THESIS / THÈSE

MASTER EN SCIENCES INFORMATIQUES

Modélisation d'une application interactive

Chandelon, Muriel; Warnant, Geneviève

Award date:
1987

Awarding institution:
Universite de Namur

FACULTÉS
UNIVERSITAIRES
N.D. DE LA PAIX
NAMUR

INSTITUT D'INFORMATIQUE

Année Académique
1986 - 1987

Modélisation d'une Application Interactive

Muriel CHANDELON
Geneviève WARNANT

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

Promoteur du mémoire : François BODART

Adresse : rue Grandgagnage, 21, B-5000 NAMUR (Belgium) - ☎ 081 - 22 90 65 - Télex 59.222 FAC NAM-B

Nous voulons tout d'abord remercier notre promoteur de mémoire, Monsieur François Bodart, pour l'attention qu'il a portée à nos travaux ainsi que pour ses précieux conseils lors de la rédaction de ce document.

Nous remercions également Joëlle Coutaz de nous avoir reçues si amicalement à Grenoble et de nous avoir fait partager son expérience dans le domaine des interfaces homme-machine. Nous sommes également reconnaissantes de l'accueil que nous ont réservé Christine Collet et Jim Crowley lors de notre stage.

L'un des auteurs, Geneviève Warnant, destine des remerciements particuliers à Marc Colpaert, pour l'aide qu'il a apportée dans les travaux de dactylographie.

Nos amis et amies nous ont procuré un énorme soutien moral durant ces longs mois de travail; nous les en remercions vivement.

Enfin, nous remercions de tout coeur nos parents pour nous avoir soutenues tout au long de ces cinq années d'études universitaires.

Table des matières

Intitulés | Pages

Intitulés | Pages

Intitulés | Pages

Intitulés | Pages

Intitulés Pages

Intitulés Pages

Table des figures

Numéro	Intitulés

Introduction

Introduction

De nombreuses études récentes s'attardent sur l'interface homme-machine et les développements de l'informatique conversationnelle. Le nombre d'applications interactives réalisées est sans cesse croissant et les postes de travail sur lesquels elles sont développées sont de plus en plus sophistiqués.

La conception de l'interface homme-machine d'une application requiert des compétences pluridisciplinaires où interviennent l'informatique, l'ergonomie et la psychologie cognitive. L'interface homme-machine doit être réalisée en fonction de l'application qu'elle concerne, mais également en fonction des possibilités offertes par le poste de travail sur lequel elle est développée et des desiderata du ou des utilisateurs auxquels elle est destinée. Dans l'état actuel des connaissances en interface homme-machine, le concepteur ne peut que proposer aux futurs utilisateurs un prototype d'interface qui sera critiqué et amélioré selon un processus itératif. La conception et la construction d'interfaces homme-machine sont donc des tâches difficiles et coûteuses. Coût et complexité augmentent encore avec l'arrivée de matériels sophistiqués et l'exigence d'un public mieux averti. Vu les nombreuses difficultés rencontrées lors de la conception du dialogue d'une application interactive, il serait souhaitable que l'interface soit spécifiée de manière précise avant d'être implantée sur un poste de travail.

L'objectif de notre mémoire est de proposer une modélisation d'une application interactive en spécifiant aussi bien les traitements que le dialogue.

Après avoir fixé notre cadre de réflexion en esquissant les différentes composantes d'une application interactive et expliqué la nécessité d'une séparation des traitements vis-à-vis du dialogue d'une telle application, nous présenterons une classification des outils existant en matière de construction et de gestion d'interface homme-machine.

Nous passerons ensuite au coeur même du mémoire. Celui-ci sera constitué de trois modèles décrivant les traitements et le dialogue de l'application ainsi que le lien qui existe entre les deux. Le modèle des traitements d'une application interactive s'inspirera des travaux de F. Bodart et Y. Pigneur [Bodart, Pigneur, 83]. Le modèle du dialogue décrira les objets sémantiques et les objets de présentation du dialogue, ainsi que la dynamique d'enchaînement de ces objets lors de l'exécution de l'application interactive. Enfin, le modèle de l'échange nous permettra de décrire la liaison et la collaboration existant entre les traitements et le dialogue.

La modélisation d'une application interactive que nous proposons peut donner lieu au développement d'outils de gestion d'une telle application. Ces outils permettront de contrôler l'exécution d'applications modélisées selon la méthodologie présentée dans ce mémoire. A plus long terme, la génération automatique d'une application interactive, sur base de ses spécifications, pourrait même être envisagée.

Nous terminerons ce travail par la description d'outils de construction d'interfaces homme-machine que nous avons utilisés, à savoir le logiciel DIALOGUE d'Apollo Computer Inc. et le système de gestion de dialogue MOUSE développé à Grenoble par Joëlle Coutaz, ainsi que par une l'exposé d'une proposition d'implémentation d'un système de gestion de dialogue qui respecterait la modélisation développée tout au long de ce mémoire.

Chapitre 1 :

Contexte du mémoire

Introduction

Ce chapitre a pour but de situer le contexte général dans lequel ce mémoire a été développé.

Nous commencerons donc par définir le type d'applications auquel s'adresse notre travail : les applications interactives. Une des caractéristiques d'une application interactive étant de posséder une part importante de dialogue avec l'utilisateur, nous nous intéresserons quelque peu aux qualités indispensables à une bonne interface homme-machine. L'importance d'avoir un dialogue adapté à l'utilisateur qui se trouve face au poste de travail sur lequel se déroule l'application sera soulignée. Nous citerons également les différents types de dialogue habituellement utilisés par les applications interactives.

Nous expliquerons ensuite les principes de base nécessaires à la conception d'une application interactive et, entre autres, la nécessité d'une séparation entre l'application fonctionnelle et son interface. Une description générale de l'architecture de toute application interactive sera donnée.

Ce chapitre se terminera par une présentation des différents types d'outils d'aide à la construction d'interfaces.

I.1 - Définition préalable

Une *application interactive* est une application informatique caractérisée par la part importante de dialogue qu'elle nécessite avec l'utilisateur pour assurer l'exécution des fonctionnalités qu'elle lui offre. Elle peut se définir à partir de ses deux composants principaux. Le premier de ces composants correspond aux traitements fonctionnels qu'elle permet d'exécuter, le second est l'interface homme-machine qui gère le dialogue et effectue les entrées/sorties propres à l'application interactive. Un système interactif est donc un système dans lequel la situation évolue grâce aux interventions réciproques de la machine et de l'utilisateur qui se trouve face au poste de travail.

Dorénavant, afin d'alléger l'écriture, nous appellerons *application* l'ensemble de l'application interactive, c'est-à-dire la partie *interface* et la partie fonctionnelle que nous désignerons également par *traitements* ou *application fonctionnelle*.

I.2 - Caractéristiques des interfaces homme-machine

I.2.1 - Qualités d'un bon dialogue

Nous présentons dans ce paragraphe une synthèse d'éléments explicitant les qualités d'un bon dialogue et qui proviennent d'un ensemble de références, à savoir [Scapin], [Petoud 87-3], [Coutaz 86-3], [Schneiderman].

Une interface homme-machine de qualité se doit de remplir certaines conditions théoriques telles que la simplicité, l'uniformité, l'adaptabilité à divers facteurs, la facilité de situation du contexte, l'efficacité, la flexibilité, la récupération des erreurs, et l'intégrité du dialogue, indépendamment du poste de travail sur lequel elle est implantée et des outils d'aide à la construction de dialogue existant sur le marché à l'heure actuelle. Ces caractéristiques seront expliquées tour à tour.

a - La première qualité d'un bon dialogue est d'être *simple* à utiliser. La période d'apprentissage doit être relativement brève et les manipulations effectuées par l'utilisateur doivent être faciles à réaliser.

b - Un dialogue *uniforme* offre une vue cohérente des entités qui le composent, des abréviations utilisées, du style d'interaction proposé à l'utilisateur et de la gestion des erreurs. Un tel dialogue facilite et améliore l'apprentissage et l'utilisation d'une application. Cette qualité d'uniformité du dialogue peut être étendue au niveau du poste de travail. Ainsi, toutes les applications développées sur un même poste de travail fourniraient à l'utilisateur des interfaces similaires.

c - Une interface de qualité se doit d'être *adaptable* aussi bien vis-à-vis du matériel que vis-à-vis des utilisateurs. Le développement des nouvelles technologies et l'évolution du matériel doivent pouvoir être intégrés dans le dialogue sans trop de difficultés. D'autre part, la diversification des utilisateurs d'applications informatiques exige l'adaptabilité du dialogue à ces différents types de personnes.

Avec l'évolution de l'informatique, on distingue aujourd'hui différentes catégories d'utilisateurs d'après les situations, les caractères des personnes ou les niveaux de connaissance. Nous donnerons ici une classification, non exhaustive, de ces utilisateurs.

Ainsi, d'un côté, on trouve le *débutant* et tous les utilisateurs *non spécialisés* qui attendent un environnement agréable, un dialogue simple permettant un apprentissage rapide de l'application. D'un autre côté, le *spécialiste* est essentiellement soucieux des performances et des fonctionnalités du dialogue. Toutefois, dans certains cas, il devient *utilisateur occasionnel* d'applications plus rarement utilisées et pour lesquelles il a alors besoin uniquement des informations essentielles à la réalisation de son travail.

D'un point de vue différent, nous pouvons distinguer les utilisateurs actifs de ceux qui ne le sont pas. Un utilisateur *actif* s'intéresse au fonctionnement de l'application et aux caractéristiques de son poste de travail alors qu'un utilisateur *passif* se contente de répondre au fur et à mesure au dialogue que lui présente l'application sans se soucier des traitements qui sont nécessaires à la production des résultats qu'il reçoit, sans chercher s'il n'y a pas de possibilités d'optimisation du dialogue. Il est clair qu'après quelques heures de manipulation, l'utilisateur actif profite beaucoup mieux que l'utilisateur passif des fonctionnalités offertes par l'application.

Dans le cadre de notre étude, nous distinguerons également l'utilisateur du poste de travail de l'utilisateur final. L'utilisateur *final* exploite les informations fournies par le système pour éventuellement prendre certaines décisions qui dépendent des résultats obtenus; il est donc le destinataire final de l'application, il comprend la sémantique de l'application et il agit sur les fonctions de l'organisation. Quant à l'utilisateur du *poste de travail*, il correspond plutôt à la personne qui introduit les informations d'entrée de l'application et vérifie l'émission des informations de sortie attendues; c'est à lui qu'est destinée l'interface homme-machine de l'application interactive. On notera cependant que ces deux types d'utilisateur se confondent de plus en plus souvent en une seule et même personne, sauf par exemple dans le cas d'applications nécessitant l'encodage d'un grand nombre d'informations.

Notons enfin que les classifications que nous donnons ici ne sont pas rigides. Un utilisateur débutant peut devenir expert après avoir acquis une certaine habileté dans la manipulation des ordinateurs. A l'inverse, nous avons déjà signalé qu'un expert réagit comme un non spécialiste face à des applications qu'il n'utilise que de manière occasionnelle. De même, l'utilisateur final d'une application interactive peut jouer le rôle d'utilisateur du poste de travail dans le cadre d'une autre application.

d - Une autre qualité d'un bon dialogue est son *efficacité* . L'efficacité d'un dialogue se juge à plusieurs points de vue : rapidité de développement, performance à l'utilisation et apprentissage rapide du dialogue sont également des qualités non négligeables d'une interface homme-machine.

e - L'interface doit être très *flexible* , particulièrement dans le processus d'acquisition ou de production de l'information, mais aussi dans le déroulement du dialogue qui ne peut pas être fixé de manière rigide. L'homme doit avoir à tout moment le sentiment de commander à la machine et non d'être son esclave.

Nous verrons plus tard , dans le cadre du chapitre 3 sur la modélisation du dialogue, les fonctions du dialogue qui permettent d'assurer cette souplesse de l'interface, tant du point de vue de la présentation des objets du dialogue à l'écran, que du point de vue de l'enchaînement du dialogue (possibilité d'effectuer des retours en arrière vers des données précédemment saisies , d'interrompre puis de reprendre un traitement,...).

f - La *récupération des erreurs* doit toujours être possible, même si un bon dialogue devrait avant tout chercher à éviter à l'utilisateur de commettre ces erreurs.

g - Un dialogue de qualité offre à l'utilisateur la possibilité de *se situer à tout moment* dans l'application et d'obtenir des renseignements concernant les actions qu'il peut effectuer.

Toute personne qui travaille mémorise temporairement une série d'informations liées au déroulement de sa tâche. Si celle-ci doit être interrompue, l'utilisateur sera confronté, au moment de la reprise, à des questions telles que : Où suis-je ?, Sur quelles données suis-je en train de travailler ?, Qu'ai-je déjà fait ?, Que me reste-t-il

à faire ?, Quelles sont les commandes dont je peux disposer ?, Comment ai-je fait pour arriver à cette situation ?, Comment sortir de cette situation ? Toutes ces interrogations peuvent se résumer en une question plus générale : *Quel est l'état du contexte dans lequel se déroule l'application ?* Et en poussant encore plus loin, l'utilisateur pourrait désirer connaître l'état général de la machine avec laquelle il travaille pour obtenir des renseignements tels que son état de charge, par exemple.

Les renseignements nécessaires à l'utilisateur pour répondre à toutes ces questions ont été classifiés par J. Nievergelt pour donner naissance au modèle "Site - Mode - Trail ". Ce modèle, que nous évoquons brièvement sur base de la présentation faite dans [Petoud 87-3], met en évidence la nécessité d'offrir à l'utilisateur la possibilité d'obtenir à tout instant des informations concernant les données courantes de l'application (Site), ses commandes courantes (Mode), ainsi que l'historique du dialogue (Trail).

A un instant donné, les données courantes d'une application sont les données sur lesquelles l'utilisateur est en train de travailler, et ses commandes courantes sont les commandes en cours d'exécution ou qui peuvent être exécutées à cet instant. Quant à l'historique du dialogue, il contient toutes les manipulations effectuées par l'utilisateur du poste de travail depuis un instant plus ou moins éloigné. Selon la dimension que l'on désire donner à cet historique, celui-ci peut "se souvenir " de toutes les interactions qui ont eu lieu depuis le début de la session de travail de l'utilisateur, ou depuis le début d'exécution de l'application, par exemple.

Des études ont montré que, confronté à une grande quantité d'informations, l'esprit humain est capable de sélectionner celles qui l'intéressent. Nievergelt développe donc l'idée qu'il faut mettre à la disposition de l'utilisateur tous les renseignements possibles et le laisser choisir.

h - Enfin, le dialogue doit également rester *intègre* par rapport aux spécifications fonctionnelles de l'application, même lorsque celles-ci ont été modifiées; dans ce cas, il faut veiller par exemple à ce que les libellés et les explications présentés par l'interface soient correctement mis à jour. La présentation à l'écran doit toujours être en parfaite concordance avec les fonctionnalités de l'application.

I.2.2 - La psychologie des utilisateurs

Nous avons déjà souligné le fait qu'il existe différents types d'utilisateurs des outils informatiques. Ce paragraphe montrera l'importance du caractère d'une personne dans l'acceptation d'une application interactive. Il s'inspire principalement d'observations développées dans [Schneiderman] mais reprend également certaines remarques provenant des sources déjà citées dans le paragraphe précédent.

Des facteurs psychologiques tels que la capacité de mémorisation des utilisateurs ou leur anxiété face à la machine doivent être pris en compte lorsqu'on parle d'optimisation de l'interface homme-machine.

La plupart des utilisateurs préfèrent un grand nombre d'opérations simples plutôt qu'un petit nombre d'opérations complexes; leur efficacité est également accrue grâce à de telles opérations. Le concepteur de l'application devra donc respecter cette préférence.

La mémoire humaine à court terme peut retenir de cinq à neuf données; il faut donc veiller à ne pas faire apparaître trop d'informations sur un même écran.

En effet, la présentation simultanée à l'écran d'un grand nombre de données fait intervenir la mémoire à long terme. Celle-ci fait des associations, hiérarchise les données et rend le travail de l'utilisateur nettement plus fatiguant.

L'anxiété de certaines personnes, due à leur peur de mal faire, peut diminuer fortement leur capacité de mémorisation à court terme. Si l'utilisateur n'est pas confiant en ses capacités d'utilisation d'un système, s'il craint de détruire des fichiers ou l'ordinateur lui-même, s'il est submergé par une foule de détails ou pressé par le temps, son anxiété diminuera ses performances. Le dialogue devra donc être conçu de manière à mettre l'utilisateur à l'aise.

Les conditions de travail plus ou moins stressantes de l'utilisateur du poste de travail sont un autre facteur à ne pas négliger. Le dialogue devra en effet être très différent si l'on s'adresse à quelqu'un qui travaille dans l'atmosphère calme d'une

annexe de bibliothèque ou si l'on s'adresse à un employé continuellement dérangé par le téléphone. Dans ce dernier cas, le dialogue doit être clair, très simple, et ne peut en aucun cas prêter à confusion.

Enfin, quand cela est possible, il faut faire participer le futur utilisateur au développement de l'application en lui faisant sentir l'importance de sa contribution dans l'élaboration de l'interface. Si l'utilisateur se sent concerné par la réalisation de l'automatisation d'une partie de son travail, il sera plus enclin à s'investir et à essayer de tirer le meilleur parti de ce nouvel outil.

Après avoir passé en revue différents facteurs qui influencent la qualité d'une interface homme-machine, nous allons décrire les types de dialogues qui existent actuellement.

I.2.3 - Les types de dialogue

Les dialogues peuvent être classés en plusieurs familles.

Dans [Petoud 87-2], on trouve la classification définie par Wasserman, qui recense sept types de dialogue :

- les questions/réponses,
- les formulaires,
- les menus,
- les langages de commandes,
- les langages d'interrogation de bases de données,
- les interfaces qui se rapprochent du langage naturel,
- les dialogues nés de nouvelles technologies et appelés WIMP (Window, Icon, Mouse, Popup).

Nous allons détailler quelque peu ces différents styles de dialogue, en intégrant les caractéristiques précisées à leur sujet dans [Petoud 87-2] et dans [Scapin] essentiellement.

a - Les *questions / réponses* :

Dans ce premier genre de dialogue, l'ordinateur pose une série de questions auxquelles l'utilisateur répond. Souvent, une question dépend de la réponse à la question qui la précède. L'utilisateur risque alors, au cours d'une séquence de questions/réponses, de devoir répondre à une question gênante sans pouvoir obtenir des renseignements sur les conséquences de la réponse.

Ce type de dialogue devient vite lourd et ennuyeux à mesure que l'utilisateur développe sa connaissance du système. Il est donc plutôt employé pour des utilisateurs débutants car il semble être celui qui conduit le moins à des erreurs.

Certaines contraintes liées à l'utilisation de ce style de dialogue sont cependant à préciser : les données à introduire doivent être connues de manière très précise lors de la conception du dialogue et leur ordre d'acquisition doit être déterminé à l'avance et imposé une fois pour toutes.

Les questions/réponses se révèlent néanmoins un mode de dialogue adéquat si les informations à obtenir ne peuvent être présentées autrement, sous forme d'une liste ou de codes.

b - Les *formulaires* :

Un autre mode de dialogue est celui où l'ordinateur présente à l'écran des formulaires similaires aux formulaires-papier; l'utilisateur en remplit les différents champs avec plus ou moins de liberté quant au passage d'un champ à un autre à l'intérieur du formulaire ou encore quant aux possibilités de correction ou de retour en arrière sur des données déjà introduites. Par exemple, un utilisateur expérimenté doit pouvoir effectuer une libre circulation au travers d'un ou même de plusieurs formulaires.

Ce système est bien sûr plus rapide que le type questions/réponses mais il est plus lourd à gérer car son exploitation se déroule en trois phases : appel du formulaire, remplissage et validation.

Toutefois, les formulaires sont particulièrement adaptés quand l'utilisateur doit entrer des informations déjà décrites sur papier, car il n'y a pas de confusion possible étant donné la correspondance directe entre le support papier et le support écran.

Un autre avantage est la facilité de détection d'erreurs syntaxiques en définissant des contraintes sur les champs d'entrée du formulaire.

c - Les *menus* :

Un menu présenté par l'ordinateur soumet à l'utilisateur une liste de choix parmi lesquels il peut effectuer une sélection. Ce style de dialogue convient bien pour le choix de commandes ou de paramètres prédéfinis, mais il ne permet pas l'introduction de données "arbitraires" c'est-à-dire de données qui ne peuvent pas être déterminées à l'avance vu leur nombre trop important ou leur variabilité. Le menu peut donc être un mode de dialogue assez restrictif quant à sa capacité d'expression ou encore être restreint par des contraintes telles que la taille de la fenêtre d'affichage car toutes les options doivent apparaître de préférence simultanément à l'écran.

Dans certains cas (options nombreuses que l'on peut regrouper en "classes" traitant du même sujet), on peut spécifier des hiérarchies de menus, mais celles-ci ne sont pas nécessairement faciles à gérer d'autant plus qu'il faut toujours laisser à l'utilisateur la possibilité de remonter dans la hiérarchie.

Les menus sont conseillés lorsque le nombre d'options est élevé car ils jouent alors le rôle d'aide-mémoire surtout pour un utilisateur occasionnel ou débutant. Quand l'utilisateur a acquis suffisamment d'expérience du système, l'utilisation de menus devient fastidieuse et peut être avantageusement remplacée par un langage de commandes, plus souple et plus rapide, comme nous le verrons dans le paragraphe suivant.

d - Les *langages de commandes* :

Les langages de commandes laissent à l'utilisateur l'initiative et le contrôle total du dialogue. En général, ils sont constitués d'un terme de commande, qui définit une fonction, et souvent de paramètres qui spécifient les différentes options de la fonction.

Ces langages ne sont acceptables (et généralement acceptés) que pour (par) des utilisateurs expérimentés qui ont intériorisé le modèle des fonctions du système et mémorisé la syntaxe du langage. Si ce n'est pas le cas, un tel style de dialogue conduit à des erreurs, à des confusions et à des frustrations. De toute façon, un tel langage devrait toujours offrir à l'utilisateur un système d'aide et de guidage qui lui permette de repérer les fonctions qui lui sont accessibles à un instant donné, de préciser les paramètres d'une commande,...

On constate que ce type de dialogue semble indiqué pour des systèmes disposant de nombreuses fonctions et pouvant accepter un large éventail d'entrées comportant notamment un certain nombre d'options.

Son avantage essentiel est sa flexibilité vis-à-vis de l'évolution du système. Un changement du système se traduit par des modifications simplifiées du logiciel car il suffit d'étendre les séquences reconnues par le système à l'introduction des commandes et d'ajouter les nouvelles fonctionnalités à celles déjà existantes.

Ses inconvénients sont liés à l'apprentissage et à la mémorisation d'un ensemble de commandes, d'autant plus que certaines d'entre elles sont peu fréquemment utilisées.

e - Les *langages d'interrogation de bases de données* :

Il s'agit ici de langages qui facilitent la consultation et la mise à jour interactives de bases de données.

Nous n'expliciterons pas ce style de dialogue car un tel langage ne peut pas être utilisé pour l'interface des applications interactives telles que nous les modélisons.

En effet, dans notre optique de travail, un utilisateur ne peut accéder à une base de données que via les traitements de l'application qui ont été implémentés. Aucun accès direct à la base de données ne lui est donc autorisé.

Dès lors, si les traitements de l'application voulaient permettre à l'utilisateur de spécifier lui-même une demande de consultation de la base de données, cette requête ne serait en fait pas explicitée par l'utilisateur conformément au langage d'interrogation de bases de données effectivement utilisé par l'application fonctionnelle pour effectuer ses accès à la base, mais bien sous une forme plus conviviale, précisée par le concepteur de dialogue.

f - Les interfaces qui se rapprochent du *langage naturel* :

Beaucoup de recherches se font actuellement dans le domaine de la communication homme-machine en langue naturelle. De nombreux problèmes restent non résolus et les propositions actuelles de développement d'interfaces en langage naturel sont limitées à des domaines spécifiques, dans lesquels un vocabulaire restreint est utilisé et pour lesquels la syntaxe grammaticale autorisée est simplifiée.

Dans le cadre de l'utilisation de ce style de dialogue, on peut toutefois évoquer le risque d'arriver à une surestimation du système, dans le sens où un utilisateur aura l'impression qu'avec une telle interface il peut obtenir tout ce qu'il désire.

g - Les dialogues nés des nouvelles *technologies WIMP* :

Le type de dialogue Window, Icon, Mouse, Popup est basé sur l'exploitation des technologies les plus récentes permettant la manipulation de fenêtres, d'icônes, d'une souris et de fenêtres superposées. Il permet de regrouper, d'intégrer et d'améliorer les techniques plus anciennes telles que la gestion par menus ou par formulaires, les langages de commandes ou d'interrogation de bases de données.

C'est sur ce type d'interface que nous nous baserons essentiellement pour expliciter notre modèle du dialogue.

Notons encore que ces différentes familles de dialogue sont généralement classifiées selon le degré de liberté qu'elles laissent à l'utilisateur. Les styles de dialogue par questions/réponses et par menus sont fondamentalement contrôlés par l'ordinateur. Les formulaires, les langages de commandes et d'interrogation de bases de données, les interfaces proches du langage naturel ou basées sur les technologies WIMP sont des modes de dialogue qui laissent un plus grand degré d'initiative à l'utilisateur. Les premiers types donnent donc lieu à des dialogues nettement plus rigides que les seconds.

Nous clôturerons ce paragraphe en signalant que nous n'avons pas voulu discuter ici des diverses formes d'implémentation possibles de ces différents styles de dialogue. Par exemple, la sélection d'une option d'un menu peut se faire par un nom, par un numéro, en la désignant avec la souris, en déplaçant le curseur,...; un langage de commandes peut être basé sur l'utilisation de touches-fonction, de codes,... Ceci impliquerait toutefois le développement de règles ergonomiques plus précises, ce qui n'entre pas dans le cadre de cette introduction. Le lecteur intéressé en trouvera un exposé clair et assez exhaustif dans [Scapin].

Après avoir détaillé les caractéristiques des interfaces homme-machine, nous pouvons présenter le principe fondamental de conception d'une application interactive. Celui-ci permettra de développer une interface qui réponde aux qualités mises en évidence dans ce paragraphe.

I.3 - Principe fondamental de conception d'une application interactive

L'interface est un facteur essentiel dans la réussite d'une application interactive. En effet, le dialogue a pour but, d'une part, de fournir aux traitements l'ensemble des informations dont ils ont besoin afin d'assurer le bon fonctionnement de l'application et, d'autre part, de permettre à l'utilisateur de visualiser les résultats des traitements effectués par l'application.

Or, comme nous l'avons déjà signalé, le développement d'une interface homme-machine est coûteux et exige des compétences multiples spécifiques à sa conception.

Ceci suggère la *séparation* nette, au sein de l'application, *des traitements* ou services fonctionnels offerts *et de l'interface* utilisateur.

Actuellement, ce principe semble d'ailleurs universellement admis; il est utilisé par bon nombre de personnes qui effectuent des travaux dans le domaine des applications interactives. On peut notamment en trouver une illustration dans [Coutaz 86-3], [Konsynski], [Rosenthal, Yen], [Ahlsen, Britts] ou dans le logiciel DIALOGUE de l'Apollo (Apollo Computer Inc.).

Si ce principe est respecté au moment de la spécification de l'application, on aboutit à une structure logicielle qui présente de nombreux avantages. En effet, une telle architecture répond également au principe de modularité généralement admis dans le cadre de la méthodologie de développement de logiciels.

a - En premier lieu, le principe de séparation de l'application en sa partie traitement et sa partie interface autorise *l'indépendance de la conception de l'interface par rapport à la conception des fonctionnalités de l'application*.

En effet, ce principe permet de considérer la conception de ces deux composants logiciels de façon indépendante. On peut donc envisager que fonctionnalités et dialogue soient définis et implémentés par des personnes distinctes, ce qui se révèle intéressant puisque les compétences et qualités requises pour chaque partie sont différentes.

Cette distinction nécessite toutefois que le niveau d'abstraction connu par l'application fonctionnelle soit celui des entités (objets manipulés) qui lui sont propres. De même, le dialogue ne doit manipuler que des objets qui lui sont significatifs. Ces objets ne sont dès lors pas nécessairement du même niveau d'abstraction; la "mise à niveau" sera réalisée par la structure d'échange comme nous l'expliciterons dans le chapitre IV.

Pour leur travail de conception, il faut donc offrir au concepteur de l'application fonctionnelle et au concepteur du dialogue des langages de spécification appropriés, de haut niveau d'abstraction. En ce qui concerne l'interface homme-machine, ce langage s'appuyera de préférence sur des techniques graphiques (langage visuel interactif) car la spécification doit aboutir à un produit fortement visuel.

Pour donner cette dimension graphique à un langage de spécification, il faut cependant que les modèles ou concepts sous-jacents au langage se prêtent facilement à une représentation graphique. Le modèle Entité-Association ainsi que les modèles des traitements, présentés dans [Bodart, Pigneur], sont un exemple d'une modélisation qui se prête bien à une représentation graphique; celle-ci a d'ailleurs été introduite dès le départ. On peut donc aisément envisager le développement d'un langage de spécification visuel interactif basé sur cette représentation.

b - De par l'indépendance de conception des traitements et de l'interface, on arrive également à garantir *l'indépendance d'implémentation de l'application face au matériel*.

Celle-ci permet de débarrasser l'application fonctionnelle de préoccupations liées à la machine sur laquelle elle est ou sera implantée.

Au niveau de l'interface, la conception est basée sur la spécification d'un terminal ou poste de travail virtuel.

Ce dernier a pour but de définir un ensemble de fonctions standard permettant de programmer des interfaces en cachant au concepteur de dialogue les particularités d'un terminal physique. Parmi ces fonctions, on trouve notamment des fonctions de traitement de messages, de manipulation de fenêtres et de boîtes de dialogue, des fonctions de gestion de menu, de lecture des entrées au clavier et à la souris, une fonction presse-papier pour la transmission d'informations entre applications, ... L'intérêt de cette définition est que toute interface peut dès lors être spécifiée sur base de conventions déterminées par le poste de travail virtuel dont la définition est commune à différents types de machines.

Cette technique permet d'adapter les interfaces à l'évolution technologique particulièrement rapide dans ce domaine. On aboutit ainsi à la construction et à la maintenance d'interfaces puissantes, qui suivent l'évolution des performances et des innovations.

c - Des deux avantages développés précédemment découle aussi *la possibilité et la facilité de ré-emploi du code*, tant en cas de modification ou d'extension de l'application fonctionnelle et/ou de son interface, qu'en cas de développement d'une application similaire pour laquelle on peut récupérer une interface déjà conçue (offrant du même coup à l'utilisateur la continuité au travers des applications développées).

d - La conception indépendante des traitements et du dialogue, en conformité avec le principe de modularité, permet enfin de faciliter la *construction itérative de prototypes de l'interface*.

En effet, il est d'autant plus aisé de développer et d'adapter progressivement des prototypes d'interface que le dialogue constitue un composant séparé par rapport aux fonctionnalités de l'application, composant lui-même découpé en modules que l'on peut remplacer facilement.

Or, dans le domaine de la construction d'interfaces, il n'y a pas de solution universelle; le prototypage reste donc une garantie d'approximation de l'interface la plus adéquate et la mieux adaptée aux desiderata des utilisateurs concernés.

I.4 - Architecture fonctionnelle d'une application interactive

Conformément au principe fondamental de séparation développé au paragraphe I.3, on retrouve les deux composants de l'application comme éléments de base de l'architecture fonctionnelle, comme le montre la figure 1.

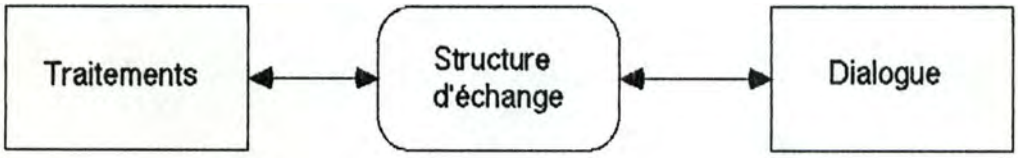


Figure 1 - Architecture fonctionnelle d'une application interactive

La partie "Traitements" implémente les fonctionnalités de l'application sans jamais se soucier du dialogue nécessaire à la saisie d'informations au terminal ou à l'affichage (ou impression) des résultats. Elle ne s'intéresse qu'à la sémantique de l'application tant du point de vue des services offerts que des objets manipulés.

La partie "Dialogue" définit l'interaction homme-machine : elle doit permettre la gestion des entrées/sorties (affichage, saisie de données avec validation syntaxique) ainsi que le contrôle du déroulement du dialogue lui-même et de l'exécution de l'application via le dialogue (choix d'un traitement offert par l'application,...).

La "Structure d'échange", quant à elle, a pour but de garantir la collaboration entre les traitements et le dialogue de l'application en vue d'assurer le bon fonctionnement de l'ensemble, tout en permettant l'indépendance respective des deux parties.

Sa fonction première consiste à assurer la transmission, entre les traitements et le dialogue, des informations qui leur sont nécessaires pour se dérouler correctement.

Elle doit ainsi :

- communiquer aux traitements les informations, saisies au terminal, qui concernent la sémantique de l'application, en les adaptant à la vue qu'en connaissent les traitements,
- fournir à l'utilisateur les résultats des traitements qui lui sont destinés, conformément à la vision qu'il en attend,
- transmettre, de façon bi-directionnelle, des commandes liées au contrôle de l'exécution de l'application (dialogue et traitements), telles qu'une demande de retour en arrière adressée par les traitements après détection d'une erreur, ou directement exigée par l'utilisateur via l'interface, une autorisation de mise à jour de la base de données de l'application après confirmation des résultats d'un traitement par l'utilisateur, ...

Au niveau de la gestion des entrées/sorties, le rôle essentiel de l'échange est d'effectuer la mise en correspondance, d'un côté, des entités manipulées par l'application et, de l'autre côté, des objets du dialogue utilisés par l'interface.

I.5 - Les outils d'aide à la construction d'interfaces homme - machine

Ce paragraphe est basé principalement sur la classification des outils pour la construction d'interfaces homme-machine proposée par Joëlle Coutaz dans [Coutaz 86-3].

Actuellement, les outils d'aide à la construction d'interfaces homme-machine sont de deux types : les boîtes à outils (ou toolbox) et les systèmes génériques. Nous allons décrire en quelques mots ce que sont ces outils.

Nous aborderons ensuite la notion d'outil de génération de dialogue.

I.5.1 - Boîtes à outil

Un toolbox est une bibliothèque de procédures facilitant l'écriture d'interfaces homme-machine. "L'éventail des fonctions offertes va de la gestion des événements physiques de bas niveau tels que les clics-souris et l'affichage de points, à la gestion d'entités de dialogue de haut niveau telles que menus et formulaires. " [Coutaz 86-3] . Ces fonctions peuvent être divisées en deux catégories : la gestion du poste de travail et la gestion du dialogue.

I.5.1.1 - Fonctions de gestion du poste de travail

" Les abstractions introduites pour la gestion du poste de travail définissent un appareil logique dont la fonction est de cacher le fonctionnement de l'appareil physique. Bien entendu, la nature de ces abstractions varie d'un toolbox à l'autre ! ... Les postes de travail visés ne se cantonnent pas au graphique. Ils sont multimédias, c'est-à-dire capables de gérer simultanément, en local comme à distance, texte, graphique et son. ...

Les abstractions usuelles, bien que variant d'un toolbox à l'autre, correspondent à des

- notions graphiques de base,

- notions textuelles (police de caractères),
- notions liées au partage du terminal (fenêtre, événement) " [Coutaz 86-3]

Les fonctions de gestion du poste de travail ont donc pour but de créer un poste de travail virtuel. Comme nous venons de le dire, les abstractions définies par ce poste de travail virtuel diffèrent d'un toolbox à l'autre, et donc d'une machine à l'autre. Or, l'une des qualités nécessaires à une bonne interface est l'indépendance de celle-ci par rapport au poste de travail sur lequel elle est implantée. Il serait donc très intéressant de pouvoir définir des abstractions communes à toute une série de stations de travail; décrire un tel poste de travail permettrait d'obtenir des interfaces portables. C'est, par exemple, ce qui pousse le groupe de recherche *Altair* à définir un poste de travail virtuel commun à trois machines différentes : le SUN, l'IBM / PC et le Macintosh. [Lepenant 87]

La description détaillée des abstractions existant sur de tels postes de travail n'est pas le but de ce document. Notons quand même que la qualité d'un toolbox est fonction de la puissance et de la richesse des primitives graphiques qu'il offre au programmeur, étant donné les types d'objet qu'il manipule.

I.5.1.2 - Fonctions de gestion du dialogue

Les fonctions de gestion du dialogue s'appuient sur les outils de niveau inférieur que sont les fonctions de gestion du poste de travail décrites précédemment. La gestion du dialogue fournit au programmeur des primitives concernant deux types d'objet :

- les objets simples comme les chaînes de caractères, les icônes, les règles graduées, les boutons, les menus et
- les objets composés comme les formulaires.

Actuellement, toute boîte à outils évoluée propose au programmeur des primitives de création et de manipulation de ces différents types d'objets.

En résumé, un toolbox est un ensemble de primitives organisables en 2 niveaux fonctionnels (gestion du poste de travail et gestion du dialogue). Du point de vue de l'interface de programmation, les abstractions du toolbox ne sont pas organisées en couches : elles sont toutes accessibles au programmeur.

I.5.1.3 - Avantages et insuffisances d'un toolbox

Les principaux avantages du toolbox sont son extensibilité et sa flexibilité. Le toolbox est un ensemble de fonctions; cet ensemble peut être facilement étendu. Grâce à ses niveaux d'abstraction variés, le toolbox laisse au programmeur le choix entre le plein contrôle de l'appareil et l'utilisation de services évolués. Les services évolués du toolbox définissent des abstractions standard; celles-ci permettent au programmeur de réaliser plus aisément des interfaces, mais elles imposent un style particulier d'interface homme-machine. " L'existence d'un style renforce la cohérence de présentation entre les applications du poste de travail. Toutefois ce style ne doit pas s'imposer. Il est important que l'implémentation des abstractions permette au programmeur de redéfinir le comportement standard. " [Coutaz 86-3]

Les inconvénients dus à cette flexibilité des boîtes à outils ne sont cependant pas négligeables.

L'ensemble de fonctions du toolbox n'impose aucune modélisation de l'application. L'utilisation du toolbox ne force donc absolument pas la séparation entre les traitements et l'interface de l'application. Or, nous avons vu que ce principe de séparation est une condition nécessaire à la réalisation d'une interface adaptable.

Le nombre de fonctions constituant un toolbox est énorme. Le programmeur ne pourra les utiliser au mieux qu'au prix de plusieurs mois d'apprentissage.

Lorsqu'un programmeur a réalisé quelques applications à l'aide d'un même toolbox, il s'aperçoit rapidement que chaque application est conçue selon le même squelette de contrôle des événements et que certains traitements se retrouvent d'une application à l'autre. De tels traitements devraient faire l'objet d'une extension du

toolbox. Puisqu'il n'en est rien, le programmeur doit se débrouiller seul mais ces constatations lui suggèrent de réutiliser, voire même de partager, ce code commun à toutes les applications. De telles idées sont à la base des systèmes génériques d'aide à la conception d'interfaces homme-machine.

I.5.2 - Les systèmes génériques

" Les systèmes génériques fournissent un squelette standard sur lequel sont greffés les composants spécifiques aux applications. Squelette et composants s'appuient sur les services d'un toolbox. On distingue deux types de systèmes génériques : les Applications Extensibles et les Systèmes de Gestion de Dialogue." [Coutaz 86-3]

I.5.2.1 - Application extensible

Les applications extensibles fournissent au programmeur un squelette d'application contenant les fonctions usuelles de l'interface homme-machine. Le programmeur ne devra donc plus s'occuper que des fonctions non standard de l'interface et des fonctionnalités de son application. Les applications extensibles permettent donc de réutiliser du code commun à toutes les applications et diminuent ainsi le travail du programmeur. Cependant, l'interface est toujours décrite au moyen d'une suite d'appels de fonctions qui font partie soit de l'application extensible, soit du toolbox selon que ces fonctions concernent ou non des composantes standard. Le programmeur doit donc toujours connaître le toolbox pour développer les parties non standard de son interface. Les Systèmes de Gestion de Dialogue fournissent quant à eux un langage de spécification de plus haut niveau de l'interface.

I.5.2.2 - Système de Gestion de Dialogue (SGD)

Le rôle d'un Système de Gestion de Dialogue (SGD) est double. Sa première fonction est de fournir à l'implémenteur d'application un langage de spécification de l'interface homme-machine. Cette spécification définit le dialogue ainsi que le lien qui existe entre ce dernier et les traitements de l'application. La seconde fonction du SGD est de présenter au programmeur un outil qui, lors de l'exécution de l'application interactive, servira à la fois de noyau de contrôle du déroulement de l'exécution et de médiateur entre l'utilisateur et les traitements, satisfaisant les demandes de réalisation de traitements exprimées par l'utilisateur et les demandes de l'application fonctionnelle qui désire recevoir certaines données introduites par l'utilisateur du poste de travail.

Contrairement aux applications extensibles, les SGD forcent donc le concepteur à effectuer une séparation nette entre les traitements et le dialogue de l'application. Les traitements peuvent être écrits sans se soucier de la saisie ou de l'affichage des données à l'écran. C'est, par exemple, le SGD qui leur transmet toutes les données dont ils ont besoin.

Une analogie peut être faite entre les SGD et les SGBD (Systèmes de Gestion de Bases de Données). Le SGBD libère le programmeur d'application des détails concernant le stockage et la recherche des données en mémoire; le SGD, quant à lui, libère le programmeur d'application des détails concernant l'acquisition et l'affichage des données à l'écran.

Actuellement, les langages de spécification des SGD sont généralement des langages textuels. Des recherches sont effectuées dans le but de fournir aux programmeurs des outils graphiques de spécification de dialogue (cfr [Coutaz 86-1]). Les implémenteurs de dialogue pourraient alors directement visualiser à l'écran le résultat de leur spécification et obtenir ainsi plus rapidement des interfaces d'une qualité supérieure.

I.5.3 - Les systèmes de génération de dialogue

En vue d'offrir des outils toujours plus puissants d'aide à la construction d'interfaces homme-machine, les chercheurs se penchent maintenant vers l'étude de systèmes de génération automatique de dialogue. De tels systèmes devraient théoriquement être capables de générer l'interface d'une application à partir des spécifications de l'application fonctionnelle c'est-à-dire à partir de la vue (abstraite) que les traitements de l'application possèdent des données à saisir ou à afficher à l'écran. Un système de génération de dialogue devrait donc spécifier automatiquement la présentation de toutes les informations que l'utilisateur voit apparaître à son poste de travail ainsi que l'ordre d'enchaînement de ces données à l'écran (la conversation).

Cependant, la conversation et la présentation d'une application interactive dépendent énormément du ou des utilisateurs auxquels cette dernière est destinée, de leurs goûts personnels en matière d'interface homme-ordinateur, de leur niveau de connaissance de la machine et de l'application elle-même... Le générateur de dialogue ne peut pas connaître toutes ces caractéristiques. Dans le cas le plus favorable, il ne pourra interpréter qu'une classification plus ou moins grossière des destinataires de l'application que le concepteur lui aura fourie. Il ne sera donc pas capable de produire une interface définitive qui plaira à coup sûr aux utilisateurs. Tout au plus pourra-t-il produire un prototype d'interface qui satisfasse relativement bien ces derniers.

I.5.4 - Synthèse

Un toolbox fournit au programmeur un ensemble de procédures standard facilitant la gestion du poste de travail et la gestion du dialogue. Les systèmes génériques fournissent une architecture standard d'application. Parmi ces systèmes génériques, "les applications extensibles proposent comme interface de programmation un langage de programmation usuel tandis que les SGD vont généralement plus loin dans l'abstraction avec un langage de spécification de haut niveau ou mieux encore un langage graphique dans le cas de la spécification directe interactive. " [Coutaz 86-3]

Quant aux systèmes de génération plus ou moins automatique de dialogue, ce sont des outils qui représentent une extension des SGD et qui permettent de faciliter le prototypage.

I.6 - En résumé

Dans ce chapitre, nous avons tout d'abord déterminé clairement le type d'applications auquel est destiné notre mémoire. Nous avons présenté un ensemble de directives pour obtenir une interface de qualité, et décrit les différentes familles de dialogue utilisées à l'heure actuelle. Nous avons souligné l'importance de la personnalité de l'utilisateur dans l'acceptation de l'interface d'une application interactive.

Ensuite, nous avons exposé le principe fondamental de conception que représente la séparation des traitements et du dialogue d'une application. Une architecture fonctionnelle d'application s'inspirant de ce principe a été proposée.

Enfin, nous venons de décrire les différents types d'outils d'aide à la construction d'interfaces homme-machine.

Nous disposons maintenant d'un certain nombre de directives ergonomiques que nous devrons respecter lors de la modélisation de toute application et nous connaissons les outils qui peuvent aider le concepteur dans sa tâche de construction des dialogues. Nous pouvons donc passer à l'étape suivante, qui est la modélisation de l'application interactive. C'est le but des trois prochains chapitres qui modéliseront successivement les fonctionnalités de l'application, son dialogue, et les échanges existant entre ces deux parties.

Chapitre 2 :

La modélisation de l'application fonctionnelle

Introduction

Avant de parler de la spécification conceptuelle d'une application fonctionnelle, il nous faut resituer cette notion d'application dans un cadre plus général : celui du système d'information dans lequel elle s'insère.

" Par système d'information (SI) d'une organisation, nous entendons une construction formée d'ensembles :

- d'informations, représentations - partielles - de faits qui intéressent l'organisation;
- de traitements, procédés d'acquisition, de mémorisation, de recherche, de communication et de transformation des informations et
- de ressources - humaines, techniques et organisationnelles - qui en assurent le fonctionnement.

Un système d'information n'est donc pas réduit au seul système informatisé ou informatique mais englobe des traitements automatisés, interactifs et même exclusivement manuels." [Bodart, Pigneur]

Dans le cadre de notre étude, nous nous intéressons bien sûr à l'ensemble de ces traitements. Dans ce chapitre, nous évoquerons, en particulier, la modélisation des fonctionnalités de l'application interactive, c'est-à-dire de l'application fonctionnelle uniquement. Dans ce domaine, diverses méthodologies ont été proposées à ce jour. En effet, la complexité de la conception des SI est un problème qui a été mis en évidence depuis de nombreuses années. Il a donc déjà fait l'objet de bon nombre d'études et de recherches, ayant abouti à la définition de divers modèles conceptuels et notamment illustré dans [Rolland, Richard], [Chen].

Vu cet état de fait, nous ne voulons pas imposer au concepteur de l'application d'utiliser une méthodologie plutôt qu'une autre car il a peut-être certaines préférences ou habitudes en la matière.

Cependant, nous préciserons certains concepts qu'il nous a fallu introduire afin de modéliser l'échange entre les traitements et le dialogue, en vue de garantir le bon fonctionnement de l'ensemble de l'application.

Quelle que soit la méthodologie adoptée par le concepteur de l'application fonctionnelle, celle-ci devra donc posséder ces concepts sous une forme ou sous une autre.

Si ce n'était pas le cas, ceci remettrait en cause l'utilisation du modèle de l'échange, présenté au chapitre 4, qui se base sur les concepts déterminés, d'une part, par le modèle de l'application fonctionnelle et, d'autre part, par le modèle de dialogue pour définir la communication entre les traitements et l'interface de l'application. Il deviendrait ainsi peu probable que l'ensemble de la méthodologie que nous proposons reste valide.

Pour mieux illustrer les notions introduites, nous allons les présenter en les intégrant à une démarche méthodologique particulière. Il s'agit de la démarche présentée en détails dans [Bodart, Pigneur] et que nous allons reprendre dans ses grandes lignes tout au long de ce chapitre.

Cette démarche fournit un environnement, sous forme de modèles, d'outils automatisés et de règles, favorable à la construction d'un schéma conceptuel des fonctionnalités d'une application, schéma qui présente les qualités de communicabilité, complétude, cohérence, faisabilité et conformité aux besoins de l'organisation.

Chaque modèle se rapporte à un aspect particulier du SI à élaborer :

- le modèle de *structuration des informations* sert à définir la sémantique des données appartenant à la base de données du SI. La structuration des informations définit les données et les relations entre celles-ci, précise leurs conditions d'existence et les valeurs qu'elles peuvent prendre;

- le modèle de *structuration des traitements* permet la décomposition, par raffinements successifs, d'un traitement global en traitements de plus en plus élémentaires,

- le modèle de la *statique des traitements* permet, pour un traitement donné, d'une part, de préciser, les messages-données et la partie de la mémoire du SI nécessaires à l'obtention des messages-résultats et, d'autre part, de spécifier, sous une forme adéquate, la procédure de traitement qui assure la transformation, à l'aide de la base de données du SI, des messages-données en messages-résultats;

- le modèle de la *dynamique des traitements* complète le modèle de la structuration des traitements en permettant de décrire leurs enchaînements;

- le modèle des *ressources* sert à caractériser le comportement de tous les éléments qui contribuent à l'exécution des procédures de traitement (ressources humaines, matérielles, financières et organisationnelles).

Pour illustrer toutes les notions développées dans ces différents modèles, nous allons fournir un exemple que nous traiterons de manière complète tout au long des trois prochains chapitres. Avant d'exposer le contenu des différents modèles de l'application fonctionnelle, nous présentons l'énoncé de cet exemple.

Enoncé de l'exemple

Le cadre dans lequel se déroule cet exemple simplifié est un organisme de mutuelle.

Une personne assurée à la mutuelle est identifiée par un numéro matricule. Elle est caractérisée par un nom, un prénom, une adresse et une référence bancaire, ainsi que par différents codes : code-pension, ,...

Une prestation médicale est toujours associée à un et un seul individu. Elle est caractérisée par la date de soins. Elle est identifiée par l'individu qu'elle concerne et la date de soins. Une prestation est composée de 1 à N lignes. Chaque ligne est relative à un acte médical et caractérisée par un numéro d'ordre, une quantité d'acte et différents montants : montant-ao (assurance obligatoire), montant-tm (ticket modérateur), montant-br (base de remboursement) et montant-dr (montant droit).

Un acte est identifié par un code contenant une lettre et une série de chiffres, et caractérisé par une base de remboursement.

II.1 - Modèle de structuration des informations

II.1.1 - Esquisse d'un modèle

L'efficacité des comportements d'une organisation repose notamment, via son SI, sur la qualité des informations utilisées dans l'organisation. Il s'avère dès lors indispensable de gérer cette ressource au même titre que toute autre ressource de l'entreprise (personnel, trésorerie,...). Pour ce faire, il convient de définir rigoureusement les informations significatives pour l'organisation (concepts, objets, faits, événements,...). Ces informations déterminent la signification attachée aux données mémorisées dans une base de données qui constitue, à ce titre, la mémoire du système d'information.

Le modèle conceptuel auquel nous nous référerons en la matière relève de l'approche Entité-Association. Celle-ci propose d'exprimer les informations du réel perçu par l'organisation à partir des trois concepts élémentaires que sont l'entité, l'association et l'attribut, et de la notion de contrainte d'intégrité. Introduisons succintement ces notions.

Les entités correspondent aux objets sémantiques connus par l'application. Les relations qui existent entre ces objets sont représentées par des associations. Entités et associations possèdent des caractéristiques (nom,...) qui sont exprimées via le concept d'attribut. Enfin, les contraintes d'intégrité explicitent les propriétés que doivent satisfaire les données appartenant à la mémoire du système d'information.

Dans le cadre de notre réflexion, nous n'avons apporté aucune modification ou extension aux concepts du modèle Entité-Association. Pour une description plus détaillée, nous renvoyons donc le lecteur à [Bodart, Pigneur] ou [Chen].

II.1.2 - Exemple de shéma Entité-Association

Par souci documentaire, nous fournissons ici le schéma Entité-Association correspondant à l'énoncé de l'exemple "mutuelle ".

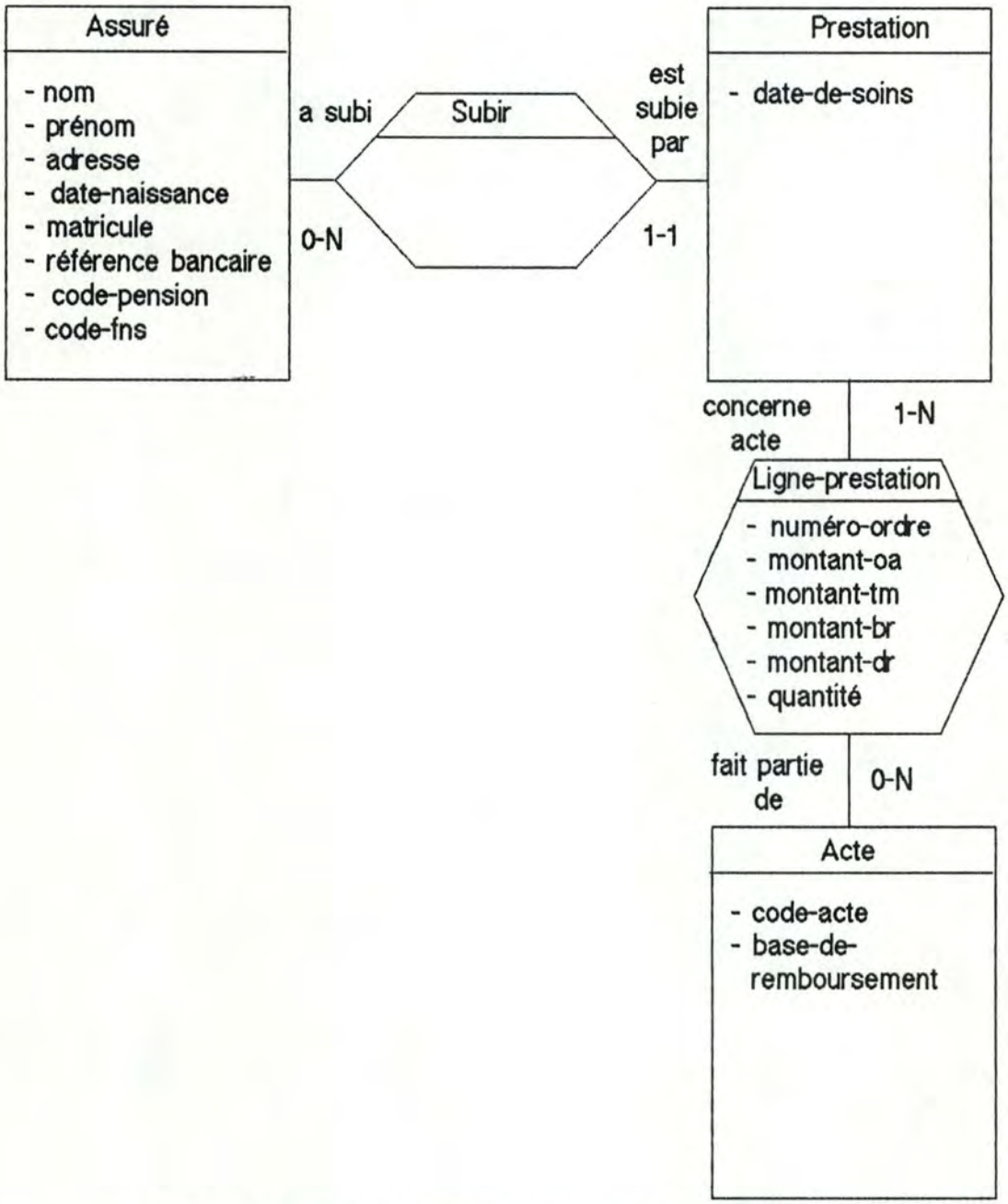


Figure 2 - Schéma Entité-Association du cas mutuelle.

Quand les données du système d'information d'une organisation ont été définies, le concepteur de l'application fonctionnelle doit en spécifier les traitements. Nous allons maintenant étudier les différents modèles qui permettent d'en donner une description précise.

II.2 - Modèle de structuration des traitements

II.2.1 - Présentation des concepts

Comme nous l'avons déjà évoqué dans l'introduction de ce chapitre, nous supposons que le concepteur de l'application fonctionnelle effectue une démarche méthodologique qui fournit la décomposition fonctionnelle et la spécification des traitements de l'application. Cette démarche procède d'une méthodologie laissée au choix du concepteur mais nous proposons ci-après la démarche présentée dans [Bodart, Pigneur] en l'adaptant compte tenu des concepts nécessaires pour la modélisation d'une application essentiellement interactive.

L'objectif de ce modèle est de fournir aux concepteurs et analystes des critères leur permettant de décomposer une application en traitements de plus en plus élémentaires.

Tout traitement est décomposé sous forme arborescente de telle sorte qu'un traitement de niveau intermédiaire i (i > 1) provient de la décomposition d'un seul traitement de niveau i -1 et se décompose en n traitements de niveau i+1; en outre, cette décomposition correspond à une démarche par raffinements successifs (conception descendante) et peut déterminer un nombre quelconque de niveaux.

Compte tenu de leur signification particulière dans le modèle présenté, on distingue cependant des niveaux ou repères privilégiés qui répondent aux définitions suivantes :

- un *projet* est "la partie d'un système d'information qui fait l'objet d'une analyse" [Bodart, Pigneur];
- une *application* (éventuellement interactive) est "un traitement quasi autonome par rapport aux autres applications d'un projet. Elle représente la dimension minimale d'un projet informatique " [Bodart, Pigneur];
- une *phase* est un traitement (manuel ou automatisable) correspondant à une unité logique de traitement. Elle possède une unité spatiale d'exécution; celle-ci implique l'absence de changement de lieu dans l'organisation et de changement de ressources lors de l'exécution de la phase. Nous distinguons les phases automatiques, les phases interactives et les phases manuelles.

Les phases *automatiques* imposent que le déroulement de leur exécution puisse avoir lieu sans interruption . Ceci détermine certaines conditions telles que l'absence de point d'attente au sein du traitement (point de décision humaine, point d'accumulation) et la disponibilité de toutes les informations au moment du déclenchement.

Les phases *interactives* nécessitent un dialogue avec l'utilisateur pour l'acquisition de messages-entrée, par exemple. Elles peuvent donc se dérouler avec des interruptions dues, soit, à l'attente de messages en provenance de l'utilisateur, soit à des demandes explicites de suspension d'un traitement de la part de l'utilisateur.

Enfin, des phases *manuelles* peuvent également être spécifiées à l'aide de cette modélisation des traitements. Dans le cadre des applications interactives que nous modélisons, nous ne les prenons cependant pas en considération car elles n'ont pas recours à l'ordinateur;

- une *fonction* "correspond au niveau élémentaire de la nomenclature des traitements..." [Bodart, Pigneur].

Plusieurs aspects de spécification peuvent être évoqués pour distinguer le niveau des fonctions des autres niveaux de décomposition des traitements de l'application. Nous retiendrons les critères évoqués dans [Bodart 87], à savoir:

. les fonctions sont associées à un objectif et à un comportement organisationnels considérés comme élémentaires par l'organisation,

. l'exécution d'une fonction doit laisser la base de données dans un état cohérent et doit vérifier les contraintes d'intégrité associées aux structures de données utilisées,

. une fonction reçoit un ou plusieurs types de message et génère un ou plusieurs types de message,

. dans une application interactive, une fonction est une opération "unitaire", c'est-à-dire que son exécution ne peut pas être interrompue par l'utilisateur.

De ces caractéristiques, on peut déduire qu'une fonction est identifiée par deux facteurs : la définition des objectifs de l'organisation qu'elle réalise et la définition des contraintes d'intégrité sur le schéma entité-association.

On conviendra donc qu'une fonction possède une sémantique simple et puisse être assimilée à une ou plusieurs primitives d'action sur la mémoire du système d'information (consulter, modifier, ajouter, supprimer) et/ou de production de messages-résultats (cfr paragraphe II.3.1).

Néanmoins, dans le cadre de la modélisation d'une application interactive, le seul critère de décomposition qui nous intéresse est le caractère non-interruptible d'une fonction et ce afin de respecter le principe de séparation de l'application fonctionnelle et de l'interface exposé au paragraphe I.3.

Fondamentalement, une fonction se déroule donc sans devoir interagir avec l'utilisateur pour quelque motif que ce soit (acquisition/affichage d'informations, correction, choix à introduire, ...). Ceci laisse toute liberté au concepteur de dialogue quant à la spécification de l'interface tant au niveau de la présentation qu'au niveau de l'enchaînement du dialogue. Si cette condition n'était pas vérifiée au sein de la méthodologie adoptée par le concepteur de l'application fonctionnelle, l'utilisation des modèles du dialogue et de l'échange que nous proposons aux chapitres 3 et 4 ne serait plus possible.

II.2.2 - Structuration des traitements de l'exemple

Le traitement que nous modélisons est l'enregistrement d'une prestation médicale concernant un individu assuré.

Ce traitement est du niveau "phase "; il peut être réalisé à partir d'un terminal, sans changement de lieu, et il présente une certaine unité logique de traitement, en travaillant sur un ensemble de données décrites sur une feuille de soins d'un individu.

Les résultats de cette phase seront l'enregistrement de la prestation dans la base de données et la production à l'écran d'un récapitulatif décrivant cette dernière.

La phase "ENREGISTREMENT D'UNE PRESTATION " peut être divisée en cinq fonctions qui, elles, ne comportent aucune interaction avec l'utilisateur du poste de travail.

Pour effectuer l'enregistrement d'une prestation subie par un individu donné, l'utilisateur du poste de travail devra introduire des informations de deux types : celles qui concernent l'individu et celles qui concernent les soins reçus par celui-ci. Il existera donc une fonction "VALIDATION-INDIVIDU " permettant de vérifier que l'individu auquel se rapporte la prestation existe dans la base de données, et une fonction "VALIDATION-ACTES ". Cette dernière vérifie, pour chaque ligne de prestation de la feuille de soins, que le code-acte qui lui est relatif existe réellement, et attribue des valeurs au numéro d'ordre et à l'attribut "quantité " de cette ligne. Notons que la caractéristique "numéro d'ordre " ne peut être définie qu'en fonction de l'appartenance d'une ligne à un ensemble de lignes constituant la feuille de soins. C'est pourquoi nous avons une seule fonction qui effectue la validation de tous les actes d'une feuille de soins, et non une fonction de validation d'un seul de ces actes qui serait exécutée plusieurs fois pour une même prestation.

La fonction suivante, quant à elle, sera chargée de déterminer certaines caractéristiques des lignes de prestation. Elle devra, en effet, calculer les montants de la base de remboursement, de l'assurance obligatoire, et du ticket modérateur de toutes les lignes de prestation d'une feuille de soins. Cette fonction sera appelée "CALCUL-CARACT-LIGNES-PREST ".

La quatrième fonction effectuera la production d'un récapitulatif de la prestation. Elle sera nommée "PRODUCTION-RECAPITULATIF ".

Enfin, la fonction "ENREGISTREMENT-PRESTATION " réalise l'opération d'enregistrement dans la base de données de toutes les informations inclues dans la feuille de soins.

Sur base de la décomposition fonctionnelle des traitements de l'application, nous pouvons maintenant préciser leur spécification statique.

II.3 - Modèle de la statique des traitements

Le but de ce modèle est de permettre la spécification de chaque traitement d'un niveau donné, d'abord sous la forme d'une boîte noire, dont on précise essentiellement les entrées et les sorties, puis en décrivant les règles de transformation des informations d'entrée en informations de sortie.

II.3.1 - Modèle de la boîte noire

" Ce modèle permet de spécifier pour un traitement d'un niveau donné :

- les *objectifs* à réaliser,
- les *performances* souhaitées,
- les *informations* en entrée et en sortie,
- les *actions primitives* qui caractérisent les relations entre ces informations et les traitements (réception d'une information en entrée, consultation de la mémoire du système d'information, création d'une information en sortie, ajout d'élément(s) à la mémoire du système d'information, suppression ou modification d'élément(s) de la mémoire du système d'information)." [Bodart, Pigneur]

Dans le cadre de ce modèle, nous avons dû particulièrement affiner les notions d'informations en entrée et d'informations en sortie en redéfinissant le concept de message.

Toute information d'entrée d'une fonction, nécessaire à la réalisation du traitement qui lui est associé, correspond à une information :

. soit obtenue par consultation de la mémoire du SI,

. soit fournie par une autre fonction pour laquelle elle constitue dès lors une information de sortie,

. soit introduite directement par l'utilisateur (valeur syntaxiquement correcte car la validation syntaxique a dû être effectuée par le composant "dialogue" de l'application au moment de l'introduction des données au terminal).

Le traitement effectué par la fonction fournit un ou plusieurs des résultats suivants :

. action sur la mémoire du SI,

. communication d'une information de sortie à une autre fonction de l'application,

. production d'une information de sortie à destination de l'utilisateur.

Toute information d'entrée ou de sortie d'une fonction est associée à la notion de "message" telle que nous la définissons maintenant.

Du point de vue des traitements, nous parlerons de messages-données ou *message d'entrée* qui constituent les informations à partir desquelles sont obtenues, par les règles de transformation ou de traitement précisées dans la spécification des fonctions, les informations qui correspondent aux *messages de sortie* ou *messages-résultats* .

Du point de vue des données, nous dirons que toute information appartenant à la mémoire du système d'information a pour origine directe ou indirecte (c'est-à-dire après transformation effectuée par un traitement) les messages véhiculés dans le SI.

Un message est dès lors défini comme véhicule des informations décrites dans la mémoire du SI. Ces informations sont échangées par les supports de communication de l'organisation :

. soit entre le SI et son environnement,

. soit, à l'intérieur du SI, entre les traitements.

Dans le premier cas, le message est dit externe; dans le second cas, il est dit interne.

En réunissant ces deux vues, nous définissons une typologie des messages fonctionnels de la façon suivante:

. un *message externe* est

soit un *message-entrée* , en provenance de l'environnement et à destination d'un traitement du SI (application fonctionnelle)

Exemples : bon de commande émis par un client

message-horaire généré par une horloge,

soit un *message-sortie* , émis par un traitement du SI à destination de l'environnement

Exemples : facture

justificatif de non livraison;

. un *message interne* est un message échangé entre des traitements du SI

Exemple : le message émis par le traitement "enregistrement d'une commande" à destination du traitement "mise-à-jour-stock", en vue de déclencher la mise à jour du stock après enregistrement d'une commande, en fonction des produits commandés qui ont pu faire l'objet d'une livraison.

Ce dernier type de message est fondamentalement non visible pour l'utilisateur puisqu'il est uniquement utilisé par les traitements du SI, comme un message de communication entre deux processus.

Si l'on adopte la décomposition de l'application en traitements de plus en plus élémentaires du type phase/fonction, on distinguera encore les messages-phase des messages-fonction.

En particulier, on peut caractériser les messages externes de la façon suivante:

. puisque la phase est définie comme une unité logique de traitement, les *messages-externes-phase* ont un sens pour l'organisation, c'est-à-dire qu'ils auront une action sur l'organisation. A ce titre, nous les appelons également *messages réels*,

. les *messages-externes-fonction* constituent des unités logiques d'information correspondant plutôt à des unités de communication avec l'environnement. A ce niveau, on parlera par exemple d'unités d'acquisition ou d'affichage d'information au terminal. Les messages-externes-fonction de l'ensemble des fonctions d'une phase présentent en fait une découpe des messages-externes-phase. Il y a donc une correspondance entre ces deux types de messages.

La même différenciation peut se faire au niveau des messages internes si le concepteur de l'application fonctionnelle le juge intéressant.

Cette distinction, sur base du niveau du traitement considéré, permet par ailleurs de préciser la notion d'environnement à laquelle nous avons fait référence pour différencier les messages externes des messages internes.

Quand nous nous plaçons au niveau de la phase, nous pouvons dire, par exemple, que l'environnement est constitué de tous les services destinataires ou émetteurs des messages réels, comme nous le montrerons également au paragraphe III.1.1 en présentant les messages du dialogue.

Au niveau de la fonction, l'environnement correspond plutôt à l'utilisateur qui communique interactivement avec les traitements de l'application via le dialogue.

C'est au niveau des messages-externes-fonction qu'il semble dès lors le plus adéquat d'établir la correspondance avec les unités d'information échangées par les traitements de l'application et l'utilisateur via l'interface.

La spécification fonctionnelle de ce type de messages doit préciser toutes les informations indispensables au concepteur du dialogue pour définir correctement la saisie et l'affichage des informations à échanger avec l'utilisateur.

Au niveau du modèle de la statique des traitements, tout message est principalement défini par son contenu informationnel. Son aspect "Présentation" est défini ultérieurement dans le modèle du dialogue développé au chapitre 3.

Le *contenu informationnel* d'un message peut être structuré à l'aide des concepts du modèle Entité-Association (E-A) : type d'entité, type d'association, attribut et contrainte d'intégrité. On peut donc décrire le contenu des messages sous la forme d'un schéma E - A.

Dans le cas de messages au contenu informationnel fortement structuré (les formulaires notamment), ce schéma pourra être assez complexe. Par exemple, le contenu informationnel d'un formulaire BON-DE-COMMANDE est composé d'attributs des types d'entité COMMANDE (Numéro-commande, Date-commande, Montant-commande), CLIENT (Numéro-client, Identité-client, Adresse-client) et PRODUIT (Numéro-produit, Libellé-produit, Prix-produit), d'attributs du type d'association LIGNE-DE-COMMANDE (Quantité-Ligne-Commande, Montant-Ligne-Commande) et doit respecter les contraintes d'intégrité définies pour ces différents éléments dans le schéma conceptuel des données.

Dans le cas de messages au contenu informationnel faiblement structuré (note écrite, communication orale, image), le schéma E - A présentant ce contenu est dégénéré : il ne comprend qu'un type d'entité dont la valeur d'un attribut est le contenu même du message (texte d'une note de service par exemple).

Les messages peuvent ainsi être considérés comme une structure dérivée des types de base que sont les types d'entité et les types d'association qui les composent. Cependant, cette structure dérivée peut, elle aussi, posséder des contraintes d'intégrité propres telles que des contraintes sur les valeurs admises pour les attributs que le message contient, compte tenu du contexte dans lequel ce dernier est émis ou reçu. Ces valeurs peuvent ainsi varier en fonction du traitement auquel le message qu'elles concernent est lui-même lié.

Par exemple, lors de l'enregistrement d'une commande d'un client qui ne possède pas de numéro de client (ce client est dès lors assimilé à un nouveau client dont la description n'a pas encore été enregistrée dans la base de données), l'adresse du client doit être obligatoirement introduite. Par contre, quand un client est identifié par un numéro de client, on considère que le client est déjà mémorisé dans la base de données avec une adresse valide et, sauf indication explicite de modification d'adresse sur le formulaire du bon de commande, l'adresse qui s'y trouve ne doit pas nécessairement être introduite.

En fonction du type de message à décrire, le contenu informationnel doit donc comporter certains éléments spécifiques, en plus de la définition des éléments qui composent le message.

Ainsi, comme nous l'avons déjà souligné, la spécification fonctionnelle d'un message-entrée précise les valeurs syntaxiquement admissibles pour ces données au moment de leur introduction, compte tenu de la fonction à laquelle elles sont destinées, puisque la validation syntaxique des données introduites par l'utilisateur au terminal doit être effectuée par le dialogue.

Quant aux messages-sortie, leur spécification diffère selon le rôle qu'il joue. Nous en avons distingué deux :

. le message *résultat "pur"* vise à transmettre à l'utilisateur les résultats de l'exécution d'un traitement dans la mesure où ce message ne résulte pas d'un traitement d'exception (erreur, par exemple) qui demande l'intervention de l'utilisateur pour décider de la suite à lui donner (abandon du traitement , correction, retour-arrière,...). La spécification du seul contenu informationnel d'un tel message est suffisante pour permettre au concepteur de dialogue de définir sa sortie,

. le *message d'erreur* signale à l'utilisateur l'invalidation sémantique d'une ou plusieurs données d'un ou plusieurs messages-entrée. La spécification fonctionnelle de ce type de message doit définir le ou les points de retour-arrière qui sont nécessaires à la correction de la ou les données invalidées, au moment où l'utilisateur décide de corriger les données introduites. Cette notion, particulièrement complexe, est introduite au paragraphe II.4.4 dans le cadre du modèle de la dynamique des traitements et est plus longuement explicitée au cours du chapitre 3 qui concerne la modélisation du dialogue.

Toutefois, on peut dès à présent préciser qu'un message d'erreur sera défini par :

- un libellé reprenant le justificatif de l'invalidation effectuée sur certaines données,
- le ou les champs du ou des messages externes invalidés, ceci afin de déterminer le ou les points de retour à utiliser. Leurs conditions d'utilisation (combinaisons "et/ou ") devront également être précisées ici. En effet, un message d'erreur peut nécessiter la correction de plusieurs champs de messages (combinaison "et ") ou la correction d'un seul champ parmi plusieurs, si l'erreur est due à un problème de cohérence des données (combinaison "ou ").

La typologie des messages de l'application fonctionnelle est résumée par la figure 3.

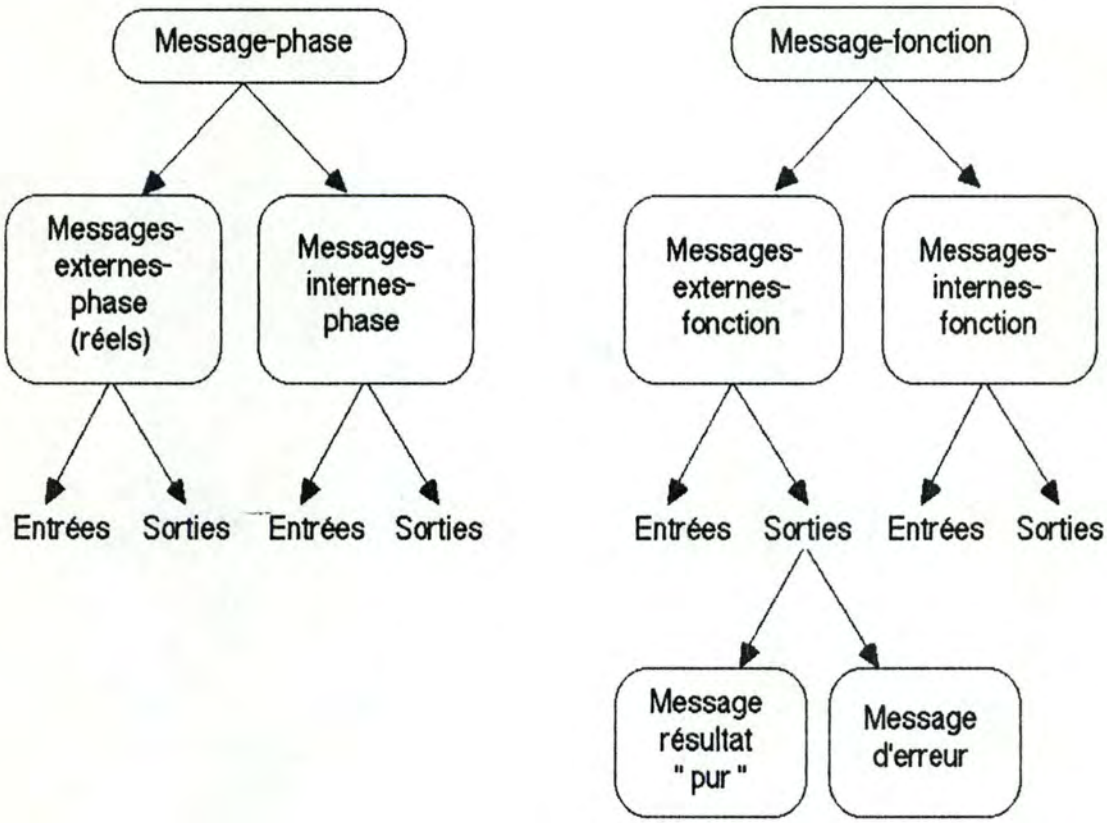


Figure 3 - Les messages de l'application fonctionnelle

II.3.2 - Règles de traitement

Le deuxième objectif de ce modèle est de compléter " la spécification de chaque traitement, normalement de niveau fonction, en décrivant les règles d'obtention des informations de sortie à partir des informations d'entrée." [Bodart, Pigneur]

Pour ce faire, le concepteur de l'application fonctionnelle dispose de plusieurs techniques ou langages, plus ou moins stricts ou formels. Ceux-ci vont de la spécification en langue naturelle à l'utilisation de tables de décision combinatoires, en passant par la formalisation mathématique (langages algébriques), les pseudo-codes, ...

L'un ou l'autre de ces langages peut être indifféremment utilisé. Nous n'introduisons aucune contrainte en ce qui les concerne puisque nous n'avons imposé aucune méthodologie pour la spécification des traitements. Celle-ci relève donc uniquement de la modélisation choisie par le concepteur de l'application fonctionnelle.

II.3.3 - Description statique des traitements de l'exemple

Nous allons successivement décrire la phase entière et chacune des fonctions déterminées lors de la découpe en niveaux des traitements (cfr paragraphe II.2.2).

II.3.3.1 - Phase "Enregistrement d'une prestation "

Objectif du traitement : - enregistrer une prestation relative à un individu,
- afficher un récapitulatif de la saisie.

Consulte dans la BD : les types d'entité Individu et Acte

Ajoute dans la BD : une entité de type Prestation
une association de type Subir
une ou plusieurs associations de type Ligne-Prestation

Message d'entrée : Une feuille de soins dont le contenu est décrit ci-après.

MESSAGE feuille de soins
CONSTITUE DE : - nom de l'assuré
- prénom
- date de naissance
- matricule
- date de prestation des soins
- 1 à N lignes constituées de : - code-acte
- montant-dr-acte
- quantité-acte.

Message produit en sortie : Un récapitulatif dont le contenu est le suivant :

MESSAGE récapitulatif

CONSTITUE DE : - nom de l'assuré
- prénom
- matricule
- date de naissance
- date de prestation des soins
- 1 à N lignes constituées de : - code-acte
  - quantité-acte
  - montant-dr-acte
  - montant-br-acte
  - montant-tm-acte
  - montant-ao-acte

Règles de traitement : - Le numéro d'ordre de la "Ligne-prestation " est attribué par compostage pour une prestation d'un individu.
- Le "montant-br " de la ligne de prestation a la valeur de l'attribut "base-remboursement-acte " de l'acte auquel la ligne est relative.
- Le calcul du ticket modérateur ("montant-tm ") s'effectue comme suit :

  Si le caractère inclu dans le "code-acte" auquel la ligne se rapporte est différent de la lettre "K", la valeur du montant-tm = 70; sinon, si le nombre inclu dans ce même "code-acte" est

  < 10 , montant-tm = 40;
  10 <= <= 29, montant-tm = 50;
  30 <= <= 49, montant-tm = 60;
  >= 50, montant-tm = 100.
- Le calcul de l'assurance est réalisé selon la formule suivante : montant-ao = (1 - (montant-tm / 100))

II.3.3.2 - Fonction "Validation-individu "

Objectif du traitement : - vérifier les informations relatives à un individu.

Message d'entrée : Un message, dont le contenu est décrit ci-après, décrivant un individu assuré.

MESSAGE données-individu
CONSTITUE DE : - nom de l'assuré
- prénom
- date de naissance
- matricule

Consulte dans la BD : une entité de type Individu

Messages produits en sortie : Si l'individu fourni par l'utilisateur existe dans la base de données, la fonction produit en sortie un message "individu-valide "; dans le cas contraire, elle produit un message d'erreur "individu-non-valide ". Le contenu de ces messages est le suivant :

MESSAGE individu-valide
CONSTITUE DE : - nom de l'assuré
- prénom
- matricule
- date de naissance

MESSAGE individu-non-valide
CONSTITUE DE : - un texte expliquant que l'individu n'existe pas dans la base de données de l'organisme de mutuelle,
- une indication de retour à la saisie des caractéristiques de l'individu.

II.3.3.3 - Fonction "Validation-actes "

Objectif du traitement : - vérifier les informations relatives aux actes de la prestation.

Référence dans la BD : plusieurs entités de type Acte

Message d'entrée : Un message, dont le contenu est décrit ci-après, décrivant les lignes d'une feuille de soins.

MESSAGE lignes-feuille-soins
CONSTITUE DE : - date de prestation
- 1 à N lignes contenant : - code-acte
- montant-dr-acte
- quantité-acte.

Messages produits en sortie : Si les codes des lignes de prestation fournies par l'utilisateur existent dans la base de données, la fonction produit en sortie un message "lignes-valides "; dans le cas contraire, elle produit un message d'erreur "lignes-non-valides ". Le contenu de ces messages est le suivant :

MESSAGE lignes-valides
CONSTITUE DE : - date de prestation
- 1 à N lignes contenant : - code-acte
- montant-dr-acte
- quantité-acte.

MESSAGE lignes-non-valides
CONSTITUE DE : - un texte expliquant qu'un ou plusieurs "code-acte" n'existent pas dans la base de données de l'organisme de mutuelle,
-une indication de retour à la saisie des lignes de prestation erronnées.

II.3.3.4 - Fonction "Calcul-caract-lignes-prest"

Objectif du traitement : - calculer, pour chaque ligne d'une prestation, ses montants base de remboursement, assurance obligatoire et ticket modérateur.

Consulte dans la BD : plusieurs entités de type Acte

Message d'entrée : c'est le message "lignes-valides " décrit ci-dessus.

Message produit en sortie : une version complétée du message d'entrée, que nous appelons "lignes-complétées.

MESSAGE lignes-complétées
CONSTITUE DE : 1 à N lignes contenant : - code-acte
- montant-dr-acte
- montant-br-acte
- montant-ao-acte
- montant-tm-acte
- quantité-acte.

Règles de traitement : - Le "montant-br " de la ligne de prestation a la valeur de l'attribut "base-remboursement-acte " de l'acte auquel la ligne est relative.
- Le " montant-ao " = (1 - (montant-tm / 100)).
- Le "montant-tm " est calculé au moyen des règles décrites dans la table de décision présentée ci-dessous.

caractère du code-acte < > K	X				
caractère du code-acte = K		X	X	X	X
nombre du code-acte < 10		X			
10 <= nombre du code-acte <= 29			X		
30 <= nombre du code-acte <= 49				X	
nombre du code-acte >= 50					X
montant-tm = 70	X				
montant-tm = 40		X			
montant-tm = 50			X		
montant-tm = 60				X	
montant-tm = 100					X

II.3.3.5 - Fonction "Production-récapitulatif "

Objectif du traitement : - fournir à l'utilisateur un récapitulatif des données concernant la prestation qu'il vient d'introduire au terminal.

Message d'entrée : Les messages "lignes-complétées " et "individu-valide " qui ont été décrits précédemment.

Message produit en sortie : Un récapitulatif dont le contenu est le suivant :

MESSAGE récapitulatif
CONSTITUE DE : - nom de l'assuré
- prénom
- matricule
- date de naissance
- date de prestation des soins

- 1 à N lignes constituées de : - code-acte
  - quantité-acte
  - montant-dr-acte
  - montant-br-acte
  - montant-tm-acte
  - montant-ao-acte

II.3.3.6 - Fonction "Enregistrement-prestation "

Objectif du traitement : - enregistrer dans la base de données toutes les informations qui concernent une feuille de soins valide.

Message d'entrée : Les messages "lignes-complétées " et "individu-valide " qui ont été décrits précédemment.

Ajoute dans la BD : une entité de type Prestation
une association de type Subir
une ou plusieurs associations de type Ligne-prestation

Après avoir introduit le schéma conceptuel des données d'un système d'information, structuré puis spécifié statiquement les traitements de l'application fonctionnelle, nous allons maintenant détailler le modèle de la dynamique des traitements qui analyse leurs conditions de déclenchement, d'exécution et d'enchaînement.

II.4 - Modèle de la dynamique des traitements

L'objectif du modèle de la dynamique des traitements est de fournir au concepteur d'application une manière de décrire l'enchaînement des traitements et leurs conditions de déclenchement lors de l'exécution de l'application.

Nous en précisons ici les éléments essentiels car, d'une part, les schémas de la dynamique des traitements peuvent être utiles au concepteur de dialogue pour définir les schémas de la dynamique du dialogue et d'autre part, certaines structures d'enchaînement définies dans le modèle de conversation du dialogue sont similaires aux structures d'enchaînement détaillées ici.

II.4.1 - Définition des concepts d'événement et de processus

Le modèle comprend deux concepts de base : l'événement et le processus. Un *événement* correspond à un changement d'état du système d'information. Quant au *processus* , c'est l'exécution d'une procédure de traitement de l'information. La procédure est considérée à ce stade comme une " boîte noire " dont seuls les effets ayant une incidence sur le comportement du système d'information sont significatifs.

La progression d'un processus peut être observée par certains changements d'état. Les trois principaux changements d'état caractérisant le cycle de vie d'un processus sont :

- le déclenchement qui correspond à la création du processus et à sa mise en attente des ressources nécessaires à son exécution,
- l'activation qui est sa mise en route réelle, après obtention des ressources ,
- la terminaison qui survient lorsque toutes les actions prévues dans la procédure du processus ont été accomplies.

Un événement joue le rôle de stimulus auquel le système d'information répond en déclenchant certains processus. La survenance d'un événement dit de base peut correspondre :

- soit à la génération d'un message (interne ou externe), donc d'une information véhiculée dans le système d'information,
- soit à un changement d'état d'un processus (déclenchement, activation, terminaison).

Par convention, on représente graphiquement un événement par un point et une relation de déclenchement par une flèche. La figure 4 illustre les types les plus courants de déclenchement d'un processus.

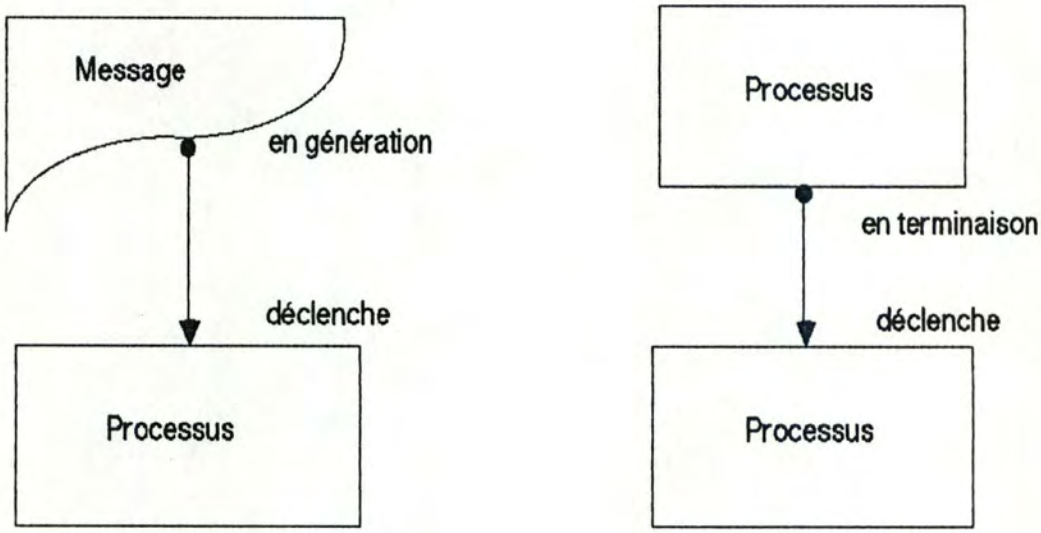


Figure 4 - Types de déclenchement d'un processus.

A gauche, un processus est déclenché par l'événément " génération d'un message".

A droite, un processus est déclenché par la terminaison du processus précédent.

II.4.2 - Structures d'enchaînement des processus

Les structures élémentaires d'enchaînement des traitements sont au nombre de six :

- l'enchaînement séquentiel,
- l'enchaînement éclaté,
- l'enchaînement multiple,
- l'enchaînement conditionnel,
- l'enchaînement convergent,
- l'enchaînement synchronisé.

Passons brièvement ces structures en revue.

Un enchaînement est dit *séquentiel* lorsque la terminaison d'un processus P1 provoque le déclenchement d'un processus P2.

Cet enchaînement peut être vu de deux manières différentes. L'enchaînement dynamique classique considère que c'est la terminaison de P1 elle-même qui déclenche P2. La deuxième solution, appelée enchaînement dynamique par les messages, considère que c'est un message M1 de fin d'exécution de P1, produit à la terminaison de ce processus, qui déclenche P2. Ces deux types d'enchaînement séquentiel sont illustrés à la figure 5.

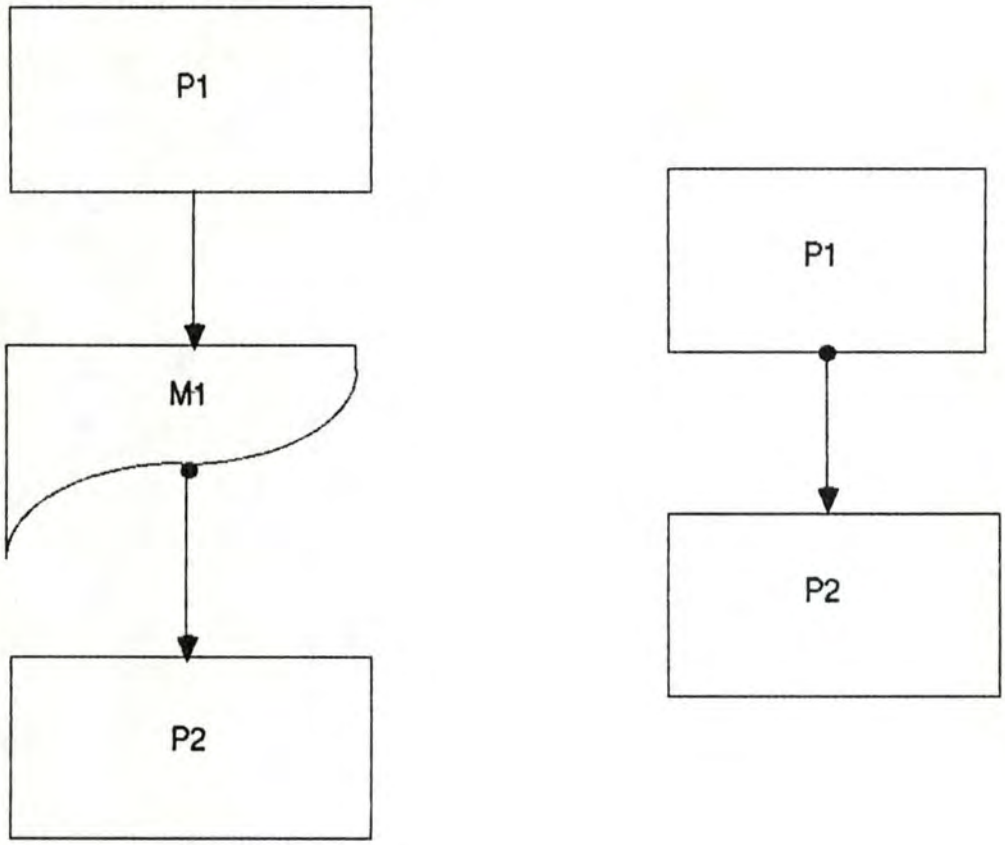


Figure 5 - Les deux types d'enchaînement séquentiel des processus. L'enchaînement dynamique par les messages, à gauche, face à la dynamique "classique", à droite.

Un enchaînement est dit *éclaté* lorsque la terminaison d'un processus P1 provoque le déclenchement simultané de plusieurs processus de types distincts P2, P3, ..., Pn. De nouveau, une possibilité existe de décrire cet enchaînement dynamique par les messages. Cette possibilité existe également pour tous les types d'enchaînement décrits dans la suite; nous ne reviendrons plus sur ce point.

Un enchaînement est *multiple* si la terminaison d'un processus P1 provoque le déclenchement simultané de plusieurs processus de type P2.

Un enchaînement est *conditionnel* lorsque la terminaison d'un processus P1 provoque le déclenchement d'un processus P2 en cas de réalisation d'une condition et le déclenchement d'un processus P3 en cas de non réalisation de cette condition.

Un enchaînement est *convergent* si le déclenchement d'un processus Pi est provoqué par la terminaison d'un processus P1 ou d'un processus P2 ou ... ou d'un processus Pn.

Pour terminer, un enchaînement est dit *synchronisé* lorsque le déclenchement d'un processus Pi est provoqué par l'arrivée de plusieurs événements (arrivée de messages et/ou terminaison d'autres processus). Le déclenchement de Pi ne sera réalisé que lorsque tous les événements participant à la synchronisation se seront produits.

Toutes ces structures d'enchaînement sont décrites de manière très complète dans [Bodart, Pigneur]. En outre, les auteurs en fournissent une représentation graphique.

II.4.3 - Schéma d'enchaînement

A ce modèle de la dynamique des traitements d'un système d'information correspond un schéma d'enchaînement des processus. Dans le cadre de la modélisation décrite dans [Bodart, Pigneur], les processus peuvent correspondre soit à des phases soit à des fonctions, suivant le niveau d'agrégation auquel le concepteur décide d'étudier la dynamique des traitements. Le schéma représentera donc soit l'enchaînement des phases de l'application, soit l'enchaînement des fonctions d'une même phase.

Dans ce cadre, les événements *initiaux* sont ceux qui ne sont pas produits par un processus appartenant au schéma et qui, malgré cela, contribuent au déclenchement de certains processus appartenant à ce schéma. De manière similaire, les événements *terminaux* sont ceux qui, étant produits par des processus faisant partie du schéma, ne participent au déclenchement d'aucun de ces processus.

L'arrivée d'un message en provenance de l'utilisateur via le dialogue correspond donc à un événement initial, de même que la génération d'un message à destination de l'utilisateur constitue un événement terminal.

II.4.4 - Retour-arrière en cours d'exécution

La définition des six structures élémentaires d'enchaînement des traitements présentées au paragraphe II.4.2, ainsi que l'interprétation des schémas de la dynamique qui sont construits à partir de ces structures, doivent être complétées par la définition de la notion de retour-arrière pendant l'exécution d'une phase.

Nous avons déjà signalé au paragraphe II.3.1 que la spécification fonctionnelle d'un message d'erreur concernant l'invalidation sémantique (donc par les traitements) d'une ou plusieurs données d'un ou plusieurs messages-entrée (en l'occurence, de messages-externes-fonction) devait préciser le ou les points de retour-arrière nécessaires à la correction de la ou des données invalidées.

Indépendamment des modalités du dialogue permettant l'acquisition des données corrigées (ces modalités sont évoquées dans le chapitre 3 traitant de la modélisation du dialogue), on peut néanmoins remarquer qu'au niveau des traitements, l'introduction de ces données aura simplement pour conséquence la génération d'une nouvelle occurence du ou des messages-externes-fonction contenant une ou plusieurs données invalidées.

Puisque le retour-arrière dans le déroulement d'une phase doit se faire, dans le schéma d'enchaînement associé à cette phase, au niveau d'un ou plusieurs de ses messages-externes-fonction, nous considérons que tout message de ce type, en provenance de l'utilisateur et contenant au moins une donnée à valider sémantiquement, correspond à un point de retour éventuel dans le schéma d'enchaînement des traitements.

Notons également qu'étant donné que la spécification fonctionnelle des messages d'erreur doit préciser le ou les points de retour à utiliser ainsi que leurs conditions d'utilisation (combinaisons et/ou) compte tenu du ou des champs du ou des messages externes invalidés (cfr paragraphe II.3.1), il semble superflu de représenter spécifiquement les points de retour-arrière dans le schéma de la dynamique des traitements car cela l'alourdirait inutilement et s'avérerait rapidement assez complexe.

Enfin, il faut préciser d'une part qu'un retour-arrière en cours d'exécution ne pourra jamais se faire qu'à l'intérieur d'une phase et d'autre part qu'il nécessitera éventuellement le "détricotage" des traitements exécutés dans le cadre de la phase concernée, jusqu'au point de retour effectivement utilisé par l'utilisateur.

II.4.5 - Quantification du modèle

Pour que la description de ce modèle de la dynamique soit complète, il nous reste à parler de ses éléments de quantification. Ceux-ci permettent de donner une évaluation de la durée de chaque processus et de la fréquence de survenance de chaque événement. Ils permettent également de fixer la probabilité de réalisation des conditions faisant partie du schéma.

II.4.6 - Enchaînement dynamique des traitements de l'exemple

Le schéma qui suit exprime les contraintes d'enchaînement des traitements du cas "mutuelle ". Nous avons choisi ici de représenter la dynamique par les messages, car elle exprime de manière plus claire la circulation des données entre les différents traitements. Dans ce schéma, les messages dont le contour est en trait continu représentent les messages externes qui proviennent de l'utilisateur ou lui sont destinés, tandis que les messages au pourtour pointillé représentent les messages internes.

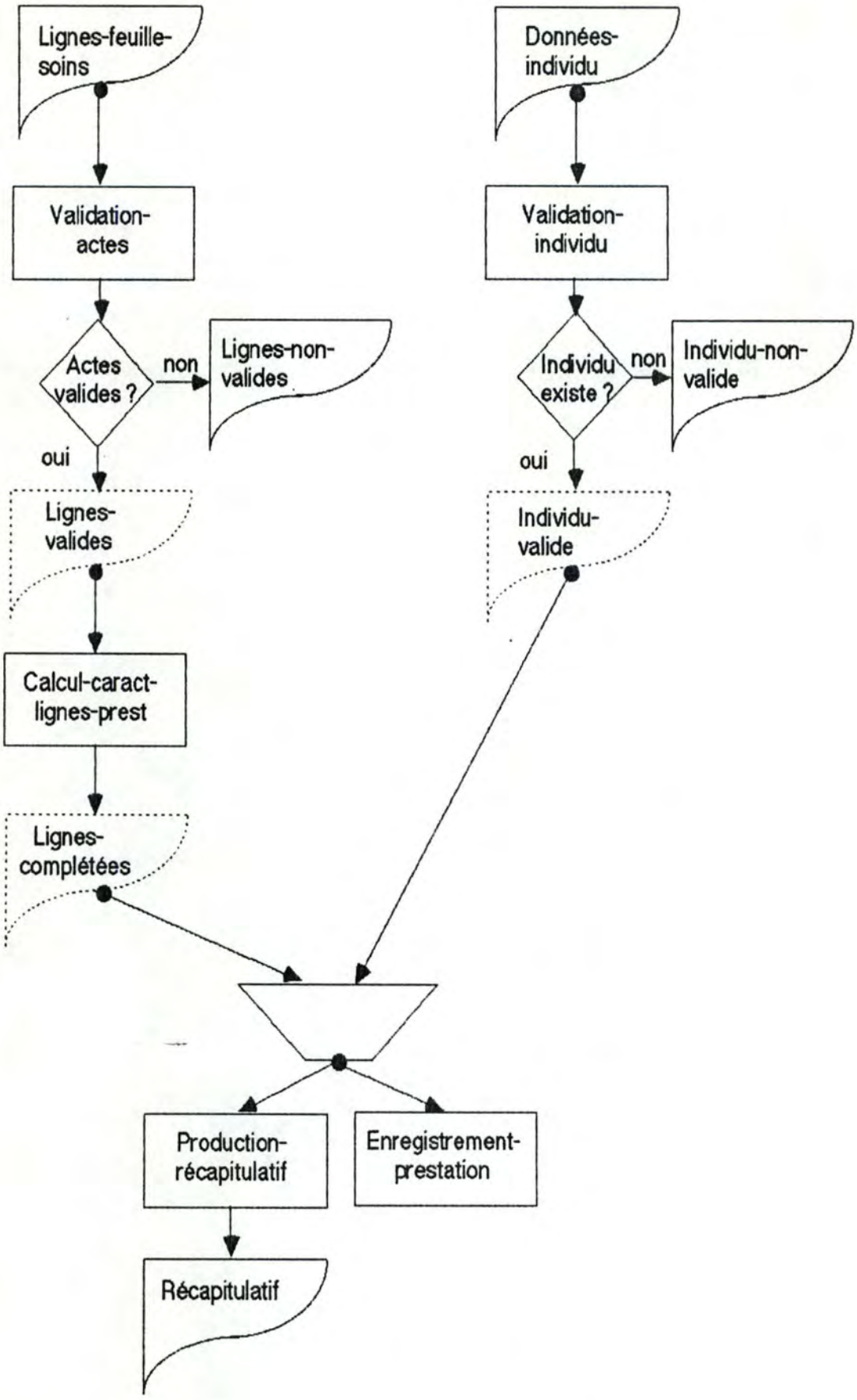


Figure 6 - Enchaînement dynamique des traitements de l'exemple "mutuelle "

Après avoir décrit les données d'une application interactive et les traitements qu'elle doit effectuer, il nous reste à modéliser les ressources dont elle dispose.

II.5 - Modèle des ressources

Pour mémoire, nous rappelons les principaux concepts du modèle des ressources. Pour plus d'informations à ce sujet, le lecteur peut consulter [Bodart, Pigneur].

Le modèle des ressources a pour but de déterminer clairement les ressources qui seront disponibles lors de l'exécution de l'application. En effet, il est important de ne pas oublier de fournir à l'application toutes les ressources (humaines, matérielles et financières, par exemple) qui lui seront nécessaires, pour que la solution conceptuelle décrite par les précédents modèles soit réalisable.

Dans ce dernier modèle, nous distinguons deux types de ressources : les ressources réutilisables ou processeurs et les ressources consommables.

Un *processeur* est une *ressource réutilisable* qui peut être requise lors de l'exécution d'un processus . Les ressources réutilisables sont celles qui sont à nouveau disponibles dès la terminaison du processus par lequel elles ont été requises. Par exemple, une personne, un terminal, un CPU ainsi que la base de données de l'application sont des processeurs.

A l'inverse, une *ressource consommable* disparaît après la terminaison du processus qui l'a requise; elle n'est réutilisable par aucun autre processus . Le papier d'impression, l'énergie, l'argent sont des ressources de ce type.

Les ressources sont utilisées par les processus de la manière suivante : un processeur est requis pour un certain temps par un processus et une certaine quantité de ressources consommables est utilisée par ce même processus . Cette relation de réquisition/consommation permettra de situer les traitements qui nécessitent une quantité importante de ressources.

II.6 - Un langage de spécification

Nous avons décrit ici les composants essentiels des différents modèles permettant de décrire les données et les traitements d'une application. Remarquons cependant que la modélisation est soutenue par un langage de spécification; celui-ci permet de décrire en détail les différents modèles. Le langage décrit dans [Bodart, Pigneur] s'appelle DSL (Dynamic Specification Language).

Le langage DSL présente les caractéristiques suivantes :

- c'est un langage textuel de description des spécifications d'un système d'information,
- il est non procédural c'est-à-dire qu'il n'impose aucun ordre pour la spécification du système d'information,
- il est basé sur le modèle Entité - Association; ce langage connait en effet un certain nombre de types d'objets (entités), autorise certaines relations entre ces objets (associations), et permet d'exprimer des propriétés associées à ces types d'objet et de relation (attributs, contraintes d'intégrité).

Le lecteur intéressé trouvera une description du langage DSL dans [Bodart, Pigneur].

Signalons que des travaux ayant pour but de remplacer le langage textuel DSL par un langage graphique sont en cours [Crespin, Piquard]. Un tel langage faciliterait grandement le travail de spécification du concepteur d'application en lui permettant de visualiser, sous forme de schémas, la description des données et des traitements de l'application.

II.7 - En résumé

Ayant décrit les données sur lesquelles l'application se base et les traitements qu'elle est chargée d'effectuer ainsi que leurs conditions de déclenchement et d'enchaînement, ayant précisé les ressources dont elle pourra et devra disposer en cours d'exécution, nous avons constitué un modèle complet des fonctionnalités de l'application interactive.

Nous avons signalé les possibilités d'utilisation du langage DSL comme langage de spécification basé sur les concepts du modèle présenté.

Nous pouvons maintenant passer à la modélisation de la seconde composante de l'application interactive : le dialogue.

Chapitre 3 :

La modélisation du dialogue

Introduction

La modélisation du dialogue d'une application interactive se fait en fonction :

- d'une source d'information (la spécification de l'application fonctionnelle) qui indique ce sur quoi doit porter le dialogue et les contraintes - d'enchaînement notamment - qu'il doit impérativement respecter,
- d'une cible de mise en oeuvre représentée par un terminal virtuel,
- de choix associés à des considérations d'ergonomie matérielle et logicielle.

Nous avons vu précédemment une manière de modéliser les traitements d'une application interactive. Nous possédons donc la source d'information nécessaire à la modélisation du dialogue d'une telle application. Dans ce chapitre, nous allons passer en revue les différentes facettes du dialogue. La spécification de celui-ci a pour rôle de décrire entièrement ce que l'utilisateur verra apparaître à l'écran de son poste de travail dans le cadre de l'application qu'il utilise.

Les messages qui paraissent à l'écran sont appelés des messages interactifs. Ces messages interactifs sont présentés à l'aide d'objets interactifs tels que la fenêtre, l'icône, ...

Les objets du dialogue étant ainsi décrits, nous passerons à l'énumération de ses fonctions, dont les deux plus importantes sont évidemment la saisie et l'affichage des messages.

Connaissant les objets et fonctions sur lesquels s'appuie tout dialogue, nous aborderons ensuite la description de l'enchaînement des messages interactifs qui constituent l'interface d'une phase interactive de l'application et les attributs de visualisation de ces messages. Nous appellerons respectivement ces deux composantes d'une interface la conversation et la présentation de l'application interactive.

Les idées développées dans ce chapitre sont inspirées des ouvrages suivants : [OMEGA 86-1], [Coutaz 86-3], [Coutaz 87], [Lermigeaux], ainsi que du toolbox Macintosh et du SGD DIALOGUE de l'Apollo.

Dans le cadre de ce mémoire, nous avons décidé de restreindre notre étude aux enchaînements de messages qui peuvent exister à l'intérieur d'une phase. Les enchaînements entre différentes phases ne sont donc pas pris en compte.

III.1 - Les objets du dialogue

Le dialogue est basé sur deux sortes d'objets : les messages interactifs et les objets interactifs. Nous allons les étudier successivement.

III.1.1 - Les messages interactifs

Tous les messages de l'application que l'utilisateur voit apparaître à son poste de travail sont des *messages interactifs.* Les messages interactifs sont des unités de saisie ou d'affichage d'informations qui ont un sens pour l'utilisateur. Ces messages interactifs se manifestent, pour la plupart, à l'écran du terminal. Cependant, il pourra arriver que certains d'entre eux soient envoyés vers une imprimante ou vers tout autre appareil périphérique. Cela ne se produira que pour des messages-sortie qui ne demandent aucune réponse de la part de l'utilisateur. La production d'une facture imprimée est un exemple d'un tel message interactif.

Les messages interactifs peuvent être classés en plusieurs catégories, mais souvenons-nous tout d'abord des différents types de message précisés au paragraphe II.3.1 décrivant le modèle de la statique des traitements de l'application fonctionnelle.

Les messages de l'application fonctionnelle peuvent être définis avec différents degrés de granularité; nous avons ainsi distingué les messages-phase et les messages-fonction.

Les messages internes sont véhiculés entre les traitements de l'application; ils sont donc a priori invisibles à l'utilisateur et n'interviennent pas dans le dialogue.

Quant aux messages externes, nous savons qu'ils proviennent ou sont destinés à l'environnement du système d'information. Dans le cadre de la modélisation d'une phase interactive, l'environnement est principalement composé du poste de travail de l'utilisateur, mais d'autres personnes faisant partie de l'organisation peuvent, par exemple, recevoir également certains messages externes.

Les messages externes peuvent être des messages d'entrée ou de sortie. Parmi les messages de sortie du niveau fonction, nous avons encore distingué les messages résultat "purs" des messages d'erreur. Cette typologie des messages-externes- fonction nous aidera à classifier les messages interactifs.

D'après la définition que nous avons donnée des messages externes, on constate que la plupart d'entre eux seront visualisés au poste de travail de l'utilisateur. Ils correspondront donc à des messages interactifs.

Les messages-externes-phase présentent de façon agrégée les informations d'une phase qui proviennent ou sont destinées à l'environnement de cette phase. Si le concepteur de dialogue se base sur ces messages pour concevoir l'interface de l'application, il sera obligé de fournir à la phase tous les messages externes d'entrée qu'elle nécessite avant le déclenchement de celle-ci, puisqu'il ne sait pas à quels sous-traitements (à quelles fonctions) de la phase sont destinés ces messages externes. De plus, les messages d'erreur ne sont pas précisés au niveau global de la phase.

Par contre, si le concepteur de dialogue se base sur la spécification des messages-externes-fonction pour décrire son dialogue, il pourra, s'il le désire, tenir compte de l'ordre dans lequel les traitements utilisent les messages pour déterminer le scénario de saisie et d'affichage des messages interactifs.

Les messages interactifs sont divisés en deux grandes catégories selon qu'ils correspondent ou non (par composition/décomposition) à un message-externe-fonction des traitements de l'application fonctionnelle. Ces deux catégories vont encore être subdivisées plusieurs fois. La hiérarchie des différents types de messages interactifs est présentée à la figure 7.

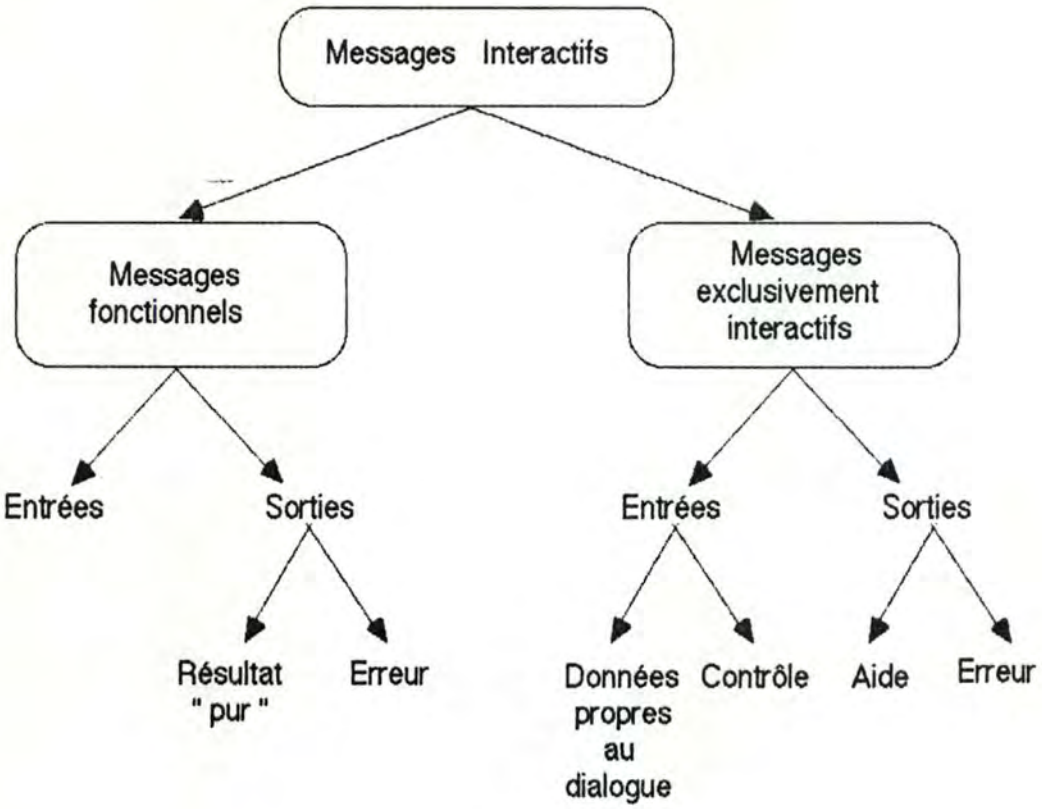


Figure 7 - Les types de messages interactifs.

La première classe de messages interactifs est celle des *messages fonctionnels*. Le contenu de ces messages trouve une correspondance parmi les messages-externes-fonction de l'application fonctionnelle. Cette correspondance sera exprimée de manière très précise par le modèle de l'échange que nous développons au chapitre 4.

Les messages-externes-fonction doivent être divisés ou regroupés pour former des messages interactifs d'entrée ou de sortie qui représentent, aux yeux de l'utilisateur, les unités du dialogue. A cet endroit de la définition des messages, le concepteur du dialogue devra tenir compte du ou des types d'utilisateurs auxquels est destinée l'application; les unités de dialogue, définies par les messages interactifs, devront avoir un sens pour ces utilisateurs.

Un *message fonctionnel d'entrée* est un ensemble d'informations nécessaires aux traitements de l'application. C'est également une unité de saisie ayant un sens pour l'utilisateur du poste de travail. Un tel message correspond à une ou plusieurs parties d'un ou plusieurs messages externes d'entrée.

Un *message fonctionnel de sortie* est constitué d'informations originaires de l'application fonctionnelle. C'est un résultat fourni par les traitements. Ces derniers peuvent produire deux types de message-résultat, que nous avons nommés, au chapitre 2, résultat "pur " et message d'erreur. Les *messages interactifs de résultat "pur "* sont des unités d'affichage d'information constituées à partir d'un regroupement ou d'une découpe des messages externes résultat "pur ". Quant aux *messages d'erreur de sortie,* ils correspondent aux messages d'erreur produits par les traitements de l'application. Ils indiquent la cause de l'erreur ainsi que son origine, c'est-à-dire l'endroit où l'utilisateur devra retourner pour effectuer les corrections nécessaires afin de relancer les traitements qui ont été annulés (le point de retour).

La seconde classe de messages interactifs est constituée de messages dont l'application fonctionnelle n'a pas connaissance. Ce sont des messages propres au dialogue, que nous appellerons *messages exclusivement interactifs.* Comme dans le cas des messages fonctionnels, ils peuvent être divisés en deux types : les messages d'entrée et les messages de sortie.

Les *messages d'entrée exclusivement interactifs* peuvent être des messages de contrôle ou des messages permettant l'introduction de certains paramètres du dialogue.

Les *messages de contrôle du dialogue* servent à orienter le déroulement de l'application. Ils permettent à l'utilisateur de choisir les traitements qu'il désire effectuer et/ou l'ordre de saisie des messages interactifs. Les messages de contrôle du dialogue sont généralement implémentés sous forme de menus, de questions-réponses, ou de langages de commandes.

Les *messages de données du dialogue* servent à introduire certains paramètres qui ne concernent que le dialogue de l'application interactive. Ils peuvent être utiles, par exemple, pour l'introduction de certaines caractéristiques de l'utilisateur qui permettraient de choisir le type de dialogue le plus approprié à ce dernier, si l'interface fournie par le concepteur de dialogue le permet.

Les *messages de sortie exclusivement interactifs* sont également classés en deux catégories : les messages d'erreur du dialogue et les messages d'aide.

Les *messages d'erreur du dialogue* doivent porter sur un champ d'un message interactif. En effet, nous avons déjà signalé au chapitre précédent que l'application fonctionnelle reçoit des messages externes d'entrée syntaxiquement corrects; il est donc du ressort de la composante "Dialogue " de l'application interactive d'effectuer les vérifications syntaxiques nécessaires et, le cas échéant, de fournir à l'utilisateur un message d'erreur exclusivement interactif concernant un message qui peut être de type fonctionnel ou exclusivement interactif.

Quant *aux messages d'aide,* ils peuvent porter sur le dialogue, sur les fonctionnalités de l'application, ou sur l'état du contexte dans lequel se déroule l'application. Dans le premier cas, ils concernent un ou plusieurs champs d'un message interactif, ou même l'entièreté d'un tel message. Dans le second cas, ils portent sur un traitement de l'application, ou sur l'enchaînement de plusieurs traitements. Dans le troisième, ils permettent à l'utilisateur de répondre à des questions telles que : " Où suis-je ? ",...

L'aide concernant les messages interactifs que l'utilisateur doit compléter est fondamentale. Elle explique la démarche à suivre, présente les valeurs autorisées pour le remplissage d'un champ,... Un message d'aide de ce type peut être rattaché au message interactif (ou à la partie de message) qu'il explicite.

Les explications sur le contexte (données courantes, commandes courantes, historique) peuvent être très utiles, principalement dans le cas où l'utilisateur a interrompu son travail pendant un certain temps.

L'aide portant sur les traitements ne sera fournie qu'à des utilisateurs actifs, s'intéressant à la manière dont l'application se déroule. Ces derniers désireront avoir accès à la spécification des traitements de l'application, ou à la description de leur enchaînement.

Nous venons de décrire les différents types de messages interactifs. Chaque message de l'application apparaissant à l'écran sera une occurence de l'un de ces types.

Nous allons maintenant définir les objets interactifs, qui constituent la seconde famille d'objets du dialogue.

III.1.2 - Les objets interactifs

Les objets interactifs sont des objets de dialogue tels que la fenêtre, l'icône, le menu, le camembert,... Ces objets interactifs seront utilisés pour réaliser la partie présentation des messages interactifs et permettre ainsi de visualiser de manière ergonomique le contenu de ces messages.

Tout objet interactif peut être décrit au moyen du modèle PAC (Présentation-Abstraction-Contrôle) de Joëlle Coutaz. La Présentation de l'objet est la façon dont il se comporte vis-à-vis de l'utilisateur. Son Abstraction correspond à la valeur représentée par l'objet. La partie Contrôle est chargée d'assurer la cohérence entre le côté abstrait et la présentation de l'objet.

Prenons comme exemple l'objet "camembert ". Sa Présentation est définie, d'une part par son aspect visuel et d'autre part par les manipulations que l'utilisateur peut effectuer sur cet objet. Son Abstraction est définie par une valeur comprise entre deux bornes. Enfin, le Contrôle d'un tel objet consiste à maintenir la cohérence entre la Présentation et l'Abstraction du camembert. Ainsi, lorsque l'utilisateur modifie interactivement la valeur présentée par le camembert, la partie Contrôle est chargée de modifier de la même manière la valeur connue par l'Abstraction de cet objet. Inversement, si les traitements de l'application modifient la valeur abstraite du camembert, la Présentation de cette valeur sous forme graphique doit être modifiée en conséquence.

La figure 8 illustre bien la découpe PAC du camembert.

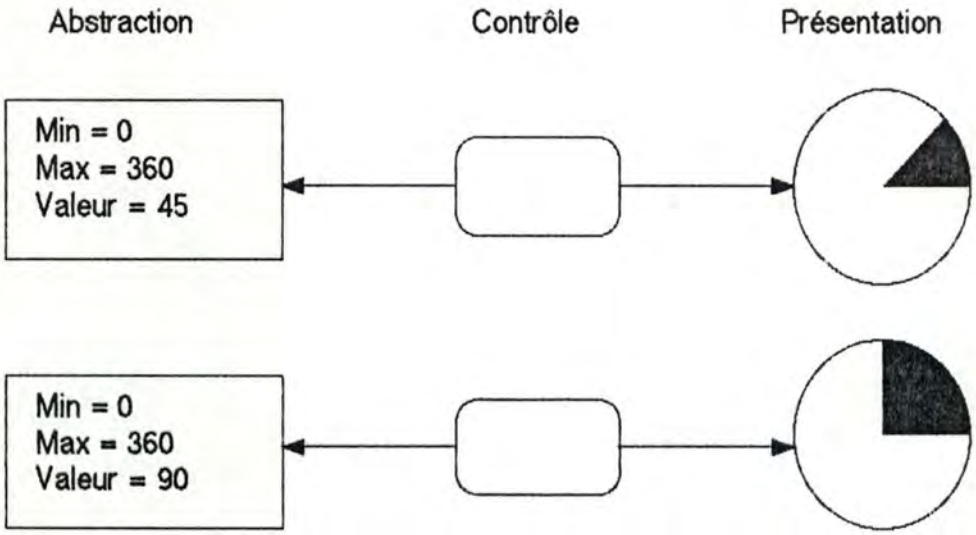


Figure 8 - Présentation, Abstraction et Contrôle de l'objet interactif "camembert ". Notons que les actions autorisées sur cet objet ont ici été négligées. Nous y reviendrons dans la suite de ce paragraphe.

Comme nous l'avons déjà fait pour les messages interactifs, nous pouvons distinguer les types d'objets interactifs des occurences de ces types d'objets.

Nous allons donner une description précise des types d'objets interactifs les plus courants. Le concepteur de dialogue pourra toujours définir un nouveau type si les objets standard qui lui sont proposés ne lui conviennent ou ne lui suffisent pas. Nous étudierons successivement la fenêtre, l'icône, le menu, le bouton de contrôle, le bouton-radio, le commutateur, la chaîne entière, la chaîne réelle, la chaîne de caractères, le booléen et le camembert. Pour cette étude, nous nous sommes inspirées des objets manipulés par le logiciel DIALOGUE de l'Apollo, de la description des ressources du toolbox Macintosh et de la définition du camembert donnée dans [Coutaz 86-3] et dans [Lermigeaux]. Nous aborderons ensuite les notions de hiérarchie d'objets et de "lien " entre ces objets. Nos sources d'information en ce domaine sont [Coutaz 86-3] et [Coutaz 87].

III.1.2.1 - La fenêtre

Le premier type d'objet interactif que nous détaillons est la *fenêtre*.

Une fenêtre est un cadre apparaissant à l'écran, qui peut contenir un certain nombre d'informations, et qui possède certaines propriétés. Nous avons choisi de décrire ici la fenêtre la plus couramment utilisée sur Macintosh. Une représentation graphique en est donnée à la figure 9 qui fait apparaître les différentes zônes d'une fenêtre.

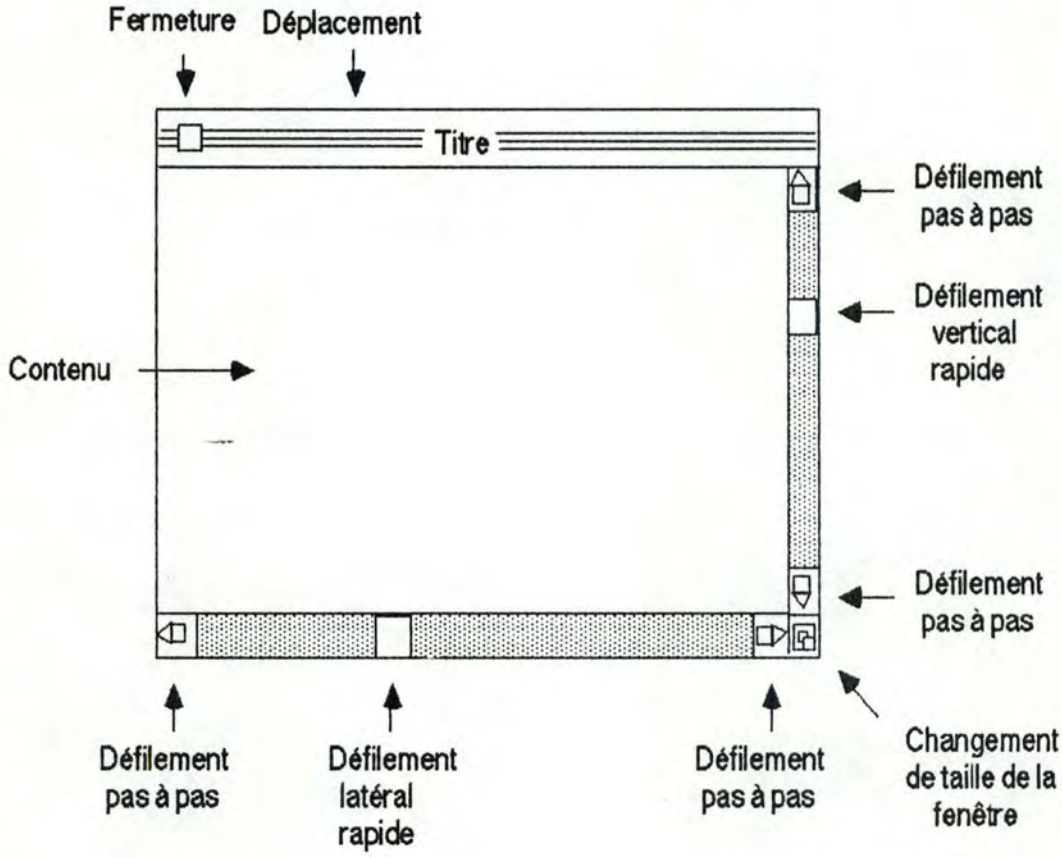


Figure 9 - Les différentes régions d'une fenêtre.

La zône principale est celle du contenu. C'est à cet endroit que seront présentées les informations contenues dans la fenêtre. Les autres zônes permettent de changer la taille de la fenêtre, de la déplacer, de la fermer, ou de la faire défiler (scrolling). Une dernière zône contient le titre de la fenêtre.

La manipulation des régions appartenant au contour de la fenêtre se fait à l'aide d'une souris. Par exemple, pour fermer la fenêtre, il suffit à l'utilisateur de positionner le suiveur de sa souris dans la zône correspondante et d'appuyer sur le bouton de celle-ci.

Les manipulations permises dans la partie "contenu " dépendent de ce qui est spécifié par les traitements fonctionnels de l'application qui manipule ces données. Les actions autorisées peuvent être, par exemple, la saisie ou la modification de certaines données que la fenêtre visualise à l'écran.

Tout ceci définit la Présentation de la fenêtre.

L'Abstraction de cet objet interactif est constituée de l'ensemble des données que cette fenêtre peut contenir. Lorsqu'une fenêtre présente à l'utilisateur un message interactif trop long pour apparaître entièrement à l'écran, la partie Abstraction de la fenêtre connait la totalité du message,tandis que la Présentation n'en connait que la part qui en est réellement visible. L'entièreté du message qui peut apparaître dans une fenêtre correspond au concept d'écran logique, concept fréquemment utilisé dans les ouvrages traitant de l'interface homme-machine [Coutaz, Herrmann], [OMEGA 86-1], [Waterkeyn].

Le Contrôle de la fenêtre consiste à mettre à jour la partie Abstraction en fonction des modifications que l'utilisateur a effectuées sur la Présentation des données contenues dans cette fenêtre, et inversement.

III.1.2.2 - L'icône

Le second type d'objet interactif que nous présentons est l'*icône* . L'icône est un rectangle dont les angles sont arrondis, dans lequel le concepteur de dialogue dessinera ce dont il a envie. Notons que le contour même du rectangle n'est pas nécessairement visible. La figure 10 donne la représentation graphique de deux icônes différentes. La première est l'icône de présentation d'un document constitué de texte, la seconde signale une erreur à l'utilisateur.

Figure 10 - Les icônes

L' utilisateur peut, à l'aide de la souris, positionner le suiveur sur l'icône et, à cet endroit, presser le bouton de la souris, pour déclencher une action associée à cet objet.

La partie Abstraction identifie le message désigné par l'icône et décrit l'action qui lui est rattachée. Cette action a pour effet de présenter à l'utilisateur le contenu du message qu'elle représente. Pour le signal "Attention ", l'action déclenchée est l'affichage d'un message d'erreur. Dans le cas du "Parchemin ", une pression sur la souris provoque l'apparition du texte que cette icône représente.

III.1.2.3 - Le menu

Le *menu* est un autre type d'objet interactif. Il permet de présenter à l'utilisateur toute une liste de choix.

La Présentation du menu est formée d'un titre et d'une série de champs désignant chacun une option, et décrivant la manière de sélectionner cette option . La figure 11 présente l'exemple d'un menu de sélection d'un objet interactif.

La partie Abstraction du menu contient, pour chaque option présentée, les actions à réaliser lors de la sélection de cette option par l'utilisateur.

Objet interactif	
fenêtre	F
icône	I
menu	M
entier	E
réel	R
caractère	C
booléen	B
camembert	K

Figure 11 - Menu de sélection d'un objet interactif.
Le texte à gauche présente les différentes options.
Les lettres à droite sont les caractères à taper pour effectuer la sélection.

III.1.2.4 - Le bouton de contrôle

Le *bouton de contrôle* est un objet dont la Présentation ressemble fortement à celle de l'icône. En effet, ce bouton est également un rectangle à coins arrondis. Cependant, le bouton contient exclusivement du texte, et non un dessin, et son rôle n'est pas de faire apparaître à l'écran le contenu d'un message. La manipulation du bouton de la souris dans la zône interne du bouton de contrôle provoque le déclenchement immédiat d'une action. L'Abstraction du bouton de contrôle décrit l'action qui y est rattachée. Cette action peut être, par exemple, la confirmation d'une information, ou la terminaison d'un traitement. Ces deux boutons de contrôle sont présentés à la figure 12.

Figure 12 - Deux exemples de bouton.
A gauche, un bouton de confirmation,
à droite, un bouton de sortie d'un traitement.

III.1.2.5 - Le bouton-radio

Le *bouton-radio* est un objet d'un type un peu différent des autres. En effet, les boutons-radio se rencontrent rarement seuls. Ils servent à présenter un choix parmi plusieurs valeurs. Le bouton-radio contient un libellé et un petit cercle qui est noir ou blanc, selon la valeur que l'on désire lui attribuer. La différence fondamentale vis-à-vis des autres objets est que, parmi un groupe de boutons, il en existe toujours un et un seul qui présente la valeur "VRAI " (noir). Les différents membres d'un groupe de boutons-radio ne sont donc pas indépendants les uns des autres

La manipulation permise sur les boutons-radio est la pression du bouton de la souris lorsque son suiveur est localisé dans le cercle du bouton. Appuyer sur la souris à l'intérieur de n'importe quel bouton provoque son passage à la valeur "VRAI ", et fait passer tous les autres boutons du groupe à la valeur "FAUX ".

L'Abstraction d'un ensemble de boutons-radio est la valeur ("VRAI " ou "FAUX ") de chacun des boutons; cette partie abstraite ne contient jamais qu'une seule valeur "VRAI".

La figure 13 fournit un exemple de boutons-radio permettant le choix d'une vitesse de transmission des informations sur une ligne-série.

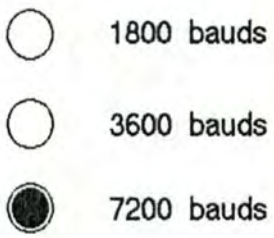


Figure 13 - Exemple de boutons-radio

III.1.2.6 - Le commutateur

Le type suivant d'objet interactif élémentaire est le *commutateur*. Comme le bouton-radio, il peut posséder deux valeurs : "VRAI " ou "FAUX ". Cependant, le commutateur apparaît comme une petite boîte accompagnée d'un titre. La boîte peut être soit remplie avec un "X " (valeur "VRAI "), soit vide (valeur "FAUX "). Le commutateur peut être utilisé pour effectuer un ou plusieurs choix parmi un certain nombre d'options. Une pression du bouton de la souris dans la boîte d'un commutateur fait changer sa valeur.

Les commutateurs de la figure 14 permettent de choisir des options déterminant le type d'un envoi reçu par une firme de vente par correspondance. Remarquons que, dans une même enveloppe, le client peut glisser un bon de commande et un formulaire de participation à un concours. Les options ne sont donc pas mutuellement exclusives, contrairement à ce qui se passe avec les boutons-radio.

☐ Ancien client

☒ Bon de commande

☒ Participation au concours

☐ Courrier divers

Figure 14 - Les commutateurs.

III.1.2.7 - Les chaînes entière et réelle, la chaîne de caractères et le booléen

La chaîne *entière* permet de présenter à l'utilisateur un nombre entier compris entre deux bornes. Sa Présentation est faite sous forme d'un ou plusieurs chiffres compris entre 0 et 9. Son Abstraction est la valeur représentée par ces chiffres.

La chaîne *réelle* permet de présenter de la même manière un nombre réel.

La *chaîne de caractères* contient un certain nombre de caractères alphabétiques, numériques, mathématiques,... Son Abstraction est constituée de la sémantique de cette chaîne de caractères.

Le *booléen* est un objet interactif qui ne peut prendre comme valeur que les chaînes de caractères "VRAI " ou "FAUX ". Un tel objet offre donc la possibilité de présenter des informations qui pourraient également être visualisés sous la forme de commutateurs.

III.1.2.8 - Le camembert

Décrivons maintenant un peu plus en détail l'objet *camembert* que nous avons présenté brièvement au début de ce paragraphe concernant les objets interactifs. Cet objet permet à l'utilisateur de choisir, sous forme graphique et de manière interactive, une valeur réelle comprise entre deux bornes. Cette valeur est déterminée à chaque instant par la position d'une aiguille mobile, manipulable au moyen d'une souris, par rapport à une aiguille fixe. Le camembert permet donc de représenter les mêmes objets que le champ réel, mais sous une forme différente. L'objet camembert a été décrit de manière très complète dans [Lermigeaux].

La représentation graphique donnée à la figure 15 laisse apparaître que le camembert est formé de six régions qui sont :

- le cadran,
- l'aiguille fixe,
- l'aiguille mobile,
- la part du cadran, comprise entre l'aiguille fixe et l'aiguille mobile, représentant le nombre (part interne),
- la part du cadran, comprise entre l'aiguille mobile et l'aiguille fixe, représentant le reste du cercle (part externe),
- l'axe central.

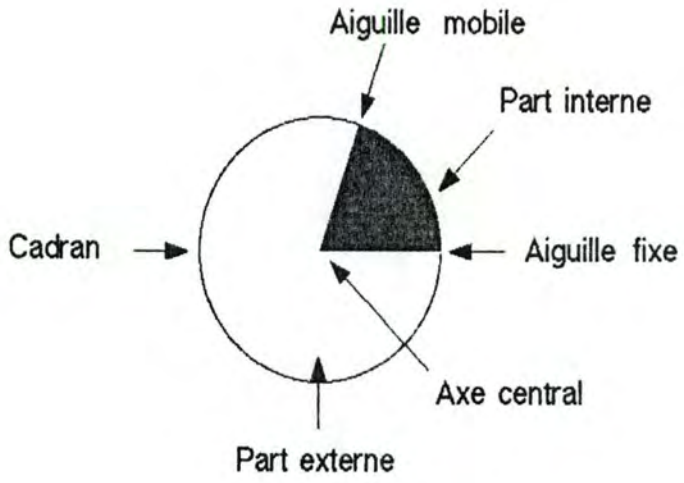


Figure 15 - Les six régions du camembert.

Le camembert est manipulé à l'aide d'une souris. Lorsque l'utilisateur désire obtenir une nouvelle valeur, il positionne la souris sur l'aiguille mobile, enfonce le bouton de celle-ci, et la déplace jusqu'à l'endroit désiré. Ensuite, il relâche le bouton de la souris. L'aiguille mobile suit la souris pendant que le bouton de cette dernière se trouve en position basse; elle se stabilise à l'endroit où l'utilisateur laisse ce bouton revenir en position haute.

Pour pouvoir distinguer facilement les deux parts de fromage, la part interne du camembert est coloriée à l'aide d'un motif, tandis que la part externe prend une autre couleur.

Nous avons, jusqu'à présent, parlé exclusivement de la partie Présentation du camembert.

Quant à son Abstraction, elle est définie par les valeurs des bornes minimale et maximale , ainsi que, par la valeur courante.

Enfin, le composant Contrôle du camembert est chargé, pendant la manipulation effectuée par l'utilisateur, de calculer, à partir de la position de l'aiguille mobile, la valeur réelle correspondant à cette position.

III.1.2.9 - L'objet "groupe"

Cet objet est un rectangle, visible ou non au poste de travail, qui permet au concepteur de dialogue de décrire des groupes d'objets interactifs. Par exemple, lors de la présentation à l'écran d'un bon de commande, il serait intéressant de pouvoir grouper d'une part les informations concernant le client et d'autre part les données se rapportant aux articles commandés par celui-ci.

Nous remarquons ici qu'il peut exister une hiérarchie d'objets interactifs. En effet, une fenêtre peut contenir des objets ou des groupes d'objets contenant eux-même d'autres objets ou groupes d'objets...

A l'intérieur d'un objet interactif de type "groupe ", on trouve plusieurs objets interactifs. Ceux-ci peuvent être des objets séparés ou des objets "liés ". Deux objets interactifs sont "liés " lorsque leurs valeurs dépendent l'une de l'autre.

Prenons comme exemple l'objet groupe de la figure 16 constitué de deux objets interactifs différents : un camembert et une chaîne numérique réelle. La chaîne indique à tout instant la valeur représentée par le camembert. Lorsque la valeur graphique du camembert est modifiée par l'utilisateur, la valeur de la chaîne numérique doit être mise à jour de la même manière. L'Abstraction du camembert et celle de la chaîne numérique doivent toujours avoir la même valeur. Pour cela, leurs parties Contrôle doivent collaborer entre elles. Pour plus de détails à ce sujet, le lecteur peut consulter [Coutaz 86-3].

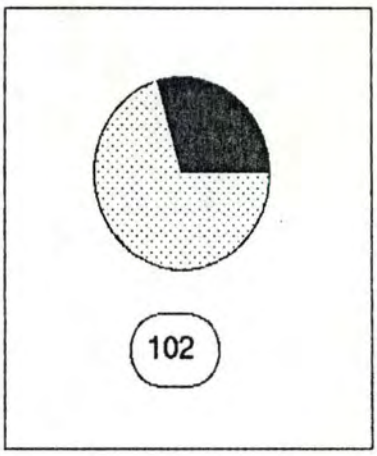


Figure 16 - L'objet "groupe " contenant un camembert et une chaîne numérique.

III.1.2.10 - Les objets définis par le concepteur du dialogue

Nous avons également déjà signalé que, si le concepteur ne trouve pas d'objet ou de groupe d'objets interactifs qui lui convienne, il peut toujours en définir d'autres lui-même. Pour décrire ces types d'objets, il est évident qu'une spécification graphique serait nettement plus aisée qu'une spécification textuelle. Il suffirait alors au concepteur de dialogue de dessiner le nouvel objet qu'il désire créer, de lui donner un nom, et de déterminer ses propriétés.

Lors de notre description d'un langage de spécification des objets interactifs, nous ne tiendrons compte que des types d'objets que nous avons définis jusqu'ici.

III.2 - Les fonctions du dialogue

III.2.1 - Saisie et affichage

Les principales fonctions du dialogue sont la *saisie* et l'*affichage* d'informations au poste de travail de l'utilisateur. Ces deux fonctions fondamentales sont à la base de tout dialogue homme-machine.

La fonction de *saisie* à l'écran effectue une validation syntaxique sur les données introduites par l'utilisateur; si celles-ci ne sont pas valides, cette fonction produit un message d'erreur et oblige l'utilisateur à effectuer les corrections nécessaires.

La saisie peut se faire avec ou sans mécanisme de "confirmation / infirmation ".

Dans le cas de la saisie avec confirmation, la fonction de saisie considérera toutes les données déjà introduites comme inexistantes, jusqu'à ce qu'elles aient été confirmées. Dès lors, si l'utilisateur infirme les données qu'il vient d'introduire, le système se comportera exactement comme si ces informations n'avaient jamais existé.

Dans le cas de la saisie sans "confirmation / infirmation ", le message sur lequel l'utilisateur travaille est considéré comme complet dès que ce dernier a introduit tous les champs qui le composent. Le problème consiste alors à décider du moment à partir duquel un message est considéré comme complet : est-ce dès que l'utilisateur a terminé d'introduire une valeur pour le dernier champ du message, ou est-ce au moment où l'utilisateur passe à un autre message, indiquant par là qu'il considère le premier message comme clôturé ?

Le choix entre ces différentes formes de saisie est important car il oriente le style de dialogue.

Dans le cas de la saisie avec confirmation, l'utilisateur décide toujours de l'enregistrement de ce qu'il a introduit en le confirmant. Dans le cas de la saisie sans confirmation où un message est considéré comme complet lorsque l'utilisateur passe au message suivant, ce dernier décide de l'enregistrement de ce qu'il a introduit en effectuant une confirmation implicite. Le passage au message interactif suivant peut, en effet, être considéré comme une confirmation implicite. Par contre, dans le cas où le message est complet dès que l'utilisateur a introduit une valeur pour chacun des champs du message, celui-ci est en quelque sorte "traqué " en vue de détecter s'il a ou non terminé l'introduction des données du message. Dans ce cas, si l'utilisateur s'aperçoit qu'il a fait une erreur en introduisant le dernier champ du message, il est trop tard pour effectuer la correction. Deux solutions sont alors possibles : soit l'application fonctionnelle détectera l'erreur, la signalera à l'utilisateur et la lui fera corriger, soit l'utilisateur devra lui-même annuler le traitement en cours et le recommencer en lui fournissant de nouvelles valeurs.

La fonction d'*affichage* a un rôle beaucoup plus simple : cette fonction fait paraître à l'écran tous les messages de sortie de l'application fonctionnelle destinés au poste de travail.

L'*impression* est également une fonction du dialogue, qui permet de produire des informations de sortie sur un support plus durable que l'écran : le papier.

De manière similaire, les informations de sortie peuvent être encore dirigées vers d'autres supports que le papier (bande magnétique,...).

III.2.2 - Aide

Le dialogue a aussi comme fonction de fournir de l'*aide* à l'utilisateur du poste de travail. Nous avons déjà signalé que l'aide peut porter soit sur le dialogue lui-même, soit sur les fonctionnalités de l'application, soit sur le contexte d'exécution de l'application.

III.2.3 - Déclenchement d'un traitement

Le dialogue possède également deux fonctions qui permettent de déclencher des traitements.

La première d'entre elles est l'*initialisation de l'application interactive*. Cette initialisation permet de déclencher les traitements de l'application et d'initialiser le dialogue. La partie "Dialogue " de l'application doit, en effet, fournir à l'utilisateur le premier écran de cette application; elle pourrait aussi permettre à l'utilisateur de s'identifier. Ensuite, sur base de cette identification, elle pourrait déterminer le type de personne avec qui elle converse et choisir le dialogue le plus approprié à cette personne. Pour cela, il faudrait cependant que le concepteur de l'application interactive ait spécifié plusieurs types de dialogue.

La seconde fonction de déclenchement des traitements est la fonction qui réalise l'*activation d'une phase* de l'application. L'utilisateur demandera le déclenchement d'une phase à partir de son terminal. C'est donc la partie "Dialogue " de l'application qui sera chargée de transmettre cette demande vers les traitements.

Remarquons que, sur un poste de travail mono-tâche, l'activation d'une phase implique automatiquement que toutes les autres phases en cours d'exécution s'interrompent. Par contre, si l'utilisateur dispose d'un poste de travail multi-tâches, celui-ci pourra faire exécuter plusieurs phases en parallèle et passer de l'une à l'autre.

III.2.4 - Interruption et reprise

Le dialogue doit pouvoir être interrompu à tout instant. Il doit donc fournir une fonction d'*interruption* des traitements. En parallèle à la fonction d'interruption, le dialogue doit également posséder une fonction de *reprise* de la conversation. A nouveau, l'aide concernant le contexte de l'application peut être très utile à la personne qui effectue la reprise d'une activité qu'il avait abandonnée. Notons que l'interruption et la reprise de tâches seront communiquées aux traitements de l'application.

En effet, lors de l'interruption d'un traitement T1, ce dernier doit libérer les ressources qu'il utilisait pour que d'autres traitements puissent y avoir accès. La reprise de T1 doit donc effectuer une nouvelle réquisition des ressources nécessaires à sa terminaison. Nous soulevons ici un problème important : à la reprise de T1, celui-ci devrait pouvoir continuer à se dérouler comme s'il n'avait pas été interrompu. Cependant, d'autres traitements peuvent avoir entretemps modifié les données sur lesquelles T1 travaillait. Que faire, dans ce cas, pour que T1 fournisse des résultats cohérents ? Le traitement T1 devrait-il être réexécuté dans sa totalité ? Cette solution ne serait pas admise par l'utilisateur. Etant donné la portée très étendue de la question, nous n'aborderons pas ce problème de l'interruption d'un traitement dans le cadre de ce mémoire.

Rappelons enfin que, dans le modèle de la structuration des traitements présenté au chapitre 2, la fonction a été définie comme un traitement ininterruptible. L'interruption et la reprise ne peuvent donc porter que sur une phase.

III.2.5 - Annulation

L'utilisateur peut, pour une raison quelconque, *désirer annuler une phase en* cours d'exécution. Il verra alors apparaître à l'écran de son poste de travail exactement ce qu'il avait sous les yeux avant de déclencher cette phase. La demande d'annulation, saisie par le composant "Dialogue " de l'application, sera communiquée aux traitements afin de stopper leur exécution, et d' "oublier " les résultats des traitements déjà effectués depuis le lancement de la phase.

III.2.6 - Retour-arrière

L'utilisateur doit pouvoir, à tout instant, effectuer un retour-arrière à l'endroit qu'il désire dans le déroulement de la conversation homme-ordinateur d'une phase, et ainsi annuler les traitements effectués et les messages introduits. La fonction suivante est donc le *retour-arrière*. Comme dans le cas de la fonction d'annulation d'une phase, l'annulation des traitements sera communiquée à l'application fonctionnelle pour que celle-ci effectue un retour-arrière à l'endroit désiré par l'utilisateur. Le retour-arrière s'effectue à l'intérieur d'une phase, alors que l'annulation porte sur l'entièreté de la phase.

L'aide fournie en ce qui concerne le contexte d'exécution de l'application et les fonctionnalités de celle-ci pourra éclairer l'utilisateur à propos de l'endroit exact auquel il désire retourner.

III.2.7 - Gestion du contexte

A tout moment, le dialogue doit pouvoir renseigner l'utilisateur à propos des données courantes (Site) et des commandes courantes (Mode) de l'application et lui fournir l'historique du déroulement d'une phase (Trail). Ceci permet de répondre à toutes les questions que l'utilisateur peut de poser à propos du contexte d'exécution de l'application.

Comme nous l'avons déjà dit au paragraphe I.2.1, cette classification a été effectuée par J. Nievergelt dans [Nievergelt]

Après avoir décrit les objets et les fonctions du dialogue, nous allons maintenant décrire les deux modèles du dialogue que sont la Présentation et la Conversation.

III.3 - La Présentation

Le modèle de la Présentation comprend les objets du dialogue présentés précédemment ainsi qu'un langage de spécification décrivant de manière complète les objets du dialogue qui apparaîtront au poste de travail.

Nous avons présenté en détail les objets interactifs ainsi que les messages interactifs. Les messages interactifs décrivent la sémantique des informations à saisir ou à afficher à l'écran, dont ils fournissent un regroupement ergonomique. Quant aux objets interactifs, ils serviront à présenter ces informations de manière plaisante à l'utilisateur. Les messages interactifs d'une application et les objets interactifs qui les visualisent dépendent donc fortement du ou des types d'utilisateurs auxquels l'interface est destinée. Si l'application est destinée à différentes classes d'utilisateurs, le concepteur du dialogue peut juger nécessaire de décrire plusieurs versions de l'interface (messages et objets interactifs), dont chacune est adaptée à un type particulier de personnes.

Le concepteur de dialogue doit donc décrire de manière détaillée les messages interactifs ainsi que les objets interactifs.

Ce que nous présentons dans ce paragraphe est une ébauche de langage de spécification des messages et objets interactifs. Nous ne prétendons pas avoir dressé une liste complète de toutes leurs caractéristiques. Cette remarque est principalement vraie pour les objets interactifs. Chaque type d'objet possède en effet des caractéristiques bien précises que nous n'avons pas relevées ici.

III.3.1 - Spécification d'un message interactif

La spécification de toute occurence de message interactif doit préciser :

- le nom du message,
- le type dont il est une occurence (message fonctionnel d'entrée, message fonctionnel de sortie "pur", message fonctionnel d'erreur, message d'aide, message de contrôle, message de donnée du dialogue, message exclusivement interactif d'erreur),

- la sémantique générale du contenu du message,
- une description de chacun de ses champs,
- une information qui précise s'il s'agit d'un message de saisie et/ou d'affichage,
- un attribut de présentation qui détermine le ou les objets interactifs qui permettront de visualiser ce message à l'écran (fenêtre, icône,...)

Pour préciser l'avant-dernière information fournie par la description d'un message interactif, signalons qu'un message peut être à la fois message d'affichage et de saisie, par exemple lorsqu'il présente à l'utilisateur une valeur provenant de l'application fonctionnelle (affichage) en lui demandant de la modifier si cette valeur ne lui convient pas (saisie).

En ce qui concerne la sémantique des messages fonctionnels d'entrée ou de sortie, le concepteur du dialogue trouvera une description complète de tous leurs champs dans la spécification des messages-externes-fonction du modèle de la statique des traitements fourni par le concepteur de l'application fonctionnelle. En effet, les messages fonctionnels du dialogue possèdent la même sémantique que les messages-externes-fonction des traitements. La seule différence est que le regroupement des informations entre les messages peut être différent.

Un champ d'un message peut être :
- simple ou répétitif,
- obligatoire ou facultatif,
- accepter éventuellement une valeur par défaut.

Reprenons l'exemple du camembert et supposons qu'il serve à faire introduire à l'utilisateur un nombre réel compris entre 0 et 360 pour permettre à l'application fonctionnelle d'effectuer un traitement quelconque sur ce nombre. La spécification d'un tel message interactif est la suivante :

Nom-message : Mess-camembert

Description : Le message est utilisé pour faire introduire par l'utilisateur du poste de travail un nombre compris entre 0 et 360.

Type : message d'entrée

Nom-Champ(s) : nb
Saisie / affichage : saisie
Objet de présentation : fenêtre-cam

"nb " est le nom d'un champ réel qui pourra être décrit d'une manière semblable.

Nom-Champ : nb
Description : réel compris entre 0 et 360
Caractéristiques : champ simple et obligatoire
Valeur Maximale : 360
Valeur Minimale : 0
Valeur par défaut : 45
Saisie / affichage : saisie
Présentation : cam

Les objets interactifs "fenêtre-cam " et "cam " serviront à présenter ce message qui contient un champ entier. Ces objets seront décrits à l'aide de la méthode de spécification exposée ci-après.

III.3.2 - Spécification d'un objet interactif

Les objets interactifs permettent au concepteur du dialogue de décrire en détail ce que l'utilisateur verra apparaître à l'écran.

La spécification de toute occurence d'objet interactif doit préciser :

- le nom de l'objet,
- de quel type d'objet il est une occurence (fenêtre, icône, menu, bouton de contrôle, bouton-radio, commutateur, chaîne entière, chaîne réelle, booléen, chaîne de caractères, camembert, groupe),
- la position de l'objet par rapport à l'écran (ou par rapport à l'objet à l'intérieur duquel il se trouve, dans le cas d'objets hiérarchisés),

- la liste des objets qui dépendent de lui par un lien de filiation directe dans la hiérarchie,
- la taille de l'objet,
- un attribut indiquant si l'objet devra ou non être confirmé lors de sa saisie à l'écran,
- un message d'aide concernant cet objet (si le concepteur le juge utile),
- des propriétés plus particulières au type d'objet dont il fait partie,
- les évènements acceptés par l'objet.

Pour terminer de spécifier le modèle de la Présentation de l'exemple du camembert, il nous reste donc à décrire les objets "fenêtre-cam " et "cam ".

Nom-objet	: fenêtre-cam
Type	: fenêtre
Position	: (100, 100)
Nom-objets-fils	: cam
Taille	: (200, 250)
Evénements	: Seuls les événements souris sont acceptés. Ce sont les événements standard définis pour le type d'objets "fenêtre " au paragraphe III.1.2.1.
Epaisseur des contours	: 1
Couleur des contours	: noir
Couleur de fond	: blanc
Confirmation ?	: oui
Aide	: ——

Nom-objet	: cam
Type	: camembert
Position	: (50, 50)
Nom-objets-fils	: ——
Taille	: 20
Evénements	: Evénement souris pour déplacer l'aiguille mobile.
Epaisseur du cadran	: 1
Couleur du cadran	: gris
Position de l'aiguille fixe	: horizontale (0°)
Largeur de l'aiguille fixe	: 1
Couleur de l'aiguille fixe	: gris
Largeur de l'aiguille mobile	: 1
Couleur de l'aiguille mobile	: gris
Valeur par défaut de l'aiguille mobile	: 45
Couleur de la part interne	: noir
Couleur de la part externe	: blanc
Confirmation ?	: non
Aide	: ——

Remarquons que l'attribut "position " est exprimé en nombre de pixels. Le coin supérieur gauche de la fenêtre "fenêtre-cam " sera situé à la position (100, 100) à partir du coin supérieur gauche de l'écran, tandis que le centre de l'objet "cam " sera situé à la position (50, 50) à partir du coin supérieur gauche de la fenêtre "fenêtre-cam " dans laquelle il se trouve.

De la même manière, la fenêtre aura une largeur de 200 pixels et une hauteur de 250, tandis que le camembert aura un rayon de 20 pixels. Notons encore que l'utilisateur pourra changer interactivement la taille de la fenêtre; une zône de celle-ci est prévue à cet effet (cfr figure 9, paragraphe III.1.2.1).

Une spécification du même style que celle décrite dans ce paragraphe pour les objets et les messages interactifs de l'exemple "camembert " peut être fournie pour tout type d'objet du dialogue. Nous ne détaillerons cependant pas cette spécification.

III.3.3 - Spécification de la Présentation de l'exemple " mutuelle "

La Présentation de la phase d'enregistrement d'une prestation doit tenir compte du ou des types d'utilisateurs auxquels ce traitement est destiné. Nous décrirons ici deux spécifications possibles de la Présentation de cette phase.

III.3.3.1 - Un seul message interactif de saisie

Dans cette première solution, les données qui doivent être encodées par l'utilisateur forment un seul message interactif. Toutes ces données sont donc saisies en même temps.

Décrivons donc les différents messages interactifs qui sont utilisés dans cette hypothèse pour effectuer la saisie et l'affichage des données.

Nom-message : Mess-feuille-de-soins

Description : Le message est utilisé pour faire introduire à l'utilisateur du poste de travail une feuille de soins qui concerne un individu affilié à la mutuelle

Type : message d'entrée

Nom-Champ(s) :
- nom-ass
- prénom-ass
- date-naissance-ass
- num-matricule
- date-prestation- soins
- 1 à N lignes constituées de :
  - code-acte
  - montant-dr-acte
  - quantité-acte

Saisie / affichage : saisie

Objet de présentation : fenêtre-feuille-de-soins

Nom-Champ : nom-ass
Description : chaîne de caractères représentant un nom de personne
Caractéristiques : champ simple et obligatoire
Longueur maximale : 30 caractères
Saisie / affichage : saisie
Présentation : nom

Nom-Champ : prénom-ass
Description : chaîne de caractères représentant un prénom
Caractéristiques : champ simple et obligatoire
Longueur maximale : 30 caractères
Saisie / affichage : saisie
Présentation : prénom

Nom-Champ : date-naissance-ass
Description : date de naissance de l'individu
Caractéristiques : champ simple et obligatoire
Saisie / affichage : saisie
Présentation : date-nais

Nom-Champ : num-matricule
Description : numéro d'identification de l'individu assuré
Caractéristiques : champ simple et obligatoire
Saisie / affichage : saisie
Présentation : matricule

Nom-Champ : date-prestation-soins
Description : date à laquelle l'individu a subi la prestation
Caractéristiques : champ simple et obligatoire
Saisie / affichage : saisie
Présentation : date-prest

Nom-Champ : code-acte
Description : code contenant une lettre et un nombre, et permettant d'identifier un acte médical
Caractéristiques : champ répétitif et obligatoire
Saisie / affichage : saisie
Présentation : code

Nom-Champ : montant-dr-acte
Description : montant droit de l'acte médical
Caractéristiques : champ répétitif et obligatoire
Saisie / affichage : saisie
Présentation : montant-dr

Nom-Champ : quantité-acte
Description : nombre de fois que l'acte médical a été presté
Caractéristiques : champ répétitif et obligatoire
Saisie / affichage : saisie
Présentation : quantité

Nom-message : Mess-erreur-individu
Description : Le message signale que l'individu introduit n'est pas affilié à la mutuelle et indique à l'utilisateur qu'il doit corriger la saisie des références d'un individu
Type : message fonctionnel d'erreur
Nom-Champ(s) : champ-erreur-individu
Saisie / affichage : affichage
Objet de présentation : fenêtre-erreur-individu

Nom-Champ : champ-erreur-individu

Description : texte décrivant les raisons de l'erreur et l'endroit où il faut effectuer le retour-arrière afin de faire les corrections

Caractéristiques : champ simple et obligatoire

Saisie / affichage : affichage

Présentation : erreur-indiv

Nom-message : Mess-erreur-actes

Description : Le message signale une erreur dans un ou plusieurs code-acte et indique à l'utilisateur qu'il doit corriger la saisie des actes médicaux de la prestation

Type : message fonctionnel d'erreur

Nom-Champ(s) : champ-erreur-actes

Saisie / affichage : affichage

Objet de présentation : fenêtre-erreur-actes

Nom-Champ : champ-erreur-actes

Description : texte décrivant les raisons de l'erreur et l'endroit où il faut effectuer le retour-arrière afin de faire les corrections

Caractéristiques : champ simple et obligatoire

Saisie / affichage : affichage

Présentation : erreur-actes

Nom-message : Mess-récapitulatif

Description : Le message est utilisé pour produire un récapitulatif des informations introduites

Type : message de résultat "pur"

Nom-Champ(s) :
- nom-ass
- prénom-ass
- date-naissance-ass
- num-matricule
- date-prestation- soins

- 1 à N lignes constituées de : - code-acte
- quantité-acte
- montant-dr-acte
- montant-br-acte
- montant-ao-acte
- montant-tm-acte

Saisie / affichage : affichage
Objet de présentation : fenêtre-récapitulatif

Excepté les trois derniers champs, ce message contient les mêmes informations que le message mess-feuille-de-soins. La seule différence est que, dans ce cas-ci, ce sont des champs d'affichage et non de saisie. Nous ne les décrirons donc pas une seconde fois. La description des trois derniers champs est la suivante :

Nom-Champ : montant-br-acte
Description : montant de la base de remboursement de l'acte médical
Caractéristiques : champ répétitif et obligatoire
Saisie / affichage : affichage
Présentation : montant-br

Nom-Champ : montant-ao-acte
Description : montant de l'assurance obligatoire
Caractéristiques : champ répétitif et obligatoire
Saisie / affichage : affichage
Présentation : montant-ao

Nom-Champ : montant-tm-acte
Description : montant du ticket modérateur
Caractéristiques : champ répétitif et obligatoire
Saisie / affichage : affichage
Présentation : montant-tm

Spécifions maintenant quelques objets interactifs.

Nom-objet : fenêtre-feuille-de-soins
Type : fenêtre
Position : (20, 20)
Nom-objets-fils : - groupe-individu
- date-prest
- groupe-actes
Taille : (300, 250)
Evénements : Seuls les événements souris sont acceptés.
Ce sont les événements standard définis pour le type d'objets "fenêtre " au paragraphe III.1.2.1
Epaisseur des contours : 1
Couleur des contours : noir
Couleur de fond : blanc
Confirmation ? : non
Aide : Cette fenêtre permet d'introduire tous les renseignements nécessaires à l'enregistrement d'une prestation médicale

Nom-objet : groupe-individu
Type : groupe simple
Position : (25, 25)
Nom-objets-fils : - nom-ass
- prénom
- date-nais
- num-matricule
Taille : (200,120)
Evénements : Aucun
Epaisseur des contours : 1
Couleur des contours : gris
Couleur de fond : blanc
Aide : ——

Nom-objet : nom

Type : chaîne de caractères

Position : (35, 35)

Nom-objets-fils : ——

Taille maximale : 30 caractères

Evénements : - clic souris pour le positionnement du curseur;
- frappe du clavier pour l'introduction du nom.

Epaisseur du cadre : 1

Couleur du cadre : blanc

Aide : ——

Les objets interactifs "prénom ", "date-nais " et "matricule " ainsi que la date "date-prest " peuvent être décrits de la même manière.

Nom-objet : groupe-actes

Type : groupe répétitif

Position : (25, 150)

Nom-objets-fils : - code
- montant-dr
- quantité
- stop
- continue

Taille : (200,100)

Evénements : Aucun

Epaisseur des contours : 1

Couleur des contours : gris

Couleur de fond : blanc

Aide : ——

Les objets "code ", "montant " et "quantité " sont du même type que l'objet "nom " spécifié ci-dessus. Nous n'en donnerons donc pas la description.

Par contre, les objets "stop " et "continue " sont très différents. En effet, ils n'apparaissent nulle part dans la spécification des messages interactifs. Ce sont des objets qui permettent de faciliter la tâche de l'utilisateur, mais qui n'ont aucune sémantique pour l'application. Ce sont des boutons de contrôle dont le rôle est de permettre à l'utilisateur du poste de travail de déterminer le nombre de répétitions à effectuer sur ce groupe, c'est-à-dire le nombre de lignes de prestation à saisir. Nous décrirons par exemple l'objet "stop " de la façon suivante :

Nom-objet	: stop
Type	: bouton de contrôle
Position	: (180, 90)
Nom-objets-fils	: ——
Taille	: (20, 10)
Texte	: STOP
Evénements	: - clic souris pour l'activation du bouton. La pression de la souris dans la zône où se trouve le bouton déclenche la terminaison de la répétition du groupe "groupe- actes "
Epaisseur du cadre	: 1
Couleur du cadre	: gris
Couleur du texte	: noir
Couleur du fond	: blanc
Aide	: ——

III.3.3.2 - Trois messages interactifs de saisie

Dans cette seconde solution, les données qui devront être encodées par l'utilisateur sont réparties à l'intérieur de trois messages, le premier concernant l'individu assuré, le deuxième contenant la date à laquelle la prestation a été effectuée et le troisième se rapportant à une ligne de prestation.

Les messages d'erreur et le récapitulatif, quant à eux, ne changent pas.

Décrivons donc les trois messages interactifs de saisie.

Nom-message : Mess-info-individu

Description : Le message est utilisé pour faire introduire à l'utilisateur du poste de travail des informations qui concernent un individu affilié à la mutuelle

Type : message d'entrée

Nom-Champ(s) : - nom-ass
- prénom-ass
- date-naissance-ass
- num-matricule

Saisie / affichage : saisie

Objet de présentation : fenêtre-info-individu

Nom-message : Mess-date-prest

Description : Le message est utilisé pour faire introduire à l'utilisateur du poste de travail la date à laquelle la prestation a été réalisée

Type : message d'entrée

Nom-Champ(s) : date de prestation de soins

Saisie / affichage : saisie

Objet de présentation : fenêtre-date

Nom-message : Mess-ligne-de-prestation-soin

Description : Le message est utilisé pour faire introduire à l'utilisateur du poste de travail une ligne de soin concernant un individu

Type : message d'entrée

Nom-Champ(s) : - code-acte
- montant-dr-acte
- quantité-acte

Saisie / affichage : saisie

Objet de présentation : fenêtre-ligne-prestation-soin

Tous les champs de ces messages sont décrits dans le paragraphe précédent. La seule différence est que, dans le cas du message mess-ligne-prestation-soin, tous les champs sont devenus simples et non plus répétitifs, puisque ce message ne contient qu'une seule ligne de soins.

Après avoir décrit cette série de messages interactifs, nous devons encore spécifier tous les objets interactifs qui les présenteront à l'utilisateur. Nous n'effectuerons cependant pas ce travail ici; en effet, les messages interactifs décrits dans le paragraphe précédent donnent une idée suffisamment précise de ce qui doit être fait.

Nous avons décrit brièvement la présentation du dialogue au poste de travail; nous allons maintenant en spécifier la conversation.

III.4 - La conversation

III.4.1 - Remarques préliminaires

Le modèle de la Conversation a pour but de fournir au concepteur de dialogue des concepts et des mécanismes lui permettant de représenter l'enchaînement des messages interactifs appartenant à une phase qui seront visualisés au poste de travail de l'utilisateur.

Le concepteur du dialogue peut juger utile de connaître l'enchaînement des traitements pour réaliser le scénario d'enchaînement des messages interactifs. Il pourra ainsi savoir dans quel ordre l'application fonctionnelle utilise les différents messages externes qui lui sont fournis par le dialogue, et déterminer un ordre similaire de saisie des messages interactifs, s'il l'estime nécessaire.

Rappelons que le concepteur de dialogue peut fournir plusieurs spécifications du modèle de la Présentation, si l'application interactive est destinée à différents types d'utilisateur. De même, plusieurs schémas de la Conversation peuvent être réalisés pour une même application. Ces schémas peuvent fournir soit différentes possibilités d'enchaînement des mêmes messages interactifs soit la description de l'enchaînement de messages différents, issus de plusieurs spécifications de la Présentation.

Enfin, il nous reste à préciser une remarque importante. Les messages de sortie peuvent ne pas faire partie de la description du scénario d'enchaînement des messages interactifs.

En effet, dans la plupart des cas, il est préférable que les messages fonctionnels de sortie, qu'ils soient des résultats "purs " ou des messages d'erreur, n'apparaissent à l'écran qu'à un moment où ils ne distraient pas l'utilisateur de la tâche qu'il effectue.

La meilleure méthode permettant d'obtenir ce résultat est de signaler à l'utilisateur qu'un message en provenance de l'application fonctionnelle est disponible, et qu'à partir de cet instant, il peut y avoir accès quand il le désire. Un moyen de signaler l'arrivée d'un tel message serait de faire paraître une icône à un endroit de l'écran réservé à cet effet. En "ouvrant " l'icône, l'utilisateur pourrait alors accéder au contenu du message au moment où il le souhaite.

Cependant, le concepteur de dialogue peut, dans certains cas, désirer introduire des messages fonctionnels de sortie dans l'enchaînement du dialogue afin d'obliger l'utilisateur à prendre connaissance de ces messages avant d'aller plus loin dans le déroulement de l'application.

De façon similaire, les messages de sortie exclusivement interactifs ne font pas partie du scénario de la conversation. L'utilisateur peut accéder aux messages d'aide quand il le désire, et non quand le concepteur de dialogue le décide. Les messages exclusivement interactifs d'erreur signalent les erreurs syntaxiques. Ils se rattachent donc à un champ d'un message interactif et ils s'enchaînent de manière standard à la saisie du champ auquel ils se rapportent, puisqu'ils apparaissent dès qu'un champ erroné est introduit ou confirmé. Le concepteur du dialogue doit être conscient de cet enchaînement standard des messages, mais il ne devra pas le décrire lors de la spécification de la Conversation de l'application interactive.

Après ces remarques préliminaires, passons maintenant à la description des différents états possibles d'un message interactif et aux divers types de structures d'enchaînement de ces messages.

III.4.2 - Les états d'un message interactif

La progression de la saisie ou de l'affichage au terminal d'un message interactif peut être décrite par certains changements d'état.

Les trois principaux changements d'état caractérisant le cycle de vie d'un message interactif de saisie sont :

- le déclenchement de saisie, qui correspond à l'autorisation pour l'utilisateur d'effectuer la saisie de ce message, s'il le désire;
- l'activation de la saisie qui est la mise en route réelle de cette dernière;
- la fin de saisie qui survient lorsque toutes les données du message ont été saisies (et confirmées, si la saisie de ce message demande une confirmation). Le message n'est alors plus accessible à l'utilisateur.

Les trois principaux changements d'état caractérisant le cycle de vie d'un message interactif d'affichage sont :

- le déclenchement de l'affichage, qui correspond à l'autorisation pour l'utilisateur d'accéder, s'il le désire, au contenu de ce message;
- l'activation de l'affichage qui est l'accès réel au message;
- la fin d'affichage qui survient lorsque l'utilisateur a terminé la lecture du message et qu'il fait disparaître celui-ci de l'écran.

III.4.3 - Structures d'enchaînement

Comme nous l'avons déjà fait pour la dynamique des traitements, nous allons envisager ici toutes les structures qui peuvent se rencontrer dans un scénario d'enchaînement des messages interactifs. Les formes d'enchaînement que nous décrivons ici ont été dérivées des structures décrites au paragraphe II.4.2

Ces structures sont les suivantes :

- le parallélisme,
- la séquence,
- l'enchaînement éclaté,
- l'enchaînement convergent,
- la synchronisation,
- l'itération,
- le choix,
- le retour-arrière.

Les principaux enchaînements seront schématisés pour clarifier la tâche du concepteur de dialogue.

Un utilisateur actif pourrait être invité à lire les schémas d'enchaînement du dialogue et des traitements de l'application; c'est pourquoi nous avons utilisé un formalisme semblable pour la description de ces deux schémas.

Remarquons enfin que, dans le schéma étudié dans ce paragraphe, nous ne faisons intervenir que les messages. Ce choix de ne pas représenter les traitements à réaliser sur ces messages possède deux justifications. La première est que, à l'opposé des fonctions présentes dans le schéma de la dynamique des traitements, les fonctions réalisées sur les messages interactifs sont des opérations standard (saisie, affichage). La seconde raison est que le fait d'introduire des traitements dans ce schéma n'apporterait aucune lumière supplémentaire sur celui-ci.

Les traitements effectués sur les messages présents dans le scénario du dialogue sont donc "cachés " par les messages auxquels ils se rapportent. A un message de saisie correspond de manière implicite une opération de saisie; à un message d'affichage correspond de la même manière une opération d'affichage.

III.4.3.1 - Le parallélisme

Un enchaînement de deux ou plusieurs messages interactifs est dit *parallèle* lorsque le concepteur du dialogue a jugé bon de laisser l'utilisateur libre de choisir leur ordre d'apparition au poste de travail.

Le concepteur ne doit pas spécifier un tel enchaînement puisque, par définition, le parallélisme laisse à l'utilisateur toute liberté de déterminer lui-même l'enchaînement.

Par exemple, si le concepteur de l'interface d'une application constate que cette dernière est destinée à des experts, il peut désirer n'imposer à ces utilisateurs aucune contrainte d'enchaînement. Il ne fixera donc aucun ordre de saisie ou d'affichage des messages interactifs. Cela correspond en fait à une structure parallèle portant sur la totalité des messages interactifs rattachés à la phase qu'il modélise.

L'enchaînement parallèle de deux messages est schématisé à la figure 17.

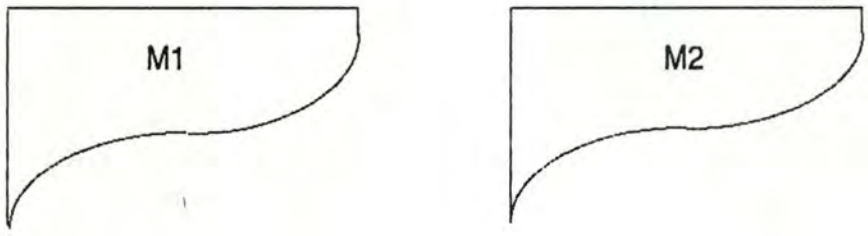


Figure 17 - Structure d'enchaînement parallèle entre deux messages M1 et M2.

III.4.3.2 - La séquence

Un enchaînement de messages est *séquentiel* lorsque la terminaison de saisie ou d'affichage d'un message M1 provoque le déclenchement de la saisie ou de l'affichage d'un message M2. M1 et M2 doivent apparaître au terminal l'un après l'autre, et dans un ordre déterminé. La figure 18 illustre ce type d'enchaînement.

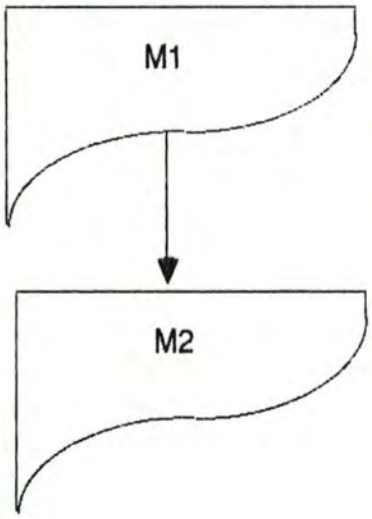


Figure 18 - La fin de saisie de M1 provoque le début de saisie de M2.

III.4.3.3 - L'enchaînement éclaté

Un enchaînement est *éclaté* lorsque la terminaison d'un message M1 (saisie ou affichage) provoque le déclenchement en parallèle de M2, M3, ..., Mn (saisie ou affichage). La fin de M1 permet à l'utilisateur de choisir l'ordre dans lequel il désire effectuer l'activation de la saisie ou l'affichage de M2, M3,..., Mn. La figure 19 fournit une représentation graphique de l'enchaînement éclaté.

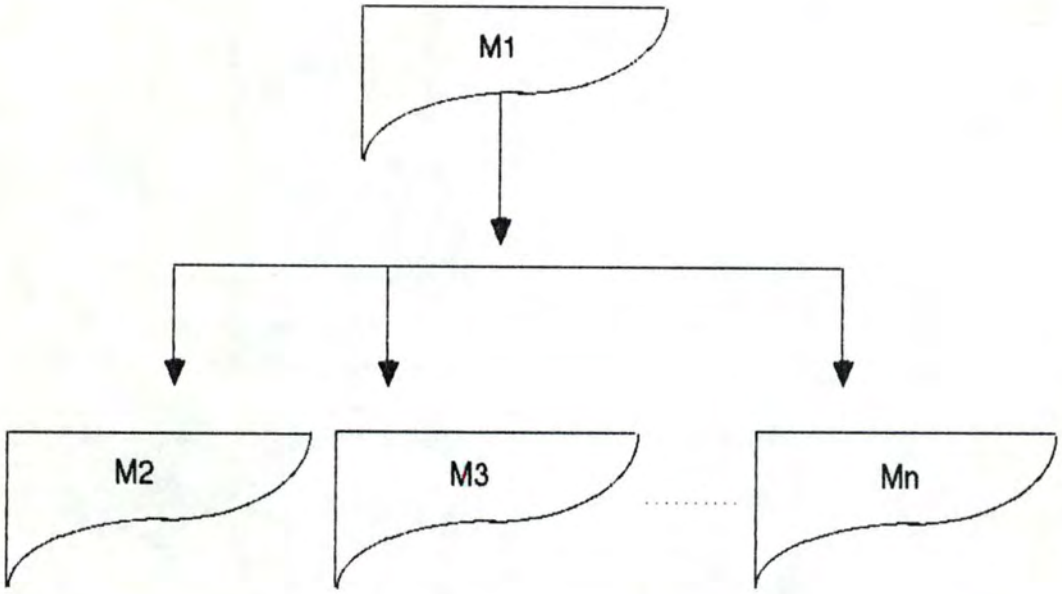


Figure 19 - L'enchaînement éclaté.

III.4.3.4 - L'enchaînement convergent

Un enchaînement de messages est *convergent* si le déclenchement de la saisie ou de l'affichage d'un message Mi est provoqué par la terminaison de la saisie ou de l'affichage d'un des messages M1 ou M2 ou M3 ou ... ou Mn.

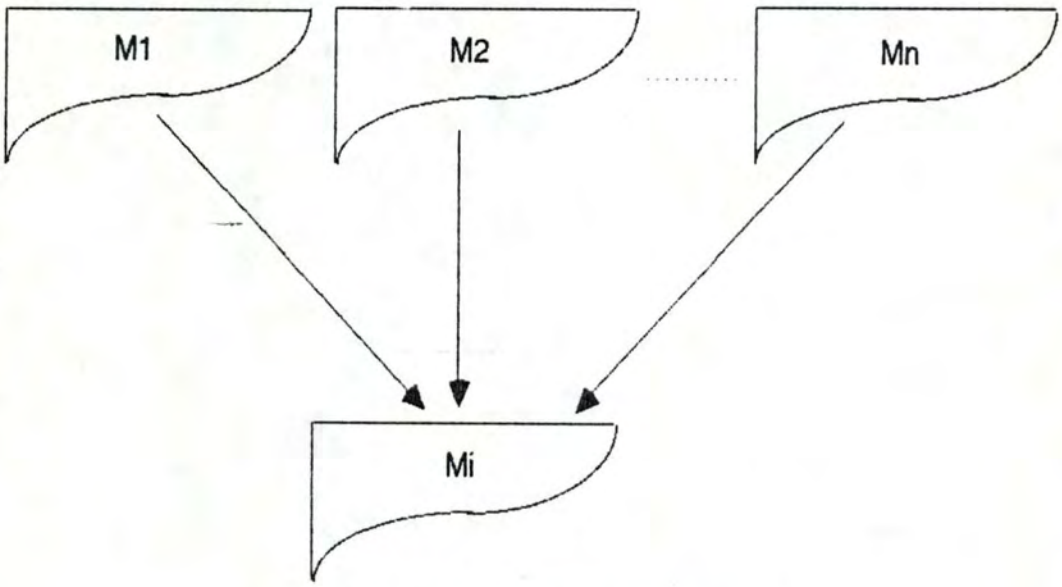


Figure 20 - L'enchaînement convergent.

III.4.3.5 - La synchronisation entre différents messages

Un mécanisme de synchronisation sera nécessaire pour décrire un type d'enchaînement dans lequel deux ou plusieurs messages interactifs doivent être saisis ou affichés avant de pouvoir passer à la saisie ou à l'affichage d'un autre message.

Le *point de synchronisation* est un mécanisme qui permet de déclencher la saisie ou l'affichage du message Mi après avoir terminé la saisie ou l'affichage de plusieurs autres messages M1, M2, ..., Mn. Les messages M1, M2, ..., Mn peuvent être différentes occurences d'un même type de message ou des occurences de types de message différents. On distingue donc deux situations qui peuvent être représentées par un point de synchronisation : la première est la conjonction, la seconde l'accumulation. La conjonction s'effectue sur des messages de types différents. L'accumulation est réalisée sur des messages d'un même type; nous reparlerons de cette dernière dans le paragraphe III.4.3.6 abordant l'enchaînement itératif.

La représentation graphique du point de synchronisation est un trapèze. Par convention, les flèches qui aboutissent au trapèze représentent la contribution des messages aux réalisations du point de synchronisation; celles qui en sont issues représentent le déclenchement de la saisie ou de l'affichage d'autres messages ou la contribution à d'autres synchronisations. La figure 21 présente le point de synchronisation.

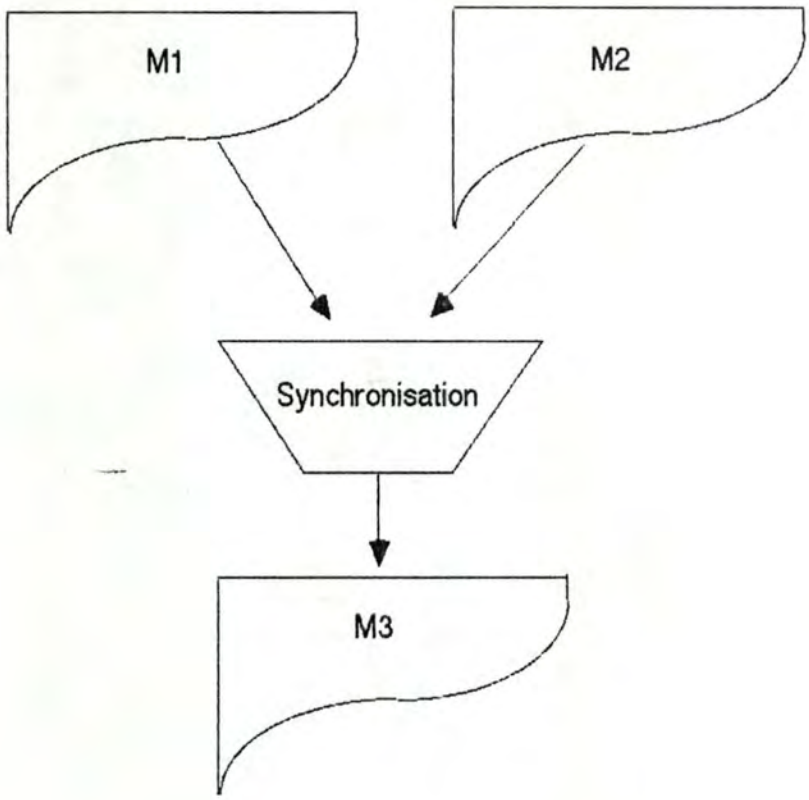


Figure 21 - La synchronisation de M1 et M2 permet de déclencher le message M3.

En termes de fonctionnement, et dans la mesure où une synchronisation correspond à une attente, un tel mécanisme induit par conséquent un phénomène de mémorisation. En effet, s'il s'agit d'attendre la fin de saisie ou d'affichage de deux messages M1 et M2 avant de déclencher la saisie ou l'affichage du message M3, le mécanisme de synchronisation devra mémoriser la fin du premier message (M1 ou M2) jusqu'à la fin du second (M2 ou M1). A ce moment, la synchronisation est réalisée; le message M3 peut donc être déclenché. La fin de saisie ou d'affichage des messages M1 et M2 peut alors être "oubliée ". Par "oubliée ", il faut comprendre que la fin de saisie de ces messages n'interviendra plus dans aucune réalisation ultérieure de la synchronisation.

Au niveau de la spécification, il suffit de définir précisément, pour chaque point de synchronisation, les éléments suivants :

- le ou les types de messages contribuant à la synchronisation, ainsi que la condition de réalisation de celle-ci;
- le nom des messages dont la saisie ou l'affichage sera déclenché à la réalisation de cette synchronisation.

III.4.3.6 - L'itération

Une structure est *itérative* si la terminaison de saisie ou d'affichage d'un message interactif M1 déclenche la saisie ou l'affichage de plusieurs messages de type M2.

La structure itérative peut porter sur un ou plusieurs messages, c'est-à-dire que la terminaison d'un message M1 peut déclencher un certain nombre de messages M2, et le même nombre de messages M3, M4, ...

Pour indiquer clairement sur quels messages porte l'itération, on peut introduire, dans la représentation graphique, un signe de fin d'itération. La fin d'itération signifie que, dans la suite de l'enchaînement, les messages seront produits en un exemplaire, mais elle signifie également que, pour pouvoir saisir ou afficher les messages faisant partie de la suite du scénario, il faudra d'abord avoir terminé la saisie ou l'affichage de tous les exemplaires des messages M2, M3, ... La fin d'itération est donc un type particulier de synchronisation, qui correspond à l'accumulation de messages que nous avons définie au paragraphe précédent. Cette synchronisation sera spécifiée exactement comme toutes les autres; sa représentation graphique un peu différente, sous la forme d'un triangle posé sur sa pointe, permet une compréhension plus rapide du type de synchronisation qui a lieu.

La figure 22 présente cette structure d'enchaînement dans sa forme la plus simple.

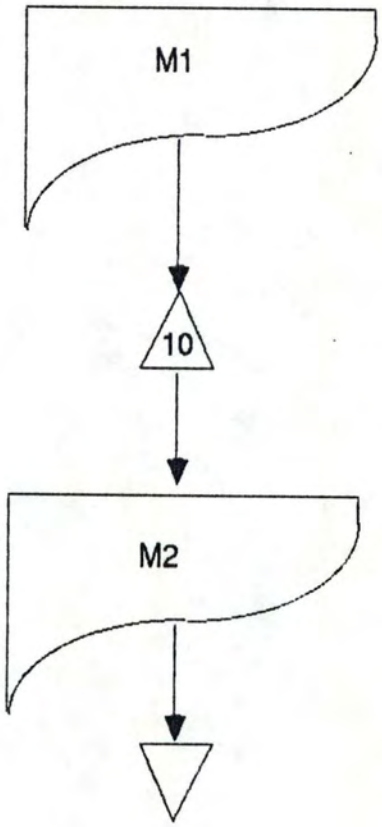


Figure 22 - L'enchaînement itératif.
Après le message interactif M1, le message M2 sera saisi 10 fois.
Les triangles délimitent l'ensemble des messages qui seront saisis en plusieurs exemplaires.

La figure 23 illustre l'itération réalisée sur les messages M2 et M3 qui, eux, sont saisis de manière parallèle c'est-à-dire dans un ordre indifférent.

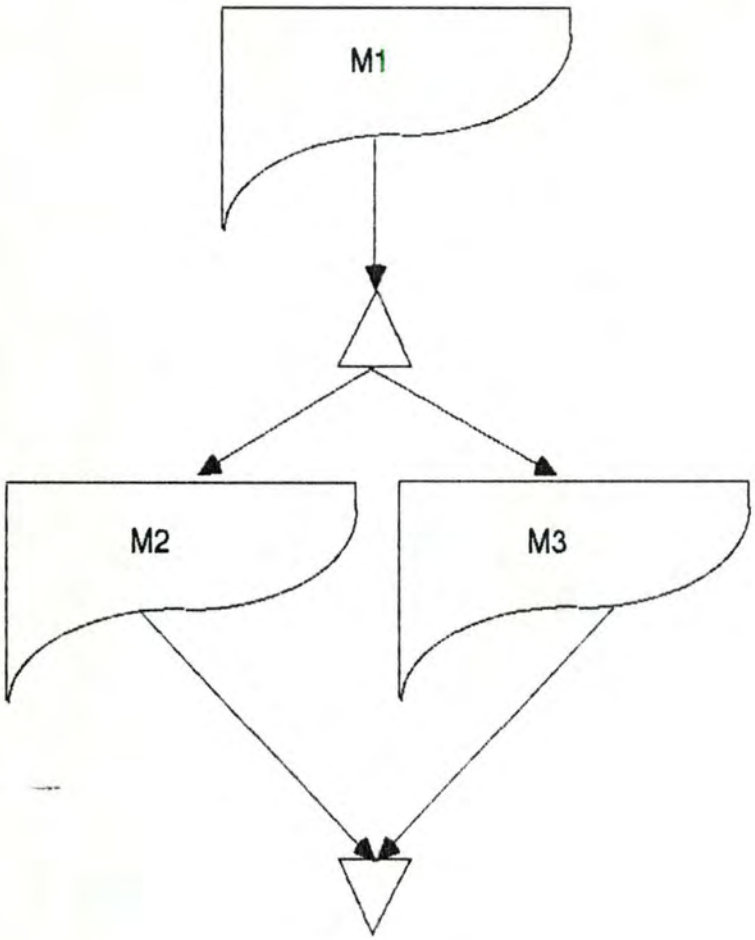


Figure 23 - Itération des messages M2 et M3.

Différents types de condition peuvent déterminer le nombre d'itérations qui seront effectuées sur les messages situés sur le schéma entre les deux triangles délimitant la structure itérative.

L'itération peut être réalisée un nombre fixe de fois; elle peut aussi s'effectuer jusqu'à ce que l'utilisateur signale qu'il désire arrêter ce type de saisie ou d'affichage.

La condition peut porter sur la valeur d'une zône saisie ou calculée pendant l'itération courante, mais elle peut également être plus complexe.

En effet, le nombre d'itérations d'une structure répétitive peut être déterminé par une combinaison de plusieurs conditions simples. Par exemple, l'itération s'arrêtera sur demande de l'utilisateur, à condition que le nombre d'itérations réalisées se trouve situé entre certaines bornes. Si l'on considère qu'une COMMANDE est constituée d'une à dix lignes de commande, l'utilisateur qui réalisera la saisie d'un bon de commande devra introduire au terminal un nombre de lignes de commande compris entre un et dix.

Dans le cadre de ce mémoire, nous ne développerons pas toutes les combinaisons possibles de ces différents conditions qui peuvent être envisagées.

III.4.3.7 - La structure de choix

Le *choix* est un type d'enchaînement par lequel la fin de saisie ou d'affichage d'un message M1 provoque le déclenchement de la saisie ou de l'affichage d'un message M2 ou M3 ou ... ou Mn.

Le message M1 est un message exclusivement interactif du type contrôle, qui permet d'orienter le dialogue.

Ce type de structure est par exemple utile pour exprimer un enchaînement où l'utilisateur doit choisir d'effectuer un traitement parmi un certain nombre, et ensuite exécuter la saisie du ou des messages nécessaires à la réalisation de ce traitement.

La représentation graphique de la structure de choix est l'hexagone, comme le montre la figure 24.

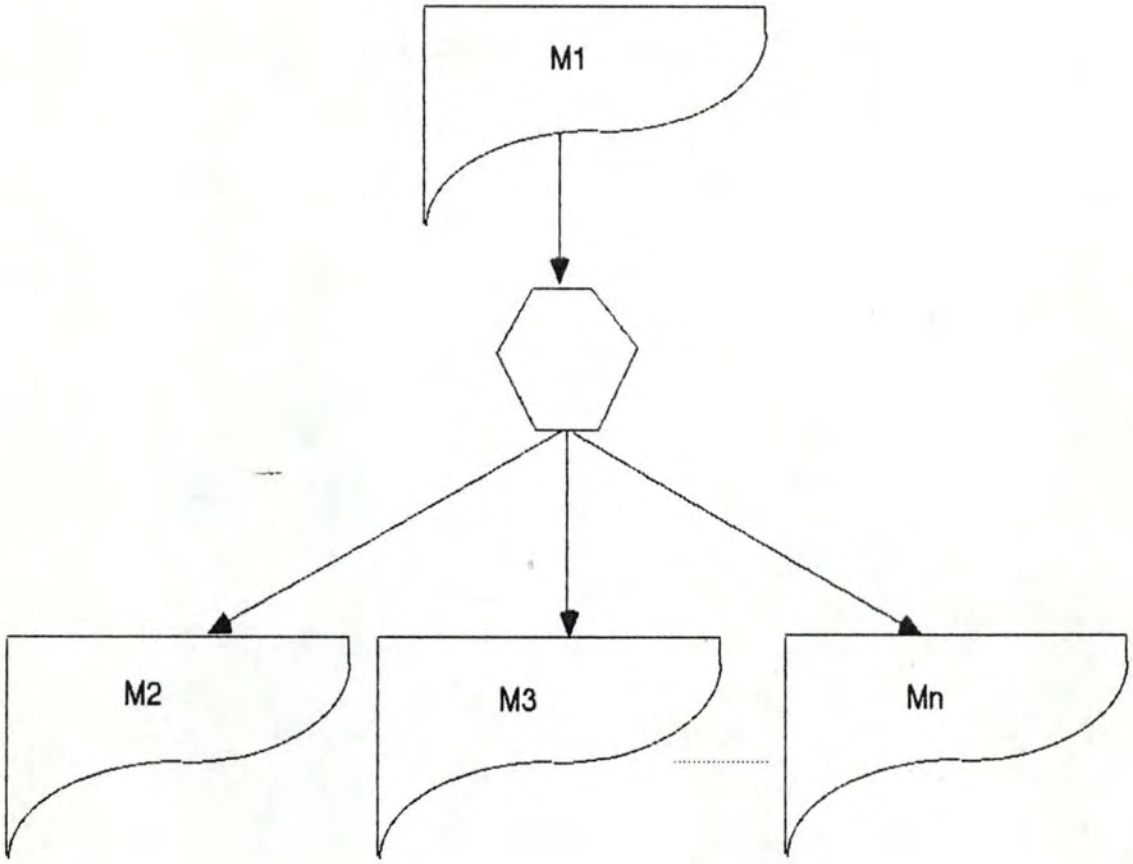


Figure 24 - Le choix

III.4.3.8 - Le retour-arrière

Le dernier enchaînement que nous étudions est le retour-arrière. Il en existe de deux types.

Le premier type est celui qui est provoqué par l'application. Nous avons déjà discuté de ce retour au paragraphe II.4.4. L'application fonctionnelle détecte une erreur sémantique dans les données qui ont été introduites par l'utilisateur. Ce dernier est averti de l'erreur par l'intermédiaire d'un message fonctionnel d'erreur.

Le message fonctionnel d'erreur doit indiquer le ou les endroits où s'effectue le retour-arrière; en effet, seule l'application fonctionnelle peut déterminer les données qui sont erronées, et par là, le ou les points de retour. Ensuite, l'utilisateur doit effectuer une nouvelle saisie (une saisie corrigée) du ou des messages invalidés par les traitements.

Ce retour-arrière se traduit par un message de sortie. En outre, il ne peut se produire qu'à certains endroits du dialogue, qui peuvent être prévus par le concepteur de dialogue d'après les renseignements fournis par la statique des traitements à propos des messages-externes-fonction d'erreur.

Les messages fonctionnels d'erreur peuvent, si le concepteur le désire, apparaître dans l'enchaînement dynamique des messages. Ils appartiendront alors à une structure de choix permettant, dans le cas où il y aurait une erreur, de revenir à la saisie du message qui a provoqué cette erreur, et dans le cas contraire, de continuer le déroulement du scénario de dialogue.

Le second type de retour-arrière est provoqué par le dialogue. A tout moment, l'utilisateur peut décider d'effectuer un retour-arrière à n'importe quel endroit du dialogue et d'effectuer une nouvelle saisie d'un ou de plusieurs messages. Ceci a bien entendu pour effet de supprimer les données qui avaient été saisies antérieurement par ces messages. De ce type de retour-arrière, on déduit que tout endroit de déclenchement de saisie ou d'affichage d'un message interactif est un point de retour potentiel.

Les retours demandés par l'utilisateur sont totalement imprévisibles pour le concepteur de dialogue. Ce type d'enchaînement ne sera donc pas spécifié par le concepteur de dialogue, ni représenté sur le schéma d'enchaînement des messages interactifs. Il est, en effet, inutile de surcharger le scénario d'enchaînement avec des informations qui sont valables partout.

Ceci termine la liste des différents types d'enchaînement des messages interactifs de saisie et d'affichage.

Pour que le concepteur de dialogue puisse spécifier de manière précise le scénario du dialogue d'une application interactive, il faudrait encore associer à ce modèle un langage de spécification. La création d'un tel langage n'a pas été réalisée dans le cadre de ce mémoire.

III.4.4 - La Conversation de l'exemple "mutuelle "

Nous avons décrit deux types de Présentation pour l'enregistrement d'une prestation. Nous exposons donc également deux schémas de la Conversation de cette phase.

Le premier d'entre eux est très simple, puisqu'il n'y a qu'un seul message à saisir. Dans ce cas, il ne sert à rien d'insérer les messages de sortie dans le schéma de la dynamique du dialogue. En effet, s'il n'y a aucun problème dans le déroulement de la phase, le récapitulatif est le seul message de sortie qui doit apparaître au poste de travail. Son apparition à l'écran ne saurait pas gêner l'utilisateur qui, à ce moment, a déjà terminé la saisie de la feuille de soins; le message sera donc affiché le plus tôt possible. Or, c'est exactement ce qui se passe s'il n'est pas présent dans le schéma de la dynamique. La figure 25 représente le premier schéma de la dynamique du dialogue de notre exemple.

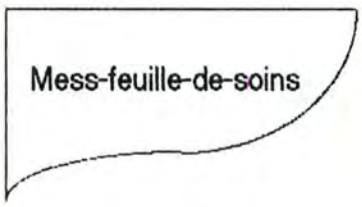


Figure 25 - Dynamique du dialogue dans le cas où il n'y a qu'un seul message de saisie.

La seconde spécification de la Présentation des messages interactifs donne lieu à un schéma d'enchaînement un peu plus compliqué présenté à la figure 26. Cet enchaînement laisse toute liberté à l'utilisateur quant à l'ordre de saisie des données concernant l'individu et de la date. Cependant, ces données doivent être totalement saisies avant qu'il lui soit permis d'introduire les lignes de prestation. Ces lignes de prestation donnent lieu à une itération dont le nombre de répétition n'est pas limité.

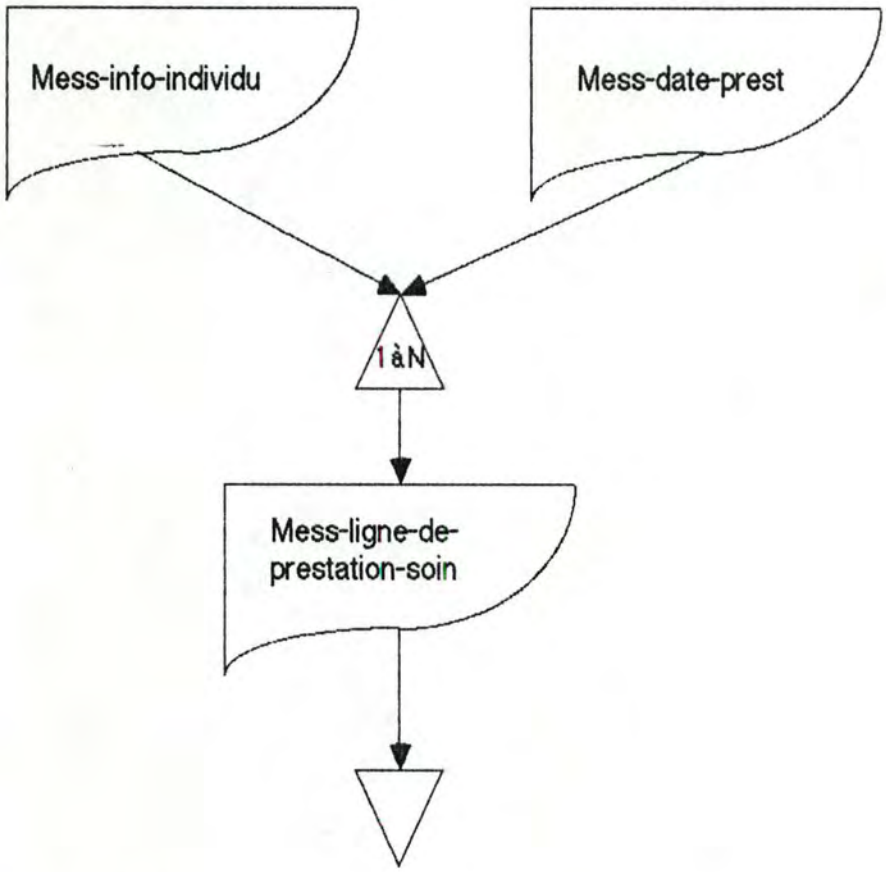


Figure 26 - Enchaînement des messages interactifs dans le cas d'une saisie en plusieurs messages.

III.5 - En résumé

Dans ce chapitre, nous avons tout d'abord décrit les messages et les objets interactifs, qui constituent les objets sur lesquels porte le dialogue. Les messages interactifs ont été classifiés. Un certain nombre d'objets interacitfs ont été énumérés et détaillés.

Nous avons ensuite cité les différentes fonctions que doit remplir le composant "Dialogue " d'une application interactive (saisie, affichage, aide, interruption, ...).

Enfin, nous avons présenté deux modèles qui permettent de décrire totalement le dialogue d'une application. Le premier d'entre eux est le modèle de la Présentation, qui utilise les objets interactifs pour offrir à l'utilisateur une vue ergonomique des messages interactifs. Le second est la Conversation, qui décrit le scénario d'enchaînement des messages à l'écran.

Après avoir modélisé les traitements et le dialogue d'une application interactive, nous allons maintenant décrire le modèle de l'Echange, qui effectue un lien entre les deux précédents modèles et permet la collaboration entre le composant "Dialogue " et le composant "Traitement " de l'application.

Chapitre 4 :

La modélisation de l'échange

Introduction

Objectif du modèle

L'objectif du modèle de l'échange entre les traitements et le dialogue d'une application est de définir les concepts liés à la transmission des informations d'un composant à l'autre en vue d'assurer la cohésion et la cohérence de l'ensemble de l'application.

Le modèle spécifie la base de la collaboration nécessaire entre les deux parties de l'application, à partir des notions qui ont été introduites et explicitées d'une part dans le modèle de l'application fonctionnelle et, d'autre part, dans le modèle du dialogue. La structure d'échange est en effet le seul support de communication, et donc de collaboration, entre les fonctionnalités et l'interface de l'application.

Application interactive, application répartie ?

La décomposition d'une application interactive en deux parties distinctes (traitements et dialogue) rappelle certains principes qui ont été définis dans le domaine de la conception d'applications réparties.

Dans le cadre de la modélisation de l'échange, nous reprendrons certaines notions mises en évidence par la gestion d'une application répartie et qui peuvent également nous être utiles. En effet, certaines caractéristiques des composants d'une application répartie sont également valables pour les composants de l'application interactive.

Ainsi, traitements et dialogue peuvent être considérés comme deux modules de l'application qui sont autonomes mais coopèrent à la réalisation d'une tâche commune globale, l'exécution de l'application, en partageant les ressources dont elle dispose. Pour certaines parties, ces modules peuvent s'exécuter en parallèle mais, de toute façon, ils ne partagent pas de variables communes, chaque composant utilisant ses propres objets. Cette dernière propriété est d'ailleurs une propriété discriminante des applications réparties par rapport à un système centralisé.

Par contre, certains problèmes rencontrés pour la gestion d'applications réparties ne se posent pas dans le cadre d'une application interactive. Par exemple, le concept de l'exclusion mutuelle de différents processus, s'exécutant en parallèle sur différents sites, est essentiel pour une application répartie : il vise à éviter l'interférence des processus notamment dans leurs accès aux données, que celles-ci soient centralisées sur un même site, réparties sur plusieurs, ou existent en copies multiples sur chaque site. Ce type de problème ne se pose pas dans notre cadre de réflexion car il n'y a pas d'interférence possible entre les deux composants de l'application. D'une part, le dialogue ne peut pas faire d'accès à la base de données de l'application. D'autre part, les deux composants de l'application (traitements et dialogue), assimilés ici aux processus concurrents de l'application répartie, assument des fonctionnalités indépendantes, si ce n'est le fait que l'acquisition et l'émission de données du point de vue des traitements se font via l'interface. Toutefois, cet échange de données se fait de telle sorte que chaque composant n'utilise que les objets qui lui sont propres et qui ne sont donc pas partagés avec l'autre. Dans notre domaine, il n'y a donc pas de risque d'interférence entre les deux composants de l'application en cours d'exécution.

IV.1 - Objets de l'échange

Par similitude avec la façon dont se fait la communication dans les applications réparties, nous prendrons comme optique que les objets manipulés par le modèle de l'échange sont des *requêtes*, ou *commandes*, paramétrées que se transmettent les deux composants de l'application afin d'acquérir certaines informations connues ou produites par l'autre composant et d'agir sur leur déroulement respectif quand cela s'avère nécessaire. Conformément au principe de séparation présenté au paragraphe I.3, ce dernier type d'action d'un composant sur l'autre devra cependant être limité le plus possible.

Cette optique, qui consiste à spécifier un seul type d'objet de l'échange, présente l'avantage d'en proposer une définition uniforme sous la forme générique des requêtes.

En analysant les caractéristiques essentielles de la communication qui s'établit normalement entre les fonctionnalités et le dialogue d'une application, nous avons déterminé deux grandes classes de requêtes, d'après le rôle qu'elles remplissent au niveau de la collaboration entre les traitements et l'interface. Nous distinguons ainsi les requêtes qui concernent :

- soit la gestion des entrées/sorties,
- soit le contrôle de l'exécution de l'application (déclenchement d'un traitement, retour-arrière dans le dialogue et/ou les traitements suite à une erreur sémantique).

Pour expliciter la façon dont la structure d'échange communique ces requêtes d'un composant de l'application à l'autre, il nous faut d'abord les décrire un peu plus en détail. C'est ce que nous allons faire maintenant.

Notons que les requêtes que nous exposons ici sont les requêtes envoyées par l'application fonctionnelle ou par le dialogueur vers l'autre composant de l'application interactive. Ces objets du modèle de l'échange ont donc déjà été cités dans les deux modèles précédents.

IV.1.1 - Requêtes de gestion des entrées/sorties

Le premier type de requête dont l'échange doit assurer la transmission correspond aux requêtes de gestion des entrées/sorties.

Ces commandes ont pour but d'effectuer la transmission, entre les traitements et le dialogue, des messages de l'application, c'est-à-dire, du point de vue de l'application fonctionnelle, de messages-externes-fonction en entrée et en sortie et, du point de vue de l'interface, de messages interactifs à saisir ou à afficher.

On peut donc considérer que les messages échangés par l'application constituent les paramètres des commandes d'acquisition et d'émission d'information pour les traitements, de saisie et d'affichage de données pour le dialogue.

Les requêtes de gestion des entrées/sorties peuvent dès lors être spécifiées de la façon suivante :

pour le composant "Traitements" :

- requérir message T1, où T1 est un message-externe-fonction d'entrée complet,
- réceptionner message T2, où T2 est un message-externe-fonction d'entrée complet,
- émettre message T3, où T3 est un message-externe-fonction de sortie complet;

pour le composant "Dialogue" :

- saisir message D1, où D1 est un message interactif d'entrée complet,
- communiquer message D2, où D2 est un message interactif d'entrée complet,
- afficher message D3, où D3 est un message interactif de sortie complet .

La sémantique de ces requêtes est la suivante :

- "requérir message T1" :

  les traitements demandent à la structure d'échange de leur fournir une occurrence du message-externe-fonction T1, nécessaire à l'exécution de la phase courante qui la requiert comme message d'entrée;

- "réceptionner message T2" :

  la structure d'échange demande aux traitements de recevoir le message-externe-fonction T2 construit à partir de valeurs introduites par l'utilisateur pour des champs du ou des messages interactifs qui correspondent à la saisie de T2;

- "émettre message T3" :

  les traitements demandent à la structure d'échange l'émission du message-externe-fonction T3, ce message étant soit un message résultat "pur", soit un message d'erreur;

- "saisir message D1" :

  la structure d'échange demande à l'interface d'effectuer la saisie auprès de l'utilisateur d'une occurrence du message interactif D1. Comme nous l'avons déjà souligné dans la description de la fonction de saisie donnée au paragraphe III.2 sur la modélisation du dialogue, cette commande inclut la validation syntaxique des données à introduire pour constituer le contenu d'une occurrence du message interactif D1;

- "communiquer message D2" :

  l'interface demande à la structure d'échange de transmettre aux traitements le message interactif D2 , message d'entrée dont l'utilisateur a terminé l'introduction.

  La requête "communiquer message Di" répond à une requête " saisir message Di" ou sert à transmettre directement aux traitements une occurrence du message d'entrée Di qui n'a pas fait l'objet d'une demande explicite d'acquisition de la part des traitements;

- "afficher message D3" :

  la structure d'échange demande à l'interface d'effectuer l'affichage du message interactif D3;

Notons que l'on peut bien entendu envisager la possibilité que les paramètres "message" de ces différentes requêtes soient des paramètres répétitifs (1 à n fois) afin de permettre le regroupement des demandes d'entrée/sortie de données émises par les deux composants de l'application.

En distinguant ces deux groupes de requêtes, il devient inutile de préciser l'origine de chaque commande car celle-ci peut être déduite de l'identificateur de la commande; le type du paramètre message (message-externe-fonction ou message interactif) peut également être déterminé de la même façon.

La relation existant entre ces diverses commandes est explicitée plus longuement au paragraphe IV.2.1.1.

IV.1.2 - Requêtes de contrôle de l'exécution de l'application

Le second type de requête que la structure d'échange transmet d'un composant de l'application à l'autre correspond aux requêtes de contrôle de l'exécution de l'application.

Certaines commandes échangées par les traitements et le dialogue sont en effet utilisées pour permettre leur collaboration en vue d'assurer le bon déroulement de l'ensemble de l'application. Leurs paramètres sont soit des messages (fonctionnels interactifs ou externes-fonction), soit d'autres structures de données telles que par exemple l'identificateur d'un traitement.

Avant d'entamer la description de ces requêtes, précisons que les traitements envisagés ici sont des phases interactives, sauf indication contraire.

Dans le sens dialogue - traitements, on trouve les requêtes suivantes :

- *activer le traitement P* (application ou phase) :

  Quand elle concerne l'application, cette requête lance les opérations d'initialisation nécesaires à son exécution (chargement de la description de l'application, vérification de la disponibilité des ressources requises par l'application,...).

Si le traitement P à activer est une phase, cette requête permet au dialogue de signaler explicitement sa demande de déclenchement aux traitements suite à une demande d'exécution de la phase provenant de l'utilisateur. Cette demande peut avoir été introduite explicitement en sélectionnant le traitement P via un menu par exemple, ou implicitement par l'introduction d'une occurence d'un message interactif associé à un message-externe-fonction d'entrée de la phase.

Notons qu'il est nécessaire d'effectuer un déclenchement explicite de la phase, que celle-ci soit interactive ou automatique. En effet, on pourrait penser que le déclenchement d'une phase interactive débutant par la saisie d'un message s'effectue lors de la réception de ce message par le composant "Traitements ". Cependant, cela signifierait que l'on traite le premier message d'une phase de manière différente des autres, puisqu'au message déclencheur de la phase serait attachée une fonction de "lancement/chargement de la phase ". Une telle distinction entre les messages n'est pas souhaitable. De plus, dans le cas d'une phase automatique, le déclenchement des traitements doit absolument se faire de manière explicite. Il est donc toujours nécessaire d'envoyer au composant "Traitement " de l'application une requête de déclenchement de phase pour que celle-ci puisse s'exécuter.

- *clôturer le traitement P* (application ou phase) :

S'il s'agit de l'application, cette requête conduit à effectuer les opérations de terminaison de l'application, c'est-à-dire les opérations nécessaires pour quitter l'application et revenir au système, celles-ci dépendant du contexte de l'application au moment de la demande de clôture (phase courante interrompue ou abandonnée,...).

S'il s'agit d'une phase, la requête provoque la mise à jour de la base de données en fonction de l'exécution de la phase et après confirmation explicite du traitement par l'utilisateur.

- *annuler le traitement P* :

Cette commande consiste en fait à faire revenir l'application dans l'état où elle se trouvait avant l'activation de la phase annulée.

- *interrompre et reprendre le traitement P* :

Par rapport à l'annulation d'un traitement, le système doit prévoir la reprise du traitement interrompu. Dans ce cas-ci, il faut donc maintenir la mémoire de ce qui a déjà été fait. On peut également envisager la possibilité de libérer certaines ressources (les processeurs) utilisées par le traitement mis en attente.

Il faut cependant signaler que, si ce n'est cette possibilité de libérer les processeurs attribués au traitement interrompu, l'interruption et la reprise d'un traitement devraient être des opérations transparentes pour l'application fonctionnelle et pour l'utilisateur.

- *retourner au message D_i*, où D_i est un message interactif :

Dans le cas où l'utilisateur doit corriger une ou plusieurs données d'un ou de plusieurs messages interactifs, le dialogue doit renvoyer les messages corrigés aux traitements en signalant quelles occurences de quels messages précédemment transmis doivent être modifiées en conséquence.

Rappelons que ce type de retour-arrière se fait uniquement à l'intérieur d'une phase et nécessite généralement un "détricotage", partiel ou total, de l'exécution de la phase avec les valeurs précédemment introduites qui ont conduit à la production d'un message d'erreur.

Dans le sens traitement - dialogue, on rencontre les requêtes suivantes :

- *accepter (refuser) le déclenchement du traitement P* :

A la requête "activer le traitement P " en provenance du dialogue, le composant "Traitements" doit répondre par la requête "accepter (refuser) le déclenchement du traitement P".

Cette commande permet essentiellement à l'application fonctionnelle de signaler au dialogue que les conditions de déclenchement du traitement P, autres que celles portant sur la réception d'un ou de plusieurs messages-externes-fonction d'entrée, sont remplies. Ces conditions concernent la réception d'un message interne ou d'un message-externe-fonction qui ne provient pas de l'utilisateur (un message horaire par exemple) ou éventuellement la non disponibilité de certaines ressources.

Cette requête peut se traduire au niveau de l'utilisateur par l'affichage d'un message explicite lui signalant l'activation du traitement ou, simplement, par la présentation des messages interactifs d'entrée permettant la saisie des messages-externes-fonction d'entrée associés aux fonctions de la phase.

- *signaler la terminaison du traitement P* :

Pour les traitements du niveau phase, cette requête permet d'indiquer à l'utilisateur la confirmation de la mise à jour de la base de données, si l'exécution de la phase a été confirmée, ou l'abandon de la phase, si elle a été annulée.

On notera qu'il semble normal qu'il existe un plus grand nombre de requêtes de contrôle dans le sens dialogue - traitements, car l'exécution d'une application interactive est généralement controlée par l'utilisateur, même si celui-ci répond simplement à des demandes du système. Comme nous l'avons déjà souligné au chapitre 1, ceci permet à l'utilisateur de ne pas se sentir l'esclave du système mais, au contraire, d'avoir l'impression que la machine et l'application sont avant tout à son service.

Pour généraliser la spécification d'une requête, nous dirons donc qu'une requête contient essentiellement :

- un identificateur,
- et une liste de paramètres "fonctionnels", c'est-à-dire correspondant à des objets sémantiques de l'application (messages-externes-fonction ou messages interactifs, identificateur de traitement).

Elle définit aussi éventuellement un ou plusieurs paramètres de contrôle de la transmission, comme nous l'expliquons au paragraphe IV.2.3 relatif à la gestion de l'échange.

Après avoir présenté les objets du modèle de l'échange, nous pouvons maintenant détailler les fonctions assumées par ce modèle.

IV.2 - Fonctions de l'échange

Etant donné que l'échange est basé sur la notion de requête, sa fonction première est l'analyse des requêtes, qui sont transmises d'un composant de l'application à l'autre par son intermédiaire, en vue de les adapter, si nécessaire, au composant destinataire. Pour ce faire, il les transforme éventuellement de telle façon que requêtes et paramètres fonctionnels correspondent à ce que connaît le composant destinataire. Après cela, ces requêtes adaptées peuvent être transmises. Les conditions de transmission des requêtes, notamment leur ordre de priorité, doivent pouvoir être définies par le concepteur de l'application. Cependant, il est préférable que des conditions standard soient définies afin de libérer le concepteur de cette tâche, s'il le désire. Le paragraphe IV.2.3 traite de ces conditions de transmission.

Pour assurer son rôle d'échange entre les traitements et le dialogue, la structure d'échange doit donc remplir les fonctionnalités détaillées ci-après.

IV.2.1 - Transformation des requêtes de gestion des entrées/sorties

IV.2.1.1 - Identification des requêtes

Les requêtes de gestion des entrées/sorties sont identifiées différemment par le composant "Traitement" et le composant "Dialogue", comme indiqué au paragraphe IV.1.1. Quand la structure d'échange reçoit, en provenance de l'application fonctionnelle, la commande "acquérir message T1" ou "émettre message T3", elle doit les transmettre à l'interface sous la forme "saisir message D1" ou "afficher message D3". Si elle reçoit de l'interface la commande "communiquer message D2", elle doit transmettre ce message D2 aux traitements, via la requête "réceptionner message T2".

Dans les deux sens, les paramètres de type "message" doivent être adaptés à la vue des messages propre à chaque composant. C'est le sujet du paragraphe suivant.

IV.2.1.2 - Mise en correspondance message-externe-fonction / message fonctionnel interactif

La structure d'échange doit effectuer la transformation des paramètres de type message-externe-fonction en messages interactifs, et inversément, afin de les rendre conformes à la vue qu'en connait le composant destinataire.

Pour ce faire, elle dispose de la spécification des messages-externes-fonction fournie par le concepteur de l'application et de la spécification des messages interactifs, fournie par le concepteur de dialogue. Ces spécifications doivent toutefois être complétées comme nous allons le préciser immédiatement, pour permettre l'échange des données conformément à la vue qu'en possède chaque composant.

La mise en correspondance de ces deux structures doit se faire champ par champ. Du point de vue du modèle de l'échange, la spécification des messages doit donc indiquer, pour chaque champ d'un message fonctionnel interactif, s'il existe une correspondance avec un champ d'un message-externe-fonction et, si oui, avec quel champ de quel message.

Notons toutefois que la correspondance avec un champ d'un message-externe-fonction n'existe pas nécessairement pour tous les champs d'un message fonctionnel interactif car certains d'entre eux peuvent ne pas faire référence à des données d'un message-externe-fonction s'il s'agit, par exemple, d'un champ introduit dans le message interactif par le concepteur de dialogue pour en faciliter la saisie ou améliorer son affichage.

En ce qui concerne les messages interactifs d'entrée, la mise en correspondance des champs qui les composent est faite avec un ou plusieurs champs d'un ou de plusieurs messages-externes-fonction d'entrée, au fur et à mesure qu'ils sont communiqués par l'interface à la structure d'échange, grâce à la commande " communiquer message D2 ".

Une ou plusieurs occurences de messages-externes-fonction peuvent ainsi être constituées à partir des valeurs d'un ou plusieurs champs d'un ou plusieurs messages interactifs. Aussitôt qu'une occurence d'un message-externe-fonction est complétée, c'est-à-dire contient une valeur pour tous les champs du message considéré, elle peut être transmise à l'application fonctionnelle par la structure d'échange grâce à la requête "réceptionner message T2".

De même, en ce qui concerne les messages interactifs résultat "pur" (messages de sortie), la mise en correspondance de messages-externes-fonction à afficher est effectuée, champ par champ également, avec le ou les messages interactifs qui vont permettre leur visualisation au terminal. Cette transformation peut avoir lieu dès réception d'un message en provenance des traitements de l'application, via la commande " émettre message T3 ".

La ou les occurences du ou des messages interactifs constituées à partir des champs d'un ou plusieurs messages-externes-fonction résultat "pur" sont transmises à l'interface dès qu'elles sont complètes. Pour ce faire, la structure d'échange utilise la commande " afficher message D3 ".

Enfin, pour les messages d'erreur, la mise en correspondance est réalisée de manière légèrement différente.

En effet, la spécification fonctionnelle d'un message-externe-fonction d'erreur ne contient pas uniquement des informations à afficher. Seul le justificatif de l'invalidation sémantique qui doit être signalée à l'utilisateur constitue un champ à afficher, via un message interactif fonctionnel d'erreur, avec lequel l'échange établit la correspondance comme pour un message résultat "pur".

Le reste de la spécification fonctionnelle des messages d'erreur fournit les informations nécessaires pour définir les points de retour-arrière vers un ou plusieurs champs d'un ou plusieurs messages-externes-fonction d'entrée et leurs conditions d'utilisation (combinaison et/ou). Dès lors, ce type de message nécessite également la mise en correspondance de ce ou ces champs avec le ou les champs d'une ou plusieurs occurences du ou des messages interactifs qui ont été utilisés pour leur introduction.

Pour terminer ce paragraphe, notons enfin que la mise en correspondance d'un message fonctionnel interatif avec un message-externe-fonction s'effectue grâce au complément de spécification des messages destiné à l'échange.

Arbitrairement, nous choisissons de l'adjoindre à la spécification des messages interactifs plutôt qu'à celle des messages-externes-fonction. Il est bien sûr évident que ce complément de spécification des messages interactifs permet en fait de compléter automatiquement et ed manière symétrique la spécification des messages-externes-fonction. Ces deux spécifications de correspondance sont mises à la disposition de la structure d'échange. On réalise ainsi la mise en correspondance des deux types de message aussi facilement et rapidement dans le sens du message-externe-fonction vers le message interactif qui le présente au terminal, que dans le sens du message interactif vers le message-externe-fonction qu'il permet de saisir.

Après avoir détaillé les opérations réalisées par l'échange pour assurer la communication des requêtes de gestion des entrées/sorties entre les deux composants de l'application, nous allons préciser la façon dont la structure d'échange effectue la transmission des requêtes de contrôle de l'exécution de l'application.

IV.2.1.3 - Spécification de cette correspondance pour le cas "mutuelle"

Comme dans les chapitres précédents, nous reprenons le cas "mutuelle " pour illustrer la méthode de spécification de la correspondance entre messages-externes-fonction et messages-fonctionnels-interactifs. A chaque champ de chaque message-fonctionnel-interactif nous associons le champ auquel il correspond dans les messages-externes-fonction. De nouveau, cette opération doit être réalisée autant de fois qu'il existe de spécifiactions de la Présentation de l'application. Nous n'effectuerons ce travail que pour la présentation dans laquelle il n'y a qu'un seul message à saisir.

Message Mess-feuille-de-soins :

Le champ nom-ass **correspond au champ** nom de l'assuré **du message-externe-fonction** données-individu.

Le champ prénom-ass **correspond au champ** prénom **du message-externe-fonction** données-individu.

Le champ date-naissance-ass **correspond au champ** date de naissance **du message-externe-fonction** données-individu.

Le champ num-matricule **correspond au champ** matricule **du message-externe-fonction** données-individu.

Le champ date-prestation-soins **correspond au champ** date de prestation des soins **du message-externe-fonction** lignes-feuille-soins.

Le champ code-acte **correspond au champ** code-acte **du message-externe-fonction** lignes-feuille-soins.

Le champ montant-dr-acte **correspond au champ** montant-dr-acte **du message-externe-fonction** lignes-feuille-soins.

Le champ quantité-acte **correspond au champ** quantité-acte **du message-externe-fonction** lignes-feuille-soins .

Un travail analogue doit être effectué pour le message Mess-récapitulatif, ainsi que pour les messages d'erreur. Pour ces deux derniers messages, la correspondance n'est pas difficile à exprimer; il suffit d'une seule phrase.

Message Mess-erreur-individu :

Le champ champ-erreur-individu **correspond au message-externe-fonction** individu-non-valide.

Message Mess-erreur-actes :

Le champ champ-erreur-actes **correspond au message-externe-fonction** lignes-non-valides.

Ceci termine la spécification de notre exemple.

IV.2.2 - Transmission des requêtes de contrôle de l'exécution de l'application

Le passage des requêtes de contrôle d'un composant de l'application à l'autre s'effectue sans nécessiter d'adaptation particulière.

En effet, au niveau des requêtes sans paramètres, celles-ci sont connues et identifiées de la même façon par les deux composants.

Ensuite, le type de requête " retourner à un message (Di ou Ti)" a comme paramètre un message fonctionnel interactif (Di) ou un message-externe-fonction d'entrée (Ti). Ces derniers doivent donc être adaptés par la structure d'échange comme il est décrit au paragraphe IV.2.1.2 (Mise en correspondance message-externe-fonction / message fonctionnel interactif).

Enfin, les autres types de requête de contrôle ont comme paramètre un identificateur d'un traitement de l'application fonctionnelle. Cet identifiant, défini par les spécifications des traitements fournies par le concepteur de l'application, peut être considéré comme connu par le concepteur de dialogue. Celui-ci peut donc l'employer tel quel pour désigner le traitement de l'application utilisé comme paramètre de la requête.

La définition des opérations effectuées par l'échange pour assurer la communication des requêtes entre les deux parties de l'application étant ainsi précisée, il nous reste à expliciter les conditions dans lesquelles la transmission s'effectue.

IV.2.3 - Gestion de l'échange

Les fonctions de gestion de l'échange visent avant tout à définir les conditions de transmission entre les traitements et le dialogue pour assurer leur communication.

Celles-ci concernent :

- le type de synchronisation adopté par les deux composants de l'application pour la transmission des requêtes (émission/réception bloquante ou non),
- l'ordre de priorité pour la prise en compte des commandes en provenance des deux composants par la structure d'échange.

IV.2.3.1 - Types de synchronisation entre "Traitements" et "Dialogue

On distingue deux types de synchronisation selon que la réception (ou l'émission) d'une requête est bloquante ou non.

Généralement, la réception d'une requête de gestion des entrées/sorties sera toujours bloquante pour le composant traitements. En effet, elle correspond à la demande d'acquisition, via la commande "requérir message T1", d'un ou de plusieurs messages; le composant se trouve donc en état d'attente d'au moins un de ces messages avant de pouvoir continuer son activité.

L'émission d'une requête peut être non bloquante, ou bloquante, selon que le composant émetteur continue son activité ou doit attendre que le récepteur soit prêt à recevoir la requête qu'on veut lui envoyer (synchronisation par rendez-vous).

Par exemple, quand l'application fonctionnelle doit transmettre à l'interface tel message-externe-fonction de sortie, on peut envisager qu'avant de continuer son activité la partie "Traitements" attende ou pas que le dialogue soit prêt à recevoir ce message pour l'afficher.

Dans le cas de l'interface, on peut toutefois souligner qu'il serait assez inefficace que le dialogue doive attendre une réponse en provenance de l'application avant de pouvoir se poursuivre, cette réponse de l'application fonctionnelle signalant au dialogue la bonne réception du dernier message qu'il lui a envoyé.

IV.2.3.2 - Ordre de priorité pour la prise en compte des requêtes par l'échange

Quand la structure d'échange reçoit une requête en provenance d'un des deux composants de l'application, elle l'enregistre dans une file d'attente en lui attribuant un numéro d'arrivée séquentiel croissant par exemple. Dès qu'elle le peut, c'est-à-dire à la fin de la transmission de la requête précédente, elle extrait la requête suivante qui doit être transmise.

En fait, l'ordre d'extraction va dépendre de la nature et de l'ordre d'arrivée des requêtes.

Avant de préciser l'ordre normal de prise en compte, par la structure d'échange, des requêtes de contrôle et des requêtes de gestion des entrées/sorties émises par les composants de l'application, rappelons tout d'abord que nous envisageons cette question dans le cadre de l'exécution d'une phase.

Dans ce contexte, la première requête acceptée par la structure d'échange est la requête "activer le traitement P" envoyée par le dialogue aux traitements et à laquelle les traitements répondent par la requête "accepter (refuser) le traitement P".

A partir de cet instant, l'échange peut recevoir aussi bien des requêtes de gestion des entrées/sorties que des requêtes de contrôle de l'exécution.

Au niveau des requêtes de gestion des entrées/sorties, la structure d'échange adopte un ordre de prise en compte qui lui permet de rester le plus possible à jour par rapport au dernier état du contexte d'exécution de l'application.

Ainsi, elle communique les requêtes d'un composant de l'application à l'autre de la façon suivante :

- du point de vue de l'entrée de données, la structure d'échange analyse en priorité toute requête "communiquer message D2" en provenance de l'interface afin que la mise en correspondance des messages-externes-fonction puisse se faire en tenant compte de toutes les données saisies auprès de l'utilisateur, y compris les plus récentes. Ces requêtes sont transformées en des requêtes de type "réceptionner message T2" et transmises aux traitements sous cette forme.

Dès lors, quand la structure d'échange analyse une commande "requérir message T1" en provenance des traitements, elle ne la communique à l'interface via la requête "saisir message D1" que si elle n'a pas reçu de requête "communiquer message D2" pour les messages interactifs correspondant à ce message fonctionnel, et ceci, compte tenu de tous les messages interactifs qui lui ont été transmis jusqu'au moment où elle analyse la commande "requérir message T1";

- du point de vue de la sortie de données, aucune priorité particulière n'est à définir; les requêtes "émettre message T3" et "afficher message D3" sont donc transmises en fonction de leur ordre d'arrivée essentiellement.

  En fait, l'ordre d'extraction pourrait dépendre du contenu des paramètres fonctionnels des requêtes transmises (message d'erreur par exemple). Bien entendu, dans ce dernier cas, seuls les traitements ou le dialogue peuvent estimer l'urgence de la requête et donc la priorité à lui attribuer. Si la structure d'échange accepte qu'un composant de l'application spécifie le niveau de priorité de ces requêtes via leurs paramètres de contrôle, c'est un des cas où des conditions particulières de transmission des requêtes peuvent être précisées.

Au niveau des requêtes de contrôle de l'exécution de l'application, la structure d'échange adopte également un ordre de prise en compte des requêtes qui lui permet de "coller" autant que possible au dernier état du contexte d'exécution de l'application.

Ainsi, la commande "retourner au message Di" prime sur la commande "afficher message Dj" si Dj est un message d'erreur qui invalide un ou plusieurs champs du message Di ou un message résultat lié au message Di.

Les requêtes "annuler le traitement P" et "interrompre le traitement P" sont prises en compte avant toute requête de gestion des entrées/sorties,...

La fin de l'exécution de la phase est indiquée explicitement via le dialogue par la requête "clôturer le traitement P" ou "annuler le traitement P".

Après réception de cette requête, la structure d'échange ne peut plus accepter que la requête "signaler la terminaison du traitement P" en provenance des traitements.

IV.3 - En résumé

Après avoir précisé l'objectif du modèle de l'échange, que ce chapitre avait pour but de décrire, nous avons spécifié les objets de la structure d'échange dont le type générique correspond à la notion de requête.

Nous avons ensuite détaillé les fonctions prises en charge par ce modèle.

D'une part, nous avons défini la façon dont les deux types de requêtes (gestion des entrées/sorties et contrôle de l'application) sont transformées par la structure d'échange afin d'être transmises d'un composant de l'application à l'autre conformément aux concepts qui leur sont propres.

D'autre part, nous avons explicité les conditions de transmission qui déterminent le mode de gestion de l'échange (type de synchronisation, ordre de priorité pour la prise en compte des requêtes par l'échange).

Avec ce chapitre, nous terminons la description des modèles de spécification d'une application interactive.

Pour donner une dimension concrète à cette modélisation, nous allons maintenant décrire une architecture d'implémentation d'une application interactive ainsi spécifiée.

Chapitre 5 :

Architectures d'implémentation

Introduction

Après avoir proposé des modèles pour la conception et la spécification d'une application interactive, nous pouvons maintenant présenter les caractéristiques d'une architecture d'implémentation de ce type d'application qui comprend à la fois des modules particuliers à l'application et un système permettant la gestion de l'exécution de celle-ci. Une partie de ce système correspond d'ailleurs à un SGD.

Nous procéderons en trois points.

Premièrement, nous développerons les critères auxquels un telle architecture devrait répondre étant donné les concepts introduits dans la modélisation suggérée.

Ensuite, nous évoquerons deux propositions de gestionnaires de dialogue que nous connaissons pour les avoir utilisés personnellement. Il s'agit du SGD "DIALOGUE " développé par la firme Apollo Computer Inc. et du SGD "MOUSE "défini sur Macintosh par Joëlle Coutaz. Nous développerons leurs caractéristiques en fonction des critères décrits par la première étape. Notons que ces systèmes obéissent au principe de séparation entre les traitements et le dialogue de l'application.

Signalons déjà qu'un gestionnaire de dialogue est également souvent désigné sous le terme "dialogueur".

Enfin, nous exposerons une architecture d'implémentation qui puisse être contrôlée par un système de gestion que nous décomposons en trois éléments distincts : le gestionnaire des traitements, le dialogueur et un échangeur permettant la collaboration entre les deux premiers composants. Nous avons donné à ce système le nom d'architecture DET (Dialogueur, Echangeur, Traitements).

V.1 - Critères d'implémentation

Nous avons présenté au paragraphe I.3 le principe de séparation d'une application en une partie "Traitements " et une partie "Interface ". Compte tenu de cette notion, l'architecture d'implémentation de l'application, et en particulier le système de gestion sur lequel celle-ci peut s'appuyer, doit respecter certains critères concernant le niveau d'abstraction des objets manipulés, la localisation du contrôle de l'exécution, la nature et l'ordonnancement des événements pris en charge par les différents composants, l'adaptabilité de l'interface et la gestion du contexte.

La classification que nous proposons pour ces critères est inspirée de [Coutaz 86-3].

V.1.1 - Niveau d'abstraction des objets manipulés

Le niveau d'abstraction définit l'unité d'échange entre l'application et le SGD. Celle-ci correspondra de préférence à une entité de haut niveau et non à un événement de bas niveau.

Par exemple, un événement de bas niveau, tel que la pression du bouton de la souris dans la zône de fermeture d'une fenêtre contenant un message de saisie, peut être transformé par le dialogueur en un événement de haut niveau, tel qu'une requête de transfert d'un message interactif complet, avant d'être envoyé vers la partie "Traitements " de l'application. En effet, le clic-souris fermant une fenêtre signifie que l'utilisateur a terminé la saisie du message. Notons cependant que la définition des entités de haut niveau est propre à chaque dialogueur. L'exemple que nous avons donné est, quant à lui, basé sur le système de gestion d'application interactive DET présenté au paragraphe V.2.3.

Si le niveau d'abstraction d'un SGD était faible, les traitements indispensables à la mise à niveau des unités de dialogue avec les objets sémantiques de l'application se trouveraient vraisemblablement intégrés dans le code de l'application fonctionnelle, alors que celle-ci ne devrait s'occuper que de la mise en oeuvre des fonctions sémantiques de l'application.

V.1.2 - Localisation du contrôle

La localisation du contrôle précise si l'exécution de l'application est dirigée par l'application fonctionnelle, par le SGD, ou de façon partagée par ces deux composants. Ces trois possibilités sont respectivement dénommées par W. Buxton et P. Tanner contrôle interne, contrôle externe et contrôle mixte [Buxton, Tanner].

Nous présenterons successivement ces trois hypothèses d'implémentation.

V.1.2.1 - L'application fonctionnelle possède le contrôle

Dans cette première hypothèse, les traitements contrôlent l'exécution de l'application. Ils demandent au dialogueur de constituer les messages d'entrée de la phase à exécuter et, sur base de cette demande explicite, le dialogueur transmet aux traitements les messages introduits. Ensuite, l'application fonctionnelle déclenche les traitements nécessaires. C'est donc bien elle qui possède le contrôle de l'enchaînement lors de l'exécution des différentes fonctions de saisie et de traitement.

V.1.2.2 - Le dialogueur contrôle l'exécution

Ici, le dialogueur dispose d'une mémoire qui lui indique les messages qui sont à introduire lors de l'exécution de la phase.

Dans ce cas, la relation entre l'application fonctionnelle et le dialogueur se fait en sens inverse. Le dialogueur transmet d'abord aux traitements tous les messages introduits par l'utilisateur. Ce n'est que s'il lui manque un message que l'application fonctionnelle le demande au dialogueur.

Quand un message est complété (et confirmé), il faut que la fonction de validation sémantique correspondante soit lancée, si toutefois il en existe une. Dans ce cas, le dialogueur doit disposer d'un mécanisme qui lui permette de déterminer quelle fonction est à lancer. Il est alors capable de déclencher les traitements nécessaires.

Le dialogueur peut, par exemple, disposer d'une table de correspondance entre les messages interactifs et les fonctions à lancer. On peut imaginer que cette table soit construite automatiquement à partir de l'analyse des spécifications de l'application (définition des messages et schéma de la dynamique des traitements). Une autre optique serait de compléter la spécification des messages en leur ajoutant le nom de la fonction à lancer qui leur est associée, mais cette solution remet en cause l'indépendance entre l'application et le dialogue. En effet, elle impose d'adjoindre à la spécification des messages fonctionnels une information destinée au gestionnaire de l'interface de l'application.

Avec ce type de contrôle, un autre problème peut se poser. Les fonctions risquent de ne plus être que des serveurs par rapport aux messages. Ainsi, un message complété déclenchera un ou plusieurs traitements s'y rapportant. Dans ce cas, comment exprimer l'enchaînement de plusieurs fonctions dont seule la première est démarrée à la fin de la saisie d'un certain message, les autres nécessitant des résultats intermédiaires produits par cette première fonction ? Lors de l'étape d'implémentation de l'application, le programmeur devra regrouper, à l'intérieur d'une seule procédure de traitement, les différentes fonctions qui doivent être exécutées selon une certaine structure d'enchaînement.

La découpe en fonctions, réalisée lors de l'étape de conception de l'application, ne transparaîtra donc plus dans l'architecture logicielle. Ceci n'empêche toutefois pas que la saisie d'un message puisse déclencher plusieurs fonctions (procédures) en parallèle, si celles-ci lui sont directement associées.

En procédant ainsi, on force un style de programmation très particulier et assez contraignant qui risque de déteindre sur l'étape de conception, si le programmeur et le concepteur de l'application interactive sont une seule et même personne. En effet, le programmeur habitué à un tel système aura tendance, même lors de la modélisation de l'application, à ne plus spécifier, pour chaque message saisi, qu'une ou plusieurs fonctions, correspondant chacune à une procédure, déclenchées directement par ce message, oubliant ainsi les critères de découpe hiérarchique des traitements imposés par le modèle qu'il utilise.

V.1.2.3 - Le dialogueur et l'application partagent le contrôle

Dans cette hypothèse de contrôle, il s'agit de laisser chaque composant diriger l'exécution de sa partie. Ainsi, le dialogueur s'occupe de la gestion de la saisie des messages interactifs et de leur enchaînement, si le concepteur de dialogue en a déterminé un. Quant à l'application fonctionnelle, elle est chargée de déclencher les traitements correspondant aux messages qu'elle reçoit. Lorsqu'il lui manque un message, l'application le demande à l'utilisateur par l'intermédiaire du dialogueur. Et lorsque l'utilisateur a complété un message, le dialogueur le transmet à l'application qui, elle, est capable de l'identifier et de déclencher les traitements adéquats.

Nous verrons au paragraphe V.2.3 comment ce style de contrôle sera implémenté dans le système de gestion d'application interactive DET.

V.1.3 - La nature et l'ordonnancement des événements reconnus par l'application

Il faut bien distinguer la notion d'événement, présentée au paragraphe II.4.1, qui intervient dans la dynamique d'enchaînement des traitements et qui a donc un sens pour l'application fonctionnelle de la notion d'événement provoqué par l'utilisateur. Comme nous l'avons expliqué au paragraphe V.1.1, ce dernier peut aller de l'événement de bas niveau comme la frappe d'une touche du clavier ou la pression de la souris jusqu'à la notion d'événement dit abstrait (de haut niveau) qui enrichit la sémantique d'un événement de bas niveau pour atteindre une sémantique significative pour l'application.

Seul le dialogueur reconnait et manipule les événements de bas niveau. Ceci se fait a priori sans figer l'ordonnancement de ce type d'événement. C'est également le dialogueur qui identifie les événements abstraits, sur base de l'information qui enrichit les événements de bas niveau qui en sont l'origine. Cette information dépend elle-même du contexte dans lequel ces derniers sont intervenus. L'ordonnancement de ce type d'événement n'est normalement pas figé, sauf éventuellement pour permettre une gestion de contexte. En effet, suivant l'état dans lequel se trouve l'application, certaines options d'un menu ou certaines actions peuvent ou non être

effectuées par l'utilisateur. Les événements qui permettent d'y accéder doivent donc être rendus valides ou invalides, suivant le cas. Nous reparlerons de cette notion de contexte au paragraphe V.1.5.

V.1.4 - L'adaptabilité

"L'adaptabilité est la faculté d'ajustement à la nouveauté d'une situation ou aux variations de l'environnement. " [Coutaz 86-3]

Au niveau de l'implémentation, "des concepts et techniques logicielles ont été définis pour la portabilité et la réutilisation de logiciels indépendamment des terminaux, des systèmes d'exploitation et des langages. " [Coutaz 86-3]. Mais, en ce qui concerne la capacité de l'interface à s'adapter à l'utilisateur, peu d'études ont été menées à ce jour, et encore moins de résultats concrets ont déjà été obtenus.

V.1.5 - Le contexte

" Un contexte est une structure d'éléments déterminant le sens d'une situation. ... En interface homme-machine, ce concept est reconnu comme nécessaire mais reste mal intégré aux outils. " [Coutaz 86-3]

Nous avons déjà explicité cette notion de contexte au paragraphe I.2.1 lors de la description des qualités d'un bon dialogue. Nous avions alors exposé le modèle Site-Mode-Trail proposé par Nievergelt, qui permet de classifier la gestion du contexte d'une application en trois parties, la première concernant les données courantes, la seconde détaillant les commandes courantes, et la dernière s'occupant de la gestion de l'historique du dialogue.

Après avoir exposé les critères selon lesquels on peut juger de la qualité d'une architecture d'implémentation incluant un SGD et permettant de gérer une application interactive, nous présentons maintenant deux gestionnaires de dialogue afin d'illustrer la façon dont ces caractéristiques peuvent être prises en compte.

V.2 - Le dialogueur

Pour rappel, nous avons défini au paragraphe III.2 les fonctions suivantes pour le modèle de dialogue :

- saisie et affichage;
- aide;
- déclenchement d'un traitement;
- interruption et reprise;
- annulation;
- retour-arrière;
- gestion du contexte.

Ces fonctions devraient être réalisées par tout SGD.

Dans le SGD " DIALOGUE " développé sur Apollo et dans le SGD " MOUSE " basé sur les deux modèles d'architecture logicielle pour applications interactives APEX et PAC [Coutaz 86-4], comment ces fonctions ont-elles été implémentées (totalement, partiellement ou pas du tout) et comment les caractéristiques présentées au paragraphe précédent sont-elles respectées?

V.2.1 - "DIALOGUE"

DIALOGUE est un système de gestion de dialogue implanté sur une station de travail Apollo. Nous allons analyser ses caractéristiques selon les différents critères que nous avons détaillés au paragraphe précédent. L'annexe 1 sera dédiée à une étude plus générale de ce logiciel. Le lecteur désirant de plus amples renseignements à propos des termes écrits en italique dans ce paragraphe pourra se référer à cette annexe.

5.2.1.1 - Niveau d'abstraction

Les unités échangées entre le SGD et l'application sont d'un niveau d'abstraction élevé. En effet, les événements de bas niveau tels que le clic-souris dans une région de l'écran sont transformés par le dialogueur (par l'intermédiaire des *tâches*) en ordres d'action que l'application fonctionnelle doit effectuer. Ces actions peuvent être par exemple l'appel d'une routine des traitements ou le changement d'état du contexte.

5.2.1.2 - Localisation du contrôle

C'est le SGD qui contrôle l'exécution des applications implémentées à l'aide de ce logiciel. La description de l'interface que le programmeur fournit à ce système permet au SGD de connaître toutes les données à saisir dans le cadre d'une application. Cette description indique également quelles sont les fonctions à lancer lors de l'introduction par l'utilisateur d'une donnée ou d'un groupe de données. Avec ce type de contrôle, les traitements ne sont plus que des serveurs par rapport aux messages saisis par le dialogueur; nous avons déjà soulevé ce problème.

Dans ce SGD, il n'existe pas de description de l'enchaînement des traitements. Cet enchaînement dynamique dépend alors exclusivement de l'ordre dans lequel l'utilisateur effectue la saisie des données.

C'est donc le dialogueur qui possède le contrôle. Il peut cependant être abandonné au profit des traitements pour un certain temps si ceux-ci ont une tâche particulière à effectuer.

5.2.1.3 - L'ordonnancement des événements

L'ordonnancement des événements utilisateurs reconnus par l'application interactive, c'est-à-dire des ordres reçus par l'application fonctionnelle tels que le déclenchement d'un traitement ou l'abandon du contrôle est effectué à l'aide d'une listeFIFO.

V.2.1.4 - L'adaptabilité

La présentation d'une application peut être modifiée sans devoir recompiler les traitements de celle-ci. Cela permet au concepteur de dialogue de procéder par itérations successives pour définir une interface qui satisfasse les utilisateurs de l'application.

Cependant, ceci ne peut se faire que lors de l'étape de conception de l'interface. En effet, la routine d'initialisation du logiciel DIALOGUE qui permet de modifier la présentation à l'écran (les *techniques*) des objets du dialogue sans recompiler l'application n'effectue pas réellement d'édition des liens entre les traitements et le fichier contenant la description du dialogue. Cette procédure d'initialisation associe automatiquement au programme que l'on veut exécuter le dernier fichier de spécification de dialogue qui a été traduit au *translateur* DIALOGUE. L'utilisateur risque alors de voir apparaître à l'écran une interface qui n'a rien à voir avec l'application qu'il tente d'exécuter.

Pour remédier à ce problème, il existe une autre procédure d'initialisation du logiciel DIALOGUE qui, elle, permet réellement l'édition des liens entre l'application fonctionnelle et le dialogue qui lui correspond, mais qui impose la recompilation de toute l'application lors d'un changement, même minime, de la présentation des messages.

L'adaptabilité de l'interface n'existe donc que lors de l'étape de conception du dialogue. De plus, puisqu'il est nécessaire d'effectuer, dans la partie concernant les traitements, une édition des liens entre une application fonctionnelle et l'interface qui lui correspond, il n'est pas possible de permettre à l'application de choisir entre différents types de présentation suivant le type d'utilisateur pour lequel elle doit s'exécuter.

V.2.1.5 - Le contexte

Dans le cadre du logiciel DIALOGUE, une gestion de contexte est effectuée pour les commandes courantes. Nous avons déjà dit que chaque événement provoqué par l'utilisateur peut déclencher certaines actions. Parmi ces actions, on

trouve l'activation ou la désactivation de certains objets du dialogue, c'est-à-dire que, suivant les actions que l'utilisateur a déjà effectuées, il pourra ou non déclencher d'autres tâches. Reprenons notre exemple de mutuelle. Le concepteur de dialogue peut désirer obliger l'utilisateur du poste de travail à introduire le nom de l'individu qui a subi les soins avant de pouvoir encoder son prénom. Il suffit de spécifier (dans la description du dialogue) que le champ "prénom-individu " est inactif (l'utilisateur ne peut effectuer aucune action dessus) et qu'il doit être activé lorsque l'utilisateur a introduit un "nom- individu " dans le champ correspondant.

Nous avons passé en revue les différents critères qui permettent de juger de la qualité d'un SGD en les adaptant au logiciel DIALOGUE. Nous allons maintenant procéder de la même façon pour étudier le SGD MOUSE.

V.2.2 - "MOUSE"

Avant de présenter ce système de gestion de dialogue, explicitons quelque peu les deux modèles, APEX et PAC, sur lesquels il s'appuye.

Ces deux modèles nous serviront également à expliquer l'architecture d'implémentation que nous proposons au paragraphe V.2.3.

Les modèles APEX et PAC ainsi que le SGD MOUSE représentent le résultat des recherches menées par Joëlle Coutaz dans le domaine de la construction d'interfaces homme-machine.

Signalons encore que le modèle APEX a notamment été appliqué à la réalisation d'un logiciel de tracé de figures géométriques [Lermigeaux]. Cette expérience a donné lieu à une implémentation complète du modèle dont l'essentiel a alors été complété et corrigé lors de la programmation d'une autre application, développée durant le stage effectué dans le cadre de ce mémoire. Une présentation des principaux résultats obtenus est détaillée dans l'annexe 2. Celle-ci contient, entre autres choses, la description et le code des principaux modules réutilisables du noyau d'APEX.

V.2.2.1 - "APEX" (APplication EXtensible)

"APEX définit une architecture logicielle qui obéit au principe fondamental de séparation modulaire entre fonctionnalités et interface, et qui intègre le logiciel commun à toute application interactive." [Coutaz 86-3]

La structure d'APEX répond en fait à des observations selon lesquelles :

- certains traitements, par exemple les séquences d'initialisation et de clôture du dialogue ou le raffraîchissement des fenêtres, sont systématiquement présents dans chaque programme, et ceci quelles que soient les fonctionnalités de l'application implémentée;

- " chaque programme s'organise autour d'une boucle "tant que" de la forme :

  tant que pas-fini **faire**
  acquérir-événement;
  traiter-événement;
  fin tant que; " [Coutaz 86-3];

  dans laquelle "événement" correspond à la notion d'événement utilisateur de bas niveau.

  Tout programme construit autour de cette boucle d'acquisition et de traitement d'événements est soumis à un modèle de contrôle essentiellement basé sur l'écoute des événements engendrés par l'utilisateur. Ce dernier est donc le "véritable pilote de l'application"[Coutaz 86-3].

L'architecture logicielle d'APEX, présentée à la figure 27, est constituée de deux composants : APPLICATION et INTERFACE.

INTERFACE correspond au logiciel de gestion d'interface commun à toute application interactive. Il implémente la syntaxe du dialogue.

APPLICATION rassemble les modules d'implémentation des traitements spécifiques à l'application.

Les relations entre ces deux composants sont réglées par un protocole d'échange de type procédural : le contrôle du dialogue est maintenu dans l'interface qui appelle, si besoin est, les services de l'application. Cette dernière est donc assimilée à une librairie de procédures.

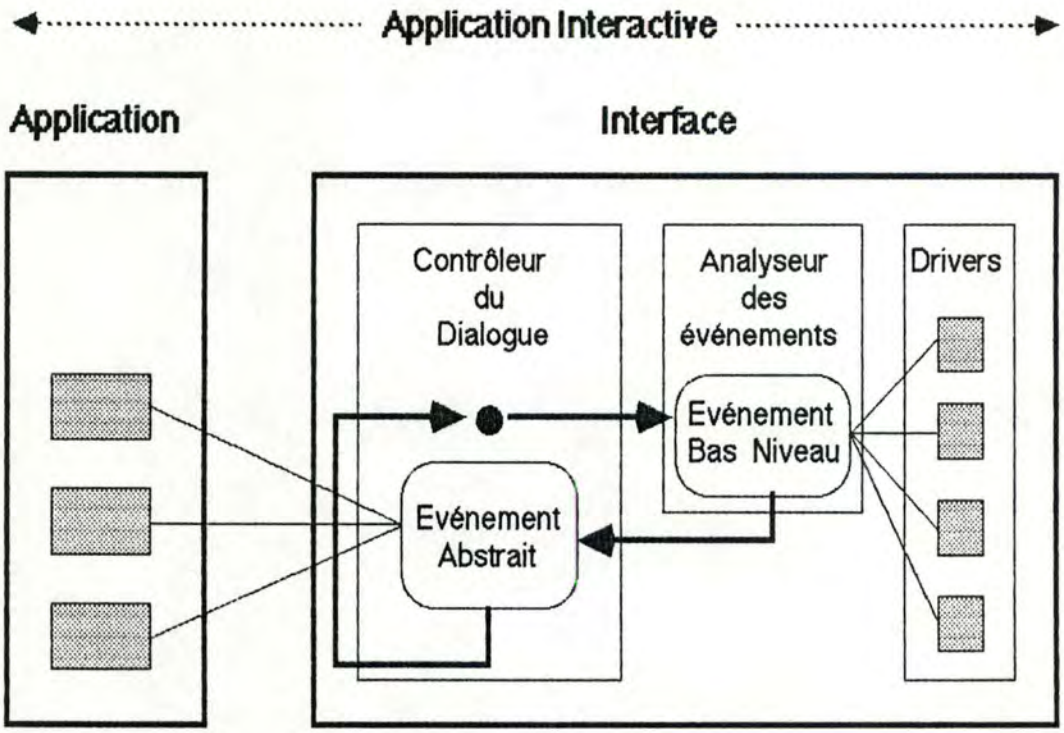


Figure 27 - Le modèle d'Application Extensible

Dans cette figure, les flêches visualisent la boucle de traitement des événements qui définit le contrôle du dialogue; les traits fins correspondent à des appels procéduraux; le point indique le point d'entrée du cycle de traitement des événements.

INTERFACE, qui constitue le noyau du modèle APEX, est lui-même structuré en plusieurs modules : le contrôleur du dialogue, l'analyseur des événements et un ensemble de serveurs, souvent désignés sous le terme "drivers", spécialisés dans la gestion d'un type particulier d'événement ou d'objet du dialogue.

Le contrôleur du dialogue est à l'écoute des événements de bas niveau générés principalement par l'utilisateur ou éventuellement par les fonctions de l'application (cfr paragraphes V.2.2.3.2 et V.2.2.3.3). Le contrôleur transmet ces événements à l'analyseur qui les enrichit et les rend au contrôleur sous forme d'événements abstraits. Quand un événement abstrait concerne l'application fonctionnelle, il reçoit de l'analyseur un code particulier qui permet au contrôleur d'identifier la procédure de l'APPLICATION qui doit être exécutée. Le contrôleur communique cet identificateur à l'application qui peut alors lancer l'exécution de la procédure afin d'effectuer le traitement sémantique lié à la réception de cet événement abstrait.

L'analyseur des événements détermine la nature de l'événement de bas niveau reçu par le contrôleur du dialogue et appelle le driver spécialisé dans la gestion de ce type d'événement si cela s'avère nécessaire.

Un driver interprète l'événement d'entrée et effectue tous les traitements indépendants de la sémantique de l'application.

Lorsqu'un événement a un effet de bord sémantique, c'est-à-dire lorsqu'un événement nécessite un traitement de la part de l'application, le serveur l'indique dans l'événement abstrait en enrichissant l'événement initial au moyen d'une information qui désigne un point d'entrée dans APPLICATION que pourra reconnaître le contrôleur.

Pour illustrer le mécanisme de contrôle du dialogue, détaillons la prise en charge d'un événement par les différents modules.

Supposons que l'utilisateur presse puis relâche le bouton de la souris alors que le suiveur de la souris se trouve dans la région du contenu d'une fenêtre.

L'événement de bas niveau, constitué par la pression du bouton de la souris par l'utilisateur, est pris en charge par le contrôleur du dialogue au point d'entrée illustré à la figure 27. Celui-ci correspond en fait à la mémorisation des événements dans une file d'attente utilisée par le système pour enregistrer tous les événements

de bas niveau qui surviennent, et ceci dans leur ordre d'arrivée. Ces événements sont extraits de la file d'attente dans un ordre FIFO.

Quand l'événement dont nous suivons le parcours est extrait de la file d'attente, il est envoyé à l'analyseur des événements. Celui-ci l'identifie comme un événement de type "MOUSEDOWN", localisé dans une fenêtre. Il appelle donc le driver-interpréteur des événements concernant les fenêtres afin que ce dernier effectue les traitements liés à l'action de l'utilisateur sur la fenêtre.

Le gestionnaire (driver) de fenêtre constate alors par une analyse plus fine de l'événement initial que celui-ci a eu lieu dans la région du contenu d'une fenêtre. Si celle-ci est la fenêtre courante (active), l'action à effectuer dans cette région dépend des fonctionnalités de l'application spécifiées pour cette fenêtre et non pas d'un traitement de manipulation de la fenêtre par le driver (modification de la taille, défilement du contenu, fermeture) (cfr paragraphe III.1.2.1).

Le driver se contente donc d'enrichir la sémantique de l'événement initial "MOUSEDOWN" par le code "INCONTENT" signifiant ainsi que la souris a été pressée à l'intérieur de la zône de contenu de la fenêtre active.

L'événement abstrait défini de la sorte est ensuite renvoyé à l'analyseur qui le transmet lui-même au contrôleur du dialogue.

Le code "INCONTENT" qui complète alors l'événement reçu par le contrôleur permet à celui-ci de déclencher la procédure de APPLICATION qui déterminera et lancera le traitement correspondant à la gestion du contenu de la fenêtre à l'intérieur de laquelle l'utilisateur a pressé la souris.

Les composants d'INTERFACE ont été implémentés sur Macintosh en utilisant le Toolbox que cette machine fournit. A ce propos, nous pouvons constater deux choses.

D'une part, toute application développée dans cet environnement peut être construite autour de la boucle d'événement identifiée dans l'architecture d'APEX, car le Toolbox a été conçu sur base de la notion d'événement.

D'autre part, cet outil a pu être utilisé pour développer un ensemble de serveurs (drivers) de plus haut niveau que ceux déjà proposés par le Toolbox tels que,par exemple, des gestionnaires évolués d'événements, de fenêtres, ou de menus, qui ont alors été utilisés comme modules de base pour les composants d'INTERFACE.

V.2.2.2 - "PAC" (Présentation-Abstraction-Contrôle)

" PAC est un raffinement d'APEX. Il structure l'architecture d'une application interactive en trois composants : Présentation, Abstraction et Contrôle.

La Présentation définit le comportement de l'application vis-à-vis de l'utilisateur, l'Abstraction correspond aux fonctionnalités de l'application, et le Contrôle maintient la cohérence entre la Présentation et l'Abstraction.

La Présentation est à son tour réalisée par un ensemble d'objets interactifs à-la-Smalltalk spécialisés dans la communication homme-machine. Tout objet interactif adhère également au modèle PAC. PAC est donc un modèle récursif applicable à tous les niveaux d'abstraction d'une application interactive." [Coutaz 86-3]

La figure 28 montre la relation qui existe entre APEX et PAC en affinant le schéma du modèle APEX présenté à la figure 27.

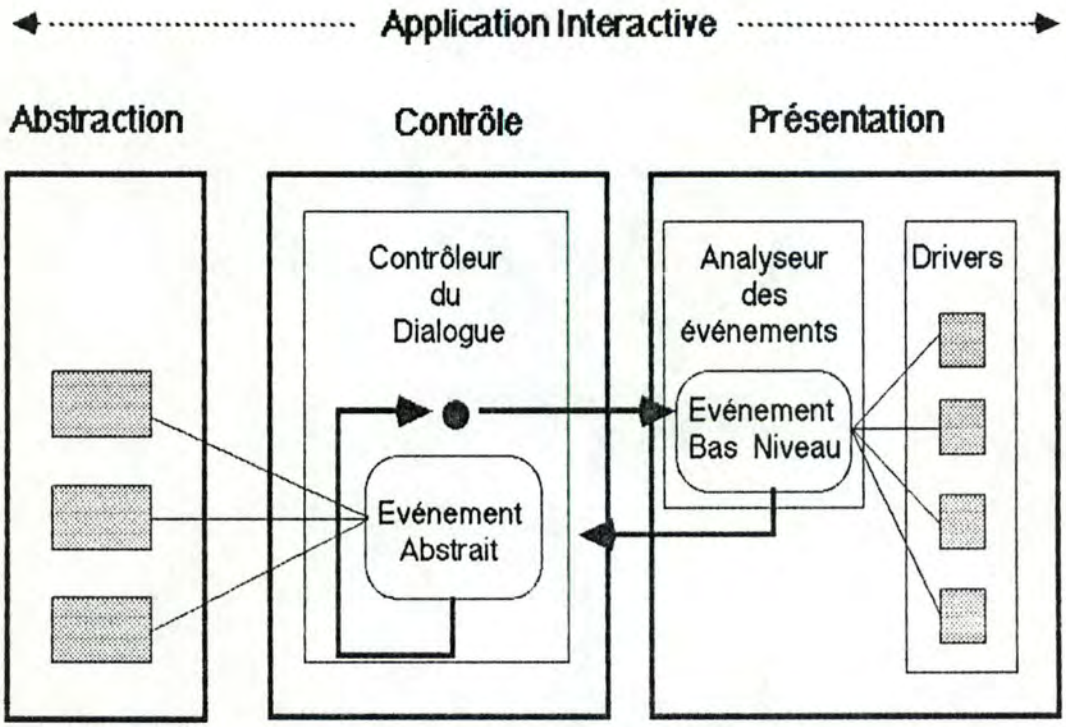


Figure 28 - Le modèle d'Application Extensible est un modèle PAC non hiérarchique

La partie APPLICATION de l'architecture APEX correspond au composant Abstraction du modèle PAC.

L'INTERFACE d'APEX regroupe le Contrôle et la Présentation de PAC. A l'intérieur de cette partie INTERFACE, le Controleur du dialogue de la structure logicielle d'APEX réalise le Contrôle PAC tandis que les drivers en définissent la Présentation.

V.2.2.3 - "MOUSE" (Management Of the User through Specifications)

Nous ne détaillerons pas ici la structure de ce système de gestion de dialogue car cela s'avérerait un peu long. Une description précise de MOUSE est donnée dans [Coutaz 86-3].

Nous soulignons donc uniquement les caractéristiques de MOUSE compte tenu des critères que nous avons fournis en la matière au paragraphe V.1.

Le système de gestion de dialogue MOUSE présente les caractéristiques détaillées ci-dessous.

V.2.2.3.1 - Niveau d'abstraction

Le niveau d'abstraction des échanges entre le SGD et l'application est déterminé par le concepteur. Il concerne donc des entités sémantiques significatives pour l'application.

V.2.2.3.2 - Localisation du contrôle

Le contrôle est essentiellement externe mais peut être considéré comme mixte du point de vue de l'application. En effet, l'application "ne peut faire préemption sur l'utilisateur mais a la possibilité d'exprimer l'arrivée d'événements asynchrones", c'est-à-dire ne correspondant pas à un appel procédural, "et formuler une demande d'intervention de l'utilisateur..." [Coutaz 86-3].

Ainsi, l'application peut définir ses propres types d'événements qui seront pris en charge par le contrôleur du dialogue et l'analyseur des événements de la même façon que les événements utilisateur de bas niveau qui arrivent à l'interface.

V.2.2.3.3 - L'ordonnancement des événements

En ce qui concerne l'ordonnancement des événements, MOUSE est "event-driven". L'ordonnancement se fait uniquement au niveau des événements de bas niveau, les événements définis par l'application étant assimilés à ces événements de bas niveau par le contrôleur du dialogue et l'analyseur des événements.

Aucun système de priorité n'est prévu; la prise en charge et l'analyse des événements se fait selon l'ordre FIFO (First In, First Out) d'extraction des événements de la file d'attente dans laquelle ils ont été mémorisés à leur survenance.

V.2.2.3.4 - L'adaptabilité

L'interface est adaptable sur intervention explicite du concepteur ou de l'utilisateur. Toute adaptation nécessite cependant la recompilation de la spécification de l'interface intégrant les nouvelles conditions et caractéristiques souhaitées.

V.2.2.3.5 - Le contexte

"La gestion du contexte est élémentaire avec la notion de validité de commandes et de paramètres, avec un "refaire" et un "défaire" à un niveau de profondeur, et une aide au niveau de l'objet." [Coutaz 86-3]

Le contexte se limite donc à définir les commandes et paramètres "acceptables" à un moment donné de l'exécution de l'application, à permettre l'annulation de la dernière action effectuée, et à fournir des messages d'aide pour la manipulation d'un objet.

Après avoir illustré l'outil "Dialogueur" à partir de deux SGD existants, nous pouvons maintenant expliciter l'architecture d'implémentation d'une application interactive que nous souhaiterions développer sur base de la modélisation que nous avons présentée au cours des chapitres 2, 3 et 4.

V.2.3 - Une architecture d'implémentation

V.2.3.1 - L'architecture d'implémentation DET (Dialogue, Echange, Traitements)

L'architecture d'implémentation d'une application interactive que nous proposons est illustrée par la figure 29.

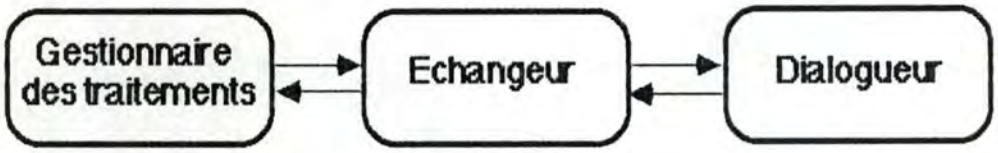


Figure 29 - L'architecture d'implémentation DET

La définition d'un échangeur, composant indépendant au sein de l'architecture logicielle de l'application, correspond en fait à la spécification d'un module qui implémente la structure d'échange définie au chapitre 4 dans le modèle de l'échange.

Les fonctions implémentées sont donc celles décrites au paragraphe IV.2, à savoir :

- l'analyse et l'adaptation des requêtes à communiquer du dialogueur à l'application et inversément, ce qui comprend entre autre la mise en correspondance des paramètres fonctionnels de ces requêtes,
- et la gestion des conditions de transmission.

Compte tenu de cette spécification de l'échangeur, le gestionnaire des traitements et le dialogueur s'occupent du contrôle de l'exécution respectivement des traitements et de l'interface de l'application.

La figure 30 illustre le cadre d'exécution de l'application en détaillant les informations utilisées par les différents composants de DET pour permettre l'exécution de celle-ci.

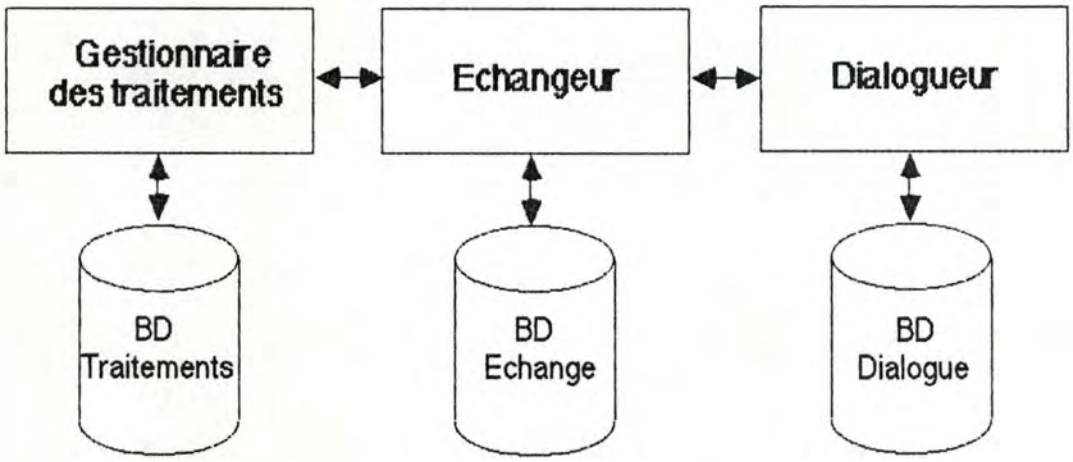


Figure 30 - L'architecture d'implémentation DET à l'exécution de l'application

BD Traitements contient la base de spécification de l'application fonctionnelle (structuration des informations, statique et dynamique des traitements,...) ainsi que la base de données de l'application.

BD Dialogue contient la spécification des messages et des objets interactifs ainsi que les schéma de la dynamique du dialogue (Conversation).

BD Echange contient une spécification de la correspondance champ par champ des messages-externes-fonction et des messages fonctionnels interactifs. Cette spécification est celle que nous avons décrite lors de la présentation du modèle de l'échange.

Au moment de la sélection d'une phase par l'utilisateur, le dialogueur est chargé d'identifier le traitement à déclencher et d'envoyer une requête à l'application fonctionnelle, via l'échangeur, pour demander l'activation de ce traitement. Il attend alors le signal de déclenchement de la phase qui lui est envoyé en retour.

Le dialogueur connait la notion de phase. Ainsi, dès qu'il identifie la phase sélectionnée par l'utilisateur, il peut déterminer les messages interactifs qu'il devra prendre en charge au cours de l'exécution de cette dernière.

Le gestionnaire doit enfin prévoir la gestion du contexte mais celle-ci peut être effectuée de manière plus ou moins fine selon les exigences qui veulent être retenues en la matière (cfr paragraphe V.2.3.5.5).

Le gestionnaire des traitements a accès à l'ensemble de la spécification des informations et des traitements de l'application fonctionnelle. A la terminaison d'une fonction, il est chargé de communiquer à l'échangeur les messages externes de sortie (s'il en existe), qui doivent être transmis à l'interface pour être présentés à l'utilisateur. Il a également la tâche de déterminer la ou les fonctions suivantes à exécuter et de les déclencher.

Tant que le dialogueur ne transmet pas au gestionnaire des traitements une requête "activer le traitement P", le gestionnaire des traitements est en attente d'une telle commande.

Quand il la reçoit, il vérifie si le traitement P peut être déclenché et, en fonction de cela, il transmet au dialogueur une réponse positive ou négative d'activation grâce à la requête "accepter (refuser) le déclenchement du traitement P".

Quand l'exécution d'une phase est terminée, le gestionnaire des traitements doit le signaler au dialogueur en lui transmettant une requête "signaler la terminaison du traitement P", via l'échangeur.

Si l'exécution de la phase est suspendue par manque d'informations en provenance de l'utilisateur (messages-externes-fonction), il demande ces données à l'interface via la requête "requérir message Ti".

Le gestionnaire des traitements a également une autre tâche importante : celle de gérer les retours-arrière.

En effet, en cas d'invalidation sémantique d'un ou plusieurs champs d'un message-externe-fonction, il doit retourner, dans l'enchaînement dynamique des fonctions, aux différents messages invalidés par un traitement. Lors d'un retour-arrière demandé par l'utilisateur, il doit revenir au point de retour-arrière effectivement utilisé par l'utilisateur. Ce point lui est indiqué par le dialogueur via la requête "retourner au message Di".

Enfin, c'est le gestionnaire des traitements qui s'occupe de la gestion de la base de données et des ressources de l'application.

V.2.3.2 - Les composants du dialogueur

Le dialogueur présenté ici s'inspire de l'architecture logicielle d'application proposée par les modèles APEX et PAC (cfr paragraphes V.2.2.1 et V.2.2.2), mais il y apporte certaines modifications et des ajouts que nous avons introduits pour répondre aux caractéristiques que nous attendons d'un SGD et pour respecter la démarche de modélisation d'une application que nous avons décrite précédemment.

La structure générale du composant INTERFACE, basée sur une décomposition fonctionnelle en trois modules (Contrôleur du dialogue, Analyseur des événements et Drivers), peut être récupérée et exploitée ici car elle reste valide indépendamment des caractéristiques que nous avons spécifiées. Il faut cependant ajouter à cela un quatrième composant qui sera chargé de manipuler les messages interactifs, c'est-à-dire, notamment, d'en gérer l'enchaînement dynamique à l'écran.

Notons que les modifications apportées aux trois composants d'INTERFACE initialement définis concernent surtout les modules "Contrôleur du dialogue " et "Analyseur des événements ". Ces modifications sont présentées ci-dessous.

Le Contrôleur du dialogue doit essentiellement être redéfini pour prendre en compte la communication par requête avec l'application fonctionnelle via l'échangeur.

L'analyseur des événements doit, lui, pouvoir distinguer et manipuler différents types d'événement, allant des événements de bas niveau aux événements abstraits ayant ou non une sémantique pour l'application fonctionnelle.

Un clic-souris ou la frappe d'une touche du clavier sont des exemples d'événement de bas niveau. Chacun de ces événement de bas niveau peut conduire à différents types d'événement abstrait en fonction de son contexte de survenance, de sa localisation,...

Ainsi, quand l'utilisateur frappe la touche HELP du clavier alors que le curseur se trouve positionné sur un objet interactif, cette action est transformée par le dialogueur en un événement abstrait du type "Demande d'aide au sujet de tel objet interactif ". Le dialogueur est chargé de traiter l'événement en accédant à la spécification de l'objet concerné et en vérifiant s'il lui est associé un message d'aide. Dans le cas où un tel texte existe, le dialogueur l'affiche à l'écran. Un événement de ce type est donc totalement géré par le dialogueur.

Par contre, dans le cas où l'utilisateur confirme les informations contenues dans un message interactif de saisie, par exemple en fermant la fenêtre dans laquelle est présenté le message, l'événement de bas niveau "clic-souris dans la zône de fermeture d'une fenêtre " doit être transformé en un événement abstrait destiné à l'application fonctionnelle, et qui correspond à une requête d'envoi du message vers le gestionnaire des traitements, par l'intermédiaire de l'échangeur.

Nous pouvons donc distinguer deux types d'événements abstraits. Les premiers concernent seulement les objets et messages exclusivement interactifs uniquement destinés à l'interface de l'application; ils ne sont donc pas traités par d'autres modules de gestion que les composants du dialogueur (cfr figure 31). Les événements abstraits du second type correspondent à la manipulation de messages fonctionnels interactifs ou de messages exclusivement interactifs qui peuvent avoir un impact sur les traitements de l'application (demande de retour-arrière, activation d'un traitement,...).

Enfin, les modules "driver" qui ont été construits comme éléments indépendants de gestion des objets du dialogue (menus, fenêtres,...), ne doivent subir aucune adaptation particulière par rapport à la façon dont ils ont été spécifiés dans APEX et PAC.

IV.2.3.3 - Les composants de l'échangeur

Compte tenu des fonctions que l'échangeur doit implémenter, on peut distinguer différents composants au sein de l'architecture de celui-ci : le contrôleur de l'échange, l'analyseur des requêtes et le "mapper".

Le contrôleur de l'échange gère le transfert des requêtes entre le gestionnaire des traitements et le dialogueur à l'aide d'une file d'attente permettant d'enregistrer les requêtes qui lui sont communiquées par l'un de ces deux composants, en leur attribuant un numéro d'arrivée et éventuellement un niveau de priorité.

L'analyseur des requêtes étudie les demandes qui ont été transmises au contrôleur et détermine la prochaine requête à prendre en charge par l'échangeur, d'après leur type, leur ordre d'arrivée dans la file d'attente et leur niveau de priorité. Les considérations développées au paragraphe IV.2.3.2 concernant l'ordre de prise en compte des requêtes par l'échangeur doivent notamment être respectées par l'analyseur des événements. Par exemple, une requête "requérir message Ti" peut ne pas être transmise au dialogueur si une requête "communiquer message Dj", où Dj est un message interactif de saisie associé au message fonctionnel Ti, lui est transmise par le mapper en vue de sa transmission au gestionnaire des traitements.

Le "mapper" effectue l'adaptation de la requête courante qui lui est fournie par l'analyseur des requêtes conformément aux règles de transformation qui ont été définies dans le modèle de l'échange. Entre autre, cela nécessite la mise en correspondance des paramètres fonctionnels (messages-externes-fonction, messages interactifs) des requêtes transmises par l'analyseur des requêtes.

Après transformation complète d'une requête et de ses paramètres, le mapper la renvoit à l'analyseur des requêtes afin que ce dernier la transmette au contrôleur des requêtes qui la communiquera alors au composant destinataire (gestionnaire des traitements ou dialogueur). Par "transformation complète", nous voulons souligner le fait qu'il n'y a pas nécessairement de correspondance bi-univoque entre une requête émise par un des deux composants de l'application et la requête adaptée et transmise par l'échangeur au composant destinataire.

Par exemple, plusieurs requêtes "communiquer message Di" peuvent être nécessaires au mapper pour construire une requête "réceptionner message Tj" si le message Tj est présenté à l'utilisateur via plusieurs messages interactifs. Cette question a déjà été abordée au paragraphe IV.2.3.2.

IV.2.3.4 - Les composants du gestionnaire des traitements

De manière similaire aux deux autres composants de cette architecture, le gestionnaire des traitements sera formé d'un contrôleur, d'un analyseur, d'un moteur de la dynamique et d'un ensemble de fonctions.

Le contrôleur sera chargé de réceptionner dans une file d'attente les requêtes en provenance de l'échangeur, et d'envoyer vers celui-ci les requêtes destinées au dialogueur.

L'analyseur déterminera le type de chaque requête et choisira la prochaine requête à prendre en charge par le moteur des traitements d'après leur type, leur ordre d'arrivée dans la file d'attente et leur niveau de priorité.

La tâche du moteur de la dynamique des traitements est de gérer l'enchaînement des traitements, d'après les messages-externes-fonction en provenance de l'échangeur, et en fonction de requêtes telles que le déclenchement d'une phase, le retour-arrière,...

La figure 31 montre bien les différents composants de chacun des trois modules de l'architecture DET.

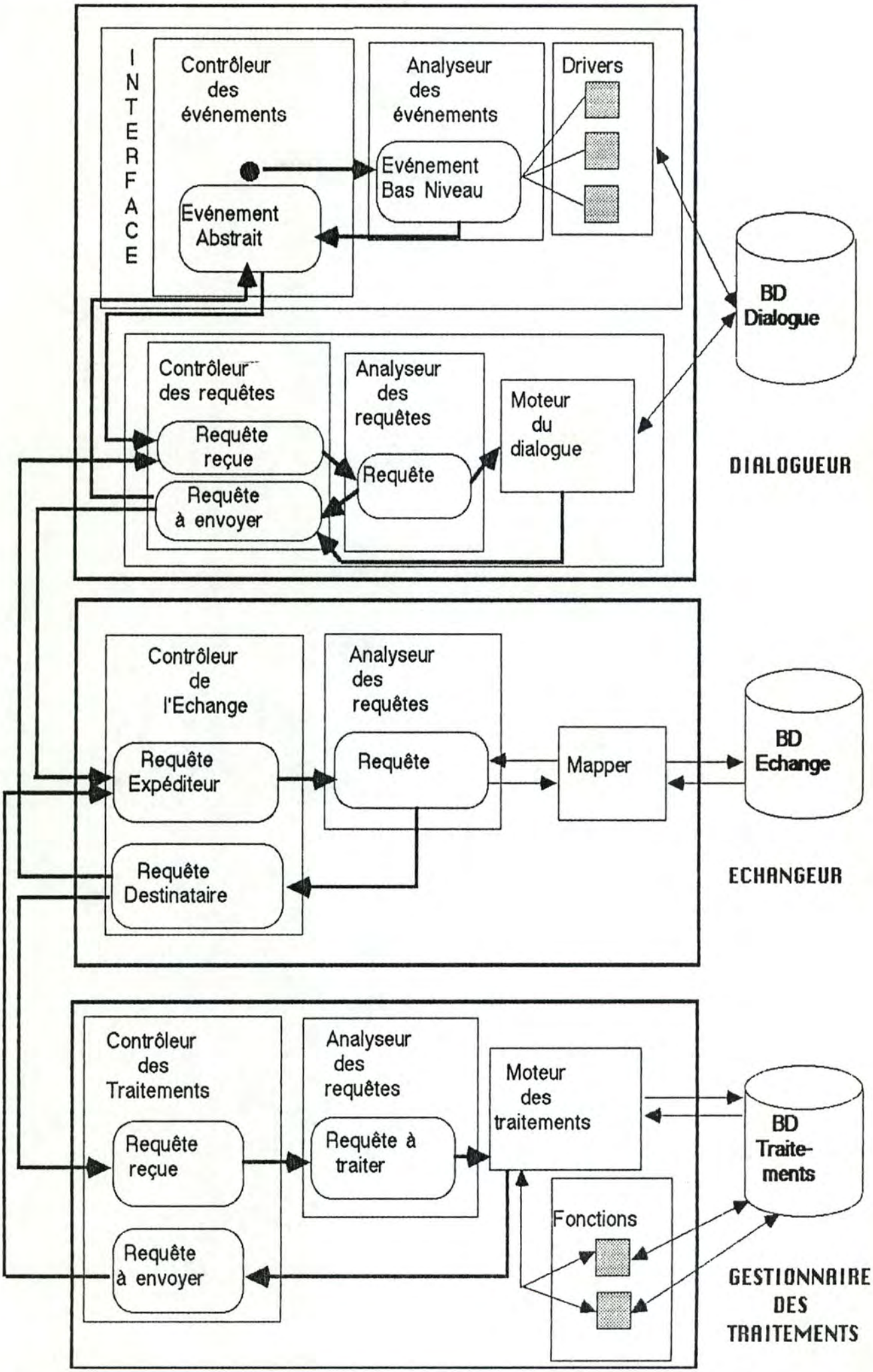

INTERFACE
Contrôleur des événements
Evénement Abstrait
Analyseur des événements
Evénement Bas Niveau
Drivers
BD Dialogue
Contrôleur des requêtes
Requête reçue
Requête à envoyer
Analyseur des requêtes
Requête
Moteur du dialogue
DIALOGUEUR
Contrôleur de l'Echange
Requête Expéditeur
Requête Destinataire
Analyseur des requêtes
Requête
Mapper
BD Echange
ECHANGEUR
Contrôleur des Traitements
Requête reçue
Requête à envoyer
Analyseur des requêtes
Requête à traiter
Moteur des traitements
Fonctions
BD Traite-ments
GESTIONNAIRE DES TRAITEMENTS

Figure 31 - Les composants de l'architecture d'implémentation DET.

Pour faciliter la compréhension du fonctionnement de cette architecture d'implémentation au moment de l'exécution d'une application interactive, nous allons maintenant expliquer le parcours suivi par un événement entre les différents modules.

Considérons le cas de l'utilisateur pressant le bouton de la souris dans la région de fermeture de la fenêtre permettant la saisie d'un message fonctionnel interactif.

L'événement de bas niveau est reçu par le contrôleur des événements qui le transmet à l'analyseur des événements. Celui-ci détecte qu'il s'agit d'un événement de type "MOUSEDOWN" localisé dans une fenêtre. Il fait donc appel au gestionnaire de fenêtre qui effectue les opérations de fenêtrage définies pour effectuer la fermeture d'une fenêtre (raffraichissement d'écran,...) et renvoit l'événement de bas niveau à l'analyseur sous la forme d'un événement abstrait dont la sémantique est enrichie de l'information "INCLOSE", indiquant qu'une action de fermeture a été effectuée sur la fenêtre courante de l'interface, en l'occurence la fenêtre "Window-i" associée à un message interactif de saisie.

Cet événement abstrait est alors traité par le moteur du dialogue qui consulte la base BD Dialogue pour identifier le message interactif qui correspond à l'objet interactif "Window-i" et déterminer s'il s'agit d'un message de saisie ou d'affichage. Dans notre cas, le moteur identifie le message interactif Di et constate qu'il s'agit d'un message de saisie. Dès lors, il génère la requête "communiquer message Di" et cette requête est transmise au contrôleur des requêtes du dialogueur afin d'être envoyée vers le contrôleur de l'échange.

Quand le contrôleur de l'échange reçoit la requête, il la passe à l'analyseur des requêtes. Lorsque celui-ci la traite, il fait appel au mapper qui en effectue la transformation. Pour ce faire, le mapper utilise la base BD Echange afin d'identifier le message-externe-fonction qui est associé au message interactif Di. En supposant que ce message interactif corresponde à un seul message-externe-fonction Ti, le mapper génére alors directement la requête "recevoir message Ti" et la communique à l'analyseur des requêtes.

L'analyseur des requêtes de l'échangeur prend en charge cette nouvelle requête. Il la communique au contrôleur de l'échange, après avoir éventuellement détruit une requête "réceptionner message Ti" transmise précédemment par le gestionnaire des traitements à l'échangeur, si le moteur des traitements l'avait générée après avoir détecté qu'il lui manquait ce message Ti.

Le contrôleur de l'échange transmet enfin la requête "recevoir message Ti" au composant auquel elle est destinée, c'est-à-dire au gestionnaire des traitements.

Quand le contrôleur des traitements prend en charge la requête, il la communique à son tour à l'analyseur des requêtes qui la transmet au moteur des traitements. Celui-ci consulte la base BD Traitements afin de détecter pour quelle fonction de la phase en cours d'exécution cette occurence du message-externe-fonction Ti constitue un message d'entrée.

Si la réception de ce message correspond à l'unique condition de déclenchement de la fonction, le moteur des traitements lance cette dernière. Tout message-externe-fonction produit en sortie de la fonction exécutée, qu'il soit un message résultat pur ou d'erreur, sera alors pris en charge par le moteur des traitements en vue de générer une requête "afficher message Ti" qui pourra être transmise au contrôleur des traitements. Ce dernier la communiquera au dialogueur via l'échangeur de façon similaire à ce que nous venons décrire dans le sens inverse.

V.2.3.5 - Les caractéristiques de DET

Comme nous l'avons fait lors de la présentation des SGD DIALOGUE et MOUSE, nous allons reprendre les différents critères d'implémentation cités au paragraphe V.1 et décrire leur mise en oeuvre dans le cadre de DET.

V.2.3.5.1 - Niveau d'abstraction

Le niveau d'abstraction des entités échangées entre le dialogueur et le gestionnaire des traitements est très élevé.

En effet, le dialogueur envoit et reçoit des commandes et des messages interactifs qu'il peut utiliser directement puisque ce sont des types d'objets dont il possède la connaissance . Quant au gestionnaire des traitements, il envoit et reçoit des requêtes de type commande ou message-externe-fonction; ces requêtes font également partie des objets qu'il manipule.

La mise en correspondance des entités de haut niveau connues par les composants "Gestionnaire des traitements " et "Dialogueur " est faite par le troisième composant de l'architecture, c'est-à-dire l' "Echangeur ", pour les rendre conformes aux objets manipulés par le composant destinataire.

V.2.3.5.2 - Localisation du contrôle

Le contrôle peut être considéré comme un contrôle mixte; il n'y a pas de domination du Dialogueur ni du Gestionnaire des Traitements. Cependant, le mode de partage de sa localisation est particulier, puisqu'il est réparti entre les quatre composants de l'application.

La partie "Interface " du dialogueur possède le contrôle sur les événements de bas niveau provoqués par l'utilisateur.

La partie ayant accès à la base de spécification du dialogue, et contenant le moteur de la conversation de ce dialogue, possède le contrôle sur les requêtes en provenance ou à destination du dialogueur ainsi que sur le déroulement du dialogue.

L'Echangeur possède le contrôle sur toutes les requêtes qu'il reçoit ou envoit aux autres composants.

Enfin, le Gestionnaire des Traitements contrôle les requêtes reçues ou envoyées par les traitements ainsi que l'enchaînement dynamique de l'application fonctionnelle.

V.2.3.5.3 - L'ordonnancement des événements

Chacun des composants possédant une partie du contrôle gère une file d'événements ou de requêtes. Toutes ces files sont a gérées d'après un ordre d'extraction qui respecte les contraintes de prise en compte des requêtes définies au paragraphe IV.2.3.2. On pourra également envisager de donner la possibilité aux composants "Traitement" et "Dialogue" de définir des niveaux de priorité spécifiques pour certains événements ou requêtes.

V.2.3.5.4 - L'Adaptabilité

Comme dans MOUSE, l'interface est adaptable sur intervention explicite du concepteur. Toute adaptation nécessite cependant la recompilation de la spécification de l'interface, intégrant ainsi les nouvelles conditons et caractéristiques souhaitées.

V.2.3.5.5 - Le contexte

En ce qui concerne le contexte,différentes possibilités de gestion, plus ou moins fine, sont envisageables. Cependant, une bonne gestion de contexte demande un investissement logiciel considérable. Il faudrait donc étudier attentivement le rapport coût/intérêt de l'utilisateur face à une gestion du contexte de qualité.

Notons cependant que la gestion des commandes courantes est absolument indispensable.

Nous terminons ainsi un rapide tour d'horizon de ce que pourrait être l'architecture d'un système de gestion d'applications interactives.

V.3 - En résumé

Dans ce chapitre, nous nous sommes attardées sur différentes possibilités d'implémentation d'un SGD.

Nous avons tout d'abord expliqué quelques critères permettant de juger de la qualité d'une telle structure logicielle.

Nous avons ensuite étudié deux systèmes que nous avons utilisés dans le cadre de notre stage : DIALOGUE et MOUSE. Ces SGD ont été décrits sur base des critères fournis en début de chapitre. Le SGD MOUSE a été introduit après une présentation des outils APEX et PAC sur lesquels il s'appuye.

Pour terminer, nous avons présenté une architecture logicielle de système de gestion d'applications interactives qui répondrait à la modélisation détaillée dans ce mémoire. L'architecture DET permet une séparation totale des traitements et du dialogue de l'application tout en assurant leur collaboration efficace. Dans le futur, cette architecture pourra faire l'objet d'une implémentation effective.

Conclusion

Conclusion

Dans ce mémoire, nous avons effectué une approche globale de la modélisation d'une application interactive. Une telle application comporte une part importante de dialogue avec l'utilisateur.

Avant de passer à l'étape de modélisation, nous nous sommes donc attardées sur les caractéristiques auxquelles doit répondre un bon dialogue. Nous avons parlé de la difficulté de conception de l'interface homme-machine d'une application, qui exige des qualités pluridisciplinaires.

Nous sommes ensuite passées à l'étape de modélisation d'une application interactive. Celle-ci comporte trois parties que nous avons envisagées tour à tour.

Des recherches menées au sujet de la spécification des données et des traitements d'une application ont donné lieu à la création de bon nombre de modèles permettant de les représenter. Ceux-ci fournissent la première partie de la modélisation d'une application interactive. Nous avons présenté l'un de ces modèles.

Le second élément d'une application interactive est le dialogue. Celui-ci a aussi donné lieu à beaucoup de recherches. Cependant, ces dernières sont essentiellement ergonomiques et trop rarement empreintes d'un souci de modélisation. Nous avons tenté de remédier à cette carence en proposant un modèle qui aide le concepteur à spécifier de manière plus précise l'interface d'une application. Ce modèle est principalement basé sur les concepts de message et d'objet interactifs. Le manque de littérature à ce sujet a néanmoins rendu notre tâche assez difficile; c'est à cette partie que nous avons donc consacré une grande part de notre réflexion.

Le troisième et dernier élément d'une application interactive qu'il faut modéliser correspond à la structure d'échange permettant la communication et la collaboration entre les traitements et le dialogue.

Dans ce cadre, nous avons essentiellement défini la notion de requête qui permet aux traitements de faire parvenir au dialogue des commandes ou des messages, et inversement.

Enfin, après avoir décrit tous ces modèles, nous nous sommes attachées à donner un aperçu des outils existant en matière d'aide à la conception du dialogue et des possibilités d'implémentation d'un système de gestion d'application interactive respectant les modèles précédemment définis.

Pour cela, nous avons détaillé deux réalisations que nous avons eu l'occasion d'étudier, DIALOGUE et MOUSE, ainsi que l'architecture du système DET destiné à gérer les trois parties de la spécification d'une application exposées lors de la présentation des modèles.

Au terme de ce travail, mentionnons encore les différents axes selon lesquels les recherches pourront être poursuivies.

Le premier axe est l'approfondissement des modèles du dialogue et de l'échange.

Différents aspects de la conversation ont été volontairement passés sous silence. L'enchaînement entre différentes phases d'une application n'a pas été abordé. La description des conditions de synchronisation de messages n'a été qu'esquissée. Le problème que nous avons soulevé à propos de la reprise d'une phase après un certain temps d'interruption reste également en suspens. Souvenons-nous, en effet, que la reprise d'un traitement risque de remettre en cause la cohérence de la base de données de l'application.

Dans le même axe de développement, il reste aussi à formaliser les langages de spécification du dialogue et de l'échange.

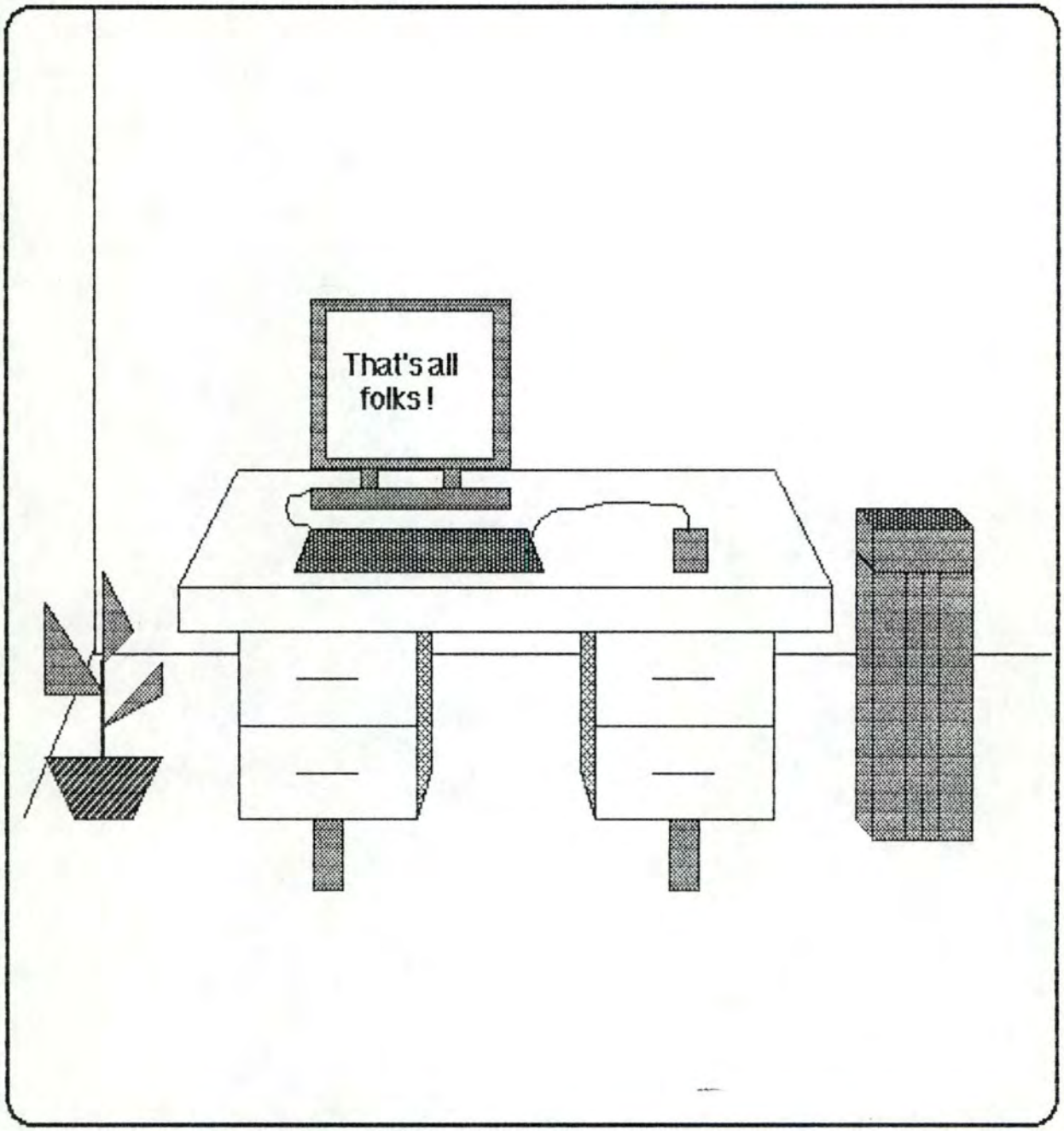
Une autre tâche à réaliser est la précision détaillée et exhaustive de l'architecture générale du système DET, ce qui permettra de passer à l'étape suivante que représente l'implémentation du système. DET permettra alors d'assurer l'exécution d'une application interactive sur base de la spécification de ses fonctionnalités et de son interface.

DET doit également être amélioré pour permettre la gestion de différentes versions d'interface, dans le cas où une application est destinée à plusieurs classes d'utilisateur.

Enfin, DET peut être enrichi d'un générateur semi-automatique de dialogue, travaillant sur base des spécifications de l'application fonctionnelle.

Pour conclure, le troisième axe de réflexion que nous suggérons d'approfondir concerne la gestion du contexte de l'application. A son niveau le plus évolué, un observateur intelligent pourrait alors se charger, sur base du contexte, de l'adaptation de l'interface vis-à-vis de l'utilisateur, en tenant compte de ses habitudes, de ses capacités, et de ses goûts. L'interface évoluerait donc en parallèle avec les progrès de l'utilisateur.

A vous de jouer !


That's all
folks !

Bibliographie

[Ahlsen, Britts]
Matts Ahlsen, Stefan Britts, " An architecture for Dialog Management in Opal ", SYSLAB, report n° 43, June 1986

[Bass 1]
Leonard J. Bass, " An approach to user specification of interactive display interfaces ", IEEE transactions on software engineering, vol. SE-11, n° 8, August 1985

[Bass 2]
Leonard J. Bass, " A generalized user interface for application programs "

[Bobrow, Stefik]
Daniel G. Bobrow, Mark Stefik, " Object-oriented programming. Themes and variations ", Intelligent Systems Laboratory, PARC, Californie, The AI Magazine, 1984-85

[Bodart, Pigneur]
François Bodart, Yves Pigneur, " Conception assistée des applications informatiques - 1 - Etude d'opportunité et analyse conceptuelle ", Masson, Paris 1983

[Bodart 86]
François Bodart, " Contribution des spécifications conceptuelles d'un système d'information à la modélisation des dialogues ", brouillon, septembre 1986

[Bodart 87]
François Bodart, " Opening Address ", Conférence sur l'approche Entité-Association, Elsevier Science Publishers b.v. (North Holland), ERI, 1987

[Brown]
James W. Brown, " Controlling the complexity of menu networks ", Jet Propulsion Laboratory, Communications of the ACM, vol. 25, num. 7, July 1982

[Buxton]
Bill Buxton, " Human-computer interface ", extrait de " Graphical input interaction technique : workshop summary ", Computer Graphics, January 1983, pp 9 à 15

[Buxton, Tanner]
W. Buxton, P. Tanner, " Some issues in future User Interface Management Systems (UIMS) development ", IFIP working group 5.2, Workshop on User Interface Management, Seeheim, West Germany, November 1983

[Carton]
Jean-Louis Carton, " Contribution à l'étude du dialogueur de l'atelier logiciel Concerto : une interface intégrée, interactive et évolutive ", Mémoire, IIE et CNAM, juin 1984

[Chen]
P. P. Chen, " Entity-Relationship Approach to Information Modeling and Analysis ", ER Institute 1981

[Conchon, Camacho,Rasser]
A. Conchon, Juan Camacho, Anne-Marie Rasser, " Une session sur le poste de travail Concerto ", Troisième Colloque - Exposition de Génie Logiciel, Versailles 27-30 mai 1986, pp 57 à 68, Express-Tirages 1986

[Cornafion]
Cornafion, " Programmation et exécution d'applications réparties " in " Systèmes informatiques répartis. Concepts et techniques ", Dunod, 1981

[Courbon -1]
Jean-Claude Courbon, " Interfaces et générateurs de dialogue dans les S.I.A.D. ", Université de Genève

[Courbon -2]
Jean-Claude Courbon, " Design of intelligent dialogue in decision support systems ", Université de Genève

[Courbon 85]
Jean-Claude Courbon, " Feature analysis of a dialog generator for decison support systems ", University of Geneva, EURO VII Congress in Bologna, June 1985

[Coutaz, Herrmann]
Joëlle Coutaz, Marc Herrmann, " Applications interactives et interface usager ", IMAG, Université de Grenoble, novembre 1983

[Coutaz 85]
Joëlle Coutaz, " Abstractions for user interface design ", IEEE Computer 18 (9), September 1985

[Coutaz 86-1]
Joëlle Coutaz, " The construction of user interfaces ", IMAG, University of Grenoble,February 1986

[Coutaz 86-2]
Joëlle Coutaz, " Projet Guide - Thème : interface homme-machine ", IMAG, Université de Grenoble, juin 1986

[Coutaz 86-3]
" La construction d'interfaces homme-machine ", IMAG, Université de Grenoble,RR 63.5-I, novembre 1986

[Coutaz 86-4]
" MOUSE, an Object-Oriented UIMS ", Bull Research Center c/o IMAG, September 1986

[Coutaz 87]
" PAC, an object oriented model for implementing user interfaces ", to appear in Sig Chi Bulletin, Automne 1987

[Crespin, Piquard]
Sophie Crespin, Patrick Piquard, " Conception d'un éditeur graphique pour le langage de spécification DSL ", Mémoire, FNDP Namur, Institut d'informatique, 1986-87

[Detrembleur]
Bernard Detrembleur, " Use and test of DIALOGUE for COMS project ", Geneva, 14 November 1986

[Faulle]
Bernard Faulle, " L'informatique conversationnelle : Méthodologie d'analyse et de présentation ", Les éditions d'orgzanisation, 1982

[Green]
Mark Green, " A methodology for the specification of graphical user interface ", Computer Graphics, vol. 15, n°3, août 1981

[Hertz]
Denis Hertz, " Modèle de dialogue utile à la réalisation d'un générateur pour les systèmes interactifs d'aide à la décision (SIAD) ", Centre Universitaire d'Informatique de l'Université de Genève, juin 1986

[Konsynski, Kuo]
Ben Konsynski, Bob Kuo, " An architecture for dialogue management : implications in user-computer dialogue design ", University of Arizona, Interfaces in Computing, 3 (1985), pp 259-275

[Krakowiak]
S. Krakowiak, " Introduction à la programmation par objets ", Université de Grenoble, octobre 1985

[Lepenant]
C. Lepenant, " Le poste de travail dans le système de gestion de données Altaïr ", Journées de travail des 9, 10 et 11 juin 1987 à Seyssel sur le thème : " Le poste de travail dans les systèmes d'information "

[Lermigeaux]
Frédéric Lermigeaux, " Un modèle d'architecture pour application interactive sur Macintosh ", Université de Grenoble, Rapport de stage de DESS Génie informatique, Laboratoire de recherche BULL-IMAG, septembre 1986

[Nievergelt]
J. Nievergelt, C. Muller, H. Sugaya, " Dialog design : principles and experiments ", Proc. Brown Boveri Symp. on Computer Systems in Process Control, Baden, September 1985

[OMEGA 86-1]
J. André, M. de Gaudemont, T. Métais, R. Roire, M. Vargoz, " Un atelier intégré pour la production et la maintenance d'applications transactionnelles : OMEGA - Manuel de référence OMEGA V1.0 ", 20 janvier 1986

[OMEGA 86-2]
François Bodart, Thierry Métais, Patrick Moulin, Pascale Stenne, "Rapprochement des méthodes et des systèmes logiciels IDA et OMEGA", document de sunthèse, 7 mars 1986

[OMEGA 86-3]
J. André, A. Gena, " Spécification du poste de travail IDA et de l'interface IDA-OMEGA " , 4 août 1986

[Petoud 87-1]
Isabelle Petoud, " Conception de l'interface homme/machine dans un environnement de gestion ", Université de Lausanne, printemps 1987

[Petoud 87-2]
Isabelle Petoud, " La conception des dialogues dans les SI ", Université de Lausanne, juin 1987

[Petoud 87-3]
Isabelle Petoud, " Systèmes de gestion de dialogue ", Université de Lausanne, 1987

[Pigneur]
Yves Pigneur, " Spécification fonctionnelle d'une phase (interactive)", brouillon, septembre 1986

[Rolland, Richard]
C. Rolland, C. Richard, " The Remora Methodology for Information System Design ", in Information System Design Methodology - a Comparative Review, pp 369-426, North-Holland, 1982

[Rosenthal,Yen]
Dave Rosenthal, Albert Yen, "User interface models summary", extrait de " Graphical input interaction technique : workshop summary ", Computer Graphics, January 1983, pp 16 à 20

[Scapin]
Dominique L. Scapin, " Guide ergonomique de conception des interfaces homme-ordinateur ", INRIA (Institut National de Recherche en Informatique et en Automatique), France

[Shneiderman]
Ben Shneiderman, "Software psychology : human factors in computer and information systems ", Cambridge, Winthrop, 1980

[Waterkeyn]
Pascal Waterkeyn, " Concevoir et réaliser l'interface usager des applications interactives ", Mémoire, FNDP Namur, Institut d'informatique, 1984-85

ANNEXES

Annexe 1 :

Le logiciel "Dialogue"

d'Apollo

PLAN

1 - Présentation générale de DIALOGUE

DIALOGUE est un outil de spécification de l'interface homme-machine d'un programme d'application. L'interface créée est composée d'une fenêtre principale, qui délimite la portion de l'écran dévolue au programme, et de portions de fenêtres qui peuvent venir se superposer ("popups"). DIALOGUE travaille en deux temps : c'est d'abord un langage doté d'un traducteur qui génère du code (des déclarations PASCAL, C et FORTRAN) et un fichier contenant la description du dialogue (fichier binaire). A l'exécution, c'est également un système qui interprète le fichier généré et anime le dialogue.

Les programmes écrits avec DIALOGUE comprennent deux parties :

- l'interface utilisateur, spécifiée dans un langage propre à DIALOGUE.
- l'application elle-même, qui contient la partie de traitement des données. (voir figure A-1)

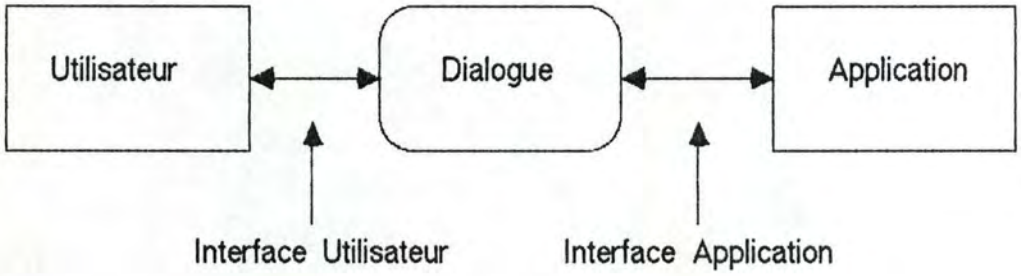


Figure A-1 : Relations entre l'utilisateur et l'application

1.1 - Avantage de cet outil

L'avantage d'utiliser un tel système est de forcer la séparation entre l'interface et l'application. Cela permet, entre autres choses, de développer plusieurs types d'interfaces pour un même programme d'application, et de fournir aux utilisateurs des interfaces semblables pour toutes les applications, puisqu'elles auront toutes été conçues avec le même outil.

1.2 - Les fichiers de DIALOGUE

Un concepteur d'application doit écrire trois fichiers différents :

- un fichier avec les commandes de DIALOGUE (spécification de l'interface)
- un fichier contenant toutes les routines de l'application
- un fichier contenant le programme principal

Le fichier de spécification de l'interface (nom-de-fichier. DPS) est soumis au translateur DIALOGUE. Celui-ci produit une description binaire du fichier (nom-de-fichier. DPD) et un fichier d'insertion spécifique à l'application (nom-de-fichier. INS.C). Le fichier d'insertion spécifique à l'application est inclu dans les fichiers de l'application en même temps que le fichier standard de DIALOGUE. L'application est ensuite compilée et transmise à l'éditeur de liens ("linkée"). (voir figure A-2)

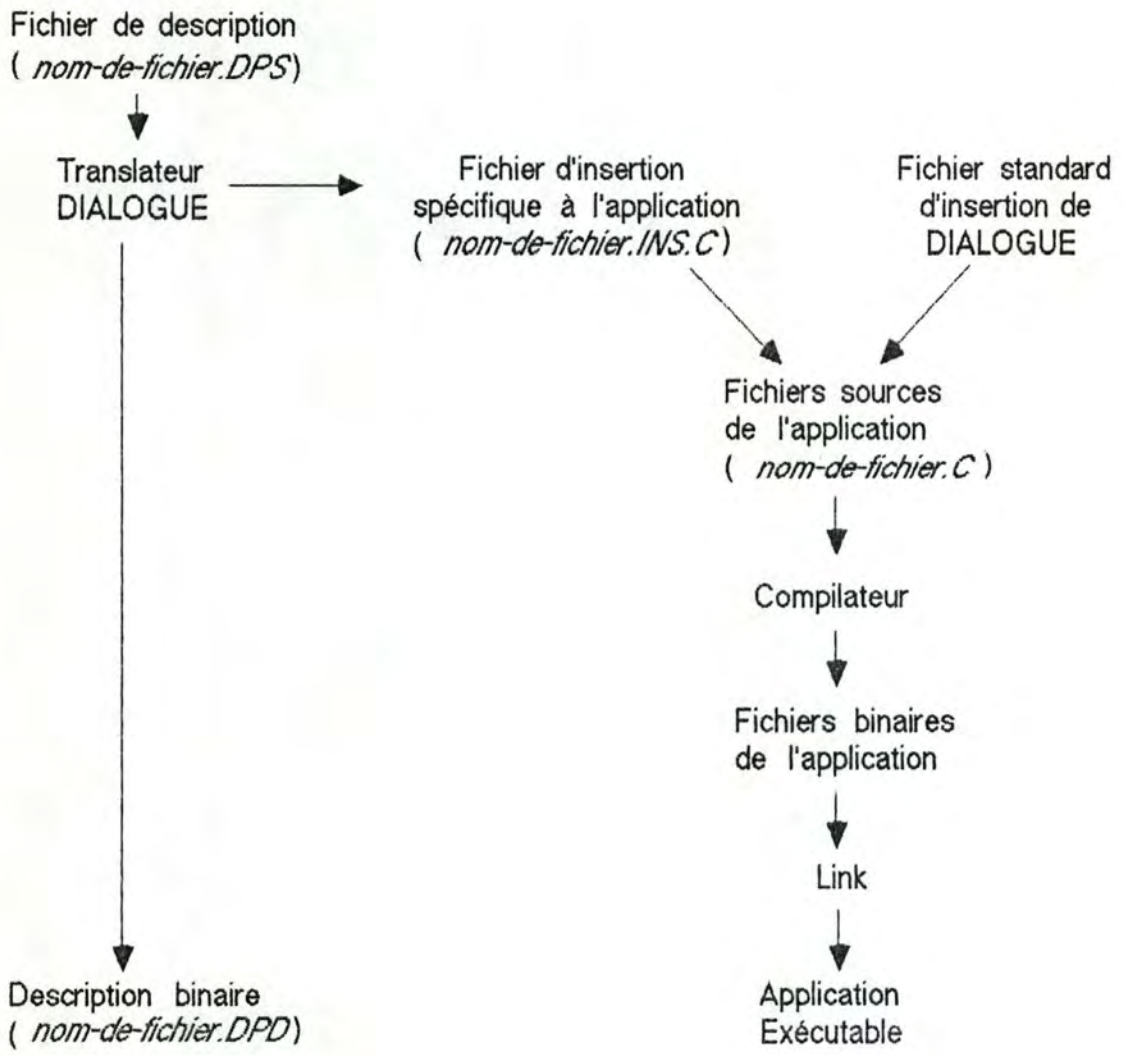


Figure A-2 : Les fichiers de l'application

1.3 - Le fichier de description (nom-de-fichier.DPS)

Le fichier de description est divisé en deux parties :

- l'Interface Application (constituée de tâches),
- l'Interface Utilisateur (constituée de techniques).

```
DIALOG

APPLICATION-INTERFACE nom

        définition de tâche
        définition de tâche
                .
                .
                .
USER-INTERFACE nom

        définition de technique de présentation
        définition de technique de présentation
                .
                .
                .
        définition de technique de structuration
        définition de technique de structuration
                .
                .
                .
END.
```

Figure A-3 : Format du fichier de description de l'interface

1.3.1 - L'Interface Application

L'interface application est composée de tâches qui représentent des opérations que l'utilisateur peut faire pour affecter l'application, comme par exemple fournir une donnée ou demander la réalisation d'une certaine opération. Une tâche représente donc une activité. Pour savoir quel type de donnée l'utilisateur doit introduire, ou à quel signal s'effectue le déclenchement d'une certaine opération, chaque tâche possède un type. Par exemple, pour vérifier si un entier est pair ou non, on doit définir une tâche de type entier. Les différents types de tâches de DIALOGUE sont :

- BOOL, donnée booléenne,
- ENUM, donnée énumérée (liste de choix parmi lesquels l'utilisateur fait une sélection),
- GMR, donnée Gmr (metafile),
- GPR, donnée Gpr (bitmap),
- INT, nombre entier,
- MSG, texte,
- NULL, pas de donnée (Une telle tâche sert aux applications qui peuvent agir sur demande de l'utilisateur sans nécessiter aucune donnée. C'est typiquement le cas d'une demande de sortie de l'application.),
- REAL, nombre réel,
- SET, donnée de type ensemble (liste de choix parmi lesquels l'utilisateur peut effectuer plusieurs sélections),
- STRING, ligne de texte.

Les données GMR et GPR sont celles qui concernent le clavier et la souris (frappe de touche, déplacement du curseur).

Une tâche peut appeler une routine de l'application, activer ou désactiver une tâche ou un groupe de tâches, ou retourner au programme principal. La définition de la tâche est décrite par la figure A-4.

```
nom-de-tâche := type:
    événement => <action>;
        .
        .
        .
end
```

Figure 4 : Définition de la tâche

Chaque tâche possède un identificateur et un type. L'identificateur est lié à une valeur dans le fichier d'insertion spécifique à l'application (nom-de-fichier.INS.C). Les identificateurs sont également utilisés pour rattacher une technique à une tâche (voir interface utilisateur).

Un événement se produit relativement à une tâche lorsque l'utilisateur fournit des données valides en entrée de la technique correspondant à cette tâche. La tâche reçoit la donnée en entrée, et effectue l'action correspondante. Les différents types d'événements sont :

- COMP (completed), qui indique qu'une donnée en provenance de l'utilisateur a été reçue, et que la tâche correspondante a été réalisée,
- HELP, qui indique que l'utilisateur a demandé de l'aide pour cette tâche,
- événements GPR et GMR.

En réponse à un événement, une tâche peut effectuer zéro, une ou plusieurs actions. Ces actions sont :

- CALL, appel d'une routine de l'application,
- ACTIVATE, activation d'une tâche ou d'un groupe de tâches,
- DEACTIVATE, désactivation d'une tâche ou d'un groupe de tâches,
- RETURN, restitution du contrôle à l'application.

Les attributs et leurs valeurs possibles sont spécifiés, pour chaque type de tâche, dans le manuel de référence de DIALOGUE (chapitre 5). Pour un entier, on peut par exemple spécifier ses valeurs maximale et minimale, ainsi qu'une valeur par défaut.

Des exemples de description de tâches sont fournis au paragraphe 2.1.

1.3.2 - L'Interface Utilisateur

L'interface utilisateur est composée de deux types de techniques. Tout d'abord, le programmeur décrit des techniques de présentation qui explicitent la présentation des tâches à l'écran. Ces techniques sont des outils visuels tels que des menus ou des icônes, et permettent à l'utilisateur d'introduire des données que DIALOGUE peut mémoriser et transmettre à l'application. Ensuite, le programmeur peut structurer les techniques de présentation, les grouper, pour indiquer leur ordre de présentation à l'écran.

Chaque technique PEUT être rattachée à une tâche, et chaque tâche DOIT avoir au moins une technique qui lui correspond. Les techniques de structuration, qui ne représentent pas réellement des objets de l'application, ne sont pas associées à des tâches. Il n'est pas non plus interdit, par exemple, de faire apparaître à l'écran une icône qui contient un message quelconque, mais n'a aucun sens pour l'application, et qui n'est donc pas rattachée à une tâche de l'interface application. Les différents types de techniques de présentation sont :

- BOOL-FIELD, pour introduire ou afficher des données booléennes,
- ENUM-FIELD, pour sélectionner un item dans une liste,
- INT-FIELD, pour introduire ou afficher des entiers,
- REAL-FIELD, pour introduire ou afficher des réels,
- SET-FIELD, pour sélectionner un ou plusieurs items dans une liste,
- STRING-FIELD, pour introduire ou afficher une ligne de texte,
- DISPLAY-FIELD, pour afficher un texte,
- GRAPHICS-AREA, pour introduire des données GMR ou GPR,
- ICON, pour générer un événement sans introduire de données,
- MENU, pour sélectionner une opération dans une liste,
- SCROLLBAR, pour faire défiler le contenu d'un menu ou d'une technique "switch" ("scrolling"),
- SWITCH, pour passer en revue une série de choix de manière circulaire (quand on arrive au dernier, on revient au premier).

Les techniques de structuration sont :

- ROW, pour organiser des techniques en ligne ou en colonne,
- ONEOF, qui contient plusieurs techniques, mais n'en affiche qu'une seule à la fois,
- WINDOW, qui définit la fenêtre dans laquelle l'application travaille,
- POPUP, pour définir un rectangle qui peut être superposé à la fenêtre initiale,
- SPACE, qui permet de définir des zônes de séparation entre les différentes techniques,

La syntaxe d'une technique est décrite dans la figure A-5.

```
nom-de-technique := type:
        événement => <action>;
                .
                .
                .
        attribut = valeur;
                .
                .
                .
        end
```

Figure A-5 : Syntaxe d'une technique

Chaque technique possède un identificateur et un type. Les techniques de structuration utilisent ces identificateurs pour désigner les techniques de présentation qu'elles regroupent.

Un événement qui concerne les techniques se produit lorsque l'utilisateur fournit des données en entrée à une technique. Un tel événement correspond à la notion d'événement de bas niveau que nous avons décrite au paragraphe V.1.1. Dans le cadre de ce logiciel, ces événements sont :

- la frappe d'une touche du clavier ou de la souris, lorsque le curseur se trouve dans une technique,
- ENTER, qui signifie que l'utilisateur a bougé le curseur à l'intérieur de la fenêtre de DIALOGUE,
- LEAVE, qui signifie que l'utilisateur a sorti le curseur d'un "popup" ou d'une "window",
- SELECT, lorsque l'utilisateur a sélectionné une icône ou un item de menu.

En réponse à un événement, une technique peut exécuter zéro, une ou plusieurs actions. Pour déclencher certaines actions, des touches par défaut sont définies, mais le programmeur peut en déterminer lui-même. La liste des actions se trouve ci-dessous. Pour plus d'informations à ce propos, veuillez vous référer au manuel DIALOGUE, page 6-37.

- ACCEPT
- BACKSPACE
- BE-CURRENT-CHILD
- CHAR-DEL
- CURSOR-LEFT OR CURSOR-RIGHT
- HELP
- INSERT-TOGGLE
- LINE-DEL
- NEXT-FIELD
- POPDOWN
- SCROLL-DOWN, LEFT, LEFT-JUSTIFY, LEFT-UNIT, RIGHT, RIGHT-JUSTIFY, RIGHT-UNIT, UP
- SELECT
- SHOW

Les attributs et leurs valeurs possibles sont explicités, pour chaque type de technique, dans le chapitre 6 du manuel de référence. On peut, par exemple, déterminer la taille d'une technique. Si on omet d'indiquer une valeur pour certains attributs, des valeurs par défaut sont utilisées par DIALOGUE. En particulier, la taille d'un champ est calculée par rapport à la taille de la fenêtre dans laquelle il se trouve.

Des exemples de définition de techniques sont fournis au paragraphe 2.1.

1.3.3 - Le fichier contenant le programme principal

Ce fichier est assez différent d'un programme principal classique. Il n'est cependant pas difficile à concevoir; il est en effet très semblable pour toutes les applications utilisant DIALOGUE. Il contient des opérations telles que l'initialisation et la clôture de DIALOGUE, l'activation des tâches et l'attente d'événements venant de l'utilisateur. Ci-dessous se trouve un squelette de programme principal, valable pour toute application.

```
/*
Insert include files
*/

#include "/sys/ins/base.ins.c"
#include "/sys/ins/dialog.ins.c"   /* Standard DIALOG Package insert file. */
#include "file-name.ins.c"         /* File generated by DIALOG when the file
                                      example.dps is passed through the
                                      translator. */

main()
{

/* Initialize application data */
status-$t   status;
            .
            .
            .
/* Initialize DIALOG */
dp-$init (unit-or-pad, dsc-file-name-len, entry-vector, dp-key, status);

/* Set default or initial values to the different tasks. */
dp-$TYPE-set-value (task-id, value, status);
```

```
/* Activate a task or task-group. */
dp-$task-activate (task-id, status);

/* Wait for an event on input. */
dp-$event-wait (task-id, event-id, status);

/* Exit from DIALOG. */
dp-$terminate (status);

}
```

1.3.4 - Le fichier des routines de l'application

Ce fichier regroupe toutes les routines de l'application, c'est-à-dire toutes les routines appelées dans une déclaration COMP d'une tâche, et toutes les routines classiques d'une application. Ces procédures peuvent, comme le programme principal, utiliser les routines de la bibliothèque ("library") DIALOGUE (voir chapitre 7 du manuel DIALOGUE). Les noms de toutes ces procédures sont de la forme : DP-$XXXXXX. Leurs fonctionnalités sont :

- initialiser et terminer DIALOGUE,
- attendre un événement,
- transmettre des données de l'application vers l'interface, et à l'inverse, transmettre des données saisies par l'interface vers l'application,
- avertir l'utilisateur de certaines conditions ou erreurs,
- traiter les types de tâches particuliers (ENUM, SET, GMR, GPR).

2 - Un exemple simple d'utilisation de DIALOGUE

Ce petit programme a pour but de déterminer si un nombre entier fourni par l'utilisateur est pair ou non. Vu sa simplicité, son seul intérêt est l'apprentissage du logiciel DIALOGUE.

2.1 - Le fichier de description de l'interface (example.dps)

Convention : Dans ce fichier, tous les mots réservés de DIALOGUE sont indiqués en lettres majuscules.

```
DIALOG

APPLICATION-INTERFACE example

exit-task:=NULL:
        COMP=> <RETURN>
        END

number-task:=INT:
        COMP=> <CALL odd-or-even>;
        MIN = 0;
        MAX = 20
        END

true-false-task:=BOOL:
        COMP=> <>
        END
```

```
message-task:=MSG:
            VALUE = "   This program determines if an integer is even or odd. "
                  &"  Position the cursor with the mouse (left button). "
                  &"  Then type a number between 0 and 20, and <RETURN>."
                  &"  That's all ! "
            END


USER-INTERFACE example

%INCLUDE "/sys/ins/dialog-user.ins.dps"


exit:=ICON:
            TASK = exit-task;
            BACKGROUND = GRAY;
            SHAPE = ROUNDED;
            SIZE = (100 350) PIXELS;
            STRING = "exit"
            END

number:=INT-FIELD:
            TASK = number-task;
            BACKGROUND = OFF;
            SHAPE = ROUNDED;
            HELP-TEXT = "you must give an integer from 0 to 20"
            END
```

```
true-false:=BOOL-FIELD:
            TASK = true-false-task;
            BACKGROUND = OFF;
            SHAPE = ROUNDED;
            HELP-TEXT = "true = even number"
                        & "false = odd number"
            END


row-bottom := ROW:
            BACKGROUND = ON;
            ORIENTATION=HORIZONTAL;
            BORDER-WIDTH = 10;
            DIVISION-WIDTH = 5;
            OUTLINE = ON;
            SHAPE = ROUNDED;
            CONTENTS = (exit
                        number
                        true-false)
            END


message := DISPLAY-TEXT:
            TASK = message-task;
            SHAPE = ROUNDED;
            SIZE = ((200 60) (250 80) (300 100)) PIXELS
            END
```

```
row-all := ROW:
            BACKGROUND = ON;
            ORIENTATION=VERTICAL;
            BORDER-WIDTH = 10;
            DIVISION-WIDTH = 5;
            OUTLINE = ON;
            SHAPE = ROUNDED;
            CONTENTS = (row-bottom
                        message)
            END



  std-window:-
            CONTENTS = row-all
            END


END.
```

2.2 - Le fichier d'insertion spécifique à l'application (example.ins.c)

Ce fichier est généré par DIALOGUE.

```
/* DOMAIN/Dialogue C insert file from //bd-cao/user/genevieve /example.dps
*/ /* Description file processed on Monday, 1 December 1986, at 10:00. */

/* Entry Vector Definition */

void odd-or-even() ;
dp-$callback-ptr-t  dp-$example-entry-vector[1]
            = {
                odd-or-event};
/* Task Definitions */
#define dp-$call-task-group 1L
#define exit-task 2L
#define number-task 3L
#define true-false-task 4L
#define message-task 5L
/* Verification Key */
#define dp-$example-key 0xF577296EL

/* End of DOMAIN/Dialogue C insert file from
//bd-cao/user/genevieve /example.dps */
```

2.3 - Le programme principal (example-main.c)

Ce programme est semblable pour toutes les applications qui utilisent DIALOGUE.

```
/* Insert includes files */

#include "/sys/ins/base.ins.c"
#include "/sys/ins/dialog.ins.c"      /* Standard DIALOG Package insert file. */
#include "example.ins.c"              /* File generated by DIALOG when the file
                                         example.dps is passed through the
                                         translator. */

extern int dp-$example-head;          /* Declaration for the initialization procedure
                                         dp-$init-from-memory. */

main()
{
status-$t        status;
dp$t-ask-id       task;
dp-$even-tid     event;

/* Initialize DIALOG */
/* ATTENTION : I use here another procedure that the one in the typical main
program to initialize DIALOGUE. There are 2 different initialization procedures. I
explain the use of these routines in paragraph 3. */
dp-$init-from-memory (stream-$stdout,&dp-$example-heap, dp-$example-key,
status) ;
```

```
/* Set default or initial values to the different tasks. */

/* ATTENTION : I have not used the initialization for the integers because there
are problems. When I put the initial value to 0, the result on my screen is 1.
Putted a 1, it gives 85617, at 2 it gives 131073, ... So, I let DIALOG make the
initialization at default value : 0. */

dp-$bool-set-value (true-false-task, true, status);

/* Activate a task or task-group. */
dp-$task-activate (dp-$all-task-group, status);

/* Wait for an event on input. */
dp-$event-wait (task, event, status);

/* Exit from DIALOG */
dp-$terminate (status);

}
```

2.4 - Les routines de l'application (example-rtn.c)

Ce paragraphe ne contient qu'une routine, qui détermine si un entier est pair ou impair.

```
#nolist
#include "/sys/ins/base.ins.c"
#include "/sys/ins/error.ins.c"
#list
#include "/sys/ins/dialog.ins.c"
#include "example.ins.c"
```

```
/* Redifine the values of true and false because the DOMAIN system defines it
somewhere else as 0 and 1.  But DIALOGUE uses the values of 0 and -1. */

#undef true
#undef false

#define true -1
#define false 0

/*-------------------------------*/
void odd-or-even ()
/*-------------------------------*/
/* This procedure is declared "void" because the DIALOG translator declares it
"void" in the file file-name.INS.C */

{
int      value-int;
int      value-bool;
status-$t status;

dp-$int-get-value (number-task, value-int, status);
if ((value-int % 2)==0)
     value-bool = true;
else
     value-bool = false;
dp-$bool-set-value (true-false-task, value-bool, status);

}
```

3 - Quelques astuces pour utiliser DIALOGUE

Lorsque vous écrivez une application en vous servant du logiciel DIALOGUE, procédez comme suit :

- Ecrire l'interface application (les tâches)
- Ecrire une interface utilisateur très simplifiée (les techniques)
- Ecrire le programme principal
- Ecrire les routines de l'application
- Ecrire l'interface utilisateur complète

La commande "DIALOG", qui passe un fichier de description d'interface au translateur, se trouve dans le catalogue ("directory") de nom "/COM".

Faites très attention; les noms de primitives du logiciel DIALOGUE (dp-$init, par exemple) sont écrits en lettres majuscules dans tout le manuel de référence, excepté les annexes. Cependant, tous ces noms doivent absolument être écrits en lettres minuscules dans les programmes, sous peine de mauvais fonctionnement.

Comme je l'ai déjà signalé dans le commentaire de la routine de l'exemple (paragraphe 2.4), la définition des valeurs TRUE et FALSE de DIALOGUE n'est pas la même que celle du reste du système APOLLO. Il faut donc redéfinir ces valeurs. Pour cela, le lecteur peut se référer au paragraphe 2.4 .

Le logiciel DIALOGUE fournit deux procédures différentes d'initialisation de l'interface. Lors de la conception d'une interface, le programmeur peut procéder de manière itérative. Dans ce cas, il est fortement conseillé d'utiliser la routine dp-$init. Grâce à cette routine, lorsque l'on modifie la partie interface utilisateur (les techniques) du fichier de description de l'interface (nom-de-fichier.DPS), il n'est pas nécessaire de recompiler toute l'application. Il suffit de repasser ce fichier de description au translateur DIALOGUE. Par contre, si l'on modifie les tâches, il faut recompiler toute

l'application et rééditer les liens correspondants. Cependant, cette procédure d'initialisation associe automatiquement au programme que l'on veut exécuter le dernier fichier qui a été traduit par le translateur DIALOGUE. On peut donc se retrouver avec une interface qui n'a rien à voir avec notre programme. La procédure dp-$init-from-memory remédie à cet inconvénient. Elle n'est cependant pas utilisable lors de la mise au point de l'interface, car à chaque modification du fichier de déclaration il est nécessaire de recompiler l'application et de rééditer les liens entre les fichiers de l'application et le fichier *file-name*-DPD.BIN, qui est fourni par le translateur. Utiliser cette procédure d'initialisation dès le départ produirait une perte de temps considérable.

4 - Les limitations de DIALOGUE

Chaque application doit avoir exactement une fenêtre. Il est donc impossible d'ouvrir plusieurs fenêtres pour la même application.

Le manuel de référence nous dit que l'on peut définir la taille des différents champs qui apparaissent sur l'écran. Après avoir testé cette possibilité, je me suis aperçue que, lorsqu'on aligne les icônes de manière verticale, donc les unes au-dessus des autres, il est en effet facile de modifier leur hauteur. Cependant, quelle que soit la largeur spécifiée, le logiciel utilise la largeur de la fenêtre comme largeur de tous les champs. Il faut alors rajouter une technique SPACE (espace vide), et indiquer la taille de cette dernière pour obtenir le résultat désiré. De manière symétrique, lors d'un alignement horizontal, on peut spécifier la largeur, mais pas la hauteur.

Les tâches de type MSG (message), ne peuvent accepter l'introduction de texte par l'utilisateur. Pour ce faire, il faut utiliser des tâches de type STRING, qui ne permettent que la saisie d'une seule ligne de texte.

" DIALOGUE fournit la possibilité de définir une fenêtre principale pour l'application, ainsi que des portions de fenêtres qui peuvent venir se superposer ("popups"). Cela ne correspond pas à une vraie gestion de fenêtre, car on ne peut pas passer librement de l'une à l'autre. Les popups sont empilés et seul le dernier est actif (c'est le seul où une interaction peut se produire). Pour pouvoir accéder à une portion recouverte, il faut désempiler les autres, ce qui les rend invisibles. " [Isabelle Petoud]

Pour toute technique, il devrait y avoir un attribut qui détermine les opérations que l'utilisateur peut effectuer dessus (lecture, écriture, ou les deux).

Annexe 2 :

APEX

PLAN

Introduction

Dans le cadre du stage effectué à Grenoble pour ce mémoire, nous avons récupéré le noyau d'APEX développé au cours des précédents tests d'implémentation effectués par Joëlle Coutaz et Fred Lermigeaux [Lermigeaux], afin de réaliser une application permettant l'émulation d'un terminal graphique (en l'occurence le Macintosh), utilisé comme périphérique d'entrée/sortie pour contrôler l'exécution et visualiser les résultats d'un programme de simulation robotique exécuté sur un ordinateur de type VAX.

Au cours de cette annexe, nous illustrerons donc l'utilisation d'APEX pour la construction d'une application interactive en présentant certains modules du programme qui a permis d'implémenter l'application considérée. Nous compléterons cet exposé par une critique des composants d'APEX tels qu'ils étaient réalisés et mis en oeuvre fin janvier 1987.

1 - Un exemple de programmation

Conformément à l'architecture d'implémentation d'une application interactive proposée par le modèle APEX, le code d'une telle application est décomposé en deux parties principales : APPLICATION et INTERFACE.

Le noyau d'APEX, INTERFACE, est lui-même constitué de différents modules réutilisables destinés à la gestion du dialogue de toute application.

Dans le cadre de cette annexe, nous présenterons l'essentiel du code des modules principaux composant le noyau d'APEX, à savoir le contrôleur du dialogue (dialog.c) et l'analyseur des événements (eevent.c).

En ce qui concerne la description d'un serveur spécialisé, nous ne développerons pas le serveur fondamental que constitue le "driver" de fenêtre qui interprète tous les événements se rapportant au fenêtrage. En effet, ce serveur met en jeu un grand nombre de concepts et de techniques qu'il serait inutile et fastidieux d'expliquer ici.

Nous expliciterons donc un serveur assez simple, à savoir le gestionnaire de menu, que nous avons personnellement implémenté. La présentation de ce driver nous permettra en outre d'illustrer également la façon dont les objets interactifs peuvent être manipulés par les serveurs spécialisée qui en ont la charge.

Pour faciliter la lecture du code des modules exposés, signalons encore que des conventions d'écriture ont été utilisées pour distinguer les identificateurs : les mots en caractères minuscules désignent des variables définies dans le programme, les mots en caractères majuscules correspondent à des constantes du programme, les mots en caractères mélangés sont des mots-clés importés du Toolbox.

Pour terminer l'exposé de notre implémentation d'une application interactive basée sur le modèle APEX, nous soulignerons enfin la façon dont l'application est elle-même décomposée en deux ensembles de fonctions : celles qui permettent la gestion sémantique de l'interface de l'application et celles qui en implémentent effectivement les fonctionnalités.

1.1 - Le noyau d'APEX (INTERFACE)

Le noyau d'APEX comprend l'essentiel des modules réutilisables de gestion de l'interface de toute application interactive, c'est-à-dire le contrôleur du dialogue (DIALOG.C), l'analyseur des événements (EEVENT.C) et un ensemble de serveurs spécialisés, dont le gestionnaire de menu (MMENU.C).

1.1.1 - Le contrôleur de dialogue

DIALOG.C est le module qui contient la boucle principale de traitement des événements. Il contrôle donc le dialogue à un très bas niveau.

Les événements de bas niveau sont extraits, un par un, de la file système qui les enregistre à leur survenance et sont transmis à l'analyseur des événements (EEVENT.C). Celui-ci les lui renverra sous forme d'événement abstrait.

Actuellement, c'est un code qui enrichit la sémantique des événements de bas niveau pour en faire des événements abstraits. Si un événement possède une signification sémantique pour l'application, ce code permet au contrôleur de déterminer le point d'entrée, dans APPLICATION, à utiliser pour prendre en charge l'événement abstrait. Tout point d'entrée correspond en fait à une fonction sémantique de l'application. Cette dernière se trouvera donc définie dans le module APPLI.C, comme indiqué au paragraphe 1.2.1.

DIALOG.C effectue également les opérations d'initialisation et de clôture du système d'exploitation, de l'application et du dialogue, ces opérations devant être effectuées quelle que soit l'application spécifique implémentée.

1.1.2 - L'analyseur des événements

EEVENT.C correspond au module analyseur des événements.

Il reçoit un événement de bas niveau du contrôleur de dialogue (programme principal) qu'il lui renverra sous forme d'événement abstrait après avoir éventuellement déclenché les traitements spécifiques à la gestion de l'interface qu'il est normal d'exécuter à la réception de l'événement considéré.

Le module contient actuellement trois fonctions : EVPROCESS(...), MOUSEEVT(...) et KEYEVT(...).

EVPROCESS(...) identifie le type d'événement à traiter : pression ou relâchement du bouton de la souris; frappe, maintien ou relâchement d'une ou de plusieurs touches du clavier, insertion d'une disquette dans le lecteur de disquettes, événement défini par l'application...

Dans les deux premiers cas (événement souris ou clavier), cette procédure fait appel à une autre procédure du module, respectivement MOUSEEVT(...) ou KEYEVT(...), qui permet d'affiner l'analyse de l'événement de bas niveau.

En fonction de cette analyse, EVPROCESS(...) soit déclenche, éventuellement via MOUSEEVT(...) ou KEYEVT(...), un serveur spécialisé dans l'interprétation et le traitement de l'événement reçu, soit renvoit directement au contrôleur de dialogue l'événement analysé après l'avoir enrichi du code qui doit lui être attribué compte tenu de sa nature.

1.1.3 - Un exemple de serveur : le gestionnaire de menu

Comme nous l'avons déjà souligné, le gestionnaire de menu présenté ici permet à la fois de donner un exemple de serveur spécialisé et d'illustrer la manipulation d'un objet interactif, en l'occurence d'un menu.

MMENU.C est en fait une librairie de procédures qui permet la gestion et le contrôle d'objets interactifs du type menu.

En particulier, la procédure MENUEVT( ) de ce module gère tout événement utilisateur de bas niveau qui s'est produit dans un menu. Elle utilise actuellement une table de correspondance élémentaire "LAUNCHSEM", définie par le concepteur de l'interface de l'application, afin d'identifier quelle fonction de l'application est à lancer pour exécuter l'action sélectionnée par l'utilisateur par l'intermédiaire d'un menu.

Ce module devrait, dans une version future, contenir d'autres procédures permettant entre autres choses la création et la mise-à-jour de menus par l'intermédiaire d'un fichier de ressources et non plus uniquement par programmation, des procédures facilitant la modification des menus présentés à l'utilisateur en cours d'exécution et en fonction du contexte d'exécution.

1.2 - Les fonctionnalités de l'application (APPLICATION)

Le module APPLI.C contient l'ensemble des fonctions du composant APPLICATION.

Au niveau des fonctions définies pour implémenter les fonctionnalités de l'interface et des traitements de l'application, une remarque importante est à faire.

Pour respecter le principe de séparation entre les composants APPLICATION et INTERFACE du modèle APEX, aucune fonction de APPLICATION ne peut être interactive. A notre niveau, cela signifie qu'elle ne peut pas prendre directement en charge un événement de bas niveau généré par l'utilisateur, c'est-à-dire extraire elle-même, en utilisant la fonction "GetNextEvent" du Toolbox, un événement de la

file système des événements, car tous les événements doivent passer par la boucle de traitement des événements localisée dans le contrôleur de dialogue.

Dès lors, la découpe en fonctions d'un traitement de l'application doit s'effectuer de telle sorte qu'une fonction n'ait jamais besoin d'un complément d'information en provenance de l'utilisateur pour s'exécuter.

Quand cela s'avère nécessaire au niveau d'un traitement, une de ses fonctions doit donc générer un événement initialement défini par l'application pour demander le complément d'information qui est nécessaire à la poursuite de l'exécution du traitement. Cet événement est reçu par le contrôleur du dialogue et transmis à l'analyseur des événements qui le transforme en événement abstrait en lui attribuant un code spécifique (du type "APPLI-DEFINED-EVENT"). Lorsque le complément d'information est constitué sur base des actions effectuées par l'utilisateur, la fonction du traitement qui est associée à cet événement abstrait est lancée par le contrôleur du dialogue, après identification de cette fonction d'après le code et la description de l'événement abstrait.

Dans le cadre du programme d'application que nous avons implémenté, la distinction est claire entre les modules de l'application destinés à la gestion du dialogue et les procédures de traitement de l'application fonctionnelle.

En effet, le programme de traitement est un programme d'intelligence artificielle qui est exécuté sur un ordinateur de type VAX. Aucune de ces procédures n'est donc prise en charge par le Macintosh. Seul l'interface de l'application est définie sur Macintosh.

1.2.1 - Le dialogue de l'application

Du côté des fonctions qui concernent la gestion sémantique de l'interface de l'application, on trouve encore deux groupes distincts dont quelques exemples se trouvent dans la version de APPLI.C présentée ici.

D'une part, il existe dans APPLI.C des fonctions chargées de prendre en charge les événements abstraits ayant une sémantique pour l'application, tels que les événements enrichis par le code "INCLOSE", "INCONTENT", "COMMAND", "CMDKEY",...

Ces points d'entrée permettent au contrôleur de dialogue d'identifier et de lancer une fonction de prise en charge de ces événements.

Il s'agit notamment des fonctions apclosew ( ), apincontent ( ), apdocommand ( ), apcmdkey ( ),...

D'autre part, on trouve dans APPLI.C des fonctions qui implémentent la sémantique de l'interface de l'application. Il s'agit par exemple de fonctions telles que b9600( ), online( ) ou local( ).

B9600( ) a pour but de fixer la vitesse de transmission de la ligne série connectant le Macintosh au VAX à 9600 bauds.

ONLINE( )permet d'ouvrir la ligne série d'après les valeurs des paramètres de contrôle de transmission définies par l'utilisateur tandis que la fonction LOCAL( ) signale le retour du terminal en mode local, c'est-à-dire non connecté au VAX.

Il est à noter que ces fonctions sont déclenchées à la demande de l'utilisateur puisque c'est ce dernier qui contrôle l'exécution de l'application.

Ainsi quand l'utilisateur veut passer du mode local en mode connecté, il le fait par sélection dans le menu sémantique de l'application qui lui permet de choisir le mode de définition du terminal. Cet événement de bas niveau est enrichi par le gestionnaire de menu du code "COMMAND", de même qu'un pointeur vers la fonction implémentant le passage du terminal dans le mode sélectionné par l'utilisateur est positionné. En l'occurence, il s'agit d'un pointeur vers la fonction online ( ). Quand le contrôleur de dialogue reçoit l'événement abstrait de type "COMMAND", il déclenche la fonction apdocommand( ), en lui transmettant le pointeur vers la fonction online( ). La fonction apdocommand( ) lance alors la fonction online( ) afin que le traitement sémantique de l'application associé à l'événement utilisateur initial soit exécuté.

1.2.2 - Les traitements de l'application

Comme nous l'avons déjà dit, aucun traitement fonctionnel de l'application traitée n'est implémenté sur Macintosh. Nous ne pouvons donc pas présenter de procédure de l'application fonctionnelle car nous n'avons pas eu à nous préoccuper de la définition de ces fonctions. Seuls les résultats de l'exécution du programme d'application fonctionnelle étaient transmis à l'interface de l'application afin d'être visualisés à l'écran.

Notons toutefois que pour prendre en charge ces résultats, nous avons dû spécifier et implémenter un serveur spécialisé dans la gestion de la ligne série utilisée pour établir la liaison de communication entre le VAX et le Macintosh.

Remarquons également que dans le cas où l'ensemble de l'application fonctionnelle aurait été développée sur Macintosh, tous les traitements non interactifs seraient définis dans APPLI.C en tant que fonctions de l'application uniquement exécutables par l'intermédiaire de points d'entrée définis dans APPLICATION et reconnus par le contrôleur de dialogue.

1.3 - Le code des programmes des différents modules

Nous présentons maintenant le code des programmes des différents modules de l'architecture d'implémentation de l'application réalisée présentés précédemment.

On trouvera dans l'ordre :

- DIALOG.C, le contrôleur de dialogue,
- EEVENT.C, l'analyseur des événements,
- MMENU.H, la déclaration des données pour le driver de menu,
- MMENU.C, le gestionnaire de menu,
- APPLI.C, les fonctions de l'interface de l'application.

```
/*
*****************************************************************************

                              DIALOG.C

This module controls the dialogue at a very low level.
It performs initialization and termination of the global running
application.
It owns the main loop of events.

NOTATION used for identifiers:

    - lower case identifiers are defined in this program.
    - upper case identifiers are constants in this program.
    - mixed case identifiers are identifiers imported from the Mac Toolbox.

Written by Joelle Coutaz, June 1986
Modified by Muriel Chandelon, December 1986

*****************************************************************************
*/

#include    ":.h:wwin.h"          /* to use the extended lib of windows */
#include    ":.h:goodc.h"

/*
GLOBALS
-------
*/

int             go;             /* to control the main loop */
EventRecord     theevent;       /* to receive event from the toolbox */
int             code;           /* used to classify events */
int             (**actptr)();   /* application routine to perform
                                   depending on user's interaction */
char            hitkey;         /* last hit key having a semantics */

/*
--------------------------------- stopexe() ------------------------------
                          To quit the application normaly
*/

stopexe()
{
go = FALSE;
}

/*
--------------------------------- The MAIN LOOP --------------------------
*/

main()
{

/*
Operating system,then application, then dialogue dependent prologues
--------------------------------------------------------------------
*/
osbegin();          /* Os initialization */

apbegin();          /* Initialize the running application "appliname"
                       In a further version, this name should be read,
                       interactively or not, in a variable string;     */

go = TRUE;
```

```
/*
Main Loop of events
-------------------

*/

while (go)

    begin
    evprocess(&theevent, &code, actptr, &hitkey);
    switch (code)
        begin
        case INCONTENT:
            /*
            dispatch to an application routine depending on the window
            which content region was clicked into
            */
            apincontent(FrontWindow(),&theevent);
            break;

        case INCLOSE:
            /*
            perform any semantic treatment linked to the closing of the
            front window
            */
            apclosew(FrontWindow());
            break;

        case COMMAND:
            /* dispatch to an application routine corresponding to the
            user's choice of any semantic action through a menu or a
            significant key
            */
            apdocommand(actptr);
            break;

        case KEY:
            /* dispatch to an application routine depending on the key
            which was hit
            */
            apkey(&hitkey);
            break;

        case CMDKEY:
            /* dispatch to an application routine depending on the command
            key which were hit
            */
            apcmdkey(&hitkey);
            break;

        case NOOPER:

        default:
            break;

        end /* switch code */

    end /* while go */

/*
Application, dialogue and system dependent epilogues
----------------------------------------------------
*/

apend();
osend();

}
```

```
*****************************************************************************

                    EEVENT.C : The EXTENDED EVENT MANAGER

This module is a library to process the low level events from the toolbox
event manager.

NOTATION used for identifiers:

    - lower case identifiers are defined in this program.
    - upper case identifiers are constants in this program.
    - mixed case identifiers are identifiers imported from the Mac Toolbox.

Written by Joelle Coutaz, June 1986
Modified by Muriel Chandelon, December 1986

Note :
  - There is a global sytem event mask that controls which event types get
    posted into the eveent queue.  Only event types corresponding to bits
    set in the system event mask are posted; all others are ignored.  When
    the system starts up, the system event mask is set to post all except
    key-up event - thats is, it is initialized to everyEvent - keyUpMask.
    If you want to make key-up events meaningful for applications, use the
    OS Event Manager procedure SetEventMask to set the system event mask to
    everyEvent (or anything else, if needed).

*****************************************************************************
*/

#include ":.h:wwin.h"
#include ":.h:goodc.h"


/*
----------------------------------mouseevt ---------------------------------
    Interprets Mouse events. It is a classic procedure that any application
    programmer on top of the Mac has to write.

    Parms in:
        - theevent : pointer to the event
    returns:
        - codeptr : to classify the type of event
        - funcptr : pointer to any procedure to launch according to the
                    selection of a menu item by clicking on it
    Remarks: the type "WindowPtr" is a pointer to a grafport (not to a
    window record)
*/

mouseevt(theevent, codeptr, funcptr)

EventRecord *theevent;
int         *codeptr;
int         (**funcptr)();

{
int             button;             /* button state (up, down,...) */
Point           *where;             /* the point (in global coordinates)
                                       where the mouse is located */

int             wheremouse;
WindowPtr       whichwindow;
Point           localwhere;
int             whereincontrol;
int             controlval;
ControlHandle   whichcontrol;

/*
Initialization
*/
button = theevent->what;
```

```
where = &theevent->where;

/*
in general, the event doesn't concern the appli
*/
*codeptr = NOOPER;
funcptr = 0;

/*
Locate the mouse location
*/
wheremouse = FindWindow(*where, &whichwindow);

switch(wheremouse)

    begin
    case inDesk:
        break;

    case inMenuBar:
        /*
        call the menu driver to find out the user's choice and process it
        (Note : A mouseUp event is taken into account through menuevt()
        by a call to MenuSelect() )
        */
        menuevt(theevent, codeptr, funcptr);
        break;

    case inSysWindow:
        break;

    case inGrow:
    case inContent:
    case inDrag:
    case inGoAway:
        /*
        call the window driver to interpret the mouse event that occured
        in a window
        */
        wevent(whichwindow, wheremouse, where, button, codeptr);
        break;

    default:
        break;
    end /* switch wheremouse */
}


/*
---------------------------------keyevt ----------------------------------

    Interprets Key events.

    This treatment is temporary until a "higher level" key handling can be
    defined (which would manage different key definitions according to the
    current state of the application, the user's preferences,...)

    Parms in:
        - theevent : pointer to the event

    returns:
        - codeptr : to classify the type of event (either COMMAND or KEY)
        - charhit : long word which contains :
                    .the ASCII character code
                    .and the key code of the hit key
                     (undefined if *codeptr = COMMAND)
        - funcptr : pointer to any procedure to launch according to the
                    selection of a menu item by using the command key
                    (undefined if *codeptr = KEY)
```

```
*/

keyevt(theevent, codeptr, charhit, funcptr)

EventRecord *theevent;
int         *codeptr;
char        *charhit;
int         (**funcptr)();

{
/*
determine wether or not the command key was also held down
*/
if (theevent->modifiers & cmdKey)
    /*
    respond to a command-character key event.
    Up to now, it only matches menu commands (but, in further versions, it
    should be extended to match actions on windows, ... depending on the
    user's preferences)
    */
    begin
        menuevt(theevent, codeptr, funcptr);
        switch (*codeptr)
        begin
        case COMMAND:
            /*
            the command-character key event is equivalent to a menu item
            significant to the application
            */
            break;

        case NOTAPPLI:
            /*
            the command-character key event is equivalent to a menu item
            corresponding to a command for a driver.
            Should launch this action using the procedure pointer funcptr.
            Not implemented yet.
            */
            break;

        case NOOPER:
            /*
            the command-character key event is not equivalent to any menu
            item. It is for the Application.
            */
            *codeptr = CMDKEY;
            *charhit = (theevent->message);
            break;

        default:
            break;

        end /* switch *codeptr */


    end  /* if (theevent->modifiers & cmdKey) */

else
    /*
    the user has not hit the command-character key

    This is a temporary solution : here, we should determine if the hit key
    has a special meaning for the application ( it is not efficient to
    return every hit key to apkey() )
    */
    begin
        *codeptr = KEY;
        *charhit = (theevent->message);

    end  /* else (theevent->modifiers & cmdKey) */
```

```
}

/*
-------------------------------- evprocess ------------------------------
    Processes events from the toolbox event manager.
    Parms out:
        - the event returned by the toolbox event manager
        - a code that classifies the type of event.
*/

evprocess(theevent, codeptr, funcptr, charhit)

EventRecord *theevent;
int         *codeptr;
int         (**funcptr)();
char        *charhit;

{
WindowPtr   whichwindow;

*codeptr = NOOPER;  /* In general, the event, does not concern the
                       application*/

if (GetNextEvent(everyEvent, theevent))
    begin

    if (IsDialogEvent(theevent))

    /*
    theevent must be handled as part of a modeless dialog
    */
        begin
        /*
        *codeptr == NOOPER
        dialogevt(theevent, codeptr);
        not implemented yet
        */
        end

    else

        /*
        theevent is not part of a dialog box
        */
        switch (theevent->what)

        begin
        case keyDown:
            /*
            theevent is a key event which must be handled according
            to the hit key(s)
            */
            /* *codeptr == NOOPER */
            keyevt(theevent, codeptr, charhit, funcptr);
            break;

        case mouseUp:                   /* Mouse up events */
        case mouseDown:                 /* Mouse down events */
            /*
            theevent is a mouse event which must be handled according
            to the mouse location
            */
            /* *codeptr == NOOPER */
            mouseevt(theevent, codeptr, funcptr);
            break;

        case autoKey:
            break;          /* should be tested as key-down and key-up */
```

```
/*****************************************************************************
                        MMENU.H

            THIS INCLUDE FILE CONTAINS THE MAPPING TABLE WITH
            THE SEMANTICS OF THE APPLICATION.
            THIS FILE SHOULD BE FILLED WITH THE EXTERNAL FUNCTION
            OF THE SPECIFIC APPLICATION ACCORDING TO THIS MODEL

*****************************************************************************/
/*
--------------------Constants for menu ressource file----------------------
*/
#define MNUMBER         4   /* actual number of application defined menus */
#define MAPPDISPL       240 /* menu id displacement used for launchsem */
#define MFIRSTAPPMID    300 /* FIRST APPLICATION MENU ID */
#define MCHOICE         20  /* maximum number of choices (items) per menu */
#define APPLEMENU   1   /* menu ID for desk accessory menu */
#define FILEMENU    2   /* menu ID for standard file menu */
#define EDITMENU    3   /* menu ID for standard edit menu */
/*
The application menu resources ID are 300,320,etc .........
*/


extern int quit();
extern int send();
extern int receive();
extern int configuration();
extern int b9600();
extern int b7200();
extern int b4800();
extern int b3600();
extern int b2400();
extern int b1800();
extern int b1200();
extern int b600();
extern int b300();
extern int b75_1200();
extern int b1200_75();
extern int modem();
extern int printer();
extern int online();
extern int local();
extern int bits7();
extern int bits8();
extern int even();
extern int odd();
extern int none();
extern int stop1();
extern int stop2();
extern int xon_xoff();
extern int tabs();


/*
--------------------THE LAUNCH TABLE----------------------------------------
Allow a simple and flexible mapping with user's selection,especially the
menu.

HOW LAUNCH IS MANAGED : Add the maximum number of choices to the menu resouce
ID each time you want to define a new menu.  The access to the associated
function is automatically computed with the displacement,the menu ID,and
the choice.

Constraints for the future version including the new interactive objects :
            - no parameter or every profile the same
            - every profile should be declared in the include file
              of the application.
```

```
----------------------------------------------------------------------------
*/
int  (*launchsem[])() = {0,
  0, 0, 0, 0, 0, 0, 0, 0, &quit, 0,0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    /* Beginning of second file menu */
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0,0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    /* Beginning of Third File menu */
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0,0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     /* Beginning of fourth File menu  (first application menu ) */
  &send, &receive, 0, 0, 0, 0, 0, 0, 0, 0,0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    /* Fifth menu  (2nd appli menu) */
  &b9600, &b7200, &b4800, &b3600, &b2400, &b1800, &b1200, &b600, &b300, 0,
    /* sixth menu  (3rd appli menu) */
  &modem, &printer, 0, 0, 0, 0, 0, 0, 0, 0,0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    /* Seventh Menu */
  &online, &local, 0, &bits7, &bits8, 0, &even, &odd, &none, 0,&stop1, &stop2,
    /* eigth Menu */
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0,0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0,0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0,0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    };
```

```
/*
*****************************************************************************
                    MMENU.C : EXTENDED MENU PACKAGE

   This module is a library of procedures that hide the low level details
about menus.

In particular, it takes care of selection of actions within menus. For
doing so, it uses the mapping table ( launchsem ) defined by the applica-
tion ( in mmenu.h ).  This table defines which application routine must be
called to perform the user's choice.

HOW LAUNCH IS MANAGED : Add the maximum number of choices to the menu
resource ID each time you want to define a new menu.  The access to the
associated function is automatically computed with the displacement,the
menu ID,and the choice.

NOTATION used for identifiers:

    - lower case identifiers are defined in this program.
    - upper case identifiers are constants in this program.
    - mixed case identifiers are identifiers imported from the Mac Toolbox.

Written by Muriel Chandelon, December 1986

Note :
  - To easily adapt the interface to different application contexts, we can
    use resource files and define
    . menus (accessed through GetNewMenu),
    . menubars (accessed through GetNewMBar),
    . ...

*****************************************************************************
*/

/*
INCLUDE FILES
-------------
*/
#include ":.h:wwin.h"
#include ":.h:goodc.h"
#include ":.h:oos.h"
#include ":.h:mmenu.h"


/*
---------------------------------menuevt ---------------------------------

       Return a pointer to the routine to perform depending on an action
       selected by the user .

       For application significant actions, mapping  is made via a "Launch
       table" which entries are defined in mmenu.h; for actions of
       standard menus, mapping is made via an internal table ( !!! still to
       define !!! )

       IN  :  whichevent : pointer to the event :
                . whichevent->what indicates whether the activation of a
                      menu has been made by using the mouse or the keybord,
                . whichevent->where or whichevent->message indicates which
                      item of which menu has been selected, if any;

       OUT :  - codeptr : to classify the type of event,
              - funcptr : pointer to the routine which should be launched
                          to perform the action corresponding to the user's
                          selected item
-----------------------------------------------------------------------------
*/
```

```
menuevt (whichevent, codeptr, funcptr)

EventRecord *whichevent;
int         *codeptr;
int         (**funcptr)();

{
long    menuresult;
int     menuid, itemnumber;
int     tabenter;           /* Entry number in the lauchsem table */

/*
Let the menu manager perform the user's actions (move of the mouse, hit of
a command-character key) till it returns the user's choice in menuresult.
*/

switch (whichevent->what)

    begin
    case keyDown:
        /*
        the user's choice, if any, has been made by using
        a command-character key
        */
        menuresult = MenuKey((whichevent->message & charCodeMask));
        break;

    case mouseDown:
        /*
        the user's choice, if any, has been made by clicking on a menu item
        */
        menuresult = MenuSelect(whichevent->where);
        break;

    end /* end of switch whichevent->what */

/*
Extract the two words of information of menuresult
(If no choice was made, MenuSelect and MenuKey return 0 in the high-order
word of menuresult and the low-order word is undefined)
*/

/*


IDnumber of the selected menu in the resource file
*/
menuid = HiWord(menuresult);

/*
number of the choice in the selected menu
*/
itemnumber = LoWord(menuresult);
```

```
/*
Which menu has been selected (if one) ?
*/

switch (menuid)

    begin
    case APPLEMENU :
        /*
        not implemented yet
        */
        *codeptr = NOOPER;
        *funcptr = 0;
        break;

    case FILEMENU :
    case EDITMENU :

        /*
        correspond to a standard menu of the interface as EDIT, FILE, ...
        not implemented yet
        */
        *codeptr = NOTAPPLI;
        *funcptr = 0 ;      /* ProcPtr to a standard routine of a driver */
        break;

    case 0 :
        /*
        either no selected item or invalid command-character key
        */
        *codeptr = NOOPER;
        *funcptr = 0;
        break;

    default :
        /*
        the user has selected an action significant to the application
        */

        *codeptr = COMMAND,

        /*
        compute the table entry
        */
        tabenter = menuid - MAPPDISPL + itemnumber;

        /*
        access to the pointer to the associated application routine
        */
        *funcptr = launchsem[tabenter];
        break;

    end /* end of switch menuid */


/*
the head of the menu must be unhighlight when the action is performed
*/
HiliteMenu(0); }
```

```
/*****************************************************************************
                                APPLI.C

    THIS SHOULD BE THE SEMANTICAL PART OF THE APPLICATION
    IN OUR ARCHITECTURE MODEL.

    It is supposed to emulate a VT100 terminal to use it as a graphical
    terminal for the VAX.

            Written by Muriel Chandelon and Genevieve Warnant
            during January 1987.

*****************************************************************************/

#include ":.h:wwin.h"
#include ":.h:goodc.h"
#include ":.h:oos.h"
#include ":.h:ssd.h"
#include ":.h:qdprof.h"
#include ":.h:appli.h"
#include ":.h:applisynt.h"
#include <:Mac #includes:DialogMgr.h> /* to manipulate Mac level dialog
                                         windows */
#include <:Mac #includes:FontMgr.h>   /* to manipulate Mac level font */

/*
ACCESS to WINDOWS
-----------------
*/
WindowPtr       windows[WMAX];  /* window pointers for the application */
int             wstatus[WMAX];

/*
APPLICATION GLOBALS
-------------------
*/

int  baud[] = {0, baud9600, baud7200, baud4800, baud3600, baud2400, baud1800,
                 baud1200, baud600, baud300, 0, baud75_1200, baud1200_75};

int  config[] = {0, 0, 0, 0, data7, data8, 0, evenParity, oddParity,
                noParity, 0, stop10, stop20};
/*
--------------------RESSOURCE ACCESS-------------------------------------
*/
MenuHandle menu;
Str255     resfname = "\pappli.rsrc";   /* name of the associated ressource
                                           file */

/*
---------------------CURRENT CONFIGURATION--------------------------------
*/
int     curbaud;    /* current baud rate item number */
int     curport;    /* current port item number */
int     curmode;    /* current mode (on line - local) item number */
int     curdatab;   /* current number of data bits per byte item number */
int     curparity;  /* current parity item number */
int     curstopb;   /* current number of stop bits item number */
Boolean curxonxoff; /* current flow control item number */
Boolean curtabs;    /* current tabs item number */

/*
-----------------------------quit-------------------------------------------
```

```
*/
quit()
{
stopexe();
}


/*
----------------------------b9600----------------------------------------
*/
b9600()
{
CheckItem(GetMHandle(320), curbaud, FALSE);
CheckItem(GetMHandle(320), 1, TRUE);
curbaud = 1;
}

/*
----------------------------online---------------------------------------
*/
online()
{
CheckItem(GetMHandle(360), curmode, FALSE);
CheckItem(GetMHandle(360), 1, TRUE);
curmode = 1;

sdopen(curport, IN, baud[curbaud], config[curstopb], config[curparity],
        config[curdatab]);
}

/*
----------------------------local----------------------------------------
*/
local()
{
sdclose(curport);
CheckItem(GetMHandle(360), curmode, FALSE);
CheckItem(GetMHandle(360), 2, TRUE);
curmode = 2;
}

/*
*****************************************************************************
*/

/*
---------------------------------apkey ----------------------------------
      Analyze a hit key.
      Launch any treatement which should be performed if hitting this key
      has a semantics for the application.

      IN :     hitkey   <pointer to the hit key>
-------------------------------------------------------------------------
*/
apkey (hitkey)

char  *hitkey;

{
if ((*hitkey & charCodeMask) == 'q')
  /*
  termination condition
  */
  /*
  This condition is not global enough : the syntactical conditions of
  termination of an application should be defined and accessed through a
  kind of "specification file"
  */
  begin
  end
```

```
else
  /*
  the hit key does not correspond to the terminal condition
  */
  begin
  /*
  should here launch any routine to perform according to a hit key which
  does not correspond to a termination condition
  */
  end;
}

/*
---------------------------------apcmdkey ---------------------------------
       Analyze the hit key used with the command key.
       Launch any treatement which should be performed if hitting this key
       has a semantics for the application.

       IN :     hitkey   <pointer to the hit key>
-----------------------------------------------------------------------------
*/
apcmdkey (hitkey)

char  *hitkey;

{
if ((*hitkey & charCodeMask) == 'q')
  /*
    example
  */
  begin
  end

else
  /*
  the hit key does not correspond to the terminal condition
  */
  begin
  /*
  should here launch any routine to perform according to a hit key which
  does not correspond to a termination condition
  */
  end;
}
/*
---------------------------------apdocommand -------------------------------
       Launch the  action corresponding to the user's choice.

       IN :     funcptr   <pointer to an application routine >
-----------------------------------------------------------------------------
*/
apdocommand (funcptr)

int   (**funcptr)();

{
if (*funcptr != 0)
 (**funcptr)();
}

/*
--------------------------------- apincontent ------------------------------

    Application dependent stuffs when the window has been made active by
    a mouse click down in the content region.
    It acts as a dispatcher to the appropriate routine according to the
    clicked window.

    IN : whichwindow the window where the click happened
```

```
        whichevent   the type of the event
-----------------------------------------------------------------------
*/
apincontent(whichwindow, whichevent)
WindowPtr whichwindow;
EventRecord *whichevent;

{
switch (LoWord(GetWRefCon(whichwindow))) /* application reference for
                                            windows*/
{
case 0 :/*Application graphic window*/
/*
        wprint(windows[APPLIW], "\pbonjour"); /* test of the application
                                                 window */
*/
        break;

default :/*unknown windows*/
        return(FALSE);
}
return (TRUE);
}


/*
--------------------------------- apclosew ------------------------------

    application dependent stuffs when the goaway region of a window
    has been selected by the user.
*/

apclosew(whichwindow)
WindowPtr whichwindow;

{
switch (LoWord(GetWRefCon(whichwindow))) /* application reference for
                                            windows*/
{
case 0 :/*Application graphic window*/
        wstatus[APPLIW] = CLOSEW;
        break;

default :/*unknown windows*/
        break;
}

/*
*****************************************************************************
*/

/*
------------------------------- apbegin ----------------------------------
    Initializations specific to the application
*/
apbegin()

{
short flags;
int dialflags;
int   i;
int resid;

OpenResFile(resfname);  /*permit to get the resource file information*/

/*
1-Initialize the menus from the resource file and put them in the menubar.
-------------------------------------------------------------------------
*/
/* STANDARD CASES :get whith ressource ID*/
```

```
   menu = GetMenu(APPLEMENU);
   AddResMenu(menu,'DRVR'); /* loading of desk accessories as items
                                    of APPLEMENU */
   InsertMenu(menu,0);/*insertion in the menubar*/

   menu = GetMenu(FILEMENU);
   InsertMenu(menu,0);/*insertion in the menubar*/

   menu = GetMenu(EDITMENU);
   InsertMenu(menu,0);/*insertion in the menubar*/

/* APPLICATION CASES : Compute the ressource ID*/
   resid = MFIRSTAPPMID ;
   for (i = 1;i <= MNUMBER;i++)  /*insertion in the menubar*/
   {
   menu = GetMenu(resid);/* User's menus*/
   InsertMenu(menu,0);/*insertion in the menubar*/
   resid += MCHOICE;/*Increment of max choice number*/
   }
   DrawMenuBar();/*display of the menubar*/

/*
2- Create the windows that are resizable, movable, etc ...
-------------------------------------------------------
*/

flags = WISRESIZABLE | WISMOVABLE | WISVISIBLE | WISDELETABLE;

windows[APPLIW] = wcreate(5,40 ,480,260, "\pVT100 WINDOW", flags,
                          documentProc, 0);

wstatus[APPLIW] = OPENW;


/*
3- init some global variables used in the application
-------------------------------------------------------
*/
curbaud = 1;     /* initialization baud rate */
CheckItem(GetMHandle(320), 1, TRUE);
curport = 1;     /* initialization port */
CheckItem(GetMHandle(340), 1, TRUE);
curmode = 2;     /* initialization mode (on line - local) */
CheckItem(GetMHandle(360), 2, TRUE);
curdatab = 5;    /* initialization number of data bits per byte */
CheckItem(GetMHandle(360), 5, TRUE);
curparity = 9;   /* initialization parity */
CheckItem(GetMHandle(360), 9, TRUE);
curstopb = 11;   /* initialization number of stop bits */
CheckItem(GetMHandle(360), 11, TRUE);
curxonxoff = FALSE; /* Initialization flow control */
CheckItem(GetMHandle(360), 14, FALSE);
curtabs = FALSE;     /* initialization tabs */
CheckItem(GetMHandle(360), 16, FALSE);

}

/*
------------------------------- apend -------------------------------------
    Called when the application has to shut down
*/
apend()
{
}

}
```

2 - Avantages et insuffisances (extensions)

Le modèle APEX présente l'avantage essentiel de fournir "une architecture logicielle saine et un noyau de gestion du dialogue réutilisable et extensible." [Coutaz 86-3]

Le noyau peut être facilement enrichi par l'ajout de nouveaux "drivers" ou de nouvelles fonctionnalités pour les drivers existants. C'est d'ailleurs ce que nous avons fait dans le cadre de l'application que nous implémentions, d'une part, en définissant un gestionnaire de menu indépendant, dont la description est donnée au paragraphe 1.1.3 de cette annexe, ainsi qu'un serveur spécialisé dans la gestion d'une ligne série permettant la liaison du Macintosh avec d'autres machines, directement ou via un réseau, local ou non, de communication; d'autre part, en complétant les fonctions des modules contrôleur de dialogue et analyseur des événements précédemment définis (par exemple, en introduisant les procédures KEYEVT(...) et MOUSEEVT (...) dans EEVENT.C)

Le protocole d'échange entre APPLICATION et INTERFACE peut aussi être affiné par la définition de nouveaux points d'entrée dans APPLICATION. Dans la nouvelle version d'APEX que nous avons mise au point, nous avons notamment introduit la notion d'événement défini par l'application ("APPLI-DEFINED-EVENT") à laquelle fut associée la spécification de nouveaux points d'entrée. Par exemple, le point d'entrée "ENDAPPLI" a été ajouté pour permettre la prise en charge d'un événement défini par l'application destiné à demander la clotûre de l'application. De son côté, le point d'entrée "CMDKEY" permet d'isoler les traitements de l'application liés à une commande de l'application sélectionnée à partir de la frappe d'une ou plusieurs touches du clavier.

En fonction de l'état d'implémentation dans lequel nous l'avons utilisé et du niveau de développement auquel nous sommes arrivés à la fin de notre stage, certaines extensions restaient cependant encore à prévoir.

On peut par exemple signaler :

- le développement d'une base d'objets interactifs simples et composés;
- le développement et la complétion des drivers actuellement disponibles;

- l'intégration au sein du noyau d'APEX d'un véritable dialogueur fonctionnant à partir de fonctionnalités de haut niveau;

- l'intégration d'APEX dans un système d'exploitation pour en augmenter l'efficacité et pouvoir l'utiliser avec n'importe quel langage évolué.