Institutional Repository - Research Portal
Dépôt Institutionnel - Portail de la Recherche
researchportal.unamur.be
University of Namur

THESIS / THÈSE

MASTER EN SCIENCES INFORMATIQUES

Architecture et spécification dans les interfaces homme-machine

Leonard, Dimitri

Award date:
1991

Awarding institution:
Universite de Namur

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 23. Jun. 2020

Facultés Universitaires Notre-Dame de la Paix - Namur
Institut d'informatique

ARCHITECTURE ET SPECIFICATION DANS LES INTERFACES HOMME-MACHINE

Dimitri LEONARD

Mémoire présenté en vue de l'obtention
du grade de Licencié et Maître en Informatique

par **Dimitri LEONARD**

Promoteur : Monsieur François Bodart

Année Académique 1990-1991

Remerciements

A Monsieur le Professeur François Bodart, promoteur de ce mémoire, j'adresse mes remerciements les plus chaleureux pour le stage qu'il m'a procuré et pour l'aide qu'il m'a apportée durant ce mémoire.

J'exprime également ma plus vive gratitude à Monsieur Benoît Sacré et à Madame Isabelle Provot de l'institut d'informatique pour les précieux entretiens que nous avons eus tout au long de l'élaboration de ce mémoire, ainsi que pour avoir mis à ma disposition leur documentation et leurs travaux personnels qui sont à la base de mon travail.

Je remercie le Professeur Joëlle Coutaz et le Professeur James Crowley de l'Université de Grenoble pour le stage très fructueux qu'ils m'ont permis d'effectuer sous leur encadrement.

Enfin, je voudrais remercier tous les membres de l'équipe de recherche du Professeur François Bodart qui m'ont aidés de près ou de loin lors de mes travaux de recherche.

Facultés Universitaires Notre-Dame de la Paix Namur
Institut d'informatique

Architecture et Spécification dans les Interfaces Homme-Machine

Dimitri LEONARD

Résumé :

Ce mémoire procède à la présentation et à l'étude de plusieurs modèles d'architecture et de plusieurs techniques de spécification qui peuvent être appliqués aux interfaces homme-machine. L'objectif est d'évaluer un modèle d'architecture et un langage de spécification du dialogue développés à l'institut d'informatique de Namur par Isabelle Provot et Benoît Sacré.

Mots-clé :

Interface Homme-Machine, Modèle d'Architecture, Techniques de Spécification, Dialogue.

Abstract :

This thesis surveys architecturals models and specification techniques that can be applied to human-computer interfaces. The general purpose is to evaluate an architectural model and a specification langage developed by Isabelle Provot et Benoît Sacré from the Namur Institute of Computer Science.

Keywords :

Human-Computer Interface, Architectural Model, Specification Techniques, Dialogue.

Mémoire de licence et maîtrise en informatique
Septembre 1991
Promoteur : Monsieur François Bodart

TABLE DES MATIERES

Section 2 Formalismes de spécification du dialogue

Introduction

1. Le sujet

Ce mémoire a trait à l'étude des modèles d'architecture logicielle et des langages de spécification pour des applications interactives appelées aussi systèmes interactifs. Il s'inscrit dans le domaine des Interfaces Homme-Machine.

Les travaux réalisés à l'Institut d'Informatique dans ce domaine, s'inscrivent dans le cadre de l'initiative lancée par le gouvernement. Ce dernier a décidé de lancer des programmes "technologies du futur" en 1991. L'UER Gestion de l'Institut a proposé un projet qui a été accepté et couvrant la définition d'un environnement informatique pour la construction d'interfaces Homme-Machine : architecture, langage de spécification, ergonomie, outils...
L'équipe de recherche du Professeur F. Bodart comprend les personnes suivantes, A-M Hennebert, J-M Lheureux, I. Provot, B. Sacré, L. Simon et J. Vanderdonckt.

Le domaine des interfaces homme-machine est un domaine de recherche en pleine évolution et en quête de fondements théoriques, notamment en matières de modèles d'architecture et formalismes de spécification.
Un modèle d'architecture pour des applications interactives, ainsi qu'un langage de spécification du dialogue d'une interface ont été développés à l'Institut d'Informatique par B. Sacre et I. Provot. L'objectif de ce mémoire est d'évaluer et de comparer ces derniers par rapport à certains modèles et formalismes de spécification existants, et ceci afin de permettre à B.Sacré et I.Provot d'ajuster les choix théoriques qu'ils ont fait avant de passer à la réalisation des outils d'un environnement pour la construction d'interfaces.

2. Organisation du mémoire

Nous consacrons le début de ce mémoire à la terminologie relative au domaine des Interfaces Homme-Machine (IHM). L'art des IHM est relativement nouveau et sa terminologie étant encore loin d'être stabilisée, nous présenterons quelques concepts et définitions du domaine des IHM dans l'intention d'aider le lecteur dans sa compréhension.
La suite du mémoire est organisée en deux sections principales. La première ayant trait aux modèles d'architecture logicielle pour des applications interactives et la seconde aux formalismes de spécification du dialogue.

Dans la première section, après avoir passé en revue la notion de modèle d'architecture d'une application interactive et son utilité, nous présenterons celui développé par I.Provot et B.Sacré [Provot et Sacré 90]. Ensuite nous effectuerons une évaluation comparative de ce modèle avec trois autres modèles. Le premier, et qui est aussi historiquement le plus ancien et à la base de nombreux travaux, est le modèle de Seeheim [Green 85].
Le second modèle envisagé, que j'ai eu l'occasion d'approfondir et d'implémenter lors de mon stage à l'Université de Grenoble, est celui développé par le Professeur J. Coutaz de l'Université de Grenoble. Il s'agit du modèle PAC [Coutaz 87,90]. Dans ce chapitre, on trouve également le compte rendu détaillé de mon stage.
Le troisième et dernier modèle de comparaison envisager est un modèle qui lui aussi a fait couler beaucoup d'encre, à savoir le modèle MVC de l'environnement Smalltalk-80 [Goldberg 84] [Cunningham 85].

Dans la seconde section, après avoir rappelé la notion de dialogue dans une application interactive et le rôle de sa spécification, nous présenterons le langage de règles pour la spécification du dialogue développé par I.Provot et B.Sacré [Provot et Sacré 90], puis on le comparera à trois autres catégories de formalisme de spécification.
Le premier type de formalisme de spécification du dialogue envisagé, et qui est aussi le plus populaire pour avoir été un des premiers a être utilisé et certainement le plus utilisé, est celui que constitue les diagrammes état-transition [Wasserman 85] [Green 86].
Le second type de formalisme envisagé est plus récent et regroupe une panoplie de langages, tous sous-jacent au même modèle, à savoir le modèle à événement. Ces langages sont appelés langages à événement et sont adaptés à la description des interfaces.
Nous verrons successivement le langage développé par Green [Green 85, 86], puis le langage événement-réponse de Hill [Hill 87].
Le troisième et dernier type de formalisme envisagé, a été développé par Jacob [Jacob 86]. Il présente une vue orientée objet de l'interface et constitue une combinaison intéressante du formalisme des diagrammes état-transition et du formalisme à événement.

Toutes ces présentations de modèles d'architecture et de formalismes de spécification font l'objet d'un long parcours de la littérature relative au domaine des IHM et du génie logiciel. Dans le domaine des IHM, on observe que la littérature est encore très imprécise et incomplète, ce qui n'a pas été sans poser certains problèmes durant la réalisation de ce mémoire.

Les évaluations réalisées à travers ce mémoire devraient permettre à I.Provot et B.Sacré d'ajuster les choix théoriques qu'ils ont fait en matière de modèle d'architecture et de formalisme de spécification du dialogue, avant de passer à la réalisation des outils d'un environnement pour la construction d'IHM.

3. Terminologie des Interfaces Homme-Machine

Le domaine des interfaces homme-machine a vu le jour il y a une dizaine d'année. C'est un domaine de recherche en pleine évolution et en quête de fondements théoriques, notamment en matières de modèles d'architecture et formalismes de spécification.

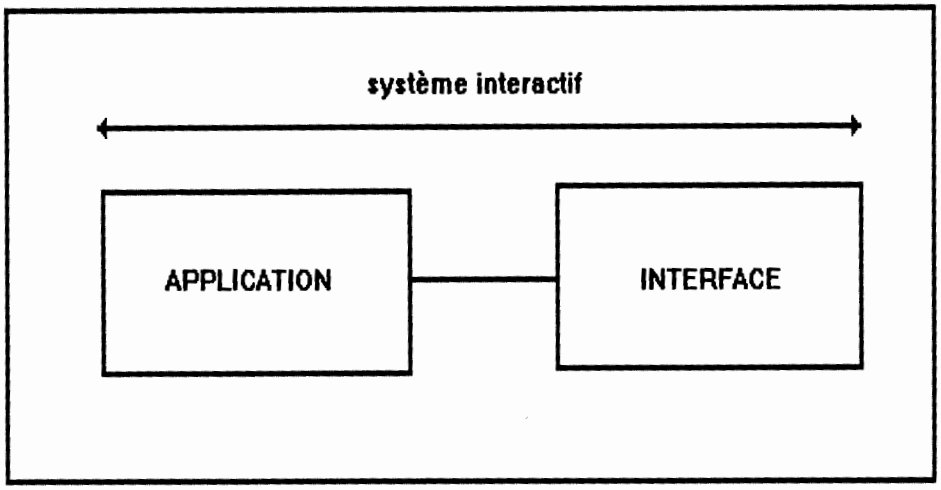
Dans ce domaine, la terminologie est encore loin d'être stabilisée. A travers la littérature abordée, nous avons pu nous rendre compte très rapidement du manque de définitions précises et cohérentes, ainsi que du flou ou de la confusion qui demeure dans la définition de nombreux concepts. Nous proposons quand même quelques définitions dans l'intention d'aider le lecteur à démarrer dans sa lecture et de l'aider dans sa compréhension des sections qui suivent.
D'autres définitions et concepts seront en plus développées ultérieurement, au fur et à mesure de leur utilisation dans les chapitres qui suivent.

3.1 Système interactif : Application et Interface

Dans [Coutaz 90], un système interactif est défini comme "l'assemblage de deux composants logiciels : l'Application et l'interface".

L'Application regroupe, pour un domaine donné, l'ensemble des composants logiciels qui réalisent les concepts et les fonctions du domaine. Ce composant implémente les fonctionnalités du programme et ne se préoccupe pas de la manière dont les données et les concepts du domaine vont être présentés à l'écran.

L'Interface regroupe l'ensemble des composants logiciels qui réalisent la présentation à l'écran (en entrée comme en sortie) des concepts et fonctions.



Dans cette étude, nous attribuerons la même signification au concept d'application interactive et à celui de système interactif. Une application interactive est un logiciel dont l'exécution implique une communication entre l'utilisateur et l'application c'est-à-dire une interaction entre l'utilisateur et le programme (par opposition au traitement BATCH).

On définira par application interactive, "tout système d'information du niveau application nécessitant une interaction entre l'application du système et les utilisateurs finaux" [Vanderdonckt 90a].
Pour une typologie des applications interactives se référer à [Petoud 90].

3.2 Le domaine des Interfaces Homme-Machine

Le domaine des interfaces homme-machine est en pleine évolution et fait intervenir plusieurs disciplines :les sciences cognitives rassemblant ergonomes et psychologues et les techniques informatiques.
Ce domaine se concentre sur le processus de développement d'IHM de qualité, incluant les activités suivantes : la spécification , la conception , l'implémentation, le prototypage, l'évaluation et la maintenance des IHM.

3.3 Système de gestion des IHM

Un tel système, appelé aussi UIMS pour "User Interface Management System", est composé d'un ensemble d'outils interactifs intégrés, destinés à supporter les activités de gestion des IHM énoncées ci-dessus [Hartson 89]. C'est précisément vers le développement d'un tel environnement que s'orientent les travaux de l'équipe de recherche du Professeur F.Bodart à l'institut d'informatique.

3.4 Boîte à outils (Toolkit)

Une boîte à outils est une bibliothèque de procédures adaptées à l'écriture d'interfaces homme-machine. L'éventail des fonctions offertes définit différents niveaux d'abstractions organisées en deux catégories : gestion du poste de travail et gestion du dialogue [Coutaz 90]. Les fonctions relatives au poste de travail gèrent le curseur, la police de caractère, les fenêtres, les événements en provenance des dispositifs d'entrée (clavier, souris) et à destination des dispositifs de sortie (son, fenêtrage). Les fonctions relatives à la gestion du dialogue s'appuient sur celles de la gestion du poste de travail et proposent toutes une série d'objets de présentation. Parmi ces objets, citons les boutons, les icônes, les champs de saisie, les menus, les barres de défilement, les formulaires...ect.

3.5 Le dialogue Homme-Machine

Le dialogue Homme-Machine dénote la communication entre l'utilisateur de l'application et cette application. Il détermine la suite des échanges d'informations (saisies et affichages) entre l'utilisateur et l'application via son interface, ainsi que la présentation des informations à l'écran.

I. Petoud dans [Petoud 90] préconise une modélisation du dialogue partagée en deux parties, la *présentation* et la *conversation*.

La présentation concerne la manière dont les informations sont affichées et présentées visuellement à l'utilisateur via son écran. Elle constitue en quelque sorte la partie statique de l'interface.

La conversation concerne l'enchaînement des écrans, des fenêtres, l'ordre des saisies et affichages des données c'est-à-dire la suite des échanges d'informations entre l'utilisateur et le système interactif. Elle constitue la partie dynamique de l'interface.

3.6 Style d'interaction

Un dialogue peut prendre plusieurs formes. On parlera de styles d'interaction.

Les techniques d'interaction forment un ensemble d'éléments variables en fonction d'une application interactive et regroupant à la fois des objets interactifs (icône, bouton, menu, boîte de dialogue) et des moyens d'interaction (clavier, souris, écran graphique).

Un style d'interaction constitue une combinaison logique de techniques d'interaction en vue d'organiser de manière unifiée et cohérente le dialogue, c'est-à-dire une suite de saisies et affichages. Pour une description détaillée des styles d'interaction voir [Shneiderman 87].

Nous énonçons ci-dessous quelques uns des principaux styles d'interaction utilisés jusqu'à ce jour :

1) Le langage de commande où l'utilisateur final du système dialogue avec l'application via un langage de commande formellement défini ; par exemple les systèmes d'exploitation Ms-Dos, Unix.

2) Le langage naturel où le langage de commande constitue un sous-ensemble significatif et bien défini de toute langue naturelle.

3) La sélection de menus où l'utilisateur effectue une (ou plusieurs) commande(s) en la choisissant dans un ensemble de commandes alternatives proposées à travers des menus. Par exemple, les menus du turbo Pascal V 5.5.

4) le remplissage de formes (formulaires) où l'utilisateur effectue une commande en remplissant des champs répartis sur une ou plusieurs formes affichées à l'écran.

5) La manipulation directe (voir ci-dessous).

6) le multi-fenêtrage où l'utilisateur voit son écran physique partagé en plusieurs fenêtres.

Notons que notre étude se restreint aux interfaces **graphiques** mettant en oeuvre comme techniques d'interaction, la souris et le clavier. Il n'est donc pas question de prendre en compte des interfaces utilisant comme techniques d'interaction avec l'utilisateur, la synthèse vocale ou des images de synthèse.

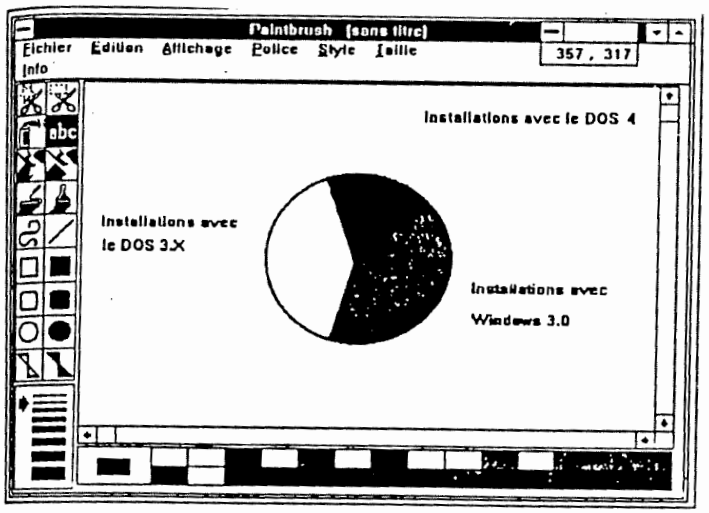
3.7 La manipulation directe

"Dans l'interaction homme-machine, on compte essentiellement deux classes de métaphores : celles qui s'inspirent du monde réel et celles qui parlent d'un monde abstrait.Dans la première catégorie l'interface s'efforce d'être la reproduction miniature du monde réel. Par exemple, un dossier électronique a la même présentation externe qu'un dossier réel : il comporte une chemise et un nom. Il peut être ouvert, reproduit, déplacé sur le bureau, rangé dans un autre dossier. L'utilisateur à l'aide de sa souris, a l'impression d'agir sur les objets. Il n'y a pas d'intermédiaire. L'utilisateur est engagé dans l'action de manière directe. On parlera de manipulation directe.

Dans la seconde classe, les interfaces montrent une image dans laquelle l'univers n'est plus représenté explicitement. L'interface devient un médium linguistique. L'utilisateur et le système sont engagés dans une conversation sur un monde supposé. L'interface devient ainsi un intermédiaire entre l'utilisateur et le monde dont il est question. Les langages de commandes ou les interfaces en langue naturelle entrent dans cette catégorie. L'utilisateur n'est plus en contact direct avec les objets mais il manipule des structures linguistiques" [Coutaz 90].

Une interface à manipulation directe présente donc à l'utilisateur un ensemble de représentations visuelles d'objets à l'écran et offre un répertoire d'actions permettant de manipuler ces objets. Ce type d'interface permet donc à l'utilisateur d'opérer directement sur les objets visibles à l'écran, et ceci à l'aide principalement de la souris et du clavier. Les résultats de ces manipulations sont immédiatement visibles à l'écran et facilement réversibles.Au lieu d'utiliser un langage de commande, parfois très complexe, pour décrire les opérations sur des objets, l'utilisateur manipule directement les objets visibles à l'écran.

Au départ la manipulation directe était essentiellement utilisée dans des applications graphiques de dessins et pour les jeux. Mais très vite ce type d'interface s'est répandu à travers d'autres types d'applications (tableurs, traitements de texte Wysiwyg...). Le Macintosh ou les environnements Ms-Windows, SunView sont des exemples mettant en application des interfaces à manipulation directe.



3.8 Quelques caractéristiques du dialogue

Nous énonçons ci-dessous quelques caractéristiques du dialogue des interfaces modernes.

3.8.1 Dialogue Multi-fils (multi-thread dialogue)

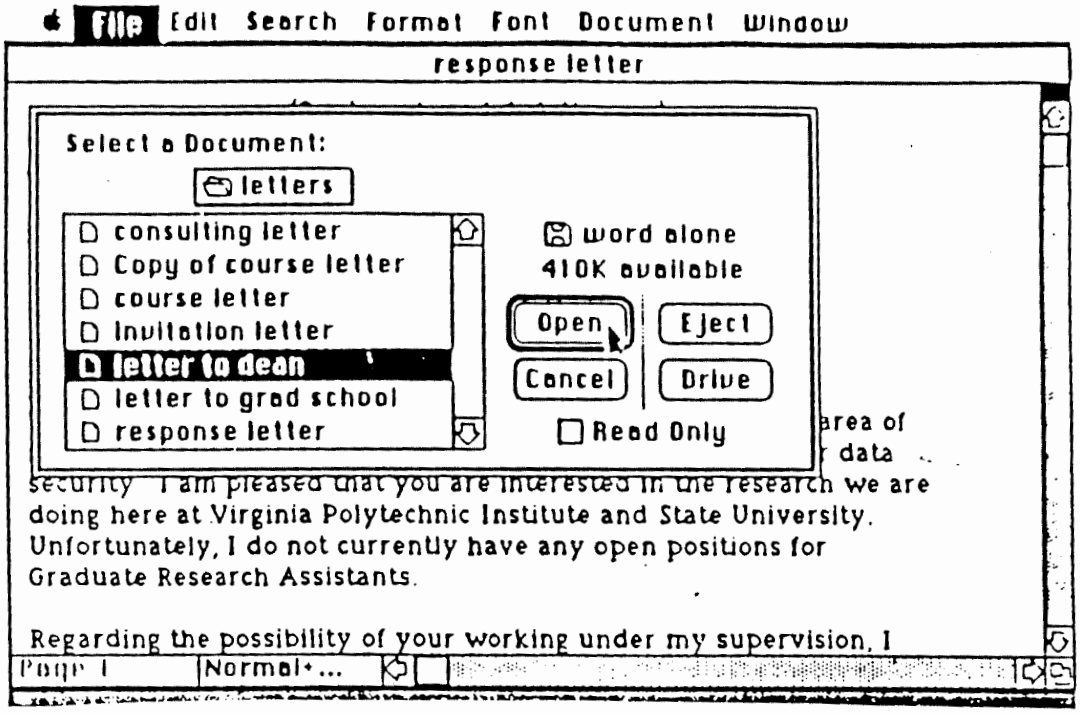
Ce concept très important et souvent lié à celui de manipulation directe, est la caractéristique majeure des interfaces modernes. Il existe de nombreuses définitions dans la littérature de cette notion appelée aussi dialogue à plusieurs fils d'activité. Nous en avons retenu deux, à savoir celles R. Hartson dans [Hartson 89] et de J. Coutaz dans [Coutaz 90].

Selon Rex Hartson un dialogue mutli-fils signifie que l'utilisateur, à tout moment du dialogue, a plusieurs chemins disponibles vers des tâches différentes, c'est-à-dire qu'il a le choix entre plusieurs fils de conversation distincts.

Selon Joëlle Coutaz, avec un dialogue à plusieurs fils d'activité, l'interaction est donc dirigée par l'utilisateur. Il peut entreprendre une tâche puis passer à une autre et revenir à la première plus tard. Il a le droit de changer de centre d'intérêt au gré de son plan d'actions, il peut agir simultanément sur la présentation de plusieurs concepts à l'écran, et plusieurs présentations de concept peuvent s'exprimer simultanément.

Un tel dialogue doit permettre à l'utilisateur d'entreprendre la spécification d'une première expression, puis de passer à la spécification d'une autre, ensuite de revenir à la première dont le début a été spécifié. Par exemple, l'utilisateur peut déplacer son curseur dans un champ de saisie, frapper quelques caractères, puis déplacer son curseur quelque part ailleurs, faire d'autres opérations et enfin retourner au champ de saisie.
On observe l'existence de plusieurs minidialogues qui correspondent chacun à un fil d'activité mentale de l'utilisateur.

L'interface du Macintosh fait un usage incessant de cette notion de dialogue mutli-fils. La figure ci-dessous nous montre la boîte de dialogue associée à la commande "Save As". Cette boîte de dialogue donne à l'utilisateur la possibilité de choisir parmi plusieurs tâches : accepter le nom de fichier par défaut, scroller et sélectionner un nom de fichier existant, entrer un nouveau nom de fichier, annuler l'opération.



3.8.2 Dialogue séquentiel et asynchrone

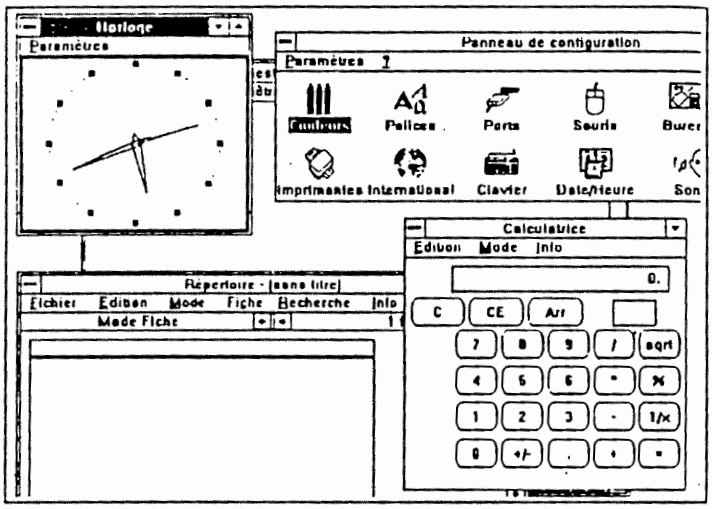
Dans un dialogue asynchrone, plusieurs tâches sont disponibles au même moment du dialogue. A tout instant, au cours de la réalisation d'une tâche, l'utilisateur final du système peut passer à une autre tâche, et revenir par après à la première plus tard. L'utilisateur peut passer d'un point du dialogue à un autre et interrompre une tâche pour en commencer une autre.

Le dialogue est asynchrone en ce sens que le séquencement de chaque sous-dialogue est indépendant l'un de l'autre. L'enchaînement des dialogues est guidé par l'utilisateur qui peut déclencher un nouveau sous-dialogue de manière asynchrone. C'est le type de dialogue que l'on retrouve très souvent dans une interface à manipulation directe qui met en oeuvre un dialogue multi-fils.

Dans un dialogue séquentiel, il y a un ordre prédéfini dans les transactions possibles pour l'utilisateur et dans l'enchaînement des sous-dialogues possibles. L'utilisateur passe de manière prévisible d'un point du dialogue à un autre. Il y a donc un séquencement prévu dans le déroulement de l'interaction. Ce séquencement est contrôlé par le système qui présente une tâche à la fois à l'utilisateur ou tout au plus un nombre limité et prédéfini de tâches.
Une interface utilisant un langage de commande fait très souvent usage d'un dialogue séquentiel.

3.8.3 Dialogue concurrent

Il s'agit d'un dialogue multi-fils où plus d'un fil d'activité du dialogue peuvent être exécutés simultanément. Pendant qu'une tâche s'exécute, une autre peut être commencée. Il y a concurrence tant du point de vue de l'utilisateur que de celui du système.
L'exemple classique nous est donné par l'interface dans laquelle, une horloge présente à l'écran, est mise-à-jour en même que l'utilisateur exécute un traitement de texte.



3.8.4 Le parallélisme dans le dialogue

Il s'agit d'une notion très récente et encore peu exploitée. La notion de parallélisme correspond ici au fait que l'utilisateur peut faire un usage simultané de plusieurs dispositifs de commande (moyens d'interaction : souris, clavier, crayon optique...). Buxton et Myers dans [Buxton 86] ont démontrés que permettre aux utilisateurs d'accomplir deux tâches simultanément augmente leurs performances.

Hill dans [Hill 87] décrit et spécifie une application de dessin faisant usage de cette notion de parallélisme du dialogue.

Section1 Architecture logicielle d'une application interactive

1.La mise au point itérative d'une interface

Dans l'état actuel des connaissances, la seule manière efficace et effective de réaliser l'interface d'une application interactive est celle d'une mise au point itérative.

Cycle de réalisation d'une interface :

1) Conception et première réalisation,
2) L'utilisateur final teste l'interface et émet ses remarques,
3) Modification de l'interface sur base des remarques émises en 2),
4) Retour au point 2).

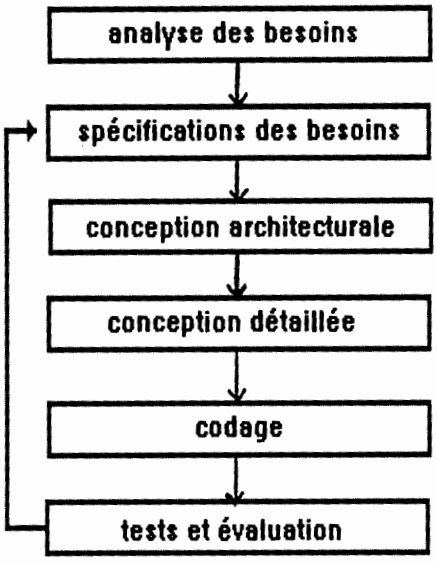
" The only reliable way to generate quality interface is to test prototypes with users and modifiy the design based on their comments. [Myers 89] "

La nécessité d'une architecture adaptée se fait donc sentir si on ne veut pas atteindre des coûts prohibitifs de développement et de mise à jour. Un modèle d'architecture doit donc, entre autre, définir les éléments directeurs d'aide à l'organisation modulaire de l'application interactive, ceci afin de faciliter cette mise au point itérative de l'interface.

Les architectures sont exprimées à l'aide de modèles. Ces modèles ne sont que des descriptions informelles, car il n'y a aucune base mathématique sous-jacente à ces modèles.

2. Modèle d'architecture

Dans le cycle de développement d'une application interactive, la conception architecturale appelée aussi conception globale se situe après les étapes de l'analyse des besoins et de la spécification des besoins.



De manière générale, concevoir une architecture logicielle, c'est construire une structure pour l'application interactive à développer [Van Lamsweerde 90]. Toute structure est faite de composants et de relations logicielles entre ces composants.
Nous envisageons la comparaison d'architectures logiques et pas physiques. C'est à dire que les composants seront des unités d'oeuvre pour l'analyste-programmeur (unité de spécification, de documentation, de conception, de validation ...).

L'utilité d'un modèle d'architecture pour la conception et la réalisation d'une application interactive est vaste. Il n'est pas question ici de traiter de ce problème en détail, mais signalons cependant les quelques points suivants qui paraissent importants.

L'organisation logicielle d'un système interactif doit (ou devrait) s'organiser selon les lignes directrices d'un modèle d'architecture. Ce dernier décrit la manière dont l'interface interagit avec l'application [Hartson 89].

"Un modèle d'architecture est censé fournir au concepteur une structure générique à partir de laquelle il lui sera possible de construire un système interactif particulier" [Coutaz 90]. Il doit offrir une vue sur la façon d'organiser les éléments constitutifs d'un système interactif.

La réalisation d'un environnement pour l'IHM (UIMS) par l'équipe de recherche du Professeur F. Bodart, a comme objectif de fournir des outils de réalisation et de génération automatique de l'IHM à partir des modèles conceptuels. Il faut dès lors auparavant, avoir définit un modèle d'architecture, de manière à savoir ce que l'on va générer et comment les éléments générés seront organisés entre-eux.

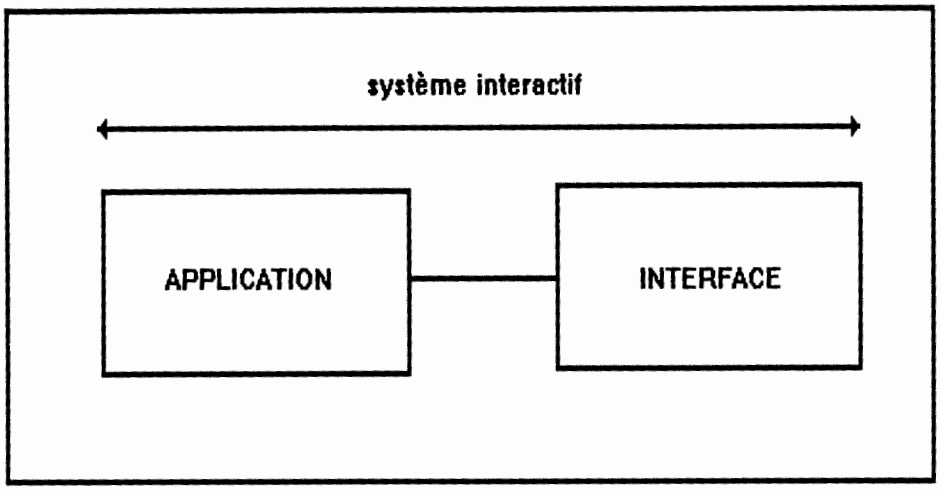
De plus, si en génie logiciel, la complexité trouve souvent sont remède dans l'usage de la modularité, cette modularité reste difficile à maîtriser en l'absence de modèle. Jusqu'à ces dernières années, on ne disposait pas de modèle de référence pour la construction des systèmes interactifs. Sans cadre directeur pour appliquer la modularité, il devient difficile de concevoir des logiciels modulaires et adaptés à l'évolution itérative des interfaces.

3. Indépendance du dialogue

Autrefois, traditionnellement c'était l'application qui contrôlait les saisies et affichages en vue de collecter les données introduites par l'utilisateur de l'application. Le code relatif au dialogue était alors dispersé à travers celui de l'application. Cette dispersion du code relatif au dialogue (présentation et conversation) n'était pas sans effets négatifs sur la facilité de modification et de maintenance du dialogue.

Ces difficultés rendaient difficile la réalisation d'interfaces de qualités, notamment à cause du fait que ce mélange des codes rendait difficile le cycle itératif de développement d'une interface.

Une vue différente du développement d'une application interactive, est celle où la sémantique de l'application est séparée de son dialogue. Tous les travaux à l'heure actuelle s'appuient sur ce fondement : celui de la séparation entre l'application et l'interface.



"Almost evryone agrees that dialogue and computation componants must be seperated. But an ideal seperation is hard to define and even harder to achieve [Hartson 89]".

Il s'agit en quelque sorte du modèle de base qui sera raffiné. Cette distinction entre application et interface définit un premier niveau de modularité.

Le problème visant à déterminer la ligne de séparation entre les composants de l'application et ceux de son interface est une tâche difficile. Nous partirons du principe qu'il y indépendance du dialogue vis à vis de l'application ou autrement dit qu'il y séparation de l'interface et de l'application si toute décision de conception qui affecte l'interface peut être isolée de toute décision de conception qui affecte la structure sémantique de l'application [Hartson 89]. De cette séparation découle l'indépendance de l'application par rapport à son interface puisque la partie interface devrait pouvoir être changée sans que la partie sémantique ne soit modifiée et vice versa.
Les vertus de cette séparation sont multiples. Dans [Vanderdonckt 90b] on retrouve une liste exemplative des avantages qui en découlent.

Chapitre 1 Présentation du modèle BIPS

Le modèle BIPS est un modèle d'architecture développé à l'institut d'informatique par B.Sacré et I.Provot [Provot et Sacré 90]. Il s'applique aux **applications interactives.**
Je l'ai appelé ainsi jusqu'à ce qu'un nom définitif lui soit attribué par ses auteurs.

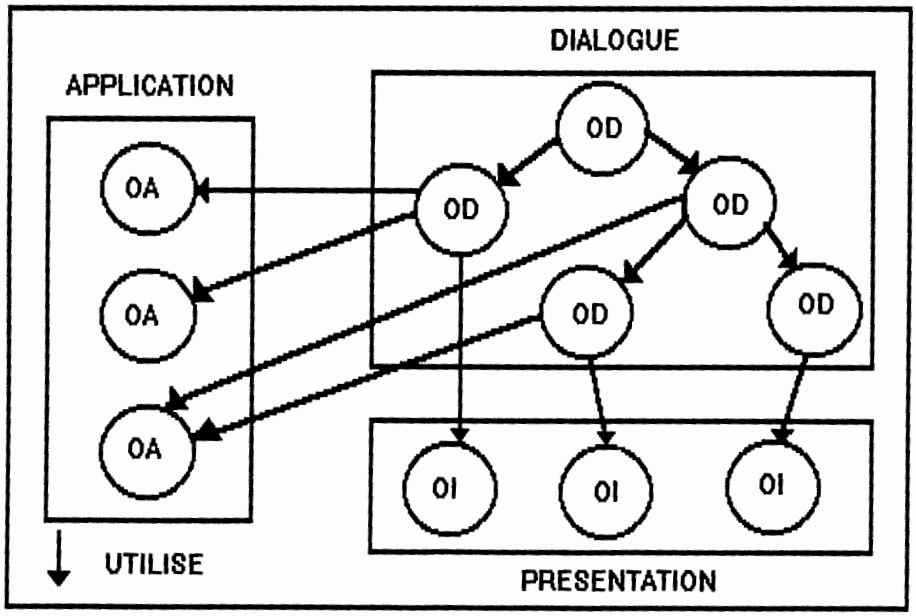
1.1 Les composants de l'architecture

Une application interactive est organisée en trois composants regroupant chacun un ensemble d'objets :

- le **dialogue**, composé d'une hiérarchie d'objets de dialogue qui utilisent les services offerts par les objets de l'application et les objets interactifs de la présentation.

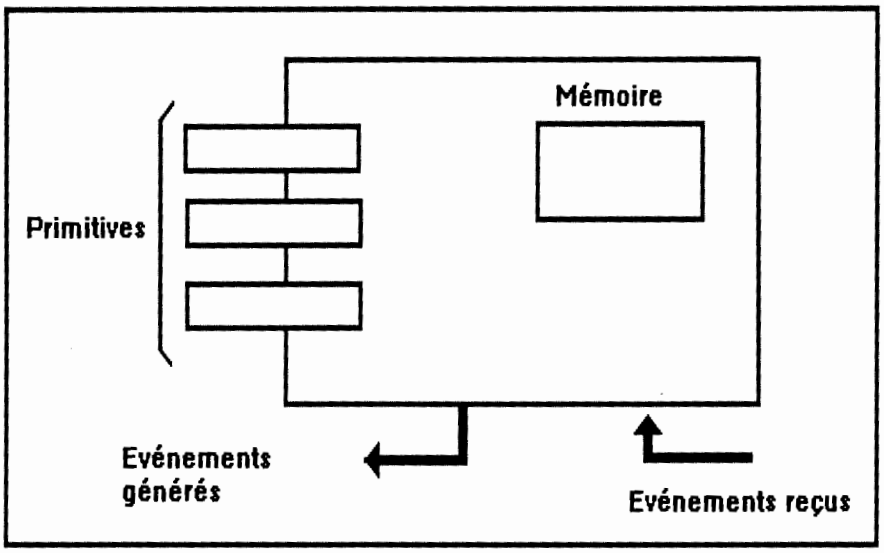
- **l'application**, vue par le composant dialogue comme un ensemble d'objets sémantiques au service du composant dialogue.

- la **présentation**, vue par le composant dialogue comme un ensemble d'objets interactifs abstraits.



1.2 Modèle descriptif d'un objet

Chaque objet est défini par son interface, à savoir ses **primitives** de manipulation et les **événements** qu'il génère. Chaque objet possède une **mémoire** composée d'un ensemble de variables et est caractérisé à un moment donné par un état défini par l'ensemble des valeurs des variables. Tous les objets sont manipulables par d'autres objets à l'aide des primitives qui constituent des services offerts par l'objet.
On définit des classes d'objets qui regroupe des objets ayant un comportement similaire.



Remarquons que le concept d'objet du modèle BIPS ne correspond pas complètement à celui d'un langage classique de programmation par objets.

Le concept d'objet repose sur le principe d'encapsulation : l'objet regroupe au sein d'une entité unique des données et les méthodes qui les manipulent. Cela permet de réaliser l'abstraction des données : l'objet est muni d'une interface qui spécifie les interactions qu'il peut avoir avec l'extérieur et la seule manière de communiquer avec lui est d'invoquer une des méthodes de son interface.

Un objet étant une unité indépendante, la seule façon pour lui d'agir sur un autre objet ou de communiquer avec d'autres objets, est d'utiliser une méthode appartenant à l'interface de cet autre objet, et de lui envoyer un message, qui demande l'exécution de la primitive en question.

Un objet du modèle BIPS est aussi défini par une interface, mais composée d'une part des primitives de manipulation, et d'autre part des **événements** qu'il génère.

Primitives
Une primitive définit une opération que l'on peut effectuer sur un objet. Ce concept s'apparente à celui de méthode d'un objet. Le déclenchement d'une primitive peut s'effectuer de façon synchrone ou asynchrone.

Evénements
Un objet peut recevoir ou générer des événements asynchrones. Il envoie des événements aux objets qui ont marqué leur intérêt pour eux. Ces événements correspondent à des changements d'état de l'objet, provoqués par l'exécution d'une primitive ou résultant du traitement des événements en provenance d'autres objets.

1.3 Le composant Présentation

1.3.1 Présentation

Un **objet interactif** est une entité de dialogue visible par l'utilisateur. Les objets interactifs sont offerts par les toolkits de construction d'interfaces homme-machine (le Macintosh Toolbox, X-Windows Toolkit, Osf/Motif Toolkit, Hp-Widgets Toolkit).

Un objet interactif est un objet de dialogue utilisé pour la saisie et l'affichage d'informations relatives à la tâche de l'opérateur. On distingue deux types d'objets : les objets interactifs élémentaires et les objets interactifs composés c'est-à-dire décomposables en objets élémentaires et/ou d'autres objets composés.
Parmi les objets interactifs élémentaires, on trouve la barre de défilement, le bouton, l'icône, la boîte de sélection, le cadran, le champ de saisie textuelle. Parmi les objets interactifs composés citons la boîte de dialogue, la fenêtre, le menu.

Afin d'être indépendant d'un toolkit particulier, nous considérons les objets interactifs comme des entités logiques ou objets interactifs abstraits (par opposition aux objets physiques propres à un toolkit particulier).
Un objet interactif abstrait définit un objet du dialogue du point de vue de son comportement et des opérations ou primitives permettant la manipulation de l'objet. Des événements abstraits asynchrones sont produits par l'objet interactif en réaction à une action significative de l'utilisateur sur l'objet.
Les événements générés par un objet interactif à destination des objets de dialogue correspondent à des actions significatives de l'utilisateur sur l'objet interactif.

On définit des **classes d'objets interactifs**. Une classe regroupe les objets qui ont un comportement similaire.

1.3.2 Exemple d'objet interactif

Voici pour exemple, une classe d'objet interactif prédéfinie. Elle correspond à la définition d'une LIST-BOX (voir aussi annexe 2). Une list-box est une liste déroulable d'éléments représentant des choix parmi lesquels l'utilisateur a la possibilité de faire une sélection (représentation graphique de l'objet interactif, voir annexe 1).

LIST-BOX

Primitives

- pr_lbx_ajout_élem	ajoute un élément dans la liste
- pr_lbx_obt_élem	permet d'obtenir l'élément sélectionné

Evénements

- ev_lbx_select	généré lorsqu'un élément de la liste est sélectionné
- ev_lbx_désélect	généré lorsqu'un élément de la liste est désélectionné

1.3.3 Objets interactifs

En annexe1, on présente quelques objets interactifs textuels au lecteur. Pour une description détaillée et une liste complète des objets interactifs existants se référer à [Provot et Vanderdonckt 90].

1.4 Le composant application

1.4.1 Présentation

Ce composant est vu par le composant de dialogue comme un ensemble d'objets de l'application.

Un objet de l'application est une entité de l'application manipulable par les objets de dialogue à l'aide de primitives et susceptible d'émettre des événements à destination d'un objet de dialogue.

L'entité de l'application s'apparente à un élément de la mémoire de l'application (entité, association, sous-schéma) et les primitives correspondent aux fonctions de l'application. Le déclenchement d'une fonction ou primitive d'un objet de l'application par un objet de dialogue peut s'effectuer de façon synchrone ou asynchrone. Le déclenchement asynchrone d'une primitive associée à un objet de l'application suppose que celui-ci puisse émettre des événements à destination de l'objet de dialogue qui a demandé l'exécution de la primitive.

On définit des **classes d'objets de l'application** . Une classe regroupe les objets qui ont un comportement similaire.

A noter que le composant Application de BIPS est dynamique ou réactionnel. L'application a en effet possibilité d'avertir le composant Dialogue (par émission d'événements d'un objet de l'application à un objet de dialogue) de modifications de ses objets susceptibles d'intéresser le composant dialogue et ce essentiellement pour qu'il puisse mettre à jour la présentation.

1.4.2 Exemple d'objet de l'application

Considérons l'exemple "enregistrement de commande-client" envisagé au point 1.6 de ce chapitre. On y mémorise les commandes de produits d'un client.
L'application est constituée d'objets Produit, Client et Commande (voir aussi annexe 2).

Voici la définition d'une classe d'objet "CLIENT".

Primitives

- pr_validation_num_client (numéro) -> booléen
  détermine si un numéro de client est valide ou non
- pr_validation_cli_client (CLI) -> booléen
  détermine si un nom et un prénom de client sont valides ou non
- pr_obt_info_client (numéro) -> CLI
  renvoie les informations concernant un client de numéro donné dans une structure qui contient le nom et le prénom du client
- pr_obt_lst_client -> CLI
  renvoie, par itération, la liste des clients existant dans une structure
  qui contient le nom et le prénom
- pr_obt_num_client (CLI) -> numéro
  retourne le numéro d'un client dont on connait le nom et le prénom
- pr_modif_adresse_client (numéro, ADRESSE)
  modifie l'adresse d'un client existant
- pr_enregistrer_client (CLIENT) -> numéro
  enregistre un nouveau client

Evénements

- ev_client_modifié
  généré lors de toute modification apportée à la liste des clients (ajout, suppression) ou lors de toute modification à des clients existants (ex: changement de nom)

1.5 Le composant dialogue

1.5.1 Présentation

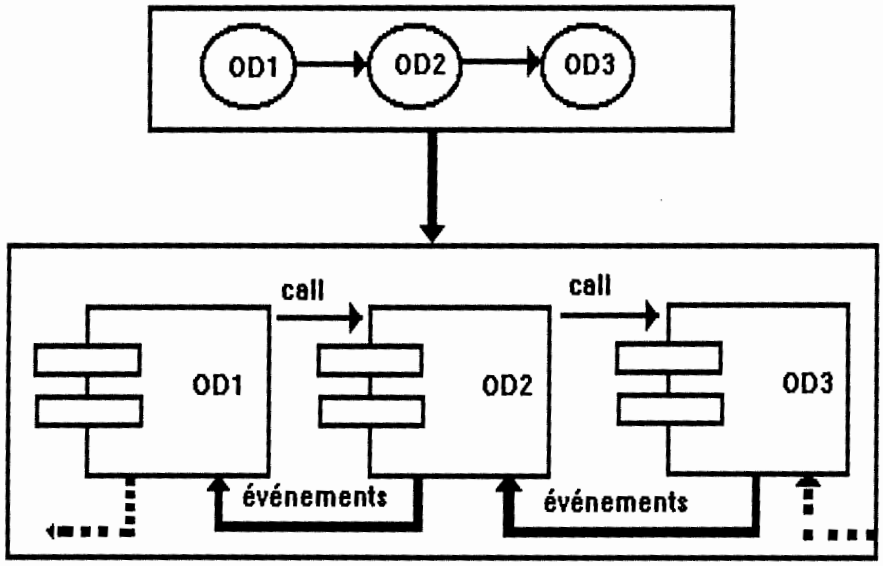
Un objet de dialogue est une entité **abstraite** de dialogue manipulable à l'aide de primitives et qui communique avec d'autres objets via des événements. Chaque objet possède une mémoire qui maintient son état et est caractérisé par un comportement défini par l'ensemble des actions associées aux changements d'états.

Structuration des classes d'objets de dialogue

On structure les objets de dialogue en niveaux d'abstraction à l'aide de la relation logicielle "**utilise**". Pour rappel "un composant A utilise un composant B" implique et réciproquement que le fonctionnement correct de A dépend de la disponibilité d'une version correcte de B.

Les objets feuilles de la hiérarchie correspondent aux objets interactifs. Ces objets peuvent être utilisés par des objets de dialogue plus abstraits (et non visible) et récursivement.

La relation "utilise" entre deux objets de dialogue se traduit par un **protocole de communication asynchrone**.



Si un objet de dialogue OD2 est relié à un autre objet OD3 par une relation "utilise", l'objet OD2 dispose de l'ensemble des primitives offertes par OD3 et OD3 avertit son père des changements d'états au moyen d'événements.

Classes d'objets de dialogue

On définit des **classes d'objets de dialogue** . Une classe regroupe les objets qui ont le même comportement. La définition de classes d'objets de dialogue permet une **réutilisabilité** des objets définis une fois pour toutes. Dans le contexte des IHM, l'utilisation de ce concept nous assure également une **cohérence** de l'interaction.

Il est possible de définir de nouvelles classes d'objets de dialogue en se basant sur des classes existantes. Les nouvelles classes seront des sous-classes ou des super-classes selon qu'elles sont une spécialisation ou une généralisation des classes existantes.

Réalisation d'une classe d'objet de dialogue

Pour créer un nouvel objet de dialogue, il convient de définir son **interface** (primitives et événements) et de décrire les **actions** effectuées en réponse aux primitives et aux événements reçus par l' objet de dialogue.
Afin de décrire le **comportement** d'un objet de dialogue, nous proposons un **langage de règles** décrit dans la section 2 de ce mémoire.

1.5.2 Exemple d'objet de dialogue

Considérons l'exemple "enregistrement de commande-client" envisagé au point 1.6 de ce chapitre (voir aussi annexe 2).
Voici la définition de l'objet de dialogue "Gestion_Clients_Existants".

Description

Gestion_Clients_Existants affiche les clients existant dans la base de donnée.

Description de l'interface

Primitives

- pr_création — crée une instance de la classe
- pr_sélection_client — détermine si un client est sélectionné ou pas

Evénements

- ev_sélection_client — généré lorsqu'un client est sélectionné
- ev_non_sélection_client — généré lorsqu'il n'y a plus de client sélectionné

Les classes d'objets interactifs utilisées

BOUTON, LISTBOX, FENETRE

Les classes d'objets de l'application utilisées
CLIENT

1.6 Exemple

Nous présentons maintenant un exemple d'architecture élaborée à partir de notre modèle.

Il s'agit de la phase "ENREGISTREMENT DE COMMANDE-CLIENT".
Cette phase a pour objectif de vérifier une commande-client et , lorsqu'elle est valide de la mémoriser. De plus, elle doit permettre de mémoriser un client qui commande pour la première fois et d'enregistrer un changement d'adresse communiqué pour un ancien client.

Elle utilise et/ou modifie une mémoire qui comprend :

- des commandes (leurs provenances et leurs lignes)
- des clients
- des produits

Elle reçoit une commande à enregistrer qui peut comprendre :

- un numéro de client
- un nom de client
- une adresse du client
- des numéros de produits
- des quantités commandées

Cette phase intègre les fonctions suivantes :

- mémorisation d'une commande
- mémorisation d'un client
- mémorisation d'une adresse
- validation d'une commande
- validation d'une ligne de commande
- validation d'un client
- validation d'un produit.

Pour la saisie d'un client, on saisit soit le numéro du client et après sa validation, dans le cas valide, son nom, prénom et adresse sont affichés; soit on saisit le nom et prénom et après validation, dans le cas valide, son numéro et son adresse sont affichés; soit on saisit le nom et le prénom et l'adresse s'il s'agit d'un nouveau client et après validation un numéro lui est attribué qui est affiché.
Il doit être possible de modifier l'adresse d'un ancien client.

Pour la saisie d'une commande, l'utilisateur entre un numéro de produit et alors après validation, dans le cas correct, le libellé et le prix unitaire du produit sont affichés; ensuite il entre une quantité de ce produit et après validation, le montant de la ligne de commande est affiché ainsi que le montant total de la commande.

Afin d'illustrer la mise à jour à l'écran d'informations de la base, nous avons introduit une fonction de service qui affiche les clients existants de la base. Cette liste de clients doit par conséquent être mise à jour lors de toute modification qui a trait à l'ensemble des clients de la base.

L'aspect présentation
Nous avons imaginé ce que seraient les écrans d'entrées/sorties et, par là-même, identifié les objets interactifs et leurs attributs de présentation (taille, position, ...) en essayant de respecter certaines règles d'ergonomie de présentation.

En choisissant de matérialiser les fonctions de l'application au sein de l'interface, nous avons opté pour un déclenchement de ces fonctions à l'initiative de l'utilisateur. Ce choix semble être communément admis pour les applications développées sous des environnements multi-fenêtrés. Il répond également à un souci ergonomique qu'il conviendra de préciser. Le choix de l'utilisation de boutons poussoir à la place d'un menu constitue quant à lui un choix relevant de l'ergonomie de présentation.

Remarquons que les choix ergonomiques de présentation impliquent également des choix quant au mode de fonctionnement de l'application qui relèvent d'un autre type d'ergonomie.

L'apect dialogue ou conversation
A ce stade de la démarche, nous disposons de l'ensemble des objets interactifs et des fonctionnalités de l'application.

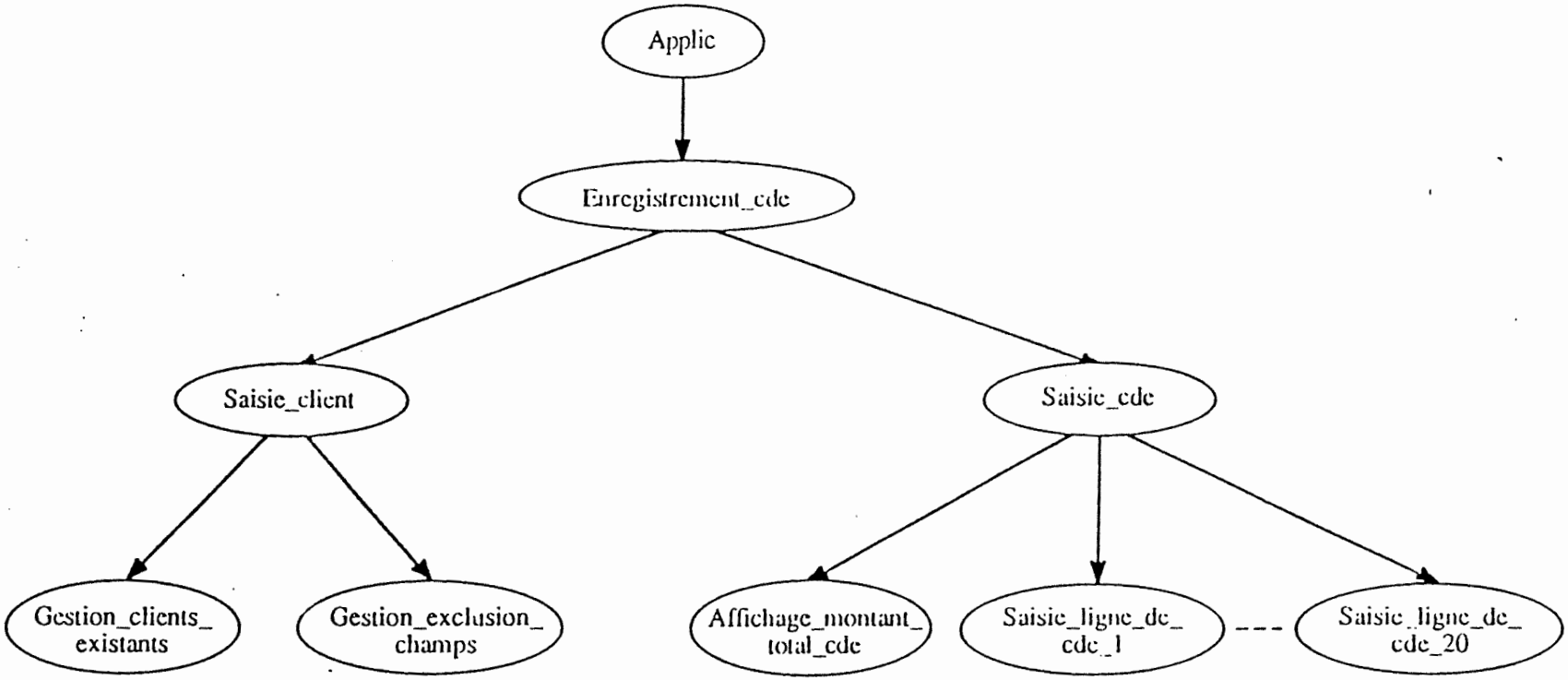
Nous avons identifié 3 objets de dialogue de base :
- saisie de la commande
- saisie du client
- enregistrement de la commande

Nous avons affiné chacun de ces composants en identifiant leurs sous-composants compte tenu de l'objectif assigné à chacun des composants et des objets interactifs manipulés.

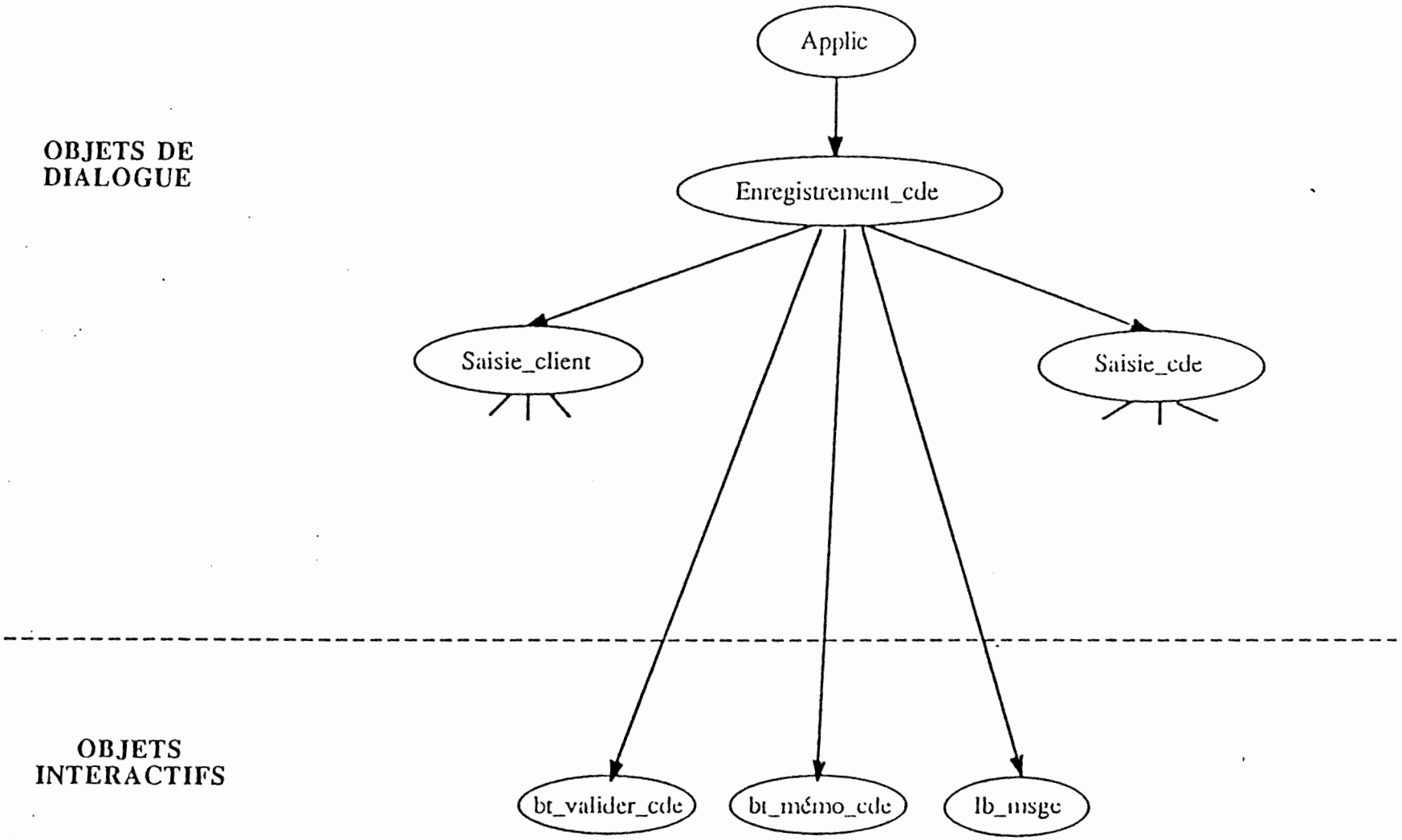
Nous avons ensuite spécifié l'interface de chaque objet de dialogue, c'est-à-dire les primitives offertes et les événements générés.

Disposant d'une spécification complète de chaque objet de dialogue, nous avons pu réaliser leur comportement individuellement en utilisant notre langage de règles (voir section 2 et annexe 2).
La réalisation du comportement de chaque objet consiste à identifier les actions à effectuer lors de l'utilisation d'une primitive et à construire l'ensemble des règles (conditions et actions) réalisant le comportement de l'objet de dialogue vis à vis des événements en provenance des objets de dialogue fils et/ou des objets de l'application.

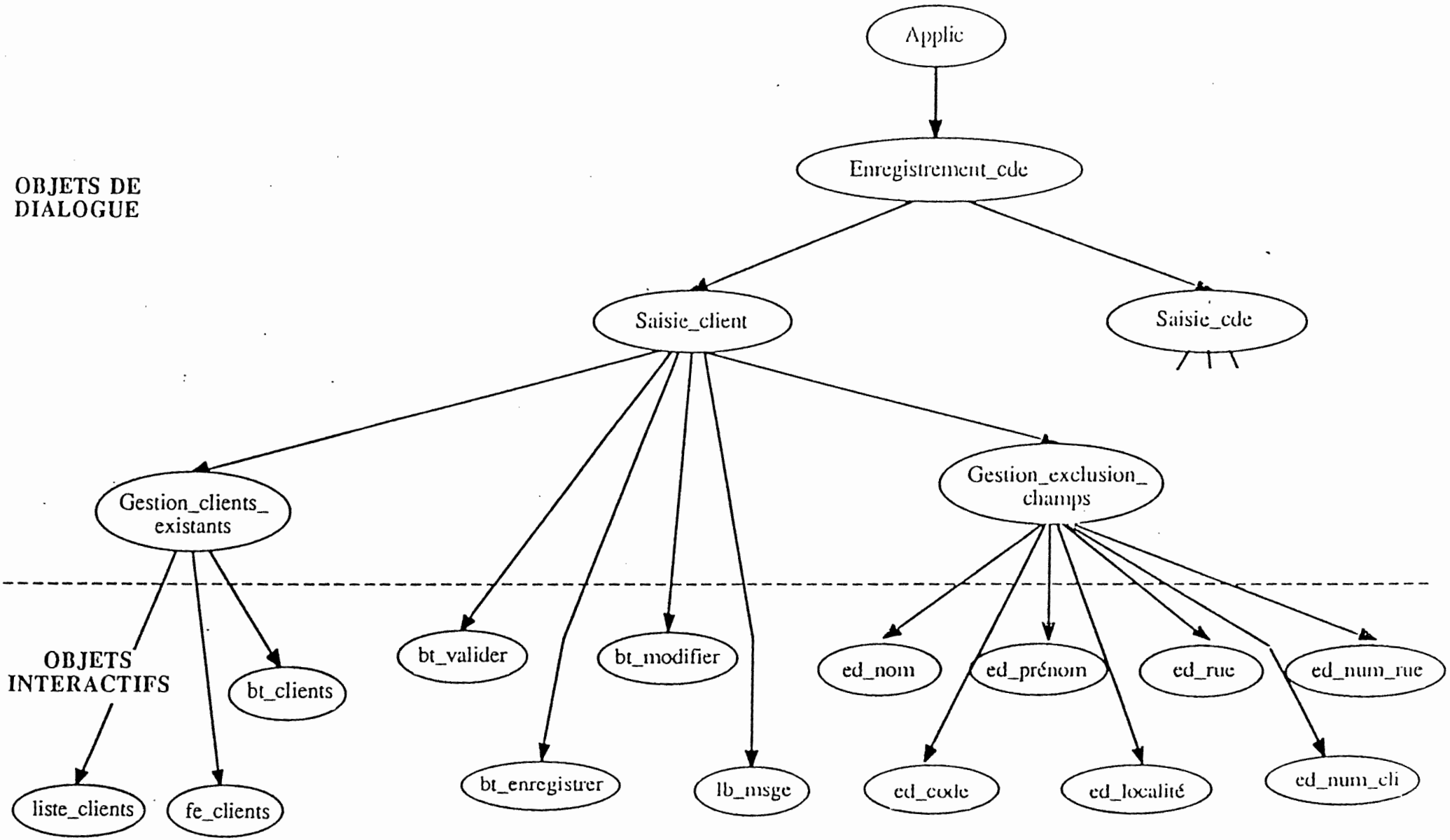
Ci-après les schémas de la hiérarchie des objets de dialogue, ainsi que l'interface de l'application .



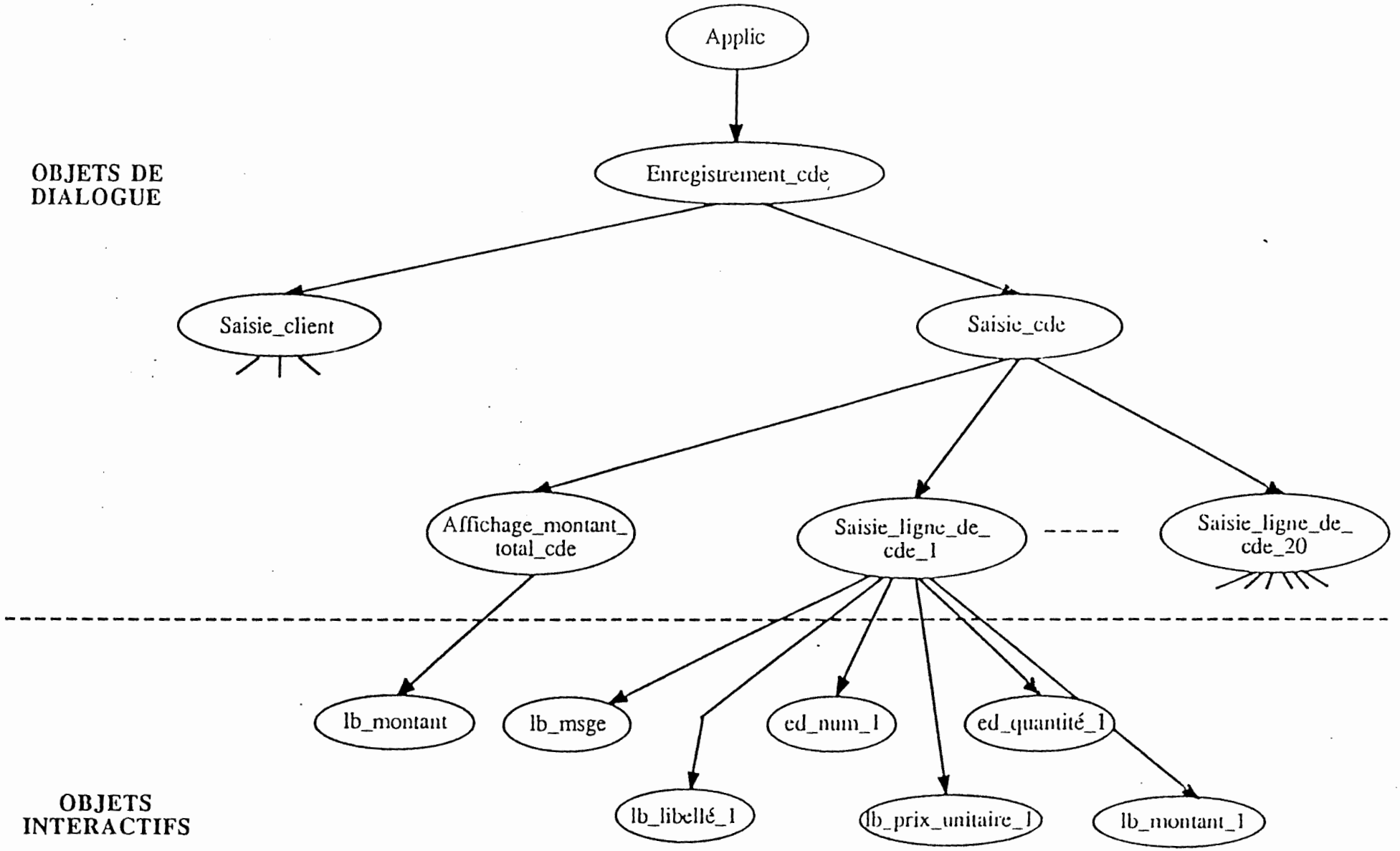
Hiérarchie des objets de dialogue



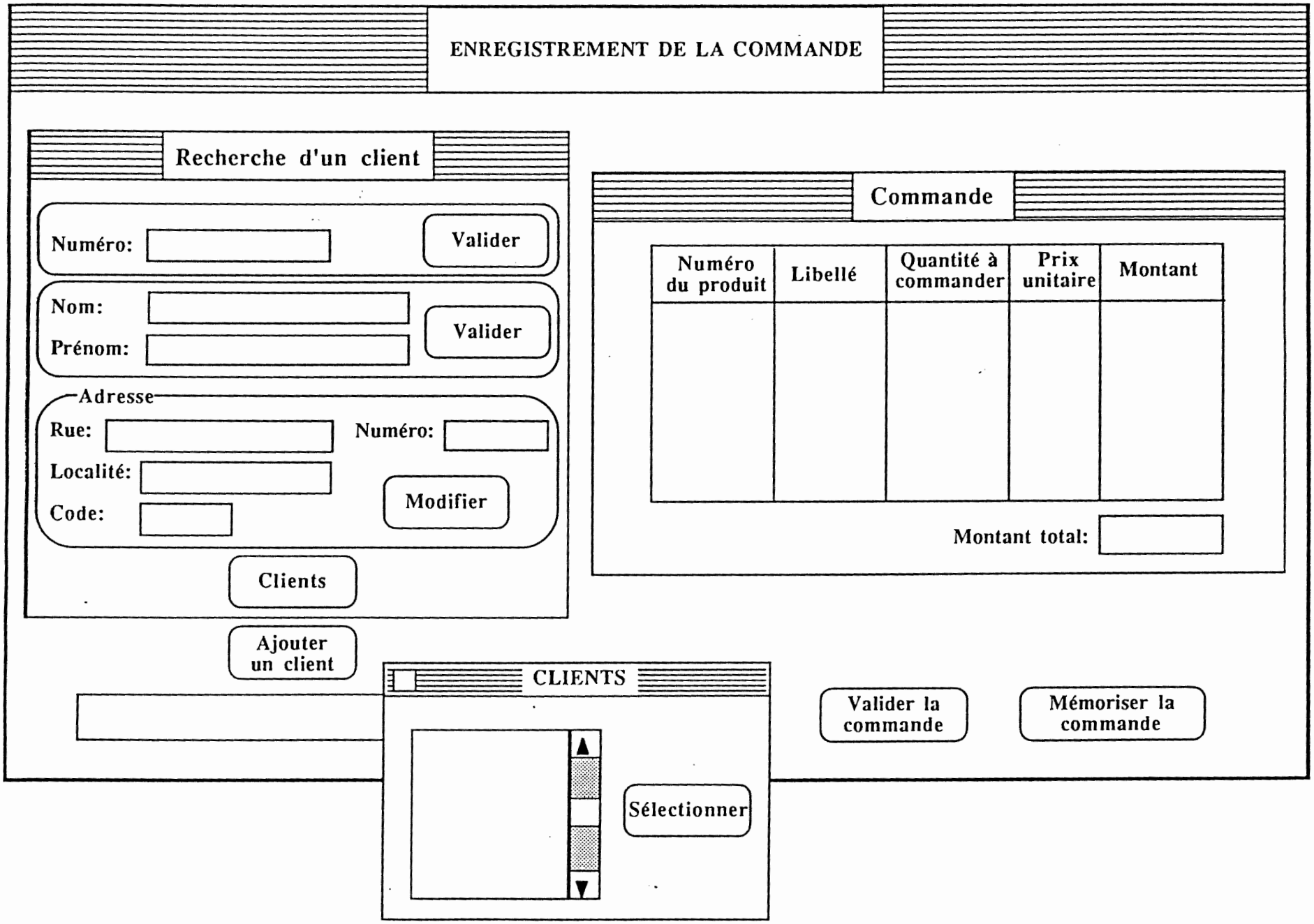
ENREGISTREMENT_CDE



SAISIE_CLIENT


Applic
OBJETS DE DIALOGUE
Enregistrement_cde
Saisie_client
Saisie_cde
Affichage_montant_ total_cde
Saisie_ligne_de_ cde_1
Saisie_ligne_de_ cde_20
lb_montant
lb_msge
ed_num_1
ed_quantité_1
OBJETS INTERACTIFS
lb_libellé_1
lb_prix_unitaire_1
lb_montant_1

SAISIE_CDE


ENREGISTREMENT DE LA COMMANDE
Recherche d'un client
Numéro:
Valider
Nom:
Prénom:
Valider
Adresse
Rue:
Numéro:
Localité:
Code:
Modifier
Clients
Ajouter un client
Commande
Numéro du produit
Libellé
Quantité à commander
Prix unitaire
Montant
Montant total:
CLIENTS
Sélectionner
Valider la commande
Mémoriser la commande

Chapitre 2 Comparaison du modèle BIPS avec d'autres modèles

2.1 Types de modèle

J.Coutaz dans [Coutaz 90] identifie trois types de modèle d'architecture : les modèles de type entrée/sortie, les modèles de type langage et les modèles de type multiagent. Chaque type de modèle offre une vue complémentaire sur la façon d'organiser une application interactive.

- Un modèle d'architecture de type langage s'inspire de l'analogie entre l'interaction homme-machine et un dialogue entre les individus. Le langage utilisé par le système et l'utilisateur s'observe selon les dimensions usuelles d'un modèle linguistique : sémantique-syntaxe-lexique.

- Un modèle d'architecture de type entrée/sortie représente l'interaction comme une succession d'opérations d'échanges et de traitements d'informations opérés à divers niveaux d'abstraction. La forme des échanges et la nature des transformations définissent le découpage fonctionnel du système.

- Un modèle d'architecture de type multiagent organise un système interactif en un ensemble d'agents coopérants. Ce système s'inspire du fonctionnement des systèmes de type stimuli-réponse. Il s'agit de modèle d'architecture orienté objet.

2.2 Le modèle de Seeheim

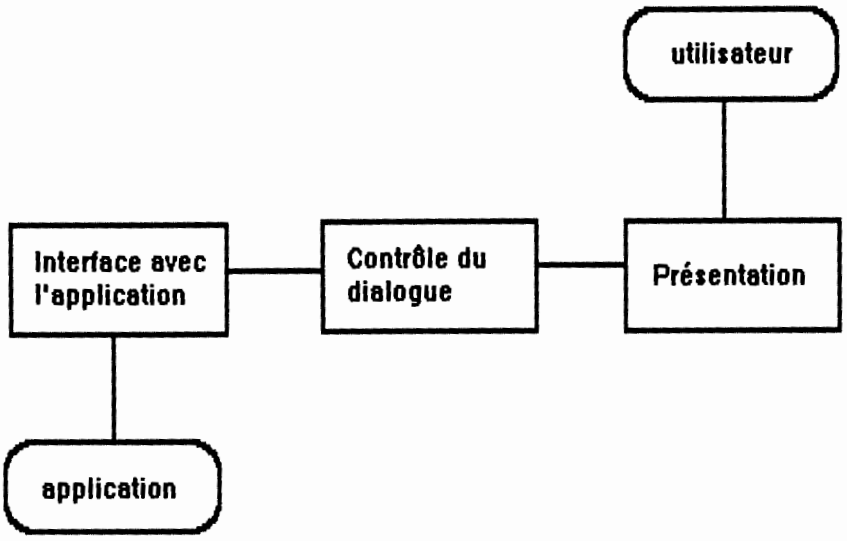
Foley et Van Dam [Foley 84] furent les premiers à plaquer une vue linguistique sur l'interaction homme-machine. Mais c'est le modèle de Seeheim qui a su le premier exprimer cette vue sous la forme d'un modèle générique utilisable.

La sémantique définit la signification (pas la forme) des phrases. Elle représente les concepts et les fonctionnalités dans un domaine donné.

La syntaxe définit les règles de construction des phrases du langage à partir des éléments syntaxiques. Un élément syntaxique est une unité qui ne peut être décomposée plus en avant sans perdre sa signification [Coutaz 90] .

Le lexique définit la production des unités syntaxiques à partir d'un vocabulaire qui comprend tout ce qu'il est possible de créer à partir des dispositifs d'entrée et de sortie : caractères, souris, son, primitives graphiques d'une bibliothèque.

2.2.1 Présentation du modèle de Seeheim



Le modèle de Seeheim est un modèle de type langage[Green 85].
Ce modèle, comme le montre la figure ci-dessus, structure une application interactive en trois composants : la présentation, le contrôleur du dialogue et l'interface avec l'application.

La présentation
Cette partie gère la représentation physique du système. C'est le seul composant qui est autorisé à connaître et à manipuler les dispositifs d'entrée/sortie. Il assure la présentation des concepts et données à l'écran.
Du point de vue linguistique, il effectue l'analyse lexicale du langage d'présentation. C'est le composant par lequel passent nécessairement les autres composants pour effectuer des échanges avec l'utilisateur.

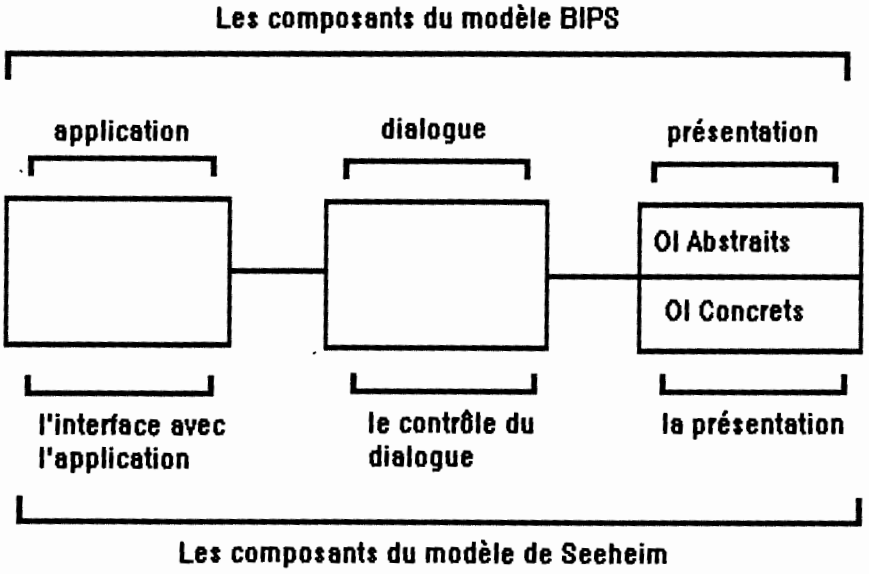
L'interface avec l'application
Ce composant définit la vue que le contrôleur du dialogue a de l'application. Rappelons que l'application réunit les concepts et les fonctions du domaine, constituant la sémantique du langage d'présentation.

Le contrôle du dialogue
Ce composant joue le rôle de médiateur entre la présentation et l'interface avec l'application. Il est responsable de l'analyse syntaxique du langage d'présentation. Il vérifie la validité et la cohérence des informations introduites.

A ces rôles de médiateur et d'analyseur syntaxique, s'ajoute celui de gestionnaire du dialogue, ou de l'état de l'présentation. Il gère le déroulement des saisies et affichages, ainsi que le séquencement des traitements.

2.2.2 Evaluation comparative avec notre modèle

Nous pouvons essayer, **au plus haut niveau d'abstraction,** d'établir une correspondance entre les composants du modèle de Seeheim et ceux du modèle BIPS.
Au plus haut niveau d'abstraction, le composant application correspond au composant interface avec l'application du modèle de Seeheim.
Au plus haut niveau d'abstraction, le composant dialogue correspond au composant contrôle du dialogue du modèle de Seeheim.
Au plus haut niveau d'abstraction, le composant présentation correspond au composant présentation du modèle de Seeheim.



Application

Le composant Application de BIPS correspond au composant Interface avec l'application du modèle de Seeheim.
Ils jouent en effet les mêmes rôles. Tous les deux ont pour objectif de définir une vue de l'application. Cette vue permet d'être mieux adaptée aux traitements de l'interface.

Dans le cas du modèle de Seeheim, l'application est vue comme une bibliothèque de routines utilisées par la partie dialogue.

Dans le cas de BIPS, l'application est vue comme un ensemble d'objets sémantiques au service du composant dialogue. Chaque objet de l'application est accessible via les primitives de son interface.

Le composant interface avec l'application reçoit du composant contrôle du dialogue des phrases du langage d'présentation à convertir dans un formalisme compréhensible par l'application. Il assure aussi le mécanisme de conversion inverse, du formalisme de l'application vers celui de son interface.
Dans BIPS, ce sont les objets du composant dialogue qui mettent en correspondance les formalismes des deux autres composants et qui utilisent leurs services.

Dialogue

Le composant dialogue du modèle BIPS correspond au composant au composant contrôle du dialogue du modèle de Seeheim. Tous les deux jouent un rôle d'intermédiaire entre l'application et sa présentation.

Ils gèrent tous les deux l'état de l'présentation et le déroulement du dialogue avec l'utilisateur. Dans le modèle de Seeheim, la définition d'un état peut inclure l'ensemble des requêtes permises. Dans le modèle BIPS, l'état de l'présentation est défini à un moment par l'ensemble des valeurs des variables constituants les mémoires des objets de dialogue et l'ensemble des primitives disponibles à ce moment définit le déroulement possible de l'interaction.

Le contrôleur du dialogue est ainsi responsable de l'analyse syntaxique du langage d'interaction : les unités syntaxiques reçues de la présentation sont assemblées en phrases. Les phrases syntaxiquement correctes correspondent à des requêtes et à des données que l'utilisateur souhaite transmettre à l'application. Dans le sens inverse, le contrôleur reçoit des phrases de sortie qu'il distribue vers les éléments spécialisés de la présentation.

Dans le modèle BIPS, le contrôle du dialogue est réparti entre les objets de dialogue qui utilisent des objets interactifs pour la présentation et les services offerts par les objets de l'application. La relation "utilise" entre deux objets se traduit par un protocole de communication asynchrone basé sur l'émission d'événements entre les objets. Le déclenchement d'une fonction ou primitive d'un objet de l'application par un objet de dialogue peut d'effectuer de façon synchrone ou asynchrone.

Présentation

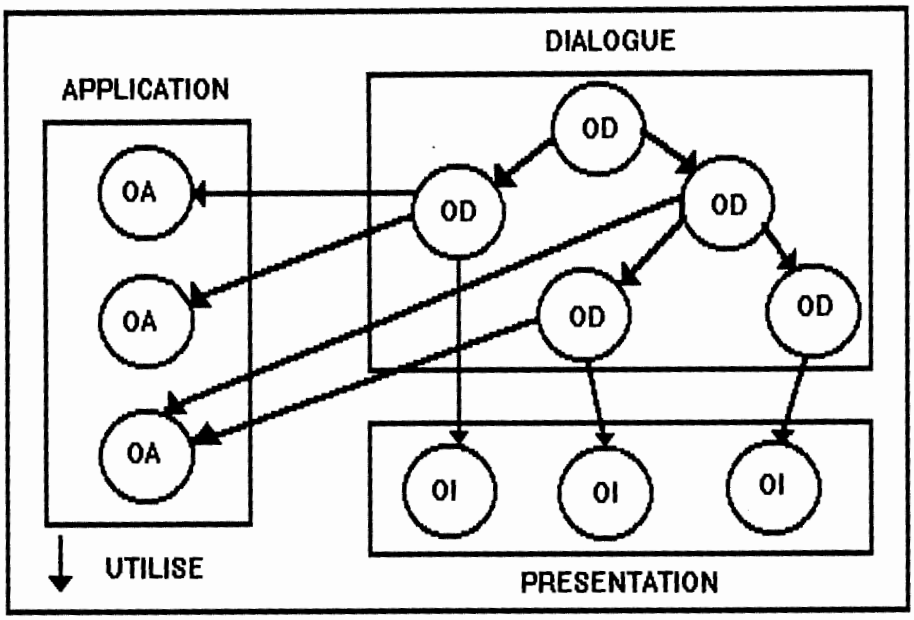
Au niveau de la présentation, la correspondance n'est pas directe. Le composant Présentation du modèle BIPS est, rappelons-le, constitué d'un ensemble d'OI abstraits. Cette interface explicite avec le niveau boîte à outils est donc indépendante des techniques d'affichage de bas niveau. Ce concept d'interface abstraite n'est pas présent dans le modèle de Seeheim.

C'est le composant présentation qui correspond au composant présentation du modèle de Seeheim. Les unités lexicales d'entrée (événement concret comme un clic souris) émis par les objets du toolkit, sont transformés en événements abstraits (par exemple, sélection d'un élément dans une liste) de plus haut niveau et sont transmis aux objets du composant dialogue. De même, les événements abstraits en provenance du composant dialogue sont transformés en termes du lexique de sortie du composant présentation.

Le composant présentation du modèle BIPS et celui du modèle de Seeheim sont les seuls composants à interagir directement avec les dispositifs physiques d'entrée/sortie. Ces composants assurent donc l'indépendance du reste des composants avec ces dispositifs. Ils gèrent la présentation à un niveau d'abstraction le plus bas.

Structuration

Si le modèle de Seeheim est applicable aux applications interactives, il ne dit rien sur la manière de structurer ses trois composants.
BIPS va plus loin en offrant une solution plus complète que Seeheim. Il offre une technique de structuration des composants de l'architecture en se basant sur la relation "utilise", laquelle permet la définition de niveaux d'abstraction. Chaque composant est modélisé sous forme d'un ensemble d'objet. Une hiérarchie est ensuite établie entre ces objets sur base de la relation logicielle "utilise".



2.2.3 Résumé de la comparaison

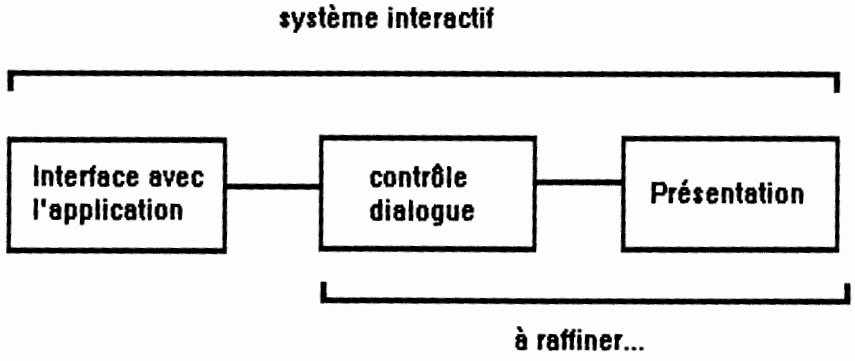
Au plus haut niveau d'abstraction, nous ne pouvons pas qualifier le modèle BIPS de modèle de type langage. La correspondance entre les composants des deux modèles n'est que partielle.

Historiquement, Seeheim constitue un des tous premiers modèles à avoir été proposé (1984) pour la modélisation des systèmes interactifs. On peut lui accordé qu'il est d'une grande simplicité. Il fournit en fait à un haut niveau d'abstraction, un cadre de pensée plutôt qu'une manière d'organiser un système interactif.
La distinction entre "Application" et "Interface" définit un premier niveau de modularité. C'est ce que nous avons appelé l'indépendance du dialogue et que l'on retrouve dans les deux modèles d'architecture. Ces deux modèles vont même plus loin dans la modularité, en identifiant un troisième composant médiateur : le composant dialogue de BIPS ou contrôle du dialogue de Seeheim.

Comme nous l'avons signalé, Seeheim n'énonce aucun principe pour l'aide à l'organisation et à la structuration de ses composants. BIPS, par contre, offre un cadre systématique pour la conception d'une architecture et de ses composants. Notre modèle définit une conception orientée objet de l'architecture d'une application interactive. Il structure les objets de l'architecture en une hiérarchie basée sur la relation "utilise" qui définit plusieurs niveaux d'abstraction.

Au début de ce mémoire, nous avons signalé qu'une des caractéristiques des interfaces actuelles, était la prise en compte de dialogues asynchrones et multi-fils. Or la dichotomie du modèle de Seeheim entre présentation et contrôle, suppose que les entrées soient traitées comme un flot séquentiel d'entités lexicales et syntaxiques, rendant difficile la prise en compte de tels types de dialogue dans une interface.

Enfin, à la gestion centralisée de l'état de l'interaction, BIPS substitue une totale répartition des charges à travers différentes classes d'objets.
Les modèles qui suivent celui de Seeheim, ainsi que BIPS, vont s'attacher à raffiner les composants dialogue et présentation. Tout comme BIPS, il présente une approche orientée objet de la modélisation d'une application interactive.



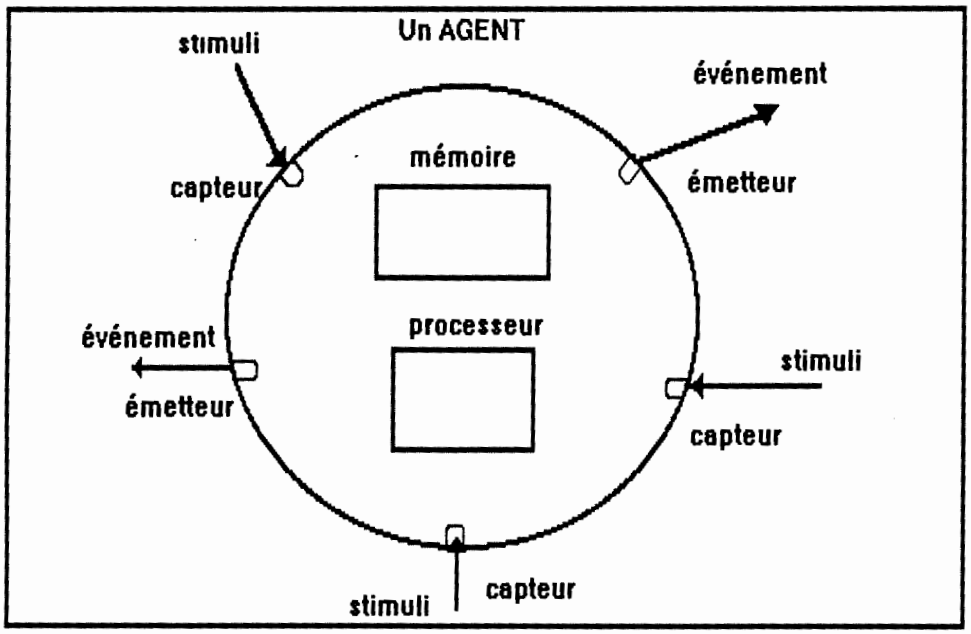
2.3 Le modèle PAC

2.3.1 Modèle de type multiagent

Les modèles que nous envisagerons maintenant sont de type multiagent. Nous allons montrer que le modèle BIPS appartient à ce type de modèle. Dès lors pour bien comprendre les modèles qui suivent, nous allons en faire une description.

Un modèle multiagent modélise un système interactif comme un ensemble d'agents coopérants et pouvant travailler en parallèle. Il s'inspire du fonctionnement des systèmes de type stimuli-réponse. Un tel système est un ensemble organisé d'agents capables de réagir à des phénomènes externes déterminés (stimuli) et de produire à leur tour des stimuli.

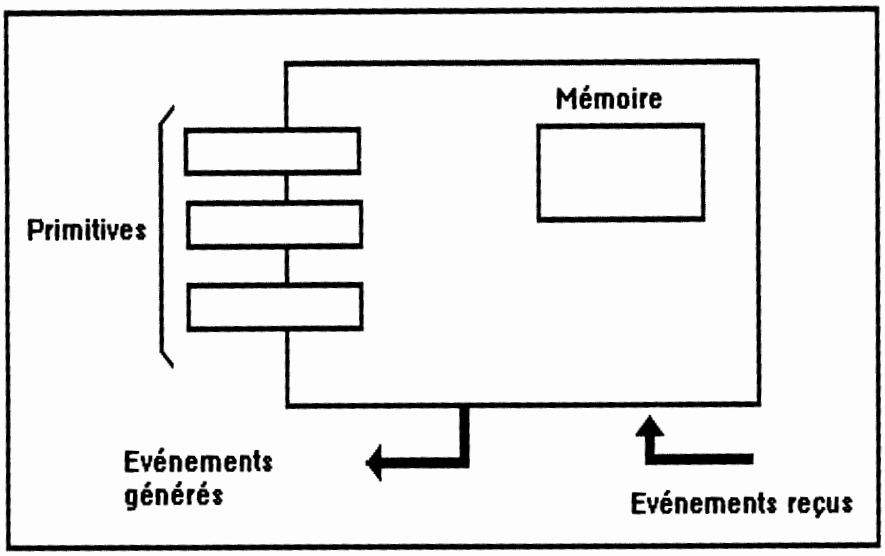
Chaque agent réactif est un système complet de traitement de l'information. Il possède un processeur, une mémoire et des dispositifs d'entrée/sortie pour capter et émettre des stimuli (événements). Ces agents réagissent à ces stimuli correspondants à des phénomènes externes déterminés et peuvent en émettre à leur tour.



Cette notion d'agent est à rapprocher de celle d'objet. Une classe définit une catégorie d'agents. Les opérateurs d'une classe constituent le répertoire d'instructions du processeur d'un agent. Les attributs constituent les éléments de la mémoire. L'appel d'un opérateur ou l'envoi d'un message modélise l'émission d'événements.

2.3.2 BIPS : Modèle de type Multiagent

Ci-dessous le rappel de la description d'un objet du modèle BIPS :



Nous voyons tout de suite apparaître des correspondances entre la notion d'agent d'un modèle multiagent et celle d'objet du modèle BIPS.

Ils réagissent et émettent tous les deux des événements. L'ensemble des événements auxquels un agent (objet) est sensible, ainsi que l'ensemble des instructions du répertoire du processeur d'un agent (primitives de manipulation) définissent le comportement de l'entité (agent ou objet). Les événements correspondent à des changements d'état de l'objet. Ces changements d'état résultent du traitement des événements en provenance d'autres objets (agents) et provoquent généralement l'émission de nouveaux événements ou l'exécution d'un traitement par un processeur.
Les primitives d'un objet du modèle BIPS, correspondent au répertoire d'instructions du processeur d'un agent. Ils possèdent une mémoire leur permettant de mémoriser leur état local dans l'interaction, ainsi que les événements survenus.

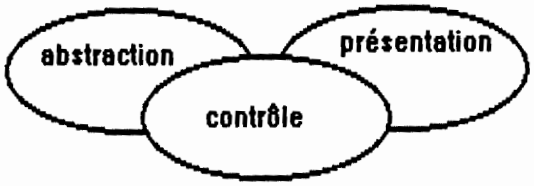
De manière plus générale, une application interactive est modélisée comme en ensemble d'agents (objets) qui coopèrent dans l'interaction. La communication entre agents se fait par émission et réception d'événements.
BIPS, tout comme le modèle multiagent, définit une approche orientée objets de la modélisation d'une application interactive.

Pour conclure, nous pouvons qualifier le modèle BIPS de modèle de type multiagent.

2.3.3 Présentation du modèle PAC

Ce modèle a été développé par J.Coutaz, professeur à l'Université de Grenoble.
Il structure de manière récursive l'architecture d'un système interactif en agents.
Chaque agent PAC est un composant à trois facettes : une Présentation, une Abstraction et un Contrôle.

Les constituants du modèle PAC



1) La présentation définit l'image du système, c-à-d son comportement en entrée comme en sortie vis-à-vis de l'utilisateur.

2) L'abstraction désigne les concepts et les fonctions du système.

3) Le contrôle maintient la cohérence entre les facettes présentation et abstraction.

Au plus haut niveau d'abstraction, un système interactif est donc un seul agent PAC. L'abstraction correspond à l'application. Le contrôle inclut les fonctions de l'interface de Seeheim. Mais la présentation, elle, rentre mal dans le cadre figé du contrôleur syntaxique et de la présentation lexicale de Seeheim.
La présentation de PAC est réalisée par un ensemble d'agents spécialisés dans l'interaction avec l'utilisateur que J.Coutaz appelle aussi objets interactifs. Ce sont comme dans le modèle BIPS des entités spécialisées dans l'interaction avec l'utilisateur.
Tout objet interactif adhère au modèle PAC.

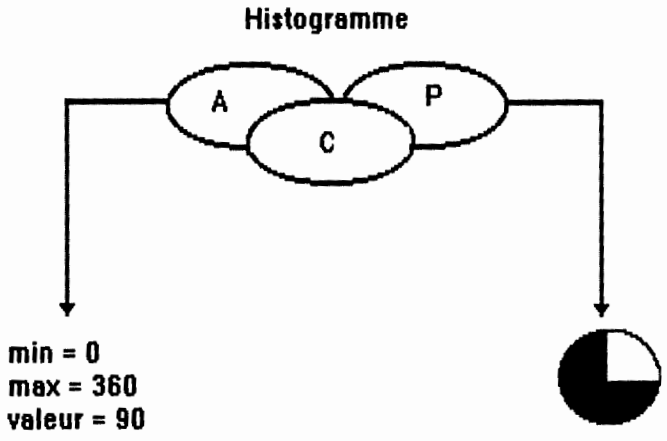
Objet interactif élémentaire

Tout objet interactif est caractérisé par une partie présentation, une partie abstraction et une partie contrôle.

La présentation définit le comportement perceptible de l'agent, c-à-d son image.
L' abstraction définit les fonctions ou attributs fonctionnels de l'agent.
Le contrôle gère les liens entre les côtés abstraction et présentation.

Chaque objet interactif possède une mémoire et un processeur dont le répertoire d'instructions définit le contrôle du point de vue des autres objets. Ses instructions permettent de traiter les événements en provenance d'autres objets interactifs et les événements générés suite aux actions de l'utilisateur sur l'objet.

Exemple tiré de [Coutaz 90] :



Ci-dessus, un exemplaire d'histogramme présenté sous la forme d'un camembert. Du côté abstraction, l'histogramme est une valeur entière comprise entre 0 et 360. Du côté présentation, c'est une surface circulaire composée de deux parties de couleurs différentes.

Objet interactif composé

Un agent PAC peut être élémentaire ou constitué d'un ensemble d'agents PAC. On dit alors qu'il est composé. Son comportement dépend de lui-même mais aussi des objets qui le constituent. L'objet composé est donc une élaboration au-dessus du fonctionnement d'agents composants. En ce sens, il définit un nouveau niveau d'abstraction.

La présentation d'un agent concerne à la fois les entrées et les sorties. Cette partie gère toutes les actions de l'utilisateur (clic souris, frappe au clavier...). Il effectue la réception des événements et l'analyse lexicale. Un événement sans effet sémantique est traité en local par la présentation qui produit un retour d'information perceptible (feedback). Un événement à effet de bord sémantique, est traité à la fois par la présentation comme précédemment, mais en plus est transmis à l'abstraction par l'intermédiaire du contrôle pour complément de traitements.

A un instant donné, l'image d'un système interactif est définie par l'ensemble des présentations des agents PAC constituants l'application interactive.

L'abstraction d'un agent est constituée de ses fonctions et attributs et définit la compétence de l'agent indépendamment des considérations de présentation. Elle définit aussi le comportement de l'agent vis-à-vis des autres agents.

A un instant donné, l'état interne d'une application interactive est donné par l'ensemble des abstractions des agents PAC constituant l'application.

Le contrôle d'un agent sert d'intermédiaire entre la présentation et l'abstraction de l'agent. Tout échange d'informations entre ces deux parties doit s'effectuer à travers le contrôle. Il fournit des mécanismes d'indirection entre présentation et abstraction. C'est aussi par leurs parties contrôles que deux agents peuvent communiquer.

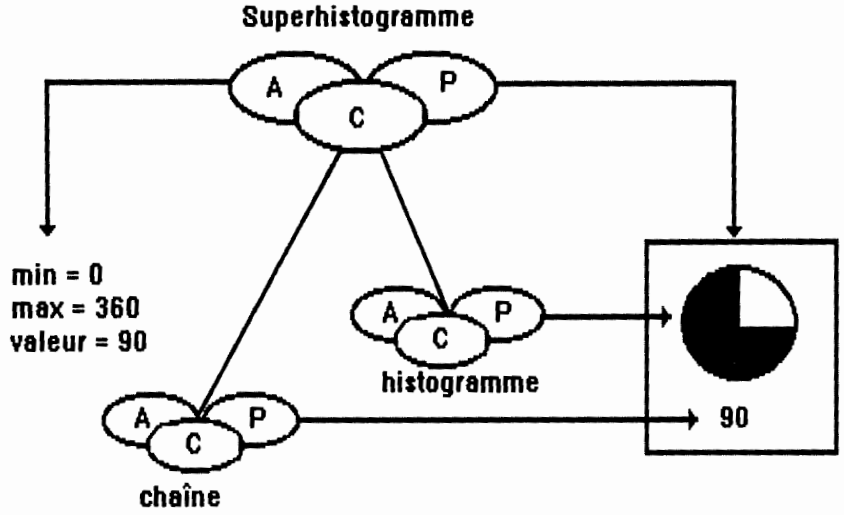
Voici d'autres fonctions que le contrôle est amené à réaliser :

- l'arbitrage qui recouvre la synchronisation et la coordination. Ces fonctions sont nécessaires entre agents mais aussi entre l'abstraction et la présentation d'un agent. En effet, ces dernières peuvent être dans certaines circonstances, le lieu d'activités asynchrones qui requièrent alors la coordination.

- la traduction de formalisme entre la présentation et l'abstraction. En effet ces deux dernières utilisent, en raison de rôles distincts, des modèles de représentation différents.

Le contrôle permet à la présentation d'un agent d'être totalement indépendante de son abstraction.
Il est important de signaler que la partie contrôle d'un agent composé gère toujours la cohérence entre les côtés abstraction et présentation mais en plus, gère la collaboration ou la dépendance entre les objets composants.

Exemple tiré de [Coutaz 90] :



L'objet interactif composé Superhistogramme est formé de deux objets élémentaires : l'histogramme de la figure précédente et une chaîne numérique. Sa présentation hérite des présentations des objets composants. L'abstraction de la chaîne est une valeur entière et sa présentation inclut la police de caractères.

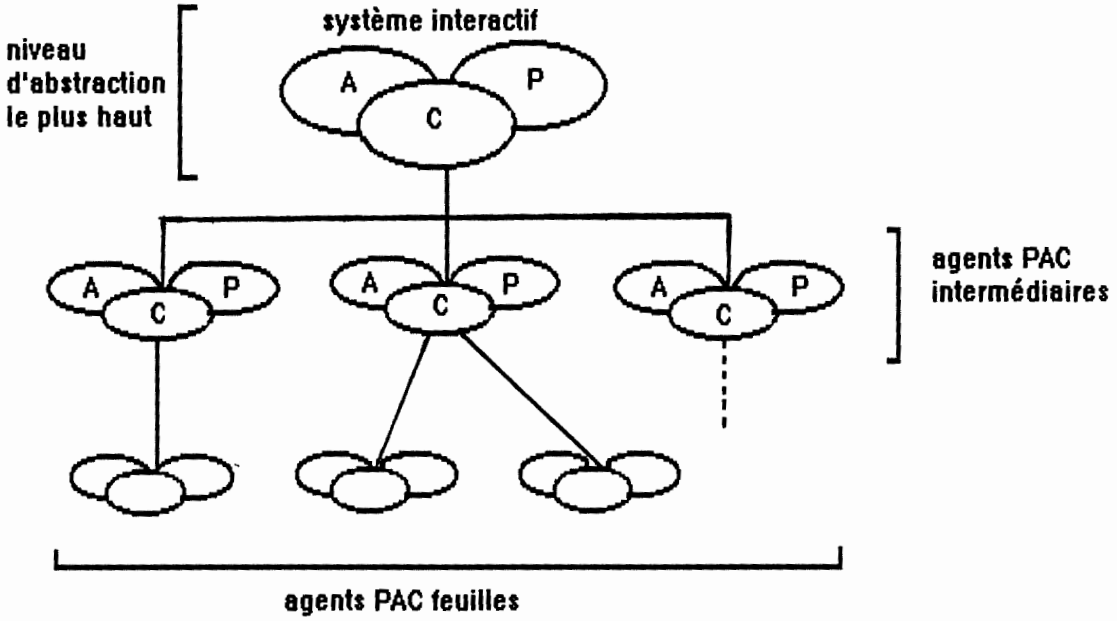
Vue complète d'un système interactif

La composition d'agents permet de construire une hiérarchie d'agents, utile à l'expression de niveaux d'abstraction. En effet par application systématique des techniques de raffinement et/ou de composition, il est possible de structurer une application interactive en une hiérarchie d'objets PAC.

Au plus haut niveau d'abstraction, on retrouve les correspondances précédemment citées.
L'abstraction correspond à l'application. C'est un producteur/consommateur de valeurs qui modélisent les concepts du domaine. La présentation organise l'image du système. Elle est constituée d'une hiérarchie d'objets interactifs.
Le contrôle, quant à lui, joue le rôle de médiateur entre ces deux derniers composants. Une de ses fonctions est de mettre en correspondance les concepts de l'application, situés dans la partie abstraction de l'agent du niveau le plus haut avec les abstractions des objets interactifs de la hiérarchie.

Les agents intermédiaires de la hiérarchie sont obtenus par raffinement et/ou composition et gèrent la coopération des agents fils ainsi que leurs rapports selon un ensemble de règles décrites dans la comparaison de BIPS avec PAC.

Les agents feuilles de la hiérarchie correspondent à des objets interactifs PAC élémentaires. Ce sont les partenaires unitaires de l'utilisateur visibles à l'écran.



2.3.4 Evaluation comparative avec notre modèle

A nouveau, au plus haut niveau d'abstraction, on retrouve dans les deux modélisations, les trois composants de base d'une application interactive. D'une part, les composants Application, Dialogue et Présentation dans BIPS. D'autre part, nous avons mis en évidence qu'une application interactive est un objet PAC composé : son abstraction constitue l'application, la présentation définit l'image de l'application, et le contrôle joue le rôle d'intermédiaire entre ces deux parties.

Le composant Application de BIPS et le modèle PAC

Le composant Application correspond à l'abstraction de l'agent PAC du niveau d'abstraction le plus haut, c'est-à-dire au sommet de la hiérarchie PAC. Il est constitué d'un ensemble d'objet d'application (dans notre exemple, les objets Clients, Produits, Commandes). Ces objets de l'application sont manipulables par les objets de dialogue.
Cette partie d'une hiérarchie PAC, qui correspond à l'application d'un système interactif, manipule les concepts du domaine et réalise les fonctions propres au

domaine d'application. Son organisation est laissée au choix du concepteur de l'application.

Dans les deux modélisations, on a bien l'indépendance de l'application vis-à-vis de son interface. Cette modularité est favorable à l'ajustement itératif de l'interface et à la réutilisation de composants.

Le composant Présentation de BIPS et le modèle PAC

La particularité du composant Présentation de BIPS, c'est qu'il peut être décomposé en deux niveaux. Le niveau des objets interactifs concrets et celui des objets interactifs abstraits.

Ce dernier permet l'indépendance de l'interface utilisateur avec un toolkit particulier. Cette couche définit un terminal abstrait c-à-d une abstraction du terminal physique. Il assure le changement de formalisme entre les objets du composant dialogue qui s'expriment en termes abstraits et les dispositifs physiques d'entrée/sortie d'un toolkit particulier. Une classe d'objets interactifs abstraits est définie par son interface c-à-d l'ensemble de ses primitives de manipulation et des événements abstraits qu'elle génère.

Des caractéristiques de l'interface d'un type application découle l'identification de classes d'objets interactifs abstraits. Le modèle PAC étant d'application plus générale, il n'offre pas explicitement cette couche d'OI abstraits mais cela pourrait être fait en fonction du type d'application envisagée.

Quant aux objets interactifs concrets du modèle BIPS, ils correspondent aux **agents PAC feuilles** de la hiérarchie c-à-d à des objets élémentaires. Ce sont en effet ces agents du niveau le plus bas de la hiérarchie qui définissent l'image du système. Ils correspondent aux partenaires unitaires de l'utilisateur et sont visibles à l'écran.

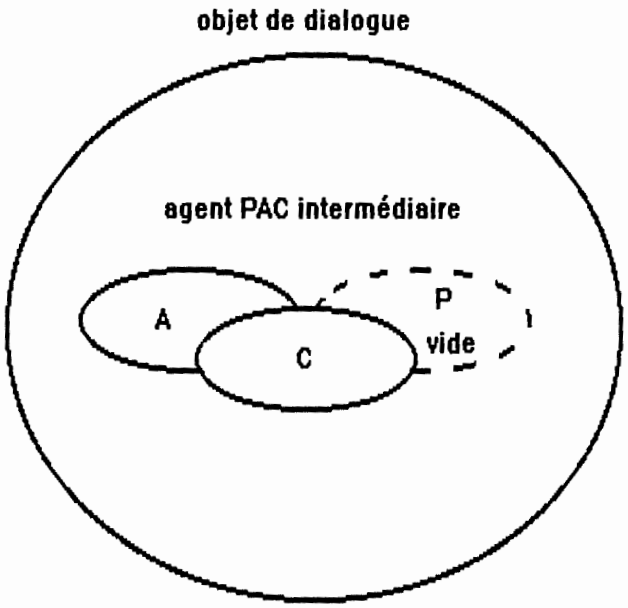
Le composant Dialogue de BIPS et le modèle PAC

Ce composant est constitué d'un ensemble d'objets de dialogue. Les objets de dialogue sont généralement obtenus par abstraction des objets interactifs.

Par rapport à PAC, les objets de dialogue apparaissent comme des agents **intermédiaires** d'une hiérarchie PAC. Un objet de dialogue est généralement obtenu par abstraction de plusieurs objets interactifs, de même qu'un agent PAC intermédiaire correspond à un objet interactif composé. L'objet composé est une élaboration au-dessus d'agents composants et définit ainsi un nouveau niveau d'abstraction.

De plus, un objet de dialogue est un objet abstrait et non visible à l'utilisateur, tout comme un objet PAC intermédiaire, qui a une présentation généralement vide et est de nature plus abstraite que les agents feuilles.

Ensuite, au sein de l'architecture BIPS, le contrôle a été dispersé à travers l'ensemble des objets du composant dialogue. On a ainsi un éclatement de la dynamique d'enchaînement des fonctions de l'application et des fonctions de dialogue. Dans une application interactive PAC, la synchronisation est répartie à travers l'ensemble des contrôles des agents de la hiérarchie.
Relevons cependant une différence majeure entre un objet de dialogue et un agent PAC. Un objet de dialogue du modèle BIPS peut directement invoquer les services des objets de l'application, c'est-à-dire qu'il peut appeler directement une fonction de l'application.
Dans PAC, c'est le Contrôle du niveau le plus haut est chargé de mettre en correspondance les fonctions de l'application avec les abstractions et les présentations des agents de la hiérarchie. Un agent PAC, pour communiquer avec l'application, doit donc passer par les contrôles de ses agents père, jusqu'au contrôle du niveau le plus haut.



Dans BIPS l'architecture globale d'une application repose sur une hiérarchie d'objets, basée sur la relation "utilise" qui permet l'expression de niveau d'abstraction.
Une hiérarchie PAC peut être vue comme un seul agent PAC composé. En effet la présentation étant réalisée par une hiérarchie d'objets PAC, elle hérite des présentations des objets composants auxquelles s'ajoutent des propriétés intrinsèques. C'est donc par application systématique des techniques de

raffinement et/ou de composition que l'on structure un système interactif en une hiérarchie PAC.

J.Coutaz énonce un certain nombre de **règles** empiriques pour gérer la cohérence et la coordination entre les objets de la hiérarchie. De plus, de cet ensemble de règles, un certain nombre d'**Agents PAC généraux** ont été identifiés.
Citons par exemple, la règle disant que dès que deux agents de même niveau dans la hiérarchie ont une relation, alors un agent père doit être créé pour maintenir cette relation et faciliter la réutilisation des deux agents. Une autre règle consiste à dire qu'un agent devrait être dédié à la gestion des erreurs.
Certaines règles relèvent du software engineering (réutilisation, modularité) et d'autres traduisent des recommandations issues de la psychologie cognitive et de l'ergonomie (vue multiple, cohérence, dialogue multi-fils).

Communication entre l'application et son interface

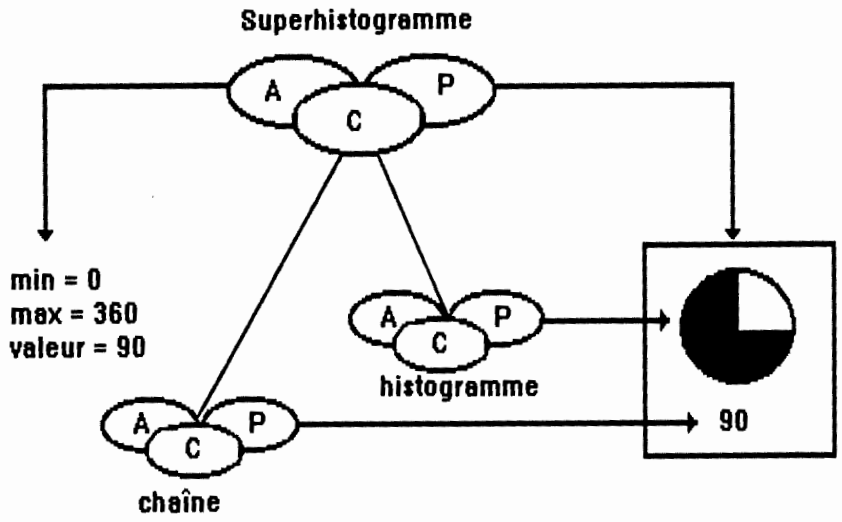
PAC :

Le Contrôle du niveau le plus haut est chargé de mettre en correspondance les concepts de l'application avec les abstractions et les présentations des agents de la hiérarchie.

Ainsi, lorsque l'application modifie un concept, le contrôle de l'agent du niveau le plus élevé exprime le changement dans les termes des abstractions des objets qui lui sont associés. Les contrôles de ces objets prennent le relais jusqu'à ce que l'effet se traduise dans les présentations et abstractions des objets élémentaires.

Dans le sens inverse, les actions de l'utilisateur sont interprétées par les présentations du niveau le plus bas, reflétées dans les abstractions du niveau le plus bas puis colportées en remontant la hiérarchie via les contrôles d'agents intermédiaires, jusqu'à ce que des abstractions d'objets interactifs soient en liaison directe avec le contrôle du niveau le plus haut et donc avec l'application [Coutaz 90].

Exemple :
Reprenons notre exemple ci-dessus de l'objet composé SuperHistogramme.



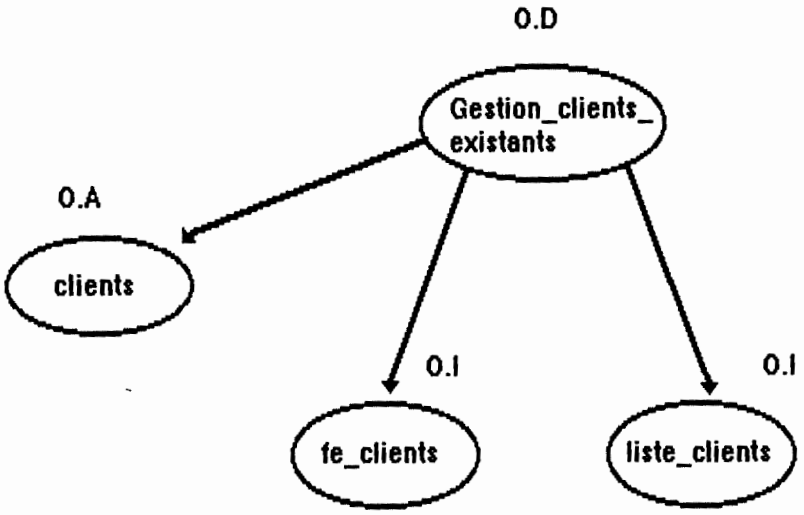
Sa valeur peut être modifiée soit par l'application, soit par l'utilisateur (présentation).
Si la modification provient de l'application, par exemple "valeur" devient 60 au lieu de 90, alors le contrôle du niveau le plus haut traduit la modification dans les termes de chacun des objets élémentaires. Chacun des objets élémentaires "histogramme" et "chaîne" reçoit une notification de changement et les contrôles de ces objets reflètent à leur tour la modification sur leur abstraction et leur présentation.
Si la modification est une conséquence des actions de l'utilisateur, le cheminement des modifications s'effectue depuis l'objet élémentaire transformé jusqu'à l'objet composé qui décide à son tour des effets de bord sur les autres constituants. L'utilisateur peut soit agir directement sur la présentation de l'histogramme élémentaire, soit modifier la valeur présentée par la chaîne. Dans le premier cas, le contrôle de l'histogramme élémentaire effectue le changement de valeur de sa partie abstraction et le signale au contrôle de l'objet composé. Ce dernier interprète le signal reçu et met à jour son abstraction (du niveau le plus haut). Il force ensuite l'objet élémentaire chaîne à se mettre à jour par le mécanisme énoncé précédemment. Dans le deuxième cas, c'est le contrôle de la chaîne qui avertit le contrôle du Superhistogramme de sa modification.

BIPS :

C'est le composant dialogue qui met en correspondance les objets de l'application et les objets interactifs du composant Présentation. Les objets de l'application et ceux de la présentation du modèle BIPS sont utilisés par les objets de dialogue.
Le composant Application de BIPS est dynamique ou réactionnel. Il a la possibilité d'avertir le composant Dialogue (par émission d'événements) de modifications de ses objets susceptibles d'intéresser le composant dialogue et ce

essentiellement pour qu'il puisse mettre à jour la présentation. L'initiative du dialogue ne provient donc pas uniquement des objets interactifs suite aux actions de l'utilisateur sur ceux-ci. Celle-ci peut également provenir des objets de l'application.

Exemple :
Reprenons une partie de notre exemple "enregistrement de commande-client". Ci-dessous la hiérarchie des objets associés à la gestion des clients existants.



Une fenêtre contenant une list-box présente à l'utilisateur la liste des clients existants dans la base de données. Cette liste est susceptible d'être modifiée à tout moment par l'application. Dès lors l'objet de l'application "clients" émet l'événement "ev-client-modifié" à destination de son objet de dialogue père "Gestion_clients_existants".
"ev-client-modifié" est événement de l'interface de l'objet de l'application "clients" qui est généré lors de toute modification apportée à la liste des clients (ajout, suppression) ou lors de toute modification à des clients existants (ex: changement de nom).
L'objet de dialogue "Gestion_clients_existants" déclenche alors une primitive de l'objet clients "pr_obt_lst_client -> CLI" qui renvoie, par itération, la liste des clients existant dans une structure qui contient le nom et le prénom. Avec ces informations, l'objet de dialogue peut mettre à jour la présentation de la list-box (l'objet interactif "liste-clients"). Il utilise pour cela la primitive "pr_lbx_ajout_élem" de l'interface de l'objet interactif "liste-clients". Elle ajoute un élément dans la liste des clients affichée à l'écran.

En conclusion, il est important de remarquer que dans les deux modèles, et par le biais des processus que nous venons d'expliquer, les échanges entre

l'application et l'interface se font au niveau d'abstraction voulu. En particulier, l'application a la possibilité de s'exprimer dans son formalisme, ce qui la rend indépendante de son interface.
De plus, l'indépendance du dialogue est assurée et permet donc de modifier le code de l'interface sans avoir à modifier celui de l'application et inversement.

Modularité et Réutilisation

Les deux modèles sont caractérisés par l'organisation fortement modulaire d'une application interactive et une grande réutilisation possible des composants. Cela étant dû notamment à la modélisation orientée objet sous-jacente. Un objet ou un agent correspond à une unité de traitement autonome et aussi à une unité de modularisation.
Il devient alors aisé de remplacer la présentation d'un concept par d'autres objets interactifs ou de modifier localement l'image d'un objet élémentaire dans les deux modèles.

Dialogues asynchrones et à plusieurs fils d'activité

Un objet du modèle BIPS est une unité de traitement de l'information caractérisée par une mémoire et un processeur d'instructions. L'état d'un objet BIPS est défini par l'ensemble des valeurs des variables qui constituent la mémoire. De même tout objet interactif PAC, élémentaire ou composé présente la même configuration.
Dès lors chaque agent (objet BIPS et objet PAC) mémorise son état local et est en mesure d'être abandonné par l'utilisateur puis d'être à nouveau activé. La manipulation d'un agent est à tout moment interruptible.

2.3.5 Résumé de la comparaison

Tout comme PAC, BIPS est un modèle de type multiagent. Ce modèle structure une application interactive en un ensemble d'agents qui coopèrent dans la conduite de l'interaction. Un agent est un système de traitement et un interlocuteur avec d'autres agents et avec l'utilisateur.

Les deux modèles définissent ainsi une approche orientée objet de la modélisation d'une application interactive. Un objet ou un agent constitue une unité de modularisation. Les systèmes ainsi obtenus sont très modulaires, ce qui encourage la mise au point itérative des interfaces. Ils concentrent leur effort sur la mise en oeuvre d'interface à dialogues multi-fils et asynchrones.

Les deux modèles offrent un cadre de construction systématique applicable à tous les niveaux d'abstraction de l'application interactive.

BIPS utilise la relation logicielle "utilise" pour structurer une application interactive en une hiérarchie d'objets. PAC utilise les techniques de raffinement et/ou de composition pour structurer une application interactive en une hiérarchie d'agents PAC.

2.3.6 Compte rendu de stage : Le projet MITHRA

Je tiens encore à remercier le Professeur J.Coutaz de l'Université de Grenoble ainsi que le Professeur J.L Crowley du laboratoire LIFIA de Grenoble de m'avoir permis d'effectuer un stage de quatre mois sous leur conduite.

Ce stage a été pour moi l'occasion de travailler à la réalisation d'une interface réelle, de taille relativement importante et nécessitant un travail d'équipe.
Il m'a permis d'approfondir le modèle d'architecture PAC développé par J.Coutaz et d'implémenter une architecture décrite à l'aide de ce modèle.
Ce stage a donc été aussi l'occasion de m'initier aux techniques de programmation sous l'environnement de multi-fenêtrage X-Windows et d'utiliser les services d'une boîte à outils pour la construction du dialogue (Toolkit Osf/Motif). Le système d'exploitation était le système Unix et la machine était station de travail Sun4.

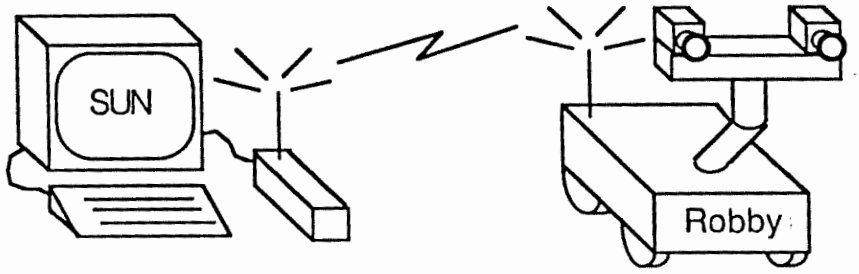
2.3.6.1 Description du projet MITHRA

MITHRA est un projet européen EUREKA EU110 qui regroupe différents partenaires industriels et universitaires. Le but de ce projet est de développer une famille de robots mobiles de télésurveilllance et de première intervention comportant un centre décisionnel embarqué. Ce robot évoluera dans un univers partiellement structuré et partiellement connu dans lequel il effectuera des missions programmées.

Une partie de ce projet consiste à développer une Interface Homme-Machine (IHM). Cette IHM doit permettre à un utilisateur d'accomplir trois tâches principales :

1) Spécifier par manipulation directe l'environnement dans lequel le robot devra évoluer.
2) Spécifier par manipulation directe des missions de surveillance.
3) Exécuter et suivre une mission de surveillance.

C'est à la réalisation de la première partie que je me suis attaché durant mon stage. Le projet a commencé il y a deux ans et comporte environ 10 personnes dont quatre s'occupent de la réalisation de l'IHM.
La figure ci-dessous illustre la configuration matérielle de l'ensemble du système.



Station de travail Robot mobile

Les utilisateurs

Il y a deux catégories d'utilisateurs : les responsables de la spécification de l'environnement et les surveillants.

Les responsables de la description de l'environnement du robot doivent avoir les aptitudes suivantes:
- une très bonne connaissance de l'environnement à décrire
- une bonne connaissance de la spécification d'environnement du logiciel c'est-à-dire qu'ils doivent être capables de décrire cet environnement en termes d'objets du logiciel (voir plus loin)

Les surveillants sont chargés de décrire la mission de surveillance et de contrôler son exécution en temps réel. Ils doivent posséder les aptitudes suivantes :
- une bonne connaissance de la spécification de mission du logiciel
- une bonne connaissance des caractéristiques et des fonctions du robot.

Les clients potentiels

Les clients susceptibles d'être intéressés par le logiciel MITHRA incluent :
-les partenaires du projet dont le domaine d'activité est la surveillance
(les sociétés CERBERUS de Mannedorf et ELKRON de Turin)
- toute autre société ou organisme de surveillance
- l'armée.

2.3.6.2 Spécification d'un environnement

La partie du projet auquel j'ai participé, concerne la réalisation d'une IHM devant permettre de spécifier l'environnement dans lequel le robot évolue. Cette interface utilise comme style d'interaction, la manipulation directe.
A l'aide de sa souris, d'un clavier et d'un écran graphique l'utilisateur crée à l'écran, un certain nombre d'objets composants l'environnement du robot. La liste des objets de l'environnement qui sont offerts par l'IHM est la suivante :

- des murs
- des portes
- des objets circulaires (poubelles, tables rondes ...)
- des objets rectangulaires (tables, chaises...)
- des balises
- des amers
- des emplacements
- des routes
- des lieux

(Pour la représentation graphique de ces objets, voir le point 2.3.6.4 les écrans du logiciel).

Chaque objet doit pouvoir être modifié, déplacé ou détruit à chaque instant.
L'environnement décrit doit pouvoir être sauvegardé sous la forme d'un fichier ASCII.
A chaque objet est associé un formulaire, contenant une description détaillée de ses attributs (voir les écrans du logiciel). Par exemple, le formulaire de l'objet mur reprend sa longueur, son épaisseur, ses coordonnées dans le plan, sa nature (bois, métal, pierre, béton...).
Un emplacement est un endroit par lequel le doit passer. Une route désigne le chemin que le robot doit emprunter pour aller d'un emplacement à un autre. Une route a comme attributs, un nom, les noms des emplacements qu'elle relie, la vitesse à laquelle le robot doit circuler sur la route (minimum, maximum, silencieuse, nominale). Un lieu représente un ensemble connexe de routes et d'emplacements et n'a qu'un nom.

Pour créer un objet de l'environnement, l'utilisateur sélectionne à l'aide de la souris un objet dans la palette d'objets disponibles (côté gauche). Ensuite, il déplace sa souris pour déterminer la longueur et l'emplacement de l'objet dans le plan.

Par exemple, pour créer un mur, l'utilisateur clique dans la palette d'objets, sur l'objet mur. Puis il déplace le curseur avec sa souris dans l'aire de dessin.

Il clique une première fois à l'emplacement désignant le début du mur. Puis à l'aide de sa souris il étire le mur jusqu'à la longueur désirée. Il clique alors une seconde pour déterminer le deuxième point de l'emplacement final du mur. Il peut à ce moment où à m'importe quel moment plus tard dans sa session, modifier les attributs par défaut d'un objet. Pour cela il clique à un endroit déterminé au milieu de l'objet, ce qui fait apparaître le formulaire associé à l'objet.

2.3.6.3 Architecture PAC

Voici ci-dessous l'architecture de la partie spécification d'un environnement de l'IHM Mithra.

L'abstraction du niveau le plus haut rassemble, le gestionnaire de base de connaissance, le contrôleur de communication avec le robot et un superviseur intelligent chargé de planifier les missions et de contrôler leurs exécutions.

La hiérarchie de l'architecture de la partie présentation comprend les objets suivants.

MUL : Multi-fenêtrage
Cet objet a pour rôle de gérer le multifenêtrage au niveau de l'activité courante. MUL maintient le lien pour chaque entité manipulée avec la liste des objets de présentation (SI) courante. C'est-à-dire qu'il conserve l'association entre l'identificateur interface des entités de l'activité courante, définie au niveau du Supercontroleur et les SI associés. Il gère donc la correspondance à la descente comme à la montée dans la hiérarchie, entre l'identificateur interface et les identificateurs des SI. Il gère aussi la répercussion des modifications entre les différentes fenêtres d'une même entité.

SI : Superviseur d'interface
Son rôle est de gérer tout ce qui concerne la fenêtre associée. Le superviseur d'interface fait principalement l'analyse syntaxique des commandes utilisateur et gère l'aiguillage de ces commandes vers les différents objets. Il assure donc la gestion de la barre de menu, la gestion de la visibilité des types d'objets, la gestion de la création des objets.

Travail
Sa présentation est constituée de la fenêtre de travail. C'est dans cet espace que sont dessinés, manipulés et créés les différents objets d'un environnement. Son rôle est d'effectuer l'analyse syntaxique d'une commande concernant la manipulation d'un objet.

Palette

L'objet Palette représente la palette d'objets (à gauche de l'écran) pour l'activité de spécification d'environnement. Son rôle est de récupérer les diverses commandes (actions) de l'utilisateur, effectuées sur la palette pour sélectionner un type d'objet et de les transmettre au SI.

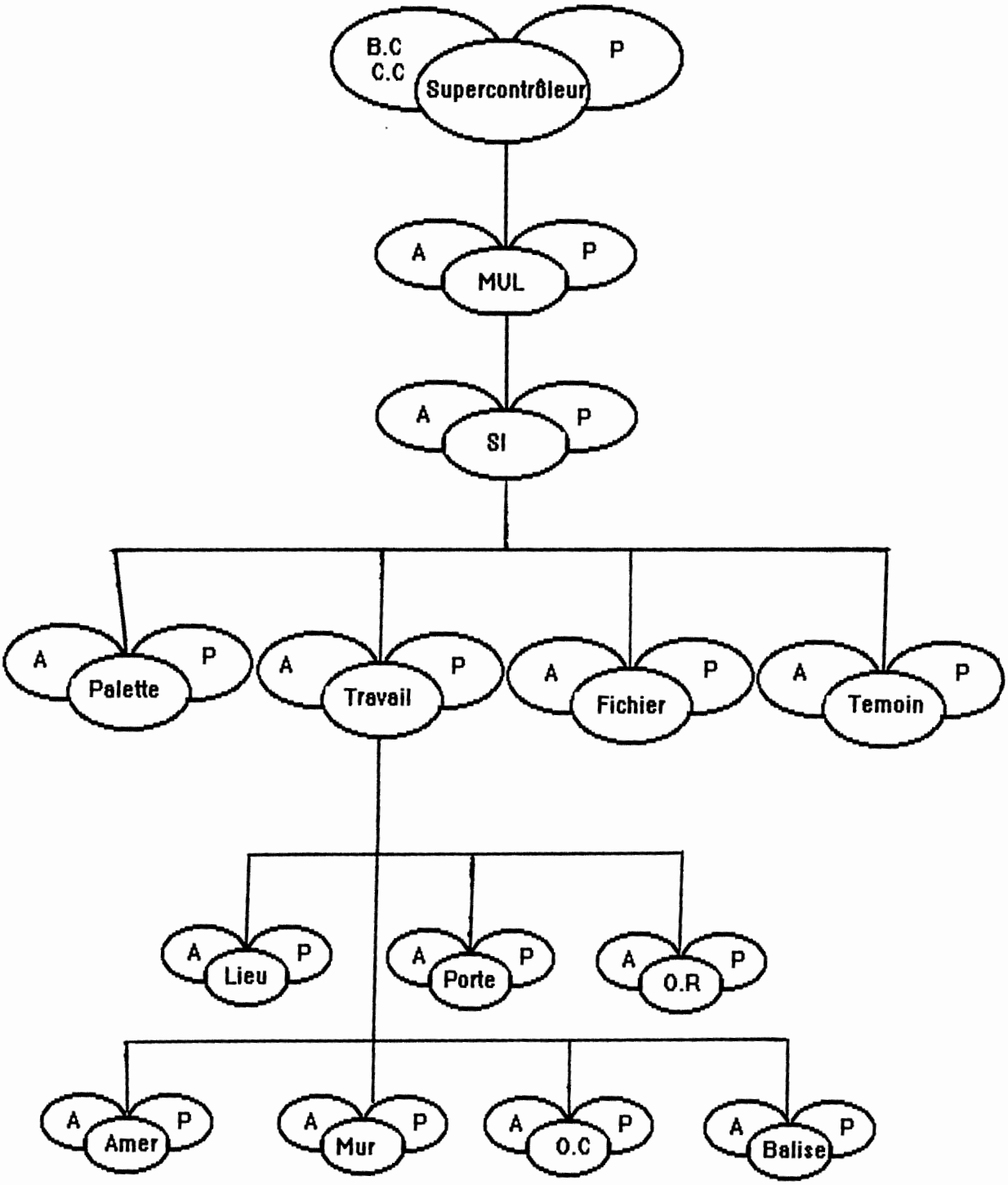
Fichier

L'objet Fichier gère la présentation des différents fichiers auxquels peut accéder l'utilisateur et gère les commandes d'accès demandées. Cet objet assure donc la gestion de tout les accès disque.

Témoin

L'objet témoin est un indicateur de la validité de la spécification actuelle. Le témoin montre trois états possibles. Il a le visage souriant si la solution est sémantiquement correcte. Il est neutre lorsque la spécification est correcte sans toutefois se présenter dans les meilleures conditions : un indicateur précise alors le nombre d'erreurs visibles et cachées. Le visage du témoin est triste lorsque la spécification est incorrecte et présente des erreurs sémantiques : un indicateur précise le nombre d'erreurs visibles et cachées.

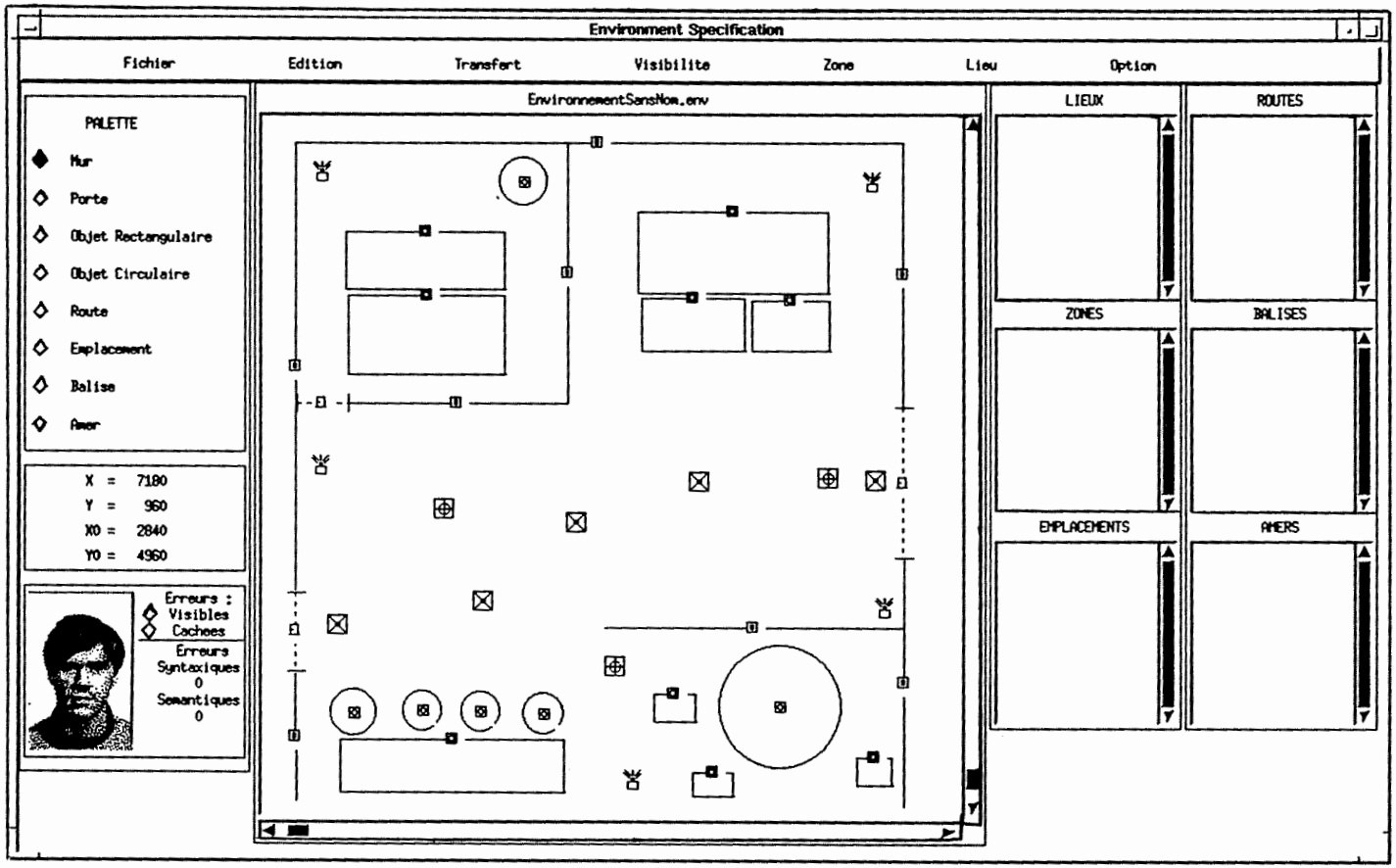
Par exemple, une route qui croise un mur est une erreur de spécification. Une porte qui ne relie aucun mur, une route doit toujours relier au moins deux emplacements.

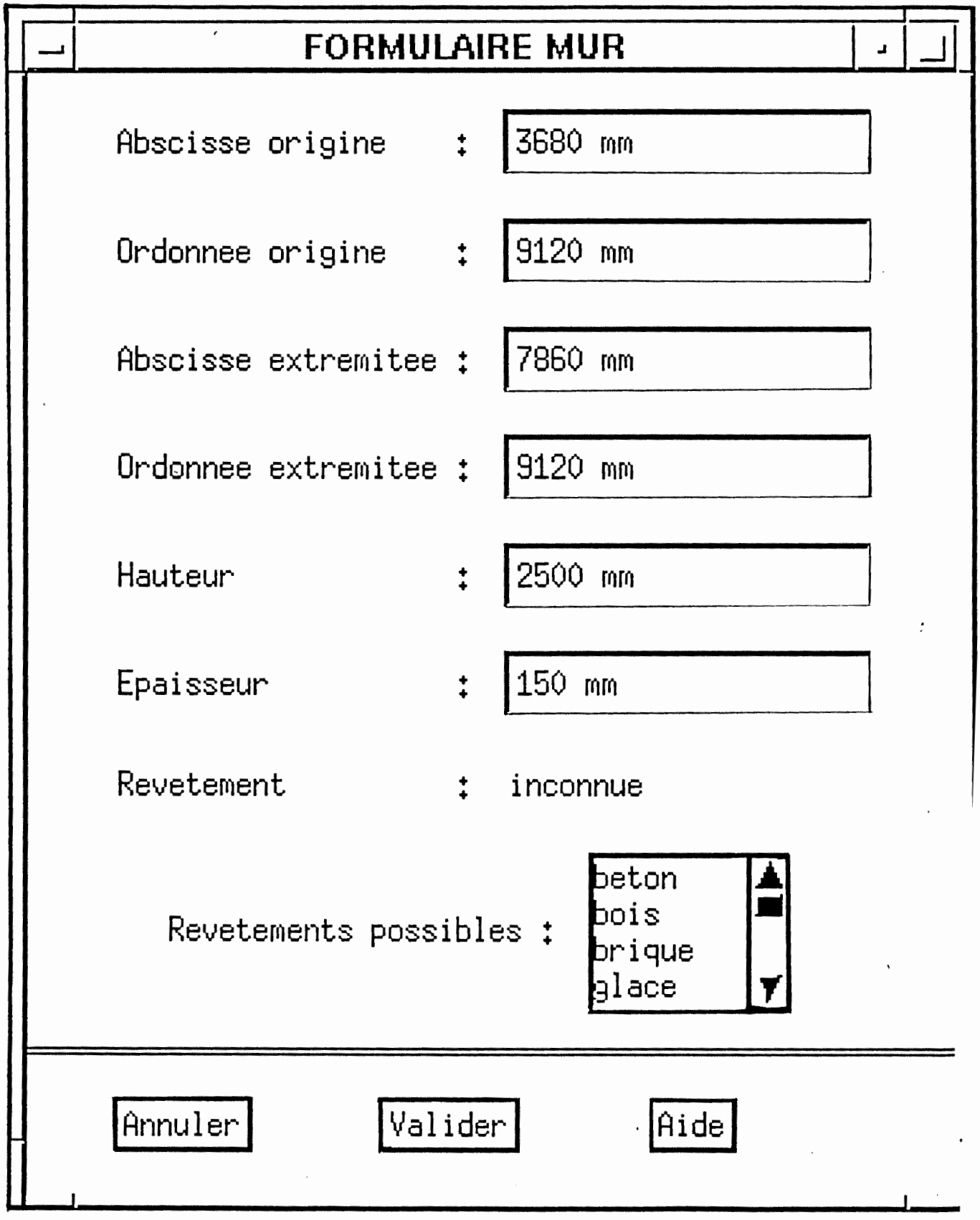

B.C
C.C
Supercontrôleur
P
A
MUL
P
A
SI
P
A
Palette
P
A
Travail
P
A
Fichier
P
A
Temoin
P
A
Lieu
P
A
Porte
P
A
O.R
P
A
Amer
P
A
Mur
P
A
O.C
P
A
Balise
P

2.3.6.4 Les écrans de l'IHM

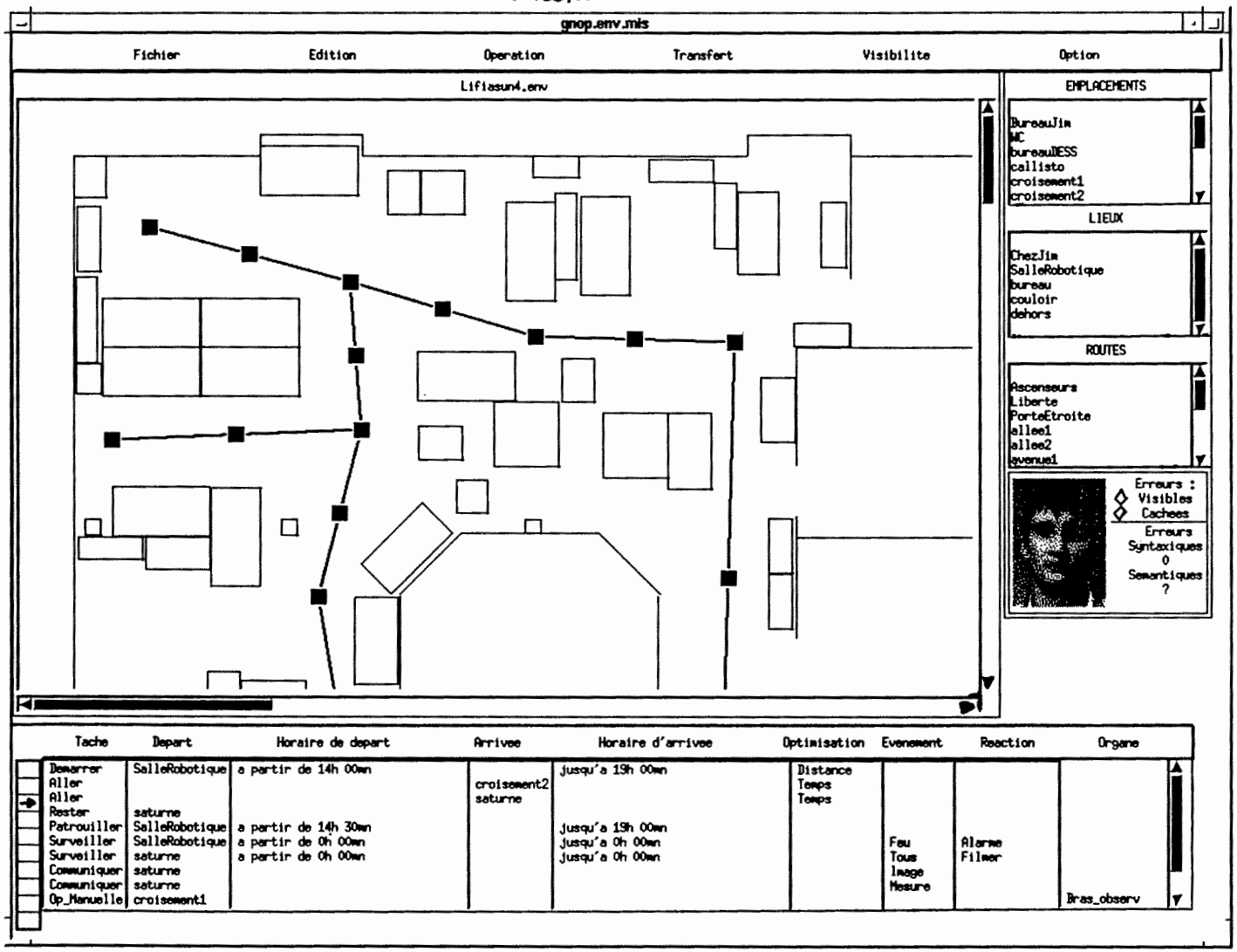
La partie spécification de l'environnement du robot du logiciel Mithra, partie sur laquelle j'ai travaillé, a été réalisé à l'aide de l'environnement X-Windows [Young 89] et de la boîte à outils Osf/Motif [Motif 90]. Le logiciel a été développé sur station de travail SUN4. Le système d'exploitation était le système UNIX.

Respectivement dans l'ordre, on présente :
Une image de l'interface associée à la partie spécification d'un environnement.
Une boîte de dialogue représentant le formulaire associé à un objet mur.
Une image de l'interface associée à la partie spécification d'une mission..


Environment Specification
Fichier
Edition
Transfert
Visibilite
Zone
Lieu
Option
PALETTE
Mur
Porte
Objet Rectangulaire
Objet Circulaire
Route
Emplacement
Balise
Amer
X = 7180
Y = 960
XO = 2840
YO = 4960
Erreurs :
Visibles
Cachees
Erreurs
Syntaxiques
0
Semantiques
0
EnvironnementSansNom.env
LIEUX
ROUTES
ZONES
BALISES
EMPLACEMENTS
AMERS


FORMULAIRE MUR
Abscisse origine : 3680 mm
Ordonnee origine : 9120 mm
Abscisse extremitee : 7860 mm
Ordonnee extremitee : 9120 mm
Hauteur : 2500 mm
Epaisseur : 150 mm
Revetement : inconnue
Revetements possibles :
beton
bois
brique
glace
Annuler
Valider
Aide

MISSION


gnop.env.mis
Fichier
Edition
Operation
Transfert
Visibilite
Option
Lifiasun4.env
EMPLACEMENTS
BureauJim
WC
bureauDESS
callisto
croisement1
croisement2
LIEUX
ChezJim
SalleRobotique
bureau
couloir
dehors
ROUTES
Ascenseurs
Liberte
PorteEtroite
allee1
allee2
avenue1
Erreurs :
Visibles
Cachees
Erreurs
Syntaxiques
0
Semantiques
?
Tache
Depart
Horaire de depart
Arrivee
Horaire d'arrivee
Optimisation
Evenement
Reaction
Organe
Demarrer
Aller
Aller
Rester
Patrouiller
Surveiller
Surveiller
Communiquer
Communiquer
Op_Manuelle
SalleRobotique
saturne
SalleRobotique
SalleRobotique
saturne
saturne
saturne
croisement1
a partir de 14h 00mn
a partir de 14h 30mn
a partir de 0h 00mn
a partir de 0h 00mn
croisement2
saturne
jusqu'a 19h 00mn
jusqu'a 19h 00mn
jusqu'a 0h 00mn
jusqu'a 0h 00mn
Distance
Temps
Temps
Feu
Tous
Image
Mesure
Alarme
Filmer
Bras_observ

2.4 Le modèle MVC de Smalltalk

2.4.1 Présentation du modèle MVC de Smalltalk

Le modèle MVC (Model, View, Controller) présenté dans [Krasner 88], [Masini 89], [Cunningham 85] de Smalltalk-80 [Goldberg 84], est également un modèle de type multiagent.

Smalltalk est un langage et un environnement de programmation orienté objet. Cet environnement fournit un ensemble d'outils pour le développement de programmes à orientation graphique et d'interface utilisateur de haut niveau.

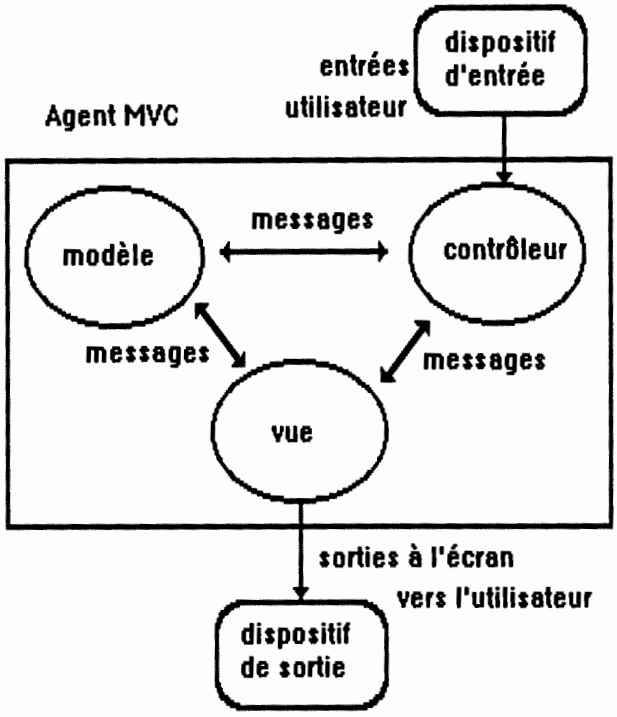
Tout comme PAC et BIPS, il structure une interface en un ensemble d'agents qui travaillent en parallèle et qui coopèrent dans la conduite de l'interaction.
Un agent MVC est composé de trois facettes : un Modèle, une Vue et un Contrôleur :

- Le modèle (model) est spécifique au domaine d'application. Il définit une représentation abstraite de la compétence de l'agent. Le modèle représente les données et les opérations qui peuvent être exécutées sur ces données.

- La vue (view) gère tout ce qui est présentation de sortie (affichage, graphisme). Elle définit la présentation du modèle à l'utilisateur. C'est le mécanisme utilisé pour afficher les données d'un modèle dans des fenêtres, à l'écran.

- Le contrôleur (controller) interprète les entrées en provenance de l'utilisateur et coordonne les modèles et vues qui lui sont associées par le biais des entrées en provenance de l'utilisateur.

C'est cette partie qui a la responsabilité d'interpréter les commandes de l'utilisateur en fonction de la vue qu'il contrôle et de déclencher une réponse correspondante de la part de la vue ou du modèle selon les cas.



La construction d'une application interactive

La construction du programme peut se faire en trois étapes correspondant chacune au développement d'un des composants énoncés ci-dessus.

La conception du modèle

Un modèle représente les données et les opérations qui peuvent être exécutées sur ces données. Un modèle doit coopérer avec les vues et les contrôleurs en vue de fournir les données devant être affichées à l'écran.

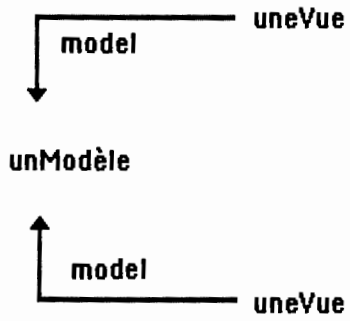
Le modèle sera implémenté de manière à ne détenir aucune information concernant l'interface-utilisateur, ceci afin de rester le plus indépendant possibles de ses vues et contrôleurs.

Un modèle constitue un **objet** au sens de la programmation orientée objet. En effet dans Smalltalk, toute entité est un objet et tout objet est une **instance d'une classe**. Toutes les classes du système sont organisées en hiérarchie dont la classe racine est la classe Object.

La conception de la vue

La vue constitue le mécanisme qui permet de présenter à l'écran l'information fournie par un modèle. Il s'agit d'un mécanisme intermédiaire entre le modèle et les fenêtres. En Smalltalk absolument tout est un objet, une vue est donc un objet et doit dès lors être une instance d'une classe. Suivant son type, une vue est une instance d'une des nombreuses classes décrivant des vues et s'insérant dans la hiérarchie du système. Ces vues sont similaires aux objets interactifs d'un toolkit.
On y trouve par exemple les classes suivantes : la classe StringHolderView spécialisée dans l'édition des textes, la classe FormHolderVieuw spécialisée dans l'édition graphique, la classe SelectionInListVieuw qui correspond à l'objet interactif List-Box permettant de lister une liste d'items et d'en sélectionner un. Une vingtaine de types différents de vues sont prédéfinies dans une bibliothèque initiale.
Toutes ces classes sont des sous-classes de la classe VIEW. Cette classe contient donc la déclaration des variables et des méthodes communes à toutes les vues.

Chaque instance d'une vue connaît le modèle qu'elle présente. Si les données du modèle sont modifiées, la vue répercute ces modifications à l'écran. La vue est donc dépendante du modèle. Cette relation de dépendance est implémentée grâce à la variable d'instance **model**, définie dans la classe View. Cette variable permet de lier chaque vue à son modèle, c'est une référence explicite. Par contre les modèles ne peuvent se référer directement à leurs propres vues, l'idée étant de garder les modèles aussi simples que possible et ne sachant pas comment ils sont représentés dans les fenêtres.
De même, la variable d'instance **controller**, définie dans la classe View, définit une référence explicite au contrôleur de la vue.



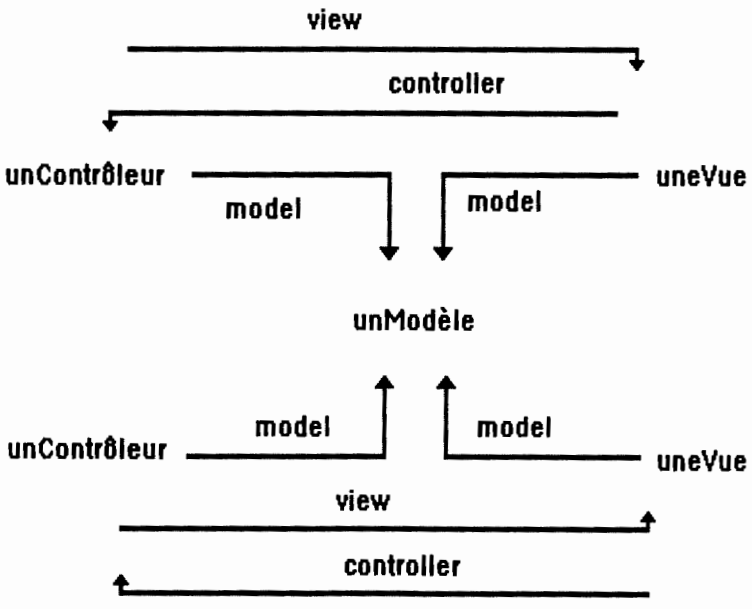
La conception du contrôle

A chaque vue est associé un contrôleur. Il se charge de gérer les communications entre l'utilisateur et les vues et/ou modèles. Il doit détecter les commandes et données introduites par l'utilisateur, les interpréter et déclencher une réponse adéquate. Ces données et commandes comprennent les mouvements de la souris, les pressions sur les boutons de la souris et du clavier.

Comme les vues et les modèles, les contrôles sont des objets Smalltalk et sont donc des instances de classes qui les décrivent. Il existe un contrôleur particulier associé à chaque type de fenêtre (vue). Ainsi une fenêtre principale, instance de la classe StandardSystemVieuw, est associée à un contrôleur de la classe StandardSystemController qui détermine la fenêtre courante. Une vue instance de la classe StringHolderVieuw spécialisé dans l'édition de texte, va de paire un contrôleur instance de la classe StringHolderController qui scrute l'arrivée des caractères en provenance du clavier et gère leur affichage dans la fenêtre. Le contrôleur MouseMenuController gère la souris.

Toutes ces classes prédéfinies sont des sous-classes de la classe Controller. C'est dans cette classe que sont définies les variables et les méthodes communes à tous les contrôleurs.
Les variables d'instance **model** et **view,** définies dans la classe Controller, contiennent les références explicites au modèle et à la vue associés au contrôleur.

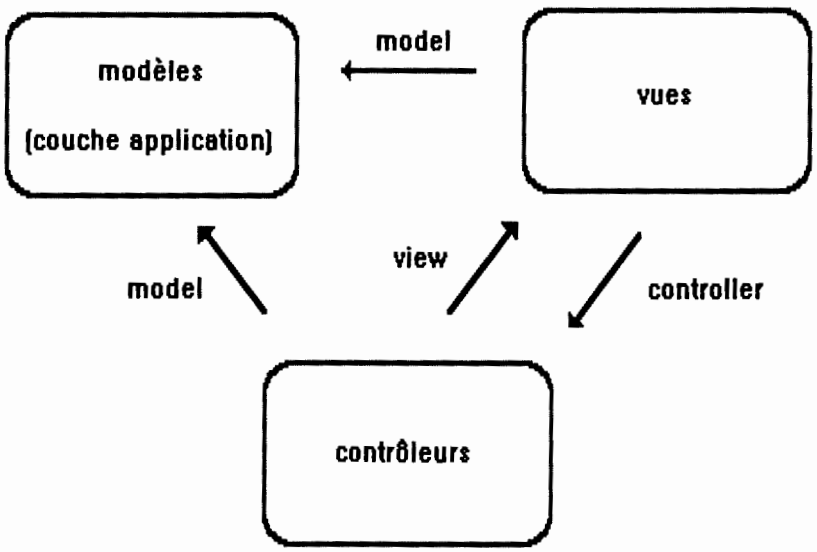
A un modèle peut être associé plusieurs vues et à chaque vue est associée un contrôleur.



Arrivée d'un événement utilisateur.
Prise en compte de l'événement par le contrôleur.
Ce dernier envoie un message au modèle pour le prévenir. Le modèle reçoit le message et effectue le traitement correspondant.
Suite au traitement effectué, le modèle envoie un message vers les vues et les contrôleurs qui lui sont associés.
Prise en compte de ces messages par les vues et les contrôleurs.
Les vues envoient alors un message au modèle pour connaître son nouvel état et mettre à jour sa présentation.
Les contrôleurs envoient aussi un message au modèle pour connaître son nouvel état et mettre à jour leurs méthodes d'interaction.
Lorsqu'un événement utilisateur arrive au contrôleur et qu'il n'implique pas l'intervention du modèle, alors un message est envoyé du contrôleur à la vue qui modifie son affichage.

2.4.2 Evaluation comparative avec notre modèle

L'architecture MVC d'une application interactive peut être modélisée sous la forme de trois composants. L'ensemble des modèles constitue le composant modèle, lequel représente l'application. Les deux autres composants sont constitués de l'ensemble des vues et des contrôles qui constituent la présentation de l'application.
On ne retrouve donc pas un composant destiné au contrôle du dialogue comme dans Seeheim ou BIPS.



Dans BIPS, l'application est regroupée dans un composant, le composant Application. Il est modélisé sous la forme d'un ensemble d'objet application. Dans MVC, l'application est dispersée dans les parties modèle des agents. Le développement de l'application et de son interface sont ainsi liés. On n'a pas comme dans BIPS, l'indépendance entre ces deux composants et donc l'indépendance du dialogue.

BIPS et MVC offre tous les deux une approche orientée objet de la modélisation d'une application. On y retrouve les notions de classes, sous-classes, instances de classes et d'héritage. Cependant BIPS se veut libre des contraintes pratiques de l'outil de réalisation. MVC est au contraire très lié aux techniques de la programmation par objet et du langage Smalltalk-80.

Au niveau de la présentation les deux modèles offrent un certain nombres de classes prédéfinies d'objets de présentation. Dans MVC ce sont des vues et leurs contrôleurs et dans BIPS ce sont les objets interactifs. Cette notion de classe qui définit un type particulier de vue et de classe de contrôleur qui lui est associée correspond à une classe d'objet interactif de notre modèle. Mais MVC distingue le traitement des entrées et sorties d'un agent dans deux types d'objets différents. Cette distinction n'est pas présente dans BIPS. Le comportement d'un objet interactif en entrée comme en sortie est encapsuler dans l'objet et ne fait pas lieu d'une distinction.

Le composant Dialogue du modèle BIPS n'a pas son équivalent dans MVC. Dans BIPS, le contrôle de l'interaction et la gestion de le dialogue est dispersé à travers l'ensemble des objets de dialogue. Dans MVC, le contrôle semble être réparti entre les trois parties d'un agent. C'est par le biais de l'envoi de messages entre les vues, les contrôles et les modèles que s'effectue la gestion du dialogue. Reprenons la classification du modèle PAC en ce qui concerne les différentes fonctions que doit assumer la partie contrôle d'une application interactive et voyons ce qu'il en est dans MVC :

- le rôle de maintien de la cohérence entre la présentation d'un concept et sa partie abstraite est assuré dans MVC par un mécanisme de diffusion de messages entre les Vues et Contrôleurs à chaque fois que le modèle change.

- le rôle d'arbitre et de coordinateur entre agents est rempli dans le modèle MVC par le contrôleur.

- le rôle de traducteur ou changement de formalismes par contre n'est pas explicité dans la modélisation. De part la définition des agents et leurs fonctionnalités, aucune de trois parties du modèle MVC ne peut remplir cette fonction.

Dès lors sans lieu explicite de synchronisation ou d'arbitrage, le concepteur du système n'a pas de cadre de référence pour l'expression de la coopération entre les agents de l'interaction.

Enfin, il reste une différence importante entre notre modèle et MVC de Smalltalk. Notre modèle est basé sur un protocole communication événementiel et donc asynchrone. Avec MVC, les objets communiquent par le biais d'envois de messages synchrones. Quand un objet MVC envoie un message, il suspend son exécution et transfert le contrôle à l'objet récepteur du message, lequel renvoie le contrôle à l'objet émetteur après avoir effectué son traitement. Avec BIPS, rien n'empêche l'objet émetteur d'être actif après émission d'un événement.

2.4.3 Résumé de la comparaison

Tout comme BIPS et PAC, le modèle MVC est un modèle de type multiagent. Il définit une approche orientée objet de la modélisation d'une application interactive. Il autorise la prise en compte d'interface à manipulation directe et de dialogues multi-fils et asynchrones.

BIPS se veut libre des contraintes pratiques de l'outil de réalisation. MVC est au contraire très lié aux techniques de la programmation par objet et du langage Smalltalk-80. MVC reste dès lors difficile à maîtriser et à mettre en oeuvre, nécessitant une connaissance approfondie de l'environnement Smalltalk et des classes d'objets de ses librairies. Le niveau d'abstraction de la modélisation est de bas niveau par rapport à des modèles comme PAC ou BIPS, lesquels permettent de travailler au niveau d'abstraction voulu.

La comparaison de BIPS avec le modèle MVC permet de montrer l'importance d'avoir l'application en un seul composant et non distribuée dans les agents comme dans MVC. L'application peut dès lors être développée indépendamment de son interface, ce qui assure l'indépendance du dialogue.

En distinguant les traitements des entrées de ceux des sorties, on peut modifier la présentation en sortie d'un agent indépendamment de son style d'interaction en entrée, mais cette distinction suggère de diviser le contrôle de l'application en deux parties : contrôle des entrées et contrôle des sorties. Dans BIPS, le comportement en entrée comme en sortie est encapsuler dans un seul objet.

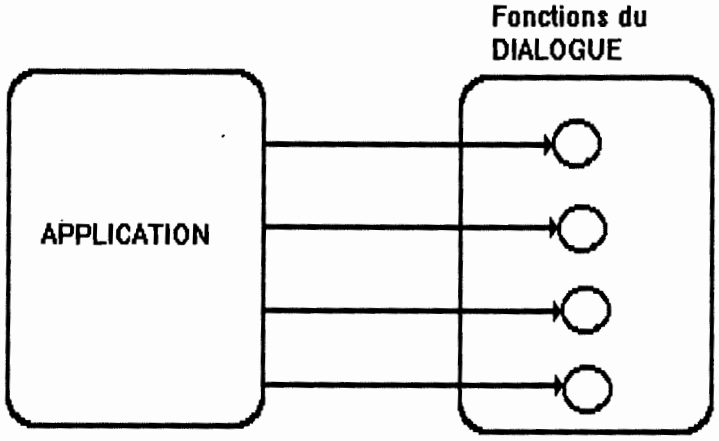
Enfin nous avons aussi signalé que la communication entre les objets de MVC se fait par le biais d'émission de messages synchrones alors que BIPS utilise un protocole de communication événementielle asynchrone entre les objets.

Conclusion

Convenons d'appeler contrôle, la gestion du déroulement d'une application interactive. Dès lors, la localisation du contrôle précise ainsi le centre d'arbitrage du flux des informations. C'est le contrôle qui accomplit le séquencement et la synchronisation des événements durant l'exécution d'une application interactive.

A l'issue de cette première section, nous avons mis en évidence la volonté de séparer les composants de l'application de ceux de l'interface. De nouveaux modèles d'architecture d'une application interactive apparaissent et le placement du contrôle dans ces architectures devient une question de recherche. Si l'on considère que le système est organisé en deux composants, le choix se situe entre l'application et l'interface.

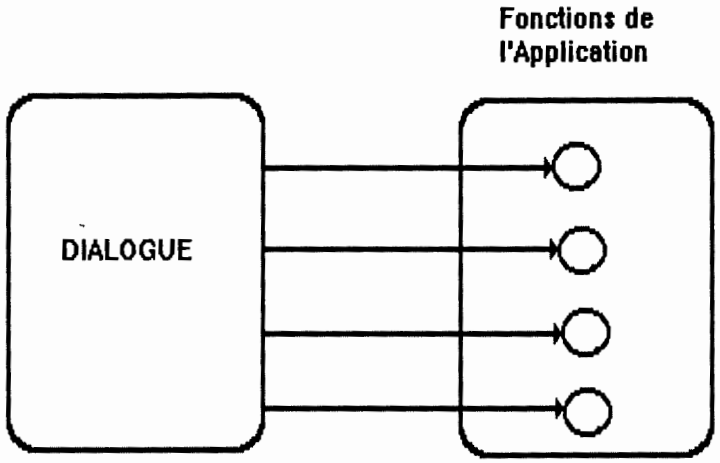
Autrefois, généralement le contrôle résidait dans le composant application. C'était l'application qui invoquait les fonctions du dialogue quand elle avait besoin de données à saisir ou quand elle voulait afficher des informations. Mais cette localisation du contrôle au sein de l'application compromet la séparation modulaire entre fonctions de l'application et celles du dialogue.



C'était donc l'application qui contrôlait les saisies et affichages en vue de collecter les données introduites par l'utilisateur de l'application. Souvent, le code relatif à l'interface et les appels aux fonctions du dialogue étaient dispersés à travers le code de l'application. Cette dispersion n'était pas sans effets négatifs sur la facilité de modification et de maintenance du composant interface. De plus, cela rendait difficile la réalisation d'interfaces de qualités, notamment à cause du fait que ce mélange des codes rendait difficile la mise au point itérative d'une interface.

C'est une vision des choses qui a été fréquemment envisagée à l'institut d'informatique, à travers un cours comme celui de génie logiciel. Un module de gestion des entrées/sorties était utilisé par des modules fonctionnels. Les fonctions du module de gestion des entrées/sorties étaient vues par le composant application comme un ensemble de services disponibles quand il fallait saisir ou afficher des informations à l'écran.

Avec l'avènement du domaine des interfaces homme-machine, la vision des choses s'est détournée au profit d'un contrôle localisé au sein du composant interface. C'est l'interface qui chargée de l'exécution d'une application interactive, de la saisie et affichage des informations et qui a la responsabilité d'appeler les fonctions de l'application en réponse aux actions de l'utilisateur. L'application peut se voir comme "un serveur sémantique".



Cette localisation du contrôle assure la séparation entre les fonctions de l'application et celles de la présentation et conduit à une meilleure séparation modulaire. Le déroulement de l'application est dirigé par l'utilisateur.

Le modèle d'architecture BIPS, développé par I. Provot et B. Sacré, est un modèle de type multiagent. Un tel type de modèle organise une application interactive selon un ensemble d'agents coopérants.
Le contrôle est localisé au sein des composants. A noter que si le composant application de BIPS n'a pas le contrôle, il a la possibilité d'exprimer (par émission d'événements asynchrones) ses besoins auprès de l'interface. L'interface reflète immédiatement l'état de l'application sous forme d'expressions de sortie mais laisse autant que possible l'utilisateur dirigé l'exécution de l'application.

Tous les éléments qui composent l'architecture adhèrent à un modèle "orientés objets". Trois classes d'objets ont été identifiées qui correspondent aux trois composants de l'architecture :

- les objets de l'application
- les objets de dialogue
- les objets interactifs

BIPS se caractérise par une organisation fortement modulaire et par la séparation entre l'application et la présentation (indépendance du dialogue). Un tel modèle encourage la mise au point itérative de la partie interface et convient particulièrement bien aux dialogues asynchrones et aux dialogues plusieurs fils d'activité.

Les objets du composants dialogue sont structurés hiérarchiquement en niveaux d'abstraction à l'aide de la relation logicielle "utilise". Les objets feuilles de cette hiérarchie correspondent aux objets interactifs et aux objets de l'application.
La structure hiérarchique des objets de dialogue suggère 2 méthodes possibles de construction du composant dialogue:

Top-down:

La démarche top-down consiste à trouver les objets de dialogue par **affinements successifs** en partant des objets du dialogue les plus **abstraits**, en connaissant ou non les objets de l'application, et en identifiant parallèlement les objets interactifs utilisés par le composant dialogue.

Cette démarche est selon nous excessivement difficile à réaliser compte tenu de l'effort d'abstraction qu'elle nécessite pour la construction des objets de dialogue. Elle peut toutefois s'avérer intéressante si l'on dispose des objets de l'application, autrement dit si l'on dispose d'une bonne connaissance des fonctionnalités de l'application.

Bottom-up:

La démarche bottom-up consiste tout d'abord à identifier les objets interactifs de l'interface, en connaissant ou non les objets de l'application, pour ensuite déduire les objets de dialogue par abstractions successives et identifier, en parallèle, les objets de l'application s'ils n'étaient pas connus au départ, ou à affiner et enrichir leurs primitives si ces objets étaient disponibles au départ.

Cette démarche présente un double intérêt:

- On conçoit le dialogue de l'application interactive à partir de quelque chose de concret, en l'occurrence les objets interactifs qui constituent la partie présentation de l'interface.

- La définition de la partie présentation comme première étape dans la conception d'une application correspond à la notion de maquettage d'écran utilisée lors de la conception d'une application comme outil de communication privilégié avec l'utilisateur final lors des étapes initiales de conception d'une application.

La construction des dialogues attachée à la description des objets interactifs avec **simulation** des objets de l'application constitue l'étape suivante naturelle au terme de laquelle nous disposons de l'interface d'une application interactive.

Section 2 Formalismes de spécification du dialogue

Introduction

1. Dialogue

Pour rappel, le dialogue dénote la communication entre l'utilisateur de l'application et cette application. Il détermine la suite des échanges d'informations (saisies et affichages) entre l'utilisateur et l'application.

Ces dernières années, de nombreux efforts de recherche ont été fait pour fournir une technique de spécification adaptée à la description des interfaces.
Toute spécification pose le problème du choix d'un formalisme. Un formalisme se juge à sa puissance d'expression et à son adéquation à l'expression de la classe de problèmes considérés.

" La puissance d'expression d'une notation détermine le domaine représentable c'est-à-dire l'ensemble des interfaces qui peuvent être décrites par cette notation.
L'adéquation d'une notation détermine la facilité de transcription du problème c'est-à-dire l'ensemble des interfaces qui peuvent être facilement décrites par cette notation .

La puissance d'expression s'évalue de façon rigoureuse grâce à la théorie des langages formels.
L'adéquation s'évalue de manière subjective. Il n'y a pas de moyen objectif pour mesurer cette adéquation d'un formalisme à la description de la classe de problèmes considérés" [Coutaz 90 et Green 86].

Cependant, l'adéquation d'un formalisme est largement responsable du choix de la solution finale. Par exemple, si l'analyse conceptuelle suggère un type d'interface alors que le formalisme n'est pas approprié à son expression, il est fort probable que les choix conceptuels seront détournés au profit d'une solution exprimable dans le formalisme considéré mais probablement inadaptée à l'utilisateur. Autrement dit, si la meilleure interface pour une application particulière est très difficile à décrire avec un formalisme donné, il y a de fortes chances que le concepteur de l'interface ne la considère pas et en choisisse une autre.

C'est à ce critère d'adéquation d'un formalisme de spécification que nous allons nous intéresser essentiellement. Nous n'envisageons donc pas de comparer notre langage de règles avec d'autres formalismes du point de vue de la puissance d'expression.

I. Petoud dans [Petoud 90] préconise une conception du dialogue partagée en deux tâches : la présentation et la conversation.

La *présentation* concerne la manière dont les informations sont affichées et présentées visuellement à l'utilisateur via son écran. Elle constitue en quelque sorte la partie statique de l'interface.

La *conversation* concerne l'enchaînement des écrans, des fenêtres, l'ordre des saisies et affichages des données c'est-à-dire la suite des échanges d'informations entre l'utilisateur et l'application interactive. Elle constitue la partie dynamique de l'interface.

L'objet de cette section sera une étude comparative de techniques permettant de spécifier la conversation. Nous n'insisterons pas sur la spécification de la partie présentation d'une interface. Rappelons seulement que dans la modélisation de B. Sacré et I. Provot, elle est réalisée par un ensemble d'objets de présentation de haut niveau appelés *objets interactifs abstraits* .

La suite de cette section s'organise de la manière suivante. Après avoir exposées quelques qualités d'une bonne spécification, nous exposerons le langage de règles développé par B. Sacré et I. Provot [Provot et Sacré 90]. Ensuite, nous passerons en revue les principaux formalismes employés pour la spécification d'interfaces jusqu'à ce jour et qui serviront à situer notre langage de règles.

Nous envisagerons successivement les diagrammes état-transition [Wasserman85] et [Green 86] puis les langages à événements. Parmi ces langages nous en retenu deux. Le premier est l'oeuvre de M.Green [Green 85, 86]. Le second appelé langage événement-réponse a été développé par R.D.Hill [Hill 87].

Une analyse formelle effectuée par Green dans [Green 86], montre que le formalisme des langages à événements a une plus grande puissance descriptive que celui des diagrammes état-transition et ainsi, que ce qui est exprimable à l'aide de ces diagrammes peut aussi l'être à l'aide des langages à événements, mais pas réciproquement.
Après, nous exposerons le formalisme développé par J.K Jacob [Jacob 86] qui présente une vue orientée objet d'une interface à manipulation directe et qui combine la notation événementielle avec celle des diagrammes état-transition pour spécifier le dialogue.

Finalement, nous terminerons par une brève description d'autres techniques de spécification mais que nous n'analyserons pas. Ces techniques sont présentées à titre d'information pour le lecteur. Il s'agit des réseaux de Pétri, des grammaires BNF et des systèmes interactifs de spécification.

Au début de leurs travaux, B.Sacré et I.Provot avaient d'abord envisagé d'utiliser le formalisme des diagrammes état-transition. Par la suite, ils se sont dirigés vers une notation événementielle afin d'éviter certains inconvénients du formalisme des diagrammes état-transition.

Finalement, à la suite à l'étude effectuée à travers ce mémoire, ils envisagent de modifier encore leur formalisme de spécification pour en arriver à un formalisme proche de celui de Jacob, combinant le formalisme à événement avec celui des diagrammes état-transition.

2. Les qualités d'une bonne spécification

Jacob, dans [Jacob 83], définit quelques critères essentiels auxquels un formalisme de spécification doit répondre.
Nous avons retenu les critères suivants :

. La spécification doit être facile à comprendre, à lire et plus simple à produire que les logiciels qui réalisent l'interface.

. La spécification doit être précise et ne laisser aucun doute sur le comportement du système pour chaque entrée-sortie possible.

. La spécification doit être assez puissante pour permettre de spécifier des systèmes complexes.

. La spécification devrait permettre la génération automatique de l'interface, ce qui constitue un des objectifs de B.Sacré et I.Provot.

A l'heure actuelle dans le domaine de l'IHM, il y a d'autres qualités attendues d'un formalisme de spécification du dialogue. De plus en plus, ce formalisme doit permettre de spécifier des interfaces à manipulation directe. Il s'agit aussi de disposer d'un formalisme permettant la spécification de dialogues à plusieurs fils d'activité et asynchrones.

Le comportement d'un système interactif est de plus en plus guidé par des événements asynchrones et de nature différente. L'expérience montre que la description de ces événements et de leurs conditions d'occurrence est une tâche difficile. En IHM, l'asynchronisme a souvent été détourné au profit de la séquentialité. Si elle est une propriété séduisante pour l'informaticien, elle correspond mal au comportement de l'utilisateur et notamment aux dialogues multi-fils.

Chapitre 3 Présentation de notre langage de règles

3.1 Présentation

Pour rappel, notre architecture est composée de trois composants regroupant chacun un ensemble d'objet. Le composant Application regroupe les objets de l'application, le composant présentation regroupe les objets interactifs abstraits et le composant Dialogue, les objets de dialogue.
Chaque objet est défini par son interface, à savoir ses primitives de manipulation et les événements qu'il génère.

Les objets de dialogue sont organisés hiérarchiquement à l'aide de la relation "utilise". Les objets feuilles de cette hiérarchie correspondent à des objets interactifs et à des objets de l'application.
Pour créer un nouvel objet de dialogue, il convient donc de définir son interface et de décrire les actions effectuées en réponse aux primitives et aux événements reçus par l'objet de dialogue.

Chaque objet de dialogue possède une mémoire composée d'un ensemble de variables et est caractérisé à un moment donné par un état défini par l'ensemble des valeurs variables.
Un objet change d'état en réaction à la réception d'un événement ou en réponse à une primitive. L'ensemble des réactions exécutées lors d'un changement d'état définit le comportement de l'objet.
Afin de décrire le **comportement** d'un objet de dialogue et donc de manière plus générale, le dialogue de l'application, on dispose d'un **langage de règles**.

3.2 Description du langage de règles

Une règle décrit les actions à effectuer lorsque certaines conditions sont satisfaites.

Elle est composée d'une partie gauche et d'une partie droite. La partie gauche de la règle exprime cette condition sous la forme d'une combinaison d'événements et d'expressions booléennes. La partie droite reprend les actions à effectuer lorsque la condition est vérifiée.

IF (ev1 op ... op evi ... op exp1 ... op expi ...) THEN actions

avec :
evi = la survenance d'un événement
op = opérateur ET , OU
expi = l'expression booléenne qui teste le contenu de la mémoire (valeurs des variables) de l'objet de dialogue c-à-d son état actuel.

Evénement

Un événement peut être
- généré par un objet de dialogue à destination de son objet père
- généré au sein d'un objet de dialogue
- le déclenchement d'une primitive offerte par l'objet de dialogue

Expression booléenne

Il s'agit d'une expression qui teste le contenu de la mémoire de l'objet de dialogue.

Actions

Une action est une instruction

- de déclenchement de primitives offertes par les objets de dialogue fils.
- de déclenchement des primitives offertes par les objets de l'application.
- de manipulation de la mémoire de l'objet de dialogue.
- d'ordonnancement des actions.
- de génération d'événements au sein de l'objet ou à destination de son objet père.

Caractéristiques

Lors de la génération d'un événement tel que défini plus haut, les règles qui définissent le comportement d'un objet de dialogue doivent être évaluées dans leur ensemble car un même événement peut provoquer l'exécution de plusieurs règles.

Lorsqu'une règle est exécutée, toutes les actions correspondantes sont exécutées avant l'exécution d'une autre règle. Cette caractéristique prend tout son sens lorsqu'une action de la règle a pour objectif de générer un événement.

Les règles peuvent être groupées en fonction d'un objectif commun. Ce groupement peut même donner lieu à la construction d'un nouvel objet de dialogue afin d'améliorer la décomposition modulaire du dialogue et de contribuer à une meilleure réutilisabilité.

3.3 Exemple

Afin d'illustrer notre langage de règles, nous avons spécifié les objets de dialogue de la phase "enregistrement de commande-client " (voir annexe 2).

Chapitre 4 Comparaison avec d'autres formalismes de spécification

L'objectif de cette partie n'est pas tellement d'énumérer tous les avantages et les inconvénients de chaque formalisme envisagé ci-dessous, mais plutôt de faire ressortir à la fois les différences majeures et les similarités entre notre langage de règles et les autres formalismes de spécification du dialogue.

4.1 Les diagrammes état-transition

4.1.1 Définition

Le formalisme des diagrammes état-transition constitue sans doute aujourd'hui encore, la technique de spécification la plus populaire. Cette technique a été fort utilisée et l'est encore.

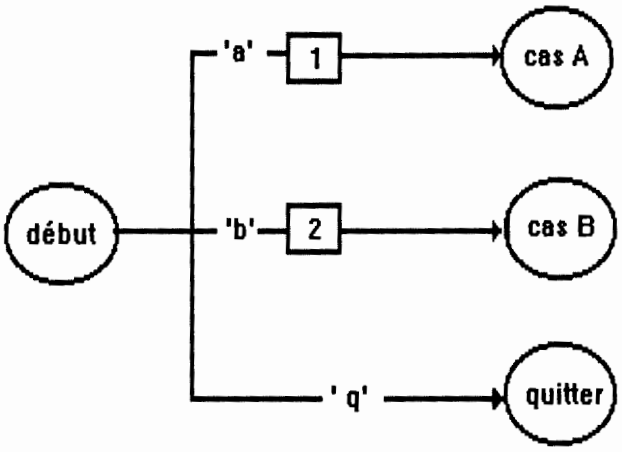
Il existe de nombreuses extensions de ce formalisme. Wasserman dans [Wasserman 85] a été l'un des premiers à construire une méthodologie de spécification et de développement d'un système interactif fondée sur ce concept (Rapid/Use).

Un diagramme état-transition se présente sous la forme d'un graphe orienté dont les noeuds correspondent à des états de l'interaction et sont représentés par des cercles et dont les arcs correspondent à des transitions représentées par des flèches.
Le modèle de base consiste à étiqueter chacun des arcs avec les actions physiques de l'utilisateur (les entrées) et à utiliser les noeuds pour représenter les états possibles de l'interaction.
Nous présentons maintenant quelques extensions de ce modèle de base, introduites par Wasserman pour permettre la spécification du dialogue de l'IHM.

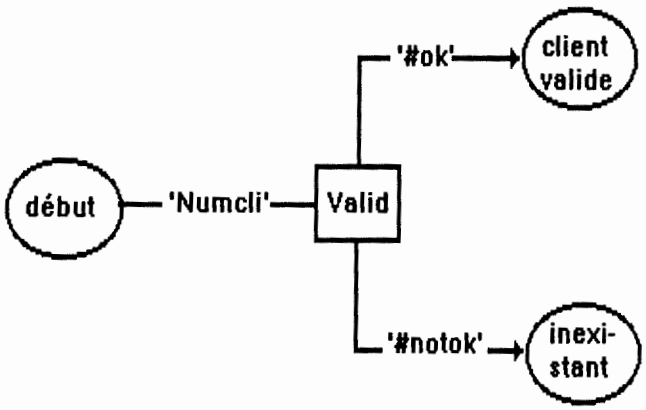
Extensions

1) Un noeud, représenté par un cercle, correspond à un état stable d'attente d'une entrée de commande par l'utilisateur. Chaque noeud dans le diagramme a un nom unique et un message de sortie peut être affiché à l'écran quand le noeud est atteint. Certains auteurs permettent d'attacher des actions aux noeuds, de sorte que quand le noeud est atteint, l'action qui lui est attachée est réalisée.
Il y a un noeud qui désigne l'état de départ et un qui désigne l'état final de l'interaction.

2) Une opération, représentée par un carré, peut être associée à la traversée d'un arc. Le dessin ci-dessous, montre un menu associé à deux traitements correspondant au cas A et au cas B. Lorsque l'utilisateur frappe la touche 'a', l'opération (traitement) 1 s'exécute (représentée par un carré libellé par le chiffre 1). Le système se retrouve alors dans le cas A (Idem 'b', opération 2, cas B).



3) La notion de variable a aussi été introduite. Elle permet de stocker des symboles qui peuvent ensuite être testés.
Le dessin ci-dessous, montre comment Wasserman peut prendre en compte des éléments sémantiques dans ses diagrammes qui sont principalement syntaxiques. Au début le système est dans un état dans lequel il attend l'introduction d'un numéro de client. L'utilisateur entre un numéro de client stocké dans la variable Numcli puis une opération de validation Valid est exécutée. Cette opération retourne dans une variable, le code "ok" si le client est valide, "notok" si il n'existe pas de client avec un tel numéro. Le système passe alors dans l'état suivant correspondant au résultat retourné.



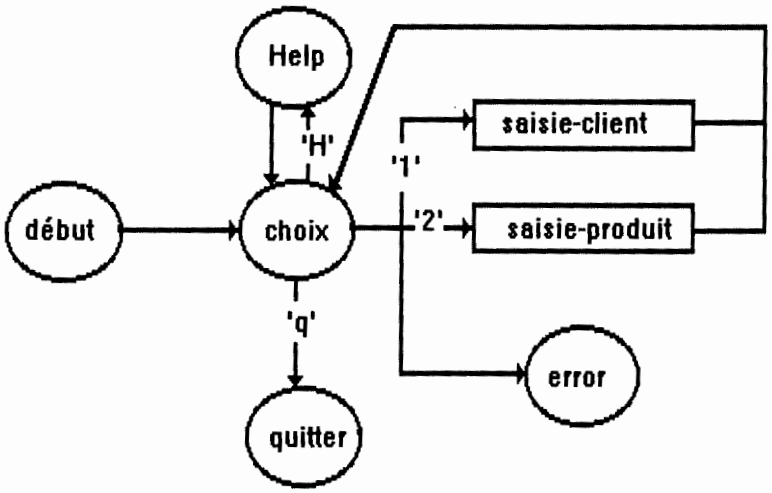
4) Un arc peut ne pas avoir de libellé et être traversé si aucun autre arc ne peut être emprunté. Il est appelé l'arc par défaut. Il correspond généralement au traitement des erreurs correspondant à des entrées inattendues de la part de l'utilisateur.

4.1.2 Les sous-diagrammes ou sous-conversations

Il s'agit de diagrammes dont certains arcs constituent des appels à d'autres diagrammes. Les arcs ont alors des noms de sous-diagrammes. Wasserman parle de sous-conversations ou de hiérarchie de diagrammes.

Un sous-diagramme est représenté dans le diagramme courant par un rectangle avec un nom de sous-diagramme à l'intérieur. Quand lors de la traversée d'un arc, on entre dans une sous-conversation, la traversée de l'arc courant est suspendue, de même que la conversation associée au diagramme courant. La traversée d'un arc étiqueté du nom d'un sous-graphe est complète lorsque le parcours du sous-graphe est achevé.

Certains auteurs autorisent des appels récursifs de diagrammes.



4.1.3 Evolution dans un diagramme

Le démarrage d'une session interactive est représenté par un état initial. Le passage d'un état à un autre s'effectue le long des arcs étiquetés. Le système se trouvant dans un état donné et l'utilisateur effectuant une action (une entrée), l'arc suivi est celui dont les conditions d'activation sont satisfaites. Si un arc ne peut être traversé, c'est que l'utilisateur vient de commettre une erreur. En général, chaque noeud comporte un arc qui conduit au traitement des anomalies.

Il existe des états dans les diagrammes pour lesquels le système n'attend pas une action de la part de l'utilisateur mais pour lesquels il produit lui-même des outputs, exécute une action ou teste une condition. Le système ne s'y arrête jamais car il effectue directement une transition vers le prochain état qui demande une entrée.

4.1.5 Evaluation comparative avec notre langage de règles

Représentation graphique

L'énorme avantage du formalisme des diagrammes état-transition par rapport à notre langage de règles, c'est qu'il a une représentation graphique très expressive. Ce formalisme présente un mécanisme explicite de représentation graphique du séquencement de l'interaction. Ces diagrammes dès lors, offre une facilité de lisibilité et de compréhension que n'offre pas un langage de spécification.

De plus, l'apprentissage d'un langage comme le notre où comme les langages à événements que nous envisagerons par la suite, est un problème non négligeable. Le formalisme des diagrammes apparaît comme plus simple à apprendre et à utiliser que notre langage de règle.

Mais cette facilité de lecture et de compréhension des graphes, due à leur aspect graphique, disparaît très vite dès que le nombre d'états augmente et que le système devient complexe.

La prise en compte de la correction des erreurs, du help et du undo amène un nombre exorbitant de noeuds et de transitions à gérer. On peut cependant faire usage des sous-diagrammes pour réduire la complexité d'un large graphe et pour le structurer.

Outils

La réalisation d'outils intégrés dans un UIMS, destinés à la production et à la modification de spécifications utilisant le formalisme des diagrammes de transition n'a rien d'un problème trivial.
La gestion de la représentation graphique de ces graphes peut s'avérer être un problème important pour des spécifications volumineuses.

Paradigme

Un diagramme état-transition modélise l'interaction entre l'utilisateur et le système sous la forme d'un graphe orienté. Le réalisateur de l'interface qui spécifie le dialogue pense en termes d'états de l'interaction et sa tâche consiste à exprimer le séquencement des états dans le temps en relation avec les actions de l'utilisateur.
Avec notre langage de règles, il n'en va pas de même. La spécification du dialogue passe par la spécification du comportement d'une hiérarchie d'objets. Pour rappel, notre architecture d'une application interactive est constituée d'une hiérarchie d'objets. On adopte une vue orientée objet du dialogue.Une fois les objets identifiés, il convient de spécifier pour chaque objet de la hiérarchie son interface (événements et primitives), puis son comportement à l'aide du langage de règles.

Interfaces modernes

Le reproche le plus important que l'on peut faire aux auteurs qui traitent le sujet des diagrammes de transitions, est de ne pas aborder l'usage de la souris, des fenêtres et des objets interactifs d'une interface à manipulation directe. Ce formalisme se limite à la description d'interfaces textuelles faisant le plus souvent usage d'un langage de commande et d'écran de type alphanumérique.

Wasserman pense que son modèle est facilement adaptable aux interfaces modernes mettant en oeuvre des nouvelles technologies, telles qu'écran graphique, souris et objets interactifs d'un toolkit. Cependant, il semble que la complexité des diagrammes explosera très rapidement lorsqu'il faudra spécifier tous les détails liés au passage d'un écran alphanumérique à un écran haute résolution sur lequel on peut dessiner, cliquer avec une souris sur un bouton, déplacer une fenêtre, faire défiler une liste d'éléments.

Dialogue séquentiel et asynchrone

Le formalisme des diagrammes de transition exprime bien la notion de séquence d'actions dans le temps. Cependant, ce formalisme reste une technique de spécification de dialogue séquentiel et pas asynchrone.
Contrairement à notre langage de règles, les diagrammes forcent souvent un ordre séquentiel dans la saisie des éléments. Si on veut autoriser un ordre de saisie quelconque, il faut alors prévoir tous les états et toutes les transitions pour tenir compte de l'ensemble des permutations possibles des éléments à saisir. L'explosion combinatoire du graphe qui en résulterait aurait comme conséquences des graphes illisibles et difficilement maîtrisables. Dès lors, il est fort probable que le réalisateur de l'interface fixera un ordre de saisie dans le dialogue.

De même, il est difficile le passage d'un point du dialogue d'un diagrammes, à un autre point dialogue d'un autre diagramme, si une transition n'a pas été prévue à cet effet. Par conséquent, la prise en compte de dialogue multi-fils et asynchrones paraît fortement compromise avec un tel formalisme. Par exemple, si l'utilisateur veut déplacer son curseur dans un champ de saisie, frapper quelques caractères, puis déplacer son curseur quelque part ailleurs, faire d'autres opérations pour enfin retourner au champ de saisie. La spécifica tion doit permettre de passer d'un diagramme à un autre et ainsi prévoir toutes les transitions possibles d'un noeud d'un graphe à un autre, autrement dit d'un point du dialogue à un autre.

De plus, les sous-diagrammes ne sont pas autonomes, en ce sens que lors de la traversée d'un arc étiqueté par le nom d'un sous-diagramme, le diagramme (dialogue) courant est suspendu pendant l'évolution dans le sous-diagramme appelé. Par contre dans notre architecture orientée objet et avec notre langage de règles, rien n'empêche qu'un objet père soit actif et donc que le dialogue qui lui est associé continue alors que son ou ses objets fils sont aussi actifs. Quand un objet émet un événement, il ne suspend pas son exécution. Il n'y a pas forcément interruption de son exécution, puis transfert du contrôle à l'objet récepteur, qui lui renverrait le contrôle à l'objet émetteur après qu'il ait fini son traitement.
Rien n'empêche en effet dans notre exemple, que l'objet de dialogue "Enregistrement_Cde" soit actif et génère un événement ou utilise une primitive offerte par l'objet de dialogue "Saisie_client", et que l'objet de dialogue "Saisie_cde" soit aussi actif et génère un événement à destination de son objet père "Enregistrement_Cde". Chaque objet étant capable de mémoriser son état courant, notre formalisme autorise des dialogues multi-fils et asynchrones.

La structuration

Les sous-diagrammes permettent de diviser un diagramme complexe en plusieurs diagrammes plus faciles à gérer [Jacob 83]. Ce mécanisme permet d'illustrer la structure d'un dialogue.
Les dialogues sont la plupart naturellement hiérarchiques et le mécanisme de sous-conversation permet de montrer cette hiérarchie dans les diagrammes.
En plus, décomposer un diagramme volumineux en plusieurs sous-diagrammes permet une meilleure lisibilité et compréhension de la spécification.

Notre langage de règles dispose également d'un mécanisme indirect de structuration de la spécification d'un dialogue. De part le modèle d'architecture sous-jacent, les objets sont structurés en une hiérarchie à l'aide de la relation "utilise" définissant des niveaux d'abstraction.

La structure hiérarchique des objets de dialogue suggère 2 méthodes possibles de construction du composant dialogue:

Top-down:

La démarche top-down consiste à trouver les objets de dialogue par **affinements successifs** en partant des objets du dialogue les plus **abstraits**, en connaissant ou non les objets de l'application, et en identifiant parallèlement les objets interactifs utilisés par le composant dialogue.

Bottom-up:

La démarche bottom-up consiste tout d'abord à identifier les objets interactifs de l'interface, en connaissant ou non les objets de l'application, pour ensuite déduire les objets de dialogue par abstractions successives et identifier, en parallèle, les objets de l'application s'ils n'étaient pas connus au départ, ou à affiner et enrichir leurs primitives si ces objets étaient disponibles au départ.
Cette démarche présente un double intérêt:

- On conçoit le dialogue de l'application interactive à partir de quelque chose de concret, en l'occurrence les objets interactifs qui constituent la partie présentation de l'interface.

- La définition de la partie présentation comme première étape dans la conception d'une application correspond à la notion de maquettage d'écran utilisée lors de la conception d'une application comme outil de communication privilégié avec l'utilisateur final lors des étapes initiales de conception d'une application.

La construction des dialogues attachée à la description des objets interactifs avec simulation des objets de l'application constitue l'étape suivante naturelle au terme de laquelle nous disposons de l'interface d'une application interactive.

Réutilisation

L'utilisation de sous-diagrammes et d'objets dans notre langage de règles permet une certaine réutilisabilité de ces composants. Ils supportent la création de librairies de sous-diagrammes ou d'objets prédéfinis.

4.2 Les langages à événements

4.2.1 Introduction

Les langages à événements sont nombreux. Plus rares sont les formalismes à événements pour l'expression des IHM [Carlsen 89, Hill 87, Green 85]. Ces langages ont tous le même modèle sous-jacent du dialogue : le modèle à événements. Ces formalismes à événements sont beaucoup moins stabilisés que le formalisme précédent des diagrammes de transition.

Ce modèle est basé sur le concept d'événements que l'on retrouve dans la plupart des packages graphiques et des boîtes à outils. Les dispositifs d'entrée et de sortie sont vus comme des sources d'événements. Chaque dispositif d'entrée génère un ou plusieurs événements quand l'utilisateur interagit avec ce dispositif. Ces événements sont placés dans une file d'attente quand ils sont générés et l'application les traite un à la fois. Dans ces packages graphiques, il y a un nombre fixe et prédéfini d'événements qui ne peuvent être générés que par les dispositifs d'entrée et de sortie.

4.2.2 Le modèle à événement de Green

4.2.2.1 Présentation

Ce modèle est une extension de l'idée de base énoncée dans l'introduction ci-dessus. Ce modèle est basé sur l'architecture de Seeheim. Il y a dès lors deux sources d'événements dans le composant contrôle du dialogue. La première source est constitué des deux autres composants de l'architecture. Le composant présentation et le composant interface avec l'application. La deuxième source d'événements est le composant dialogue lui même. Il y a un nombre arbitraire de types d'événements.

Le système à événements est composé d'un ensemble d'événements et de gestionnaires d'événements qui peuvent s'exécuter en parallèle. Dans ce modèle le composant contrôle du dialogue est vu comme une collection de gestionnaires d'événement.

Quand un événement est généré, il est envoyé à un gestionnaire d' événement. Un gestionnaire d' événement est un processus qui quand il reçoit un des événements qu'il peut traiter, exécute une procédure. Cette procédure peut réaliser certains calculs, générer de nouveaux événements, créer de nouveaux gestionnaires d'événement ou en détruire.

Dans le cas le plus simple, il y a des événements et des gestionnaires d'événements spécialisés. Chaque événement appartient à un type qui ne peut être traité que par un gestionnaire d'événements et celui-ci ne peut traiter que ce type d'événement. Par contre le gestionnaire peut générer plusieurs types d'événements différents.

Les modèles plus complexes ont des systèmes de distribution des événements pouvant fonctionner selon différents principes. D'une part un gestionnaire d'événements peut être actif ou passif. Il ne peut traiter un événement que s'il est actif. D'autre part, le gestionnaire peut manifester de l'intérêt pour certains types d'événements. S'il ne s'annonce pas comme désirant recevoir tel type d'événement, il n'est pas informé de leur survenance et ce , même s'il est capable de le gérer.

L'événement peut être traité par un seul gestionnaire d'événements ou par plusieurs. S'il ne peut être traité que par un seul gestionnaire mais que plusieurs sont susceptibles de s'y intéresser, un mécanisme de priorité ou d'arbitrage doit être défini.

4.2.2.2 Comportement d'un gestionnaire d'événement

Le comportement d'un gestionnaire d'événement est décrit par un canevas ("Template") comportant plusieurs sections. La première section définit les paramètres du gestionnaire d'événement, c'est-à-dire les événements auxquels il réagit. La seconde section définit ses variables locales et la dernière section définit les procédures utilisées pour traiter les événements.

Lorsqu'un gestionnaire d'événement est créé, son canevas doit être spécifié. Plusieurs gestionnaires d'événement peuvent être créés à partir du même canevas de base. C'est à l'exécution que les instances du gestionnaire d'événement sont crées. Une fois créé, un gestionnaire d'événement est dans son état actif et peut alors réagir aux événements.

4.2.2.3 Exemple

Voici ci-dessous la spécification d'un exemple tiré de [Green 85]. Il s'agit du gestionnaire d'événement associé à la séquence du login lorsqu'un utilisateur se connecte à son terminal.

```
Eventhandler login Is
        Token
        keyboardstring s;

        Var
        int state = 0;
        string user_id, password;

        Event Init  {
           print "login :";
        }

        Event s : string {
           if (state = = 0) {
              user_id = s;
              state = 1;
              print "password :";
              }
          else {
               password = s;
               state = 0;
               process_login (user_id, password);
                  };
        }

End Eventhandler login;
```

Le gestionnaire d'événement réagit à deux types d'événements. L'événement Init est envoyé quand le gestionnaire d'événement est crée. L'autre événement est reçu quand l'utilisateur frappe un string au clavier. La variable "state" est utilisée pour déterminer si le string est l'identificateur de l'utilisateur ou son mot de passe. En pratique, les déclarations "print" et "process_login" sont des événements envoyés respectivement au composant présentation et interface avec l'application.

De manière générale, la structure d'un gestionnaire d'événement est :

```
Eventhandler event_handler_name Is

    Token
    token_name event_name;
        .
        .
        .

    Var
    type variable_name = initial_value;
        .
        .
        .

    Event event_name : type {
        statements
    }
        .
        .
        .
    Event event_name : type {
        statements
    }

    End event_handler_name;
```

4.2.3 Les systèmes événement-réponse

Le langage envisagé est un langage de type événement-réponse issu des systèmes événement-réponse et proposé par R. D.Hill dans [Hill 87]. Cette section sera composée de deux parties. La première présente de manière assez formelle le modèle sous-jacent à un système événement-réponse (SER). La seconde partie présente la syntaxe de ce langage de spécification, appelé langage événement-réponse (LER).

4.2.3.1 Présentation formelle du modèle SER

Un SER est 4-tuple (F, E, R, S) où :

. F est un ensemble d'indicateurs (flags) : variables booléennes (positionnées ON, OFF)
. E est un ensemble d'événements
. R est un ensemble de règles de réponse ayant les formes suivantes :

(a) evt F1 -------> F2
(b) F1 -------> F2

avec "evt" est un élément de E et, F1 et F2 sont des sous-ensembles de F.
Une règle de la forme (a) est une règle régulière.
Une règle de la forme (b) est une E-règle.

L'ensemble F1 et le "evt" optionnel sont appelés la condition de la règle.
L'ensemble F2 est appelé l'action de la règle.

. S inclus dans F est l'ensemble des flags qui sont initialement positionnés ON

Les événements en entrée sont mémorisés dans une file d'attente unique.

Une règle de réponse est activable quand :

1) chaque flag dans F1 est positionné ON
2) si la règle est une règle régulière, "evt" est l'événement en tête dans la file d'attente

L'exécution d'un SER procède en répétant les deux pas suivant :

1) Considérant les E-règles, répéter jusqu'à ce qu'il n'y ait plus de règle activable :

a) identifier et marquer toutes les règles qui sont activables étant donné les valeurs courantes des flags, et
b) simultanément, activer toutes les règles marquées

2) Considérant seulement les règles régulières :

a) identifier et marquer toutes les règles qui sont activables étant donné les valeurs courantes des flags et l'événement en tête dans la file d'attente
b) simultanément, activer toutes les règles marquées, et
c) enlever l'événement en tête dans la file d'attente et passer à l'événement suivant.

Activer une règle consiste en trois étapes qui doivent être exécutées séquentiellement. Les trois étapes sont :
1) Tous les flags apparaissant dans les conditions (F1) de toutes les règles marquées sont mis à OFF c'est-à-dire désactivés
2) Tous les flags apparaissant dans les actions (F2) de toutes les règles marquées sont positionnés à ON
3) Toutes les règles marquées redeviennent non marquées.

4.2.3.2 Présentation du LER

Les indicateurs (flags) et les événements ont une structure de record dont les champs peuvent être lus et modifiés.

En ce qui concerne un flag, en plus du champ contenant la valeur (ON,OFF) du flag, d'autres champs peuvent être déclarés. La structure d'un événement est similaire, excepté que le champ de la valeur booléenne n'existe bien sûr pas.

Pour ce qui concerne les expressions de sortie, un nouveau type d'élément est ajouté au modèle SER. Il s'agit des événements de sortie. Un événement de sortie à la même structure qu'un événement classique (structure de record avec des champs). La seule restriction est que ces événements de sortie ne peuvent pas apparaître dans des conditions de règles.

Dans le formalisme SER, les indicateurs sont utilisés comme des variables pour stocker de l'information qui arrivent des dispositifs de commande en entrée (présentation) ou des routines de l'application.
Les événements de sortie sont utilisés pour signaler et pour envoyer de l'information aux dispositifs de sortie et aux routines de l'application. Ils servent donc à communiquer avec l'environnement c'est-à-dire l'application et la présentation.

Syntaxe du langage
Les règles ont la forme générale : condition ----> action .

Dans une condition , le nom de l'événement apparaît en premier, suivi de la liste des flags. Dans le cas des E-rules, "---" apparaît à la place du nom de l'événement.

La partie action d'une règle est une extension de la liste des flags dont il faut modifier la valeur (F2). Ces extensions sont constituées des opérations d'assignation et d'envoi. Donc, une action consiste soit à donner une valeur (ON) à chaque élément d'une liste de flag, et/ou à envoyer des événements de sortie et/ou à assigner des valeurs aux champs de flags et d' événements de sortie.

Pour signaler l'envoi d'un événement de sortie, on fait suivre son nom d'un "!".
Pour signaler qu'un flag est mis à ON, on fait suivre son nom d'un "↑".

Les assignations portent sur des valeurs de champs soit de flags soit d'événements de sortie. Elles ont la forme suivante :

<target>.<value-field> <--- <source>.<value-field>
avec <target> le nom d'un événement de sortie ou d'un flag,
<value-field> le nom d'un champ de valeur de la cible ou de la source,
<source> le nom d'un événement ou d'un flag. On peut aussi mettre une expression constante dans le membre de droite de l'assignation.

Pour une assignation simultanée de valeurs aux différents champs d'un flag ou d'un événement de sortie, on dispose de l'abréviation suivante :
<target> <---
.<value-field> <--- <source>.<value-field>
.<value-field> <--- <source>.<value-field>

De même, pour l'envoi d'un événement de sortie et l'assignation simultanée de ses champs, on dispose de l'écriture abrégée suivante :

<target> !
.<value-field> <--- <source>.<value-field>
.<value-field> <--- <source>.<value-field>

Finalement, il nous reste à définir l'ensemble S des flags initialement ON. Pour cela, on dispose d'un événement spécial appelé "initially", qui est envoyé à tous les modules quand on débute l'exécution de l'application. Similairement, l'événement "finally" est envoyé à tous les modules quand l'application se termine.

4.2.3.4 Exemple

L'exemple de spécification ci-dessous décrit la saisie du nom et du prénom d'un client. Le dialogue est séquentiel. Il y a d'abord saisie du nom puis du prénom. Une fois ces informations saisies, le client est enregistré. Un message "client-enregistré" est alors affiché à l'écran.

```
initially --->
            nom↑


evt-exist-nom  nom --->
                        client.nom <--- evt-exist-nom.nom
                        prenom↑


evt-exist-prenom  prenom --->
                        client.prenom <--- evt-exist-prenom.prenom
                        client !


evt-fin-enregistrer --->
                        evt-client-enregistré !
                        nom↑
```

Commentaires :

L'exécution de la spécification commence avec tous les flags positionnés à OFF et avec l'événement "initially" en tête dans la file d'attente des événements. La première règle est alors activée. Le flag "nom" est alors ON. Le prochain événement ne peut être que "evt-exist-nom" généré quand le nom a été saisi. La deuxième règle est alors activée, le flag "prenom " est positionné à ON et le champ nom d'une structure de client est garni. Une fois l'événement "evt-exist-prenom" généré et indiquant que le prénom du client a été saisi, le champ prenom de la structure est garni et cette structure de client est envoyée à l'application sous forme d'un événement de sortie pour faire l'objet du traitement d'enregistrement. L'événement " evt-fin-enregistrer" est ensuite généré par l'application une fois l'enregistrement du client accompli et un message de sortie " evt-client-enregistré" est envoyé à la présentation pour être affiché à l'écran.

Ci-dessous maintenant, une version multi-thread de l'exemple précédent. L'utilisateur a le libre choix d'entrer les informations nom et prénom du client dans l'ordre qu'il désire et quand il le désire, voir même simultanément. Dès que ces informations ont été saisies, l'enregistrement du client est effectué.

```
initially --->
              nom↑
              prenom↑

evt-exist-nom   nom --->
                              client.nom <--- evt-exist-nom.nom
                               nom-saisi↑

evt-exist-prenom   prenom --->
                              client.prenom <--- evt-exist-prenom.prenom
                              prenom-saisi↑

--- nom-saisi  prenom-saisi  --->
                              client !
                              attendre-fin-enregistrer↑

evt-fin-enregistrer  attendre-fin-enregistrer  --->
                              evt-client-enregistré !
                              nom↑
                              prenom↑
```

Une fois le nom et le prénom saisis (les flag nom-saisi et prenom-saisi sont positionnés), la E-règle est activée. Cette règle envoie la structure de client à l'application son enregistrement. Cette spécification contient donc deux sous-dialogues concurrents et la E-règle fournit un moyen simple de synchronisation. L'indicateur "attendre-fin-enregistrer" permet au dialogue de se synchroniser avec l'application avant d'émettre le message de sortie "evt-client-enregistré" vers la présentation.

4.2.4 Evaluation comparative avec notre langage de règles

Green

Le désavantage évident de la notation de Green, c'est qu'elle ressemble plus à un programme, ou à un ensemble d'algorithmes.
De plus, les gestionnaires d'événement sont décrits dans un langage syntaxiquement très proche du langage C. Par conséquent, ne serait-il pas aussi préférable d'écrire directement le code C. De plus l'usage de ce formalisme pourrait se limiter aux programmeurs C.

Dialogue Multi-fils et Asynchrone
Tout comme notre langage de règles, ce formalisme supporte la spécification de dialogues mutli-fils et asynchrones. Puisque chaque gestionnaire d'événement à son propre état local et que plusieurs gestionnaires d'événement peuvent être actifs au même moment, l'utilisateur est libre de déplacer d'un point du dialogue à un autre, sans pour cela devoir compléter la commande courante ou sauver explicitement l'état du dialogue. L'utilisateur peut ainsi être impliqué dans plusieurs dialogues séparés ou communicants au même moment.
De même dans notre langage, chaque objet possède une mémoire composée d'un ensemble de variables qui lui permet de mémoriser son état local.

Chaque objet de notre langage de règles ou chaque gestionnaire d'événement peut être vu comme un processus séparé. Si l'on veut spécifier un dialogue usant du multi-fenêtrage, chaque fenêtre peut être décrite soit avec le formalisme des diagrammes de transition, soit avec un formalisme à événement. Cependant, lorsqu'il s'agit de décrire les communications entre les fenêtres, le formalisme des diagrammes de transition rencontre des problèmes insurmontables alors qu'un formalisme à événement est parfaitement approprié. Par exemple, une opération de "couper-coller" un texte d'une fenêtre dans une autre, peut facilement être traité par émission d'événements d'une fenêtre à l'autre.

Système événement-réponse

Le langage LER est un des premiers exemples d'expression du parallélisme dans la spécification d'interfaces. Le parallélisme résidant ici dans la possibilité d'un usage simultané de plusieurs dispositifs de commande. Le langage ERL permet de spécifier des interfaces où l'utilisateur est libre de manipuler plusieurs dispositifs de commande pour réaliser différentes tâches en même temps. Par exemple, dans une application de dessin, l'utilisateur peut à l'aide d'une main dessiner une droite en déplaçant la souris et à l'aide de l'autre main sélectionner la taille du trait, sa couleur et d'autres paramètres en utilisant un autre dispositif de commande comme le clavier .

Un système événement-réponse est également bien adapté à la spécification de dialogues multi-fils et donc d'interfaces à manipulation directe comme nous l'avons à travers l'exemple. Dans un ERS, chaque sous-dialogue d'un dialogue multi-fils peut s'exécuter en parallèle. Un dialogue complexe sera représenté comme un ensemble de dialogues plus simples qui s'exécuteront en parallèle. L'objet de l'interface sélectionné à l'écran détermine quelles dialogues sont actifs.

Résumé des comparaisons

Après cette présentation de formalismes à événement, nous pouvons répertorier notre langage de règles dans cette catégorie. L'interaction est modélisée comme un ensemble de machines élémentaires capables de répondre à des événements. Un réalisateur d'interface pense en termes d'événements et de processeurs de traitements. Sa tâche consiste à identifier les événements, à définir la nature des traitements, leur enchaînement et leurs points de synchronisation.

La caractéristique majeure des formalismes à événement, est qu'ils sont particulièrement bien adaptés à la spécification de dialogues asynchrones et multi-fils.

C'est Green qui, dans un article de 1986 [Green 86], a établi que le formalisme à événement à la plus grande puissance descriptive que le formalisme des diagrammes état-transition. Le formalisme à événement peut être employé pour spécifier aussi bien un dialogue séquentiel , qu'un dialogue asynchrone. Ceci a comme autre conséquence qu'un dialogue décrit dans la notation des diagrammes de transition ou à l'aide de la grammaire BNF, peut toujours être converti dans la notation événementielle.

Les langages à événements nécessitent un temps d'apprentissage non négligeable et reste moins expressif qu'un diagramme état-transition.

4.3 Le formalisme de Jacob

Ce formalisme présente une vue orientée objet d'une interface à manipulation directe et combine une notation événementielle avec celle des diagrammes état-transition.

4.3.1 Interface à manipulation directe

Jacob dans [JACOB 86] présente un formalisme adapté à la spécification d'interfaces à manipulation directe. Selon lui, une interface à manipulation directe rassemble une collection d'objets interactifs présentés à l'écran. L'utilisateur peut interagir directement avec ces objets. Une fois un objet sélectionné, l'utilisateur peut dialoguer avec cet objet. Chaque objet a donc son propre dialogue, que l'utilisateur peut activer ou désactiver. Par exemple, l'utilisateur peut déplacer son curseur dans un champ de saisie, frapper quelques caractères, puis déplacer son curseur quelque part ailleurs, faire d'autres opérations et enfin retourner au champ de saisie. Il convient donc de pouvoir spécifier une multitude de mini-dialogues, chacun pouvant être interrompu et repris à tout moment.

4.3.2 Le langage de spécification

Spécifier une interface à manipulation directe au moyen du formalisme des diagrammes état-transition, n'est pas sans posé certains problèmes comme nous l'avons vu précédemment.
Nous avons par exemple signalé l'explosion combinatoire du graphe.

Le formalisme de Jacob permet de spécifier séparément le dialogue associer à chacun des objets interactifs de l'interface. Il décompose la spécification d'un dialogue multi-fils en plusieurs spécifications simple-fils. Les objets interactifs envisagés par Jacob sont similaires à ceux envisagés dans notre architecture et dans notre langage de règles. Il s'agit par exemple de boutons, de champs de saisie, de scroll-bar ou de menus.

Une spécification se résume à trois étapes :
(1) définir une collection d'objets interactifs
(2) spécifier le comportement interne (la syntaxe) de chaque objet
(3) fournir un mécanisme pour combiner ces objets dans une interface utilisateur.

Comment spécifier la syntaxe d'un objet interactif ?

C'est ici que réside l'originalité du formalisme de Jacob. Il spécifie séparément le comportement de chaque objet par un simple diagramme, les transitions correspondant à des événements. Ces événements sont générés par l'objet interactif suite aux actions de l'utilisateur sur cet objet.

Comment combiner les spécifications des différents objets ?

Un mécanisme de coordination (le coordinateur) a été défini afin de gérer les appels aux diagrammes individuels. Il active ou désactive les diagrammes des objets interactifs et il alloue les événements aux objets. Les appels aux diagrammes se font par un mécanisme semblable à celui d'un appel à une routine. Le coordinateur peut suspendre un diagramme lors d'un appel à un autre diagramme et mémoriser l'état du diagramme interrompu. Le dialogue d'un diagramme reprend à partir de l'état interrompu. Il n'y a qu'un diagramme à la fois qui est actif au cours de l'interaction.

Dans le diagramme actif, on passe d'un noeud à un autre via les arcs de transition selon les événements générés. C'est le coordinateur qui examine et alloue l'événement suivant. A partir du noeud atteint, la transition appropriée à l'événement est alors franchie pour arriver à un nouvel état.
Cela continue de cette manière dans le diagramme actif jusqu'au moment où on atteint un noeud pour lequel aucune transition n'est franchissable étant donné l'événement d'entrée. A ce moment, le coordinateur interrompt et mémorise l'état du diagramme courant . Il examine les diagrammes des autres objets interactifs et leurs états, à la recherche d'un diagramme qui dans l'état où il est peut accepter l'événement d'entrée. Le diagramme, sélectionné est alors activé. Un événement d'entrée qui ne correspond à aucune transition d'aucun graphe est alors traité comme une erreur conduisant à un diagramme spécifique.

Voici le canevas de spécification d'un objet interactif :

From : une liste d'objets interactifs desquels l'objet spécifié hérite des éléments de ces objets.

Ivars : une liste de variables instanciées de l'objet interactif. On peut aussi trouver dans cette partie, des objets interactifs de plus bas niveau (plus élémentaires) qui seront utilisés comme des composants de l'objet spécifié. Un objet peut en effet en contenir un ou plusieurs.

Methods : Des définitions de procédures propres à l'objet interactif.

Tokens : On trouve dans cette partie la définition des événements d'entrée et de sortie utilisés dans le diagramme état-transition de syntaxe de l'objet interactif.

Syntax : le diagramme état-transition

Subs : une liste de sous-diagrammes appelés comme des routines dans la syntaxe du diagramme spécifié.

4.3.3 Exemple

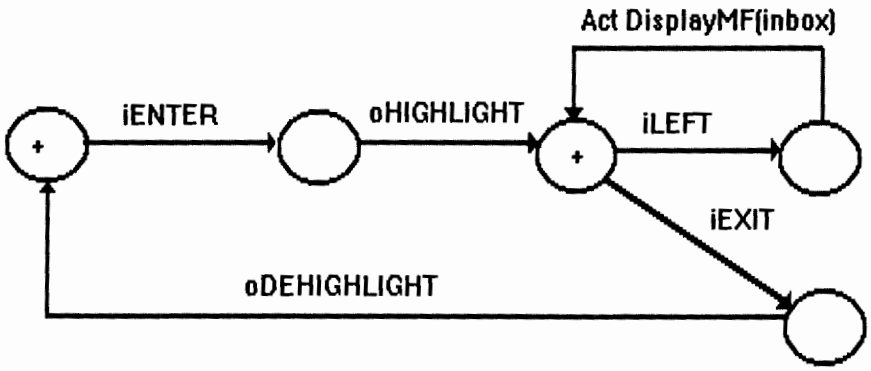
Ci-dessous un exemple de spécification tiré de [Jacob 86]. Il s'agit d'un bouton poussoir qui lorsqu'il est activé affiche un message à l'écran.

INTERACTION_OBJECT MessageDisplayButton is

Ivars :
position = {100 200 64 24} (coordonnées du bouton représenté comme un rectangle)

Methods :
draw() {Affiche_Texte_Bouton (position "display")} (label du bouton)

Tokens :
iLEFT {clic souris sur le bouton gauche}
iENTER {le curseur entre dans la surface du bouton }
iEXIT {le curseur sort de la surface du bouton}
oHIGHLIGHT {mise en inverse vidéo du bouton}
oDEHIGHLIGHT {désactive l'inverse vidéo du bouton}
Syntax :



end INTERACTION_OBJECT;

Commentaires :

Dans un diagramme, on peut associer à une transition un événement d'entrée ou de sortie, le nom d'un autre diagramme à appeler comme une sous-routine, une action à réaliser ou une condition à tester, ou rien, et dans ce cas, cette transition est empruntée si aucune autre ne peut l'être. Les noms des événements d'entrée commencent par un "i" et ceux de sortie par un "o". Une action comme **Act DisplayMf(inbox)** appelle une procédure définie dans l'application. Les états où il est possible d'interrompre le dialogue sont marqués d'une croix "+".

Ci-dessus, quand le curseur entre dans la surface du bouton de coordonnées "position", l'événement d'entrée iENTER est généré. Cet événement est défini localement dans l'objet MessageDisplayButton. Le diagramme de cet objet est alors appelé par le coordinateur et l'évolution dans le diagramme commence. L'événement iENTER est accepté et le bouton est ensuite mis en inverse vidéo "oHIGHLIGHT". On arrive à un état où on attend une nouvelle action de l'utilisateur. S'il clique sur le bouton de gauche de la souris l'événement iLEFT est généré et on passe à l'état suivant. Dans cet état, une action d'affichage est alors exécutée avant de revenir à l'état précédent. Sinon, si le curseur de la souris sort du bouton, c'est l'événement iEXIT qui est généré et l'on revient à l'état de départ du diagramme.

4.3.4 Héritage

Avec ce langage de spécification, un objet interactif hérite des parties Ivars, Methods, Tokens et Subs de tous les objets énoncés dans sa section **From**. Il est toujours possible à un objet de déclarer ses propres sections, ce qui a pour effet d'annuler l'héritage des autres objets et de le rendre spécifique.
L'exemple ci-dessous montre quelques objets interactifs génériques. L'objet GenericItem définit quelques événements (Tokens) qui sont applicables à un large nombre d'objets interactifs. Les événements iEnter et iExit sont définis de manière générique en terme d'instance d'une variable non spécifiée, "position". Cette variable sera définie par l'objet qui hérite de l'objet GenericItem. Highlighter est un autre objet générique qui gère l'activation et la désactivation en inverse vidéo . Il contient les événements oHighlight et oDehighlight définit aussi en terme d'instance de la variable "position". On trouve aussi un exemple de l'emploi de sous-diagrammes, Enterhigh et Exithigh. Finalement, un bouton générique GenericButton est défini. L'action dans son diagramme de syntaxe appelle une procédure "DoAction", que chaque objet en héritant peut redéfinir à sa manière. Il est alors possible de redéfinir l'objet de notre exemple au point 4.3.3 en fonction de tous ces objets génériques.

INTERACTION_OBJECT GenericItem is

Tokens :
iEnter {le curseur entre dans la surface du bouton }
iExit {le curseur sort de la surface du bouton}
iLeft {clic souris sur le bouton gauche}
iMiddle {clic souris sur le bouton du milieu}
iRight {clic souris sur le bouton droit}
iChar {frappe au clavier}

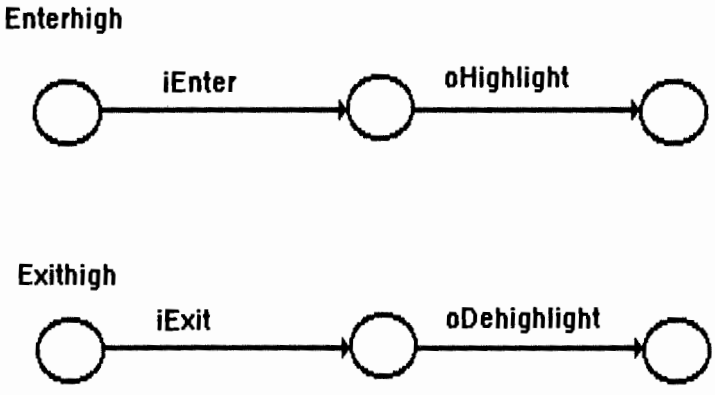
end INTERACTION_OBJECT;

INTERACTION_OBJECT Highlighter is

Ivars :
isHighlighted = false

Tokens :
oHIGHLIGHT { if not isHighlighted then
InvertRect(position) isHighlighted = true
endif}
oDEHIGHLIGHT { if isHighlighted then
InvertRect(position) isHighlighted = false
endif}

Subs :



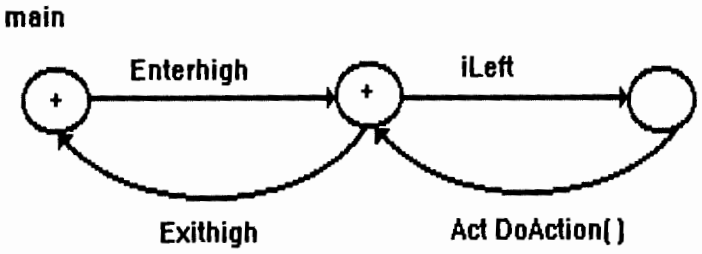
end INTERACTION_OBJECT;

INTERACTION_OBJECT GenericButton is

From : Highlighter, GenericItem

Methods : Draw()

Syntaxe :



end INTERACTION_OBJECT;

INTERACTION_OBJECT MessageDisplayButton is;
From : GenericButton

Ivars :
position = {100 200 64 24}
legend = "Display"

Methods : DoAction () {DisplayMessage}

end INTERACTION_OBJECT;

4.3.5 Evaluation comparative avec notre langage de règles

Jacob définit un formalisme présentant quelques caractéristiques très intéressantes comme l'héritage et l'usage des diagrammes de transition. Il parvient à concilier de manière subtile le formalisme des diagrammes état-transition avec le modèle à événement.
Il décompose en quelque sorte un large diagramme de transition en une collection de sous-diagrammes associés aux objets interactifs. Il définit ensuite un mécanisme implicite permettant de gérer ces objets.

Comme dans BIPS, il modélise une interface à manipulation directe sous la forme d'un ensemble d'objets. Les objets qu'il envisage se limite à des objets interactifs élémentaires comme les boutons, menus, boîtes de dialogue.

Il devrait être possible d'étendre ce formalisme à des objets de plus haut niveau comme nos objets de dialogue.

Grâce à une librairie d'objets génériques et un mécanisme d'héritage, on évite la répétition dans les spécifications. Il est devrait être aisé d'exprimer à l'aide du formalisme de Jacob, une hiérarchie d'objets telle que proposée dans l'architecture BIPS. Une première étape consisterait à définir la hiérarchie des classes d'objets interactifs comme le fait Jacob. Ensuite, il conviendrait de pouvoir spécifier les classes d'objets de dialogues. Un objet de dialogue peut en utilisé un ou plusieurs autres. Il utilise aussi des objets interactifs correspondants aux feuilles de la hiérarchie. Le comportement de chaque objet de dialogue est exprimé à l'aide d'un diagramme de transition.

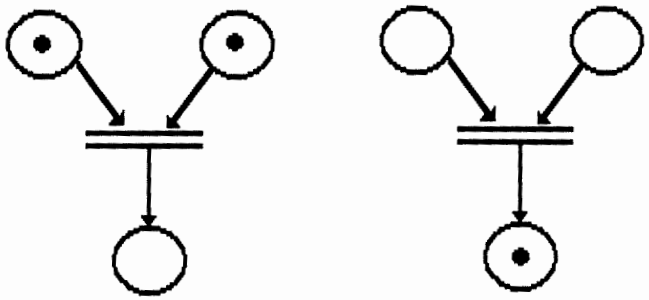
4.4 Autres techniques de spécification du dialogue

Dans cette partie, nous présentons brièvement trois autres formalismes de spécification que l'on retrouve fréquemment dans la littérature. Nous ne les analyserons pas en détail. Nous les présentons au lecteur à titre d'information.

4.4.1 Les réseaux de Pétri

Un réseau de Pétri est un quintuplet R = {P,T, Pré,Post, Mo}.
P est un ensemble de places qui ressemblent aux états des diagrammes état-transition. T est un ensemble de transitions, Pré et Post sont des ensembles respectivement de Pré et de Postconditions. Mo est le marquage initial du réseau. Un réseau de Pétri est un arbre avec deux types de noeuds appelés Places et Transitions. Les noeuds sont reliés par des arcs orientés.

Mo est un ensemble de places qui sont marquées à l'aide de jetons. Une transition peut être franchie si les places qui lui sont reliées dans le sens place-transition ont chacune un jeton. Lorsque la transition est franchie, chaque place de sortie reliée à la transition reçoit un jeton et le jeton est retiré des places de départ comme le montre le dessin ci-dessous.



Les réseaux de Pétri ont un pouvoir d'expression plus grand que les diagrammes état-transition. Ils ont été utilisés pour spécifier le parallélisme, ce qui était impossible avec les diagrammes de transition. De plus, à la différence avec les diagrammes état-transition, les états ne sont pas représentés par des places mais par le marquage, d'où une expression graphique plus compacte. Notons encore que les diagrammes état-transition sont un cas particulier des réseaux de Pétri où chaque transition n'a qu'une place en entrée et une place en sortie.

Le modèle de base des réseaux de Pétri a été enrichi de plusieurs concepts pour permettre une meilleure spécification des interfaces. Citons par exemple la possibilité d'avoir plusieurs arcs entre une place et une transition, ce qui signifie que pour franchir la transition, il doit y avoir au moins autant de jetons que d'arcs dans la place de départ. La notion de temps peut également être prise en compte, de même que l'exclusivité qui assure qu'une seule place parmi plusieurs a un jeton. Certains ont étendu la notion de jeton à celle de structure de donnée.

4.4.2 Grammaire BNF

La grammaire est utilisée pour décrire le langage employé par l'utilisateur pour communiquer avec le système.
En spécification d'interface, les éléments terminaux d'une grammaire BNF servent à représenter les actions physiques élémentaires de l'utilisateur, ou si l'on s'en réfère au modèle de Seeheim, ce sont les événements générés par le composant présentation. Les éléments terminaux sont combinés dans des règles de production pour former des structures de plus haut niveau appelées éléments non-terminaux. Les non-terminaux et les règles de production définissent l'espace des commandes et donc le langage utilisé par l'utilisateur lors de son interaction avec le système. Il est possible d'attacher des actions de l'application à chaque production de la grammaire.

Ci-dessous l'énoncé d'un exemple de spécification qui décrit les actions que l'utilisateur doit exécuter pour dessiner une droite avec sa souris dans une application graphique. Il doit d'abord cliquer (button) une fois pour signaler le point de départ de la droite, ensuite il déplace (move) la souris jusqu'à la longueur désirée et termine en cliquant une deuxième fois avec la souris.

```
line ----> button action1 end_point
end_point ----> move action2 end_point | button action3
action1 ----> {record first point}
action2 ----> { draw line to current position}
action3 ----> {record second point}
```

Les grammaires hors contexte modélisent l'interaction sous-forme d'un langage. Un réalisateur d'interface pense en termes de transformation d'éléments non-terminaux. Sa tâche consiste à définir la forme du langage.

Ce genre de formalisme reste cependant difficile à lire et à comprendre. Il nécessite un temps apprentissage non négligeable. Par exemple, les éléments non-terminaux dans une expression peuvent être remplacés par plusieurs éléments non-terminaux à travers plusieurs itérations successives avant qu'un

élément terminal soit atteint. Cette structure d'arbre à plusieurs niveaux est difficile à suivre, et au moment où on atteint les feuilles (éléments terminaux) on a vite oublier l'expression de départ au sommet de la hiérarchie. Une spécification écrite dans un tel formalisme est donc ardue à lire et à comprendre.

4.4.3 Les systèmes de spécification interactifs

L'objectif des outils de spécification interactive d'interfaces est la construction de prototypes d'interface sans programmation [Coutaz 90]. Ils permettent au concepteur d'interface :
- de créer des tableaux de bord (ou formulaires) par assemblage d'objets de présentation prédéfinis.
- de décrire les enchaînements entre les tableaux de bord.
- d'associer les éléments de l'interface aux concepts de l'application.

Cette classe d'outils s'appuie sur :
- le principe de la séparation modulaire entre l'application et la présentation.
- l'existence d'une base d'objets interactifs d'usage général tels que les boutons, la barre de défilement, les menus, les champs de saisie...

L'idée de base est de permettre au concepteur de l'interface de placer sur l'écran des labels, du texte, des boutons, des icônes (donc les objets de présentation) là où l'utilisateur final du système les verra à l'exécution. A chaque objet sensible aux dispositifs de commande, il pourra associer une fonction de l'application qui sera exécutée lorsque l'utilisateur agira sur l'objet.

"Les outils de spécification apportent une réponse au problème de l'apprentissage d'un langage par un habillage graphique de la manipulation directe" [Coutaz 90].

Conclusion

Tout comme pour la conception architecturale, la spécification des interfaces homme-machine (IHM) reste un art plus qu'une science.
Le comportement d'une application interactive est, nous l'avons vu, piloté par des événements conceptuellement asynchrones et de nature différente. L'expérience montre que la description de ces événements et de leurs conditions d'occurrence est une tâche difficile. Il est important de disposer de formalismes de spécification adaptés à la description du caractère dynamique des interfaces homme-machine.

En génie logiciel, les techniques de spécification sont nombreuses : spécification par assertions, les types abstraits et la spécification algébrique, spécification prédicative. Cependant ces techniques restent essentiellement utilisées pour la spécification de modules fonctionnelles. Utilisées pour décrire l'IHM et son dialogue, elles deviennent vite trop lourdes et trop complexes.
Plus récents sont les formalismes pour l'expression des IHM et du dialogue. On trouve deux techniques essentielles : les automates (les diagrammes état-transition, les réseaux de Pétri) et les formalismes à événements.

La technique des diagrammes est bien maîtrisée mais il faut savoir l'appliquer correctement de façon à tenir compte de tous les choix possibles qu'il faut laisser à l'utilisateur. Le système doit être capable dans chaque état, de répondre avec précision à toutes les actions de l'utilisateur. Reportée à la spécification, cette condition se traduit par la spécification de toutes les actions utilisateur imaginables dans un état donné. Nous avons évoqué le caractère fastidieux de cette tâche et l'explosion combinatoire du graphe qui en résulte. La conséquence étant la spécification d'un dialogue séquentiel, ou l'utilisateur a comme seul choix le cheminement restreint dans le diagramme.

La notion de traitant d'événements et de langage à événements est utilisée depuis longtemps dans les systèmes d'exploitation. Elle a l'avantage d'organiser les traitements d'un système en une multitude d'agents coopérants prêts à régir aux événements. C'est sur ce modèle à événement qu'est fondé le langage de règles développé par I.Provot et B.Sacré.
Certes, il reste moins facile à comprendre et à lire que les diagrammes de transition mais sa puissance d'expression est plus grande que celle des diagrammes état-transition.

Les caractéristiques de ce langage sont :
- l'indépendance par rapport à l'outil de réalisation
- l'orientation objet utilisée pour supporter des dialogues asynchrones et multi-fils
- l'extensibilité, puisqu'on peut facilement étendre l'ensemble des classes d'objets interactifs
- l'utilisation d'une approche orientée événement et règles pour la définition du comportement des objets de dialogues.

Dans une seconde étape, il conviendrait de supporter le langage textuel par un langage graphique équivalent. On pourrait envisager d'utiliser, comme le fait Jacob [Jacob 86], le formalisme des diagrammes état-transition pour la description du comportement des classes d'objets de dialogue, ce qui faciliterait l'écriture des règles et la lisibilité des spécifications.

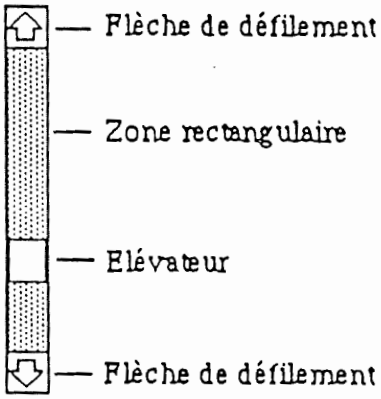
A partir du langage de règles, on pourrait aussi envisager une génération automatique ou plus exactement un outil de génération d'interfaces, s'appuyant sur ce langage.

Annexe 1 : Objets Interactifs

Barre de défilement

Contrôle de forme rectangulaire juxtaposé à la droite (gauche) ou au bas (haut) d'une fenêtre ou d'une liste dont la taille ne permet pas de visualiser l'entièreté de l'information. La barre de défilement permet à l'utilisateur de modifier la partie visible de l'information à l'intérieur de la fenêtre ou de la liste.

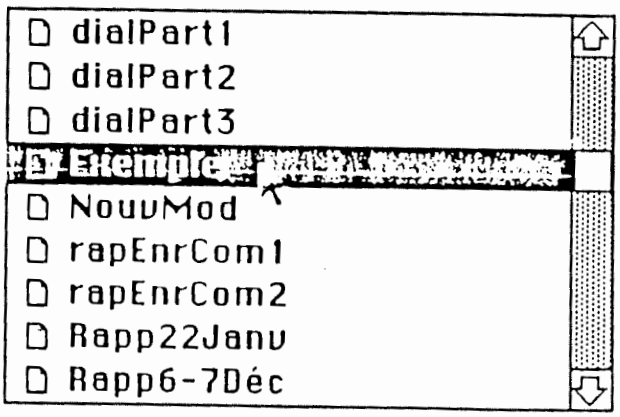
Représentation graphique :



List Box

Liste déroulable d'éléments représentant des choix parmi lesquels l'utilisateur a la possibilité de faire une ou plusieurs sélections.

Représentation graphique :



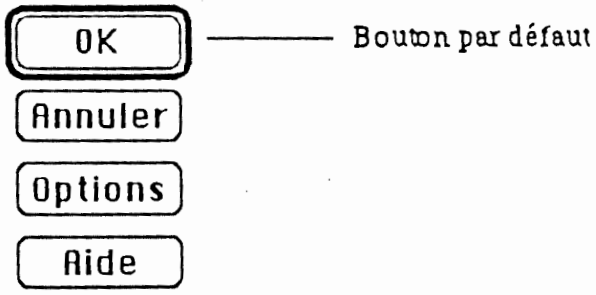
Contrôle

Objet graphique apparaissant habituellement dans une boîte de dialogue. Les contrôles permettent à l'utilisateur d'interagir avec une application en introduisant des données ou en déclenchant une action dont l'exécution produit des résultats instantanés audibles ou visibles.
Il existe plusieurs type de contrôle :
- bouton
- bouton-radio
- icône-radio
- check-box
- cadran
- champ d'édition (edit-box)

Bouton

Contrôle utilisé pour déclencher une action dont le résultat est décrit par un label. Un seul bouton peut être activé à la fois. Il est possible de définir un bouton par défaut pour indiquer le choix le plus probable. Le bouton par défaut est entouré d'une double ligne.

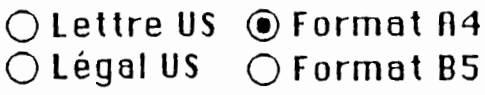
Représentation graphique :



Bouton radio

Mécanisme de sélection typiquement présenté par groupe. Les boutons-radio permettent à l'utilisateur de faire un choix parmi un groupe d'options. Les options d'un même groupe sont mutuellement exclusives et, à tout moment, un bouton-radio et un seul est sélectionné par groupe.

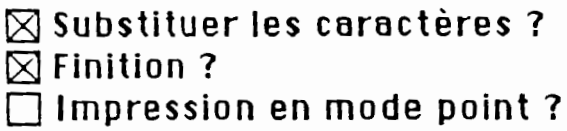
Représentation graphique :



Check-Box

Mécanisme de sélection typiquement présenté par groupe. Les check-Box permettent à l'utilisateur de faire un choix parmi un groupe d'options. Les options d'un même groupe ne sont pas mutuellement exclusives et, à tout moment, une ou plusieurs check-Box est sélectionnées par groupe.

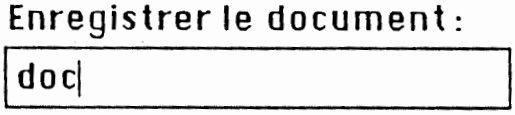
Représentation graphique :



Champ d'édition (Edit-Box)

Contrôle permettant à l'utilisateur d'introduire et de manipuler des chaînes de caractères en utilisant le clavier. Il est constitué d'un label et d'une zone rectangulaire dans laquelle le texte peut être introduit.

Représentation graphique :



Boîte de dialogue

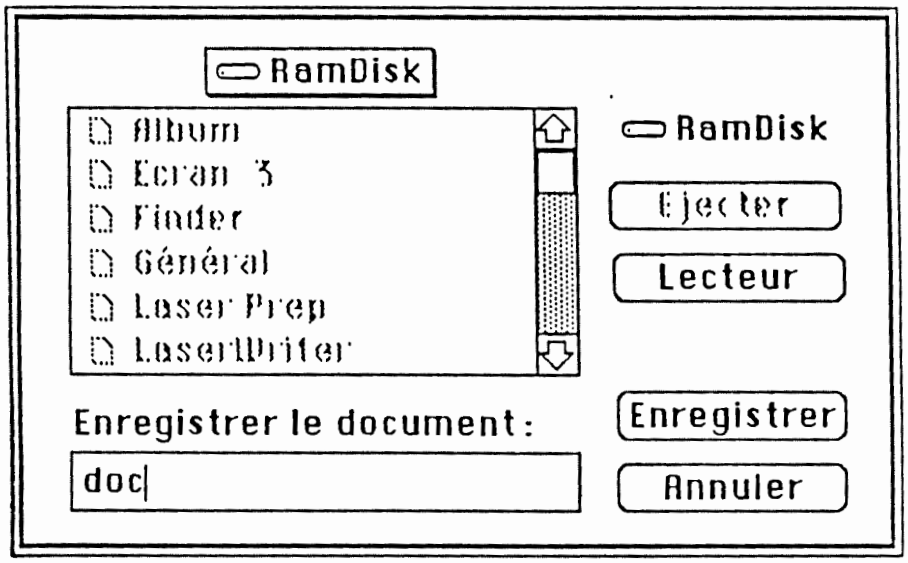
Fenêtre d'un type particulier dans laquelle sont affichés des messages ou des contrôles.

Une boîte de dialogue est utilisée pour fournir de l'information à l'utilisateur ou pour permettre à celui-ci d'introduire ou de modifier des données ou des paramètres à propos d'une commande qu'il désire exécuter. Une boîte de dialogue contient le plus souvent au moins un des deux boutons suivants : ok utilisé pour confirmer une commande et cancel pour la supprimer. Il existe deux types de boîtes de dialogue : les modales et les non modales.

Boîte de dialogue modale

Boîte de dialogue d'un type particulier exigeant une réponse immédiate de la part de l'utilisateur. Cela signifie qu'il est incapable de faire quoi que ce soit d'autre tant qu'il n'a pas introduit les informations demandées.

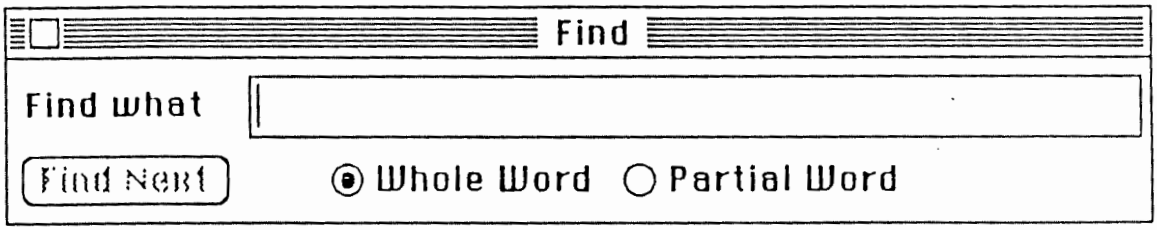
Représentation graphique :



Boîte de dialogue non modale

Boîte de dialogue d'un type particulier n'exigeant pas une réponse immédiate de la part de l'utilisateur.Il peut par exemple continuer à travailler dans la fenêtre-document dans laquelle il se trouvait avant de déclencher la commande qui a fait apparaître la boîte de dialogue.

Représentation graphique :

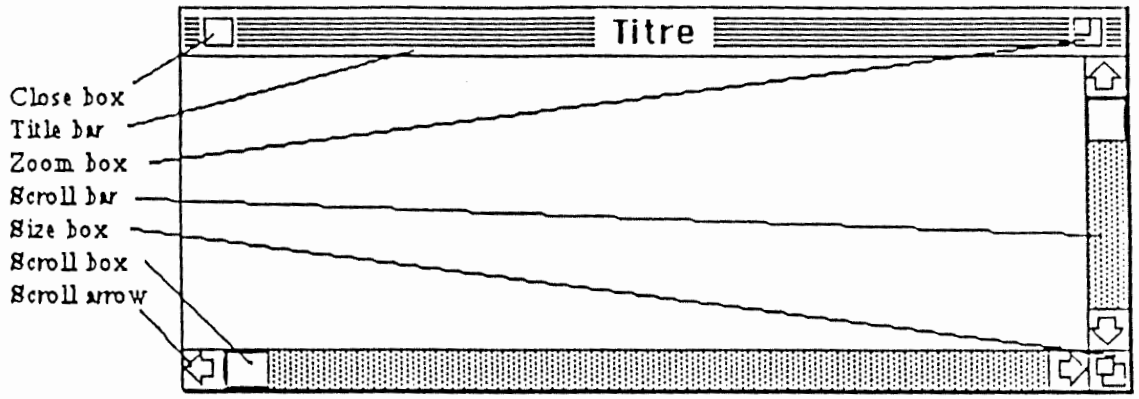


Fenêtre

Zone rectangulaire apparaissant à l'écran, servant à visualiser les données déterminées par l'application et permettant à l'utilisateur à l'utilisateur d'interagir avec celles-ci.

Une application peut afficher une ou plusieurs fenêtres. Lorsque plusieurs fenêtres sont simultanément visibles à l'écran, elles se superposent. L'utilisateur ne travaille que sur une fenêtre à la fois. C'est la fenêtre active.

Représentation graphique :



Menu

Le concept de menu fournit à l'utilisateur la possibilité de déclencher une série de commandes qui concernent l'application avec laquelle il travaille ou d'établir des états. Un menu consiste en un nom et une liste de commandes appelées items de menu. Un item de menu est éventuellement accompagné d'un accélérateur, combinaison de touches-clavier permettant de sélectionner un item sans passer par le processus de sélection par la souris.

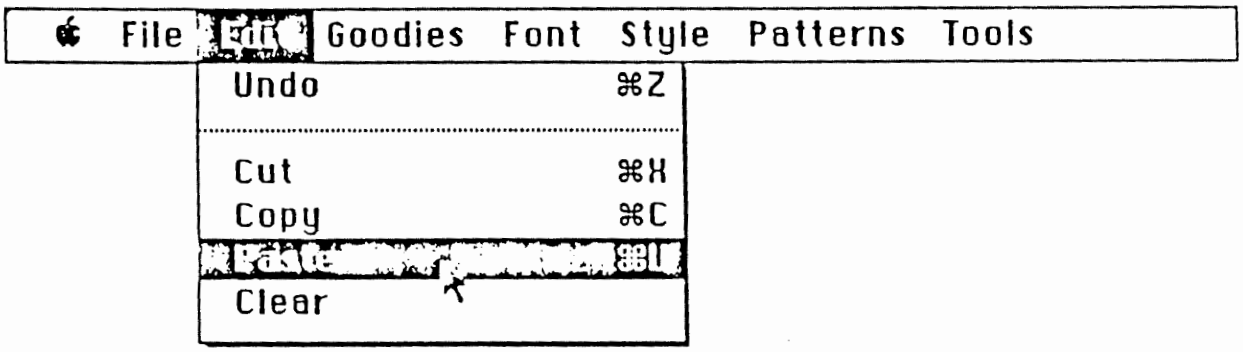
Il existe différents types de menus :

- le menu déroulant
- le sous-menu
- le pop-up menu

Le menu déroulant

Type de menu le plus classique. A tout moment, les noms des menus utilisables sont affichés dans une barre de menu. C'est une barre horizontale à l'écran réservée à la présentation des menus.

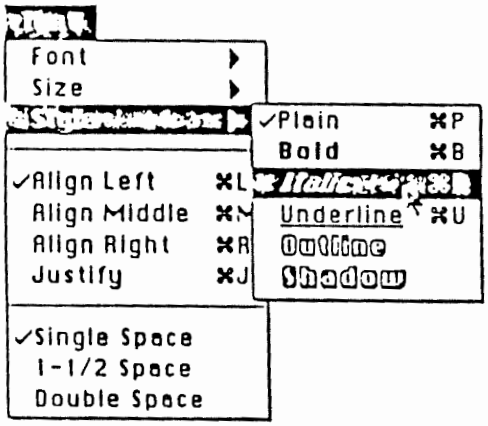
Représentation graphique :



Sous-menu

Menu qui n'apparaît pas dans la barre de menu. La notion de sous-menu permet de créer une hiérarchie de menus et de fournir à l'utilisateur le moyen de préciser de plus en plus la commande qu'il désire exécuter.

Représentation graphique :



Annexe 2 : Spécification de l'exemple

La spécification qui suit est tirée de [Provot et Sacré 90].

Le composant Interaction

Les classes d'objets interactifs prédéfinies de l'exemple

OBJET INTERACTIF

Toutes les classes d'objets interactifs possèdent les primitives suivantes:

- pr_création — crée une instance d'une classe donnée
- pr_affichage — affiche l'objet interactif à l'écran
- pr_effacement — efface l'objet interactif de l'écran
- pr_obt_attr — retrouve la valeur d'un attribut
- pr_chg_attr — change la valeur d'un attribut
- pr_suppression — supprime l'objet interactif
- pr_activation — rend l'objet interactif manipulable
- pr_désactivation — rend l'objet interactif non manipulable
- pr_gr_activation (groupe)
  équivaut à
  pr_activation (groupe.élément1) et pr_activation (groupe.élément2) et ...
- pr_gr_désactivation (groupe)
  équivaut à
  pr_désactivation (groupe.élt1) et pr_désactivation (groupe.élt2) et ...

EDIT BOX

Primitives

- pr_ed_initialisation affecte des valeurs par défaut à l'edit
- pr_ed_réinitialisation remet l'edit à blanc
- pr_ed_obt_contenu retourne le contenu de l'edit
- pr_ed_exist_contenu détermine si l'edit possède un contenu
- pr_ed_plus_de_contenu détermine s'il n'y a plus rien dans l'edit
- pr_ed_inverse met en évidence une partie du contenu de l'edit

Evénements

- ev_ed_exist_contenu généré lorsque l'utilisateur frappe un caractère dans une boîte d'édition vide
- ev_ed_chg_contenu généré lorsque l'utilisateur change le contenu de la boîte d'édition
- ev_ed_plus_de_contenu généré lorsque l'utilisateur efface le contenu d'une boîte d'édition
- ev_ed_modif généré lorsque le contenu de l'edit est modifié
- ev_ed_sortie généré quand le curseur sort de l'edit
- ev_gr_ed_exist_contenu (groupe, opérateur)
  équivaut à
  case opérateur
    OU: ev_ed_exist_contenu (groupe.élt1)
        ou ev_ed_exist_contenu (groupe.élt2)
        ou ...
    ET: (ev_ed_exist_contenu (groupe.élt1)
        et pr_ed_exist_contenu (groupe.élt2)
        et pr_ed_exist_contenu (groupe.élt3) et ...)
        (ev_ed_exist_contenu (groupe.élt2)
        et pr_ed_exist_contenu (groupe.élt1)
        et pr_ed_exist_contenu (groupe.élt3) et ...)
      ou _
- ev_gr_ed_plus_de_contenu (groupe, opérateur)
  équivaut à
  case opérateur
    OU: ev_ed_plus_de_contenu (groupe.élt1)
        ou ev_ed_plus_de_contenu (groupe.élt2)
        ou ...
    ET: (ev_ed_plus_de_contenu (groupe.élt1)
        et pr_ed_plus_de_contenu (groupe.élt2)
        et pr_ed_plus_de_contenu (groupe.élt3) et ...)

ou
(ev_ed_plus_de_contenu (groupe.élt2)
et pr_ed_plus_de_contenu (groupe.élt1)
et pr_ed_plus_de_contenu (groupe.élt3) et ...)
ou _

- ev_gr_chg_contenu (groupe)
  équivaut à
  ev_ed_chg_contenu (groupe.élt1)
  ou ev_ed_chg_contenu (groupe.élt2)
  ou ...

LABEL

Primitives

- pr_lb_initialisation — affecte des valeurs par défaut au label
- pr_lb_réinitialisation — remet le label à blanc
- pr_lb_chg_contenu — modifie le contenu du label

BOUTON

Evénements

- ev_bt_select — généré lorsque le bouton est sélectionné

CURSEUR

Primitives

- pr_placer_crs — place le curseur dans un objet interactif donné

LIST BOX

Primitives

- pr_lbx_ajout_élem ajoute un élément dans la liste
- pr_lbx_obt_élem permet d'obtenir l'élément sélectionné

Evénements

- ev_lbx_select généré lorsqu'un élément de la liste est sélectionné
- ev_lbx_désélect généré lorsqu'un élément de la liste est désélectionné

FENETRE

Evénements

- ev_fe_affichage généré lorsque la fenêtre vient d'être affichée
- ev_fe_effacement généré lorsque la fenêtre vient d'être effacée
- ev_fe_ouverture généré lorsque la fenêtre vient d'être ouverte
- ev_fe_fermeture généré lorsque la fenêtre vient d'être fermée

Le composant Application

PRODUIT

Primitives

- pr_validation_num_produit (numéro) -> booléen
  détermine si un numéro de produit est valide ou non
- pr_validation_quantité_produit (numéro, quantité) -> booléen
  détermine si une quantité restante d'un produit de numéro donné et supérieure ou égale à la quantité commandée
- pr_obt_info_produit (numéro) -> PRODUIT
  renvoit les informations concernant un produit de numéro donné dans une structure de type PRODUIT qui contient le libellé et le prix

CLIENT

Primitives

- pr_validation_num_client (numéro) -> booléen
  détermine si un numéro de client est valide ou non
- pr_validation_cli_client (CLI) -> booléen
  détermine si un nom et un prénom de client sont valides ou non
- pr_obt_info_client (numéro) -> CLI
  renvoit les informations concernant un client de numéro donné dans une structure qui contient le nom et le prénom du client
- pr_obt_lst_client -> CLI
  renvoit, par itération, la liste des clients existant dans une structure qui contient le nom et le prénom
- pr_obt_num_client (CLI) -> numéro
  retourne le numéro d'un client dont on connait le nom et le prénom
- pr_modif_adresse_client (numéro, ADRESSE)
  modifie l'adresse d'un client existant
- pr_enregistrer_client (CLIENT) -> numéro
  enregistre un nouveau client

Evénements

- ev_client_modifié
  généré lors de toute modification apportée à la liste des clients (ajout, suppression) ou lors de toute modification à des clients existants (ex: changement de nom)

COMMANDE

Primitives

- pr_validation_commande (COMMANDE, numéro_client) -> booléen
  détermine si une commande passée par un client de numéro numéro_client est valide ou non
- pr_enregistrement_commande (COMMANDE, numéro_client) ->booléen
  enregistre une nouvelle commande passée par un client de numéro numéro_client et validée

Le composant Dialogue

Introduction: explication du cannevas

Description

La description contient l'objectif assigné à la classe d'objets de dialogue

Description de l'interface

L'interface correspond à la partie visible d'une classe d'objets de dialogue. Elle comprend:

- les **primitives** offertes par la classe d'objet de dialogue
- les **événements** générés par l'objet de dialogue

Les classes d'objets de dialogue utilisées

Les classes d'objets interactifs utilisées

Les classes d'objets de l'application utilisées

Réalisation du **comportement**

- déclaration des variables
- réaction aux **primitives**
- réaction aux **événements** des objets de dialogue **fils**

Les classes d'objets de l'exemple

1. La saisie de la commande

Description

Saisie du numéro de produit

Il y a validation du numéro de produit dès que le champ perd le curseur. Si le numéro correspond à un numéro de produit existant, il y a affichage du libellé et du prix unitaire et le champ quantité est activé.
Si le numéro n'existe pas, il y affichage d'un message d'erreur, inversion du contenu du champ erroné et placement du curseur à l'intérieur.

Saisie de la quantité commandée

Le numéro du produit doit être saisi avant d'introduire la quantité.
Il y a validation de la quantité dès que le champ perd le curseur. Si la quantité commandée est inférieure ou égale à la quantité en stock, la ligne de commande est valide. Le montant de la ligne et le montant total sont calculés et affichés.
Si la quantité est supérieure à la quantité en stock, il y affichage d'un message d'erreur, inversion du contenu du champ erroné et placement du curseur à l'intérieur.

Modification du numéro de produit

Si l'utilisateur modifie le numéro de produit d'une ligne de commande **valide**, le libellé, le prix unitaire, le montant de la ligne et la quantité commandée sont effacés et on modifie le montant total en conséquence.

Modification de la quantité commandée

Si l'utilisateur modifie la quantité commandée d'une ligne de commande **valide**, le montant de la ligne et le montant total sont recalculés.

Terminaison de la saisie de la commande

La saisie de la commande est considérée comme terminée si l'on dispose d'un numéro de produit et d'une quantité valides.

SAISIE_LIGNE_DE_CDE

Description

SAISIE_LIGNE_DE_CDE saisit une ligne de commande valide c'est-à-dire composée d'un numéro de produit valide et d'une quantité commandée inférieure ou égale à la quantité disponible.

Description de l'interface

Primitives

- pr_création — création d'une instance de la classe
- pr_réinitialisation — remise à blanc d'une ligne de commande
- pr_obt_contenu — permet d'obtenir le contenu de la ligne de commande, c'est-à-dire numéro et quantité
- pr_ligne_cde_valide — détermine si une ligne de commande est valide (numéro de produit existant et quantité suffisante en stock) ou non
- pr_obt_montant — permet d'obtenir le montant d'une ligne de commande valide

Evénements

- ev_ligne_cde_valide — l'utilisateur a introduit une ligne de commande valide
- ev_ligne_cde_modifiée — correspond au passage d'un état dans lequel la ligne de commande est valide à un état où elle est en cours de modification.

Les classes d'objets de dialogue utilisées

Les classes d'objets interactifs utilisées

EDIT, LABEL

Les classes d'objets de l'application utilisées

PRODUIT

Réalisation du comportement

Déclaration des variables

```
EDIT     ed_num, ed_quantité
LABEL    lb_libellé, lb_prix_unitaire, lb_montant, lb_msge
PRODUIT  prod
           libellé
           prix_unitaire
```

```
INFO_LIGNE  info_ligne
                num
                quantité
ENTIER      num, quantité, montant
BOOLEEN     ligne_cde_valide /* true si le numéro de produit est valide
            et si la  quantité commandée est inférieure ou égale à la
            quantité disponible */
```

Réaction aux primitives

```
IF  pr_création
THEN    /* on suppose qu'après l'exécution de cette primitive,
        toutes les variables sont correctement initialisées  */
        pr_ed_désactivation (ed_quantité)
        ligne_cde_valide <- False

IF  pr_réinitialisation
THEN    pr_ed_réinitialisation (ed_num)
        pr_ed_réinitialisation (ed_quantité)
        pr_lb_réinitialisation (lb_libellé)
        pr_lb_réinitialisation (lb_pri_unitaire)
        pr_lb_réinitialisation (lb_montant)

IF  pr_obt_contenu
THEN    info_ligne.num <- pr_ed_obt_contenu (ed_num)
        info_ligne.quantité <- pr_ed_obt_contenu (ed_quantité)
        return (info_ligne)

IF  pr_ligne_cde_valide
THEN    return (ligne_cde_valide)

IF  pr_obt_montant
THEN    return (pr_lb_obt_contenu (lb_montant))
```

Réaction aux événements

IF **ev_ed_sortie (ed_num)**
THEN num <- pr_ed_obt_contenu (ed_num)
IF pr_validation_num_produit (num) = True
THEN prod <- pr_obt_info_produit (num)
pr_lb_chg_contenu (lb_libellé, prod.libellé)
pr_lb_chg_contenu (lb_prix_unitaire, prod.prix_unitaire)
pr_activation (ed_quantité)
ELSE pr_lb_chg_contenu(lb_msge,"Numéro incorrect")
pr_ed_inverse (ed_num)
pr_placer_crs (ed_num)

IF **ev_ed_sortie (ed_quantité)**
/* PRE: on dispose du prix unitaire dans prod */
THEN quantité <- pr_ed_obt_contenu (ed_quantité)
num <- pr_ed_obt_contenu (ed_num)
IF pr_validation_quantité_produit (num, quantité) = True
THEN montant <- prod.prix_unitaire * quantité
pr_lb_chg_contenu (lb_montant, montant)
ligne_cde_valide <- True
générer (ev_ligne_cde_valide)

IF **(ev_ed_modif (ed_num)) et (ligne_cde_valide = True)**
THEN pr_lb_réinitialisation (lb_libellé, lb_prix_unitaire, lb_montant)
pr_ed_réinitialisation (ed_quantité)
pr_ed_désactivation (ed_quantité)
ligne_cde_valide <- False
générer (ev_ligne_cde_modifiée)

IF **(ev_ed_modif (ed_quantité)) et (ligne_cde_valide = True)**
THEN ligne_cde_valide <- False
générer (ev_ligne_cde_modifiée)

AFFICHAGE_MONTANT_TOTAL_CDE

Description

AFFICHAGE_MONTANT_TOTAL_CDE gère le calcul et l'affichage du montant total de la commande en cours de saisie c'est-à-dire le total de chaque ligne de commande valide.

Description de l'interface

Primitives

- pr_création — création d'une instance de la classe
- pr_initialisation — initialisation du montant total de la commande
- pr_ajout_montant — ajoute au montant total de la commande, le montant d'une ligne de commande
- pr_soustr_montant — soustrait du montant total de la commande, le montant d'une ligne de commande
- pr_obt_montant_total — permet d'obtenir le montant total de la commande

Les classes d'objets de dialogue utilisées

Les classes d'objets interactifs utilisées

LABEL

Les classes d'objets de l'application utilisées

Réalisation du comportement

Déclaration des variables

```
LABEL      lb_montant_total
ENTIER     montant_total, montant
```

Réaction aux primitives

IF **pr_création**
THEN /* on suppose qu'après l'exécution de cette primitive,
toutes les variables sont correctement initialisées */

IF **pr_initialisation**
THEN pr_lb_chg_contenu (lb_montant_total, 0)
montant_total <- 0
IF **pr_soustr_montant (montant)**
THEN montant_total <- montant_total - montant
pr_lb_chg_contenu (lb_montant_total, montant_total)

IF **pr_ajout_montant (montant)**
THEN montant_total <- montant_total + montant
pr_lb_chg_contenu (lb_montant_total, montant_total)

IF **pr_obt_montant_total**
THEN return (montant_total)

SAISIE_CDE

Description

SAISIE_CDE saisit et affiche n lignes de commandes valides.

Description de l'interface

Primitives

- pr_création création d'une instance de la classe
- pr_initialisation initialisation de la commande
- pr_obt_cde permet d'obtenir l'ensemble des lignes de commande valides
- pr_cde_existante détermine s'il existe au moins une ligne de commande valide

Evénements

- ev_cde_existante événement généré lorsqu'une commande contient au moins une ligne de commande valide
- ev_cde_inexistante événement généré lorsqu'une commande ne contient pas de ligne de commande valide

Les classes d'objets de dialogue utilisées

SAISIE_LIGNE_DE_CDE, AFFICHAGE_MONTANT_TOTAL_CDE

Les classes d'objets interactifs utilisées

Les classes d'objets de l'application utilisées

Réalisation du comportement

Déclaration des variables

```
ENTIERmontant
SAISIE_LIGNE_DE-CDE          saisie_ligne_de_cde
SAISIE_LIGNE_DE-CDE          saisie_ligne_de_cde_1
_
SAISIE_LIGNE_DE-CDE          saisie_ligne_de_cde_20
AFFICHAGE_MONTANT_TOTAL_CDE  montant_total
TAB [1..20] of INFO_LIGNE    ligne_cde  /*ligne_cde désigne un tableau
                             de 20 éléments de type INFO_LIGNE*/
```

Réaction aux primitives

```
IF pr_création
THEN  /* on suppose qu'après l'exécution de cette primitive,
        toutes les variables sont correctement initialisées  */
      pr_création(AFFICHAGE_MONTANT_TOTAL_CDE,
SAISIE_LIGNE_DE_CDE)

IF pr_initialisation
THEN  pr_ initialisation (saisie_ligne_de_cde_1)
      pr_ initialisation (saisie_ligne_de_cde_20)
      pr_initialisation (montant_total)
```

```
IF pr_obt_cde
THEN  IF pr_ligne_cde_valide (saisie_ligne_de_cde_1)
      THEN  ligne_cde [1] <- pr_obt_contenu (saisie_ligne_de_cde_1)
      ELSE  ligne_cde [1] <- NULL
      _
      IF pr_ligne_cde_valide (saisie_ligne_de_cde_20)
      THEN  ligne_cde [20] <- pr_obt_contenu(saisie_ligne_de_cde_20)
      ELSE  ligne_cde [20] <- NULL


IF pr_cde_existante
THEN  IF (pr_ligne_cde_valide (ligne_cde [1]) = True)
          ou (pr_ligne_cde_valide (ligne_cde [2]) = True)
              ...
          ou (pr_ligne_cde_valide (ligne_cde [20]) = True)
      THEN  return (True)
      ELSE  return (False)
```

Réaction aux événements

```
IF ev_ligne_cde_valide (saisie_ligne_de_cde)
THEN  montant <- pr_obt_montant (saisie_ligne_de_cde)
      pr_ajout_montant (montant_total, montant)
      générer (ev_cde_existante)

IF ev_ligne_cde_modifiée (saisie_ligne_de_cde)
THEN  montant <- pr_obt_montant (saisie_ligne_de_cde)
      pr_soustr_montant (montant_total, montant)
      IF pr_ligne_cde_valide (saisie_ligne_de_cde_1)
      THEN  générer (ev_cde_existante)
            return
      ...
      IF pr_ligne_cde_valide (saisie_ligne_de_cde_20)
      THEN  générer (ev_cde_existante)
            return
      générer (ev_cde_inexistante)
```

2. La saisie du client

Description

Hypothèse: un client est identifié par son numéro ou par le couple (nom, prénom).

Saisie du numéro du client

Il y a activation du bouton "valider".
Si le numéro saisi correspond à un numéro de client existant, il y a affichage du nom, du prénom et de l'adresse du client et activation du bouton "modifier l'adresse".
Si le numéro saisi ne correspond pas à un numéro de client existant, il y affichage d'un message d'erreur, inversion du contenu du champ erroné et placement du curseur à l'intérieur.

Saisie du nom et du prénom du client

Il y a activation du bouton "valider".
Si le nom et le prénom saisis correspondent à ceux d'un client existant, il y a affichage du numéro et de l'adresse du client et activation du bouton "modifier l'adresse".
Si le nom et le prénom saisis ne correspondent pas à ceux d'un client existant, il y affichage d'un message d'erreur, inversion du contenu des champs nom et prénom et placement du curseur dans le champ nom.

Saisie du nom et du prénom du client

Il y a activation du bouton "enregistrer".

Sélection du bouton "clients"

L'utilisateur sélectionne le bouton "clients". On affiche la liste des clients existants.
Si il sélectionne un élément de la liste, les champs nom et prénom sont garnis automatiquement.

Bouton "valider"

Objectif: valider un client connaissant son numéro ou son nom et son prénom
Activable quand le numéro est saisi ou quand le nom et le prénom sont saisis
Désactivable quand les champs numéro, nom et prénom sont vides.

Bouton "enregistrer"

Objectif: enregistrer un client connaissant son nom, son prénom et son adresse
Activable quand le nom, le prénom et l'adresse sont saisis
Désactivable quand les champs nom, prénom et adresse sont vides.

Bouton "modifier l'adresse"

Objectif: modifier l'adresse d'un client existant
Activable quand un client a été validé
Désactivé tant qu'un client n'a pas été validé.

Bouton "clients"

Objectif: afficher la liste des clients (nom et prénom) se trouvant dans la BD
Activé tant que la liste de client n'a pas été affichée
Désactivable quand la liste de client est affichée.

Terminaison de la saisie du client

La saisie du client est considérée comme terminée si l'on dispose d'un client valide.

GESTION_EXCLUSION_CHAMPS

Description

GESTION_EXCLUSION_CHAMPS interdit la saisie simultanée du numéro de client et du reste des informations sur le client (nom, prénom, rue, numéro de rue, localité, code).

Description de l'interface

Primitives

- pr_création création d'une instance de la classe

Les classes d'objets de dialogue utilisées

Les classes d'objets interactifs utilisées

EDIT

Les classes d'objets de l'application utilisées

Réalisation du comportement

Déclaration des variables

EDITed_num_cli, ed_nom, ed_prénom, ed_num_rue, ed_rue, ed_localité,
GROUPE_INFO_CLI gr_info_cli
ed_nom
ed_prénom
gr_adresse
GROUPE_ADRESSE gr_adresse
ed_rue
ed_num_rue
ed_localité
ed_code

Réaction aux primitives

IF **pr_création**
THEN /* on suppose qu'après l'exécution de cette primitive,
toutes les variables sont correctement initialisées */

Réaction aux événements

IF **ev_ed_exist_contenu (ed_num_cli)**
THEN pr_gr_désactivation (gr_info_cli)

IF **ev_ed_plus_de_contenu (ed_num_cli)**
THEN pr_gr_activation (gr_info_cli)

IF **ev_gr_ed_exist_contenu (gr_info_cli, OU)**
/* l'utilisateur a saisi au moins un champ du groupe */
THEN pr_ed_désactivation (ed_num_cli)

IF **ev_gr_ed_plus_de_contenu (gr_info_cli, ET)**
/* rien dans aucun champ du groupe */
THEN pr_ed_activation (ed_num_cli)

GESTION_CLIENTS_EXISTANTS

Description

GESTION_CLIENTS_EXISTANTS affiche les clients existant dans la base de donnée.

Description de l'interface

Primitives

- pr_création crée une instance de la classe
- pr_sélection_client détermine si un client est sélectionné ou pas

Evénements

- ev_sélection_client généré lorsqu'un client est sélectionné
- ev_non_sélection_client généré lorsqu'il n'y a plus de client sélectionné

Les classes d'objets de dialogue utilisées

Les classes d'objets interactifs utilisées

BOUTON, LISTBOX, FENETRE

Les classes d'objets de l'application utilisées

CLIENT

Réalisation du comportement

Déclaration des variables

```
BOUTON          bt_clients
FENETRE         fe_clients
LISTBOX         lbx_clients
STRING [1..50]  élem
STRUCT          cli
                   nom
                   prénom
```

Réaction aux primitives

IF **pr_création**
THEN /* on suppose qu'après l'exécution de cette primitive,
toutes les variables sont correctement initialisées */
cli.nom <- NULL
cli.prénom <- NULL
CLIENT (GESTION_CLIENTS_EXISTANTS, ev_client_modifié)
cli <- pr_obt_lst_client
WHILE (cli <> NULL)
élem <- fusion (cli)
pr_lbx_ajout_élem (élem)
cli <- pr_obt_lst_client

IF **pr_sélection_client**
/* renvoit le client actuellement sélectionné*/
THEN return (cli)

Réaction aux événements

IF **ev_fe_fermeture (fe_clients)**
THEN pr_activation (bt_clients)

IF **ev_bt_select (bt_clients)**
THEN pr_désactivation (bt_clients)
pr_afficher_fenêtre (fe_clients)

IF **ev_lbx_select (lbx_clients)**
THEN élem <- pr_lbx_obt_élem
scind (élem, cli.nom, cli.prénom)
générer (ev_sélection_client)

IF **ev_lbx_désélect (lbx_clients)**
THEN cli.nom <- NULL
cli.prénom <- NULL
générer (ev_non_sélection_client)

IF **ev_client_modifié**
THEN cli <- pr_obt_lst_client
WHILE (cli <> NULL)
élem <- fusion (cli)
pr_lbx_ajout_élem (élem)
cli <- pr_obt_lst_client

SAISIE_CLIENT

Description

SAISIE_CLIENT saisit un numéro de client valide qui résulte de l'identification d'un numéro de client existant ou de l'enregistrement d'un nouveau client, avec la possibilité de modifier l'adresse.

Description de l'interface

Primitives

- pr_création — crée une instance de la classe
- pr_initialisation — initialise le client
- pr_obt_client — retourne le numéro d'un client existant
- pr_client_existant — détermine si un client valide est disponible

Evénements

- ev_client_existant — généré lorsqu'un client valide est identifié
- ev_client_inexistant — généré lorsqu'il n'y a plus de client valide identifié

Les classes d'objets de dialogue utilisées

GESTION_CLIENTS_EXISTANTS, GESTION_EXCLUSION_CHAMPS

Les classes d'objets interactifs utilisées

BOUTON, EDIT, LABEL

Les classes d'objets de l'application utilisées

CLIENT

Réalisation du comportement

Déclaration des variables

ENTIER num
BOOLEEN client_existant
BOUTON bt_valider, bt_modifier, bt_enregistrer

```
EDIT    ed_num_cli, ed_nom, ed_prénom, ed_rue, ed_code, ed_localité,
                ed_num_rue
LABEL           lb_msge
GROUPE_NOM  gr_nom
                    ed_nom
                    ed_prénom
GROUPE_ADRESSE      gr_adresse
                            ed_rue
                            ed_num_rue
                            ed_localité
                            ed_code
GROUPE_INFO_CLI     gr_info_cli
                            gr_nom
                            gr_adresse
STRUCT          cli
                    nom
                    prénom
STRUCT          adresse
                    rue
                    num_rue
                    code
                    localité
STRUCT          client
                    cli
                    adresse
```

Réaction aux primitives

IF **pr_création**
THEN /* on suppose qu'après l'exécution de cette primitive,
toutes les variables sont correctement initialisées */
pr_création (GESTION_EXCLUSION_CHAMPS)

IF **pr_obt_client**
THEN num <- pr_ed_obt_contenu (ed_num_cli)
return (num)

Réaction aux événements

IF **ev_ed_exist_contenu (ed_num_cli)**
THEN pr_activation (bt_valider)

```
IF (ev_gr_ed_exist_contenu (gr_nom, ET)
THEN   pr_activation (bt_valider)

IF ev_ed_plus_de_contenu (ed_num_cli)
THEN   pr_désactivation (bt_valider)

IF ev_gr_ed_plus_de_contenu (gr_nom, OU)
THEN   pr_désactivation (bt_valider)


IF ev_gr_ed_exist_contenu (gr_info_cli, ET)
THEN   pr_bt_activation (bt_enregistrer)

IF ev_gr_ed_plus_de_contenu (gr_info_cli, OU)
THEN   pr_bt_désactivation (bt_enregistrer)

IF ev_bt_select (bt_valider)
THEN   IF pr_ed_exist_contenu (ed_num) = True
  [        THEN   num <- pr_ed_obt_contenu (ed_num_cli)
                  client_existant <- pr_validation_num_client (num)
           ELSE   cli.nom <- pr_ed_obt_contenu (ed_nom)
                  cli.prénom <- pr_ed_obt_contenu (ed_prénom)
                  client_existant <- pr_validation_cli_client (cli)
                  IF client_existant = True
                  THEN    num <- pr_obt_num_client (cli)
           IF client_existant = True
           THEN   client <- pr_obt_info_client (num)
                  pr_ed_chg_contenu <- (ed_num_cli, client.num)
                  pr_ed_chg_contenu <- (ed_nom, client.nom)
                  pr_ed_chg_contenu <- (ed_prénom, client.prénom)
                  pr_ed_chg_contenu <- (ed_rue, client.rue)
                  pr_ed_chg_contenu <- (ed_num_rue, client.rue)
                  pr_ed_chg_contenu <- (ed_code, client.code)
                  pr_ed_chg_contenu <- (ed_localité, client.localité)
                  pr_activation (bt_modifier)
                  générer (ev_client_existant)
           ELSE   pr_lb_chg_contenu (lb_msge, "client inexistant")
                  pr_désactivation (bt_modifier)
                  générer (ev_client_inexistant)
```

IF **ev_bt_select (bt_modifier)**
THEN /* modifier l'adresse d'un client */
num <- pr_ed_obt_contenu (ed_num_cli)
adresse.rue <- pr_ed_obt_contenu (ed_rue)
adresse.num_rue <- pr_ed_obt_contenu (ed_num_rue)
adresse.localité <- pr_ed_obt_contenu (ed_localité)
adresse.code <- pr_ed_obt_contenu (ed_code)
pr_modification_adresse (num, adresse)

IF **ev_selection_client (gestion_clients_existants)**
THEN cli <- pr_sélection_client
pr_ed_chg_contenu (ed_nom, cli.nom)
pr_ed_chg_contenu (ed_prénom, cli.prénom)

IF **ev_ed_chg_contenu (ed_num_cli) and client_existant**
THEN pr_gr_réinitialisation (gr_info_cli)
client_existant <- False
générer (ev_client_inexistant)

IF **ev_gr_ed_chg_contenu (gr_cli) and client_existant**
THEN pr_ed_réinitialisation (ed_num)
pr_gr_réinitialisation (gr_adresse)
client_existant <- False
générer (ev_client_inexistant)

IF **ev_bt_sélect (bt_enregistrer)**
THEN client.cli.nom <- pr_ed_obt_contenu (ed_nom)
client.cli.prénom <- pr_ed_obt_contenu (ed_prénom)
client.adresse.rue <- pr_ed_obt_contenu (ed_rue)
client.adresse.num_rue <- pr_ed_obt_contenu (ed_num_rue)
client.adresse.localité <- pr_ed_obt_contenu (ed_localité)
client.adresse.code <- pr_ed_obt_contenu (ed_code)
num <- pr_enregistrer_client (client)
IF num = 0
THEN pr_lb_chg_contenu (lb_msge, "client non enregistré")
ELSE pr_ed_chg_contenu (ed_num_cli, num)
générer (ev_client_existant)

IF **ev_gr_ed_plus_de_contenu (gr_adresse, OU)**
THEN pr_désactivation (bt_modifier)

IF **ev_gr_ed_exist_contenu (gr_adresse, ET)**
THEN pr_activation (bt_modifier) .

3. L'enregistrement de la command

Description

Bouton "valider la commande"

Objectif: valider la commande
Activable lorsqu'on dispose d'un client valide et d'au moins une ligne de commande valide
Désactivé tant qu'un client n'est pas valide ou tant que la commande ne contient pas au moins une ligne valide.

Bouton "Mémoriser la commande"

Objectif: mémoriser la commande
Activable lorsque la commande est valide
Désactivé tant qu'on ne dispose pas d'une commande valide

ENREGISTREMENT_CDE

Description

ENREGISTREMENT_CDE enregistre une commande valide pour un client existant.

Description de l'interface

Primitives

- pr_création crée une instance de la classe

Evénements

- ev_cde_enreg généré lorsque l'enregistrement d'une commande pour un client a été effectué

Les classes d'objets de dialogue utilisées

SAISIE_CLIENT, SAISIE_CDE

Les classes d'objets interactifs utilisées

BOUTON, LABEL

Les classes d'objets de l'application utilisées

COMMANDE

Réalisation du comportement

Déclaration

```
BOUTON          bt_valider_cde, bt_mémo_cde
LABEL           lb_msge
INF_TAB[1..20]  commande
ENTIERnum
```

Réaction aux primitives

```
IF pr_création
THEN  /* on suppose qu'après l'exécution de cette primitive,
      toutes les variables sont correctement initialisées */
```

Réaction aux événements

```
IF ev_cde_existante (saisie_cde)
THEN  IF pr_client_existant = True
      THEN  /* permettre la validation */
            pr_activation (bt_valider_cde)

IF ev_client_existant (saisie_client)
THEN  IF pr_cde_existante = True
      THEN  pr_validation (bt_valider_cde)

IF ev_bt_select (bt_valider_cde)
THEN  /* rechercher le numéro du client */
      num <-pr_obt_client
      /* rechercher la commande */
      commande <- pr_obt_cde
      /* valider la commande */

      IF pr_validation_cde (commande,num) = True
      THEN  pr_activation (bt_mémo_cde)
      ELSE  pr_lb_chg_contenu (lb_msge, "commande non valide")
```

```
IF ev_bt_select (bt_mémo_cde)
THEN  /* mémoriser la commande */
      IF pr_enregistrement_cde (commande,num) = True
      THEN  pr_lb_chg_contenu (lb_msge, "commande enregistrée")
            générer (ev_cde_enreg)
      ELSE  pr_lb_chg_contenu (lb_msge, "problème lors de la

IF ev_client_inexistant (saisie_client)
THEN  pr_désactivation (bt_valider_cde)

IF ev_cde_inexistant (saisie_cde)
THEN  pr_désactivation (bt_valider_cde)
```

Bibliographie

[Bodart 89]
Bodart F. et Pigneur Y., Conception des Systèmes d'information, Méthode, Modèles et outils, 2 ème édition, Masson (1989)

[Coutaz 85]
J.Coutaz, Abstractions for User Interface Design, IEEE Computer, vol. 18, 9 (1985)

[Coutaz 87]
J.Coutaz, PAC, an Implemention Model for Dialog Design, Proceedings of the Interact'87 conference, , North Holland, (1987)

[Coutaz 90]
J.Coutaz, Interfaces Homme-Ordinateur Conception et Réalisation,
Dunod Informatique (1990)

[Cunnigham 85]
W. Cunningham, The Construction of Smalltalk-80 Applications, Draft November (1985)

[Foley 84]
Foley J.D et Van Dam A., Fundamentals of Interactive Computer Graphics, Addison-Wesley, Reading Massachusetts (1984)

[Goldberg 84]
A. Goldberg, Smalltalk-80, The Interactive Programming Environment, Addison-Wesley Publ (1984)

[Green 85]
M.Green, The University of Alberta User Interface System, Computer Graphics,
vol 19, 3 (1985)

[Green 86]
M. Green, A Survey of Three Dialogue Models, ACM Transactions on Graphics,
vol 5, 3 (1986)

[Hartson 89]
R. Hartson, D. Hix, Human-Computer Interface Development : Concepts and Systems for its Management, ACM Computing Surveys, (1989)

[Hartson 89]
R. Hartson, User-Interface Management Control and Communication, IEEE Computer (1989)

[Heitz 87]
M. Heitz, Hood une Méthode de Conception Hiérarchisée Orientée Objet pour le Développement des Gros Logiciels Techniques et Temps-réel, Bigre n°57, Journées ADA France, le parallélisme en ADA (1987)

[Hill 87]
R. D Hill, Event-Response Systems Atechnique for Specifying Mutli-Thread Dialogues, Proceedings of the Conference onHuman Factors in Computing Systems and Graphics Interface CHI+GI' 87 (1987)

[Jacob 83]
Robert J.K Jacob, Using Formal Specifications in the Design of a Human-Computer Interface, Communications of the ACM, vol.23, 4 (1983)

[Jacob 86]
Robert J.K Jacob, A Specification Language for Direct-Manipulation User Interfaces,
ACM Transactions on Graphics, vol. 5, 4 (1986)

[Krasner 88]
G. Krasner et S. Pope, A Cookbook for Using the Model-View-Controller, User Interface Paradigm in Smalltalk-80, Journal of Object Oriented Programming (1988)

[Lamb 88]
D.A. Lamb, Software Engineering : Planning for Change, Prentice-Hall International Editions (1988)

[Masini 89]
G. Masini, A. Napoli, D. Colnet, D.Leonard, K. Tombre, Les Langages à Objets, chez InterEditions iia (1989)

[Meyer 89]
B. Meyer, Conception et Programmation par Objets, pour du logiciel de qualité, InterEditions, Paris (1989)

[MYERS 89]
B. A. MYERS, User Interface Tools : Introduction and Survey, IEEE Computer (1989)

[Norman 86]
D. A. Norman, S. W. Draper, User Centered System Design; Lawrence Erlbaum Associate, Publishers (1986)

[Olsen 90]
Dan R. Olsen, Propositional Production Systems for Dialog Description, CHI'90 Proceedings of the Conference on Human Factors in Computing Systems (1990)

[Osf/Motif 90]
Open Software Foundation, Eleven Cambridge Center, Cambridge, MA 021242 (1990)

[Petoud 90]
I. Petoud, Génération Automatique de l'Interface Homme-Machine d'une Application Interactive de Gestion Hautement Interactive, Thèse présentée à l'école des Hautes Etudes Commerciales de l'Université de Lausanne (1990)

[Provot et Sacré 90]
I. Provot et B. Sacré, Vers une Approche Orientée Objet de la Modélisation du Dialogue d'une Application Interactive, Bips FUNDP (1990)

[Provot et vanderdonckt 90]
I. Provot et J. Vanderdonckt , Les objets interactifs : Classification et Typologie, FUNDP (1990)

[Scapin 87]
D.L Scapin, Guide Ergonomique de Conception des Interfaces homme-machine, Rapport INRIA 87 (1987)

[Shneiderman 87]
B. Shneiderman, Designing the User Interface, Strategies for Effective Human-Computer Interaction, Addison Wesley Publishing Company (1987)

[Vanderdonckt 90a]
J. Vanderdonckt, Expérimentation des environnements transactionnels puor les interfaces homme-machine des applications interactives de gestion, Rapport de recherche FUNDP (1990)

[Vanderdonckt 90b]
J.Vanderdonckt, Vers une nouvelle architecture des applications interactives de gestion pour une famille d'outils graphiques, rapport IHM/Archi/1, FUNDP (1990)

[Van Lamsweerde 90]
A. Van Lamsweerde, Cours de Génie Logiciel, 2 ème licence FUNDP (1990)

[Wasserman 85]
A.Wasserman, Extending State Transition Diagrams for the Specification of Human-Computer Interaction, IEEE Transactions on Software Engineering, vol. 11, 8 (1985)

[Young 89]
D. A. Young, X Windows Systems Programming and Applications with Xtoolkit, Prentice Hall (1989)