').
```

il devient possible de changer en une ligne la police de caractères utilisée par tout menu "foo" comportant dans sa définition :

```
menu_font(foo,F) :- font_for_menus(F).
```

Ce type d'écriture permet une modularisation très fine du code et supplée à l'absence de l'héritage des propriétés qui serait offert dans un environnement orienté objets.

2.6.2.3 Menus dynamiques

Nous avons dit, dans la description externe de l'interface, que PLANEX, étant donné le choix d'une application, pouvait offrir le menu de tous les objets (ou de toutes les opérations) existants dans cette application et que, étant donné le choix d'un objet (d'une opération), il pouvait proposer le choix parmi toutes les opérations (respectivement tous les objets) associées à cet objet.

On doit donc définir les associations suivantes:

```
application A → liste d'objets B;
application A → liste d'opérations P;
application A + objet Bi → liste d'opérations Pi; (pour tout i défini
dans B)
application A + opération Pj → liste d'objets Bj ; (pour tout j défini
dans P)
```

Ces associations dans la version actuelle sont simplement déclarées en Prolog :

```
association('Startheque ', 'classeur', [créer, ouvrir, 'modifier_attributs'])
```

L'évaluation de la clause `define_all_dynamic_menus` génère tous les menus correspondants. Dans une version ultérieure on pourrait envisager de générer ces "associations" par analyse syntaxique directe des bases de connaissance décrivant les fonctionnalités des applications. Ceci est théoriquement possible (toute l'information nécessaire est présente dans les bases de connaissances) mais non réalisé et probablement pas très simple.

2.6.2.4 Automate de gestion de l'interface

Les déclarations décrites ci-dessus décrivent ce qu'il est convenu d'appeler la géométrie des objets du dialogue. Nous décrirons ici la façon dont est spécifiée la dynamique du dialogue lui-même. Là encore, il s'agit d'une convention d'écriture en Prolog et non d'un formalisme spécifique. L'extension ou la modification de ces conventions d'écriture est donc toujours possible.

Le principe suivi ici est que chaque évènement généré par l'utilisateur (choix dans un menu, saisie d'une chaîne de caractères dans une boîte de dialogue, etc.) provoque une transition d'état de l'interface homme-machine et au moins dans certains cas, peut provoquer l'exécution d'une procédure interne à l'application (ici PLANEX). A tout évènement on va donc associer sa sémantique: actions au niveau de l'interface et actions au niveau sémantique interne de l'application. Dans la version actuelle, ces descriptions s'écrivent sous la forme:

```
exec(Event, Past_Event, Button) :-

    asserta(event_buffer, Event, Past_Event)[3],
    action_interne_A(...),
    action_affichage_1(...),
    action_affichage_2(..., New_Event, New_Button, ...),
    exec(New_Event, Event, New_Button).

exec(New_Event, Event, New_Button) :- etc.
```

Le paquet des clauses exec définit donc l'automate gérant les transitions entre états de l'interface homme-machine et gérant les appels aux procédures internes de l'application PLANEX. La mise au point de cet automate est donc entièrement manuelle et donc assez fastidieuse et source d'erreurs. La méthode n'est réaliste que grâce au fait que Prolog est un interprète et donc que le cycle édition-test-modification est très rapide.

On remarquera que la sémantique d'un évènement peut dépendre de l'événement précédent : pour prendre un exemple très simple, une boîte de confirmation ne génère toujours que l'un des deux évènements "OK" ou "ABORT". Il est clair que la sémantique de l'un de ces évènements dépend de l'historique de l'interaction. Toutefois, avec la méthode utilisée pour décrire le graphe des transitions, il suffit de ne connaître à chaque instant que l'évènement immédiatement précédent pour lever toute ambiguïté (constatation empirique et non démonstration formelle).

[3] Ces "assertions" n'ont aucun rôle dans le fonctionnement de l'automate gérant le dialogue. L'utilisation de cette mémorisation de la file d'évènements est décrite dans la section suivante.

2.6.2.5 Grammaire d'interprétation du flux de désignations

Lorsque l'utilisateur a spécifié une requête à PLANEX, il la valide et espère obtenir une réponse qui le satisfasse. A ce moment nous disposons d'une suite de désignations et éventuellement de saisies (dates, noms, etc.), suite à laquelle il convient de donner une sémantique qui sera représentée par une requête valide pour le générateur de plans. Pour autoriser toutes les modifications possibles à la requête lors de sa création (annulations de choix dans des menus à un moment quelconque du dialogue), cette interprétation de la suite d'évènements n'est faite qu'après validation explicite de la requête. Par ailleurs, des suites d'évènements différentes peuvent avoir une même sémantique, puisque l'utilisateur a une certaine liberté dans l'ordre de choix des éléments constitutifs de la requête. L'interprétation de cette dernière est donc basée sur l'utilisation d'une grammaire qui reconstruit la requête dans le formalisme utilisé par le générateur de plans. Par exemple les trois clauses suivantes extraites de cette grammaire définissent trois méthodes possibles pour formuler une requête "*Comment imprimer...*":

```
action(print(Objet,printing_specifs(L), Appli)) →

    application(Appli), /* sélection de l'application */
    [imprimer], /* sélection de l'opération */
    objet(Objet), /* sélection de l'objet imprimable */
    champs_impression(L). /* sélection des paramètres d'impression */

action(print(Objet,printing_specifs(L), Appli)) →

    application(Appli),
    [imprimer],
    champs_impression(L),
    objet(Objet).

action(print(Objet,printing_specifs(L), Appli)) →

    application(Appli),
    objet(Objet), /* sélection d'un objet quelconque de Appli */
    [imprimer], /* si l'on a pu choisir imprimer c'est qu'il était imprimable!*/
    champs_impression(L).
```

2.6.2.6 Afficheur de plan

La représentation interne du plan généré obéit à la même grammaire que celle décrivant l'ensemble des opérations légales dans les applications cibles. A titre d'exemple voici un plan tel qu'il est produit par le générateur:

Requête utilisateur:

```
Comment transférer un tableau de startheque au tableur?
```

Représentation interne de la requête:

```
transferred(document(U1, array, U2), appli(startheque), appli(multiplan)).
```

Plan généré:

```
enter(appli(startheque))
search(document(V1, array(V2)), doc_list(V3, V4), appli(startheque))
select(document(V1, array(V2)), doc_list(V3, V4), appli(startheque))
transfer(document(V1, array(V2)), appli(startheque), appli(multiplan))
```

Traduction "mot à mot":

```
Entrer dans l'application startheque
Rechercher le document sans critère de sélection, du type tableau non localisé, dans la liste des documents sans critère de tri sans lieu de rangement, dans l'application startheque.
```
etc.

Réponse que l'on souhaite afficher:

```
Entrer dans l'application startheque
Rechercher le tableau dans la liste des documents
Sélectionner le tableau
Transférer le tableau de l'application startheque vers l'application multiplan
```

Dans la version actuelle, la réponse affichée est malheureusement plus proche de la traduction "mot à mot" que de la réponse "souhaitée". La grammaire qui assure la traduction des étapes du plan en énoncés en langue naturelle est en effet encore très fruste. En fait il n'y a pas là de difficulté théorique insurmontable mais plutôt la confirmation que la synthèse d'énoncés en langue naturelle nécessite un solide modèle linguistique. Les travaux de L. Danlos autour du système INTERIX pourrait utilement servir de point de référence dans ce domaine. La version actuelle de notre grammaire relève plutôt du bricolage de linguiste amateur.

•Le plan comportant n étapes est affiché dans une fenêtre. Devant l'intitulé de chaque étape, l'utilisateur se voit proposer une petite zone rectangulaire numérotée 1 à n, sensible aux désignations. La sélection de l'une de ces boîtes provoque l'affichage dans une nouvelle fenêtre du fichier de documentation associé à la commande utilisée dans cette étape du plan. Cette consultation du manuel en ligne indexé par les commandes n'offre donc pas d'originalité particulière. Les relations entre actions du plan et fichiers du manuel en ligne sont spécifiées dans un fichier "d'associations" qui comporte essentiellement un paquet de clauses du type :

```
doc(Action(Objet(Parametre_1, Parametre_2), Application), fichier_X).
```

En instanciant plus ou moins les différentes variables de ce prédicat, on peut différencier à volonté la documentation en fonction non seulement de la nature de l'opération (variable `Action`), mais de l'objet sur lequel elle porte (`Objet`), ainsi que des valeurs des différents paramètres. En pratique, le nombre d'arguments d'une action varie de l'une à l'autre. Là encore c'est la grammaire des actions légales qui définit le format de ces clauses.

2.6.2.7 Contrôle général

Le lancement de l'application *Starthèque* se fait dans le démonstrateur actuel à l'aide d'un prédicat **run** qui initialise une fenêtre graphique générale `base_window`, génère les différents objets du dialogue (représentations internes des menus, etc.), affiche le menu de base offert à l'utilisateur et entre dans une boucle d'attente d'évènements qui seront récupérés et traités par les clauses **exec** décrites plus haut. Différents outils d'aide à la mise au point sont également prévus.

2.7 Performances

Temps de réponse.— Le démonstrateur existant, tournant sur un poste de travail de la classe Sun/3.50 ou Bull SPS7-300 sous UNIX, écrit entièrement avec l'interprète Prolog d'Edimbourg, génère une réponse avec un temps d'attente variant entre une et huit secondes selon la complexité du plan. La phase de spécification interactive de la requête n'impose aucun temps d'attente perceptible à l'utilisateur (après initialisation du système). L'utilisation d'un compilateur Prolog (SP_Prolog, Quintus, BIM, etc.) permettrait de gagner un facteur six à huit dans la rapidité de réponse, et permettrait donc des performances parfaitement acceptables pour une évaluation en vraie grandeur.

Taille mémoire.— La taille mémoire minimum nécessaire pour la version actuelle est de l'ordre de deux à trois méga-octets selon le nombre et le volume des bases de connaissances consultées. Un gain difficile à évaluer a priori serait possible au prix d'un effort important de "nettoyage" et d'optimisation du code.

Conclusions générales

Avantages et limitations des systèmes d'aide à la planification

Le développement d'un prototype comme PLANEX permet de faire empiriquement une démonstration de la faisabilité d'un système d'aide à la planification dans un environnement logiciel bureautique. Il serait maintenant utile de procéder à une évaluation ergonomique systématique d'un tel outil. Nous avons dans la première partie souligné l'intérêt que peut avoir une telle assistance pour les utilisateurs des logiciels bureautiques: aide directe à la planification de l'action (choix des procédures de travail) évidemment, mais aussi aide à la construction progressive d'une représentation mentale du modèle conceptuel des applications.

On peut également faire l'hypothèse qu'un tel système d'aide peut conduire à des utilisations "conservatrices" des logiciels cibles, réduisant l'initiative des utilisateurs: diverses observations ont montré l'importance des comportements "d'essais et erreurs" dans l'apprentissage par la découverte des logiciels. Il y a donc un compromis à trouver entre le "dirigisme" de l'aide qui conduit rapidement à des comportements efficaces et adaptés et l'encouragement à la manipulation libre et spontanée qui reste indispensable pour la constitution de la représentation mentale des outils informatiques. Seules des observations empiriques peuvent fonder des décisions de conception dans ce domaine.

Problèmes d'implémentation

Cette réalisation permet également de mieux appréhender les difficultés à résoudre pour implémenter un système d'aide "à bases de connaissance". Les principaux problèmes que pose la réalisation d'un tel système d'aide sont:

-a- le choix d'un langage et d'un environnement de développement adéquats;

-b- le choix d'un système support permettant une bonne intégration logicielle avec les applications cibles;

-c- la définition d'une méthodologie de programmation permettant l'extension du système d'aide vers de nouvelles applications cibles;

-d- la définition d'une méthodologie simple permettant la mise à jour régulière des connaissances utilisées par le système d'aide.

Ces choix sont bien évidemment étroitement dépendants les uns des autres.

Langage et environnement de programmation

Notre expérience rappelle, si cela était nécessaire, que la définition d'une logique "situationnelle" (Lifschitz, 1987), c'est-à-dire d'un formalisme logique permettant de décrire des états du monde et des règles de transformations de ces états, constitue une approche à la fois élégante et puissante pour représenter les fonctionnalités des logiciels interactifs et les plans d'action à l'intérieur du dialogue homme-machine. Il est clair que le langage Prolog qui permet une écriture très directe des formules de cette logique situationnelle est un choix "évident" pour coder les bases de connaissance du système: on évite ainsi d'avoir à utiliser une couche *ad hoc* de "représentation des connaissances", ainsi que cela aurait été nécessaire avec un autre langage de programmation.

Nous croyons qu'il était plus original de vérifier expérimentalement que Prolog constitue également un choix défendable pour implémenter les modules de gestion de l'interface utilisateur. Certes, il serait totalement absurde de vouloir réaliser un gestionnaire de fenêtres en Prolog. Nous avons également signalé que la réalisation de la boîte à outils de haut niveau permettant de créer et de gérer des objets de dialogue complexes (menus, formulaires, etc.) était possible en Prolog, (c'est ce qui est fait dans PLANEX), mais que cette approche était un pis-aller, sacrifiant en partie la robustesse et l'efficacité de l'interface. Mais il apparaît clairement que la spécification de l'interface, aussi bien au niveau de sa géométrie qu'au niveau de la dynamique du dialogue, peut parfaitement se faire sous forme de déclarations logiques et que cette approche (au prix d'une certaine discipline d'écriture) garantit une bonne modularité et extensibilité du module de l'interface utilisateur. Notre conclusion est à cet égard que la programmation logique doit être considérée comme un candidat sérieux en tant qu'interface de programmation pour les UIMS (voir Michard & Monceyron, 1986; Pfaff, 1983) orientés vers le prototypage rapide. Il est très vraisemblable que la diffusion massive d'une boîte à outils puissante et robuste qui va accompagner le gestionnaire X_Windows, encouragera la réalisation d'UIMS utilisant ce substrat. La logique se prêtant fort bien à la représentation symbolique de haut niveau des interfaces utilisateurs et Prolog permettant une interprétation immédiate d'une telle représentation, il est certain que l'approche qui était ouverte à l'occasion de la réalisation de PLANEX sera reprise sur des bases plus saines et plus efficaces, dans de nombreux sites [4].

Enfin, rappelons que l'algorithme de génération de plans dans un monde "STRIPS" représente trois pages de code Prolog (et donc quelques dizaines de kilo-octets en mé-

[4] Cette certitude est étayée par le fait qu'il existe au moins un projet ESPRIT (ALPES), dont l'un des objectifs est la création d'un UIMS "intelligent" en Prolog et qu'au moins trois laboratoires ou entreprises différents travaillent actuellement à de telles réalisations.

moire), à comparer au logiciel volumineux équivalent en Lisp et ce avec une efficacité qui serait plutôt meilleure, si l'on en croit les comparaisons faites voici plusieurs années par D. Warren, rapportées dans (Kluzniak & Szpakowisz, 1985).

Intégration logicielle

Le prototype existant est "déconnecté" des applications cibles dont il prétend expliquer l'utilisation. Si cet état de fait peut être transitoirement admis pour un démonstrateur de faisabilité, il est clair qu'une nouvelle implémentation devra impérativement accéder à l'état courant de la ou des applications actives: le plan généré doit en effet être totalement adapté à cet état. Par exemple, il ne serait pas acceptable de faire commencer un plan par *"Entrer dans Starpost"*, alors que l'utilisateur s'y trouve déjà! Heureusement il n'y a pas là de difficulté théorique ou pratique, dès lors que le développement du système d'aide prend en considération les mécanismes généraux de communication inter-applications disponibles dans l'environnement bureautique considéré.

Méthodologie d'extension et de mise à jour

La situation représentée par PLANEX à cet égard n'est que partiellement satisfaisante: la prise en considération d'une nouvelle application cible nécessite:

-a- de créer la grammaire décrivant les objets et les opérations légales dans cette application. Il s'agira en fait le plus souvent d'une extension des grammaires déjà disponibles, de nombreuses applications bureautiques ayant en commun bon nombre d'objets et même de fonctionnalités. Rappelons que cette grammaire ne figure pas explicitement dans les modules de PLANEX mais sa création sur papier est un préalable indispensable aux étapes suivantes;

-b- de créer la base de connaissance décrivant les objets et opérations propres à cette application en utilisant le formalisme décrit plus haut;

-c- de mettre à jour le fichier des clauses "associations" permettant le calcul des menus dynamiques de l'interface;

-d- de créer les fichiers de documentation en ligne;

-e- de mettre à jour le fichier des clauses "doc" permettant de retrouver les fichiers de documentation en ligne concernant chaque opération;

-f- d'enrichir si nécessaire (ce sera généralement le cas) la grammaire et le lexique permettant la traduction du flux d'évènements en une requête valide pour le générateur de plans;

-g- d'enrichir la grammaire et le lexique utilisés pour produire les énoncés en langue naturelle décrivant les étapes du plan (opérations paramétrées).

A titre d'information, l'un des auteurs du présent rapport a réalisé les étapes -a- et -b- en trois mois pour l'application *Starthèque*, sans connaissance préalable de PLANEX et du langage Prolog, mais en bénéficiant de l'aide de ses collègues qui avaient antérieurement créé la base de connaissances de *Starpost*.

L'étape -c- est très rapide et simple. L'étape -d- est relativement longue, mais elle est plus ou moins prévue lors du développement de n'importe quel logiciel d'application. Les étapes -f- et surtout -g- sont malheureusement difficiles, sources d'erreurs et nécessitent une bonne connaissance technique: connaissance de Prolog et de l'environnement PLANEX pour l'étape -f- et de plus connaissances linguistiques générales pour l'étape -g-. Il est clair que c'est à ce dernier niveau que des efforts importants restent à faire pour que de tels systèmes d'aide puissent être aisément créés et mis à jour en tenant compte des contraintes de production industrielle.

La solution doit être recherchée non pas vers une simplification du système mais au contraire vers une sophistication beaucoup plus grande : la qualité des énoncés explicatifs ou descriptifs du plan généré est en effet un critère fondamental dans l'évaluation subjective que les futurs utilisateurs feront de tels systèmes.

Voies de recherches et développements envisageables

PLANEX doit être considéré comme une étape dans la réalisation de systèmes d'assistance à l'utilisateur de plus en plus puissants. Nous recensons ici quelques unes des directions dans lesquelles des évolutions devront être faites.

Planification hiérarchique

PLANEX fournit aux utilisateurs la liste complète des actions à réaliser pour atteindre un objectif. Or les tâches ne sont pas composées d'actions élémentaires indépendantes: certaines actions sont elles-mêmes décomposables. Le plan généré par PLANEX est susceptible de contenir des actions situées à des niveaux hiérarchiques différents. Par exemple, pour transférer un texte de *Starthèque* dans *Startext*, il faut :

1. *Entrer dans Starthèque*
2. *Rechercher le document*
3. *Sélectionner le document*
4. *Copier le document dans le répertoire courant*
5. *Transférer le document dans Startext*

Les actions de recherche et de sélection (étapes 2 et 3) visent en fait à satisfaire des prérequis de l'action de copie en local (étape 4). Il peut être intéressant dans ce

cas de proposer un plan ne donnant que les opérations 1, 4 et 5, quitte à ce que si l'utilisateur demandait des détails sur l'opération "Copier en local", il obtienne l'expansion du sous-plan correspondant, comportant les opérations 2 et 3. Cette possibilité nécessite toutefois d'utiliser un algorithme de génération de plan beaucoup plus complexe que celui employé dans PLANEX, cette complexité ne se justifiant pleinement que si les extensions mentionnées ci-dessous étaient entreprises.

Mode tuteur

Il peut être tentant d'utiliser les mêmes connaissances sur les applications cibles pour proposer à l'utilisateur un mode "tuteur" dans lequel il se verrait proposer des tâches "exercices" à seule fin d'illustrer tel ou tel concept introduit par les explications concernant le logiciel cible. Nous pensons toutefois qu'il s'agit là d'un autre projet, la réalisation d'un tuteur posant des problèmes originaux non triviaux qui ne sont pas seulement techniques (stratégie pédagogique à utiliser en particulier).

Planification itérative et conditionnelle

Une extension de l'algorithme de génération de plan serait nécessaire pour traiter les plans itératifs (répéter telle opération jusqu'à ce que telle condition soit satisfaite) et conditionnelle (si vous êtes dans telle situation, faire A, sinon faire B). La planification itérative et conditionnelle dans toute sa généralité pose tous les problèmes théoriques de la génération automatique de programmes. Il est possible d'envisager de traiter la classe restreinte de plans itératifs pour lesquels la fonction d'évaluation définissant la convergence du processus itératif (et donc sa condition d'arrêt) est connue d'avance et peut être évaluée à chaque cycle de calcul du plan (plan partiellement pré-défini). Un travail exploratoire est nécessaire dans ce domaine pour trouver des algorithmes efficaces et évaluer l'intérêt de cette extension.

Représentation du bureau

Une évolution "de fond" des systèmes d'aide est non seulement envisageable mais à notre avis inéluctable: il s'agit d'utiliser non plus seulement des connaissances sur le fonctionnement des logiciels cibles, mais également des connaissances "externes" sur le bureau en général, les objets qui y sont manipulés, les procédures de travail collectives ou individuelles qui y sont en vigueur. Il est clair en effet que les plans d'actions "à la PLANEX" qui décrivent des phases de travail "face à l'écran", ne sont que des éléments artificiellement isolés dans les procédures de travail des opérateurs dans les organisations réelles. On souhaiterait par exemple pouvoir répondre à la question "*Comment créer mon rapport annuel d'activité?*", la réponse comprenant entre autres une description des phases interactives d'utilisation d'outils informatiques (éditeur de texte, tableur, formatteur de documents, etc.), mais intégrant aussi toutes les phases de travail (ou au moins certaines d'entre elles) non informatiques et toutes les contraintes organisationnelles à satisfaire pour la création de ce type précis de document : dans une organisation donnée, un rapport d'activité comporte certains types d'informations, est structuré selon un plan souvent

imposé, est mis en page selon une norme elle aussi souvent imposée. Ce sont toutes ces caractéristiques du rapport d'activité qu'il faudrait pouvoir prendre en considération pour fournir à l'opérateur (par exemple, nouveau venu dans l'organisation) une réponse circonstanciée satisfaisante.

Un des principaux problèmes à résoudre pour avancer dans cette direction est évidemment de fournir des mécanismes et des outils simples d'emploi permettant aux utilisateurs eux-mêmes d'enrichir, de préciser et de mettre à jour la représentation du bureau sous-jacente. Il s'agit là d'un domaine de recherche très lié à l'édition-validation des bases de connaissances des systèmes experts par les experts du domaine eux-mêmes.

Représentation du temps

Dans cette perspective de prise en considération des contraintes organisationnelles, il serait indispensable de disposer d'une représentation du temps, permettant de représenter par exemple la notion d'urgence ou de priorité des différentes opérations. La sous-tâche *"établir un bilan comptable"* n'a pas forcément la même procédure de réalisation selon la date à laquelle elle est entreprise. De plus beaucoup de tâches réelles sont collectives, et pour en décrire les procédures d'exécution, il faut être capable de représenter des contraintes du type : *"Lorsque l'agent A a fait telle opération et que l'agent B a fait telle autre, alors l'agent C doit réaliser telle sous-tâche dans un délai de huit jours"*. La représentation de ces contraintes temporelles nécessite une extension de la logique "situationnelle" classique. Plusieurs orientations sont décrites dans (Georgeff & Lansky, 1987).

Exploration du manuel en ligne en mode HYPERTEXT

Les logiciels du type HYPERTEXT fournissent un substrat général pour organiser une documentation technique volumineuse. Il serait intéressant dans un futur système d'assistance d'utiliser un tel outil plutôt qu'une classique hiérarchie de répertoires, principalement pour permettre à l'utilisateur une exploration (*"browsing"*) libre de cette documentation. Nous considérons toutefois ce point comme relativement secondaire, les expériences faites avec les systèmes HYPERTEXT existants montrant qu'il ne sont pas en eux-mêmes suffisants pour permettre à l'utilisateur une navigation efficace et confortable (Foss, 1988).

Bibliographie

1. BRUNHOFF, T., SWICK, R., SCHEIFLER, B., NEWMAN, R., GETTYS, J. (1987). *X Window System Release Notes. Version 11. Release 1*, MIT, Project Athena.

2. BULL (1985). *Bull Questar 400. Starpost : manuel de référence.* Version 1.00/2.00, document provisoire n° 82 F3 95DL Rev1.

3. BULL (1985). *Bull Questar 400. Starpost : manuel utilisateur.* Version 2.00, document provisoire n° 82 F3 56DL Rev2.

4. BULL (1985). *Bull Questar 400. Starthèque : manuel de référence.* Version 1.00/2.00, document provisoire n° 82 F3 96DL Rev1.

5. BULL (1985). *Bull Questar 400. Starthèque : manuel utilisateur.* Version 2.00, document provisoire n° 82 F3 57DL Rev2.

6. CONKLIN, J. (1987). Hypertext: An introduction and survey. *Computer*, September, 17-41.

7. DANLOS L., GUEZ S., SABBAGH S. (1985). Conception d'un système d'aide intelligent. *Intellectica* **1**(1).

8. FIKES, R.E., NILSSON, N.J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, **2**(3-4), 189-208.

9. FISCHER, G., LEMKE, A., SCHWAB, Th. (1984). Active help systems. In VAN DEN VEER, G.C., TAUBER, M.J., GREEN, T.R.G., GORNY, P., Eds, *Readings on Cognitive Ergonomics — Mind and Computers.* Berlin: Springer-Verlag, 116-131.

10. FOSS, C. (1988). Effective browsing in Hypertext Systems. In *Proceedings of RIAO 88: User-Oriented Content-Based Text and Image Handling*, Cambridge, MA: Massachusetts Institute of Technology.

11. FRIEDLAND, P.E. (1979). *Knowledge-Based Experiment Design in Molecular Genetics.* Doctoral Dissertation, Report n° 79-771, Stanford University, Department of Computer Science.

12. GEORGEFF (M.P.), LANSKY, A.L., Eds (1987). *Reasoning about Actions an Plans*, Los Altos, California: Morgan Kaufmann Publishers.

13. GUEZ, S. (1987). *INTERIX. Conception et réalisation d'un système d'aide intelligent sur UNIX.* Thèse de Docteur-ingénieur, Paris: École centrale de Paris.

14. JACKSON, P., LEFRERE, P., (1984) On the application of rule-based techniques to the design of advice-giving systems. *International Journal of Man-Machine Studies*, **20**, 63-86.

15. KLUZNIAK, F., SZPAKOWISZ, S. (1985). *Prolog for Programmers.* London: Academic Press.

16. LIFSCHITZ, V. (1987). On the semantics of STRIPS. In GEORGEFF (M.P.), LANSKY, A.L., Eds. *Reasoning about Actions an Plans*, Los Altos, California: Morgan Kaufmann Publishers, 1-9.

17. MICHARD A. (1985). Reconnaissance et génération de plans d'actions. Application à la réalisation de systèmes auto-explicatifs. *Cognitiva 85*, Paris : CESTA.

18. MICHARD A., MONCEYRON E. (1986). *Le système graphique ASH-PROLOG et son utilisation pour le prototypage rapide d'interfaces homme-machine.* Rapport technique n° 76, INRIA, Rocquencourt.

19. NORMAN, D.A. (1984). Stages and levels in human-machine interaction. *International Journal of Man-Machine Studies*, **21**, 365-375.

20. NORMAN, D.A. (1986). Cognitive engineering. In NORMAN, D.A., DRAPER, S.W., Eds. *User Centered System Design.* Hillsdale, New Jersey: Lawrence Erlbaum Associates, 31-61.

21. PATO, J.N., REISS, S.P., BROWN, M.H. (1984). *The Brown Workstation Environment.* Brown University, Department of Computer Science, Providence.

22. PFAFF, G.E., Ed. (1985). *User Interface Management Systems.* Berlin: Springer.

23. RICHARD, J.F. (1983). *Logique de fonctionnement et logique d'utilisation.* Rapport de recherche n° 202, INRIA, Rocquencourt.

24. SAINT-DIZIER, P. (1984). *Etude et réalisation de Esope : un système paramétrable pour la traduction de sous-ensembles du français en logique*, thèse de Docteur de troisième cycle, Université de Rennes I.

25. SENACH, B. (1987). Intelligence des logiciels d'aide à l'utilisation et modélisation de l'activité de l'utilisateur, *Cahiers d'Ergonomie des logiciels*, Ed. Bull-PMTET.

26. WARREN, D.H.D. (1974). *Warplan: A System for Generating Plans.* Department of Computational Logic Memo 76, University of Edinburgh.

27. WILENSKY, R., ARENS, Y., CHIN, D. (1984). Talking to UNIX in English: An overview of UC. *Communications of the ACM*, **27**(6), 574-593.

28. ZISSOS, A.Y., WITTEN, I.H. (1985). User modelling for a computer coach: a case study. *International Journal of Man-Machine Studies*, **23**, 729-750.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique