Gestion Automatique du Dialogue Homme-Machine à partir de Spécifications Conceptuelles

Jean-Claude Tarby

Année 1993

T H E S E

présentée en vue de l'obtention du

Doctorat de l'UNIVERSITE TOULOUSE I

Spécialité : ***INFORMATIQUE***

par

Jean-Claude TARBY

Gestion Automatique du Dialogue Homme-Machine à partir de Spécifications Conceptuelles

Soutenue le 20 septembre 1993 à l'Université Toulouse I devant le jury composé de :

M.	**René CAUBET**	*Président*
Mme.	**Marie-France BARTHET**	*Directeur*
Mme.	**Joëlle COUTAZ**	*Rapporteurs*
Mme.	**Jocelyne NANARD**	
Mlle.	**Isabelle PETOUD**	*Examinateur*

Laboratoire d'accueil : L.I.S. - Université Toulouse I - Place Anatole France - 31042 Toulouse Cedex

Résumé

Cette thèse s'inscrit dans le domaine des Interfaces Homme-Machine (IHM). Elle s'articule autour de deux thèmes principaux qui sont la Spécification et la Gestion Automatique du Dialogue Homme-Machine.

Le travail présenté est basé sur la méthode Diane+ conçue initialement pour la spécification du dialogue homme-machine. Diane+ repose sur la planification hiérarchique et intègre le niveau de l'utilisateur. Elle est utilisée pour spécifier la répartition des tâches entre l'homme et la machine tout en laissant une latitude décisionnelle à l'utilisateur. Cette spécification utilise un formalisme simple et concis prenant en compte les répartitions les plus complexes. Ce formalisme permet, à partir des spécifications, de :

- générer l'interface homme-machine en intégrant des règles d'ergonomie générale,
- générer une partie du code des traitements,
- gérer automatiquement la dynamique de l'application (noyau fonctionnel et interface homme-machine),
- gérer automatiquement l'aide d'utilisation, l'aide fonctionnelle étant quant à elle implémentée par le concepteur.

De par les origines de Diane+ et les caractéristiques des IHM, ce travail utilise conjointement le modèle tâche et le modèle objet.

Mots-clés

Interface Homme-Machine, Dialogue Homme-Machine, Aide d'utilisation, Méthode Diane, Modèle Tâche, Modèle Objet, Ergonomie du logiciel, Génération Automatique, Gestion Automatique.

Abstract

This dissertation is about Human-Computer Interfaces (HCI). It is built on two main themes which are the Specification of Human-Computer Dialogue and the Automatic Management of Human-Computer Dialogue.

This work is based on the Diane+ method which was initially created for the Specification of Human-Computer Dialogue. Diane+ is based upon hierarchical planning and integrates the level of users. It is used to specify the tasks distribution between human and computer while leaving a decisionnal freedom to the user. This specification uses a simple and concise formalism which takes the most complex specifications into consideration. This formalism permits, from specifications:

- to generate the Human-Computer Interface including general ergonomics rules,
- to generate a part of processing code,
- to manage automatically the dynamics of the application (functionnal core and Human-Computer Interface),
- to manage automatically the utilisation help. As to functionnal help, it is implemented by the designer.

Because of Diane+'s origins and the Human-Computer Interfaces features, this works uses both task model and object model.

Key words

Human-Computer Interface, Human-Computer Dialogue, Utilisation Help, Diane Method, Task Model, Objet Model, Software Ergonomics, Automatic Generation, Automatic Management.

Remerciements

Je tiens à exprimer ici ma gratitude et mes remerciements à tous ceux qui ont participé activement à ce travail :

Marie-France BARTHET, Professeur à l'Université Toulouse I, qui m'a encadré tout au long de ma thèse et qui, par ses conseils et ses encouragements constants, a permis la concrétisation de cet travail,

Joëlle COUTAZ, Professeur à l'Université Joseph Fourrier à Grenoble, et Jocelyne NANARD, Maître de Conférences à l'Université Montpellier I, qui m'ont fait l'honneur d'être rapporteurs et dont les lectures méticuleuses ont permis des améliorations notables tant sur le fond que sur la forme,

Isabelle PETOUD, Professeur Assistant à l'Université de Lausanne, et René CAUBET, Professeur à l'Université Toulouse III, qui ont accepté de participer au jury,

Les membres du Laboratoire d'Informatique de l'Université TOULOUSE I dont la gentillesse et le dynamisme m'ont permis de bénéficier d'un cadre de travail unique.

Une pensée particulière pour les membres de ma famille sans lesquels ces remerciements n'existeraient pas aujourd'hui.

Sommaire

Chapitre 3 : La Génération Automatique

Partie II

Chapitre 4 : Objectifs de Génération et de Gestion Automatique à partir de Spécifications

Chapitre 5 : La méthode Diane+

Chapitre 6 : Génération et Gestion Automatique à partir de Spécifications

Introduction

Contexte de travail

Les interfaces homme-machine connaissent en ce moment une mutation. Tout au long de ces dernières années nous avons vu apparaître de nombreux produits permettant de réduire la charge de travail des développeurs et des concepteurs. Dans le premier cas nous trouvons par exemple les boîtes à outils et les UIMS (User Interface Management System), dans le second cas nous trouvons les AGL (Atelier de Génie Logiciel). Le changement qui s'opère à l'heure actuelle peut s'expliquer par le fait que les informaticiens demandent davantage à ces produits. Ils savent qu'à présent il est facile de réaliser des maquettes et des prototypes. Ils savent également que beaucoup de produits permettent de développer une même application pour des environnements différents. Leurs demandes s'orientent donc vers des outils plus puissants (en génération, en ergonomie, en utilisabilité, etc.) et permettant de réduire encore plus la charge de travail, tout en augmentant la facilité d'utilisation.

Si l'on retourne quelques années en arrière, plus précisément au début des années 80, on constate que le terme Interface Homme-Machine (abrégé en IHM) était quasiment inconnu. La micro-informatique en était à ses balbutiements et on privilégiait avant tout la validité des applications, c'est-à-dire la réalisation effective des traitements qu'on en attendait. L'arrivée du premier Macintosh a révélé au public l'existence des boîtes à outils (toolbox). Il a également contribué à une petite révolution dans le monde informatique puisqu'il était le premier à utiliser une interface graphique à manipulation directe. Avec lui sont apparus de nombreux termes tels que "icônes", "dossier", "menus", "souris" dont certains existaient déjà depuis de nombreuses années, mais restaient inusités car confinés dans le milieu de la recherche.

Les toolbox ont réduit potentiellement le temps de développement des applications. Nous utilisons le terme "potentiel" car d'un côté elles permettaient de construire des applications qui se basaient sur un standard visuel (fenêtres, boutons, menus, etc.) et comportemental (un clic pour un bouton ou un menu, deux clics pour une icône, etc.), mais d'un autre côté elles demandaient un temps d'adaptation et d'apprentissage non négligeable. De plus, leur utilisation demeurait anarchique puisque seuls des "outils" élémentaires (appelés primitives) étaient fournis aux développeurs. L'étape suivante a contribué à réduire cette latitude créative en enfermant les primitives dans des squelettes d'applications. Les développeurs étaient alors libérés de la gestion de certaines tâches élémentaires telles que la scrutation du clavier ou celle de la souris. Pourtant le gain de production restait faible car il fallait encore écrire complètement le code pour l'interface. Les toolkits ont répondu à ce besoin en fournissant aux concepteurs des possibilités de maquettage et aux développeurs des possibilités de génération de code (uniquement au niveau de l'interface) [MYERS 89]. La dernière étape correspond à l'arrivée des UIMS (User Interface Management System) qui fournissent en plus d'une génération de code plus puissante qu'avec les toolkits (par exemple en générant du code pour un environnement laissé au choix de l'informaticien) des possibilités de prototypage par l'intermédiaire d'éditeurs spécialisés de maquettes et de programmes (C ou C++ le plus souvent).

Axes de réflexion de la thèse

Malgré leur évolution, tous les outils pré-cités possèdent des lacunes identiques qui sont :

- *l'absence d'intégration de l'utilisateur*. Chaque application produite est réalisée pour un utilisateur type correspondant aux spécifications de l'application. Ceci implique deux issues possibles si un utilisateur ne correspond pas à cet idéal : soit il s'adapte bon gré mal gré à l'application, soit il rejette l'application.
- *l'absence d'intégration de méthode de spécification et de conception*. On dispose à l'heure actuelle de nombreuses méthodes de spécification et de conception (orienté-objet ou non), mais aucun outil ne les intègre vraiment complètement. Les concepteurs sont obligés la plupart du temps de faire leur spécification et leur conception, puis l'implémentation, sans qu'il existe vraiment une passerelle informatique entre ces deux stades. Certes les AGL répondent en partie à cette demande, mais uniquement au point de vue des données. Les parties traitements et dialogue homme-machine sont entièrement laissées de côté.
- *l'absence de gestion du dialogue homme-machine*. La réalisation d'applications avec ces outils oblige les informaticiens à programmer entièrement eux-mêmes la gestion du dialogue homme-machine. Certains outils proposent cependant des services pour représenter des enchaînements d'écrans, mais toujours de façon rudimentaire.
- *l'absence d'évaluation*. Chaque application doit subir des évaluations pour être validée (évaluation de la robustesse, de l'utilisabilité, de l'utilité, de la cohérence, etc.). Aucun outil ne propose à l'heure actuelle des évaluations de ce genre. Il est donc encore possible de réaliser des applications totalement inutilisables, inconfortables ou incohérentes.
- *l'absence d'ergonomie*. Pour l'instant cette ergonomie est à la charge des différents intervenants lors du développement d'une application. Plus précisément le concepteur doit intégrer, lors de la spécification et de la conception, les recommandations des ergonomes que le programmeur doit ensuite implémenter. Par ailleurs ces recommandations se répartissent en deux catégories, celles qui sont invariables quelle que soit la nature de l'application et celles qui sont propres à chaque application. K. Y. Lim [LIM 92] remarque avec justesse que malheureusement jusqu'à présent l'ergonomie et l'intégration de l'utilisateur sont prises trop tard et trop peu en compte.

Cette thèse apporte une solution aux trois premières lacunes citées. Nous nous basons pour cela sur le travail de M-F Barthet [BARTHET 86] qui a fourni une réponse au problème de l'intégration de l'utilisateur dans le dialogue homme-machine. Plus précisément, l'absence de spécification et l'absence d'intégration de l'utilisateur sont résolues par l'intermédiaire de la méthode *Diane* [BARTHET 88] qui *permet de spécifier la répartition des tâches entre l'homme et la machine*. Diane intervient après que les données et les traitements aient été clairement définis. Elle ne rajoute ni traitement ni donnée à l'application spécifiée. Diane est orienté-tâche ; elle décompose une application en fonction des buts à atteindre, ces buts étant composés de procédures, elles-mêmes composées d'opérations. A chaque but correspondent des procédures qui sont définies en fonction des postes de travail et du niveau de l'utilisateur (débutant, expert, avec un niveau de confidentialité fort ou faible, etc.). Une procédure décrit les opérations et les enchaînements entre opérations permettant d'atteindre le but associé. Les opérations permettent de distinguer les traitements réalisés par la machine de ceux réalisés avec la complicité de l'utilisateur.
L'absence de gestion du dialogue homme-machine est également résolue par l'intermédiaire de Diane. La version originelle de cette méthode n'avait pour objectif que la spécification. Nous avons donc dû ajouter et modifier certaines notions afin de prendre en compte tous les

cas possibles de répartition de tâches en vue de la génération et de la gestion automatique. Nous emploierons dorénavant le nom **Diane+** pour désigner cette nouvelle version. Les modifications apportées concernent principalement :

- *la révision de la notion de précédence qui lie les opérations*,
- *l'ajout de la notion de contraintes sur les imbrications d'opérations*,
- *l'ajout d'attributs décrivant les opérations*.

L'essentiel de notre travail porte sur la génération et la gestion automatique, à partir des spécifications, de l'interface homme-machine et de l'aide d'utilisation. Diane+ est une des rares méthodes à l'heure actuelle proposant cette caractéristique qui, nous le rappelons, n'existe dans aucun outil de construction d'IHM du marché. Pour réaliser cette caractéristique nous utilisons une approche objet couplée avec l'utilisation de règles d'enchaînement dérivant directement des spécifications Diane+. Les règles sont utilisées par un moteur d'inférence qui joue un rôle primordial dans la gestion du dialogue et de l'aide. Notre travail (comme celui de M-F Barthet) repose sur l'hypothèse que les données et les traitements sont déjà spécifiés et que les données sont déjà implémentées. Nous présentons cependant dans cette thèse un dérivé du modèle PAC [COUTAZ 88] que nous utilisons conjointement avec la notion d'objet naturel [BRES 90] pour concevoir la couche de base des données. Le modèle résultant est appelé OPAC (Objet naturel PAC). Ce modèle nous paraît intéressant car il fournit des données offrant une représentation externe et interne gérée à un niveau élémentaire par les données elles-mêmes. Diane+ "cimente" ces données en les associant aux opérations qui composent les procédures. Remarquons dès à présent que le niveau de l'utilisateur est supposé connu lorsqu'on aborde la réalisation de l'application. La détermination du niveau de connaissance de l'utilisateur ainsi que l'évaluation de l'évolution de ce niveau ne rentrent pas dans le cadre de cette thèse.

Composition de la thèse

Cette thèse est décomposée en deux parties. La première partie, qui regroupe les chapitres 1 à 3, présente le contexte de notre travail en se focalisant sur le problème de la gestion du dialogue homme-machine et celui de la génération automatique. La seconde partie est composée des chapitres 4, 5 et 6, et détaille plus particulièrement la méthode Diane+ et ses extensions ainsi que son application à la génération et à la gestion automatique.

Le chapitre 1 présente un état de l'art des IHM. Il définit tout d'abord (§ 2) quelques termes importants pour la lecture de cette thèse (Représentation Externe, Représentation Interne, Représentation Conceptuelle, Production Verticale, Production Horizontale…). Nous abordons ensuite l'évolution des outils utilisés pour la construction d'IHM (§ 3) et nous terminons par une discussion sur les avantages et les inconvénients des IHM actuelles ainsi que l'évolution probable à plus ou moins long terme dans ce domaine (§ 4).

Le chapitre 2 traite essentiellement du dialogue homme-machine. Nous y définissons le terme Dialogue (§ 2) tel que nous l'utilisons dans le cadre de notre travail, c'est-à-dire la gestion des interactions de l'homme sur la machine. Puis nous nous focalisons sur la partie Contrôle du Dialogue (§ 3) qui est sous-jacente à l'ensemble de cette thèse. Nous y définissons principalement les rôles du contrôleur de dialogue. Le § 4 présente, quant à lui, les modèles d'applications interactives en décrivant les différents types de contrôle de dialogue existants (§ 4.2), ainsi que les différents modèles (conceptuels et/ou d'implémentation) faisant intervenir la gestion du dialogue homme-machine (§ 4.3). Le chapitre s'achève par une

présentation des différents formalismes employés pour représenter et/ou gérer le dialogue homme-machine (§ 5).

Le chapitre 3 est axé sur le problème de la génération automatique qui constitue également une part importante de notre travail. Il présente tout d'abord les différents types de génération existant dans le domaine des IHM, c'est-à-dire à partir de la Représentation Externe, Interne ou Conceptuelle (§ 2). Le § 3 détaille les différents modules que l'on est capable de générer à l'heure actuelle, c'est-à-dire la présentation, le contrôleur de dialogue et le noyau fonctionnel, tout en présentant les limites de ces générations. Nous terminons ce chapitre par un récapitulatif des produits les plus connus en matière de génération automatique (§ 4).

Le chapitre 4 établit le lien entre la partie I et sert d'introduction à la partie II. Il a pour but de situer notre travail par rapport à ceux présentés dans les chapitres antérieurs en exposant les différents objectifs que nous nous sommes fixés, c'est-à-dire la spécification du dialogue homme-machine avec Diane+ (§ 2.2), la génération du dialogue (§ 2.3), la génération de l'interface (§ 2.4), la gestion automatique de la dynamique (§ 2.5), la gestion automatique de l'aide (§ 2.6) et l'implémentation en objet (§ 2.7).

Le chapitre 5 est consacré entièrement à la méthode Diane+ dont il expose les principes et les concepts. Les notions de buts, de tâches, d'activité, d'opérations et d'intégration de l'utilisateur y sont détaillées (§ 2). Le § 3 montre comment Diane+ gère les tâches en fonction des postes de travail par l'intermédiaire des procédures prévues, minimales et effectives. Nous donnons ensuite au § 4 deux exemples de spécifications Diane+, le premier (la messagerie électronique) étant orienté vers une spécification globale alors que le second (la gestion d'une bibliothèque) est plutôt orienté vers une spécification détaillée, ceci ayant pour but de montrer que Diane+ peut être utilisée à tous les niveaux de spécifications. Le chapitre se poursuit par une comparaison entre Diane+ et d'autres méthodes de spécification et de conception (§ 5). Nous avons décomposé cette comparaison en deux parties : Diane+ et les méthodes orienté-buts (§ 5.1), Diane+ et les méthodes orienté-objets (§ 5.2). Nous montrons par cette comparaison que Diane+ ne peut se suffire à elle-même, mais qu'elle peut par contre être utilisée en complément essentiel à d'autres méthodes pour une spécification (voire une implémentation) efficace du dialogue homme-machine. En raison du caractère à la fois orienté-tâche et orienté-objet de notre travail, il nous a paru essentiel de présenter une réflexion sur le passage du modèle tâche au modèle objet (§ 6). Le modèle de données OPAC qui est issu de cette réflexion et que nous utilisons dans notre travail est exposé à la fin du chapitre (§ 7).

Le chapitre 6 parachève cette thèse en montrant comment Diane+ facilite la réalisation d'applications interactives. Le § 2 montre comment nous utilisons les spécifications Diane+ pour la génération et la gestion automatique. Ce paragraphe explique également la notion d'opérations de base (§ 2.1) qui nous paraît essentielle pour bâtir des applications avec une réutilisation maximale de composants élémentaires tout en conservant une norme comportementale. Le § 2.2 s'intéresse plus particulièrement à la génération automatique du code des opérations (et des procédures) Diane+, ainsi qu'à celle de l'interface. Le § 2.3 présente les règles d'enchaînement et leur intérêt pour la gestion automatique de la dynamique et de l'aide d'utilisation. Le § 2.4 propose une discussion sur l'évaluation automatique des spécifications et de l'ergonomie des applications créées avec Diane+. Enfin le § 3 détaille la représentation interne des différents concepts présentés dans les deux derniers chapitres, c'est-à-dire les opérations, les procédures, les buts et les règles. Il expose également

l'implémentation et le fonctionnement du contrôleur de dialogue. Il se base pour cela sur les spécifications d'un outil dédié à Diane+ dont la maquette fait l'objet du § 3.6.

PARTIE I

Chapitre 1
Évolution des IHM

1. Introduction

Les Interfaces Homme-Machine (en abrégé : IHM) peuvent être abordées sous divers angles : informatique, linguistique, ergonomique... Chacune de ces approches définit ses propres concepts : utilité, pragmatique, représentation conceptuelle, etc. Pourtant il semble que toutes ces approches tournent autour de deux idées centrales : le produit que l'on veut obtenir et le produit que l'on a obtenu, ou plus précisément, d'une part ce que l'on veut réaliser et d'autre part la démarche de conception et l'utilisation du produit. En ergonomie par exemple, Senach [SENACH 90a] différencie l'*utilité* de l'*utilisabilité*. La première permet de dire si le produit répond aux besoins de l'utilisateur alors que la seconde décrit la facilité d'apprentissage et d'utilisation du produit. En fait l'utilité peut se rapprocher des notions sémantiques, syntaxiques et lexicales données en linguistique et de la représentation conceptuelle en informatique. L'utilisabilité se rapproche, quant à elle, des notions de pragmatique en linguistique et de la représentation externe en informatique.

Ces notions que nous venons d'énumérer sont décrites plus en détail dans la première partie de ce chapitre. La deuxième partie est consacrée à l'évolution des outils en matière d'IHM. Nous terminerons ce chapitre par une discussion sur les IHM actuelles et leur évolution probable.

2. Quelques concepts importants en matière d'IHM

La production d'applications interactives a modifié les étapes nécessaires à la production d'applications informatiques. Auparavant le développement d'une application était entièrement centré sur l'informatique (les applications ne prenaient que très peu en compte l'utilisateur) et n'avait pour but que de reproduire des tâches manuelles (par exemple des facturations). Le développement était décomposé en cinq étapes :

- *l'analyse de l'existant et l'analyse des besoins* qu'on retrouvait la plupart du temps uniquement dans l'informatique des organisations,
- *la spécification* basée sur l'ergonomie cognitive et l'informatique,
- *la conception* qui se situe uniquement au plan informatique (conception logicielle),
- *la réalisation* qui ne constitue que l'implémentation machine de l'application,
- *l'évaluation* qui fait intervenir l'ergonomie cognitive.

Analyse de l'existant

L'analyse de l'existant est composée du recueil des moyens, des dispositifs, des matériels, etc., dont on dispose avant d'entreprendre la création d'un nouveau système ou la modification d'un système existant. Elle peut être complétée par une critique de l'existant qui permet d'éviter d'anciennes erreurs pour la spécification.

Spécification

La spécification décrit le système auquel on désire aboutir. Cette description n'est faite que pour expliciter ce que doit réaliser le système, c'est-à-dire les buts à atteindre, et la représentation qu'en aura l'utilisateur final (Représentation Externe). On peut dire que la spécification répond à la question “Quoi ?”.

Conception logicielle

La conception logicielle décrit le “Comment ?” de l'application. Elle reprend les spécifications et les explicite en fonction du type d'outil que l'on utilisera pour l'implémentation. On décrit donc la manière d'atteindre les buts que l'on a spécifiés ainsi que la façon de représenter les données que l'on manipulera.

Réalisation

La réalisation permet de représenter la conception en fonction d'un outil particulier. Elle correspond à l'implémentation machine de cette dernière et dépend fortement de l'outil choisi. Ainsi une approche objet lors de la conception pourra être réalisée par l'utilisation de Smalltalk ou d'Eiffel.

Évaluation

L'évaluation a pour rôle de vérifier que le nouveau système répond aux besoins de l'utilisateur en fonction des spécifications demandées. Dans le cas où cette évaluation révèle des points négatifs, un processus de correction est mis en route.

Le processus énoncé plus haut est en réalité itératif et peut conduire à un recouvrement partiel ou total des phases. Par exemple, l'évaluation peut se faire soit à la fin soit tout au long du développement de l'application. Dans le cas où elle fait apparaître des défauts, on exécute de nouveau le cycle (nouvelles spécifications et/ou nouvelle conception et/ou nouvelle réalisation, etc.).

L'intégration d'interfaces homme-machine a impliqué l'ajout de nouvelles contraintes dans les étapes énoncées plus haut :

- l'analyse de l'existant s'est vue augmentée de l'étude des utilisateurs actuels et futurs, de l'étude des tâches existantes et de l'étude de l'organisation,
- la spécification regroupe en plus la spécification des utilisateurs, des tâches et le modèle des IHM,
- la conception est augmentée de la conception des IHM,
- la réalisation intègre en plus la réalisation des IHM,
- l'évaluation doit également évaluer les IHM ainsi que les améliorations des tâches interactives par rapport à des tâches non-interactives.

Ce nouveau processus fait appel à des outils de développement qui seront détaillés dans le Chapitre 2.

De nouvelles notions ont fait leur apparition avec ce processus de développement (Figure 1.1). On constate par exemple qu'une interface est l'aboutissement de trois notions fondamentales : la

Représentation Conceptuelle (RC), la *Représentation Externe* (RE) et la *Représentation Interne* (RI).

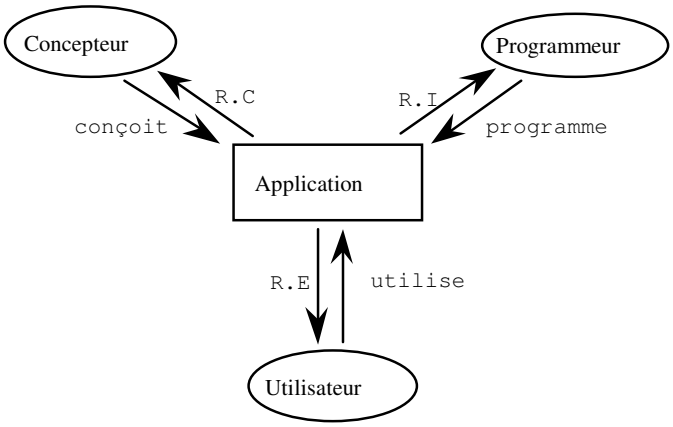


Figure 1.1 : Les trois représentations d'une application

Représentation Conceptuelle

Créer une application, c'est avant tout définir ses objectifs, ses fonctionnalités, son public, etc. Les phases de spécification et de conception vont nous permettre de répartir les rôles entre l'usager et l'application, définir un modèle pour les données, de même que l'allure générale des écrans. Tout ceci compose la *Représentation Conceptuelle* (abrégé en RC). La méthode DIANE+ que nous étudierons dans les Chapitres 5 et 6, intègre à cette Représentation Conceptuelle des éléments de psychologie cognitive tels que la logique d'utilisation [RICHARD 83] [BARTHET 91].

La Représentation Conceptuelle regroupe tous les concepts manipulés par l'application. Ces concepts peuvent être de natures différentes, par exemple les données manipulées, les tâches à réaliser, le niveau des utilisateurs ou la gestion des interactions. Divers formalismes peuvent être employés pour représenter cette Représentation Conceptuelle ; citons par exemple les Réseaux de Pétri ou les diagrammes de transitions pour la gestion du dialogue ou bien encore les graphes entités-associations pour les données.

Représentation Externe

La Représentation Externe (RE) a pour but de définir de manière précise la représentation spatiale ("Look and Feel") et le comportement de l'application à l'écran. Elle est composée de l'ensemble des écrans que l'application finale affichera ainsi que les différents éléments d'interaction tels que les menus et les icônes.

La Représentation Externe est un compromis entre trois facteurs : l'existant qui peut fournir par exemple des formulaires de référence pour des applications administratives, les contraintes machine (type d'écran, temps de réponse, etc.) et les critères d'ergonomie générale (écran non surchargé, nombre maximum de menus...). Cette représentation peut être obtenue de diverses manières : description par un langage de programmation, maquette créée interactivement, etc.

Représentation Interne

La Représentation Interne (RI) est en fait la représentation de l'application en machine. Nous verrons plus loin que la décomposition de cette RI en trois modules est courante, mais qu'il en existe beaucoup d'autres. La personne chargée de l'intégration de l'application en machine doit tenir compte de la Représentation Externe et de la Représentation Conceptuelle. Son travail est entièrement basé sur ces deux étapes puisqu'il est le mariage des résultats de la RE (par exemple du code généré par un UIMS) et de ceux de la RC (par exemple le choix d'un type de langage).

A partir de ces trois représentations, on peut remarquer qu'il existe deux types de productions d'interfaces : la *production verticale* et la *production horizontale* (Figure 1.2). La production horizontale permet de passer de la Représentation Externe à la Représentation Interne (et réciproquement) tandis que la production verticale permet de passer de la Représentation Conceptuelle à la Représentation Externe et à la Représentation Interne.

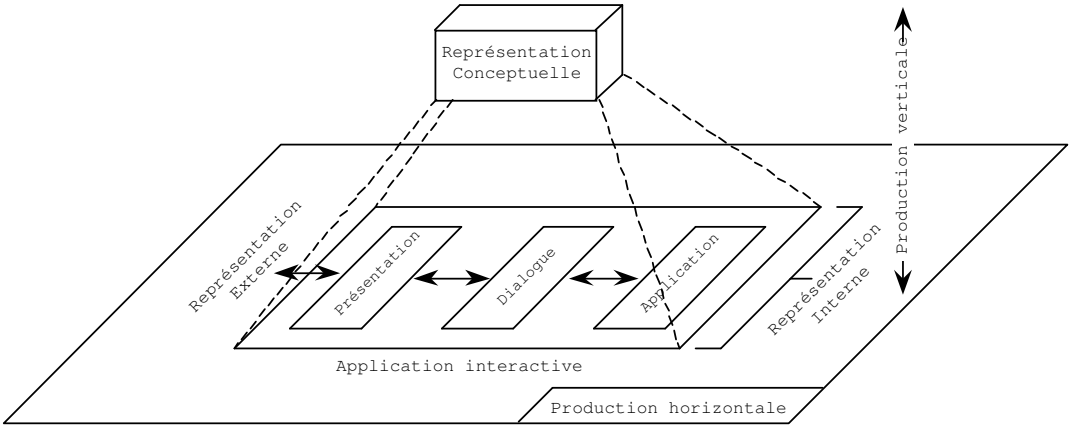


Figure 1.2 : Les deux niveaux de production d'interfaces

Production horizontale

La plupart des travaux actuels concernent la production horizontale. Parmi les trois représentations d'une interface, la production horizontale ne concerne que la Représentation Externe et la Représentation Interne, c'est-à-dire le niveau syntaxique et lexical à l'opposé de la Représentation Conceptuelle qui représente le niveau sémantique et pragmatique. Les boîtes à outils par exemple permettent de créer une Représentation Externe à partir d'appels de procédures (Représentation Interne). A l'inverse les UIMS génèrent une Représentation Interne à partir de la Représentation Externe construite directement à l'écran. Le but de la production horizontale est de réduire la "distance sémantique"[1] [HUTCHINS 86] par l'intermédiaire de la manipulation directe. L'utilisateur doit pouvoir travailler sur des objets qui représentent son monde réel et ce travail doit se faire dans des conditions les plus proches possibles des conditions réelles. Les icônes du Macintosh sont un exemple classique.

Bien que de plus en plus efficaces, tant sur le plan technique que sur le plan ergonomique, la production horizontale a un inconvénient majeur : elle n'intègre pas d'**étape conceptuelle**. Dans la production horizontale, le réalisateur de l'interface ne peut que reprendre les résultats de la phase

[1] Hutchins définit la "distance sémantique" comme l'écart entre les concepts présentés par l'interface et les concepts propres à l'utilisateur.

conceptuelle afin de les intégrer au mieux dans la Représentation Externe et dans la Représentation Interne.

La production horizontale est donc issue d'une étape conceptuelle, mais celle-ci est implicite. C'est le cas par exemple des interfaces que l'on construit au-dessus d'applications existantes. Généralement le programmeur ne connaît que les entrées-sorties possibles entre l'interface et l'application. Il lui importe peu de savoir que la base de données X est relationnelle ou objet, ou que le fichier Y est séquentiel ou à accès direct, mais la représentation conceptuelle de l'application est sous-jacente à son travail.

Production verticale

La production verticale est beaucoup plus proche des besoins actuels. Elle respecte l'évolution logique du développement d'une interface. On commence par concevoir l'application en tenant compte des spécifications et des caractéristiques de l'utilisateur, puis on affine sa Représentation Externe du point de vue lexical et syntaxique (par exemple la taille des boutons ou la disposition des menus) et enfin on confie ces deux travaux aux personnes chargées de les intégrer dans la Représentation Interne. Le résultat est une interface mieux étudiée, mieux structurée et donc facilement extensible. La phase conceptuelle bénéficie de l'apport de méthodes existantes telles que Merise [TARDIEU 85] [TARDIEU 86], Axial [PELLAUMAIL 86], SADT [MARCA 88] ou Diane [BARTHET 88], ce qui permet sa mise en place de façon rigoureuse.

La production verticale ne sert pas à dire comment on représente une opération, mais plutôt pourquoi elle existe. En d'autres termes, la production verticale est la **raison d'être** et la production horizontale est la **façon d'être**. Pour la création de menus par exemple (Figure 1.3), le rôle de la production horizontale est de dire que l'on a:

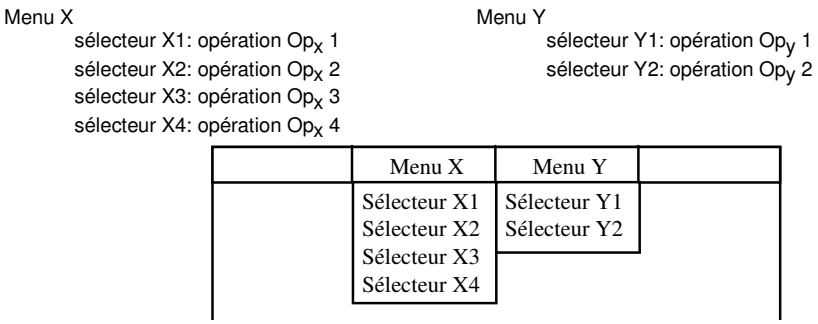


Figure 1.3 : Création de menus

La production verticale doit permettre de dire *pourquoi* on a un *menu* X ***et*** un *menu* Y, et *pourquoi* on a classé les opérations de cette manière.

Le passage de la Représentation Conceptuelle à la Représentation Externe et à la Représentation Interne peut se faire de trois façons : manuelle, assistée ou automatique. Un exemple de passage de la Représentation Conceptuelle vers la Représentation Externe est le travail d'Isabelle Petoud [PETOUD 90] qui permet de générer depuis la Représentation Conceptuelle d'une application les différentes fenêtres nécessaires à celle-ci (caractéristique également présente dans le travail de V. Normand [NORMAND 92a]). Il faut noter par ailleurs qu'il existe des travaux tels que GUISE [BOUSSE 91] ou ThingLab [BORNING 79] qui utilisent déjà la génération automatique dans la production horizontale (cf Chapitre 3). Le travail présenté dans cette thèse utilise ce principe de génération automatique à partir de la Représentation Conceptuelle.

D'une manière générale, on peut conclure en disant que le but recherché par la réalisation d'une IHM est de réduire au maximum les distances d'exécution[2], d'évaluation[3] [NORMAN 86], sémantique et articulatoire[4] de l'utilisateur [NANARD 90].

3. Évolution dans la construction des IHM

Ce paragraphe a deux objectifs : d'une part donner un bref historique des IHM afin de mettre en relief les étapes importantes dans ce domaine, et d'autre part montrer l'évolution des outils servant à la construction des IHM.

3.1. Un bref historique

La souris, bien que d'utilisation récente, est apparue il y aura bientôt trente ans puisqu'elle a vu le jour en 1964 chez Xerox. Elle devait contribuer quelques années plus tard à une petite révolution informatique. 1969 voit l'apparition du graphisme et de la notion de fenêtre. Ces deux nouveautés vont se retrouver dans Smalltalk qui naît en 1972 à Xerox PARC (Palo Alto Research Center). Son environnement graphique et multi-fenêtré, conjugué avec l'utilisation de la souris, ont véritablement lancé la recherche en matière d'IHM. Depuis quelques années déjà, des chercheurs essayaient de représenter, de formaliser et d'améliorer le dialogue homme-machine. En 1968 par exemple, W. M. Newman [NEWMAN 68] avait proposé un gestionnaire de réactions qui est considéré actuellement comme le premier UIMS. Pourtant ce n'est qu'en 1982 qu'apparaîtra pour la première fois ce terme d'UIMS [KASIK 82].

La même année, le workshop sur les techniques d'interaction et de saisies graphiques de SEATTLE permet de définir le contenu d'un UIMS, son rôle dans le développement des applications et met en relief la nécessité de connaissances pluridisciplinaires pour la recherche en matière d'IHM.

Steve Jobs, après une visite à Xerox PARC durant laquelle il découvrit l'environnement Smalltalk, créa son premier Macintosh en 1983. Toutes les publicités d'alors montraient le fameux petit ordinateur avec son écran orné d'icônes manipulées directement par la souris, alors que les publicités pour les micro-ordinateurs ne montraient pas encore d'interface graphique. Le retard pris par les PCs allait être long à rattraper.

Suite au workshop de SEATTLE, la ville de SEEHEIM en Allemagne accueillit à son tour un workshop centré sur le rôle, le modèle, la structure et la construction des UIMS. Il en ressortit le modèle qui devait prendre le nom de la ville, le modèle de Seeheim (Figure 1.4). Ce dernier fait apparaître trois composants logiques[5] qui sont greffés sur le noyau applicatif. Ces composants sont :

- la *présentation* qui est responsable de l'apparence de l'interface,
- le *contrôleur de dialogue* qui est responsable de la gestion des échanges entre la présentation et l'application,

[2] “La distance d'exécution traduit l'effort de mise en correspondance entre la représentation mentale interne de la tâche à effectuer et sa représentation physique externe” [COUTAZ 90].

[3] La distance d'évaluation est la différence entre le résultat obtenu (après une commande par exemple) et le résultat escompté.

[4] “La distance articulatoire traduit l'effort de traduction entre la connaissance sémantique que l'utilisateur a du système et sa connaissance syntaxique des éléments de présentation du système” [COUTAZ 90].

[5] “Logique” est à prendre dans le sens opposé de “physique”.

- *le modèle de l'interface de l'application* qui est directement connecté sur le noyau applicatif et qui déclenche les traitements.

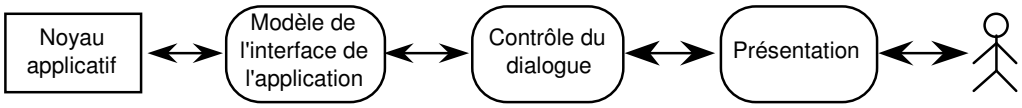


Figure 1.4 : Le modèle de Seeheim

Depuis 1983, ce modèle de Seeheim a donné naissance à beaucoup d'autres modèles (PAC, Hudson, Arch...). De plus en plus contesté à l'heure actuelle puisqu'il va à l'encontre du concept objet, il reste cependant à la base de nombreux travaux.

1985 voit l'apparition de l'environnement GEM sous DOS dont la ressemblance avec l'interface du Macintosh est relativement frappante. GEM n'aura cependant pas le succès escompté et Microsoft en profitera pour créer Windows qui deviendra l'environnement graphique de référence des PCs. Il faut remarquer que la première version de Windows (1986) était très limitée ; il était impossible par exemple de superposer des fenêtres (cette limitation n'existant pas sur Macintosh à la même époque). Parallèlement à l'évolution de DOS et Windows, les stations de travail ont vu petit à petit apparaître des environnements qui leur étaient propres ; Open Look et Motif (1989) [OSF 89] qui sont bâtis sur X-Window [SCHEIFLER 86] sont devenus des normes de facto pour les stations de la même manière que Windows pour les PCs.

De 1983 à 1993, les environnements de développement n'ont fait qu'évoluer mais tout en gardant leurs caractéristiques de départ. Certains produits se sont vus augmentés de fonctionnalités intéressantes comme le pseudo-multi-tâche avec Windows[6] 3.0 ou le système 7.0 du Macintosh, mais rien de vraiment nouveau n'est apparu à part la machine NeXT de Steve Jobs. C'est en effet la première machine qui dispose d'un système entièrement objet, spécialement conçu pour le développement d'applications hautement interactives.

3.2. Évolution des outils

Que l'on travaille dans le but de construire des maquettes[7] ou des prototypes[8], la tâche est ardue lorsqu'on ne dispose d'aucun outil particulier. Il y a encore quelques années (avant 1980), un programmeur devait tout faire lui-même par l'intermédiaire d'un langage de programmation qui impliquait bien souvent la gestion d'événements bas niveaux.

L'arrivée du Macintosh sur le marché a contribué à l'amélioration du développement des applications interactives. Son système d'exploitation était le premier à proposer une boîte à outils[9] (Toolbox) [ROSE 86] qui donnait un accès direct aux commandes de base nécessaires à la gestion des interactions et du multi-fenêtrage. Attrayantes de prime abord, les boîtes à outils ne déchargent

[6] En 1990 IBM introduit la norme CUA [CUA 89] pour favoriser le développement d'un standard dans les IHM. Windows 3.0 et OS2 PM 1.2 en ont été les premiers bénéficiaires.

[7] Une maquette est une ébauche plus ou moins complète du produit final. Plus précisément, c'est un ensemble d'objets graphiques organisés pour donner une image fidèle de l'écran que verra l'utilisateur du futur logiciel. Les fonctionnalités de la future application n'étant pas implémentées, les manipulations de ces objets n'ont que des répercussions syntaxiques telles que l'apparition d'un menu fugitif, par exemple [MICHARD 90].

[8] Un prototype est une version du logiciel réalisant l'ensemble des fonctionnalités de la version définitive, mais sans souci de performance [MICHARD 90].

[9] Une *boîte à outils* est une librairie de primitives prêtes à l'utilisation pour la construction d'IHM.

les concepteurs que dans les tâches élémentaires pour la gestion des IHM et n'apportent aucune solution tangible pour la gestion du dialogue.

Les informaticiens se sont alors penchés sur la possibilité de créer des outils qui permettraient d'enchaîner plus ou moins complètement les fonctions des toolbox afin de simplifier la tâche des programmeurs. Ainsi sont apparus les squelettes d'application appelés encore "applications cadres" (Apex [COUTAZ 87], Grow [BARTH 86], MacApp [SCHMUCKER 86] [APPLE 89]). Ces outils sont en fait une couche qui englobe les boîtes à outils. Ils sont basés sur une notion de boucle d'acquisition/traitement des événements provenant de l'utilisateur. La gestion du dialogue reste donc très rudimentaire et surtout très rigide.

L'étape suivante a été l'apparition des environnements de travail et de leur kit de développement (Toolkit ou encore Software Development Kit SDK). A l'heure actuelle tous les environnements possèdent leur toolkit (Windows SDK, Presentation Manager SDK...). L'utilisation des environnements graphiques a fortement encouragé le développement d'applications compatibles avec ces environnements. Ils ont été les premiers à permettre véritablement d'une part la *création interactive* des écrans et des icônes sans qu'il soit nécessaire d'écrire du code, et d'autre part la *séparation de l'interface avec l'application.* Il faut noter cependant que les toolkits obligent le programmeur à tout écrire en ce qui concerne la gestion du dialogue homme-machine.

Rapidement les programmeurs se sont trouvés bloqués par les limites des toolkits. Ils souhaitaient pouvoir relier facilement les maquettes aux traitements à l'aide d'un seul et même outil. Les UIMS[10] (User Interface Management System) ont répondu à leurs besoins [BARTH 86], [KARSENTY 87], [OLSEN 89], [BOUSSE 89], [LINTON 89], [ROUDAUD 90], [SHAN 90], [GRØNBÆK 91]. Tous les UIMS possèdent deux rôles : *conception* et *réalisation* d'interfaces. D'une manière générale, ils sont à l'heure actuelle une des meilleures solutions pour la construction d'IHM bien que la gestion du dialogue homme-machine reste à la charge du programmeur[11] et que la gestion de l'aide soit quasi inexistante. Le terme d'UIMS s'est vu traduit en français par SGIU qui signifie Système de Gestion d'Interfaces Utilisateur, mais qui signifie parfois Système de Génération d'Interfaces Utilisateur. Cette caractéristique de générer l'interface se retrouve dans des produits tels que Graffiti [KARSENTY 87], Mickey [OLSEN 89], Scenarioo [ROUDAUD 90], Guise [BOUSSE 91], Siroco [NORMAND 92a], Dost [DEWAN 87], Vitamin [TAHON 90] ou ApplBuilder [GRØNBÆK 91]. Les générations sont de plusieurs types et se situent à deux niveaux correspondant aux deux niveaux de production précédemment énoncés :

- la génération horizontale qui consiste à :
    - générer le code des écrans à partir d'un dessin à l'écran (les toolkits possèdent également cette faculté),
- la génération verticale qui consiste à :
    - générer les écrans à partir de spécifications sous forme de grammaire ou d'un langage quelconque ou bien encore à partir d'un Modèle Conceptuel des Données (MCD) [PETOUD 90]. G. Singh [SINGH 89] propose par exemple un outil qui génère la représentation externe à partir d'une description du système et éventuellement des préférences de l'utilisateur. Nous verrons dans le chapitre 6 que notre système génère également la Représentation Externe à partir de spécifications conceptuelles,

[10] On trouve dans la littérature un autre terme, UIDS (User Interface Development System), qui semble mieux convenir à la notion de création d'interface que celui d'UIMS [MYERS 89].

[11] Les UIMS apportent toutefois des solutions pour la gestion élémentaire du dialogue. Ainsi la gestion des widgets et des enchaînements d'écrans peut être partiellement prise en compte de manière automatique après les spécifications faites par le programmeur (par exemple par l'intermédiaire de conditions de déclenchements).

- générer le comportement de l'interface au niveau lexical et syntaxique, par exemple la syntaxe d'un éditeur de formules mathématiques à partir d'une grammaire à attributs ou d'un langage spécialisé [FRANCHI 89]. Nous verrons dans le chapitre 6 que notre système génère et gère automatiquement la Représentation Externe à partir de spécifications conceptuelles,
- générer le module de gestion des interactions à partir de spécifications exprimées dans une forme quelconque [SINGH 89] [DUVAL 91] ou bien encore à partir d'exemples donnés par le programmeur [MYERS 88b]. Nous verrons dans le chapitre 6 comment notre système gère automatiquement les interactions de l'utilisateur,
- générer les spécifications de l'application (interface et traitements) qui serviront à leur tour à générer les représentations externe et interne de l'application [KOLSKI 91].

A l'heure actuelle, la majorité des UIMS font appel à la programmation événementielle et objet. Il a donc fallu développer des méthodes d'analyse et de conception adaptées à ce type de travail [CASTELLANI 91]. On peut citer la méthode de Grady BOOCH [BOOCH 91], les méthodes HOOD [HEITZ 87] [HOOD 89], OOA [COAD 90], ObjectOry [JACOBSON 89], OOSD [WASSERMAN 91] ou bien encore O* [ROLLAND 91]. D. De Champeaux [DE CHAMPEAUX 91] remarque que la plupart des méthodes d'analyse orienté-objet ne sont pas assez précises, c'est-à-dire qu'elles n'intègrent pas un langage d'analyse suffisamment riche pour décrire sémantiquement et de manière unique et rigoureuse le domaine à traiter. De plus elles ne possèdent pas de processus de développement, c'est-à-dire un cadre où l'on puisse décomposer et recomposer un problème.

Conclusion sur l'évolution des IHM et des outils

Les outils n'ont cessé d'évoluer depuis l'apparition des IHM. Leur nombre croît sans cesse et il est devenu impossible de les classer et de les comparer suivant un seul critère. Plusieurs synthèses et classifications ont déjà été faites à ce sujet [HARTSON 89], [MYERS 89b], [COUTAZ 90], [NANARD 90], [PETOUD 90], [EL MRABET 91]. De toutes il ressort qu'aucun outil n'est le remède miracle pour le développement d'IHM. Chacun a ses spécialités et ses domaines d'application ce qui n'a pas empêché les IHM de s'être considérablement améliorées aussi bien sur le plan du développement que sur le plan de l'utilisation.

4. Discussion sur les IHM actuelles

Ce paragraphe a pour objectif de donner une évaluation générale des IHM et des outils de développement sans dissocier de type d'outil ou d'interface en particulier. Il est composé de critiques d'ordre général auxquelles s'ajoute un paragraphe concernant l'évaluation des IHM. Les améliorations que l'on peut espérer à court ou à long terme dans le domaine des IHM concluent cette partie.

4.1. Inconvénients des IHM

La plupart des outils ne permettent d'implémenter (ou de spécifier) que les entrées utilisées par l'interface. Ils ne garantissent pas des échanges suffisants et permanents entre l'application et le contrôleur de dialogue ce qui a pour effet de laisser à l'application la charge de produire les expressions de sortie. Ainsi la programmation d'un bouton se fait en général très facilement en déclarant la procédure à activer en cas de clic, mais la mise à jour des zones qui en dépendent reste

souvent à la charge de l'application, donc du programmeur. L'approche utilisée dans cette thèse résout le problème de la cohérence entre la Présentation et le Noyau Fonctionnel.

Bien que pratiquement toutes les interfaces possèdent à l'heure actuelle des services tels que Undo ou le Couper/Coller, ces services ne sont généralement pas implémentés dans les UIMS et restent donc à réaliser par le programmeur[12]. Ainsi le Couper/Coller existe souvent dans la boîte à outils de base, mais le fait de vouloir l'utiliser impose de connaître son fonctionnement interne ainsi que les formats des données que l'on peut y stocker. Copier un dessin ou un texte est identique au niveau conceptuel, mais bien différent au niveau implémentation ! Il serait intéressant d'avoir une architecture où toutes les entités manipulées seraient basées sur le même format, permettant ainsi des manipulations de tout ordre sans se soucier du type de donnée qu'on manipule. Notre travail apporte une solution quant aux services Quitter l'application, OK, Annuler et quelques autres services utilisés dans les bases de données.

L'aide est devenue un service essentiel dans les IHM. Cette aide peut être contextuelle ou non, mais elle est fonctionnelle[13] dans tous les cas. Il est regrettable cependant que la plupart des outils actuels ne fournissent aucun service pour le développement et l'intégration de ce type d'aide et encore moins pour l'aide d'utilisation. Nous verrons dans le Chapitre 6 qu'elle est la solution que nous apportons à ce problème.

Il n'est pas possible pour l'instant de développer des applications sans qu'une langue soit utilisée de manière implicite. Bien que le module chargé de la représentation externe de l'application soit généralement développé plus ou moins à part, il fait appel directement à un vocabulaire pour les commandes, pour l'aide, etc. Si l'on veut que l'application soit utilisée dans un pays qui ne parle pas la langue qui a servi à l'implémentation de départ, il faut reprendre chacun des messages et les traduire dans la nouvelle langue. Ce travail fastidieux est long et la traduction reste très souvent, et pour le malheur des utilisateurs, du mot à mot[14]. Il arrive ainsi que des messages d'aide soient tronqués à l'écran car la fenêtre prévue pour le message en anglais n'a pas été redimensionnée pour le message en français qui est souvent plus long. Certains messages ont parfois l'apparence d'un charabia en mélangeant deux langues (par exemple : "Voulez-vous formater une autre disquette (**Y/N**) ?" ou bien encore "Impossible d'afficher un élément **deleted**"). Des travaux sur l'internationalisation sont toutefois en cours depuis quelques années [NEILSEN 90].
Alors que les techniques actuelles intègrent de plus en plus le multi-média, il est exceptionnel de trouver des interfaces et encore plus rarement des outils[15] permettant de gérer ce multi-média[16]. On constate cependant que certains nouveaux matériels tentent de remédier à cela en intégrant des possibilités de sons, de graphismes de synthèse ou encore d'images vidéo.

---

[12] Étant donné que ces services sont fortement dépendants de l'application à laquelle ils sont rattachés, leur implémentation relève bien souvent de la débrouillardise du développeur. Ainsi dans MacDraw il est possible d'annuler la dernière opération car les opérations ne sont effectivement enregistrées qu'avec un coup de retard.

[13] L'aide fonctionnelle est un service qui explique le fonctionnement d'une commande (par exemple l'impression d'un document) alors que l'aide d'utilisation permet d'expliquer comment réaliser une tâche (par exemple comment mettre au format A3 un document qui était au format A4).

[14] Il arrive encore fréquemment que cette traduction soit incorrecte. Word 5.0 pour Macintosh présente par exemple le message suivant lors de la conversion de formats : "Le document **appraître** dans **un** nouvelle fenêtre sans titre"

[15] On peut toutefois citer la boîte à outils QuickTime qui procure une extension multi-média au système 7 du Macintosh.

[16] Nous parlons ici de multi-média qu'il ne faut pas confondre avec le multi-modal qui est la possibilité d'utiliser conjointement plusieurs médias au lieu de les utiliser séparément. Ainsi le Macintosh ou le Next peuvent être considérés comme multi-média alors que les mondes virtuels qui font leur apparition peuvent être considérés comme multi-modal. En effet dans le premier cas la machine peut séparément ou simultanément afficher une image vidéo ou jouer de la musique alors que dans le second cas, c'est l'interaction entre le graphisme et des manipulations tridimensionnelles qui fait évoluer l'environnement.

Développer une application demande le plus souvent une activité d'équipe. Or les outils ne sont conçus en général que pour des développements individuels et la conjonction entre plusieurs personnes doit être faite manuellement. Certains AGL et UIMS (tels que ENVY/Developer [ENVY 90]) possèdent cependant des fonctionnalités destinées à gérer le travail d'équipe (cohérence des développements individuels, gestions des versions successives des applications...), mais leur nombre est encore très limité.

Une des lacunes les plus importantes est sans doute le manque d'intégration d'un modèle de l'utilisateur dans l'interface. Les systèmes tutoriels intelligents [GOLDSTEIN 82] [SLEEMAN 82] [WENGER 87] sont un des domaines où cette intégration aura le plus de retombées. Si l'on arrive à évaluer et à représenter en machine les connaissances de l'utilisateur, ses croyances, ses erreurs, etc., on peut imaginer des interfaces qui peuvent non seulement être adaptées à un type d'utilisateur (plus précisément à un utilisateur particulier) mais qui peuvent de plus s'adapter automatiquement en fonction de l'évolution du niveau de l'utilisateur. Ceci est très important dans les domaines d'apprentissage comme celui de la lecture, des langues étrangères ou encore de la musique.

L'évaluation est le dernier point que nous aborderons ici. Le développement d'une application interactive suit des étapes bien précises pendant (ou après) lesquelles intervient une phase d'évaluation. Cependant cette étape est pour l'instant entièrement manuelle même si elle repose quelquefois sur des outils informatiques. Elle peut être facilitée par un apport d'ergonomie au moment de la conception, par exemple grâce à des règles d'ergonomie, mais sa validation ne peut se faire que par l'intermédiaire d'un spécialiste humain, l'ergonome.

4.2. Avantages des IHM

Les avantages des interfaces actuelles peuvent se grouper en deux catégories : ceux qui relèvent de la facilité de développement et de maintenance et ceux qui relèvent de la facilité d'apprentissage.

4.2.1. Développement et maintenance des IHM

Les modèles utilisés pour formaliser les interfaces ont provoqué des séparations dans les applications interactives. Ces séparations peuvent être celle des données et des traitements, ou bien de l'interface et de l'application ou bien encore de la représentation et du comportement. Dans tous les cas, les modèles ont encouragé des développements indépendants et en parallèle. Les outils ont été adaptés à cette nouvelle approche ce qui a eu pour effet d'améliorer le développement ainsi que la maintenance et l'évolution des applications développées. Les concepteurs essaient pour cela d'intégrer au maximum dans leurs outils les propriétés énoncées ci-dessous [HARTSON 89] :

- *la fonctionnalité* qui permet d'évaluer ce qu'un outil peut faire, c'est-à-dire les styles et techniques d'interface qu'il peut utiliser et produire,
- *l'utilisabilité* qui concerne la facilité d'utilisation d'un outil pour produire des interfaces par rapport à ses fonctionnalités,
- *la complétude* qui permet de représenter l'interface sur tous les plans (par exemple syntaxique, lexical et sémantique),
- *l'extensibilité* qui permet de pallier le fait que la complétude est rarement atteinte. Elle implique la modifiabilité des représentations ainsi que l'extensibilité des styles d'interaction et des périphériques,
- *l'échappement* qui est la possibilité d'utiliser la programmation classique au lieu d'un outil inadéquat afin d'augmenter la fonctionnalité. Ceci implique une compatibilité entre l'environnement de programmation et l'outil,

- *la manipulation directe* qui permet de manipuler interactivement les représentations visuelles d'objets correspondant à la tâche de l'utilisateur ce qui permet des résultats immédiatement visibles et facilement réversibles,
- *l'intégration* qui doit permettre à un ensemble d'outils d'être intégré au sein d'une même interface et d'utiliser une interface uniforme entre les outils,
- *la localité de définition* qui permet de définir une fois pour toutes un choix de conception ou d'implémentation qui se répercute alors automatiquement. Le mécanisme d'héritage des systèmes objets permet de valider facilement cette propriété,
- *le guidage structuré* qui permet d'implémenter des aides de tout type par l'intermédiaire de tutoriels, d'aide en ligne ou de suggestions de la machine (par exemple des valeurs par défaut).

4.2.2. Apprentissage des IHM

Les applications interactives actuelles demandent des phases d'apprentissage de plus en plus courtes. Ceci s'explique par l'augmentation de quatre facteurs principaux :

- *la normalisation* au niveau externe et comportemental : la sélection d'un élément se fait par exemple toujours de la même manière, un clic pour un bouton ou bien en faisant glisser la souris tout en appuyant sur le bouton de gauche pour du texte.
- *l'aide contextuelle* : les manuels en papier font place de plus en plus à des manuels informatiques, c'est-à-dire directement intégrés à l'application. Ils sont généralement interactifs (hypertextes) et s'adaptent au contexte ; par exemple le fait de demander de l'aide alors qu'une boîte de dialogue est ouverte provoquera l'affichage de l'aide pour cette boîte. Il faut remarquer cependant que cette aide contextuelle est encore très limitée,
- *la manipulation directe* qui permet des interactions de haut niveau et immédiates sans passer par l'intermédiaire de langages de commandes ou d'outils divers. Il est ainsi possible de supprimer directement des documents en les faisant glisser dans une poubelle,
- *le réalisme*[17] des interactions qui est dû en partie à la manipulation directe. Ce réalisme a pour effet de placer l'utilisateur dans un environnement qui lui semble réel ou tout au moins crédible. Par exemple les environnements actuels représentent les boutons en relief améliorant ainsi considérablement la notion de bouton par rapport à celle que se faisaient les utilisateurs lorsqu'il n'y avait pas de relief. De même le fait de jeter un élément à la poubelle et de voir celle-ci grossir comme si elle était pleine, est beaucoup plus explicite que de taper une commande de suppression.

4.3. Évaluation des IHM

Toutes les évaluations d'une IHM visent à identifier ou à prévoir les difficultés que rencontrent les utilisateurs et à en caractériser les points forts et les points faibles [SENACH 90a]. L'évaluation d'une IHM et sa qualité sont donc fortement liées. Cette évaluation peut survenir en fin de processus de développement mais en général il est préférable que les phases d'évaluation se fassent tout au long du processus. B. Senach [SENACH 90a] distingue trois types d'évaluation (Figure 1.5) :

- *l'évaluation d'alternatives de conception* : lorsqu'on dispose de plusieurs solutions et qu'aucun critère ne nous permet a priori de les départager, la comparaison des résultats obtenus avec les différentes solutions proposées permet de choisir la meilleure sinon la plus satisfaisante,

[17] Un terme résumant ce réalisme est né lors de l'apparition des IHM ; il s'agit du WYSIWYG. Les premières manifestations ont été l'interface du Macintosh et des traitements de texte tels que Word.

- *l'évaluation itérative* qui comprend la détection des imperfections et la mesure des améliorations apportées tout au long du processus de développement,
- *les bancs d'essai* qui permettent d'évaluer le produit fini et qui fournissent un diagnostic plus global.

“Il faut noter que cette organisation reste théorique étant donné que dans la réalité, du fait des contraintes importantes qui pèsent sur le développement, les approches sont souvent moins structurées et les tests ne sont pas systématiques” [SENACH 90a]. De plus l'évaluation est fortement dépendante du contexte, c'est-à-dire du public auquel s'adresse le système à évaluer.

Les deux dimensions d’évaluation, décrites par B. Senach [SENACH 90a], sont recueillies par des moyens divers tels que les questionnaires, les capteurs ou les observations. Elles peuvent également être obtenues par l'intermédiaire des prototypes qui permettent un réalisme dans les simulations ainsi qu'une détection précoce de problèmes potentiels. De la sorte il est possible par exemple d'adapter très tôt le dialogue homme-machine en fonction des habitudes ou du niveau du public. Pourtant les prototypes contiennent certaines limites dans leurs simulations [SENACH 90a] [SENACH 90b] :

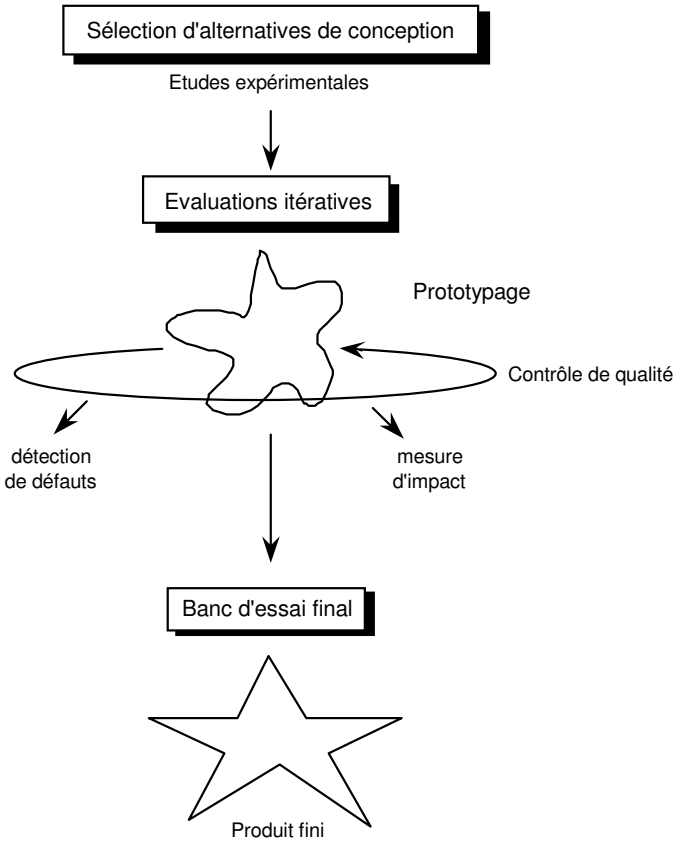


Figure 1.5 : Les trois types d'évaluation

- les conditions de travail ne sont pas reproduites et donc il est impossible d'étudier le comportement réel d'un utilisateur stressé ou fortement motivé,
- les tests ne portent généralement que sur des produits isolés, alors que la compatibilité entre les divers systèmes de l'environnement de l'utilisateur est une dimension essentielle de l'utilisabilité [NIELSEN 91], [NIELSEN 92a], [NIELSEN 92b], [NIELSEN 93a], [NIELSEN 93b],

- les tâches testées ne sont que des simplifications des tâches réelles auxquelles aura à faire face l'utilisateur. De plus les conditions de travail viendront modifier ces tâches (flou de certaines situations, tâches multiples, etc.) et il n'est pas possible de prédire les difficultés ultérieures rencontrées sur site,
- le prototype donne un aspect immédiat du futur produit mais il ne tient pas compte de la phase d'apprentissage ainsi que de l'utilisation intensive du produit final qui font souvent apparaître les véritables difficultés.

4.4. Perspectives à venir

4.4.1. Génération automatique

Quelques produits actuels permettent déjà de générer en partie les applications interactives. Certains génèrent la représentation externe [PETOUD 90], d'autres la description lexicale et syntaxique de l'application [SINGH 89], d'autres encore le module de contrôle du dialogue homme-machine [NORMAND 92a]. Ces travaux montrent tout d'abord que l'idée de générer automatiquement une application n'est pas du domaine de l'utopie, et ensuite que la génération se situe à des niveaux d'abstraction de plus en plus hauts. On peut donc espérer un jour arriver à une génération totale d'application à partir seulement de contraintes et d'objectifs à réaliser. Cet ensemble permettra de générer des spécifications précises en fonction du contexte (matériel, public, etc.) qui à leur tour fourniront par le biais d'outils choisis automatiquement une implémentation portable ou bien propre à un matériel. Cette implémentation pourra être basée sur des composants intelligents, entièrement et directement réutilisables (tels des composants électroniques) et groupés dans des architectures multi-expertes [SADOU 91].

4.4.2. Évaluation

Il existe des outils qui permettent d'évaluer a posteriori une interface [BALBO 92a] [BALBO 92b] [PALANQUE 92] [POLLIER 91]. Grâce aux recherches en représentation des connaissances, sur les systèmes d'aide à la décision et en ergonomie, on peut espérer voir un jour apparaître des outils d'aide à la conception qui permettront des développements d'applications interactives plus efficaces. On peut supposer que ces outils fourniront des conseils sur le choix de la représentation externe (par exemple le choix d'une police de caractères) et de la représentation conceptuelle (par exemple le modèle de données à utiliser). On peut espérer qu'ils donneront une évaluation de la qualité de l'interface tout au long du processus de développement et qu'ils dispenseront des conseils pour des améliorations a posteriori par exemple dans le cas d'une évolution des fonctionnalités.

Un nouveau type d'évaluation est en pleine expansion : l'évaluation a priori. Elle permet de dire s'il est intéressant de développer tel ou tel système en fonction du public visé, des contraintes matérielles ou humaines, etc. Ce type d'évaluation est basé en grande partie sur la simulation cognitive et permet d'évaluer aussi bien l'utilité que l'utilisabilité.

4.4.3. Interfaces adaptatives et interfaces expertes

La génération automatique permet déjà de créer des interfaces adaptées à un type d'utilisateur connu à l'avance. Ce type d'interface est appelée interface adaptable c'est-à-dire propre à un type particulier (*adaptabilité*). La notion *d'adaptativité* est plus forte ; elle signifie qu'une application s'adapte automatiquement à un nouveau contexte (nouveau matériel, nouveaux utilisateurs, etc.). Ce concept est déjà en cours de recherche depuis de longues années et on le retrouve par exemple

dans les systèmes tutoriels intelligents qui s'adaptent au niveau de leurs élèves grâce à leur modèle. D'autres produits prennent en compte les connaissances ou les méta-connaissances de l'utilisateur [BRISSON 90]. La faculté d'intégration que possèdent certains produits [BERRADA 92] est également une sorte d'adaptativité puisqu'elle correspond à des adaptations de natures différentes. On peut espérer voir un jour apparaître des produits capables d'identifier les utilisateurs et ainsi se mettre à leur niveau, les conseiller en fonction de ce niveau, etc.
Deux tendances s'affrontent actuellement : bâtir des interfaces capables de gérer toutes sortes de produits [BERRADA 92] ou bien construire des interfaces spécialisées dans un domaine. La solution serait peut être de combiner les deux, de sorte à avoir une architecture répartie qui serait composée d'un ensemble d'interfaces expertes dans un domaine [BRAJNIK 86] et gérée par une interface intégrant toutes ces sous-interfaces. Avec ce système il serait possible d'avoir une machine qui pourrait résoudre des problèmes mathématiques, créer des images de synthèse, animer des séquences vidéo, servir pour l'interrogation en langage naturel de bases de données, etc., le tout avec une interface homogène.

4.4.4. Communication multi-média et multi-modale

Il y a quelques années la reconnaissance d'écriture était encore du domaine de l'imaginaire. Pourtant des ordinateurs portables et sans clavier se vendent aujourd'hui au même prix que des micro-ordinateurs[18]. De la même façon, la traduction automatique des langues étrangères paraissait impossible[19] ; pour quelques centaines de francs on peut acheter aujourd'hui des traducteurs (de mots ou d'expressions, il est vrai !) qui comprennent cinq ou dix langues. D'autre part il existe d'ores et déjà des produits commandés par la voix et l'œil. On peut donc envisager dans un avenir proche des machines entièrement commandées par la voix et les gestes et capables d'être utilisées dans tous les pays.

5. Conclusion

Le besoin d'interactivité, la volonté de faciliter l'apprentissage et le développement, l'apparition de nouvelles technologies, tous ces éléments ont contribué à l'évolution des applications interactives et plus particulièrement à celle des IHM. De nouvelles méthodes et de nouveaux outils sont apparus. De plus en plus les concepts d'objet et de multimédia envahissent les IHM, et on peut se demander si l'évolution technologique ne va pas remettre en cause tôt ou tard les concepts de base. Notre nécessité de décomposer un problème en étapes et en sous-problèmes nous a amenés à travailler en termes de spécification, de conception, etc. Puis sont apparus des outils qui permettent de réaliser ces tâches de manière assistée. Ne va-t-on pas finir par gommer la production horizontale ou bien encore remplacer les représentations conceptuelle, externe et interne par d'autres vues spécialement conçues pour de nouvelles technologies ? Il apparaît pour l'instant que l'évolution des outils actuels semble condenser les phases de spécification, conception et réalisation en une seule étape grâce à la notion d'objet, de multi-agent et aux possibilités de génération. Il paraît cependant évident que la phase de spécification sera d'une manière ou d'une autre toujours nécessaire.

---

[18] La reconnaissance de l'écriture a permis également la reconnaissance des gestes. Ainsi il existe des portables sans clavier qui reconnaissent certains gestes, ceci d'une part afin d'éviter le transport d'un clavier et d'une souris, d'autre part afin de rester homogène quant aux techniques d'interaction qui utilisent généralement des stylos optiques. Par exemple, le fait de dessiner une croix sur un élément a pour effet de le supprimer.

[19] On peut rappeler à ce sujet la traduction automatique de l'anglais en russe puis du russe en anglais de la phrase "L'esprit est fort mais la chair est faible" qui donnait "La vodka est forte mais la viande est pourrie" !

Chapitre 2
Le Dialogue Homme-Machine

1. Introduction

Dans le chapitre précédent, nous avons évoqué la séparation entre l'interface[20] et l'application sous-jacente. Cette séparation a fait apparaître le problème de la gestion du dialogue homme-machine. Comment le représenter et le stocker ? Vaut-il mieux le disperser dans l'application, le laisser à la charge de l'interface ou bien encore créer un module chargé spécialement de sa gestion ? Bien que certains produits utilisent encore aujourd'hui l'une ou l'autre des deux premières solutions, le cas le plus fréquent est le dernier ; le modèle de Seeheim (cf § 4.3.1.1) en est le premier représentant. Disposer d'un module dédié permet une gestion plus simple du dialogue puisque ce dernier est concentré en un lieu unique. Pour la même raison, le comportement de l'interface peut être facilement modifié. Cette propriété de concentration de la gestion du dialogue permet également la spécialisation du dialogue. On peut ainsi disposer d'un ou plusieurs modules, chacun spécialisé dans un domaine et pouvant communiquer avec les autres. Nous verrons que cette propriété se retrouve souvent ; c'est le cas par exemple du modèle Arch (cf § 4.3.3.1) ou des systèmes multi-agent (cf § 4.3.3.2). Grâce à des techniques telles que la programmation objet et la programmation événementielle par exemple, il est possible de faire communiquer des entités de natures différentes ou bien encore de dialoguer simultanément avec plusieurs entités qui peuvent être ou non concurrentes[21].

Nous allons tout d'abord définir exactement le sens que nous donnons au terme dialogue homme-machine dans cette thèse puis nous présenterons de manière générale le dialogue homme-machine dans le contexte informatique ainsi que les caractéristiques requises pour un contrôleur de dialogue. La troisième partie décrit certains modèles d'application axés sur la gestion du dialogue homme-machine. La dernière partie est consacrée aux formalismes utilisés pour la conception et l'implémentation de dialogue. Elle est composée de formalismes basés sur les états et les événements.

2. Qu'est-ce que le dialogue ?

Le terme **dialogue** est devenu aussi courant que celui de souris ou objet. Pourtant sa définition varie suivant les auteurs. Nous donnons ici trois définitions générales qui résument bien la différence des notions abordées dans le domaine des IHM[22].

---

[20] Dorénavant nous désignerons par *Interface* l'ensemble des modules (présentation, contrôleur de dialogue, etc.) qui se greffent sur le noyau applicatif et qui permettent à l'utilisateur de manipuler le noyau applicatif. Lorsque nous voudrons désigner la partie externe de l'interface (ce que voit l'utilisateur), nous emploierons les termes *Présentation* ou *Représentation Externe*.

[21] Il est ainsi possible de dessiner en travaillant simultanément avec la souris et le clavier [HILL 86], ou effectuer une mise à jour simultanée de plusieurs zones à l'écran (fenêtres, listes, boutons...) ou bien encore gérer simultanément du graphisme et du son.

[22] Les autres aspects du dialogue (par exemple en linguistique) ne sont pas abordés dans cette thèse.

> *Définition 1* : Le **dialogue** est l'ensemble des échanges entre un utilisateur et une machine (ou un ensemble de logiciels).

Le terme *échange* regroupe ici des notions telles que les manipulations avec un écran tactile ou l'utilisation d'un microphone, d'un scanner, d'une table à digitaliser, etc. Cette définition est une définition générale qui intègre les définitions 2 et 3.

> *Définition 2* : Le **dialogue** est l'ensemble des interactions produites par un humain sur un ordinateur.

Une **interaction** est une *action* qui provoque une *réaction* perceptible. Elle regroupe des notions telles que les clics souris, la frappe de touches, les pressions sur un écran tactile, etc. Cette définition est la plus courante en matière d'IHM et c'est elle que nous utiliserons par défaut dans la suite de cette thèse.

> *Définition 3* : Le **dialogue** est l'utilisation du langage naturel pour travailler sur un ordinateur.

Cette définition restrictive ne s'applique qu'au domaine de recherche sur le langage naturel. Le but de cette recherche est de parvenir à travailler avec une machine par l'intermédiaire d'ordres énoncés dans la langue de tous les jours, l'ordinateur répondant également dans cette langue. Ce dialogue peut se faire de plusieurs façons : écrite, orale ou les deux à la fois. Il existe déjà des logiciels tels que le tableur Lotus sur lesquels on a interfacé un module de reconnaissance vocale. Plusieurs sociétés sont d'ailleurs intéressées par cet aspect ; c'est le cas par exemple d'Apple qui projette de fournir le prochain système pour Macintosh (voire une nouvelle machine) avec une commande vocale intégrée.

La commande vocale n'est qu'un aspect du langage naturel. Elle ne correspond qu'aux échanges humain-machine et non machine-humain. On ne peut donc pas parler véritablement de dialogue. Pourtant ce type de recherche n'est pas très loin du nôtre puisque le système informatique subit les commandes énoncées par l'humain tout comme un logiciel interactif subit les actions de l'utilisateur.

Quelle que soit la définition que l'on puisse donner du dialogue, la notion de communication est un dénominateur commun.

> *Définition* : La **communication** est un processus reposant sur un échange entre deux ou plusieurs entités, utilisant un code et rendant compréhensible une information transmise d'un émetteur à un récepteur.

Cette définition générale s'applique parfaitement à notre sujet où les entités sont d'une part un humain et d'autre part une machine, l'information est composée de données informatiques telles qu'un numéro de client ou un fichier d'articles, et le code utilisé est le dialogue que nous avons expliqué dans la définition 2. Ce code peut suivre des règles de présentation (cliquer sur un menu provoque l'affichage de ce dernier), des règles sémantiques (on ne pourra pas enregistrer un client tant qu'on n'a pas rempli sa fiche), etc. La gestion de ces règles peut relever de l'application ou bien de l'interface.

Dans cette thèse, le dialogue est donc l'ensemble des interactions d'un humain sur une machine. Cette définition cache une notion fondamentale, la gestion des tâches. Gestion des interactions et

gestion des tâches sont en effet plus ou moins indissociables puisque les widgets[23] sont manipulés dans le dessein de réaliser des tâches. La gestion des interactions correspond simplement à un passage obligatoire pour la gestion des tâches (les tâches peuvent être représentées de plusieurs façons comme nous le verrons dans le Chapitre 5). Quel que soit le type de représentation, le contrôleur de dialogue est la plupart du temps responsable de leurs enchaînements et de leurs déclenchements.
On distingue trois types de tâches [RAS 83] :

- les *tâches procédurales* (ou *familières*) : elles sont les plus faciles à traiter puisqu'on peut facilement les traduire sous forme algorithmique, donc par l'intermédiaire de procédures. Les liens entre procédures sont connus à l'avance et le hasard n'intervient pas dans les enchaînements. Les tâches bureautiques en sont un exemple classique. Plusieurs méthodes permettent de traiter ce type de tâches : Merise [TARDIEU 85] [TARDIEU 86], Axial [PELLAUMAIL 86], SADT [MARCA 88]... Dans le reste de ce rapport, nous nous intéresserons uniquement à cette catégorie de tâches.

- les *tâches créatives* (ou *à planification opportuniste*) : contrairement aux précédentes, elles laissent une totale liberté à l'utilisateur (forte latitude décisionnelle) ce qui implique qu'il est pratiquement impossible de connaître à l'avance les enchaînements entre les procédures et entre les objets (au sens large) manipulés. Pour traiter ce type de tâches, on traduit les traitements élémentaires en procédures et la gestion de ces procédures passe par un ou plusieurs modules spécialisés. M. Berrada Tazi décrit ce type de tâches ainsi que leur gestion dans l'environnement HyperStation [NANARD 90] [BERRADA 92]. Il utilise des scripts de traduction qui lui permettent de faire communiquer des entités de types différents ne se connaissant pas. Il parvient ainsi à dessiner en trois dimensions la représentation d'une molécule à partir de sa formule qu'il passe comme paramètre à un module de dessins 3D.

- les *tâches d'aide à la décision* (ou *expertes*) : de même que les tâches précédentes, elles laissent une grande latitude décisionnelle à l'utilisateur. Cependant cette latitude est plus faible du fait du caractère d'aide fournit par le système environnant. Tantôt l'utilisateur devra faire des choix, tantôt il sera guidé par le système. Généralement les traitements élémentaires sont couplés avec un moteur d'inférence qui guide le travail. Pour traiter ce type de tâches, on a recours à des méthodes spécialisées telles que la méthode KADS [HICKMANN 89] [BRUNET 91].

La gestion des tâches procédurales passe le plus souvent par la planification hiérarchique [SACERDOTI 74] [SACERDOTI 77] (Figure 2.1). On établit une sorte de plan d'action pour atteindre un objectif. Pour une tâche donnée (enregistrement de client), on définit un but principal[24] (enregistrer un client). Ensuite on affine ce but en le décomposant en sous-buts (enregistrer le numéro de code client, les renseignements habituels, son taux de remise, etc.). Cette décomposition en sous-buts se poursuit jusqu'à l'obtention d'opérations élémentaires ou de suites d'opérations élémentaires. Chacune de ces opérations possède en général au minimum des conditions associées à son déclenchement (pré-conditions).

[23] Un widget (abréviation de "window gadget") est un élément graphique qui permet à l'utilisateur de communiquer avec l'application qui les gère. Les exemples les plus courants sont les boutons et les icônes.
[24] Cette façon de traiter un problème est impossible avec les autres types de tâches puisque le but n'est pas connu à l'avance et est susceptible d'évoluer à tout instant, surtout pour les tâches créatives.

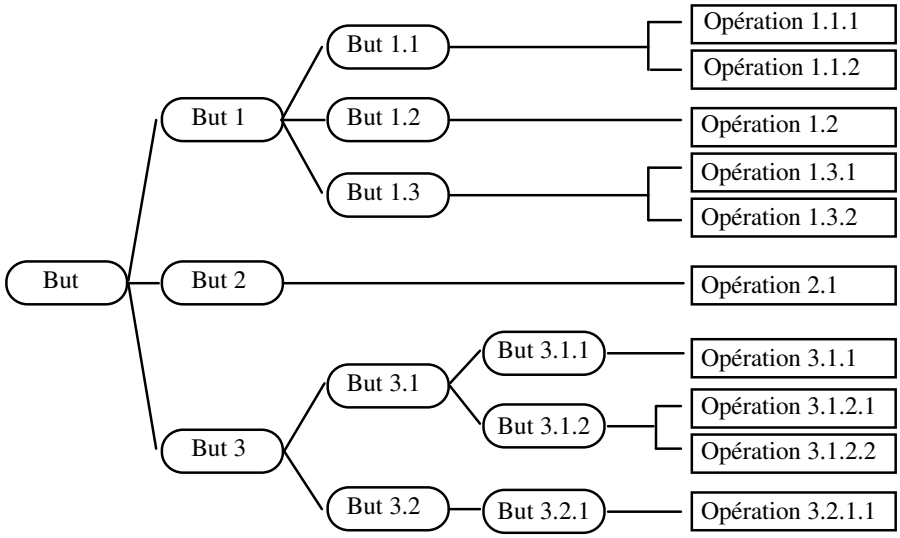


Figure 2.1 : La planification hiérarchique

Bien que fortement structurée, la planification hiérarchique laisse une marge de manœuvre à l'humain ; elle autorise des choix, des possibilités de séquentialités, etc., tout en gardant comme objectif premier la réalisation du but principal. Nous verrons dans le Chapitre 5 que plusieurs méthodes (dont Diane) reprennent la planification hiérarchique pour la gestion des tâches.

3. Le contrôleur de Dialogue

Nous appellerons **Contrôleur de Dialogue**[25], dans le reste de cette thèse, le module (ou l'ensemble des modules) dédié au dialogue. Ce module a deux rôles principaux : l'analyse et le contrôle du dialogue.

3.1. L'analyse du dialogue

On distingue trois types d'analyse : lexicale, syntaxique et sémantique. Cette décomposition, résultant des recherches en linguistique, permet de décoder et d'interpréter tout langage qui respecte au moins des règles de syntaxe et de grammaire. On a voulu appliquer cette décomposition aux langages informatiques qui respectent ce principe. Les trois analyses pré-citées permettent ainsi d'écrire des compilateurs et des interpréteurs de bonne qualité, ce qui incite à appliquer cette même décomposition plus spécialement au dialogue homme-machine. La plupart des modèles d'application que nous verrons plus loin associent chacune des analyses à un module, parfois même les regroupe dans un seul. Pourtant lorsque deux personnes discutent et que l'une énonce une phrase correcte sémantiquement et incorrecte syntaxiquement, l'autre personne peut toutefois la comprendre alors que l'inverse n'est pas forcément vrai[26]. Ainsi il n'est pas nécessaire d'analyser d'abord la syntaxe puis la sémantique pour comprendre une phrase. En fait les trois analyses se font de façon simultanée et en inter-relation. Ce phénomène pourrait être reproduit dans les IHM. Pourtant tous les modèles utilisent une analyse modulaire de deux ou plusieurs niveaux. De fait, l'architecture qui en ressort en est grandement influencée. On y trouve

[25] Nous reprenons ici le terme utilisé dans le modèle de Seeheim exposé plus loin.

[26] Par exemple la phrase "J'ai venu en voiture" est facilement rapprochée de "Je suis venu en voiture", alors que "Une maison vole très ancien" ne veut rien dire.

généralement autant de modules de traitement que de décompositions. Par exemple, le modèle de Seeheim utilise une analyse à trois niveaux possède trois modules, chacun dédié à une analyse particulière.

Les recherches actuelles tendent à utiliser le modèle multi-agent qui permet de gérer simultanément plusieurs analyses et surtout de les faire communiquer. Plutôt que de disposer de trois modules (ou plus) chargé chacun d'une analyse particulière (lexicale, syntaxique ou sémantique), il est alors possible de disperser ces analyses au gré des agents qui en sont responsables. Bien que la gestion centralisée du dialogue (telle que celle de Seeheim) permette une maintenance plus aisée, elle implique le passage obligatoire de toutes les informations par le module dédié, d'où un nombre important d'échanges qui souvent ne sont pas nécessaires. On peut en effet envisager deux points de vue différents pour l'analyse du dialogue : depuis l'application ou depuis l'interface. Chacun de ces deux points de vue possède une sémantique, une syntaxe et un lexique. On peut alors imaginer un module dédié à l'analyse pour chacun de ces points de vue et non pas pour chacune des analyses.

En conclusion nous pouvons dire qu'il paraît judicieux de séparer l'analyse du côté application de l'analyse du côté interface. Il est en effet toujours possible de gérer des objets de présentation indépendamment des objets destinés à représenter les données (chacun ayant leur propre sémantique et syntaxe). Par contre il est impossible à l'heure actuelle de dire s'il faut ou non déporter certaines caractéristiques des données dans l'interface. Ce cas peut se révéler particulièrement intéressant pour les applications multi-utilisateurs où l'application est implantée sur un site central. Dans ce cas on peut imaginer l'application possédant un minimum de contrôles vitaux ainsi qu'un minimum de présentation pour les données, alors que chaque poste posséderait une interface qui lui serait propre (donc redéfinissable) et capable de gérer une partie de la syntaxe (voire même toute la syntaxe) ainsi qu'une partie de la sémantique (par exemple la sémantique commune à plusieurs applications). Ainsi une interface saurait toujours comment saisir une date ou un numéro INSEE sans être obligée de le redéfinir dans chaque application. L'avantage de cette technique est d'avoir des retours d'information beaucoup plus rapides puisqu'ils ne transitent plus par des liens physiques de longueurs non négligeables.

3.2. Le contrôle du dialogue

Le contrôle correspond à la validité des déclenchements des opérations. Cette validité dépend de deux paramètres : les données et les traitements antérieurs. D'une manière plus générale on peut dire que la validité des déclenchements dépend du contexte. Si on suppose que les trois analyses que nous venons de décrire sont réparties dans plusieurs modules, ce contrôle est considérablement réduit. Une commande ne peut être accessible à l'écran que si le contrôleur l'a autorisée. Par conséquent un bouton non grisé signifie que son déclenchement est possible et inversement un bouton grisé signifie que son déclenchement est impossible. Dans les deux cas, l'interface a été informée de la présentation à donner au bouton. Le contrôleur a pu décider seul de cette information ou bien celle-ci lui a été communiquée par l'application. Dans le premier cas, la validité du déclenchement dépend des opérations antérieures puisque le contrôleur est chargé principalement de gérer les enchaînements de tâches. Dans le second cas, la validité du déclenchement dépend des données puisque c'est l'application qui est responsable de la gestion des données (au niveau sémantique principalement).

Le contrôle du dialogue signifie également la gestion des enchaînements des opérations suite aux interactions de l'utilisateur. La planification hiérarchique permet de spécifier le chemin à parcourir pour atteindre un but. Elle peut également expliciter l'ordre dans lequel les sous-buts ou les

opérations élémentaires doivent être déclenchés. Cet ordre doit être respectée par le contrôleur de dialogue. Ainsi le fait d'avoir achevé une opération permet d'en exécuter d'autres et ceci doit se répercuter à l'écran par exemple en rendant un menu, un bouton ou une zone de saisie accessibles. Il est donc intéressant que le contrôleur connaisse les enchaînements nécessaires pour atteindre les buts. Nous verrons dans les Chapitres 5 et 6 comment nous le représentons et le gérons avec Diane+.

En résumé le contrôleur de dialogue a pour fonction de vérifier la validité des interactions et de gérer les enchaînements de traitements qui en découlent.

3.3. Caractéristiques d'un contrôleur de dialogue

L'augmentation de l'ergonomie, de l'interactivité et de la manipulation directe dans les applications a permis à un public de plus en plus nombreux, mais surtout de plus en plus varié, de manipuler des logiciels. La diversité de ce public a obligé les concepteurs à prendre en compte tous les types de public au sein d'une même application alors qu'auparavant c'était le public qui s'adaptait au logiciel. La gestion du dialogue homme-machine est donc devenu un travail à part entière et doit respecter certaines propriétés d'ordre général qui permettent de définir un bon contrôleur de dialogue. En fait on retrouve les critères du génie logiciel, c'est-à-dire validité, robustesse, extensibilité, réutilisabilité, etc. Nous donnons ci-dessous quelques-uns des plus importants [SCAPIN 87] :

- *flexibilité* : à cause de la diversité de compétence et d'assiduité du public, le contrôleur doit pouvoir fournir des types de dialogue différents en fonction du type du public [CARLSON 77] [THIMBLEBY 80] [BENBASAT 82] [CAREY 82] [BAILEY 83] [FALZON 89]. Par exemple un débutant n'utilisera que les menus alors qu'un expert utilisera de préférence les raccourcis clavier ou les touches de fonction. Nous verrons dans le Chapitre 5 que les procédures effectives de Diane permettent aussi de répondre à ce critère.

- *adaptabilité* : le contrôleur de dialogue doit être capable de s'adapter au niveau et au style de son interlocuteur. Deux types de solutions sont possibles : soit plusieurs styles sont directement implémentés dans le système et celui-ci en choisit un en fonction de l'utilisateur, soit il n'y a que des règles (au sens large) de style et c'est en fonction de l'utilisateur que le contrôleur les applique ou non [TREUD 76] [THIMBLEBY 80] [MOZEICO 82]. Ce critère est primordial dans les systèmes tutoriels intelligents puisque ceux-ci doivent non seulement reconnaître le niveau des élèves, mais également se mettre à leur niveau, donc s'adapter.

- *intégrité* : le contrôleur de dialogue doit être capable de protéger le système contre des accès ou des modifications non autorisés. C'est une des propriétés fondamentales d'un contrôleur de dialogue.

- *recouvrement* : il doit être possible de récupérer des erreurs faites durant une session de dialogue avec relativement peu d'efforts. Bannon [BANNON 83] reconnaît que la possibilité de l'utilisateur à suspendre et reprendre des opérations est critique dans la performance des tâches. La fonction défaire/refaire (Undo/Redo) que l'on trouve de plus en plus dans les IHM en est l'exemple le plus frappant. Cette fonction est cependant le plus souvent implémentée de façon ad hoc ; il serait intéressant de disposer d'outils permettant de la générer à partir de ses spécifications. Un palliatif déjà utilisé consiste à conserver un historique des actions qui ont été exécutées depuis le début de la session ou sur plusieurs sessions. Il est donc possible dans certains cas de défaire une action et d'annuler toutes celles qui ont suivi [PETOUD 90].

- *partage* : à partir d'une même structure de contrôle de dialogue, il doit être possible de travailler simultanément sur plusieurs tâches d'une même application ou de plusieurs applications [KUO 85b]. On parle alors de dialogues multi-fil[27]. Par contre, le multi-fil implique la gestion de tâches concurrentes qui n'est pas toujours facile à traiter.

- *robustesse* : elle représente l'aptitude du contrôleur de dialogue à fonctionner dans des conditions anormales. Ceci implique la gestion des erreurs prévisibles telles que les erreurs d'intention[28] et d'exécution[29] et celle des erreurs imprévisibles telles que les interruptions de tâches, etc.

3.4. Lien entre le contrôleur de dialogue et le noyau fonctionnel

Comment s'établit le lien entre le contrôleur de dialogue et le noyau fonctionnel ? Le modèle de Seeheim (cf § 4.3.1.1) utilise un modèle spécifique qui sert d'interface avec l'application. Des modèles tels que MVC (cf § 4.3.2.2) et PAC (cf § 4.3.3.3) utilisent l'application sous-jacente par l'intermédiaire des services qu'elle propose. Dans la majorité des cas en effet, l'application est vue comme un ensemble de primitives[30] que l'on peut appeler indépendamment. Ces primitives sont censées être indépendantes de l'interface, c'est-à-dire de la représentation externe que l'interface utilise. L'appel à ces primitives est déclenché par les événements générés lors des interactions, mais leur exécution est gérée par le contrôleur qui vérifie que les pré-conditions (sur les données et/ou les traitements) sont bien réalisées. Dans Mickey [OLSEN 89] les services sont stockés dans des "units" semblables à celles du Turbo Pascal. Ces "units" sont composées d'une partie *interface* qui décrit les caractéristiques des services et d'une partie *implémentation* qui décrit les traitements de ces services. En général la partie interface est publique alors que la partie implémentation est privée.

Les échanges entre les services de l'application et le reste (contrôleur et interface) peuvent se faire de plusieurs manières. Une première méthode demande à chaque service de renvoyer une valeur (facultative) tenant lieu de résultat de son exécution. Une deuxième méthode consiste à utiliser des variables partagées comme dans Mickey. Le module qui a besoin de connaître la valeur d'une variable peut la consulter à tout instant. Une troisième solution est l'utilisation des valeurs actives [KARSENTY 87] qui sont des variables partagées avec mise à jour automatique. Si une valeur active change de valeur, elle avise alors toutes les entités qui en dépendent (propagation de contraintes), permettant ainsi une meilleure cohérence du système. Une dernière solution repose sur l'utilisation de canaux spécialisés dans ces échanges. Ces canaux peuvent être à simple ou double sens, typés ou non, avec ou sans protocole, etc. Dans tous les cas, il y a des entités responsables de la communication entre les modules.

Pendant longtemps le module dédié au dialogue a été écrit entièrement par le programmeur d'où une perte de temps de par l'écriture et les erreurs de programmation. Petit à petit des outils apparaissent et permettent d'aider le programmeur à réaliser le contrôleur de dialogue. Nous

---

[27] "Multi-threads" en anglais.

[28] Une erreur d'intention consiste à vouloir exécuter un traitement qui sémantiquement n'est pas valide, par exemple vouloir coller le presse-papier alors que ce dernier est vide.

[29] Une erreur d'exécution consiste à exécuter un traitement suite à une fausse manipulation, par exemple sélectionner une mauvaise commande dans un menu.

[30] Une primitive est un traitement élémentaire qui ne fait appel à aucun événement extérieur autre que celui qui l'a appelé. Par ailleurs une primitive peut appeler une autre primitive.

verrons dans le Chapitre 3 que certains de ces outils ont comme point de départ des spécifications, alors que d'autres utilisent plutôt des maquettes d'écran.

4. Les modèles d'applications interactives

Les modèles d'applications et d'interfaces sont nombreux. Certains proposent des architectures, d'autres tentent de représenter les caractéristiques d'un système. Nous allons pour notre part nous focaliser ici sur les modèles qui ont recours à un contrôleur de dialogue. Celui-ci pourra être constitué d'un module particulier ou au contraire être dispersé entre tous les éléments du modèle. Le but de ce paragraphe n'est pas de dire qu'un modèle a un pouvoir d'expression plus grand qu'un autre ou qu'il est plus puissant, plus rapide, plus fiable, etc., nous voulons simplement mettre en relief les différentes techniques de représentation du contrôle du dialogue, ceci après avoir présenté rapidement les différents types de contrôle de dialogue à l'intérieur d'une application interactive.

Le terme **modèle** a souvent plusieurs sens. Parmi les modèles existants, on peut distinguer des modèles conceptuels, des modèles d'implémentation, des modèles génériques, des modèles occurrences[31], etc. Bien souvent la littérature informatique, et plus spécialement celle concernant les IHM, a tendance a faire un amalgame de tous ces termes. On peut cependant regrouper les modèles en deux catégories par rapport à leur capacité de modélisation d'un système : les modèles conceptuels et les modèles d'implémentation (Figure 2.2).

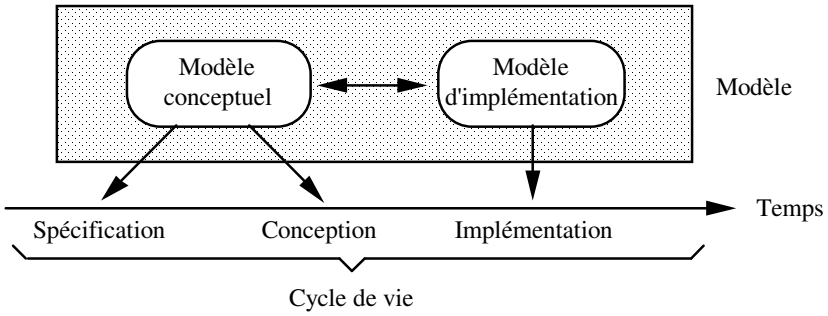


Figure 2.2 : Les deux catégories de modèles pour la modélisation d'un système

Le modèle conceptuel, comme son nom l'indique, permet de représenter des concepts. Ces derniers peuvent d'ailleurs être issus des besoins des programmeurs. C'est le cas par exemple du concept objet qui est apparu par l'intermédiaire des programmeurs qui avaient besoin de réutiliser du code. Les programmeurs ont alors utilisé des formalismes d'implémentation pour gérer ces nouvelles notions. Puis ces notions d'objets ont été formalisées et sont devenues des concepts avec leurs propres modèles, leurs propres formalismes et leurs propres méthodes.

Les modèles conceptuels ont l'avantage de permettre la validation d'un système sans se préoccuper de la technique d'implémentation. Ceci permet de disposer de modèles différents suivant l'étape du cycle de vie dans laquelle on se situe. Bien qu'il soit difficile de faire des subdivisions au sein de ces modèles, on peut toutefois distinguer des modèles conceptuels qui sont plutôt axés sur la spécification et des modèles conceptuels qui sont plutôt axés sur la conception. Ainsi le modèle de Seeheim peut être vu comme un modèle de conception plutôt qu'un modèle de spécification. Par

[31] Ces termes seront définis plus précisément dans le § 4.1.

ailleurs il ne peut être classé en aucun cas comme modèle d'implémentation[32]. A l'opposé, le modèle Arch est plus proche d'un modèle d'implémentation que d'un modèle de spécification. A l'heure actuelle il n'existe pratiquement aucun modèle de spécification pour les IHM contrairement aux modèles de conception et d'implémentation dont le nombre ne cesse de croître. Il existe cependant des méthodes et des formalismes qui permettent de spécifier une application interactive, c'est le cas par exemple de Diane [BARTHET 88] et des Réseaux de Petri à objets [SIBERTIN 85].

Le fait de disposer de modèles différents de la spécification jusqu'à l'implémentation permet à la fois d'utiliser des modèles adaptés à chaque étape du cycle de vie et de réutiliser des modèles ayant fait leurs preuves. Ainsi le modèle de Seeheim met en valeur la séparation entre la présentation et le noyau fonctionnel. Par contre son implémentation se révèle impossible directement de par son laconisme. On peut alors utiliser un modèle intermédiaire tel que le modèle Arch qui sera à son tour implémenté par un modèle particulier (un modèle objet par exemple). Ainsi on peut réaliser le passage d'un modèle trois couches (présentation, contrôleur de dialogue et noyau fonctionnel) à un modèle uniforme (les objets peuvent contenir à la fois les données, les traitements et les présentations associées). La Figure 2.3 résume cette situation.

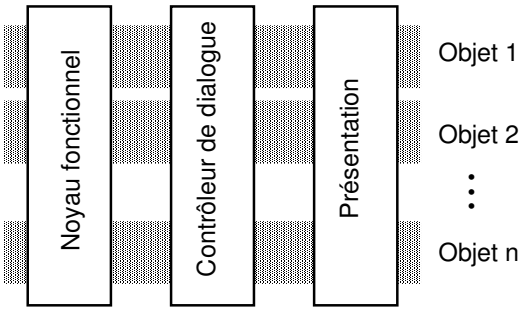


Figure 2.3 : Passage d'un modèle à un autre lors du cycle de vie

L'idéal serait de disposer d'un modèle qui couvre tout le cycle de vie et qui fournisse une granularité plus ou moins fine suivant les étapes (grosse pour la spécification et très fine pour l'implémentation). Pour l'instant un tel modèle n'existe pas et les travaux s'orientent vers des tentatives d'unification de modèles existants ou bien encore vers des équivalences entre formalismes ce qui permet par exemple de convertir des Réseaux de Petri en diagrammes de transitions. L'avantage de cette dernière solution est d'une part la possibilité d'une traduction automatique de ces modèles et d'autre part la validation formelle. La traduction automatique permet d'utiliser un modèle auquel sont habituées certaines personnes (par exemple les concepteurs) tout en l'implémentant par l'intermédiaire d'un autre qui peut être plus économique en mémoire ou plus rapide. La validation formelle permet d'effectuer des contrôles qui sont impossibles avec certains modèles. Ainsi des diagrammes de transitions peuvent se révéler invalidables à cause d'une explosion combinatoire des états alors que le même diagramme mis sous la forme d'un Réseau de Petri pourra tout à fait être validé.

4.1. Modèles, formalismes et cycle de vie

La littérature actuelle en informatique énumère souvent de nombreux modèles et de nombreux formalismes utilisés en IHM. Rares sont les articles qui tentent de les comparer scrupuleusement.

[32] Nous verrons plus loin (cf § 4.3.1.1) les raisons de cette impossibilité et les modifications qui ont été apportées pour tenter de trouver une solution à ce problème.

La raison en est simple : pour comparer deux éléments, il faut avoir les mêmes bases de comparaison. Or, bien souvent les articles se contentent de donner des caractéristiques générales des modèles (nombre et rôles des modules composant le modèle, etc.) sans donner d'évaluation précise de leur pouvoir d'expression ou de leur capacité à être implémentés. Cette difficulté à comparer les modèles (et donc les applications qui les utilisent) provient du nombre important des paramètres qui permettent la création des modèles. La figure 2.4 résume la situation. Une personne qui observe un système réel, le perçoit en fonction de ses capacités cognitives et cherche à le représenter en fonction des objectifs qu'elle y associe. Pour cela elle dispose de modèles généraux (modèles génériques) qu'elle utilise comme filtre sur le système réel. Il en ressort alors un modèle qui est une occurrence du modèle général utilisé. L'occurrence permet de représenter un certain nombre d'éléments du système réel (au plus le même nombre que le modèle générique et que le système réel). Les caractéristiques du modèle occurrence et le nombre d'éléments représentés nous permettent d'évaluer son pouvoir d'expression et ainsi de le comparer avec d'autres modèles occurrences qui auraient pu être employés.

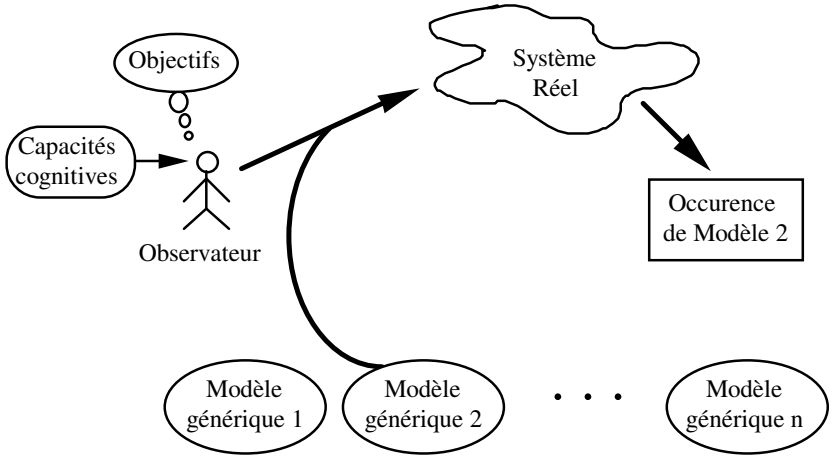


Figure 2.4 : Modèles génériques et modèles occurrences

Pour que les comparaisons soient efficaces, il convient de les étaler tout au long du cycle de vie d'une application et donc il est nécessaire d'utiliser des critères différents suivant les étapes :

- *la spécification* : elle donne les caractéristiques générales et détaillées du système ainsi que son comportement par rapport à l'utilisateur. Nous avons vu qu'il n'existe pas de modèle de spécification en IHM, mais des méthodes telles que Merise ou Axial répondent aux besoins de la spécification. Le but recherché avant tout est la validation des spécifications par l'utilisateur. Pour cela on définit les objectifs à réaliser sous forme de boîtes noires qu'on affine jusqu'à un niveau de détail maximal, toujours par rapport à l'utilisateur. Ces boîtes noires font intervenir des événements et des contraintes de gestion que le modèle (ou la méthode) de spécification doit être capable d'intégrer. Par ailleurs le modèle doit pouvoir suivre l'évolution du système réel au cours du temps (par exemple un changement dans les règles de gestion). Les spécifications terminées, on obtient les concepts et les fonctionnalités détaillés du système modélisé. Pour évaluer le modèle utilisé, on peut se baser sur les critères suivants (toujours par rapport à l'utilisateur) :
  - le modèle est simple et lisible,
  - le modèle est validable, c'est-à-dire que le système modélisé répond à la demande du client,
  - le modèle résume le système dans ses grandes lignes et il permet d'avoir une vue globale, claire et précise des caractéristiques du système.

La comparaison de modèles de spécification n'est pas facile à cause du niveau d'abstraction élevé. Il est cependant possible d'évaluer le pouvoir d'expression du modèle utilisé, c'est-à-dire ses capacités à représenter les éléments qui composent le système ainsi que leur comportement.

- *la conception* affine la spécification en y apportant le comportement de l'application par rapport à la machine et en détaillant le comportement interne de l'application vis-à-vis de l'utilisateur. La conception permet donc une validation du modèle par rapport à l'informatique. Cette validation est effectuée cette fois-ci par l'informaticien et non plus par l'utilisateur. La conception peut utiliser des modèles tels que ceux de Seeheim ou de Hudson. De par sa proximité de l'implémentation, on préférera utiliser des modèles orientés vers l'implémentation tels que le modèle Arch, le modèle PAC, le modèle multi-agent, etc., ou bien encore des formalismes tels que les diagrammes de transitions, les Réseaux de Petri, les grammaires, etc. L'utilisation de ces représentations permet surtout une validation formelle syntaxique et sémantique du système, ce qui est plus difficile durant la spécification.
  La conception fournit donc une décomposition élémentaire des fonctions et des concepts issus de la spécification. Cette décomposition est directement orientée machine et elle fait intervenir de nouveaux paramètres (par exemple des variables locales à certains traitements).
  Les critères qui permettent de comparer les modèles et les formalismes utilisés à cette étape sont issus du génie logiciel. Ce sont la validité (le système répond à la demande du client et de l'informaticien), la compatibilité (entre les modules, les concepts et les fonctionnalités), la modularité, la lisibilité, et la validabilité (validation formelle).

- *l'implémentation* est la traduction de la conception en terme de code. Elle est réalisée par l'utilisation de formalismes qui peuvent être les mêmes qu'en conception. L'implémentation peut être réalisée complètement par l'informaticien ou bien être générée en partie par des outils tels que les UIMS ou bien encore par des traducteurs de formalismes comme ceux que l'on trouve dans USE.
  Les critères d'évaluation à ce niveau sont essentiellement ceux du génie logiciel puisqu'on se trouve très près de la machine. Ce sont par exemple la robustesse, l'extensibilité, la modifiabilité, la réutilisabilité, l'efficacité, la portabilité, l'intégrité, etc.

Justifier un choix de modèle n'est donc pas simple. Dire que le modèle qu'on utilise est plus judicieux qu'un autre parce qu'il permet un temps de réponse plus court qu'un autre n'est pas suffisant. Cela prouve simplement qu'au niveau de l'implémentation il possède un avantage par rapport au modèle comparé. En fait il est impossible de trouver un modèle parfait et l'informaticien doit constamment jouer sur des compromis entre les critères cités plus haut[33]. La seule façon d'évaluer un modèle est donc de le comparer avec des modèles qui se situent dans la même phase du cycle de vie et suivant les critères qui s'y rapportent. Il devient alors possible de comparer Arch avec PAC ou bien MVC avec le modèle multi-agent.

4.2. Les différents types de contrôle du dialogue

P. Tanner et W. Buxton [TANNER 85] ont défini la notion de contrôle[34] dans les applications interactives comme la partie qui est chargée de gérer les enchaînements des actions[35]. Suivant les

[33] Par exemple si on augmente la liberté d'accès de l'utilisateur, on relâche les contrôles et donc le critère d'intégrité diminue de manière inversement proportionnelle.

[34] Le but de ce paragraphe est d'énumérer les différentes façons de gérer le dialogue à l'intérieur d'une application et non pas comment répartir le dialogue entre l'utilisateur et la machine. Il faut cependant remarquer que le premier influence le second. Nous avons déjà énoncé les différents types de tâches qu'un utilisateur peut exécuter (cf § 2).

auteurs on lui associe différents noms : vérificateur du séquencement correct des actions pour P. Tanner [TANNER 85], pilote de l'application pour M-F. Barthet [BARTHET 88] ou bien encore centre d'arbitrage du flux d'information pour J. Coutaz [COUTAZ 88]. Le contrôle du dialogue peut être de quatre types : interne, externe, mixte ou global.

Une application à **contrôle interne** est une application qui ne présente pas de module dédié au dialogue. Il ne faut pas confondre ce genre d'applications avec celles qui ont un module fonctionnel et un module de dialogue, et qui n'en présentent plus qu'un seul après avoir été compilées. Dans les applications à contrôle interne, le code du dialogue est mélangé avec celui des traitements. On retrouve cette caractéristique dans la plupart des applications qui datent d'avant 1980, c'est-à-dire avant l'apparition sur le marché des IHM de haut niveau. Le contrôle interne implique un dialogue rigide puisqu'il est en quelque sorte câblé dans l'application. De ce fait toute évolution ergonomique (comme le dialogue multi-fil) ou matérielle (comme l'utilisation d'une souris au lieu du clavier) est impossible à réaliser sans une refonte complète de l'application.

Le **contrôle externe** permet de dissocier le module de dialogue du module (ou noyau) fonctionnel. Ce dernier devient l'esclave du module de dialogue. L'utilisateur qui communique avec le module de dialogue par l'intermédiaire de l'interface ne fait qu'appeler les fonctions de l'application à son gré et le module de dialogue doit assurer l'utilisation correcte de ces fonctions. Pour réaliser des applications de ce type, il est nécessaire d'écrire des fonctions qui évitent d'appeler elles-mêmes les routines du dialogue, par exemple une fonction qui à la fois afficherait une liste et rendrait accessible un menu.

Le **contrôle mixte** est un compromis des deux types de contrôles précédents. Il correspond à un contrôle réparti entre le module fonctionnel et le module de dialogue. Il est ainsi possible de donner dans certains cas le contrôle à l'application plutôt qu'à l'utilisateur (par exemple pour un utilisateur novice ou non agrémenté pour des accès confidentiels). Ce type de contrôle a une influence directe sur la répartition du dialogue puisque dans certains cas l'utilisateur aura le contrôle et dans d'autres non.

Le dernier type est le **contrôle global**. Il correspond à un module supplémentaire qui commande le module de dialogue et le module fonctionnel. Ces deux modules sont donc esclaves et ils sont vus comme des bibliothèques de routines. Cette architecture est un peu trompeuse puisqu'elle cache le véritable dialogue, mais elle permet d'avoir pour un même module de dialogue des comportements différents par exemple suivant le niveau de l'utilisateur.

Conclusion

Le contrôle interne est de moins en moins courant, mais il est indispensable dans des applications où l'humain a peu ou pas de contrôle sur la machine. C'est le cas par exemple des applications embarquées (satellites) ou consultatives (Minitel) bien que ces dernières tendent à utiliser de plus en plus un contrôle externe. La raison de ce choix est dûe à des raisons évidentes de maintenance et d'évolution. Les contrôles mixtes et externes possèdent en avantage supplémentaire, le “face-lifting”, c'est-à-dire la possibilité d'habiller une application existante avec une nouvelle interface sans toucher au noyau fonctionnel lui-même [BARTHET 92]. Bien que J. Coutaz [COUTAZ 91] pense qu'avec le parallélisme le choix d'un contrôle ou d'un autre n'ait plus de raison d'être, la

Nous verrons dans le Chapitre 5 quelques méthodes qui permettent de représenter la répartition du dialogue entre l'homme et la machine pour les tâches procédurales.

[35] Il ne faut pas confondre les enchaînements d'actions avec les enchaînements de traitements. Les premiers seraient par exemple saisir puis imprimer des clients alors que les seconds seraient créer la fenêtre de saisie des clients puis afficher cette fenêtre.

tendance actuelle semble plutôt se porter sur le contrôle externe grâce à l'utilisation des UIMS et de la programmation événementielle.

4.3. Modèles d'applications interactives

La littérature actuelle a souvent tendance à mélanger les sens donnés au mot modèle (cf § 4.1). Le problème de la typologie des modèles vient en fait de l'ancienneté de certains d'entre eux. Les plus anciens, comme le modèle de Seeheim, peuvent être vus comme des modèles originels. Ils sont souvent imprécis mais ont l'avantage de poser de bonnes questions sur le plan conceptuel. Ils ont donné naissance à des modèles plus précis et plus orientés vers l'implémentation directe ce que ne permettent pas des modèles tels que celui de Seeheim. Les modèles que nous présentons ici sont de deux types différents (conceptuels et implémentation) ; nous les exposons dans le but de mettre en relief la partie destinée à gérer le dialogue homme-machine.

Dans un souci de concision, nous ne présentons ci-dessous que quelques uns des modèles les plus connus. Nous avons omis pour cela des modèles tels que le modèle à réseau global [BOUSSE 91], le modèle trois modules [HARTSON 90] ou le modèle trois couches [VIGNAUD 89]. Les modèles présentés sont regroupés en trois catégories : les modèles conceptuels, les modèles d'implémentation et les modèles qui sont à l'intersection de ces deux groupes.

4.3.1. Les modèles conceptuels

4.3.1.1. Le modèle de Seeheim

Le modèle de Seeheim[36] [PFAFF 85] est sans conteste possible le premier modèle qui a mis en valeur le contrôleur de dialogue par l'intermédiaire de la séparation d'une application en trois modules. Ce modèle (Figure 2.5) a pour but de distinguer la gestion du dialogue de la sémantique de l'application [EL MRABET 91]. Il est composé de trois modules qui sont greffés sur le noyau applicatif. Chacun de ces modules est distinct des autres et possède des rôles bien déterminés. Ces trois modules sont :

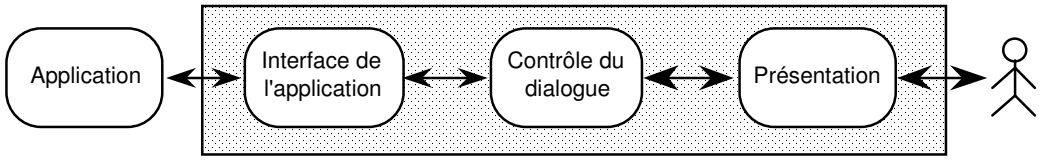


Figure 2.5 : Le modèle de Seeheim

- la **présentation** qui est responsable de la gestion purement lexicale de l'application. Son rôle est de :
    - gérer la présentation et l'affichage des entités manipulées. Elle doit donc répondre aux critères d'ergonomie et de facilité d'apprentissage (utilisabilité) d'une application. Sa qualité influence en partie l'utilité de l'application puisque c'est par elle que transitent tous les résultats.
    - gérer les événements physiques d'entrée/sortie. Elle est chargée de “convertir les événements de bas niveau en événements de haut niveau” [PETOUD 90] ou plus

[36] Le modèle de Seeheim a été utilisé dans de nombreux travaux. Citons par exemple le modèle GWUIMS [SIBERT 86] où les trois composants Présentation, Contrôleur de Dialogue et Interface de l’Application sont réalisés par trois classes d'objets qui sont respectivement les R_Objets (Representation Objects), le I_Objets (Interaction Objects) et les A_Objets (Abstraction Objects).

simplement de “traduire les informations entre le format du monde réel extérieur et celui du monde informatique intérieur” [COUTAZ 88].

La présentation est donc chargée de la gestion de la représentation externe de l'application. Ce rôle bénéficie cependant d'un certain flou ; la gestion des “callbacks” n’est en effet pas clairement abordée dans ce modèle et chaque concepteur doit adapter la définition de la présentation de la meilleure façon.

- le **modèle de l'interface de l'application** ou plus simplement **l'interface de l'application** qui correspond à la vue qu'à le contrôleur de dialogue sur ce que J. Coutaz appelle le noyau fonctionnel [COUTAZ 91c]. Elle a pour rôle :
  - de relier le noyau fonctionnel et le module de contrôle du dialogue en faisant appel aux procédures de l'application [EL MRABET 91].
  - de convertir les informations issues du contrôleur de dialogue en concepts manipulables par l'application [COUTAZ 88]. L'interface de l'application doit également savoir comment sont structurés les services de l’application et comment les utiliser. En d'autres termes, elle doit connaître la sémantique de l'application.

- le **contrôleur de dialogue** qui est le médiateur entre l'utilisateur et l'application, c'est-à-dire entre la présentation et l'interface de l'application. Son rôle est de :
  - contrôler les échanges au niveau syntaxique,
  - gérer la dynamique du dialogue.

  On prévoit en général un scénario qui décrit l'ordre des séquences d'exécution des opérations ainsi que la succession des écrans [PETOUD 90]. Plusieurs formalismes peuvent être utilisés pour représenter ces scénarios : Use [WASSERMAN 82a] [WASSERMAN 82b] [WASSERMAN 85], Diane [BARTHET 88], RPO [SIBERTIN 85]... Le contrôleur est donc “responsable de la coordination entre l'affichage (présentation) et l'état d'avancement des traitements” [PETOUD 90]. De même que pour la présentation, le rôle du Contrôleur de Dialogue reste assez flou. Ainsi les callbacks, qui sont les reflets de l'avancement des interactions, doivent-ils passer par le contrôleur ? D'après ce qui précède, il semble que dans tous les cas cela soit vrai. Malheureusement le temps de réponse s’en trouve allongé et il est souhaitable de considérer deux types de callbacks : ceux qui demande une vérification auprès de l'application (et donc qui transitent par le Contrôleur de Dialogue) et ceux qui n'en demandent pas (et donc qui restent à la charge de la Présentation).

Le modèle de Seeheim définit donc relativement clairement les rôles au niveau conceptuel des trois modules de l’interface d'une application interactive. Malheureusement l'implémentation de ce modèle présente des difficultés puisque les rôles des modules ne sont pas suffisamment explicites. De plus il ne correspond pas dans l'absolu au modèle de programmation objet et événementiel puisque ces derniers prônent une répartition du contrôle entre les objets qui sont manipulés. C'est pourquoi le modèle de Seeheim a donné naissance à plusieurs modèles que nous verrons plus loin ainsi qu'à des versions “améliorées” dont nous donnons ici deux exemples :

- “Seeheim modifié” [ALTY 89] qui ajoute une connexion liant directement l’application avec la présentation,
- “Seeheim étendu” [KARSENTY 91] qui éclate le module interface de l'application en trois modules plus spécifiques (abstractions de l’interface, abstractions du noyau de l’application, contrôleur de l’application).

Le modèle que nous utilisons pour notre travail se rapproche de Seeheim puisqu’on y trouve séparément le Contrôleur de Dialogue, l’Interface de l’application et le Noyau Fonctionnel. Par contre la Présentation est répandue sur ces trois modules.

4.3.1.2. Le modèle de Hudson

Le modèle de Hudson [HUDSON 87] est une alternative au modèle de Seeheim. Il ne comporte pas de contrôleur de dialogue (Figure 2.6) et le composant interface de l'application est remplacé par un ensemble d'objets actifs qui récupèrent une partie des responsabilités de la couche contrôle de dialogue ainsi qu'une partie du dialogue avec les outils de l'application [BOUSSE 91].

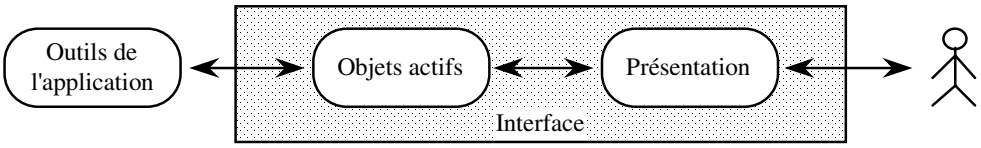


Figure 2.6 : Le modèle de Hudson

La **présentation** se charge de la représentation externe de l'application. Elle est plus puissante que celle de Seeheim puisqu'elle récupère une partie du dialogue. Ainsi un menu se charge de gérer tous ses changements d'états jusqu'à la sélection définitive d'une de ses commandes.

Un **objet actif** est le produit de deux définitions : un type abstrait qui le décrit en tant que structure de données et un ensemble de procédures définissant son comportement [BOUSSE 91].

Le modèle de Hudson a trois avantages majeurs :
- il minimise la syntaxe dans le dialogue grâce au caractère décentralisé des objets actifs,
- le comportement propre des objets actifs permet à de nombreux événements liés à la sémantique d'être gérés à l'intérieur de l'interface ce qui permet un retour sémantique rapide,
- les outils de l'application ne sont pas passifs comme dans le modèle de Seeheim. Ils peuvent émettre ou recevoir des requêtes vis-à-vis de l'interface ce qui donne un contrôle de type mixte au système.

Le modèle de Hudson ne peut cependant pas résoudre tous les problèmes liés aux IHM. Ainsi les objets doivent être dotés de méthodes répercutant leur changement d'état auprès des objets des autres couches (présentation et outils) afin de maintenir la cohérence. De plus il n'y a pas de composant responsable de la communication avec les outils de l'application. Celle-ci est répartie dans les objets actifs et sa modification peut s'avérer difficile.

Notre modèle incorpore des objets qui peuvent être rapprochés des objets actifs puisqu'ils comportent des traitements élémentaires nécessaires à la gestion des données qu'ils contiennent. Ces objets sont également en partie responsable de la gestion du dialogue.

4.3.2. Les modèles d'implémentation

4.3.2.1. Le modèle à architecture distribuée

Ce modèle [CLEMENT 90] est à la fois utilisé comme environnement de programmation et comme structure d'implémentation. Il est basé sur les concepts de graphes et d'objets. Chaque objet est supposé ne pas être fortement couplé avec les autres dans le sens où on n'écrira jamais "x envoie message à y". Par contre chaque objet est relié à d'autres objets selon un graphe. Ainsi un objet est un nœud avec des ports d'entrées et des ports de sorties (Figure 2.7). La communication entre objets se fait par envois d'événements selon le format suivant :

emit <identifiant objet> <identifiant port> <valeur>.

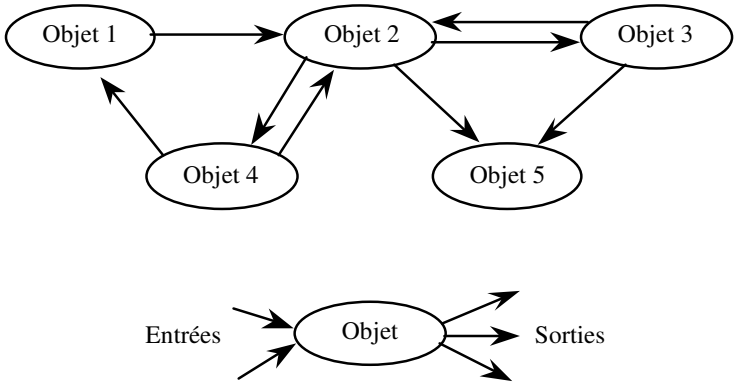


Figure 2.7 : Le modèle à architecture distribuée.
La partie inférieure représente un objet du graphe avec ses ports d'entrée/sortie

La structure du graphe des objets est dynamique ce qui permet de tester plusieurs configurations différentes.

La structure de graphe se retrouve dans la structure de l'application complète. Celle-ci se décompose en trois modules de la même façon que dans le modèle de Seeheim (application, contrôle du dialogue et présentation). D. Clément [CLEMENT 90] ne donne pas d'indication sur le module application et sur le module présentation. On peut cependant imaginer leur donner la même structure que celle du contrôleur de dialogue qui est en fait un graphe d'objets. Le modèle à architecture distribuée peut donc être vu comme un modèle récursif de la même façon que PAC. L'application complète serait alors un graphe, chacun de ses trois composants de base étant également un graphe (Figure 2.8).

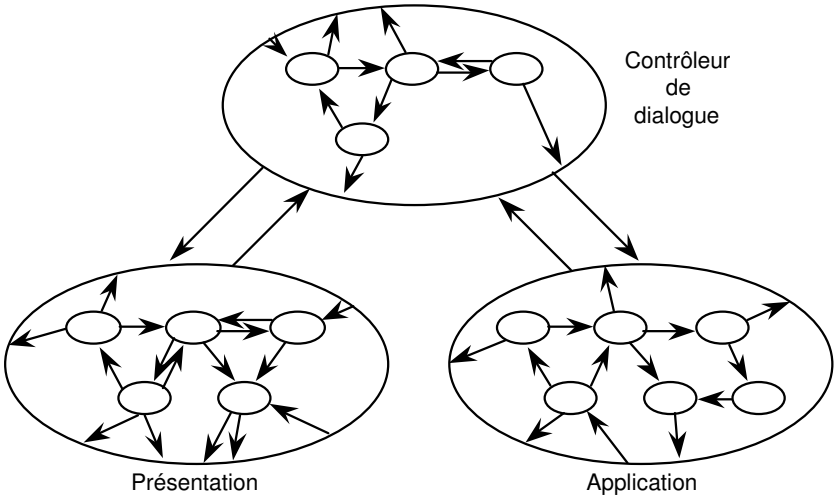


Figure 2.8 : Une application avec architecture distribuée

Outre les caractéristiques déjà énoncées dans le modèle de Seeheim, cette architecture a l'avantage d'être dynamique et se rapproche de cette façon des réseaux neuronaux. On peut d'ailleurs imaginer un réseau de ce type dédié uniquement au dialogue ce qui permettrait une adaptativité complète en fonction par exemple du niveau de l'utilisateur ou de la fréquence d'utilisation.
L'architecture distribuée permet l'élaboration de modules non fortement couplés, mais elle a l'inconvénient d'être difficile à mettre en œuvre à cause de l'effort de décomposition et de gestion des liens qu'elle demande.

4.3.2.2. Le modèle MVC

MVC est le modèle multi-agent utilisé par Smalltalk [GOLDBERG 84]. Le but de ce modèle est d'avoir un système qui soit composé d'un ensemble de triplets autonomes et pouvant communiquer entre eux (Figure 2.9). Il est composé de trois éléments qui sont :

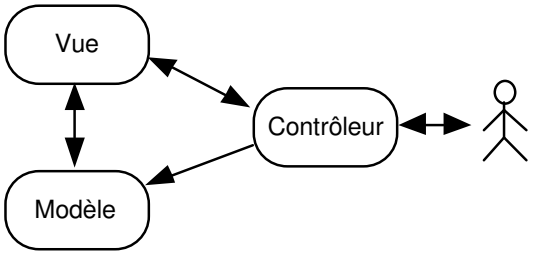


Figure 2.9 : Le modèle MVC

- le **Modèle** qui est la structure de données que l'on veut représenter à l'écran et qui est composée d'objets, plus exactement d'instances de classes Smalltalk.

- la **Vue** qui est la représentation externe du Modèle. C'est par elle que l'utilisateur perçoit le Modèle (inputs) et que le Modèle reflète ses changements (outputs). Grâce à la séparation des données et de leur représentation, un Modèle peut avoir plusieurs vues différentes alors qu'une Vue ne peut être associée qu'à un seul Modèle. L'imbrication des Vues (donc le regroupement des Modèles) reste cependant possible, ce qui permet de développer des prototypes qui soient tous basés sur une même structure de données et de passer d'une représentation à l'autre sans changer le code de l'application sous-jacente.

- le **Contrôleur** qui régule les interactions entre la Vue et le Modèle. Il est chargé de gérer les actions de l'utilisateur sur la Vue. Lorsque l'utilisateur manipule la Vue, le Contrôleur informe le Modèle des interactions faites. Ce dernier modifie alors son état et informe la Vue du nouvel aspect qu'elle doit prendre suite à cette modification.

Le modèle MVC permet a priori de travailler de manière modulaire impliquant ainsi une relative clarté et réutilisabilité dans le code de l'application. Malheureusement son implémentation dans Smalltalk révèle de nombreux problèmes. Les événements que manipule Smalltalk sont de bas niveau et les Contrôleurs contiennent des routines qui ne sont sensibles qu'à des événements tels que le clic souris ou l'enfoncement d'une touche. Pour cela ils doivent être constamment à l'écoute des entrées utilisateur et donc ils utilisent une grande partie du temps CPU. La technique utilisée par Smalltalk (polling) pour gérer les interactions utilisateur limite les capacités du modèle. Chaque contrôleur garde en effet la main tant que l'utilisateur ne provoque pas d'action sur un autre contrôleur. Les problèmes deviennent encore plus ardus si on désire travailler en multi-tâche auquel cas les entrées multiples sont impossibles à réaliser.
Un autre problème est l'enchevêtrement du code. Les boucles qui scrutent les entrées utilisateur sont souvent imbriquées et il est difficile de retrouver le fil conducteur du contrôle. Ainsi pour ajouter une nouvelle vue, on doit construire un nouveau contrôleur et donc ajouter encore des boucles de scrutation que l'on doit relier à celles existantes.

Le modèle MVC a été un précurseur dans le monde des interfaces bien qu'il comporte quelques inconvénients. Smalltalk/V a résolu ces problèmes dans ses dernières versions. Ainsi une fenêtre est par défaut son propre modèle et son contrôleur y est directement intégré. L'inconvénient de ce choix est la rigidité des comportements des fenêtres bien qu'on puisse toutefois les redéfinir, mais

l'avantage majeur est la capacité à traiter des événements haut niveau. Toutes les fenêtres sont en effet gérées par un seul contrôleur. Lorsqu'un événement survient dans une fenêtre, le contrôleur l'intercepte, le décode et l'envoie automatiquement à la fenêtre émettrice sous la forme d'un message de haut niveau, par exemple "ButtonLeftDoubleClic" ou encore "OnePageScroll". La gestion des événements est ainsi beaucoup plus claire et agréable et elle peut facilement être étendue.
Une autre solution est apportée par Y-P Shan [SHAN 89] qui a ajouté un contrôle dirigé événement au fonctionnement de base de Smalltalk. Par ailleurs son architecture permet de faire cohabiter les deux systèmes sans modification du code MVC.

L'implémentation de nos travaux étant écrite en Smalltalk, elle repose donc en partie sur le modèle MVC.

4.3.2.3. Le modèle multi-couche

J-D Fekete [FEKETE 91] utilise un modèle multi-couche pour la gestion des interactions dans le cadre d'applications graphiques telles que MacDraw. Nous le présentons dans l'objectif d'une gestion de dialogue tel que nous l'avons vu jusqu'à présent. Ce modèle est représenté sur la figure 2.10.

Dans le cadre d'une application graphique, ce modèle fonctionne de la façon suivante : lorsqu'un événement utilisateur survient, il est intercepté par la première couche (ici la grille), puis il transite de couche en couche tant que la couche réceptrice n'est pas apte à le traiter. Un événement peut donc passer à travers toutes les couches avant d'être traité. Sur la figure 2.10 nous avons trois couches : une pour la grille, une pour le dessin et une pour la sélection. Le nombre de couches n'étant a priori pas limité, on pourrait par exemple construire une couche qui traiterait tous les événements non reconnus par les couches supérieures au lieu de laisser cette tâche à la couche de sélection.

Cette décomposition en couches est intéressante pour les applications graphiques puisqu'elle permet de répartir les contrôles sur les interactions en fonction de leur type (dessin d'une forme géométrique, déplacement d'objets, remplissage de formes...). Cependant on peut imaginer cette même décomposition pour la gestion du dialogue dans une application interactive[37]. Ainsi on aurait :

- une couche pour la communication avec l'extérieur. Elle ne serait là que pour établir le lien entre le hardware et l'application. Elle s'occuperait par exemple d'intercepter les clics souris ou la frappe d'une touche ou bien encore elle pourrait enregistrer la voix de l'utilisateur.

- une couche qui transformerait les données issues de la couche précédente en événements logiques de bas niveau. Le clic souris détecté par la couche supérieure (par exemple un signal sur le port série) serait transformé en un événement tel que "Clic en (x,y)".

- une couche qui transformerait les événements bas niveau précédents en événements haut niveau. Le clic souris précédent pourrait par exemple devenir "Scrolling d'une page vers le haut".

- trois couches successives, une pour chaque analyse (syntaxique, lexicale et sémantique) des événements précédents.

[37] Cette décomposition n'est qu'un exemple que nous proposons. Il est tout à fait possible de l'affiner ou de le modifier.

- une couche qui serait responsable du déclenchement des services offerts par l'application. Elle aurait pour rôle d'activer les déclenchements et renvoyer les résultats des exécutions ou bien encore traiter les événements invalides qui auraient pu passer.

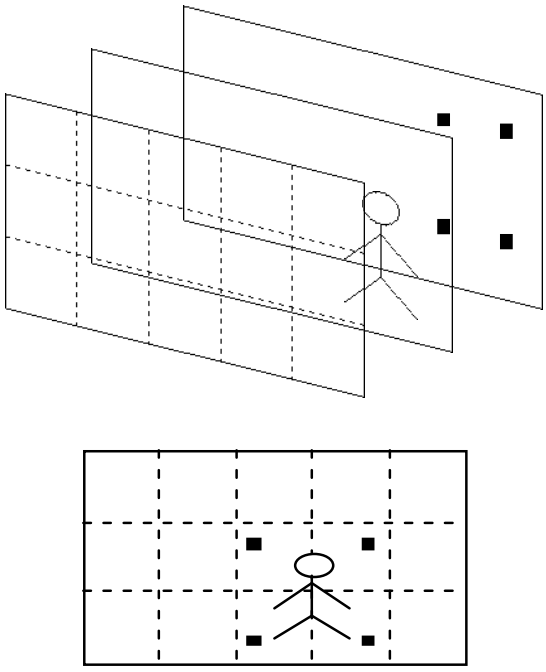


Figure 2.10 : Le modèle multi-couche
La partie supérieure montre la décomposition en couches,
la partie inférieure montre le résultat à l'écran.

Cette décomposition se rapproche du modèle présenté par [KUO 85a] qui présente quatre couches. Dans ce modèle, la première couche est fortement dépendante du hardware et peu dépendante du système auquel elle est rattachée. En fait elle est composée d'unités de telle sorte qu'à chaque type de matériel utilisé, il corresponde une de ces unités. Les couches suivantes sont des transformations similaires à celles citées plus haut, mais plus orientées vers la sémantique et la syntaxe des interactions. La dernière couche est l'analyseur de dialogue qui est le dernier à permettre ou interdire le déclenchement des services demandés.

La décomposition en couches avec passage de l'une à l'autre n'est pas sans rappeler le concept de classes et de messages. On peut facilement réaliser ce modèle en couches en implémentant chacune d'elle par une classe. L'ensemble des couches serait par exemple un ensemble de classes imbriquées. Dans ce cas une interaction (c'est-à-dire un message) pourrait facilement transiter d'une couche à l'autre grâce à la notion d'héritage des langages objet.

Le modèle que nous avons utilisé dans notre travail est également multi-couche. La gestion du dialogue est répartie dans les six couches qui le composent. Ce modèle sera exposé en détail au Chapitre 6.

4.3.3. Les modèles hybrides

4.3.3.1. Le modèle Arch

Le modèle Arch [BASS 91] [ARCH 92] est une extension du modèle de Seeheim puisqu'il éclate en deux les modules extrémités, c'est-à-dire la présentation et l'interface de l'application. On peut le qualifier de modèle d'implémentation puisqu'il repose sur une toolkit et qu'il définit clairement les rôles de chacun de ses composants ainsi que les liens qui les unissent, rendant ainsi aisée une implémentation directe.

Le **composant spécifique au domaine** implémente les fonctionnalités du domaine indépendamment des interactions avec l'utilisateur. Il est donc une restriction du noyau applicatif du modèle de Seeheim car, dans ce dernier, rien n'empêche une primitive de faire appel à une interaction.

Le **composant d'interaction avec la toolkit** implémente les interactions physiques avec l'utilisateur. Il est basé sur un ensemble fixe mais extensible de classes d'objets d'interaction. Cette collection de classes dépend de la capacité d'évolution de la toolkit sur laquelle repose le composant. Ce dernier peut d'ailleurs être vu comme une restriction de la présentation du modèle de Seeheim puisqu'il s'interdit le changement du niveau d'abstraction des événements et des données manipulées.

Le **composant adaptateur au domaine** a pour rôle :
- d'établir le lien entre le contrôleur de dialogue et le composant spécifique au domaine,
- d'implémenter les tâches relatives au domaine et qui sont nécessaires aux opérations de l'utilisateur, mais qui ne sont pas disponibles dans le composant spécifique du domaine,
- de réorganiser les données ; il collecte par exemple des données pour les afficher dans une liste,
- de détecter et de reporter les erreurs sémantiques.

Le **composant de présentation** est un médiateur entre le contrôleur de dialogue et le composant d'interaction avec le toolkit. Il est responsable de la fourniture d'objets indépendants du toolkit pour leur utilisation par le contrôleur de dialogue.

Le **contrôleur de dialogue** de l'application est la clef de voûte de ce modèle. Son rôle est triple :
- il est responsable du séquencement des tâches,
- il établit un lien entre les formalismes spécifiques au domaine et ceux spécifiques à l'interface,
- il maintient la cohérence des vues multiples que l'on peut avoir sur un même élément.

Le contrôleur de l'application a donc un rôle plus complet que celui du modèle de Seeheim.

La caractéristique commune de chacun de ces cinq composants est de pouvoir absorber au mieux les changements de ses voisins. Cette caractéristique se ressent surtout au niveau des deux médiateurs, le composant de présentation et celui de l'adaptation au domaine. Pour communiquer, ces cinq modules échangent obligatoirement des données. Conceptuellement des objets permettent de faire le lien entre les modules [BASS 91] ; les flèches de la figure 2.11 signifient que ces objets peuvent être manipulés par l'un ou l'autre de ses voisins proches en entrée ou en sortie. Ces objets sont les suivants :
- *les opérations du domaine* qui utilisent des structures de données propres à ce domaine.
- *les objets du domaine* qui sont en fait des abstractions du domaine. Ils sont créés à partir des structures et des opérations du domaine par le composant adaptateur du domaine.

- *les objets de présentation* qui sont des objets virtuels qui contrôlent les interactions de façon indépendante du média utilisé. Ils contiennent les données à présenter et les événements à générer.
- *les objets d'interaction* qui sont conçus spécialement pour un média. A un objet de présentation peut correspondre plusieurs objets d'interaction alors qu'à un objet d'interaction ne correspond qu'un seul objet de présentation.

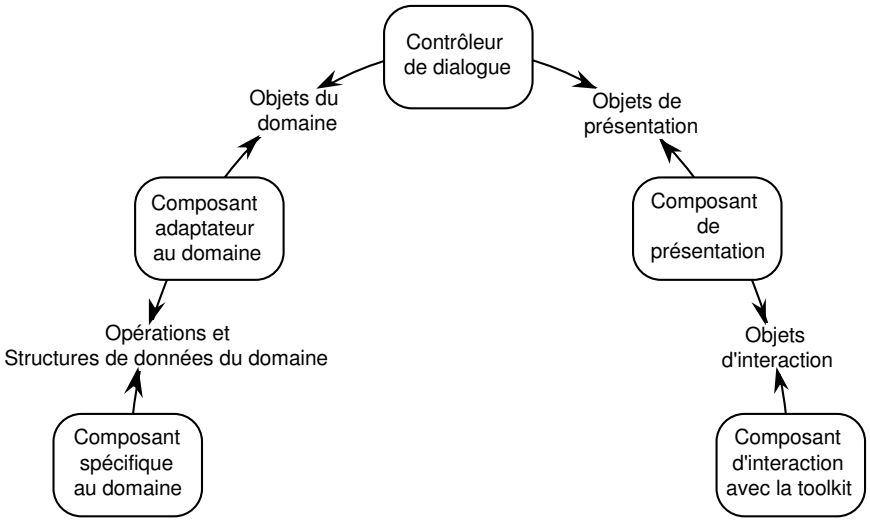


Figure 2.11 : Le modèle Arch

4.3.3.2. Le modèle Multi-agent

Le modèle de Seeheim a l'inconvénient majeur d'être un système centralisé. Le modèle multi-agent est quant à lui un système réparti en fonction des agents qui le composent. Son fonctionnement est basé sur le principe des stimuli-réponses (les agents sont réactifs) et les communications entre agents s'établissent au moyen d'événements.

Un **agent** est composé :
- de *récepteurs* qui reçoivent les *événements* issus des autres agents. Les récepteurs peuvent être accompagnés de filtres afin de n'être sensibles qu'à certains types d'événements,
- d'*émetteurs* qui transmettent des événements typés au système et non pas à un agent particulier,
- de *mémoires* à deux niveaux : un niveau pour enregistrer les événements détectés et un niveau pour mémoriser l'état de l'agent,
- d'un *processeur cyclique* qui traite les événements un par un.

Le fonctionnement général d'un système multi-agent est le suivant : un événement est détecté et capté par tous les agents dont les récepteurs actifs à cet instant sont capables de l'intercepter. Cet événement est ajouté à la file d'attente de chaque agent qui l'a détecté. Quand vient son tour, le traitement de l'événement provoque un changement d'état de l'agent qui peut alors à son tour émettre de nouveaux événements. Grâce à ce type de fonctionnement, il est possible de travailler en dialogue multi-fil en associant à chaque fil un agent spécifique (ou une grappe d'agents) à une partie du dialogue. On dispose ainsi d'un système, supportant le parallélisme, à contrôle réparti et non plus centralisé dans un seul composant.

Les caractéristiques des systèmes multi-agent permettent une bonne implémentation en langage objet [BERRADA 92]. A un type d'agent correspond une classe dont les attributs représentent les mémoires des agents ; les opérations de traitement des événements sont stockées sous la forme de méthodes, et la communication par événements est remplacée par l'envoi de messages. La faculté d'instanciation et d'héritage des langages objet permet de créer dynamiquement des agents, donc d'avoir un système évolutif, ainsi que de spécialiser des agents par rapport à d'autres agents. Cette similitude entre langage objet et multi-agent provient des propriétés de modularité et de coopération des langages objet. Un objet est considéré en effet comme une entité autonome tout comme l'est un agent. Par ailleurs la communication par messages (ou par événements pour les agents) permet de faire dialoguer entre elles des entités qui ne se connaissent pas et sans contact direct, permettant ainsi des échanges de formats différents.

Le concept d'agent a été utilisé dans plusieurs modèles, par exemple MVC et PAC ainsi que dans des UIMS tels que Serpent [COUTAZ 91d] et GWUIMS [SIBERT 86].

Le concept d'agent est également utilisé dans d'autres domaines que celui des interfaces, c'est le cas par exemple de l'intelligence artificielle dans le cadre de l'intelligence distribuée ou de la planification [PORTEJOIE 91a] [PORTEJOIE 91b].

4.3.3.3. Le modèle PAC

PAC [COUTAZ 88] est un modèle multi-agent dont chaque agent est constitué de trois parties (Figure 2.12) :

- la **Présentation** qui est l'image que perçoit l'utilisateur de l'agent. Elle correspond à la Représentation Externe de l'agent,
- l'**Abstraction** qui est l'ensemble des concepts et des fonctions de l'agent. Elle définit donc les fonctionnalités sémantiques de l'agent,
- le **Contrôle** qui maintient la cohérence entre la Présentation et l'Abstraction. Il établit un lien entre ces deux parties tout en jouant un rôle d'arbitre pour résoudre les conflits, les synchronisations, les rafraîchissements, etc., ainsi qu'un rôle de traducteur puisque les trois composants s'échangent des informations qui peuvent avoir des formats différents.

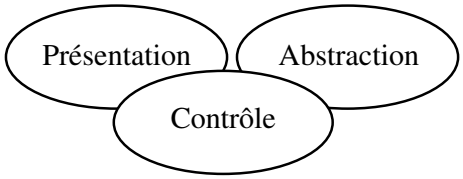


Figure 2.12 : Un agent PAC

Le travail de R. Hill [HILL 92] est également très proche de cette décomposition. R. Hill décompose les données en trois parties qui sont l'Abstraction, la Vue (équivalent à la Présentation de PAC) et les Liens (équivalent aux Contrôles de PAC) reliant une Abstraction à une Vue. Ce modèle (appelé ALV) a été conçu principalement pour les applications multi-utilisateurs, l'Abstraction regroupant les données communes à tous les utilisateurs et pouvant être reliée à plusieurs Vues. Par contre, bien que d'apparence similaire, PAC et MVC diffèrent en plusieurs points. On trouvera une comparaison de ces deux modules dans [COUTAZ 91a].

Une application basée sur le modèle PAC est constituée d'un ensemble d'agents PAC structurés hiérarchiquement et de manière récursive (Figure 2.13). Un agent PAC peut ainsi être décomposé

en un ensemble d'agents PAC. Dans la figure 2.13, l'objet le plus haut dans l'arborescence correspond aux trois parties principales de l'application, les sous-niveaux étant des objets intermédiaires[38] et les feuilles étant les objets directement manipulables les plus élémentaires que l'on puisse avoir. Il faut remarquer qu'un objet père hérite des Présentations de ses fils et non l'inverse.

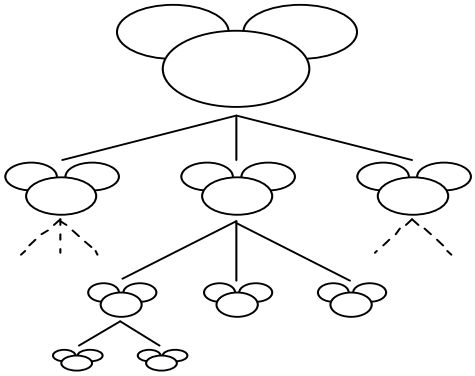


Figure 2.13 : Un agent PAC complexe

La Présentation d'un agent PAC regroupe un ensemble d'agents spécialisés dans l'interaction avec l'utilisateur, les *objets interactifs* [COUTAZ 88]. Les objets interactifs sont eux-mêmes des agents PAC et respectent donc la décomposition en trois parties. Un objet interactif simple est composé d'une image qui définit le comportement visible par l'utilisateur (Présentation) des fonctions qu'il utilise (Abstraction), et d'un module pour gérer la connexion entre ces deux modules (Contrôle). Un objet interactif composé définit un agent structuré dont le comportement dépend de lui-même et de ses constituants. Sa Présentation hérite des propriétés des Présentations de ses constituants et y ajoute ses propres caractéristiques. Son Abstraction lui est propre et le Contrôle vérifie la cohérence entre Abstraction et Présentation tout en gérant la collaboration entre ses composants. Il faut noter que dans tous les cas les trois composants ne sont pas nécessaires. Seul le contrôle est primordial pour garder la cohérence.

Nous verrons dans le Chapitre 5 que nous utilisons un dérivé du modèle PAC pour représenter les données.

Le modèle PAC-Amodeus

L. Nigay [NIGAY 91] [COUTAZ 91b] propose un modèle hybride de PAC et Arch (Figure 2.14). Le modèle PAC-Amodeus adopte les mêmes composants que le modèle Arch et leur assigne les mêmes rôles. Nous trouvons cinq composants : le Contrôleur de Dialogue, le Noyau Fonctionnel, l'Adaptateur du Noyau Fonctionnel, la Présentation et le Composant d'Interaction. Seul le Contrôleur de Dialogue diffère par rapport au modèle Arch puisqu'il est composé d'un ensemble d'agents PAC. Le Contrôleur de Dialogue manipule des tâches décomposables en sous-tâches et sa structure multi-agent PAC permet le dialogue multi-fil (une tâche = un fil, et un fil peut être géré par un ou plusieurs objets PAC).

[38] J. Coutaz [COUTAZ 91] remarque que PAC ne définit malheureusement pas clairement le rôle de ces agents intermédiaires. J. Coutaz utilise dans [COUTAZ 90] ces objets intermédiaires par exemple pour regrouper des éléments de natures et de comportements différents.

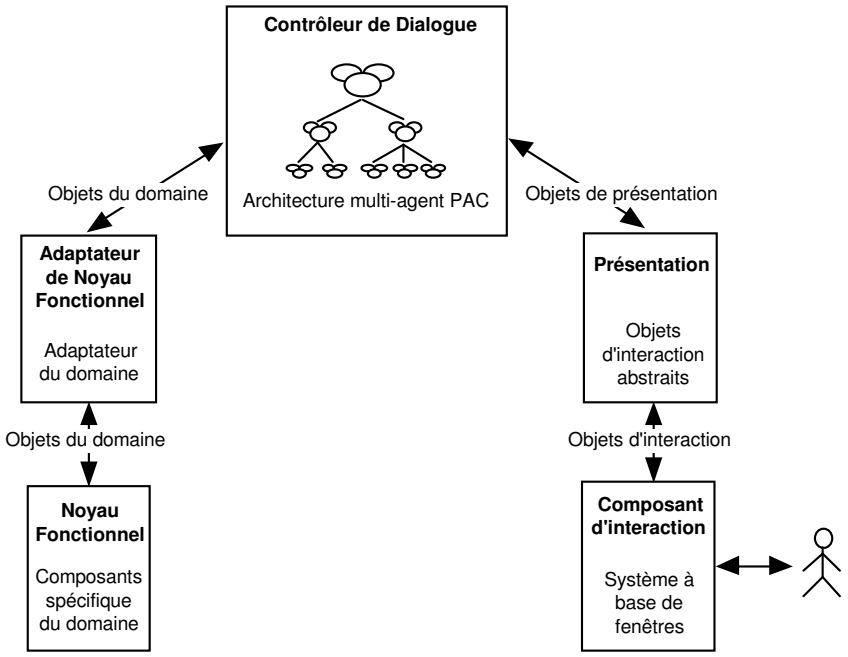


Figure 2.14 : Le modèle PAC-Amodeus

4.3.4. Conclusion sur les modèles

Les modèles que nous venons de voir diffèrent en bien des points. Certains sont orientés vers une implémentation directe, d'autres vers la spécification d'un système dans ses grandes lignes. Ils ont cependant comme point commun la faculté de schématiser le comportement du système. L'intérêt des modèles est avant tout de permettre une appréhension du système que l'on doit modéliser. Le modèle de Seeheim répond tout à fait à ce critère puisqu'il montre bien qu'il faut séparer l'application de l'interface et que la jonction se fait par l'intermédiaire du contrôleur de dialogue. Le modèle PAC met en relief une structure hiérarchique d'éléments semblables. Cependant ces modèles (que l'on peut qualifier de génériques puisqu'ils donnent naissance à des modèles plus précis) ne permettent pas d'évaluer les applications qui les utilisent. Il est nécessaire pour cela de travailler sur les modèles d'architecture (c'est-à-dire les modèles instances issus des modèles génériques) tel que le modèle Arch.

5. Les formalismes de la dynamique

Les modèles (conceptuels ou d'implémentation) d'application permettent de définir la structure générale d'une application (c'est-à-dire le QUOI) sans décrire précisément comment elle va fonctionner. Ce sont les formalismes qui permettent de décrire le comportement de l'application ou parfois uniquement celui de l'interface. Nous allons voir dans ce paragraphe quelques formalismes (connus ou moins connus) en nous basant sur trois axes de réflexion : les formalismes basés sur la gestion d'états, les formalismes basés sur la gestion d'événements et les formalismes hybrides, c'est-à-dire basés à la fois sur les états et les événements. Les formalismes que nous allons voir sont de différents types. Ils sont orientés tantôt conception tantôt implémentation, mais dans tous les cas ils permettent de représenter la dynamique du dialogue homme-machine. Nous avons choisi de les regrouper en fonction de leur orientation principale car la plupart d'entre eux utilisent à la fois les états et les événements.

5.1. Les formalismes à états

Une application peut être vue comme un ensemble d'états qu'elle est susceptible de prendre au cours des sessions de travail. Ces états sont en fait des étapes intermédiaires par lesquelles l'utilisateur est obligé de passer pour atteindre son but. De manière générale les formalismes basés sur les états permettent de décrire les états et la façon de passer d'un état à un autre. Ce passage peut être décrit par un langage, une grammaire ou bien encore un graphe. Dans tous les cas, l'énumération des états et des passages entre états doit être exhaustive pour que le système soit validé. L'avantage principal des formalismes à états est leur capacité à être validé formellement. On peut vérifier par exemple qu'un automate est vivant ou qu'il ne possède pas de puits. Nous allons voir dans ce qui suit quelques uns des formalismes de ce type les plus utilisés dans les interfaces.

5.1.1. Les états finis

Les états finis constituent un des formalismes les mieux maîtrisés pour les IHM étant donné leur capacité à être traduits sous forme de grammaires (équivalence automate/grammaire). Un système basé sur ce formalisme (par exemple Rapid/Use [WASSERMAN 85] [WASSERMAN 86] ou les Statecharts qui autorisent également le parallélisme [HAREL 88] [HAREL 90] [VAN ZIJL 91]) manipule cinq éléments : un ensemble fini d'états, un ensemble fini d'entrées, un ensemble fini de sorties, une fonction qui permet de passer à l'état suivant en fonction de l'état précédent et de l'entrée, et une fonction de sortie qui renvoie le résultat en fonction de ces mêmes paramètres.

Les états correspondent à l'ensemble des valeurs (au sens large) que peut prendre le système. Ces états peuvent être des états graphiques, textuels ou autres.
Les entrées énumèrent toutes les perturbations qui surviennent et qui peuvent faire évoluer un état. Elles peuvent provenir des interactions de l'utilisateur, de capteurs ou de résultats calculés. L'ensemble des entrées est donc équivalent à l'ensemble des événements que le système est capable de gérer.
Les sorties correspondent aux résultats que l'on obtient suite aux entrées. Ces sorties peuvent par exemple donner la position de la souris ou bien renvoyer la position d'un enregistrement dans un fichier.
Pour représenter plus clairement les deux fonctions citées plus haut, on représente la plupart du temps les états finis avec leurs relations sous forme de diagrammes (Figure 2.15). Il existe toujours un état initial (représenté sur la figure par l'état 0 doublement cerclé), mais il peut exister plusieurs états finals (ici le 1 et le 4). On remarquera également qu'un état peut s'appeler lui-même une ou plusieurs fois en fonction de certaines entrées.

Les diagrammes d'états finis (encore appelés diagramme de transitions) sont simples et parlants pour de petits problèmes. Malheureusement leur complexité croît rapidement pour des problèmes complexes [GREEN 86]. On a alors recours à des diagrammes hiérarchisés, les RTN (Recursive Transition Network), qui permettent de remplacer un diagramme par un état dans un diagramme de niveau supérieur.
De manière similaire, les fonctions qui manipulent les entrées et les sorties dans les diagrammes de transition sont trop simplistes. En ajoutant sur les arcs des variables qui peuvent être testées, on obtient des ATN (Augmented Transition Network) qui permettent de gérer des entrées plus complexes que les diagrammes classiques. D'une manière générale, les variables (également appelées registres) ne sont utilisées que pour le dialogue et ne sont donc pas accessibles pour l'application.

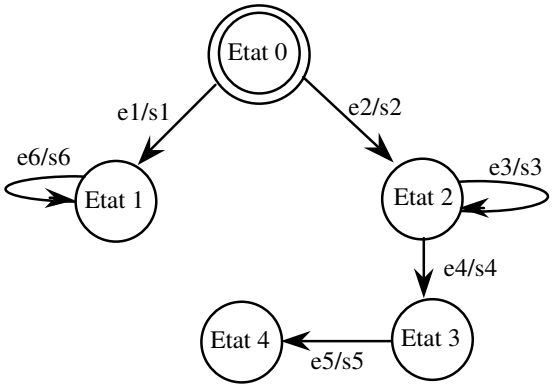


Figure 2.15 : Un exemple de diagramme d'états finis.
Les e_i sont les entrées et les s_i sont les sorties.

Les diagrammes à états finis présentent de nombreux inconvénients pour les IHM. Tout d'abord ils ne permettent pas le parallélisme[39] et la synchronisation ce qui empêche toute gestion asynchrone et donc le dialogue multi-fil. On peut cependant approcher ce dernier en prévoyant tous les cas possibles d'enchaînements ce qui implique un nombre énorme d'états et de transitions. De manière plus générale, dès que l'on veut schématiser des traitements complexes, le diagramme devient vite illisible. On peut alors faire appel aux RTN, mais dans ce cas la vision d'ensemble est réduite.
Au niveau du formalisme, les diagrammes à états finis mettent en relief les états que peut prendre un système. Cependant pour lister exhaustivement ces états ainsi que les changements d'états, on est obligé de parcourir l'ensemble des arcs. Dans le cas de diagrammes complexes, la tâche devient vite fastidieuse.
Alors que l'on recherche dans les IHM une grande souplesse dans le choix des actions, les diagrammes de transitions imposent un dialogue figé (l'ordre des actions est figé). Si l'on veut éliminer cet inconvénient, on doit prévoir comme pour le dialogue multi-fil tous les enchaînements possibles avec les conséquences que cela impose. De plus la gestion des erreurs n'est pas prise en compte à moins de prévoir d'une part tous les cas d'erreurs possibles (et surtout ne pas en oublier !) et d'autre part prévoir toutes les séquences d'actions à suivre lorsqu'on rencontre une erreur ce qui implique, là encore, un accroissement considérable du nombre des états.

5.1.2. Les grammaires

Les grammaires sont utilisées depuis longtemps pour décrire le dialogue homme-machine. L'avantage des grammaires est d'être un formalisme concis et rigoureux. Par contre elles sont parfois difficiles à lire ou à utiliser surtout pour le déboggage. Par ailleurs elles ne permettent pas, en général, de coupler les événements en entrée avec ceux en sortie. On rencontre énormément de types de grammaires, parfois même des grammaires ad hoc. Certaines ne sont utilisées que pour formaliser le dialogue alors que d'autres servent également à l'implémenter ou à l'évaluer [REISNER 81].

Un autre avantage des grammaires est de fournir un niveau d'abstraction élevé. En outre elles permettent la génération de code [YAP 88] ce qui permet de passer presque directement du conceptuel à la représentation interne. On a reproché aux grammaires leur inefficacité vis-à-vis du parallélisme et de la concurrence. S-K Yap [YAP 88] démontre le contraire en utilisant une

[39] R. J. K. Jacob [JACOB 86] est à notre connaissance le seul qui ait étendu ce formalisme en permettant le parallélisme entre les diagrammes.

grammaire dans laquelle il introduit des opérateurs de concurrence qui lui permettent de gérer des dialogues multi-fil. De la même façon, Penguin [YAP 90] est un langage basé sur une grammaire context-free améliorée permettant de traiter la concurrence, et qui donne au langage une nature "réactive". Bien que S-K Yap [YAP 88] affirme que les grammaires context-free sont inadéquates pour le dialogue homme-machine, M. Green [GREEN 86] démontre le contraire en associant à ces grammaires des actions à exécuter par la machine. M. Green [GREEN 86] démontre par ailleurs qu'il est possible de passer des ATN et des grammaires context-free aux événements.

Les grammaires peuvent être également utilisées pour décrire des langages de programmation ou de commandes. C'est le cas de CLG [MORAN 81] qui permet de représenter un système sur plusieurs niveaux d'abstraction : tâche, sémantique, syntaxique et interaction. Chaque niveau est une description complète du système et constitue une vue plus fine du niveau précédent.

5.1.3. Les graphes d'enchaînements

Ce type de graphe est utilisé par certains types d'applications particuliers. I. Petoud [PETOUD 90] l'utilise par exemple pour représenter l'évolution du dialogue au cours d'une session. Dans ce cas, le graphe d'enchaînement représente toutes les opérations que propose le système (chaque opération est supposée déclenchable par l'utilisateur) ainsi que les liens de précédence entre les opérations[40]. Lorsque l'une d'elles est achevée, elle est marquée dans le graphe. L'utilisateur sait donc à tout moment quelles sont les opérations qu'il a déjà effectuées. De plus l'annulation d'une opération provoque l'annulation de toutes celles qui en découlent. Le graphe d'enchaînement est par ailleurs accompagné d'un tableau de bord qui renseigne l'utilisateur sur l'état de l'opération en cours.

Le graphe d'enchaînement est donc une représentation à part entière bien qu'on puisse le rapprocher des Réseaux de Petri en associant une opération à une place (une place marquée correspond à une opération exécutée). On peut lui reprocher cependant d'être trop lourd pour des applications complexes (de même que pour les graphes en général) et d'être difficilement utilisable par des non-informaticiens. Par ailleurs chaque opération devant être déclenchée par l'utilisateur, la latitude décisionnelle est maximale et sans possibilité de restriction, par exemple pour automatiser des enchaînements. Son utilisation s'avère cependant intéressante pour des tests lors de la spécification du dialogue. Le concepteur peut ainsi vérifier la viabilité du système, la possibilité d'atteindre toutes les opérations, etc. Le formalisme Diane+, que nous verrons au Chapitre 5, pourrait être appliqué à ce principe.

5.1.4. Les contraintes

Le principe des contraintes permet de spécifier des caractéristiques sans se soucier de la manière dont elles sont réalisées. Plus précisément la programmation par contraintes fixe des buts à atteindre sans qu'on ait besoin de coder les traitements pour y parvenir [BOUSSE 91]. En cela le principe des contraintes possède un caractère déclaratif au même titre que les règles d'inférence.

Le fonctionnement classique des systèmes qui utilisent ce principe est le suivant : une contrainte est une équation à n variables qui doit être vraie à tout moment. Si la valeur de l'une des variables vient à changer, le résolveur de contraintes rétablit l'équation en modifiant une ou plusieurs variables de l'équation. L'environnement de programmation ThingLab [BORNING 79] [BORNING 86a] [BORNING 86b] respecte ce fonctionnement.

[40] Un dérivé de ce type de graphe est utilisé par Aïda/Masaï qui représente sous forme de graphe l'ensemble des fenêtres que manipule le système en y associant les conditions d'enchaînement (clic souris, touche enfoncée...).

D'autres systèmes sont également basés sur les contraintes sans qu'il soit nécessaire pour autant d'écrire des équations à n variables. Péridot [MYERS 88b] par exemple infère des contraintes graphiques qui seront utilisées par la suite pour la présentation et la gestion de l'interface. Le système Constraint [VANDER 88] utilise quant à lui une grammaire à contraintes graphiques et syntaxiques pour gérer la cohérence entre la représentation interne et externe de l'interface. Dans le même ordre d'idée, on trouve Coniac [DOMENJOZ 90] qui fonctionne avec des règles de syntaxe et de sémantique qui sont compilées et liées à l'application. Ces règles permettent de dire ce qui doit être entré (c'est-à-dire au minimum), ce qui doit être affiché (idem) et ce qui peut être entré (c'est-à-dire l'éventail des possibilités). On est donc également en présence de contraintes même si celles-ci sont formulées sous forme de règles[41]. Les boîtes actives [ADREIT 89a] [ADREIT 89b] utilisent des contraintes de présentation (par exemple sur l'alignement) et d'interaction ainsi que des démons pour définir leur comportement. Les démons sont également utilisés par Garnet [MYERS 89b] pour implémenter les contraintes (les démons se déclenchent automatiquement lorsque les contraintes associées doivent être résolues).

Les contraintes peuvent donc être de deux types : explicites (ThingLab, Constraint ou boîtes actives) ou implicites (Péridot). Les UIMS utilisent couramment les dernières, principalement à cause de l'augmentation importante de la part de manipulation directe dans la construction des écrans. Lorsqu'on bâtit une interface, on peut en effet facilement utiliser des contraintes d'alignement ou de disposition par rapport à d'autres éléments ou bien encore des règles pour les changements de taille. Une fenêtre qui contient des boutons et des listes pourra par exemple obliger les listes à subir ses modifications de taille alors que les boutons devront rester toujours de la même taille.

Les contraintes sont intéressantes pour le dialogue homme-machine de même que pour l'ergonomie visuelle des interfaces. Il semble cependant qu'il soit difficile de n'utiliser que cette technique pour spécifier et implémenter des applications interactives complexes. Par contre, il est tout à fait possible de l'utiliser conjointement avec d'autres techniques (c'est le cas avec les UIMS). On pourrait les utiliser par exemple pendant la conception pour décrire quelles sont les opérations interdites à un instant donné. Cette caractéristique est intéressante pour les tâches créatives qui laissent en général une grande liberté de choix pour les opérations. Dire que toutes les opérations sont accessibles sauf celles de caractère dangereux[42] par exemple serait plus rapide à écrire qu'à décrire par l'intermédiaire d'un graphe.

5.1.5. Les langages visuels

Un langage visuel est un langage dont l'écriture passe nécessairement par une représentation graphique. Ce sont des langages qui permettent la programmation iconique (appelée parfois programmation "visuelle"). De plus en plus utilisés actuellement, ils servent à construire des interfaces et à définir leur comportement (Mode [SHAN 90], Péridot [MYERS 88b]). Il est possible de simuler le comportement d'une interface par l'intermédiaire de tel langage [CHATTY 91], ou encore de le construire dynamiquement grâce à la programmation par la démonstration [MIYASHITA 92] [BOS 92] [FISHER 92] [HASHIMOTO 92] [KURLANDER 92]. Les langages visuels peuvent également servir à utiliser une interface sans connaître un langage de commande. Citons par exemple ConMan [HAEBERLI 88] (Figure 2.16)

[41] Citons par exemple la saisie d'un en-tête qui n'autoriserait que Monsieur, Madame ou Mademoiselle. La règle s'écrirait : en-tête --> "Madame | Mademoiselle | Monsieur".

[42] Associer un caractère (ou une nature) aux opérations est une solution élégante pour les classer. Cela permet par exemple de déclencher automatiquement des contrôles lors de leur activation ou bien encore de générer des menus en fonction de ces natures [NORMAND 92a].

qui permet la manipulation de dessins en trois dimensions uniquement par l'intermédiaire de canaux graphiques et de divers curseurs (rotation, intensité, etc.). Sur cette figure, l'utilisateur a commencé par dessiner la demi-coupe verticale d'un verre (fenêtre curved). Puis à l'aide de la commande obj out (objet out), il envoie cette coupe comme entrée d'une zone de dessin (sweep). Il applique ensuite sur l'objet en entrée de cette zone une rotation selon l'axe vertical (fenêtre viewed de droite) dont il envoie la valeur par le canal trans xfm, ainsi qu'un décalage (fenêtre viewed de gauche) par le canal view xfm.

La figure 2.17 représente, quant à elle, un programme écrit avec un langage imagé appelé Mandala [MANDALA 84]. Ce langage a été mis au point par J. Lanier à l'institut de recherche VPL à Palo Alto. L'utilisateur inscrit ses instructions en déplaçant les symboles graphiques sur l'écran et en déclenchant leur mouvement. La figure 2.17 se lit de la façon suivante :

> En haut un kangourou saute d'une triple clef, ce qui déclenche un canon à trois voix. Il atterrit sur une portée qui correspond à un affichage musical des notes, puis sur un bloc de glace ; la séquence d'instructions est alors figée, ce qui signifie qu'on peut ensuite s'y référer par un symbole unique. Un symbole graphique peut représenter une hiérarchie de structures de programmation. Ici la triple clef se "dilate" dans la boucle représentée au-dessous ; cette boucle est exécutée quatre fois et les trois "oiseaux"[43] chantent la portée en canon avec un décalage total de quatre mesures. La suite d'instructions représentée par chaque oiseau est indiquée à droite de la boucle. Quand l'oiseau survole une note, il chante cette note et à la fin de la portée, il revient au début.

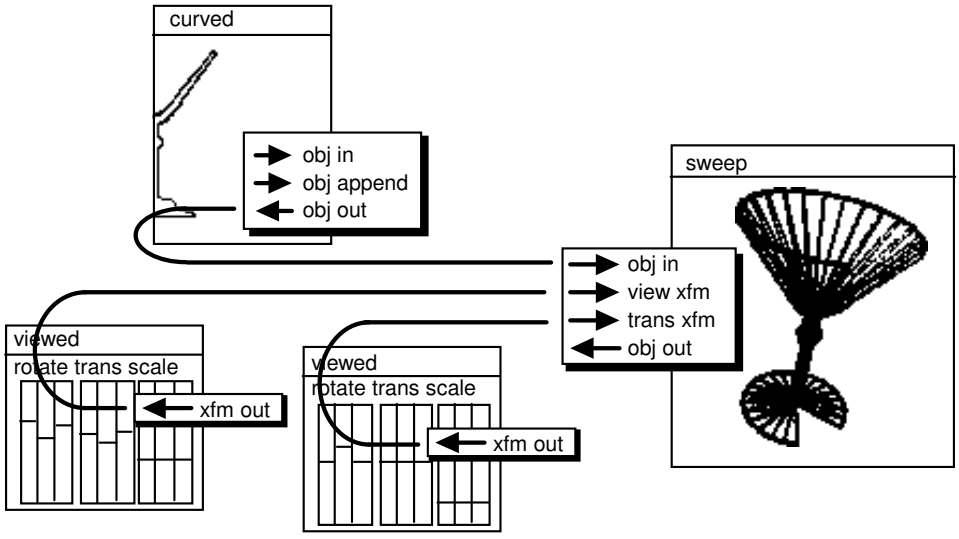


Figure 2.16 : Exemple de dessin réalisé avec ConMan

[43] Sur le dessin original, chaque oiseau a une couleur différente que l'on retrouve sur les portées et dans la boucle, ce qui permet d'identifier l'ordre d'exécution des différentes voix.

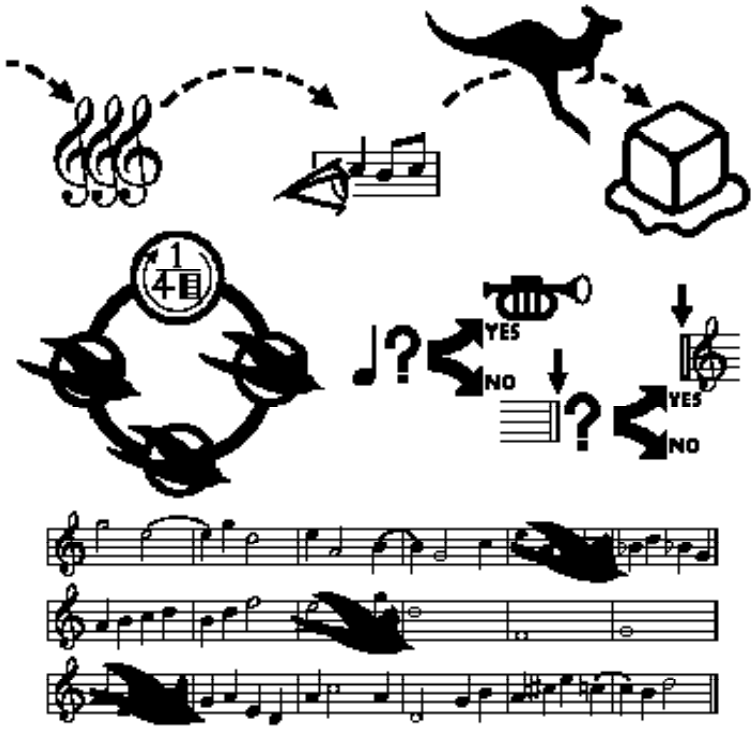


Figure 2.17 : Exemple de programme écrit avec le langage graphique Mandala

Cet exemple montre qu'il est possible de définir graphiquement le comportement d'un programme tout en y intégrant l'utilisateur. On peut donc imaginer une programmation iconographique pour le dialogue homme-machine. Celle-ci a l'avantage d'être claire et concise (par exemple le décalage des trois voix). Par contre la définition d'une norme dans les icônes apparaît nécessaire afin d'éviter une prolifération excessive et personnelle. On peut imaginer par exemple des icônes pour les types de données courants telles que les clients, les factures, etc., ainsi que pour les opérations telles que saisir, valider, supprimer...

5.2. Les formalismes à événements

Le concept d'événement a complètement révolutionné le monde des IHM. Il a permis à celles-ci d'être pilotées de plus en plus par l'utilisateur et non plus uniquement par le système. Les événements permettent, à l'inverse des états, de décrire à quoi l'interface (ou l'application) est susceptible de réagir. Les états permettent de dire quelles sont les étapes à suivre alors que les événements permettent de dire comment y arriver. Les événements peuvent servir aussi bien à spécifier le comportement d'une application qu'à l'implémenter. I. Chakravarty [CHAKRAVARTY 89] utilise par exemple le formalisme EDG pour représenter le dialogue en graphe d'événements. F. Bodart [BODART 89a] [BODART 89b] et I. Petoud [PETOUD 90] utilisent de même des graphes d'enchaînements basés sur les événements pour représenter la dynamique des traitements.

5.2.1. L'approche événementielle

La programmation par événements est issue de la réflexion des ergonomes sur la manipulation des programmes. Cette réflexion avait pour but d'obtenir des logiciels qui soient dirigés par les utilisateurs et non des programmes qui dirigent les utilisateurs. Les actions de l'utilisateur sur l'application génèrent des **stimuli** qui entraînent des changements d'état qui peuvent, à leur tour,

générer de nouveaux événements. L'insertion d'une disquette, une demande de formatage ou d'ouverture de fichier, une impression, la saisie d'un mot de passe, l'agrandissement d'une fenêtre, un scrolling, sont des exemples de stimuli. Le système ERS [HILL 87b], par exemple, est basé sur ce principe.

Les systèmes basés sur la programmation par événements sont des systèmes **ouverts** contrairement aux systèmes basés sur la programmation classique (systèmes fermés). Cette ouverture est à deux niveaux :

- *ouverture sur l'exécution* : le déroulement des sessions de travail n'est plus linéaire, mais subit les volontés de l'utilisateur,
- *ouverture sur l'évolution* : la conception et la maintenance de tels systèmes sont plus faciles puisque le concept d'événement implique une quasi totale indépendance des données et des traitements. On raisonne en terme de {un stimulus une action}. Les stimuli sont indépendants les uns des autres et on peut donc en ajouter et en retirer sans trop modifier le système existant.

Lien objet-événement

Les notions d'objet et d'événement sont proches l'une de l'autre, c'est pourquoi on les associe de plus en plus en matière d'IHM. Cette similitude s'explique par le fait que les objets sont indépendants, tout comme les événements, et qu'ils communiquent par des messages que l'on peut rapprocher des événements. Il existe cependant une différence importante : les objets impliquent qu'on doit préciser le destinataire du message, ce qu'on ne fait pas avec les événements. Par ailleurs les événements ne possèdent pas les caractéristiques d'héritage ou de polymorphisme des objets.

Conclusion

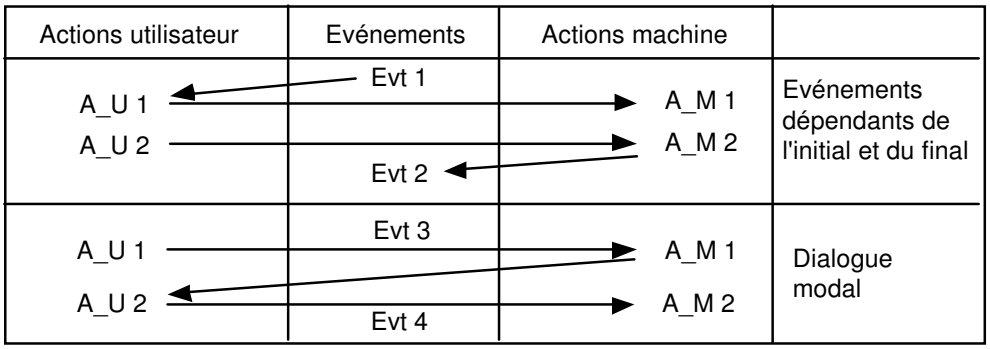
La programmation événementielle permet de simplifier l'écriture des traitements puisqu'on modularise beaucoup plus qu'avec un langage classique. De plus on peut intégrer le parallélisme sans trop de difficultés (pourvu que le système hôte le permette) et aboutir ainsi au dialogue multi-fil. Malheureusement l'approche événementielle a encore, à l'heure actuelle, des inconvénients que l'on ne sait pas résoudre faute de modèles suffisamment maîtrisés (sur le plan mathématique entre autres). Citons par exemple l'explosion combinatoire ou les appels récursifs ou bloquants[44]. De plus l'écriture et la maintenance du code ne sont pas toujours aisées pour les systèmes qui manipulent de nombreux générateurs d'événements comme les environnements de travail de type Windows.

Un autre inconvénient majeur est l'impossibilité de traiter directement la commande Undo. Chaque événement fait progresser le système vers un état qu'on peut considérer comme final. Rien ne permet de revenir en arrière après le déclenchement d'une opération à moins de disposer d'un module dédié à la mémorisation des étapes antérieures (gestion d'un historique). Le travail de I. Petoud [PETOUD 90] en est un exemple. Il répond à cette lacune grâce au graphe d'enchaînement (cf § 5.1.3).

[44] Si “A” produit un événement destiné à “B” et que “B” en produit un à son tour pour “A”, le système sera complètement bloqué. Ce type de blocage est difficile à détecter lors de l'écriture du code si on ne possède pas de vérificateur spécialisé.

5.2.2. Les scénarios

Les scénarios que nous présentons ici sont utilisés dans la méthode Grai [BERARD 90]. Ils permettent de représenter non seulement les actions de l'utilisateur et celle de la machine, mais également de dire qui pilote qui et à quelle occasion. Les scénarios se rapprochent en cela de la méthode Diane que nous verrons dans le Chapitre 5. Nous donnons ci-dessous deux exemples de scénarios, l'un pour une situation où les événements sont liés avec les situations initiales et finales, et l'autre où le dialogue est modal, c'est-à-dire que l'utilisateur doit attendre que la machine termine le traitement concernant la dernière action de l'utilisateur avant de pouvoir agir de nouveau. Dans le premier cas, l'utilisateur agit sur le système à la suite d'un événement (Evt 1) produit par ce dernier. L'action (A_U 1) de l'utilisateur entraîne une réaction du système (A_M 1). Dans le même temps, l'utilisateur a eu la possibilité d'agir de nouveau sur le système (A_U 2) ce qui a provoqué une autre réaction du système (A_M 2) ainsi que la production d'un événement (Evt 2). Dans le second cas, l'utilisateur agit sur le système (A_U 1), ce qui produit un événement qui déclenche une action machine (A_M 1). Celle-ci achevée, l'utilisateur peut de nouveau agir sur le système afin de produire de nouveau des événements, etc.



Exemples de scénarios avec la méthode Grai.

5.2.3. Les scripts

Hypercard [APPLE 88] est certainement le logiciel le plus connu dans l'univers Macintosh quant à la création d'applications interactives. Sa notoriété est dûe sans aucun doute à l'utilisation des scripts. Basé sur une approche objet et événementielle, le Macintosh était parfait pour le maniement de cet outil. A chaque élément interactif on associe une procédure écrite dans un langage spécialisé (Hypertalk). Cette procédure est déclenchée automatiquement par le logiciel lorsque l'élément interactif associé est manipulé par l'utilisateur.

En fait la notion de scripts est très proche de la programmation objet et événementielle. Lorsqu'un objet reçoit un message (qui est lui-même issu d'un événement), il exécute le traitement (le script) correspondant. Le terme script est donc plus du vocabulaire qu'une notion à part entière et c'est pourquoi, suivant les auteurs, on lui attribue des sens différents ou bien on le rapproche de notions existantes (par exemple des règles [COLLET 87]).

5.3. Les formalismes hybrides

Les formalismes hybrides sont des formalismes qui utilisent à la fois les états et les événements. Certains sont issus de formalismes basés sur les états auxquels on a ajouté la notion d'événement (par exemple les règles), d'autres ont été entièrement bâtis dans le but d'une utilisation conjointe

des états et des événements (par exemple les langages tels que Esterel [BOUSSINOT 91]). Leur représentation peut être graphique ou textuelle, mais elle fait toujours ressortir la correspondance entre les états et les événements.

5.3.1. Les objets

Le domaine des IHM nous oblige de plus en plus à réaliser des interfaces évolutives avec le minimum d'efforts et tout en perdant le moins de temps possible pour la réalisation. Les objets satisfont en partie à cette demande. Ils possèdent la majorité des critères principaux du génie logiciel (modularité, robustesse, etc.) [MEYER 90]. La notion de classes nous fournit la modularité, l'héritage nous fournit la réutilisabilité et l'extensibilité tandis que les messages nous fournissent un moyen de contrôle simple et cohérent. Grâce aux classes, il est possible de regrouper les traitements par type de comportement et non plus seulement par procédures. Les classes permettent par ailleurs de définir le comportement d'une application de manière générale. Ce n'est qu'à l'exécution que l'on travaillera sur les représentants de ces classes, c'est-à-dire sur les instances. Par exemple à l'implémentation on décrira les méthodes liées à la classe Client et on dira que la classe Facture est liée à la classe Client. A l'exécution l'application manipulera une facture précise (c'est-à-dire une instance de la classe Facture) et un client précis (idem).

La définition du terme objet n'est pas encore à l'heure actuelle unanimement reconnue. Elle évolue suivant les auteurs [LINDGREEN 74] [BENCI 76] [MIJARES 76] [PEAUCELLE 77] [GUSTAFSON 82], [MEYER 90] et surtout en fonction de l'utilisation que l'on veut en faire (base de données, calcul scientifique, simulation...). Parmi toutes les définitions que l'on peut trouver, on peut dire qu'un objet est constitué d'un ensemble d'attributs qui représentent son état, et d'un ensemble de traitements (méthodes) qu'il est capable de réaliser. Ces méthodes sont les seuls points d'accès aux attributs ce qui procure aux objets à la fois un faible couplage et une forte structuration.

En général les objets communiquent par échanges de messages qui déclenchent des appels de méthodes. Un objet qui reçoit un message sait toujours s'il est capable ou non de le traiter. Cette caractéristique provient de la notion d'héritage qui permet à un objet fils de répondre aux mêmes méthodes que son père. Par ailleurs un objet fils peut répondre différemment de son père pour la même méthode en surchargeant ou en restreignant la méthode de son père.
La notion d'héritage est intrinsèquement liée avec celle de classe et elle est très importante en programmation objet. Elle permet à la fois de réutiliser du code existant sans connaître son contenu et de redéfinir une partie du comportement d'un objet sans modifier (ou presque) le reste. La réutilisation et la surcharge, qui sont issues de l'héritage, permettent de généraliser ou de spécialiser un comportement. A partir d'une classe générale Polygones, on peut par exemple aboutir à des classes telles que Carré ou Triangle en restreignant les traitements à chaque niveau (spécialisation), ou à l'inverse partir d'une classe telle que LigneDroite et aboutir aux polygones en généralisant de plus en plus. En fait le choix se fait en fonction des objets que l'on manipule et des traitements que l'on veut réaliser. Il n'y a pas vraiment de principe de choix et cela peut se révéler ennuyeux dans certains cas. Par exemple vaut-il mieux dire qu'un cercle est la spécialisation d'une ellipse ou bien qu'une ellipse est une extension des cercles ? Le choix de l'une ou l'autre des solutions peut se révéler important au niveau des temps de traitements (périmètre, surface...) si on doit manipuler de nombreux objets.

Le formalisme objet est un formalisme particulièrement adapté pour l'implémentation des interfaces homme-machine. Il permet la gestion d'entités autonomes dans leurs états et dans leurs

comportements[45]. Une fenêtre a un comportement qui dans la plupart des cas, est indépendant des autres. Il est donc facile de lui associer une structure d'objet. Smalltalk par exemple utilise cette approche avec le modèle MVC (une fenêtre utilise des messages pour communiquer avec d'autres fenêtres). On conserve ainsi une relative indépendance. Étant donné que l'on cherche à donner une latitude décisionnelle de plus en plus large à l'utilisateur, une fenêtre doit permettre de travailler sur plusieurs tâches en même temps. Les objets fournissent cette possibilité grâce à leur autonomie.

Les objets sont un des rares formalismes qui permettent de travailler aussi bien par états que par événements, avec les mêmes facilités et le même pouvoir d'expression. Les attributs représentent les états que l'objet est capable de prendre, les méthodes représentent les messages (événements) auxquels il est capable de répondre. Cette propriété leur permet d'être utilisés pour implémenter des modèles (ou pour traduire des formalismes) issus de ces deux tendances.

Le concept objet n'est pas récent. Il est apparu véritablement avec Smalltalk en 1972 [GOLDBERG 84] et on le retrouve dans de nombreux produits : Graffiti [KARSENTY 87], Aïda/Masaï [ILOG], GWUIMS [SIBERT 86], MacApp [SCHMUCKER 86], etc. Il permet de travailler aussi bien au niveau conceptuel qu'au niveau implémentation. Mode [SHAN 90] utilise par exemple des objets pour décrire la sémantique des actions des éléments de l'interface et P-A Brès [BRES 90] définit la notion d'objet naturel (c'est-à-dire un objet qui est significatif pour l'utilisateur, par exemple un client ou un article) dans les systèmes d'information. On le retrouve également dans diverses architectures [CLEMENT 90] [RUSSI 90] ou méthodes telles que Rémora[46] [ROLLAND 88].

L'orienté-objet n'est cependant pas la panacée. Utiliser des objets n'implique pas que le système final sera évolutif ou modulaire. Dans la plupart des cas, on est obligé d'utiliser une méthode associée à ce type de conception (Hood [HEITZ 87] [HOOD 89], Grady Booch [BOOCH 91]...). Une solution intéressante est peut-être d'associer un langage objet à un langage classique, le premier étant utilisé pour les données (classes) et une partie des traitements (héritage), le second pour les primitives de l'application. Dans le cadre de la gestion du dialogue, les langages orienté-objet paraissent être une bonne voie. Des modifications sont apportées aux concepts de base afin de permettre un éventail d'applications de plus en plus large. Citons par exemple S. Talbot [TALBOT 91] qui définit des concepts d'exceptions au niveau des objets.

5.3.2. Les règles

Les innovations en Intelligence Artificielle ont contribué à l'évolution des IHM. Les systèmes experts sont en effet de bons générateurs de dialogue dans le sens où ils sont capables de produire des questions et des réponses suivant des faits établis. Les informaticiens ont donc eu l'idée de les utiliser pour gérer ou pour créer des interfaces.

---

[45] On peut facilement ajouter de nouveaux objets et les connecter à ceux existants, mais cette caractéristique cache un inconvénient : la difficulté du suivi d'erreurs. Puisque chaque objet est autonome, il n'y a plus de programme principal qui contrôle le système. Pour connaître le fonctionnement du système, on est obligé de connaître au minimum le comportement de chacun de ses composants ainsi que les liens entre eux, ce qui peut rendre difficile le déboggage. Une erreur qui apparaît dans l'objet X pourra par exemple provenir d'un objet Y après avoir cheminé dans plusieurs autres objets. Par contre si les objets sont suffisamment robustes (au sens génie logiciel), l'erreur ne se propagera pas ou peu. De la même façon si les objets sont suffisamment indépendants, l'ajout d'un nouvel objet ne doit perturber que ces voisins directs et non pas tout le système.

[46] Rémora est basée sur trois composants (les objets, les opérations et les événements), et son formalisme a été utilisé pour spécifier le dialogue des IHM par Vignaud [VIGNAUD 89]. Ce formalisme permet de mettre en relation les objets manipulés, les opérations qu'ils manipulent ainsi que les conditions associées (événements).

Les règles de production sont une manière de représenter du dialogue sous forme textuelle. Elles permettent une écriture déclarative et non plus procédurale ce qui permet une évolution relativement aisée, mais implique une lisibilité plus ardue à cause du caractère de vrac qu'elles imposent.

Nous verrons au Chapitre 6 comment nous utilisons les règles pour gérer la dynamique et l'aide.

5.3.2.1. Spécification et conception avec les inférences

L'inconvénient que présente la plupart des formalismes est la nécessité d'écrire complètement les traitements (ceux de l'application et ceux du dialogue). Cette écriture est généralement réservée à un expert de la programmation, mais elle peut également être complétée par les critiques d'ergonomes et de concepteurs. C. Wiecha [WIECHA 89] remarque qu'il serait judicieux de coder la connaissance de ces experts sous forme de règles afin de disposer d'une génération automatique du code de l'interface. Plus précisément C. Wiecha propose de disposer de règles de style, écrites par un expert de style, qui permettraient de générer des interfaces en fonction de l'utilisateur et de l'application. Le rôle de l'expert ne serait pas de dire qu'il faut utiliser la police Times en 12 points comme police par défaut dans un traitement de texte, mais plutôt de donner les règles qui lui permettent d'aboutir à cette conclusion. On utiliserait donc des règles portant sur des méta-connaissances plutôt que sur des connaissances simples. La difficulté principale de cette approche est de parvenir à coder la connaissance de l'expert sous forme de règles exploitables. C. Märtin [MÄRTIN 90] reprend cette idée de manière plus simpliste. Il utilise pour cela des règles de présentation élémentaires qui, une fois combinées avec des règles correspondant aux désirs de l'utilisateur, permettent de générer une interface dont la présentation est adaptée à ce dernier.

Les règles peuvent être utilisées à un niveau d'abstraction encore plus élevé. Ergo-Conceptor [KOLSKI 91] intègre par exemple des règles de production permettant de générer automatiquement les spécifications de l'interface qui pourront ensuite être exploitées directement par le concepteur. Les règles utilisent de plus une base de données ergonomiques dans le but de respecter des normes dans la présentation et le comportement. Il existe d'autres systèmes qui utilisent cette particularité. C'est le cas de UIDE [FOLEY 89] qui génère également à partir d'inférences les spécifications et le code de l'application depuis une base de données. Certains systèmes sont plus spécialisés dans le dialogue homme-machine comme Synics [GUEST 82] ou Serpent [BASS 90] qui utilisent des règles de production pour représenter le dialogue.

5.3.2.2. Implémentation avec les inférences

La plupart des systèmes qui emploient des règles de production pour le dialogue utilisent en général des règles qui ont la forme $\{\text{condition} \Rightarrow \text{action}\}$ où la condition est exprimée par une combinaison de "et" et de "ou" et qui utilisent des événements et des booléens. Les actions sont exprimées quant à elles par une suite de commandes à déclencher, de messages à envoyer ou d'événements à générer. Sassafras [HILL 86] utilise un langage spécial qui est basé sur ce type de règles pour les dialogues concurrents. Il n'utilise que des booléens et des événements. Le système ERS [HILL 87c] est également basé sur ce principe.

Il est donc courant d'associer les règles de production et les événements. On peut alors écrire les traitements accompagnés de leurs conditions de déclenchement de façon isolée et non pas dispersée à travers tous les objets du système susceptibles de gérer ces événements. La gestion du dialogue devient donc plus modulaire, mais elle perd d'un autre côté un peu de son efficacité par rapport à une gestion dispersée. Une gestion groupée oblige en effet tous les événements à transiter par un seul et même module alors qu'il peut être intéressant dans certains cas d'avoir des

modules séparés qui traitent chacun les événements dont ils ont la charge. Un avantage des règles est le type déclaratif qu'elles utilisent. Une règle telle que “If mouse.clicked = yes then x := mouse.x and y := mouse.y” [HARRIS 90] est aisément compréhensible.

Les systèmes actuels qui utilisent les règles ne fonctionnent qu'en chaînage avant. De ce fait ils récupèrent les inconvénients de l'approche événementielle, par exemple l'impossibilité du retour arrière (Undo). Résoudre de tels problèmes est évidemment ardu, mais l'enjeu est d'importance. Une première solution consisterait à tenir à jour un historique qui permettrait d'annuler les dernières opérations, mais cela risque d'entraîner des conflits sur les déclenchements des règles par exemple à cause d'incompatibilité. Une seconde solution serait peut-être de disposer de règles réversibles, c'est-à-dire capables de fonctionner normalement dans le cas où les prémisses sont vraies et d'exécuter le contraire des traitements prévus dans le cas où les prémisses sont fausses [BEAUDOUIN 91].

Les auteurs utilisent souvent plusieurs bases de règles en interaction. C'est le cas de D. Benyon [BENYON 90] qui utilise un toolkit avec trois bases de règles : une pour l'adaptation du système au niveau de la présentation, une pour des inférences permettant la mise à jour du modèle de l'utilisateur et une autre pour évaluer l'efficacité du système ainsi créé. A l'opposé on trouve des systèmes qui utilisent des règles de manière dispersée. Fakir [COLLET 87] utilise des formulaires dont la dynamique est basée sur des règles qui permettent à la fois des calculs et un contrôle sémantique des saisies. Chaque formulaire possède son propre ensemble de règles. Ces règles sont écrites dans un langage particulier et se présentent sous la forme “WHEN {événements} BEFORE {actions} AFTER {actions}”, ce qui se traduit par “quand l'ensemble des événements est vrai, exécuter BEFORE. Si le résultat est correct, exécuter AFTER sinon arrêter le traitement”.

Certains systèmes utilisent les inférences sans manipuler explicitement des règles. F. Schiele [SCHIELE 90] génère par exemple des tâches qui sont inférées à partir des actions de l'utilisateur. Les deux commandes “Dessiner cercle”, “Remplir cercle avec du rouge” deviennent ainsi équivalentes à la nouvelle commande “Dessiner cercle (couleur) := Dessiner cercle (c) + Remplir (c) avec (couleur)”. Le système crée ainsi de nouvelles tâches pour optimiser les séquences d'actions répétées. Sur le même principe, on trouve Eager [CYPHER 91] ou Péridot [MYERS 88b] qui, à partir des actions répétées de l'utilisateur, parviennent à généraliser ces actions afin de les traduire sous forme visuelle pour la présentation.

Conclusion sur l’utilisation des règles

L'utilisation des règles est intéressante pour trois raisons majeures :

- elles sont généralement écrites dans un langage clair, simple et non ambigu. Nous verrons dans le Chapitre 6 que la syntaxe des règles que nous utilisons est très simple.
- leur écriture permet d'exprimer des traitements complexes de façon déclarative et non plus procédurale,
- il est relativement facile de maintenir une base de règles sans trop modifier ou perturber l'existant, ce qui n'est pas le cas avec le procédural. Les règles que nous utilisons sont réparties dans les différents objets qui les manipulent. Ainsi la modification d’une règle ne se répercute pas en dehors de l’objet concerné.

Par contre les règles sont synonymes de :

- difficulté de traduction de la connaissance en règles. Ce travail est réservé aux cognitiens qui bénéficient d'une grande expérience dans ce domaine. Nos règles n’exprimant que des enchaînements élémentaires, nous ne sommes donc pas confrontés à ce type de problème.

- difficulté de maintenance lorsque les bases deviennent volumineuses (problèmes de cohérence). Le cloisonnement des règles et leur génération automatique nous évite ce désagrément.
- rendement insuffisant si les bases sont trop volumineuses ou mal construites. Dans le cadre des IHM, il est intéressant de se demander si un moteur d'inférence est plus rapide que du traitement événementiel pur,
- absence de concurrence bien que certains travaux relatent de bases de règles parallèles [COAD 90]. Notre implémentation en Smalltalk/V résout partiellement ce problème étant donné que ce langage nous fournit un environnement pseudo-multi-tâche.

5.3.3. Les Réseaux de Petri

Ils sont utilisés principalement pour représenter des traitements séquentiels parfaitement connus à l'avance. C'est le cas par exemple des opérations qui sont dispensées dans un atelier flexible. Ils n'ont cessé d'évoluer depuis leur création [PETRI 62] en se voyant augmentés de notions de couleurs pour différencier les jetons, de notion de temporalité, etc. Il existe à l'heure actuelle des systèmes qui utilisent les Réseaux de Petri en conjugaison avec une approche objet ou événementielle. Scénarioo représente le dialogue homme-machine par un ensemble de graphes d'événements. Chaque élément composite de l'interface (par exemple une fenêtre, mais pas un bouton) est relié à un de ces graphes. En fait chaque graphe est un Réseau de Petri[47]. La figure 2.18.a représente un exemple de fenêtre de saisie de mot de passe associée à une fenêtre d'aide, la figure 2.18.b représente le réseau associé.

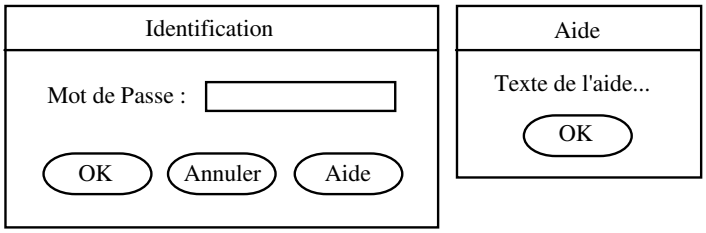
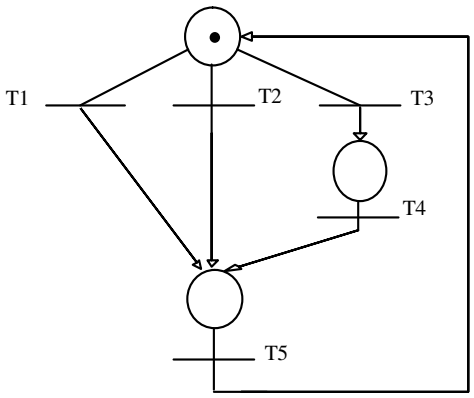


Figure 2.18.a : Une fenêtre de saisie de mot de passe et sa fenêtre d'aide associée



T1 correspond au clic sur OK et déclenche la vérification du mot de passe
T2 correspond au clic sur Annuler et ferme la fenêtre
T3 correspond au clic sur Aide et affiche la fenêtre d'aide
T4 correspond au clic sur OK dans l'aide et ferme la fenêtre d'aide
T5 ne sert qu'à terminer proprement le réseau et ne déclenche rien.

Figure 2.18.b : Le graphe d'événement associé à la fenêtre de saisie de mot de passe.

Protob [RUSSI 90] utilise également les Réseaux de Petri ainsi que l'approche objet. Chaque objet est un Réseau de Petri amélioré. Le fonctionnement d'un tel réseau est le même que celui des réseaux classiques ; ils ont la caractéristique supplémentaire de disposer de “portes” qui leur permettent d'échanger des jetons avec d'autres réseaux (Figure 2.19). Protob possède également la

[47] Il existe également un Réseau de Pétri particulier (appelé graphe démon) qui sert à gérer les requêtes non prévues.

caractéristique de décomposer des objets complexes en sous-objets et donc un réseau en sous-réseaux.

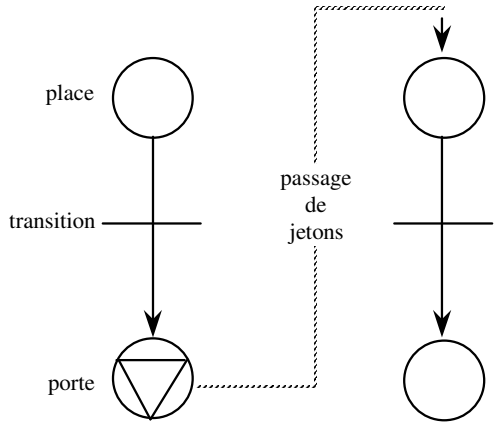


Figure 2.19 : Un Réseau de Petri avec porte dans Protob

Les Réseaux de Petri (RdP) sont clairs et lisibles. Pourtant leur manque de structuration et leur faible pouvoir d'expression conduit à les rendre difficile à utiliser pour la modélisation de systèmes réels. Leur éventuelle utilisation dans les IHM demande de nouvelles fonctionnalités telles que la gestion des événements et des objets. Trois extensions récentes permettent d'aborder la réalisation d'applications interactives : les Réseaux de Petri à Objets, les Objets Coopératifs et les Objets Coopératifs Interactifs.

- les *Réseaux de Petri à Objets* (RPO) [SIBERTIN 85] [BASTIDE 90] sont un formalisme utilisé pour modéliser le comportement général d'un système. Un Réseau de Petri à Objets est un Réseau de Petri dans lequel :
  - les jetons sont remplacés par des objets. Les RPO prennent donc en compte les données dans les modèles,
  - les arcs sont étiquetés par des variables qui décrivent le flux des objets,
  - des pré-conditions et des post-conditions permettent de conditionner le franchissement d'une transition sur la valeur des objets,
  - les transitions permettent d'exécuter des traitements qui sont composés d'appels de méthodes.

  L'intégration des objets permet d'augmenter le pouvoir d'expression et la concision des Réseaux de Petri, d'un autre côté elle réduit l'étendue de la validation formelle qui ne prend pas en compte la valeur des objets (indécidabilité accrue). Enfin les RPO ne favorisent pas non plus la modularité puisqu'un système est modélisé par un seul réseau.

- les *Objets Coopératifs* [BASTIDE 91] [BASTIDE 92] sont un formalisme couplant les RPO et l'approche objet. Un objet coopératif est défini par ses d'attributs, ses méthodes et son comportement vis-à-vis du système. Le comportement est modélisé par un RPO appelé OBCS pour OBject Control Structure. Un système modélisé par les objets coopératifs est constitué d'un ensemble d'objets coopératifs qui communiquent. Le comportement du système est modélisé par un RPO (appelé WSCS pour Whole System Control Structure) qui peut être reconstruit automatiquement à partir des OBCS des objets coopératifs.
  Les objets coopératifs permettent donc de définir à la fois la dynamique des objets et celle du système complet. Il est possible de les analyser formellement pour déterminer si un système est bien construit ou non (par exemple s'il existe une bonne compatibilité entre les objets coopératifs qui le composent). En conclusion les objets coopératifs fournissent :

- des systèmes bien structurés,
- des Réseaux de Petri plus petits qu'avec l'approche Réseau de Petri à Objets,
- un langage objet qui permet la gestion de la concurrence.

Malheureusement on retrouve encore quelques inconvénients énoncés pour les RPO, principalement la réduction de la validation formelle qui reste indépendante de la valeur des objets. La décomposition des systèmes est basée sur leur décomposition en systèmes d'objets ; ces objets étant de faible taille, les techniques de structuration classique deviennent inutiles[48].

- les *Objets Coopératifs Interactifs* [PALANQUE 92] [PALANQUE 93] sont également un formalisme. Ce dernier, basé sur les Objets Coopératifs, apporte une solution plus spécialement tournée vers les interfaces. Dans cette approche, un objet est défini par ses attributs, ses méthodes, et son comportement (OBCS) vis-à-vis de l'environnement auxquels vient s'ajouter une présentation (ensemble de widgets). Les objets coopératifs interactifs possèdent également une fonction d'activation qui, à chaque couple (widget, action sur widget) associe un service (c'est-à-dire une méthode) d'un objet coopératif interactif. Une méthode de conception est associée aux Objets Coopératifs Interactifs. Avec cette méthode le comportement d'une fenêtre est modélisé par un seul objet coopératif interactif (Figure 2.20.a et 20.b).

La représentation externe est gérée par les objets coopératifs interactifs par l'intermédiaire de l'activation et de la désactivation des widgets qui sont directement dépendantes de la franchissabilité des transitions. Grâce à la validation formelle, les objets coopératifs interactifs permettent une validation a priori du comportement de l'interface. Toutefois ils ne sont vraiment manipulables que par des informaticiens. Par ailleurs la méthode de conception qui leur est associée n'intègre pas de facteurs humains (par exemple le niveau de l'utilisateur).

Remarquons pour terminer que les objets coopératifs interactifs ont l'avantage d'expliciter à la fois les états et les événements [PALANQUE 93]. Les places d'états (par exemple Sélectionné dans la figure 2.20.b) permettent de décrire les états que le système peut prendre, les places d'événements[49] décrivent les relations entre l'application et son environnement et les arcs permettent de visualiser les changements d'états en fonction des événements en entrée.

[48] Il est toutefois possible de structurer les RPO par les techniques de structuration de Réseaux de Petri classique [HUBERT 87].

[49] Une place d'événement est une place qui ne possède que des arcs en sortie. Elles sont représentées sur l'OBCS avec des arcs brisés (par exemple avant les transitions Éditer et Quitter). Leur rôle est de récupérer les événements en provenance de l'environnement (par exemple l'interface) par l'intermédiaire des liens associées (par exemple des widgets dans les IHM).

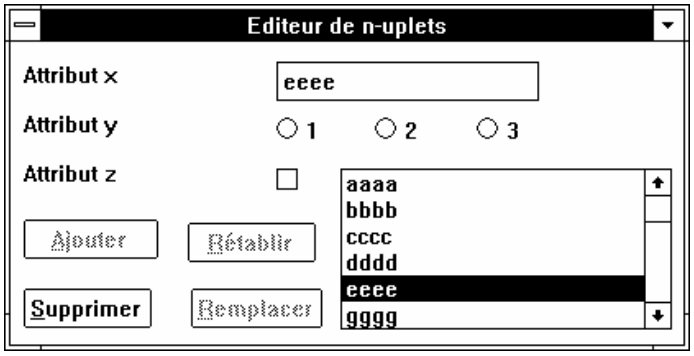


Figure 2.20.a : Exemple de représentation externe d'un objet coopératif interactif (extrait de [PALANQUE 92])

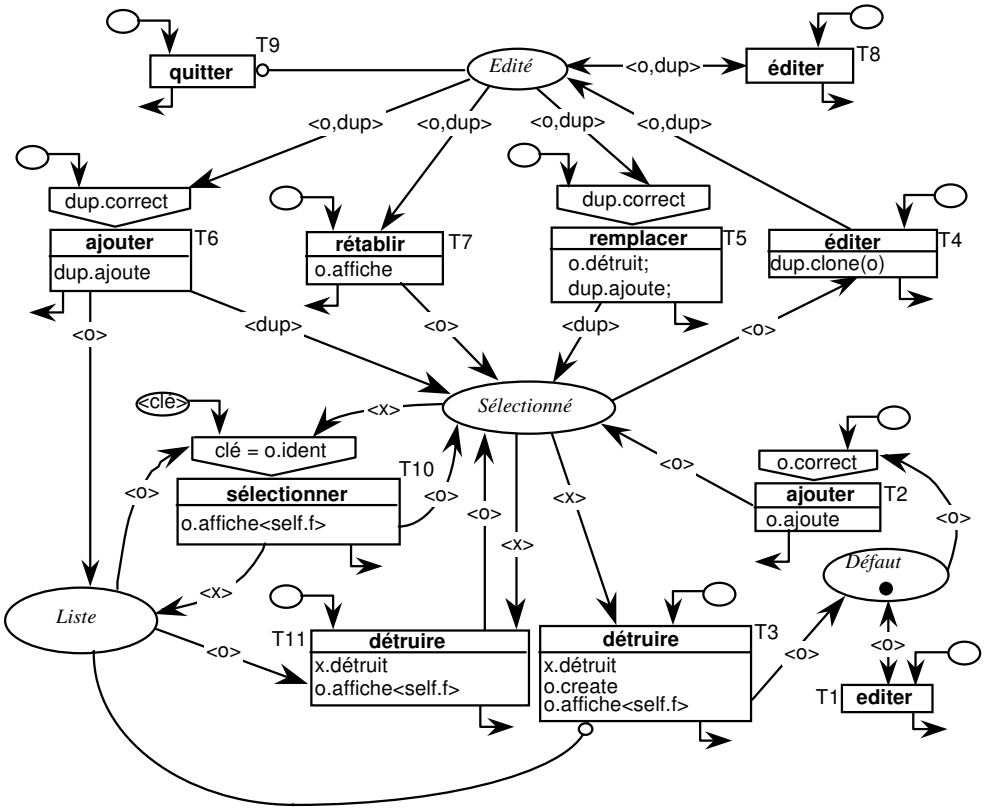


Figure 2.20.b : OBCS de l'objet coopératif interactif représenté par la fenêtre de la figure 2.20.a (extrait de [PALANQUE 92])

5.3.4. Les langages spécialisés

Outre les grammaires, il est possible de spécifier le dialogue homme-machine directement par l'intermédiaire d'un langage de programmation. Depuis longtemps on utilise des langages tels que C ou Pascal, mais ceux-ci sont remplacés de plus en plus par des langages adaptés aux IHM. Ces derniers permettent entre autres d'élever le niveau d'abstraction et de spécifier plus facilement les enchaînements d'opérations ainsi que les conditions associées. Certains de ces langages permettent de générer du code dans un langage classique comme C facilitant ainsi la portabilité. Nous allons rapidement énumérer ici quelques uns de ces langages spécialisés afin de montrer un éventail des possibilités qu'ils offrent.

IDL (Interface Definition Language) [FOLEY 87] [BOUSSE 89] permet de spécifier formellement une interface. Il décrit les types d'objets manipulés, leurs propriétés, les relations entre les classes et les actions que l'on peut exécuter sur ces objets. Malheureusement il n'existe pas de méthode associée pour décrire le dialogue en lui-même.

Serpent [BASS 90] possède son propre langage Slang. Un programme Slang définit et énumère une collection d'objets d'interaction ainsi que les actions accessibles pour l'utilisateur. Un objet dans Serpent est constitué d'attributs de présentation et de méthodes d'interaction. De par son écriture, il ressemble au code généré par un éditeur de fenêtres tels qu'on en trouve dans les toolkits.

Siroco [NORMAND 92a] est un langage utilisé dans l'environnement Guide. Il permet d'implémenter la spécification conceptuelle d'une application et de décrire les données et les opérations qui constituent les concepts du domaine de l'application par rapport aux tâches de l'utilisateur (les applications gérées par Siroco sont centrées sur les données). Il donne naissance à une architecture à base de workspaces[50], à du code (en-tête d'opérations, contrôleurs divers, types de données) et à une représentation externe (menus, fenêtres). Siroco a en outre l'avantage de distinguer la dimension de fonctionnement (grâce aux concepts) de celle d'utilisation (grâce aux perspectives et aux workspaces).

Gradient [ALTY 89] utilise le langage Kee pour spécifier le dialogue. Ce code Kee est généré à partir d'une description graphique semblable à des diagrammes états-transitions. L'avantage de ce code est sa capacité à être analysé algébriquement pour contrôler sa cohérence.

Alexander [ALEXANDER 87] propose un modèle qu'on peut rapprocher des langages de par son formalisme. “L'objectif de EventCSP, fondé sur CSP (Communication Sequential Processes), est d'abstraire la structure du dialogue des détails des entrées-sorties et des transformations d'états qui interviennent” [PETOUD 90]. EventCSP est basé sur des connexions entre événements et processus. Il propose des structures de choix, de séquencialité, de parallélisme et d'interruption de processus.

Esterel [BOUSSINOT 91] est un langage qui est utilisé surtout pour les systèmes réactifs temps réel. Il se rapproche de la programmation événementielle par l'utilisation de signaux. Un traitement peut émettre des signaux (emit signal) ou en attendre (wait signal) et deux traitements peuvent fonctionner en concurrence.

Algae [FLECCHIA 87] est également un langage événementiel qui génère du C étendu. Les différences notables avec C sont de nouvelles déclarations telles que *event*. Algae autorise la programmation concurrente et donc par conséquent le dialogue multi-fil.

5.3.5. Les graphes de relations

Ils sont utilisés dans la méthode de conception proposée dans [MÄRTIN 91]. Cette méthode repose sur un modèle entité-association étendu et sur un modèle orienté-objet. Les relations entre les objets ainsi que l'évolution de ceux-ci sont représentés par trois graphes :

[50] Les workspaces sont des espaces de travail virtuels dans lesquels l'utilisateur peut manipuler des opérations qui sont agencées suivant des arbres de séquencement. Les workspaces sont accessibles à tout instant pour l'utilisateur à condition qu'ils soient actifs. Pour une messagerie électronique, on dispose par exemple d'un workspace Traiter message et d'un autre workspace qui est Connexion à la messagerie.

- le *graphe de relations inter-objets*[51] a pour objectif de déterminer la dynamique des objets du système ainsi que la dynamique globale du système. Il s'appuie sur deux notions essentielles :
  - les événements externes qui déclenchent des activités pour le système, et les événements internes qui font évoluer les objets,
  - les états des objets qui correspondent à des attentes d'événements déterminés.

  Un tel graphe est construit selon deux axes : la chronologie des événements et la dynamique de chaque objet (Figure 2.21).

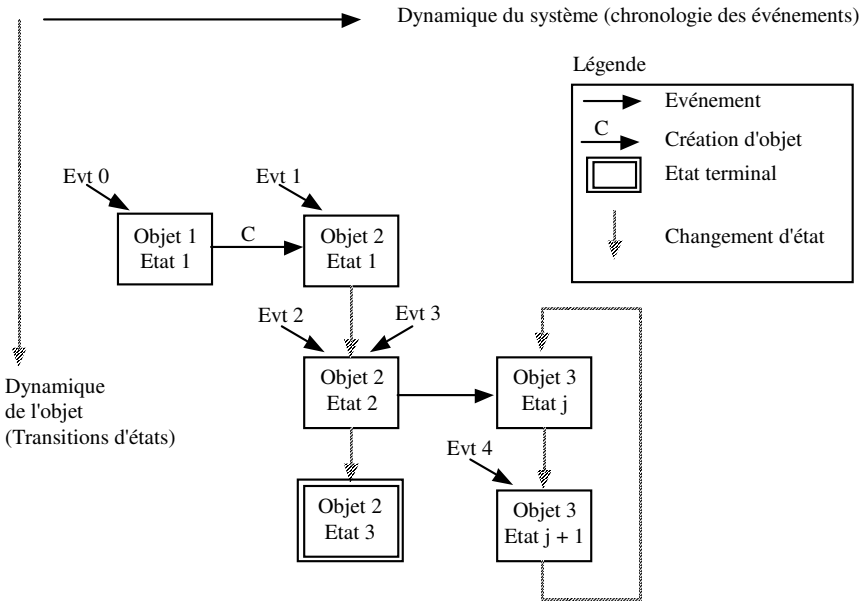


Figure 2.21 : Graphe des relations inter-objets

- le *graphe des états d'un objet* qui met en évidence le passage d'un état à un autre pour un même objet. Il doit faire apparaître les événements qui conditionnent les changement d'états ainsi que les post-conditions qui permettent de s'orienter sur un état plutôt que sur un autre (Figure 2.22).

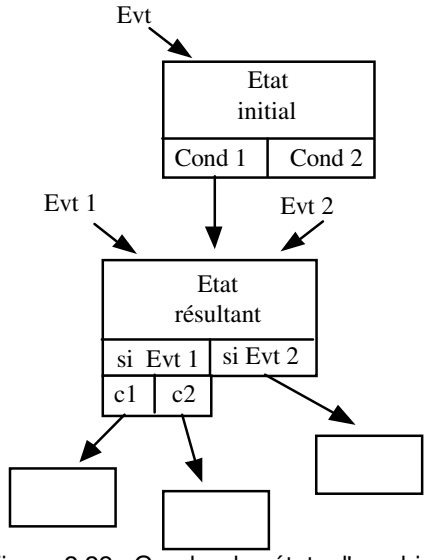


Figure 2.22 : Graphe des états d'un objet

[51] On retrouve une graphe équivalent dans Merise/2 [PANET 91] avec le graphe de cycle de vie des objets.

- le *graphe de communication entre les objets* qui offre une vue globale du système et plus particulièrement les objets impliqués dans le processus, les points d'entrées et de sorties des processus et enfin la chronologie des communications entre les objets du système par rapport au processus décrit (Figure 2.23).

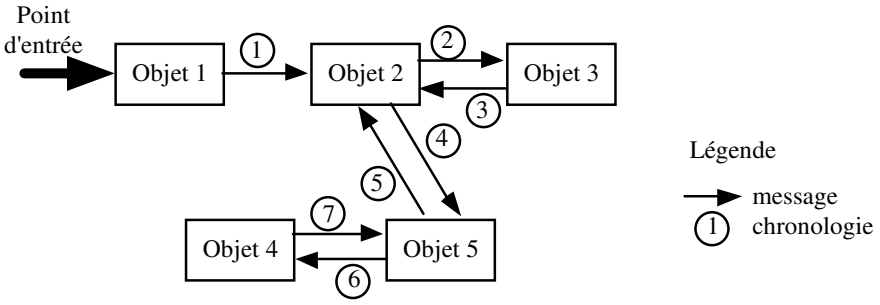


Figure 2.23 : Graphe de communication entre les objets

La méthode présentée s'applique aux systèmes d'information, mais on peut imaginer l'appliquer aux interfaces homme-machine de manière générale. Un objet de la méthode deviendrait alors un objet de la représentation externe ou bien un objet manipulé par l'application sous-jacente. Cette méthode aurait alors l'avantage de décomposer clairement l'interface en objets élémentaires et bien structurés au niveau comportemental. De plus le passage à l'implémentation en objet est relativement aisé (cf [MÄRTIN 91] p772). Par contre elle est contraignante sur les communications entre les objets (chronologie fixée et rigide[52]) et elle a l'inconvénient d'être lourde à mettre en œuvre de par le nombre de graphes à écrire.

5.4. Conclusion sur les formalismes

Nous venons de voir quelques formalismes intéressants pour la gestion du dialogue dans les IHM. Ils peuvent être utilisés séparément ou conjointement. A l'heure actuelle il semble que les formalismes qui utilisent conjointement les états et les événements soient les plus prometteurs grâce à leur puissance d'expression, à la latitude décisionnelle qu'ils laissent à l'utilisateur et à leur capacité d'évolution. Par ailleurs les constructeurs de matériel et de logiciels s'engagent de plus en plus dans cette voie. Les environnements sont à présent tous axés sur la gestion par événements ce qui pousse les concepteurs et les utilisateurs à continuer dans cette direction. Une question demeure cependant quel que soit le formalisme et même quel que soit le logiciel : faut-il construire le dialogue homme-machine à partir des données ou des traitements ? Il est évident que les deux sont intimement liés, mais bien que la plupart des méthodes intègrent à la fois la gestion des données et des traitements, aucune ne répond franchement à la question.

6. Conclusion

Nous avons montré dans ce chapitre les différents aspects du dialogue homme-machine (définitions, formalismes de représentation, types de contrôle, etc.). Ceci nous permet de constater que la recherche sur ce point précis des IHM est très active et promet encore de nombreuses découvertes. La tendance actuelle s'orientant surtout vers l'utilisation du concept objet et vers une

[52] On pourrait s'abstraire de cette contrainte en spécifiant les messages qui voyagent entre les objets sans leur associer une chronologie.

génération automatique des interfaces (cf Chapitre 3), seuls quelques uns des formalismes cités plus haut peuvent véritablement être utilisés dans cette optique. Par ailleurs les logiciels des années à venir feront sans aucun doute de plus en plus appel au multimodal et au collecticiel, et il reste à savoir si les résultats actuels en matière de dialogue homme-machine ne devront pas être remis en cause pour ce nouveau type de communication.

Chapitre 3
La Génération Automatique

1. Introduction

Terme à la mode depuis quelques années au même plan que "objet", la "génération automatique" n'est pas encore à l'heure actuelle clairement définie. On peut dire en effet que la génération automatique consiste à faire produire du "code" à un ordinateur à partir de "données" que l'utilisateur lui a fournies. Nous verrons que ce code et ces données peuvent être de natures diverses et à usage variés.
La génération automatique des applications interactives touche plusieurs domaines différents et proches à la fois. Elle est utilisée aussi bien lors de la conception que lors du prototypage. Par ailleurs elle peut être appliquée à toute l'application ou uniquement à l'interface.

L'emploi du terme "génération" est-il vraiment correct dans les produits actuels ? Nous pensons que le terme "génération" est souvent mal employé. Lorsqu'on parle de génération d'interface par exemple, les outils se contentent de traduire une représentation en une autre équivalente. Par exemple les toolbox traduisent la primitive CréeFenêtre (x,y,h,l) en un rectangle à l'écran et en une entité manipulable par le système, mais c'est l'utilisateur qui a décidé lui-même d'utiliser cette primitive et ses paramètres auxquels il a donné une valeur. De même, pour les toolkits et les UIMS, l'utilisateur crée à l'écran son interface (donc il choisit tous les composants avec leurs attributs tels que la couleur et la taille) et l'UIMS (ou le toolkit) se charge de traduire l'interface sous forme de code réutilisable par l'application pour laquelle elle a été créée. Dans les deux cas, l'utilisateur est l'instigateur de l'interface. Or, quand on énonce le terme de "génération", on pense à une **création** et non pas à une traduction. La plupart des sociétés qui commercialisent des UIMS l'ont compris ; elles appellent dorénavant leurs produits des "constructeurs d'interfaces"[53] et non plus des "générateurs d'interfaces". Nous verrons dans le Chapitre 6 que nous parlons vraiment de génération d'interface car l'utilisateur ne choisit pas tel type de fenêtre ou de menu, mais c'est à partir des spécifications du dialogue que le système choisit pour lui. Le système crée donc vraiment l'interface. On retrouve également cette notion de création dans des systèmes tels que MacIda [PETOUD 90], Siroco-Guide [NORMAND 92a], Génius [JANSSEN 93], ACE [JOHNSON 92] et UIDE [FOLEY 93b].
Malheureusement le terme "génération" est déjà employé dans des sens différents. Nous n'avons pas la prétention de le redéfinir ici, c'est pourquoi, dans ce chapitre, nous l'utiliserons avec toutes ses significations actuelles.
Nous allons voir tout d'abord les différents types de génération à partir des trois représentations d'une application interactive. La seconde partie donnera quant à elle quelques exemples de générateurs en fonction des modules générés.

[53] Le terme de "constructeurs d'interface" est malheureusement souvent remplacé dans les publicités ou lors des démonstrations de ces produits par le terme "interface builder" qui implique une confusion avec l'outil du même nom de la société NeXT.

2. Les différents types de génération automatique

Si l'on se reporte aux trois représentations d'une application interactive (Figure 3.1), on constate qu'il est possible d'avoir quatre types de génération (représentées par les flèches sur la figure) :

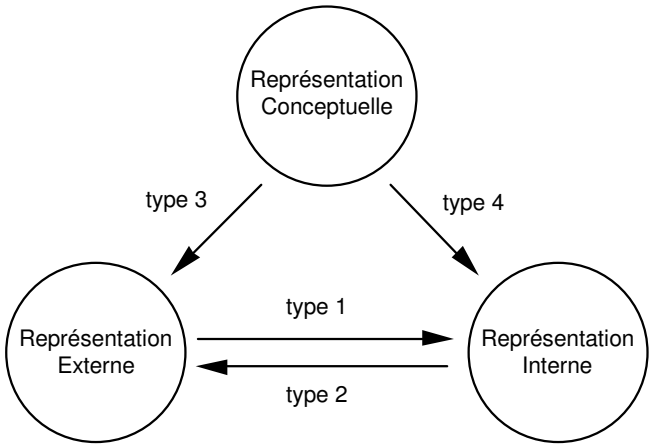


Figure 3.1 : Les quatre types de génération associées aux trois représentations d'une application interactive.

- *de la Représentation Externe vers la Représentation Interne*. C'est une des premières formes apparues en génération automatique, c'est également la plus facile à réaliser. On la trouve principalement dans les UIMS. Son travail consiste à générer du code à partir d'une maquette d'écran. L'utilisateur dessine pour cela ses fenêtres, ses menus, ses boutons, etc., de manière directe et interactive par l'intermédiaire d'un éditeur spécialisé. Pendant cette phase de maquettage, des "objets" sont créés et mis à jour, par exemple une fenêtre contient des boutons et des menus, et il faut donc créer ces liens entre les objets qui les représentent. Lors de la génération, ces objets sont traduits dans un code directement exploitable par l'application. Ils peuvent être exprimés sous forme de modules qui seront compilés puis liés à l'application ou bien encore sous forme de modules indépendants de l'application (par exemple les ressources sur Macintosh). La première solution permet en général de choisir un langage pour la génération (le plus souvent C ou C++ pour garder une portabilité optimale), la deuxième solution facilite surtout la maintenance et l'évolution de la représentation externe. Ainsi les couleurs des fenêtres, les polices de caractères utilisées, etc., sont autant de paramètres qu'on peut avoir à changer à tout instant sans être obligé de tout recompiler. Les environnements de travail tels que Windows utilisent cette solution. Toutes les applications sont basées sur une définition commune des ressources utilisées. Ainsi, changer la couleur de fond (définie par défaut pour les fenêtres) fait mettre à jour automatiquement toutes les fenêtres ouvertes, quelle que soit l'application qui les utilise.
  Ce type de génération est donc un apport intéressant à la création d'applications interactives. Il permet un gain de temps dans le maquettage et réduit les erreurs d'écriture de code. D'un autre côté il se révèle largement insuffisant pour créer complètement une application interactive.

- *de la Représentation Interne vers la Représentation Externe*. Ce type de génération est apparu à peu près en même temps que le précédent. Mickey [OLSEN 89] en est un bon exemple. Pour réaliser une génération de ce type, on part généralement du code de l'application ou si possible uniquement de celui de l'interface. Ce code contient en principe des structures de données qui décrivent les données utilisées et optionnellement la

représentation externe voulue. Par exemple le type Client contient des variables pour le numéro de client, le nom du client, son sexe, son numéro INSEE, etc. Une analyse de ce type permet de créer une fenêtre dédiée à la gestion des clients. Cette fenêtre peut regrouper les attributs énoncés dans le type Client. Par ailleurs une analyse plus poussée sur les attributs peut faire apparaître que la variable utilisée pour le sexe est d'un type énuméré (homme, femme). On peut donc lui associer une représentation externe sous la forme de boutons radio. Cette façon de procéder est utilisée par exemple par G. Singh [SINGH 89] qui peut transformer l'attribut Volume [0:10] en un potentiomètre gradué de 0 à 10 par pas de 1. On peut encore appliquer le même principe pour la création des menus ou des boutons. En extrayant du code les en-têtes de procédures ou les appels de fonctions, on peut créer facilement des menus [OLSEN 89]. Par contre le classement des opérations dans ces menus, de même que le classement des menus eux-mêmes, ne peut pas être pris en compte de manière aussi simple par la génération. Nous verrons dans la deuxième partie de ce rapport quelle est la solution que nous proposons.
La génération de l'interne vers l'externe est donc intéressante, mais l'avance technologique tend à la rendre de plus en plus caduque. On préfère passer directement des spécifications à la représentation externe sans être obligé d'écrire du code intermédiaire.

- *de la Représentation Conceptuelle vers la Représentation Externe*. La représentation conceptuelle permet de définir la syntaxe et la sémantique de l'application. Il était donc intéressant de pouvoir passer directement des spécifications conceptuelles à la représentation externe (et interne comme nous le verrons juste après). Au niveau de la syntaxe, il devient alors possible d'exprimer graphiquement des règles telles que celles utilisées pour la saisie des dates, des numéros INSEE, etc. Dans la figure 3.2, la boîte de dialogue a été créée à partir du type INSEE qui est constitué de deux listes, une de treize chiffres et une de deux chiffres. Les zones de saisie sont donc au nombre de deux (treize chiffres et deux chiffres). La boîte de dialogue ne permettra donc que la frappe de caractères numériques.

  Au niveau de la sémantique, la génération est plus difficile. Ainsi pour une saisie de date, il faut spécifier les règles de saisie, par exemple 28 jours pour février sauf si l'année est bissextile. De telles règles sémantiques doivent être en général traitées par l'application qui, à son tour, est chargée de mettre à jour la représentation externe. Par exemple la saisie de Monsieur pour un client (Figure 3.3) rend invalide la zone de saisie du nom de l'époux.

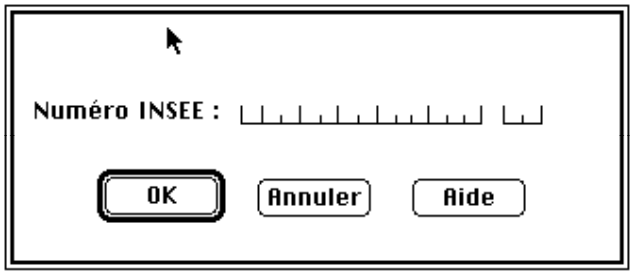


Figure 3.2 : Exemple de génération de boîte de dialogue à partir de spécifications conceptuelles syntaxiques.

Ce type de gestion de la représentation externe n'est pas simple à mettre en œuvre automatiquement. Pour la figure 3.3, il faudrait par exemple comparer la structure du type Client pour faire apparaître une différence de nombre de champs en fonction de la variable associée à Monsieur, Madame, Mademoiselle.

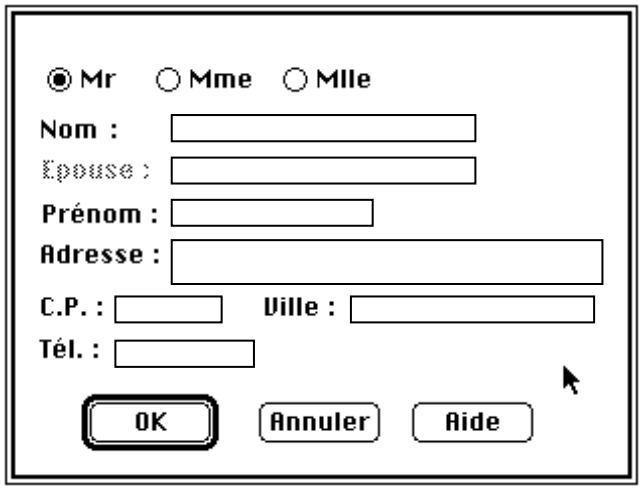


Figure 3.3 : Exemple de gestion automatique de la sémantique à partir des spécifications.

Il existe cependant des travaux qui permettent de générer la représentation externe à partir des spécifications conceptuelles. Citons par exemple Siroco-Guide [NORMAND 92a] [NORMAND 92b], MacIda [PETOUD 89] [PETOUD 90], Génius [JANSSEN 93], ACE [JOHNSON 92], ainsi que [FOLEY 92] [FOLEY 93b] ou dans un genre différent [FRANCHI 89] qui génère des éditeurs graphiques basés sur un langage (par exemple un éditeur de formules mathématiques) à partir de la description de ce langage.
L'avantage de cette catégorie de génération est d'éliminer l'étape intermédiaire qui consistait à traduire les spécifications dans un langage afin d'en extraire les composants de l'interface ainsi que sa gestion. L'inconvénient majeur à l'heure actuelle est la difficulté que l'on éprouve à écrire les spécifications de telle sorte qu'elles soient à la fois lisibles pour celui qui les écrits et suffisamment puissantes et manipulables pour pouvoir générer le maximum de la partie externe et de la gestion de l'interface. Nous verrons dans les Chapitres 5 et 6 que la méthode Diane+ permet de spécifier, de manière lisible (à la fois pour l'informaticien et pour l'utilisateur), la répartition des tâches entre l'homme et la machine. Ces spécifications sont par ailleurs suffisamment puissantes pour donner naissance à la Représentation Externe qui est chargée de les gérer.

- *de la Représentation Conceptuelle vers la Représentation Interne*. Nous venons de voir que les spécifications conceptuelles étaient très utiles pour la génération de l'interface, mais qu'elles n'étaient pas faciles à mettre en œuvre. Nous retrouvons ici à peu près le même problème. Le terme Représentation Interne est synonyme de structures d'informations (au sens large). Ces structures sont par exemple des en-têtes d'opérations, des classes d'objets, des modules de gestion de dialogue, etc. Ce type de génération est difficile à réaliser car il fait intervenir la sémantique de l'application. Ainsi la génération de contrôleur de dialogue fait appel entre autres à la création d'en-têtes de procédures et à la sémantique de ces procédures. Alors que l'on peut extraire facilement les premiers des spécifications fonctionnelles des données à manipuler, la sémantique fait intervenir de nombreux paramètres tels que des contraintes d'intégrité, des règles de gestion, etc. L'approche objet permet de pallier certains de ces problèmes. Étant donné que l'on regroupe les traitements en fonction des données, la sémantique se trouve plus ou moins confinée à l'intérieur de l'objet auquel elle se rapporte. Il devient alors plus facile de générer l'application sous forme de classes d'objets avec des méthodes de gestion qui leur sont propres [NORMAND 92a].
  Le passage de Diane à Diane+ intègre ce type de génération. Ainsi, à partir des spécifications du dialogue, nous générons les objets qui représentent les concepts exposés

dans les spécifications (opérations, procédures, et buts principalement) et qui seront chargés de la gestion de l'application (dynamique et aide).

Conclusion

Il existe donc quatre grands types de génération automatique. Si on se reporte à la figure 3.1, on constate qu'il reste encore deux types possibles qui sont de la Représentation Externe vers la représentation Conceptuelle et de la Représentation Interne vers la Représentation Conceptuelle. Ces deux générations ne sont pas réalisées à l'heure actuelle. La première est quasiment du domaine de l'utopie car elle signifie qu'il est possible de déduire par exemple de la sémantique à partir d'une maquette. Cela est peu probable puisqu'en général la Représentation Externe n'est qu'une vue de la Représentation Conceptuelle. Cette vue dépend du contexte, de l'utilisateur, etc. Il est difficile d'imaginer, sans aucune connaissance préalable du système, des liens entre des entités graphiques (par exemple entre un bouton Ajouter et une list box).
La seconde génération se rapproche du "reverse-engineering" puisqu'elle consiste à déduire de la sémantique et du conceptuel à partir du code de l'application. Des recherches sont en cours sur ce domaine et ont déjà donné naissance à quelques produits commerciaux, mais la plupart du temps ils ne concernent que des données ou des traitements. A notre connaissance aucun produit ne fait du "reverse-engineering" pour les IHM.

3. Les modules générés

A chaque type de génération énoncée précédemment peut correspondre un ou plusieurs modules générés. La figure 3.4 représente de manière générale le fonctionnement d'un générateur d'application interactive. A partir de spécifications diverses (conceptuelles, fonctionnelles, etc.) qui peuvent se présenter sous une forme quelconque (graphe, langage, grammaire, etc.), le générateur produit du code pour un ou plusieurs modules de l'application (la présentation, le contrôleur de dialogue, le noyau fonctionnel ou les modules chargés des liaisons entre ces parties). Il existe des générateurs qui couvrent toute l'application alors que d'autres sont spécialisés dans une partie (par exemple la présentation pour les UIMS). Les tableaux 1 et 2 à la fin de cette partie regroupent quelques produits classés suivant les générations qu'ils procurent. Nous allons à présent détailler ces différentes générations.

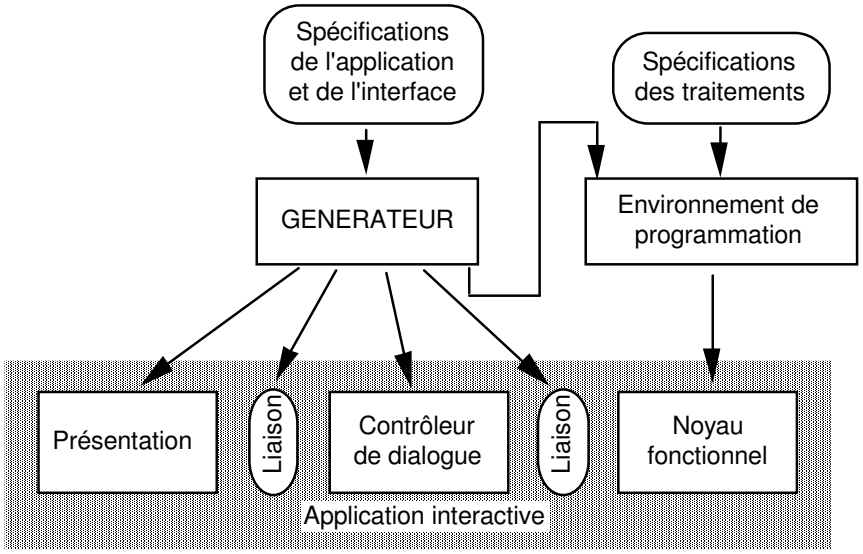


Figure 3.4 : Fonctionnement général d'un générateur d'application interactive (inspiré de [NANARD 90])

3.1. Génération de la présentation

Cette génération est la plus répandue et la plus ancienne. La présentation correspond à la partie visible de l'application et il est possible de l'obtenir de deux façons : par la Représentation Conceptuelle (par exemple en définissant les concepts manipulés par l'application [NORMAND 92a] ou la sémantique des données [JOHNSON 92]) ou par la Représentation Interne (par exemple à partir de la description d'une base de données [PETOUD 90] [JANSSEN 93]). Quel que soit le choix, le résultat est la plupart du temps un ensemble d'écrans ou plus exactement de maquettes d'écrans. Ces écrans peuvent être de natures différentes : fenêtres textuelles, fenêtres graphiques, icônes, menus, etc. Leur représentation à l'écran varie d'un environnement à l'autre, par exemple les menus sont horizontaux et solidaires dans Windows alors qu'ils sont verticaux et sécables dans l'environnement NeXTStep. Pourtant ils possèdent des caractéristiques communes qui sont de regrouper des commandes suivant une certaine logique[54] et de pouvoir déclencher des actions à partir du choix d'une des commandes.

Les toolbox, les toolkits et les UIMS sont des exemples de générateurs de présentation à partir des Représentations Externe et Interne. A présent les derniers sont les plus utilisés grâce à leur facilité d'utilisation, leur rapidité de traitement et leur possibilité de générer des interfaces dans des environnements différents. Ce type d'UIMS a l'énorme avantage de permettre le développement d'interface pour un environnement alors que l'on travaille sur un environnement différent (par exemple développer une interface Motif sous Dos).

Il existe beaucoup de produits qui génèrent la présentation. Nous citerons ici cinq exemples :

- ACE [JOHNSON 92] génère les widgets en fonction de la sémantique des données manipulées. Pour chaque donnée, ACE propose un ou plusieurs widgets en fonction du type de la donnée et du critère de choix. Pour un choix de couleurs par exemple, le critère de choix sera "un parmi N". Les widgets proposés seront par conséquent des boutons radio, une listbox et une combo box.

[54] Nous verrons plus loin que cette logique est une logique de fonctionnement et non pas une logique d'utilisation.

- J. Foley [FOLEY 92] [FOLEY 93b] propose un outil générant la présentation indépendamment de l'environnement. Cet outil est basé sur un ensemble de règles ergonomiques telles que[55] : "Pour représenter un entier non modifiable par l'utilisateur, dont le domaine de valeurs est connu et dont la précision d'affichage est peu importante, on utilisera un widget de type jauge".
- MacIda [PETOUD 89] [PETOUD 90] permet d'obtenir une Représentation Externe à partir des spécifications conceptuelles des données à manipuler. Cette Représentation Externe est obtenue depuis un schéma de type MCD dont elle reprend la disposition spatiale (le travail de C. Janssen [JANSSEN 93] suit un principe similaire de génération). L'interface obtenue peut ensuite être remaniée afin d'améliorer la présentation vis-à-vis de l'utilisateur.
- Paul Franchi-Zannettacci [FRANCHI 89] a développé un générateur d'éditeurs graphiques basés sur un langage. Pour cela on décrit la syntaxe de ce langage grâce à une grammaire à attributs et par l'intermédiaire de boîtes qui donnent les contraintes graphiques. On utilise également un langage spécialisé GSL qui permet de définir directement les contraintes graphiques sur les boîtes.
- Centaur [BORRAS 88] est un environnement générique interactif. A partir de la description syntaxique et sémantique d'un langage, il produit un environnement spécifique à ce langage. Cet environnement est constitué entre autres d'un éditeur de structures du langage, d'un interpréteur et d'un déboggueur.

3.2. Génération du contrôleur de dialogue

Après que les outils aient résolu le problème de la génération de la face externe des applications, l'étape suivante a consisté à améliorer la production de la face interne. Celle-ci passe par la production d'une part du module chargé du dialogue homme-machine, d'autre part du noyau fonctionnel. Nous nous intéressons ici au premier cas, le § 3.3 au second.

Produire une maquette n'est pas très bénéfique si l'on est obligé de programmer soi-même sa gestion par l'application. Il apparaît donc évident d'essayer de faire effectuer ce travail par la machine. Le principe de cette production est similaire à celui de la présentation (Figure 3.4). On fournit des spécifications au générateur qui s'occupe alors de les traduire sous forme de code ou dans une représentation qui sera reprise par d'autres modules. Les spécifications peuvent prendre plusieurs formes : langages graphiques [ALTY 89], Réseaux de Petri [PETRI 62], descriptions textuelles [FRANCHI 89]... Elles peuvent également prendre en compte ou non l'utilisateur, les enchaînements d'écrans, la concurrence, etc. Quoiqu'il en soit, le générateur produit la plupart du temps un module spécialisé dans la gestion du dialogue. Les spécifications peuvent être de deux natures différentes : conceptuelles ou internes. Dans le premier cas on trouve par exemple la description des commandes à implémenter [SINGH 89] ou des concepts à manipuler [NORMAND 92a], dans le deuxième on trouve des produits qui permettent de décrire les enchaînements d'écrans en fonction des interactions [ILOG] [GIESKENS 92] [JANSSEN 93] ou qui transforme une description de dialogue dans un autre langage (par exemple une grammaire dans un langage de programmation [YAP 88]).

La génération du contrôleur de dialogue n'est pas une fin en soi. Il paraît judicieux de générer à la fois le contrôleur de dialogue et la présentation puisqu'ils sont en étroite liaison. En effet, les interactions de l'utilisateur passe par la présentation avant d'atteindre le contrôleur. D'un autre côté le contrôleur doit mettre à jour la présentation en fonction de l'état de l'application. De leur côté les UIMS ont commencé à répondre à ce besoin. Lorsqu'on construit une interface avec ce

[55] La règle est écrite ici dans un style "langage naturel " qui n'est pas celui du système en question.

type d'outils, il est possible d'associer à chaque élément de l'interface un ou plusieurs traitements en fonction des entrées utilisateur et plus rarement du contexte (ou de l'état) de l'application. Certains UIMS permettent également ce genre de traitements entre les écrans. On peut dire par exemple : pour un double-clic dans cette zone, on effectue le traitement X_1 si la condition C_1 est vraie, X_2 si C_2 est vraie, etc. De la même façon, il est possible de dire qu'on passe de l'écran E_1 à l'écran E_2 lorsqu'on clique sur le bouton OK par exemple. Ces spécifications sont ensuite traduites sous une forme manipulable par l'application cible. Malheureusement c'est l'humain qui a dû écrire ces spécifications et donc on augmente d'une part le temps de développement de l'application et d'autre part le risque d'erreur lors de l'écriture du code. Une solution efficace consiste à générer ces spécifications à partir de spécifications d'ordre général. Nous verrons dans les Chapitres 5 et 6 que nous répondons en partie à ce problème grâce aux spécifications Diane+ qui nous permettent de gérer automatiquement la dynamique de l'application. Des systèmes tels que Siroco-Guide [NORMAND 92a] utilisent un principe similaire.

3.3. Génération du noyau fonctionnel

Le noyau fonctionnel est composé à la fois des traitements à réaliser et des données manipulées par ces traitements. Sa génération suppose donc la génération de ces deux composants.

3.3.1. Génération des traitements

La partie traitement du noyau fonctionnel est certainement le seul des modules d'une application interactive qu'on ne pourra pas générer entièrement avant longtemps. Alors qu'il est possible de donner des spécifications pour les autres modules (par exemple les paramètres d'une commande ou les caractéristiques des enchaînements d'écrans), il est pour l'instant impossible de générer les traitements que l'on veut faire réaliser par l'application. Comment en effet décrire des traitements qui sont de nature purement algorithmique tels que tracer un cercle, justifier un texte entre deux marges, etc… ? Il est toujours possible, bien évidemment, de décomposer un traitement en traitements élémentaires, mais ces derniers sont souvent de nature procédurale. Par exemple pour enregistrer un client, il faut enregistrer son numéro, son nom, son adresse, etc., mais l'enregistrement des données est toujours fait par une procédure qui a due être écrite à un moment ou un autre. Il reste toujours possible d'imaginer une génération des traitements en fonction des données manipulées, par exemple si le type Client est un type qui contient des variables Numéro, Nom, Adresse, etc., on peut imaginer que le système créera automatiquement des procédures pour remplir chacune de ces variables. Un problème apparaît alors : les traitements générés par la machine peuvent ne pas être désirés par l'utilisateur ou bien encore ne pas correspondre pas à l'attente de l'utilisateur. Dans les deux cas l'humain doit intervenir. Par ailleurs il est probable qu'écrire les spécifications détaillées des traitements à générer soit aussi long et complexe que les écrire directement (par exemple en D.A.O pour une procédure de visualisation avec la gestion des surfaces cachées). L'idéal serait de disposer d'une bibliothèque de procédures qui seraient complètement réutilisables (par exemple des bibliothèques de tri, d'outils de dessins, etc.) et qui pourrait fonctionner pour tout type de données (par exemple une procédure de tri pourrait aussi bien trier des noms, des graphes, des enregistrements dans une base de données, etc.). Cela est déjà réalisé en partie grâce aux langages objet, mais la standardisation est encore très loin.

A l'heure actuelle, pour réaliser un noyau fonctionnel on doit donc disposer au moins des spécifications de l'application. Celles-ci sont traduites dans un langage de programmation et le code résultant peut être groupé avec celui issu du générateur (Figure 3.4) à des fins de compilation ou de linkage. Le résultat final est un module qui contient dans la majorité des cas un ensemble de primitives à la disposition de l'utilisateur.

3.3.2. Génération des données

Cette génération ne concerne que la structure des données et non pas leurs valeurs. On cherche à créer par exemple une structure de base de données ou un ensemble de classes d'objets. On utilise pour cela des spécifications diverses, par exemple le type des attributs qui apparaissent dans un MCD ou bien encore des spécifications d'ordre conceptuel plus ou moins élevé. Ainsi Siroco-Guide [NORMAND 92a] décrit les concepts manipulés par l'application en donnant leur nom, les attributs qui les décrivent et les opérations qu'ils sont capables d'exécuter. Ces spécifications sont ensuite traduites automatiquement pour donner naissance à des classes d'objets (structures de données). Un autre exemple de génération des données est MacIda [PETOUD 90] qui génère quant à lui un type Pascal par attribut.

Il existe également des systèmes qui génèrent à la fois les structures de données et la présentation associée à ces structures. C'est le cas par exemple de Tramis [TRAMIS 92] qui est capable de produire une présentation pour les objets qu'il manipule. A partir d'un MCD, on groupe les entités et les relations dans des ensembles qui sont représentatifs pour l'utilisateur ; ces groupes sont appelés des "objets naturels" [BRES 90]. Puisque ces objets naturels comportent des entités et des relations parfaitement identifiées, il est possible de générer des fenêtres pour leur manipulation directe (saisie, suppression, modification, etc.). Par contre les traitements issus de l'application ne peuvent pas être générés de cette façon, mais nous verrons dans le chapitre 5 que l'on peut utiliser les objets naturels (et leur présentation) conjointement avec les traitements à réaliser.

Les données ne sont pas les seuls concepts que l'on peut générer dans le noyau fonctionnel. On peut avoir besoin également de concepts qui manipulent les données ou les traitements et qui ne font pas partie de ces deux catégories. Si l'on reprend l'exemple de Siroco-Guide, celui-ci décrit des espaces de travail ("Workspace") et des vues sur ces espaces de travail ("Perspectives") d'une manière semblable aux données. Les espaces de travail permettent de grouper les traitements nécessaires à la réalisation d'une tâche (par exemple envoyer un message dans une messagerie électronique) et les perspectives définissent une vue restrictive sur les données et les opérations. Ces deux concepts sont présents dans le noyau fonctionnel au même titre que les données et ils sont générés en même temps que ceux-ci.

3.4. Générations des modules de liaison

Une grande majorité des produits actuels ne contiennent pas de tels modules. La plupart du temps en effet, ils sont directement intégrés dans l'un ou l'autre des trois autres modules (le plus souvent le contrôleur de dialogue). Le rôle de ces modules, quand ils existent, est d'amortir les échanges qui peuvent se produire entre les trois modules. Par exemple ils sont chargés de convertir des formats différents afin de les rendre compatibles ou bien encore de traduire des événements d'un niveau d'abstraction à un autre. A notre connaissance, il n'existe pas à l'heure actuelle de générateur produisant des modules de liaison de manière significative comme pour la présentation par exemple. Ces modules sont "créés" de deux façons différentes : soit les générateurs les intègrent directement au reste du système, soit le concepteur les code directement.

Le problème de la génération de ces modules est due en fait au caractère ad hoc qu'ils possèdent. En effet leurs rôles n'étant pas clairement définis, il est difficile d'extraire une méthode de génération. Certains systèmes les utilisent comme interface de traduction de formats, d'autres comme traducteur de concepts, etc. Pour que leur génération devienne possible de manière générale, il faudra d'abord définir leurs conditions d'existence et leur sémantique de la même

façon que cela a été fait pour les trois autres modules (présentation, contrôleur de dialogue et noyau fonctionnel).

4. Récapitulatif

Remarques :

- les tableaux présentés regroupent quelques-uns des systèmes les plus connus. Nous les avons classés en fonction de certaines caractéristiques, mais dans certains cas les systèmes ne répondent qu'en partie à l'une ou l'autre d'entre elles. Par exemple Siroco-Guide ne génère pas complètement le noyau fonctionnel mais seulement des classes vides correspondant aux différents concepts manipulés par l'application (données, espaces de travail, perpectives, session, etc.).
- le premier tableau est issu de [NANARD 90]. Il classe quelques systèmes en fonction du type de spécification utilisé pour la génération. Trois types généraux sont présentés, ce qui implique l'inhibition de certaines sous-catégories ; par exemple dans les spécifications graphiques, on a regroupé des systèmes qui utilisent directement des widgets (par exemple Interface Builder) et d'autres qui les construisent de toute pièce par démonstration (Péridot).
- le second tableau reprend la plupart des systèmes énumérés précédemment et les classe en fonction des modules qu'ils génèrent. Étant donné que certains systèmes génèrent plusieurs modules, il est donc normal de les retrouver plusieurs fois dans le tableau.

Type de spécification	Formalisme utilisé	Systèmes
Spécification basée sur un langage	*Réseau de menus*	Tiger [KASIK 82]
	Diagrammes de transitions	[JACOB 85] Rapid/Use [WASSERMAN 85]
	Grammaires hors contexte	Syngraph [OLSEN 83]
	Langages à événements	Algae [FLECCHIA 87] Sassafras [HILL 86]
	Langage déclaratif	Squeak [CARDELLI 85] Cousin [HAYES 85] Domain-Dialog [SCHULERT 85] GWUIMS [SIBERT 86] Serpent [BASS 90] Tube [HERRMANN 89]
Spécification graphique		Menulay [BUXTON 83] Trillium [HENDERSON 86] Mike [OLSEN 86] Graffiti [KARSENTY 87] Péridot [MYERS 88b] Hypercard [HARVEY 88] DialogEditor [CARDELLI 87] Interface Builder [WEBSTER 89]
Spécification fonctionnelle		UIDE [FOLEY 88] Mike [OLSEN 86] Jade [VANDER 90] ITS [WIECHA 90]

Tableau 1 : Classification de systèmes de production d'interfaces en fonction du type de spécification employé.

Modules générés	Systèmes
Présentation	MacIda [PETOUD 89] [PETOUD 90] [OLSEN 89] UofA [SINGH 91] [SINGH 89] [FRANCHI 89] Centaur [BORRAS 88] Péridot [MYERS 88b] Graffiti [KARSENTY 87] [BERRADA 92] [PINGAND 89] [JANSSEN 93] [FOLEY 92] [JOHNSON 92] ITS [WIECHA 90] Jade [VANDER 90] UIDE [FOLEY 88] Don [KIM 92] Chisel [SINGH 92]
Contrôle du dialogue	MacIda [PETOUD 89] [PETOUD 90] Scenarioo [ROUDAUD 90] [YAP 88] [SINGH 89] [SCHIELE 90] Gradient [ALTY 89] Dost [DEWAN 87] [HAGIYA 90] [DUVAL 91] [FRANCHI 89] ApplBuilder [GRØNBÆK 91] Centaur [BORRAS 88] Algae [FLECCHIA 87] Siroco-Guide [NORMAND 92a, 92b] [BERRADA 92] [PINGAND 89] [GIESKENS 92]
Noyau fonctionnel	Siroco-Guide [NORMAND 92a, 92b] [BOUSSE 91] [BOUSSE 89]

Tableau 2 : Classification de systèmes de production d'interfaces en fonction des modules générés.

5. Conclusion

La génération automatique des applications interactives possède actuellement les avantages et les inconvénients suivants :

- avantages :
  - elle permet un gain de temps pour l'écriture et le déboggage, ainsi que pour la maintenance,
  - elle améliore la lisibilité du code produit,
  - elle augmente l'utilisabilité des systèmes.

- inconvénients :
    - l'humain doit souvent reprendre le résultat de la génération pour l'affiner et l'adapter exactement à ses besoins. Ceci est dû à l'incapacité de prendre totalement en compte de manière automatique les demandes des concepteurs. On parvient à résoudre par exemple le choix de la disposition spatiale des widgets, mais on ne peut pas encore générer automatiquement les traitements.
    - elle couvre plus ou moins complètement l'application suivant son domaine. On peut dire que la couverture de l'application par la génération est inversement proportionnelle à la latitude décisionnelle de l'application. Pour une application à faible latitude décisionnelle, on peut générer le noyau fonctionnel ainsi que la présentation et le contrôleur de dialogue étant donné que l'on est capable de décrire à l'avance presque complètement les actions de l'utilisateur et le comportement de l'application. Par contre pour une application à forte latitude décisionnelle, on ne peut générer que le noyau fonctionnel car le contrôle du dialogue est très restreint voire même inexistant. Les parties Présentation et Contrôleur de Dialogue sont donc à la charge du programmeur.

La génération automatique d'une application complète à partir de sa représentation conceptuelle est donc la prochaine étape à réaliser dans le domaine des IHM. Pour l'instant les travaux s'orientent vers une génération plus efficace de la présentation et du contrôleur de dialogue en utilisant pour la plupart des spécifications fonctionnelles ou conceptuelles de l'application (données et/ou traitements). Notre travail s'inscrit dans ce cadre de recherche. Les spécifications que nous utilisons nous permettent la génération d'une partie de la Présentation ainsi que la génération totale du Contrôleur de Dialogue et de l'aide d'utilisation, le modèle de données OPAC nous fournit quant à lui une partie du noyau fonctionnel et l'autre partie de la Présentation.

PARTIE II

Chapitre 4
Objectifs de Génération et de Gestion Automatique à partir de Spécifications

1. Introduction

Les chapitres précédents ont montré l'évolution des IHM sous différents angles : historique, modèles et formalismes. Nous avons vu qu'il reste encore de nombreux problèmes à régler (modèles de conception imprécis, dialogue homme-machine plus ou moins bien géré, etc.). La suite de ce rapport tentera d'apporter des éléments de réponse sur certains d'entre eux. Nous voulons, dans ce chapitre, situer notre travail par rapport à ceux présentés précédemment en exposant rapidement les différents objectifs que nous nous sommes fixés.

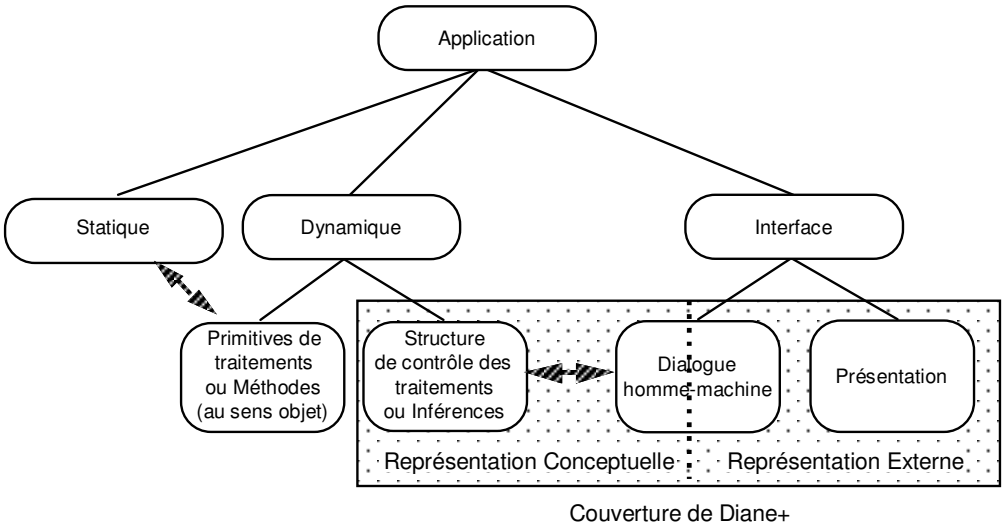


Figure 4.1 : Découpage conceptuel d'une application

Un découpage conceptuel d'une application interactive (Figure 4.1) montre les trois parties essentielles à l'implémentation d'un telle application : la *statique* qui est constituée principalement des données et qui est gérée par les primitives de traitements, la *dynamique* qui se répartit sur les contrôles sur et entre les traitements, enfin l'*interface* qui est basée sur le dialogue homme-machine et sur la présentation vis-à-vis de l'utilisateur. Il faut noter que le dialogue homme-machine n'est pas totalement indépendant des autres parties. Les contrôles entre traitements sont en effet en étroite relation avec le dialogue homme-machine et ces deux modules doivent s'échanger des informations pour un fonctionnement correct. Or, bien qu'en général la partie statique et la partie "traitements" soient clairement spécifiées et implémentées, le contrôle sur les traitements et le dialogue homme-machine bénéficient toujours d'un certain flou. Suivant les modèles et les formalismes, on peut les trouver totalement indépendants ou en relation (déportation de certains rôles d'un côté ou de l'autre) ou bien encore imbriqués l'un dans l'autre.

C'est pourquoi notre travail a trois objectifs principaux :

- *une spécification de la répartition du dialogue*[56] *entre l'homme et la machine* qui soit simple, claire et précise, d'abord pour l'utilisateur et ensuite pour l'informaticien,
- *la génération du contrôleur de dialogue et de l'interface* à partir de ces spécifications,
- *une gestion automatique de l'interface générée* ainsi que de *l'aide* lors des sessions de travail.

2. Objectifs de Diane+

2.1. Démarche de conception

La figure 4.2 représente notre démarche de conception qui repose sur la méthode Diane+ issue de Diane [BARTHET 88]. Dans un premier temps nous spécifions les buts et les procédures[57] en nous basant sur les opérations[58] (si elles existent, sinon nous les spécifions également) ainsi que sur la description des données (supposée existante). Puis nous générons le code des opérations, ainsi que celui des procédures, des buts et de l'interface. Nous générons également les règles d'enchaînement qui représentent les précédences entre les opérations. Ces divers éléments sont ensuite utilisés par le contrôleur de dialogue qui se charge de gérer la représentation externe ainsi que les traitements et l'aide d'utilisation.

2.2. Spécifications du dialogue homme-machine avec la méthode Diane+

Des méthodes, telles que Merise ou SADT, facilitent l'informatisation des traitements, mais elles ne permettent pas une répartition claire des traitements entre l'homme et la machine. D'autres méthodes, telles que Axial et Use, permettent de prévoir la dynamique du dialogue en précisant les enchaînements des traitements ainsi que l'origine des interactions (écrans, clavier, mémoire de masse, etc.). Malheureusement l'inconvénient majeur de ces méthodes est la rigidité des dialogues qu'elles permettent de manipuler. Les méthodes orienté-objet actuelles, telles que OOA [COAD 90], ont l'avantage de créer des applications interactives à partir des objets manipulés par ces applications. Le concept objet facilite, de par la notion d'autonomie, la gestion d'un dialogue homme-machine moins rigide. Nous verrons plus précisément dans le chapitre suivant ces différentes méthodes. De façon générale on peut dire que les méthodes actuelles ne permettent pas un dialogue homme-machine très convivial ou au contraire sont tellement vagues que la gestion du dialogue est entièrement à la charge du programmeur ou de l'utilisateur. Nous rappelons que nous travaillons uniquement dans le cadre d'applications de type pré-planifiées [RAS 83] et donc que l'utilisateur ne peut pas être entièrement libre dans le choix de ses actions contrairement aux applications de type créatif.

---

[56] Le terme dialogue est à prendre dans le sens que nous lui avons donné au chapitre 2, c'est-à-dire l'ensemble des interactions de l'homme sur la machine.

[57] Une procédure est la description formelle et détaillée d'une tâche permettant de décrire l'enchaînement des différentes opérations [BARTHET 88].

[58] Une opération est un ensemble de transactions qui peuvent être exécutées consécutivement et sans attente d'événement externe au couple homme-machine [BARTHET 88].

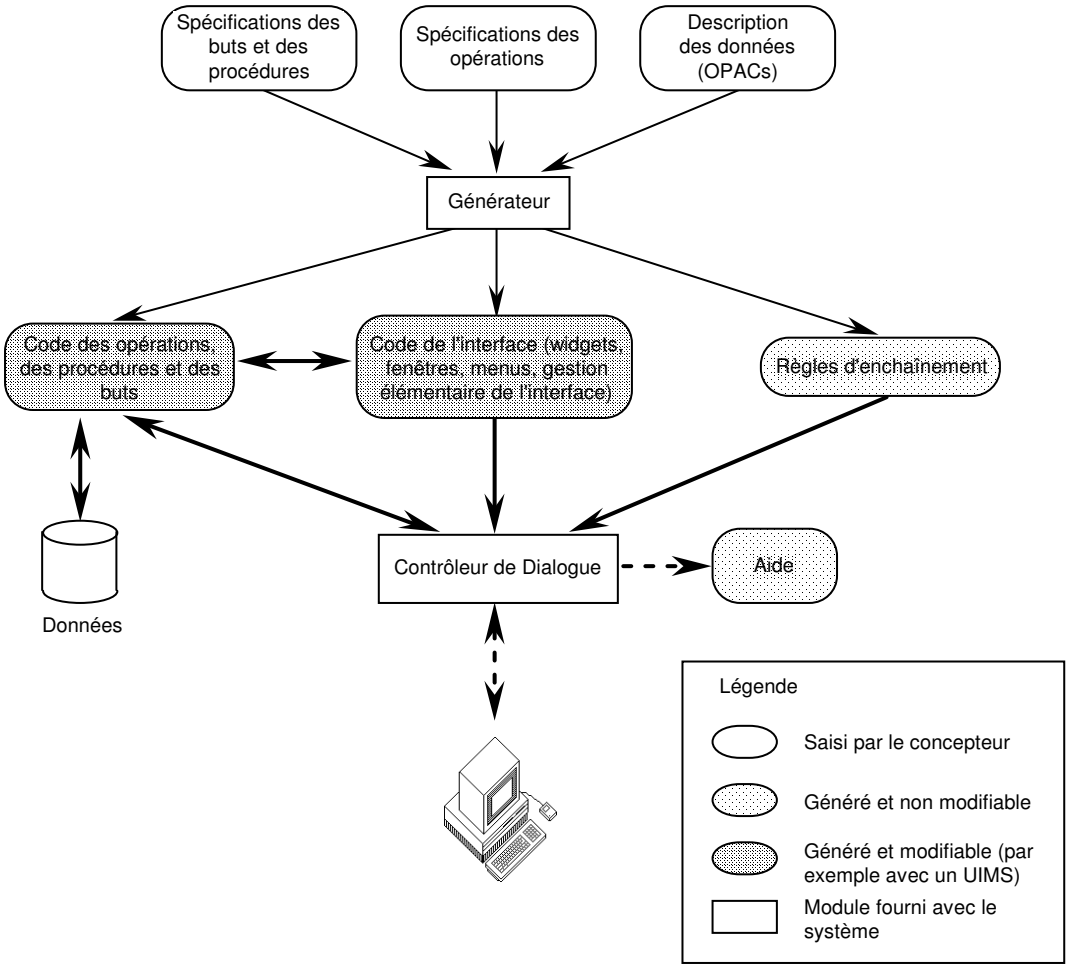


Figure 4.2 : Démarche de conception avec Diane+

Diane+ intervient après que l'on ait défini les données et les traitements, et que l'on se pose le problème de la répartition des tâches et des contrôles entre l'homme et la machine[59]. Elle permet de regrouper les opérations nécessaires à un poste de travail en fonction des traitements à réaliser. Elle intègre donc l'utilisateur dans le dialogue ce que ne font pas la plupart des autres méthodes. Basée sur la planification hiérarchique, elle structure ces opérations sous forme de procédures en fonction des buts à atteindre et du type de l'utilisateur (compétence, fréquence d'utilisation, niveau de confidentialité, etc.). Pour gérer cette répartition, trois types de procédures sont utilisées :

- *les procédures prévues* qui sont le reflet d'une utilisation cohérente des opérations disponibles pour la réalisation d'un but. Elles sont surtout utilisées pour l'apprentissage et pour l'aide. De par leur caractère logique, elles ne sont pas nécessairement optimisées (par exemple pour des experts du domaine ou des utilisateurs expérimentés).

- *les procédures effectives* qui sont les enchaînements tels qu'ils sont réalisés effectivement sur le site réel ou lors de simulations. On peut différencier deux types de procédures effectives : d'une part celles qui sont réalisées avant le développement du nouveau système informatique, et d'autre part celles qui sont réalisées après l'implémentation. Les premières permettent d'analyser les habitudes des utilisateurs ainsi que les failles de l'existant, les

[59] A ce titre, Diane se rapproche du travail de K. Y. Lim [LIM 92] qui a utilisé une approche similaire avec la méthode Jackson.

secondes sont le reflet de l'utilisabilité du nouveau système et permettent ainsi son évaluation.

- *les procédures minimales* qui correspondent au noyau dur du dialogue homme-machine. Elles sont extraites des procédures prévues et effectives et contiennent d'une part l'ensemble des opérations nécessaires à la réalisation d'un but (ainsi que celles susceptibles d'être utilisées) et d'autre part le contrôle minimal obligatoire de la machine sur l'homme.

Toutes ces procédures contiennent également les enchaînements obligatoires entre les opérations, ainsi que des propriétés sur les opérations (les déclencheurs, le mode d'exécution, etc.). Diane permet de décrire la latitude décisionnelle de l'utilisateur en fonction de son niveau et par l'intermédiaire des procédures et des opérations ainsi que par leurs enchaînements. Par ailleurs, cette latitude décisionnelle possède une borne supérieure qui est la procédure minimale, permettant ainsi un contrôle sur la cohérence des enchaînements.

Avec Diane+, il existe une procédure minimale par poste de travail pour lequel peuvent exister plusieurs procédures prévues et plusieurs procédures effectives, créées en fonction des types d'utilisateurs pour ce poste. Cela signifie que pour un même but, on peut avoir plusieurs ensembles de ce type s'il existe plusieurs postes de travail qui doivent réaliser ce but. Les procédures minimales et prévues sont implémentées une fois pour toutes ; elles sont invariantes et l'utilisateur ne peut en aucun cas les modifier. Par contre, les procédures effectives sont dynamiques puisqu'elles sont le reflet des interactions des utilisateurs. Lors de l'implémentation sur site, le nouveau système peut déjà contenir les procédures effectives correspondant à l'utilisation de l'ancien système. Puis, au fur et à mesure de l'utilisation du nouveau système, les usagers peuvent créer eux-mêmes leurs procédures effectives ou bien encore le système peut les créer à leur place par exemple à partir de la fréquence d'utilisation.

Notons pour finir que la description des procédures correspond à la représentation conceptuelle de l'application au niveau du dialogue alors que la description des écrans ainsi que leur dynamique correspond à la représentation externe de l'application (Figure 4.1).

2.3. Génération du dialogue

Il n'existe pas à l'heure actuelle beaucoup de méthodes de conception qui permettent d'aboutir à la génération automatique du contrôleur de dialogue. Par contre on constate l'apparition de nombreux outils qui répondent à cette demande (MacIda [PETOUD 90], Siroco-Guide [NORMAND 92a], etc.). Notre travail n'est donc pas fondamentalement nouveau bien qu'il s'appuie sur une méthode, ce que les autres outils ne font pas habituellement. L'avantage de Diane est de ne pas imposer de méthode d'implémentation particulière. Elle ne connaît donc pas la façon dont sont représentés en machine les données et les traitements ce qui signifie qu'on peut l'appliquer à tous les types d'implémentation. Il est évident que la génération du contrôleur de dialogue dépend du type d'implémentation choisi, mais le fonctionnement de ce dernier reste le même quel que soit ce type.

La connaissance préalable des procédures prévues et minimales (parfois même de procédures effectives) nous permet, grâce à Diane+, leur génération sous une forme directement exploitable. Plusieurs démarches sont disponibles. Pour notre part, nous nous proposons d'implémenter les procédures Diane+ par l'intermédiaire d'objets représentant les opérations et les données ainsi que par l'intermédiaire de règles d'enchaînement. Ces dernières sont manipulées par le contrôleur de dialogue qui devra donc fonctionner comme un moteur d'inférence. Le code généré contient à la fois les en-têtes des opérations avec leurs paramètres ainsi que tout ce qui est nécessaire à la

dynamique du système (contraintes de déclenchement, nature des opérations, enchaînements obligatoires ou non, etc.). Nous avons choisi d'utiliser des règles d'enchaînement pour deux raisons principales :

- *gérer les enchaînements d'opérations de manière simple et automatique*. Étant donné que les règles nous donnent les enchaînements minimaux obligatoires, tout ce qui n'est pas obligatoire est donc laissé au choix de l'utilisateur. Les règles nous permettent donc à la fois de gérer les contrôles d'enchaînement et la latitude décisionnelle de l'utilisateur.
- *gérer automatiquement l'aide*. Les règles pouvant être utilisées généralement en chaînage avant et arrière, il devient possible de gérer l'aide d'utilisation dont l'objectif dans ce travail est quadruple :
  - avec le chaînage avant, elle permet de :
    - *décrire ce qui se passe après une action*,
    - *donner la suite d'actions permettant de terminer un traitement*, c'est-à-dire une procédure ou une opération.
  - avec le chaînage arrière, elle permet de :
    - *donner le chemin permettant d'aboutir à un but de l'application*,
    - *d'expliquer l'inaccessibilité des éléments de la Représentation Externe*.

L'avantage de la génération automatique ici est d'une part la **cohérence** entre les spécifications et l'implémentation, et d'autre part un **gain de temps** dans le développement. Grâce au formalisme simple et précis de Diane, il est possible de valider la latitude décisionnelle auprès de l'utilisateur avant même de faire une maquette de l'application. Si l'utilisateur estime que le système est correct, la génération automatique permet de conserver cette validation de manière sûre.

2.4. Génération de l'interface

La méthode Diane+ suppose que les données et les traitements sont déjà connus. Elle assemble les traitements en fonction des postes de travail en leur associant différentes propriétés qui permettent de définir une latitude décisionnelle. Les données quant à elles ne subissent pas de modification bien qu'elles puissent avoir des comportements propres en fonction du contexte lors de leur manipulation. Lorsqu'on assemble les opérations dans les procédures, on regroupe donc un ensemble fini de types de données. Ces types étant connus, il est possible de leur associer une représentation externe (par exemple un type énuméré peut être représenté par un ensemble de boutons radio). Nous verrons dans le chapitre 6 que nous nous basons pour cela sur le modèle OPAC qui est dérivé du modèle PAC.

De la même façon qu'il est possible de générer le code des procédures, il est possible de générer le code des écrans correspondant à ces procédures. Ainsi une opération qui utilise trois données de types différents peut être représentée par une fenêtre avec des zones de saisie pour les attributs de ces données. Le chapitre 6 montrera plus en détail les différents types de génération que l'on peut obtenir à partir des spécifications Diane+. D'un autre côté l'assemblage d'opérations en une procédure a souvent une signification pour l'utilisateur (par exemple une procédure permettant de saisir une facture). Si cette procédure est composée de nombreuses opérations que l'utilisateur peut déclencher à son gré, il est intéressant de regrouper ces opérations dans un menu correspondant à la procédure. La génération de l'interface contient également cet aspect. Notre logique de travail est donc une logique d'utilisation plutôt qu'une logique de fonctionnement.

2.5. Gestion automatique de la dynamique

Le dialogue homme-machine est directement et entièrement décrit dans les spécifications Diane. Cela signifie que tous les cas d'enchaînements y sont déjà prévus, de même que toutes les interdictions d'enchaînement. Il est donc facile de gérer le dialogue à partir du code généré depuis les spécifications, c'est-à-dire les règles d'enchaînement. Puisque ces règles contiennent à la fois les opérations, les paramètres des opérations et les contraintes d'enchaînement, le contrôleur de dialogue (c'est-à-dire le moteur d'inférence) se contente de vérifier si des règles peuvent être déclenchées à la suite d'une interaction de l'utilisateur ou de tout autre événement survenant lors d'une session. Puisque le système comporte au moins les procédures minimales (qui correspondent à la latitude décisionnelle la plus large possible), nous sommes sûrs que l'utilisateur ne pourra pas enfreindre les règles du dialogue données lors des spécifications. Il est prévu bien évidemment de pouvoir restreindre cette latitude en fonction des types d'utilisateurs voire même au fur et à mesure des sessions, par exemple si un usager veut toujours voir enchaîner certaines opérations qui ne l'étaient pas auparavant.

Les règles nous permettent donc une gestion automatique, facile et efficace de la dynamique des traitements. Mais elles ont également l'énorme avantage de permettre une gestion similaire de l'interface qui a été générée. En effet certains widgets générés correspondent à des opérations. Avec notre approche, les opérations qui deviennent invalides avisent aussitôt tous les widgets qui en dépendent, les rendant à leur tour inaccessibles pour l'utilisateur. De même, les menus générés suivent l'évolution des enchaînements. Au fur et à mesure que les opérations sont déclenchées, certaines deviennent valides ou invalides, de même que d'autres procédures (et donc d'autres menus) le deviennent également. D'habitude cette gestion dynamique de l'interface est laissée entièrement au soin de l'informaticien. Avec notre système, seul le moteur d'inférence en a la charge.

2.6. Gestion automatique de l'aide

Étant donné que les règles contiennent tous les enchaînements réalisables au sein d'une procédure, voire entre les procédures, nous pouvons les utiliser pour gérer les deux types d'aide nécessaire à une application : l'aide fonctionnelle et l'aide d'utilisation (ces deux aides sont bien évidemment basées sur les logiques qui s'y rapportent).

La logique fonctionnelle (ou de fonctionnement) est la plus répandue dans les applications informatiques. Elle permet d'avoir les renseignements nécessaires à une utilisation correcte des fonctions du système. L'aide fonctionnelle d'un traitement de texte permettra à l'utilisateur de choisir correctement les paramètres d'impression en fonction du type de papier désiré ou bien encore lui dira à quoi correspondent les sigles utilisés dans la règle (retrait des paragraphes, marques de tabulations...). Ce type d'aide n'est pas compliqué à gérer puisqu'il est purement statique. Il suffit de répertorier toutes les sources d'aide possibles (fenêtres, opérations, menus, etc.) et de leur associer une aide textuelle ou graphique. Pour choisir une aide particulière, la plupart des systèmes proposent une liste de rubriques, rarement exhaustive pour lesquelles l'utilisateur choisit celle(s) qui l'intéresse(nt). Nous envisageons de reprendre ce système avec en plus une aide locale (que proposent d'ailleurs d'autres systèmes) qui permet d'accéder directement à l'aide en rapport avec l'endroit où se trouve l'utilisateur. Certains auteurs qualifient ce type d'aide "d'aide contextuelle" puisqu'elle correspond à la situation présente, alors que d'autres estiment qu'une aide contextuelle correspond à une aide qui tient compte des valeurs des données, de l'historique des manipulations précédentes, etc. Pour notre part, nous décidons de garder cette dénomination d'aide contextuelle bien que le contexte véritable ne soit pas pris en compte.

Le deuxième type d'aide, l'aide d'utilisation, est de loin la plus intéressante bien que très peu de systèmes actuels la proposent. Elle consiste à donner quatre types d'informations :

- les conséquences d'une action de l'utilisateur. Ce type d'aide correspond à "*Que se passe-t-il si...?*". L'aide fournie est très utile par exemple en période d'apprentissage où l'utilisateur dispose alors d'un moyen beaucoup plus puissant que le "Undo" classique, ce dernier n'étant d'ailleurs pas supprimé. L'aide permet **d'anticiper** les réactions du système, le Undo permet de les annuler si l'utilisateur a décidé de les provoquer. Ce type d'aide est typique d'un fonctionnement en chaînage avant du moteur d'inférence. On donne au moteur un fait (l'utilisateur déclenche telle action), et celui-ci fournit tous les résultats déduits de ce fait. Ces résultats peuvent être textuels ou bien encore graphiques tel que dans [FOLEY 90b], mais dans tous les cas ils ne font que **refléter la simulation des conséquences** de l'action de l'utilisateur. Avec ce type d'aide, le système prend "virtuellement" l'état qu'il aurait si l'utilisateur avait effectivement déclenché son action.

- le chemin à parcourir pour atteindre un but ("*Comment faire pour…?*") . Cette aide est la plus puissante puisqu'elle donne la suite d'opérations à réaliser pour atteindre un but en fonction des opérations qui ont déjà été exécutées. Puisque les spécifications du dialogue sont basées sur la planification hiérarchique (c'est-à-dire sur une hiérarchie de buts et de sous-buts), les règles peuvent être utilisées en chaînage arrière pour retrouver le chemin qui a permis d'arriver à une étape particulière. Par exemple l'utilisateur d'un traitement de texte peut demander "Comment faire pour mettre du texte en deux colonnes" ou "Comment faire pour justifier du texte". Ce deuxième exemple est très parlant. La plupart des traitements de texte actuels contiennent bien évidemment la réponse à cette question, mais celle-ci est souvent difficile à trouver. Dans Word par exemple, il faut chercher dans la rubrique "Mise en forme" puis dans la rubrique "Paragraphe". Or si l'utilisateur ne sait pas que chaque paragraphe a un format qui lui est propre, il n'aura pas tout de suite l'idée de chercher l'aide dans cette rubrique. C'est pourquoi il faut fournir à l'utilisateur la liste de tous les buts que le système peut réaliser directement (par exemple formater du texte) ainsi que ceux qu'il est susceptible de réaliser (par exemple comment faire tenir plus de texte dans une page). Les premiers sont directement issus des procédures du dialogue, les seconds sont bâtis à partir des premiers (par exemple faire tenir plus de texte sur une page fait appel à changer la taille des caractères, changer les marges, etc.).

- *pourquoi un élément de l'interface n'est pas accessible alors qu'il est visible ?* Dans un menu par exemple, des commandes peuvent apparaître grisées. L'utilisateur demande donc une explication pour cette inaccessibilité. Le système va alors utiliser la description de cet élément inaccessible pour trouver les concepts Diane+ qui lui sont associés. Il pourra alors en déduire que certaines pré-conditions ne sont pas vérifiées, ou bien encore que certaines opérations ne sont pas achevées. Cette aide est fortement en relation avec la précédente puisqu'après avoir demandé POURQUOI tel élément est inaccessible, l'utilisateur peut éprouver le besoin de savoir COMMENT le rendre accessible. Dans notre exemple de commande de menu grisée, le système dira par exemple que l'inaccessibilité est dûe au fait que la commande précédente est elle aussi inaccessible. Dans ce cas, l'utilisateur voudra savoir comment rendre cette commande précédente accessible.

- *comment terminer une opération ou une procédure ?* Cette aide est surtout utilisée en conjonction avec l'aide "Comment faire pour". Les résultats de cette dernière peuvent en effet faire apparaître des réponses telles que : "Cette commande est inaccessible car l'opération X n'est pas terminée". Dans ce cas, l'utilisateur voudra savoir comment terminer l'opération X en question. Cette aide ne doit pas être confondue avec l'aide "Comment faire

pour" que l'on pourrait être tenté d'appliquer à la question "Comment faire pour terminer...". Cette aide est en fait beaucoup plus locale que "Comment faire pour" qui utilise, quant à elle, la totalité de la spécification de l'application. Elle reste en fait confinée à l'intérieur du bloc qui contient l'opération ou la procédure choisie.
Une autre application de cette aide est lorsque l'utilisateur veut quitter l'application. Le système inspecte alors l'état des opérations, et si certaines sont encore actives, ils affiche leurs noms et demande une confirmation de sortie qui implique l'interruption de ces opérations. Dans le cas où l'utilisateur désire achever correctement ces opérations et en supposant qu'il ne soit pas un expert de l'application, l'aide "Comment terminer" lui est d'un grand secours.

La gestion de l'aide est donc très facile grâce aux procédures issues des spécifications du dialogue. Il est à noter que le premier avantage de ces dernières est que l'aide en est directement issue. Cela signifie que si les procédures sont modifiées, l'aide subit immédiatement les changements et cela sans aucune intervention extérieure. On a donc un gain de temps et surtout une grande cohérence entre les spécifications et la réalité.

2.7. Implémentation en objet

Nous venons de voir les différents objectifs que nous nous proposons d'atteindre et qui seront présentés plus en détail dans les chapitres suivants. Il nous reste à présenter le type d'implémentation que nous avons choisi. L'orienté-objet nous est apparu sans aucune ambiguïté comme la technique la plus efficace. La raison est triple :

- *elle est proche des méthodes d'implémentation des IHM*. Tous les outils dont nous disposons aujourd'hui sont basés sur le concept d'objet. Les environnements de travail tels que Windows en sont la preuve,
- *elle facilite l'implémentation des concepts* énoncés plus haut. Ainsi le moteur d'inférence est un objet qui a une vie propre, les opérations sont capables de gérer elles-mêmes leur comportement en fonction de leur nature et de leur état,
- *elle est puissante sur le plan du génie logiciel*. Les critères du génie logiciel sont presque tous validés avec l'approche objet. On peut citer par exemple la réutilisation et la robustesse.

Notre travail est donc influencé par cette approche objet. Cela ne signifie pas qu'il soit inexploitable avec une autre approche, mais sa mise en œuvre en est facilitée. Les prochains chapitres nous montrerons par exemple comment nous gérons la propagation des événements sur le plan externe et sur le plan interne. Nous verrons que, grâce au concept objet, les opérations sont pratiquement autonomes ; le résultat principal est que le moteur d'inférence a une charge de travail moindre, ce qui accroît son efficacité.

3. Conclusion

Notre travail est basé sur les traitements plus que sur les données comme cela est le cas par exemple avec MacIda qui se base sur des modèles conceptuels de données. Nous avons délibérément choisi cette voie d'une part parce qu'elle nous paraît plus proche de la réalité (les ergonomes constatent en effet qu'il est plus fréquent que les utilisateurs parlent en termes de tâches plutôt qu'en termes de données) et d'autre part parce qu'elle nous fournit de nombreux avantages (aide d'utilisation, gestion automatique de la dynamique, etc.). Certes l'utilisation du concept objet (alors qu'on parle de tâche) paraît contradictoire. Nous verrons dans le chapitre suivant que cela n'est pas le cas.

Chapitre 5
La méthode Diane+

1. Introduction

Nous présentons dans ce chapitre la méthode Diane+ que nous avons choisie pour la spécification et la conception d'applications interactives. Nous verrons que cette méthode répond à certains problèmes non résolus par d'autres méthodes tel que celui de l'intégration de l'utilisateur dans le dialogue homme-machine. Diane+ permet également la construction de dialogues en fonction du niveau de l'utilisateur par l'intermédiaire des procédures effectives, minimales et prévues.
Nous allons tout d'abord définir, après une présentation générale de la méthode Diane qui est à l'origine de notre travail, quelques concepts fondamentaux pour la compréhension de Diane+ (but, tâche, précédence, etc.). Nous aborderons ensuite au § 3 les procédures (rôles, intérêts, etc.). Nous poursuivons avec quelques exemples de procédures Diane+ (§ 4) afin de clarifier les idées énoncées au cours de ce chapitre. Nous comparons ensuite Diane+ avec d'autres méthodes de conception (orienté-objet ou orienté-tâche) tout en regardant s'il est possible de les utiliser conjointement avec notre méthode. Une discussion qui nous paraît essentielle pour le développement des applications à l'heure actuelle et qui concerne le passage des tâches aux objets [TARBY 92], vient après. Nous exposons pour terminer le modèle de données (OPAC) que nous utilisons dans cette thèse.
Ce chapitre présente Diane+ avant tout sous l'aspect des traitements. Le chapitre suivant montrera comment s'établit le lien entre Diane+ et le modèle OPAC ainsi que les implications sur la génération et la gestion automatique.

2. Concepts fondamentaux de Diane+

2.1. Présentation de Diane

Diane+ est issue de Diane [BARTHET 88] qui est l'aboutissement d'une réflexion sur la conception des logiciels et sur leur utilisation. Diane est basée sur Merise dont elle comble certaines lacunes au niveau de la conception de l'IHM principalement. Étant donné que Merise analyse le travail en termes de tâches et d'opérations, il est normal de retrouver ces concepts dans cette méthode et dans la nôtre. Par ailleurs, Diane utilise la planification hiérarchique qui lui permet de structurer les tâches et les opérations en fonction des postes de travail concernés et des buts à atteindre.

Pour chaque poste de travail, on définit tout d'abord les tâches à réaliser. Puis pour chaque tâche, on construit les procédures minimales, prévues et éventuellement effectives que les futurs utilisateurs devront respecter. Ces procédures fournissent à la fois la répartition du dialogue homme-machine et la marge de manœuvre dont les utilisateurs disposent. Elles sont composées d'opérations (qui peuvent elles-mêmes se décomposer en opérations plus élémentaires) qui font appels aux services du noyau fonctionnel. Diane ne décrit donc pas formellement les traitements à réaliser contrairement à UAN [HARTSON 92] ou UIDE [FOLEY 89]. Le contenu des opérations doit pour l'instant être écrit par le concepteur.

Diane n'a pour but que la spécification du dialogue. Son utilisation s'avère donc difficile pour la gestion automatique du dialogue et de l'interface. Par ailleurs les relations entre des opérations de niveaux différents sont inexistantes et la notion de précédence ne permet pas de représenter tous les cas d'enchaînements. Notre travail a donc consisté à :

- *modifier la notion de précédence entre opérations* (cf § 2.4.2) afin de prendre en compte tous les cas d'enchaînement,
- *ajouter la notion de contraintes sur les opérations* (cf § 2.4.3) afin de donner le maximum de précision à la spécification de la latitude décisionnelle,
- *étendre l'utilisation de Diane dans le cycle de vie* afin de rendre possible la génération et la gestion automatique de l'aide et de l'interface [TARBY 91b] [TARBY 93], ce dernier point étant abordé dans le chapitre 6.

2.2. Logique de fonctionnement et logique d'utilisation

J-F Richard [RICHARD 83] déclare qu'il est possible de travailler avec un logiciel suivant une logique de fonctionnement ou suivant une logique d'utilisation. Nous allons voir que Diane+ permet également ces deux approches, ce que peu de méthodes permettent à l'heure actuelle.

La plupart des applications informatiques actuelles présentent une aide qui décrit les commandes proposées dans l'application. Ainsi l'aide d'Excel explique comment entrer une formule ou donne la liste exhaustive des fonctions financières ; l'aide de Word explique comment numéroter un plan ou comment mettre du texte en gras. Ce type d'aide repose sur ce que Richard qualifie de *Logique de Fonctionnement*, c'est-à-dire que le système explique l'effet de chaque commande, ce que l'on peut schématiser par "**si action P alors effet Q**".

Cette logique de fonctionnement est facile à implémenter puisqu'elle correspond exactement au raisonnement de l'informaticien au moment de la réalisation de l'application. En effet ce dernier va implémenter les fonctions qui ont été spécifiées auparavant en pensant aux conséquences du déclenchement de ces fonctions. La logique de fonctionnement regroupe donc les fonctions avec leurs résultats, par exemple "que se passe-t-il lorsqu'on appuie sur le bouton Options", "que se passe-t-il lorsqu'on passe en mode Aperçu avant impression" ?

Ce type de description des fonctions permet d'implémenter relativement facilement l'aide correspondante, c'est-à-dire l'aide de fonctionnement. Malheureusement cette aide est insuffisante pour une bonne exploitation de l'application à moins qu'on ne connaisse suffisamment l'application auparavant ! Un exemple significatif est celui de la justification d'un paragraphe avec Word. Si l'utilisateur connaît le fonctionnement général des traitements de texte, il cherchera cette aide par analogie d'abord dans les rubriques "Paragraphe" ou "Justifier du texte" qui n'existent malheureusement pas[60]. Il sera alors obligé de réfléchir un peu plus au nom de la rubrique d'aide à trouver. Il peut également parcourir la liste complète des rubriques (ce que fera également un utilisateur novice) et finira par trouver l'aide qu'il cherche dans la sous-rubrique "Paragraphe" de la rubrique "Mise en forme". L'aide fournie à cet instant est effectivement claire (ou relativement claire), mais il a fallu développer tout un raisonnement avant de la trouver. Ce type d'aide peut expliquer pourquoi peu d'utilisateurs lisent complètement et/ou attentivement les documentations, alors que nombre d'entre eux préfèrent apprendre en "essayant". Il serait plus judicieux de fournir une liste de manipulations réalisables (par exemple mettre du texte en gras, justifier du texte, etc.), et associer à chacune d'elles une ou plusieurs rubriques où chercher l'information.

[60] Ce choix s'explique par le fait que chaque paragraphe possède son propre style, donc si l'on veut justifier du texte, on doit au minimum justifier le paragraphe qui le contient, d'où la rubrique "Paragraphe".

Ceci nous amène au deuxième type de logique, la *Logique d'Utilisation*. Basée sur la planification hiérarchique, elle explique comment utiliser le système pour obtenir un résultat donné, plus précisément elle explicite le chemin à suivre pour atteindre un objectif. Ce chemin peut être unique ou non, de même qu'il peut être figé ou variable. Ainsi pour travailler en mode plan avec Word, on peut commencer par créer un nouveau plan ou bien encore transformer du texte existant en plan. Pour créer une table des matières, on peut utiliser le mode plan ou bien encore les entrées de table des matières. Par contre si l'on a construit le texte sur un plan, il vaut mieux utiliser le plan plutôt que les entrées de table des matières. La logique d'utilisation peut donc être schématisée par "**pour obtenir le résultat Q on peut faire la commande P**". Cette définition montre d'une part que l'on aborde les problèmes de manière totalement inverse par rapport à la logique de fonctionnement, et d'autre part que cela permet de donner une liberté de choix à l'utilisateur.

La logique d'utilisation facilite l'apprentissage puisqu'elle donne les étapes à parcourir pour atteindre un objectif particulier. Il est donc intuitif de l'associer avec l'aide. Si l'on regroupe tous les objectifs réalisables par l'application et qu'à chacun d'eux on associe les étapes à réaliser (ou des rubriques d'aide de type fonctionnement), un utilisateur novice pourra toujours trouver la solution à ses problèmes. Ainsi un traitement de texte tel que Word devrait contenir des rubriques d'aide telles que "Justifier du texte", "Structurer un document", "Changer les marges", etc.

Les deux logiques que nous venons de voir ne sont pas redondantes, bien au contraire elles sont totalement complémentaires. La logique d'utilisation fournit la liste des objectifs que l'on peut réaliser ainsi que la façon de les atteindre, alors que la logique de fonctionnement explique le fonctionnement de chacune des fonctions de l'application. Nous verrons dans le chapitre 6 que ces deux types d'aide peuvent être implémentés facilement par l'intermédiaire de Diane+ et de règles d'inférence. Retenons pour l'instant que la logique d'utilisation est celle que nous utiliserons avant tout avec Diane+.

2.3. Planification hiérarchique

Nous avons déjà défini la planification hiérarchique dans le chapitre 2 (cf § 2). Nous avons vu qu'elle permet de regrouper des traitements élémentaires (opérations) en fonction de buts à atteindre. Ces buts sont également hiérarchisés (on parle alors de sous-buts) et permettent d'atteindre des buts plus généraux. L'arbre de décomposition obtenu fournit l'ensemble des chemins possibles pour un but donné. Précisons que plusieurs chemins sont possibles pour parvenir à même but. Des méthodes telles que MAD associent à ces arbres de décomposition des possibilités de séquencialité, de choix, de simultanéité, etc., ces possibilités étant également présentes dans des travaux tels que Siroco-Guide.

La planification hiérarchique est à l'origine de la logique d'utilisation. Puisqu'elle décompose les buts en sous-buts et en opérations, elle définit par conséquent le chemin à emprunter pour atteindre un objectif (un but). Cette décomposition peut être générale ou bien encore adaptée aux différents types d'utilisateur. Ainsi on peut avoir une décomposition par niveau de connaissance (expert ou novice du domaine de l'application), par fréquence d'utilisation de l'application (utilisateur occasionnel ou non) ou bien encore par niveau de confidentialité (forte ou faible). Nous retrouvons tous ces types de décomposition avec les procédures Diane+.

L'avantage principal de la planification hiérarchique, outre la décomposition des buts, est de pouvoir représenter le modèle cognitif de l'utilisateur. Une analyse poussée de l'existant permet de mettre en valeur les tâches réalisées par les utilisateurs, leur séquencement, les choix dans les

actions, etc. Ce découpage est très important pour le futur système qui devra s'en inspirer tout en évitant au maximum les inconvénients de l'ancien système. Malheureusement le défaut majeur de la planification est de ne pas faire intervenir dans les opérations la répartition du contrôle entre l'homme et la machine. Certes la décomposition est issue d'une analyse en fonction d'un type particulier d'utilisateur, mais les opérations qui constituent les feuilles de l'arbre sont toutes placées sur un pied d'égalité. La planification ne prévoit pas en effet d'associer à chaque opération un type ou un déclencheur comme Diane+ le fait.

Remarquons pour terminer que la planification ne permet en général qu'une conception descendante alors que Diane+ peut également être utilisée pour une conception ascendante dans le cadre d'un rhabillage d'application par exemple [BARTHET 92].

2.4. Buts, tâches, activités

La planification hiérarchique est basée sur la notion de buts, un *but* étant un état désiré que l'on cherche à réaliser. A cet but correspond également la notion de *tâche*, c'est-à-dire un ensemble de traitements qui doivent être exécutés pour atteindre le but correspondant. Par exemple le but gérer le stock peut se décomposer en deux sous-buts qui sont gérer les achats et gérer les ventes et qui donnent lieu chacun à des tâches qui leur sont propres. En effet, pour gérer les achats, on n'a pas nécessairement besoin de connaître les ventes et réciproquement. Or ces deux tâches sont essentielles pour gérer le stock. La notion de but est donc associée à celle de tâche alors que le contraire est faux ; les tâches peuvent être exécutées sans que l'on connaisse le but pour lequel elles ont été créées alors qu'à un but correspond nécessairement des tâches.

Il faut différencier la notion de tâche de la notion d'*activité*. La tâche est statique ; elle décrit les traitements à exécuter et éventuellement les enchaînements qui lient ces traitements. L'activité est dynamique ; elle décrit l'exécution d'une tâche dans un contexte donné. Ainsi la tâche Commander un livre est statique (remplir le bon de commande, envoyer le bon de commande, etc.) alors que l'activité Commander le livre 20000 lieux sous les mers correspond à l'occurrence de la tâche Commander un livre pour le livre 20000 lieux sous les mers. Cette activité peut respecter la description originale de la tâche Commander un livre, mais elle peut également en modifier la description dans les limites autorisées par la description de la tâche. On peut différencier les tâches et l'activité de la façon suivante (Figure 5.1) :

- la *Tâche* est l'ensemble des traitements devant être exécutés pour atteindre un but (ici Commander un livre). Elle correspond au niveau d'abstraction le plus élevé et n'aura en général pas d'occurrence puisque celles-ci rentreront soit dans le cadre des tâches effectives soit dans le cadre des tâches prévues.
- la *Tâche Prévue* est l'ensemble des traitements permettant un déroulement normal[61] de la tâche et ceci pour un poste de travail donné. Dans le cas de la commande de livre, la Tâche Prévue pour un bibliothécaire serait par exemple : chercher la référence du livre, puis remplir le bon de commande, et enfin envoyer le bon de commande.
- la *Tâche Effective* est l'ensemble des traitements exécutés lors d'une session de travail. Dans l'exemple de la commande, la Tâche Effective pour un bibliothécaire serait par exemple : remplir le bon de commande, et envoyer le bon de commande. La recherche de la référence du livre peut être ignorée car on peut supposer que le bibliothécaire connaît suffisamment les livres qu'il commande. La Tâche Effective dépend donc fortement du poste de travail et du niveau de l'utilisateur.

[61] Normal est à prendre ici dans le sens de "habituel", ou "logique" suivant le cas.

- *l'Activité* est l'occurrence d'une tâche pour un ensemble de données (ici le livre 20000 lieux sous les mers).

Lors de l'étude de l'existant, on recueille les activités par des observations ou des enregistrements. On en extrait alors les tâches correspondantes (les Tâches Effectives) qui donneront naissance aux procédures effectives, puis les Tâches Prévues qui fourniront les procédures prévues. La dernière étape consiste à extraire le noyau dur de l'ensemble de ces tâches, ce noyau dur donnant naissance aux procédures minimales.

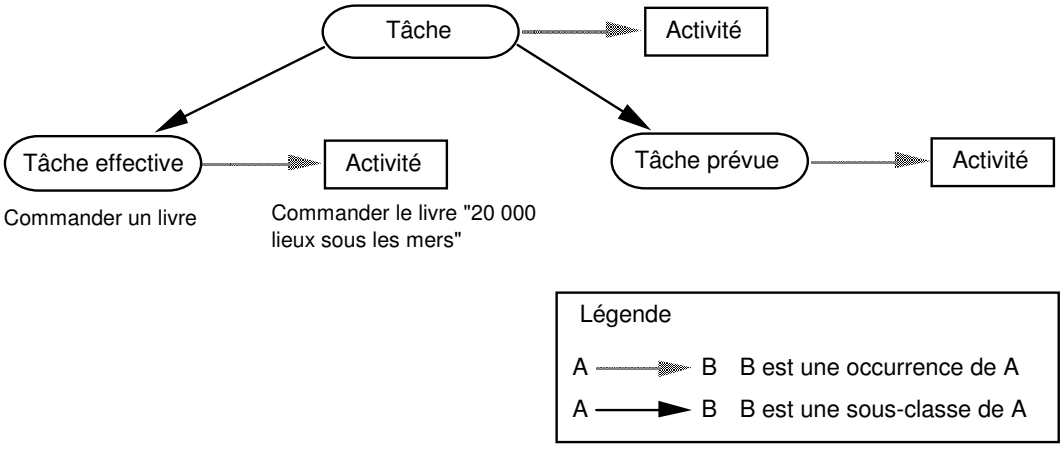


Figure 5.1 : Lien entre Tâche et Activité

2.5. Représentation des opérations

Nous venons de voir qu'une tâche est constituée d'un ensemble de traitements. Ces traitements sont en fait des opérations, élémentaires ou non, qui peuvent être liées entre elles. Nous allons voir la composition d'une opération Diane+ et comment elle peut être reliée à d'autres opérations.

2.5.1. Description d'une opération

Une opération est un ensemble de traitements qui permettent de faire passer le système d'un état à un autre. On peut ainsi rapprocher les opérations des transitions des diagrammes états-transitions. Théoriquement la décomposition des tâches en opérations devrait se terminer à partir du moment où l'on obtient uniquement des opérations élémentaires au sens Merise, c'est-à-dire des traitements qui ne nécessitent pas d'événement extérieur au couple homme-machine. En réalité nous arrêtons la décomposition dès que l'on obtient des opérations qui ne requièrent pas de réaction de la part de la machine. Ainsi la saisie d'un code client se décomposera en une opération qui récupère le code entré par l'utilisateur et une opération qui contrôle la validité de ce code. Si l'on décide que le contrôle doit se faire tout au long de la saisie, on décomposera l'opération de saisie en autant de couples (récupération de la touche frappée, contrôle de la frappe) qu'il y a de caractères dans le code client.

Outre son contenu (traitements) et son nom, une opération Diane+ possède des attributs qui lui confèrent différents statuts[62]. Nous allons à présent les énumérer (Figure 5.2).

[62] Nous verrons, dans le chapitre 6, que l'évolution de l'état d'une opération Diane+ au cours d'une session de travail dépend de son statut.

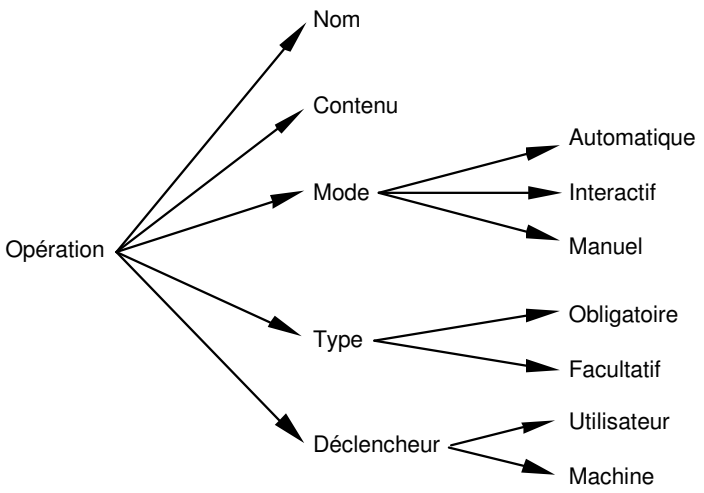


Figure 5.2 : Description d'une opération Diane+

Une opération peut avoir un *mode automatique* ou *interactif*. Dans le premier cas, l'ordinateur est le seul à avoir le contrôle de l'opération[63] ; l'utilisateur n'intervient donc pas au cours de l'opération. Ce cas se présente par exemple pour des impressions ou pour des affichages. Dans le cas d'une opération interactive, l'ordinateur a besoin de données entrées par l'utilisateur pour effectuer les traitements de l'opération. Ceci se présente pour des saisies, des consultations, etc. L'opération subit donc un dialogue homme-machine local[64], ce qui signifie que la machine peut être aux ordres de l'utilisateur (par exemple l'utilisateur saisit les données dans l'ordre qu'il veut) ou à l'inverse que l'utilisateur subit des contraintes imposées par la machine (par exemple pour la saisie d'une date, la machine peut imposer d'entrer dans l'ordre : le jour, le mois et l'année). Les deux modes d'opérations que nous venons de voir représentent à eux seuls l'ensemble des opérations que l'on peut trouver dans une application informatique. Il existe également des opérations qui, bien qu'elles ne soient pas informatiques, soient utiles à la compréhension des tâches, par exemple lors de l'analyse de l'existant. Ce sont les opérations *manuelles* ; elles ne font nullement intervenir l'informatique pour être réalisées. Ce sont par exemple des opérations tels que des signatures ou des manipulations physiques. Ces opérations peuvent cependant être très utiles pour représenter plus complètement les tâches ou pour permettre aux concepteurs d'avoir des repères. La figure 5.3.a reproduit les trois modes d'opérations.

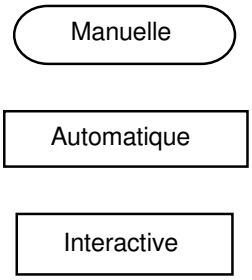


Figure 5.3.a : Les trois modes d'opérations Diane+

[63] Nous supposons dans le reste de ce rapport que chaque opération est interruptible par l'utilisateur à moins que le concepteur l'ait interdit.

[64] Nous employons le terme "local" pour différencier ce dialogue du dialogue homme-machine général qui concerne toute l'application.

Une opération possède également un *type*. Celui-ci peut être *obligatoire* ou *facultatif* (Figure 5.3.b). Dans le cas d'un type obligatoire, l'opération doit impérativement être exécutée pour que le but auquel elle appartient soit atteint. Si le type est facultatif, l'opération peut être exécutée ou non sans que l'obtention du but soit remise en cause (ceci à condition que l'opération facultative ne soit pas soumise à des contraintes par une opération l'englobant). Prenons deux exemples : si l'on considère le but enregistrer un client, l'opération saisir le nom du client est obligatoire alors que l'opération imprimer un client est facultative. Ainsi toutes les opérations de consultation, d'impression, de tri, etc., sont en général facultatives.

Figure 5.3.b : Une opération obligatoire interactive et une opération facultative interactive

Le troisième attribut concerne le *déclencheur* des opérations. Celui-ci peut être soit l'utilisateur soit l'ordinateur. Par défaut une opération est supposée déclenchée par l'ordinateur, c'est-à-dire que toutes les opérations des figures 5.3.a et 5.3.b sont déclenchées par l'ordinateur. Pour préciser qu'une opération est déclenchée par l'utilisateur, on lui ajoute ce que nous appelons un déclenchement utilisateur (ou encore un déclencheur) que nous représentons de la manière suivante (Figure 5.4) :

Figure 5.4 : Le symbole du déclenchement utilisateur

Si l'on reprend l'exemple de l'impression, bien que cette opération soit automatique, elle demande à être déclenchée par l'utilisateur. De même une consultation doit être déclenchée par l'utilisateur. On représente alors ces opérations de la façon suivante (Figure 5.5) :

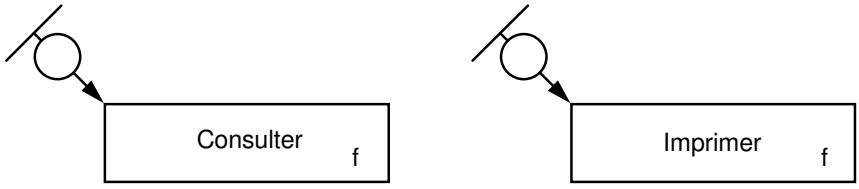


Figure 5.5 : Une opération interactive facultative à déclenchement utilisateur et une opération automatique facultative à déclenchement utilisateur

Lorsqu'une opération comporte un déclencheur, cela suppose également que cette opération peut être déclenchée autant de fois que l'utilisateur le souhaite. Le déclencheur est donc synonyme de boucle sur une opération[65]. La figure 5.6 représente deux opérations en cascade et avec des déclencheurs. L'utilisateur doit les déclencher dans l'ordre imposé, mais en déclenchant à loisir chacune d'elles. Ainsi les séquences (AB), (AAB), (ABB), (AAABBB), (AABBBABBABBABB) ou de manière plus générale $(A^+B^+)^+$ sont autorisées[66]. Si "B" avait été facultative, on aurait écrit $(A^+B^*)^+$.

[65] Cette caractéristique peut être limitée par des contraintes (cf § 2.4.3).

[66] "+" signifie que le nombre de répétitions va de 1 à l'infini , "*" signifie que le nombre de répétitions va de 0 à l'infini.

Outre ces différentes caractéristiques, les opérations à déclencheur peuvent également être des *opérations par défaut*. Nous représentons ce type d'opérations par un double cadre (Figure 5.7). Lorsqu'on se trouve en présence d'opérations de ce type, l'utilisateur peut déclencher l'opération par défaut en tapant sur Entrée. Dans la figure 5.7, l'utilisateur a le choix entre "A" et "B". S'il tape sur Entrée, il déclenche "B". Celle-ci achevée, il peut déclencher "A" si le type de "A" ou les contraintes de l'opération mère (si elle existe) le permettent. S'il ne tape pas sur Entrée, il pourra déclencher "A" ou "B" en premier.

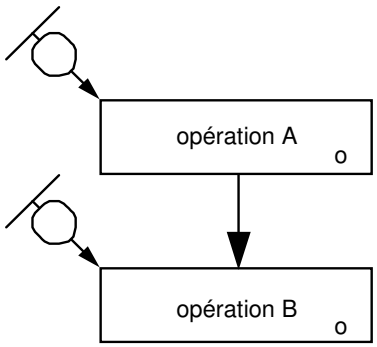


Figure 5.6 : Deux opérations permettant une boucle sur chacune d'elles.

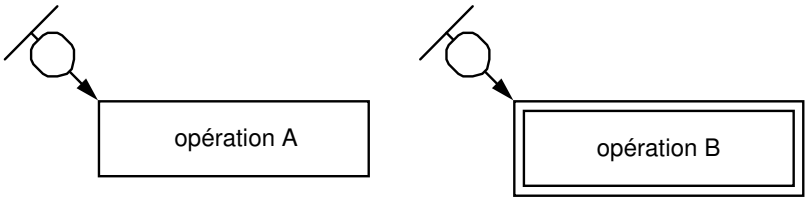


Figure 5.7 : Deux opérations dont une par défaut (B)

Ce type d'opération "par défaut" est souvent utilisé dans les IHM. L'exemple le plus courant est la validation par le bouton OK ou l'annulation par le bouton Cancel. On trouve également des exemples de ce type d'opérations dans une sélection par défaut dans une liste lors de son ouverture ou bien encore le choix d'une police de caractères à la création d'un nouveau document.

Nous pouvons déduire de ce qui précède certaines implications. Ainsi une opération facultative est toujours à déclenchement utilisateur[67]. Ceci s'explique par le fait que si l'opération est facultative, d'une part elle doit être disponible à tout instant (en supposant que le système le permette), et d'autre part cela signifie que la machine ne peut pas savoir quand elle sera déclenchée (l'utilisateur doit donc pouvoir la déclencher quand il le veut), à moins d'intégrer des méta-connaissances sur l'utilisateur et les traitements.

Règle 1 : Une opération facultative est toujours à déclenchement utilisateur.

[67] Les règles que nous donnons dans ce chapitre permettent une vérification automatique de la validité syntaxique des procédures et des opérations. D'autres vérifications peuvent être faites de manière similaire. Cependant ce travail n'a pas été approfondi jusqu'à présent. Il fera l'objet d'une étude détaillée par la suite.

Une deuxième implication est que toutes les opérations qui ne sont pas à déclenchement utilisateur sont obligatoires. En effet si ce n'était pas le cas, comment le système pourrait-il déterminer si oui ou non il faut déclencher l'opération en question.

Règle 2 :	Une opération qui ne possède pas de déclenchement utilisateur est toujours obligatoire (aux conditions de déclenchement près).

Nous donnons ci-dessous quatre autres règles qui sont immuables de par la sémantique des imbrications. Ces quatre règles sont :

Règle 3 :	Une opération automatique ne peut contenir que des opérations filles[68] automatiques.

Règle 4 :	Une opération automatique à déclenchement utilisateur ne peut contenir que des opérations filles automatiques avec ou sans déclenchement utilisateur.

Règle 5 :	Une opération interactive peut contenir des opérations filles automatiques et interactives.

Règle 6 :	Une opération interactive à déclenchement utilisateur peut contenir des opérations filles automatiques et interactives, avec ou sans déclenchement utilisateur.

Terminons en remarquant qu'une opération Diane+, comme dans beaucoup d'autres méthodes, possède des pré-conditions et des post-conditions.

2.5.2. Notion de précédence

Nous avons vu qu'une tâche est un ensemble d'opérations se déroulant dans le temps. Il est donc nécessaire de donner une structure permettant de relier temporellement ces opérations entre elles. Intervient alors la notion de *précédence*.

La relation de précédence permet de relier deux opérations. Diane [BARTHET 88] distingue deux types de précédence :

- *la précédence permanente*. Relier deux opérations par une précédence permanente signifie que ces opérations devront obligatoirement être exécutées consécutivement pour que le but auquel elles appartiennent puisse être atteint.

- *la précédence indicative*. Relier deux opérations par une précédence indicative signifie que ces opérations peuvent être exécutées dans n'importe quel ordre, mais il est préférable de les enchaîner suivant l'ordre prescrit. Ce type de précédence est utile pour l'apprentissage. Un utilisateur qui ne connaît pas l'application pourra ainsi suivre les enchaînements conseillés pour une bonne utilisation de l'application. Par exemple une personne utilisant pour la

[68] Dans un souci de concision, nous appellerons opération "mère" l'opération qui contient d'autres opérations et opération "fille" une sous-opération.

première fois un logiciel d'emprunt de livres dans une bibliothèque pourrait connaître ainsi le processus à suivre pour faire un emprunt.

La précédence indicative ne sera pas abordée dans notre travail. Nous avons en effet préféré modifier la notion de précédence permanente afin d'augmenter les possibilités d'enchaînements, les procédures prévues prenant en compte, quant à elles, les enchaînements conseillés (en plus des enchaînements obligatoires). Dans [BARTHET 88], M-F Barthet définit la précédence permanente comme une relation entre deux opérations obligatoires et signifiant par conséquent que ces deux opérations devront être exécutées toutes les deux et dans cet ordre.
La figure 5.8 représente les deux cas de précédence entre deux opérations A et B. Dans le cas de la précédence permanente (cas de gauche), cela se traduit par "pour que B soit exécutable il faut que A ait été exécutée". Dans le cas de la précédence indicative (cas de droite), le schéma se traduit par "il faut exécuter A et B, et il est préférable de commencer par A".

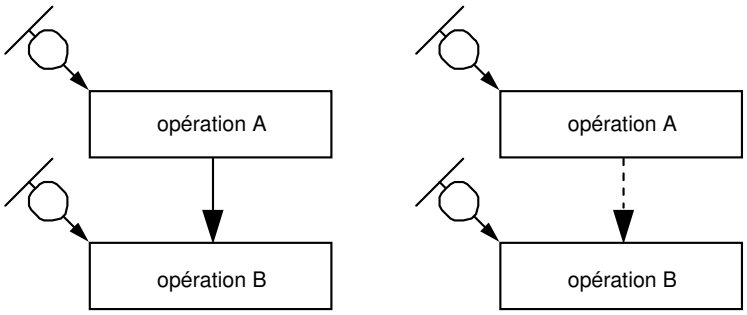


Figure 5.8 : Les deux types de précédence entre des opérations Diane. A gauche la précédence permanente, à droite la précédence indicative.

Avec la précédence permanente il reste toutefois possible d'intercaler, lors des sessions de travail, une autre opération entre les deux précédentes, mais cela suppose que l'opération intercalée est à déclenchement utilisateur. Ainsi sur la figure 5.9, l'opération C peut être déclenchée avant A, entre A et B, ou après B. Si cette opération C est obligatoire, le but correspondant à ces trois opérations est atteint quel que soit l'endroit où a été insérée l'opération C, mais il faut tout de même que C ait été exécutée. Par contre si C est facultative, le but est atteint dans tous les cas, c'est-à-dire que C ait été exécutée ou non et ceci quel que soit l'endroit.

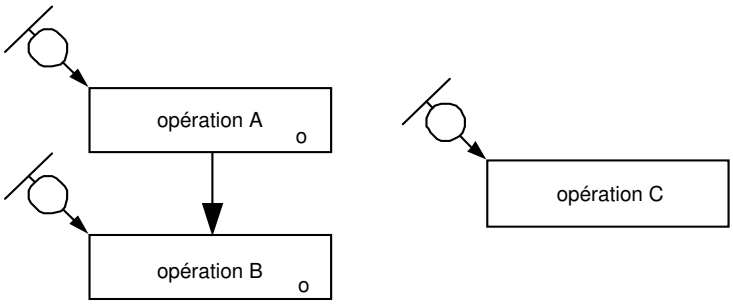


Figure 5.9 : Exemple de précédence permanente avec insertion possible d'une autre opération dans l'enchaînement

Dans notre travail nous avons redéfini la précédence permanente en disant que si deux opérations A et B sont reliées par cette précédence, cela signifie que "pour que B puisse être exécutée, il faut et il suffit que **toutes les opérations obligatoires précédentes aient été exécutées**". Grâce à cette nouvelle définition, nous élargissons les propriétés énoncées dans [BARTHET 88] ce qui permet d'avoir des situations comme celles exposée sur la figure 5.10.

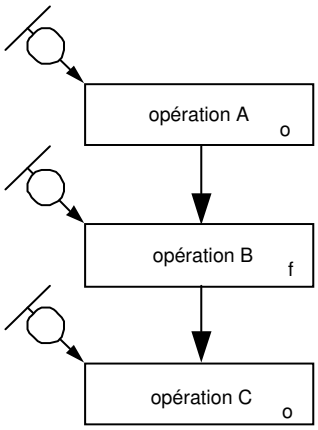


Figure 5.10 : Un exemple d'enchaînement avec précédence permanente reliant des opérations obligatoires et facultatives. Cette figure revient à écrire "A & C | A & B & C" où "&" est l'opérateur temporel ET (équivalent à "PUIS") et "|" est l'opérateur OU non exclusif.

Cette figure 5.10 nous apprend trois choses :

- on peut avoir des précédences permanentes entre des opérations obligatoires et facultatives,
- pour que le but correspondant soit atteint, il faut exécuter d'abord "A", puis "C",
- l'opération "B" n'est pas nécessaire à l'obtention du but, mais si l'utilisateur choisit de la déclencher, il ne peut le faire qu'entre "A" et "C".
  Ce cas peut se produire par exemple dans une messagerie électronique. L'opération "B" correspondrait alors à la possibilité d'imprimer le message que l'utilisateur est en train de lire. Une fois qu'il est sorti de l'éditeur (qui lui sert à écrire ou à afficher des messages), cette commande n'a plus de sens, d'où l'intérêt de la représenter entre deux autres opérations correspondant à son contexte.

2.5.3. Décomposition et contraintes

Diane comme beaucoup d'autres méthodes permet d'intégrer des opérations dans d'autres opérations afin d'avoir des vues plus ou moins détaillées sur ces opérations. Dans Diane+, le fait de regrouper des opérations dans une opération plus générale implique que cette opération générale n'est pas associée à un traitement propre, c'est-à-dire qu'elle ne manipule pas directement elle-même les données de l'application. L'opération générale peut cependant exécuter des pré- et des post-actions au niveau de la représentation externe comme nous le verrons au Chapitre 6.

Le nombre d'imbrications n'est pas limité. Il dépend du niveau de complexité de l'application et de la précision que l'on cherche à obtenir. Une opération ne contient pas seulement d'autres opérations, mais également les précédences qui les relient. Ainsi sur la figure 5.11, l'opération "A" contient trois sous-opérations liées entre elles par des précédences permanentes, de même que l'opération "A.2" en contient deux. Sur cette figure, la partie grisée correspond au détail de l'opération "A". Dans les spécifications Diane+, nous utiliserons soit l'opération générale, soit sa décomposition, soit les deux à la fois. Le premier cas permet de décrire des vues générales sur les traitements, le second cas permet d'en voir les détails, le troisième cas permettant surtout d'éviter des ambiguïtés de schéma (par exemple à cause d'une surcharge de blocs de détails).

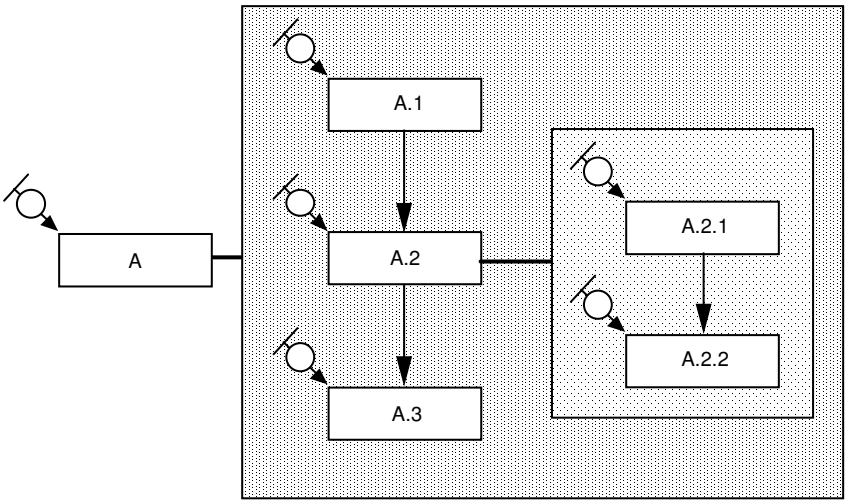


Figure 5.11 : Une opération et sa décomposition en sous-opérations

Sur la figure 5.11, l'opération "A" ne sera considérée terminée que lorsque toutes ses sous-opérations ("A.1", "A.2" et "A.3") le seront également. Or "A.2" contient également des sous-opérations. Le même processus doit donc être appliqué pour "A.2". Étant donné les liens de précédence unissant toutes ces opérations, l'ordre dans lequel elles devront se dérouler est donc : "A" (début), "A.1", "A.2" (début), "A.2.1", "A.2.2", "A.2" (fin), "A.3", "A" (fin).

Il est par ailleurs possible de ne pas lier des sous-opérations entre elles. Ainsi sur la figure 5.12, une fois que l'on a déclenché "A", on peut déclencher dans n'importe quel ordre "A.1", "A.2" et "A.3". Par contre le déclenchement de "A.2" impose l'exécution de "A.2.1" puis de "A.2.2".

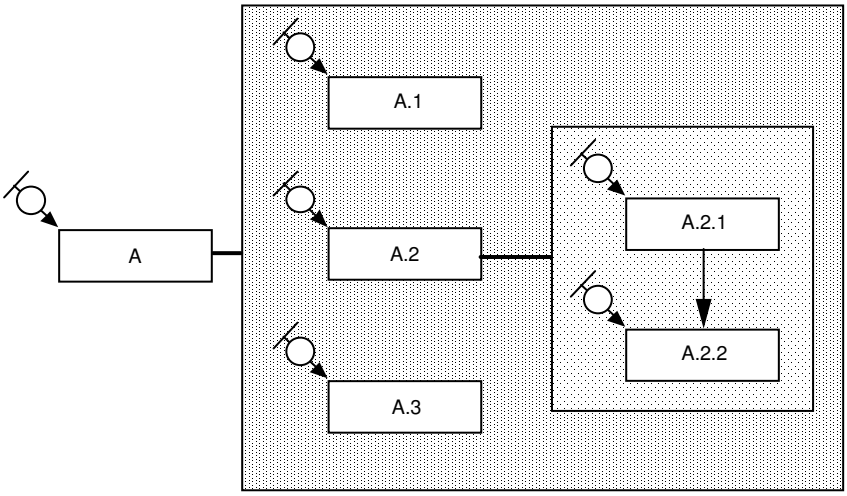


Figure 5.12 : Une opération avec des sous-opérations avec et sans liens de précédence.

Dans ces deux exemples (Figures 5.11 et 5.12), nous n'avons pas fait apparaître pour des raisons de simplicité les types des opérations (obligatoire ou facultatif). Or Diane+ impose de choisir un type pour chaque opération. Il est donc possible d'avoir des opérations obligatoires qui contiennent d'autres opérations obligatoires et des opérations facultatives qui contiennent d'autres opérations facultatives. Mais il est également possible d'avoir des opérations obligatoires qui contiennent des opérations facultatives et réciproquement. Cette caractéristique de Diane+ nous a amenés à intégrer des contraintes sur les opérations, principalement pour la phase de conception où l'on a

besoin de préciser la répartition du contrôle du dialogue. Ces contraintes (qui sont détaillées juste après) sont nécessaires pour représenter tous les cas de répartition du contrôle. Nous supposons pour l'instant que dans les quatre cas énoncés, il n'y a pas de mélange possible. Cela signifie que dans une opération mère, on ne trouvera que des opérations filles de même type. Des exemples intégrant les différents mélanges seront vus plus loin. Nous détaillons à présent ces différents cas :

- opération obligatoire avec des sous-opérations obligatoires. L'opération mère doit être exécutée pour que le but soit atteint ce qui implique que toutes ses sous-opérations doivent également être exécutées.

- opération obligatoire avec des sous-opérations facultatives. L'opération mère doit être exécutée pour que le but puisse être atteint. Or d'après ce que nous avons dit précédemment, les opérations filles doivent également être exécutées. Celles-ci étant toutes facultatives, il est nécessaire d'ajouter des contraintes sur les opérations filles au niveau de l'opération mère. Ainsi on peut spécifier que "aucune", "au moins une", "au plus une", "une et une seule", "un nombre quelconque" ou "toutes" les opérations filles doivent être exécutées. Si l'on examine en détail ces différentes contraintes, on s'aperçoit que le type facultatif des opérations filles restreint le nombre des possibilités. Reprenons-les en détail :
  - *aucune* opération fille. Cette contrainte est équivalente à dire qu'il n'y a pas de sous-opérations. Il y a donc contradiction. Par conséquent cette contrainte est impossible ici.
  - *au moins une* opération fille. Cette contrainte correspond en quelque sorte à une relation 1-n entre l'opération mère et ses filles. A partir du moment où une des opérations filles a été exécutée, l'opération mère est considérée terminée.
  - *au plus une* opération fille. Les deux seuls choix sont 0 ou 1 opération fille à exécuter (relation 0-1). Cela implique que l'on n'est pas obligé de déclencher une des opérations filles pour atteindre le but, mais dans le cas où on choisit effectivement l'exécution de l'une d'entre elles, les autres deviennent inaccessibles. Or cette contrainte signifie également que l'opération mère n'est pas obligatoire pour le but. Elle est donc facultative, d'où contradiction. Cette contrainte est donc impossible pour une opération mère obligatoire. Nous verrons plus loin qu'elle est intéressante pour les opérations mères facultatives.
  - *une et une seule* opération fille. Une fois que l'on a déclenché une opération fille et que celle-ci s'achève, l'opération mère est considérée terminée.
  - *un nombre quelconque* d'opérations filles. Cela signifie que l'on peut déclencher de 0 à n opérations filles et que ce choix n'influe pas sur l'obtention du but. On aboutit donc à une nouvelle contradiction. Cette contrainte est donc impossible ici.
  - *toutes* les opérations filles. Toutes les opérations filles doivent être exécutées ce qui revient à dire qu'elles sont toutes obligatoires, d'où une contradiction par rapport à notre hypothèse de départ. Cette contrainte est inutile puisqu'elle est équivalente au cas où l'opération mère et les opérations filles sont toutes obligatoires.

  Les contraintes qui peuvent s'appliquer à ce cas sont donc : "au moins une" et "une et une seule" opération fille.

Règle 7 :	Si une opération obligatoire contient des sous-opérations facultatives, les seules contraintes possibles sur les sous-opérations facultatives sont "au moins une" et "une et une seule".

- opération facultative avec des sous-opérations facultatives. L'opération mère ne doit pas être nécessairement exécutée pour que le but soit atteint, mais dans l'affirmative cela suppose que l'on peut choisir zéro, une ou plusieurs de ses opérations filles. On retrouve alors les

contraintes précédentes. Dans le cas présent, les seules qui soient acceptables sont : “au moins une”, “au plus une” et “une et une seule”. Les autres sont impossibles (“toutes” et “aucune”) ou inutile (“nombre quelconque”).

Règle 8 :	Si une opération facultative contient des sous-opérations facultatives, les seules contraintes possibles sur les sous-opérations sont “au moins une”, “au plus une” et “une et une seule”.

- opération facultative avec des sous-opérations obligatoires. L’opération mère n’est pas nécessaire au but, mais si l’utilisateur la déclenche, toutes ses sous-opérations devront être exécutées. Il n’y a donc pas de contrainte possible.

En résumé, **les contraintes sur les filles s’appliquent toujours sur des opérations facultatives**. Pour représenter une opération mère avec une contrainte sur ses filles, nous employons un cadre double contenant la contrainte dans sa partie supérieure. La chiffre de gauche indique le nombre minimum de filles facultatives à exécuter, le chiffre de droite le nombre maximum. Dans la figure 5.13, l’opération “A” possède des sous-opérations avec la contrainte “une et une seule” alors que l’opération “B” a la contrainte “au moins une”.

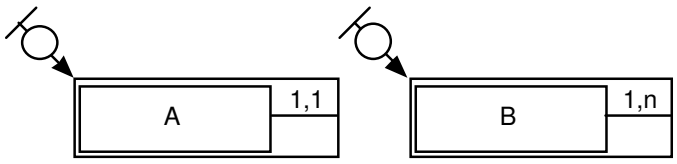
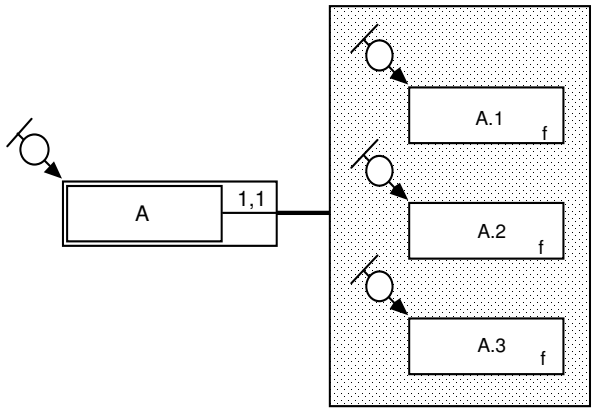
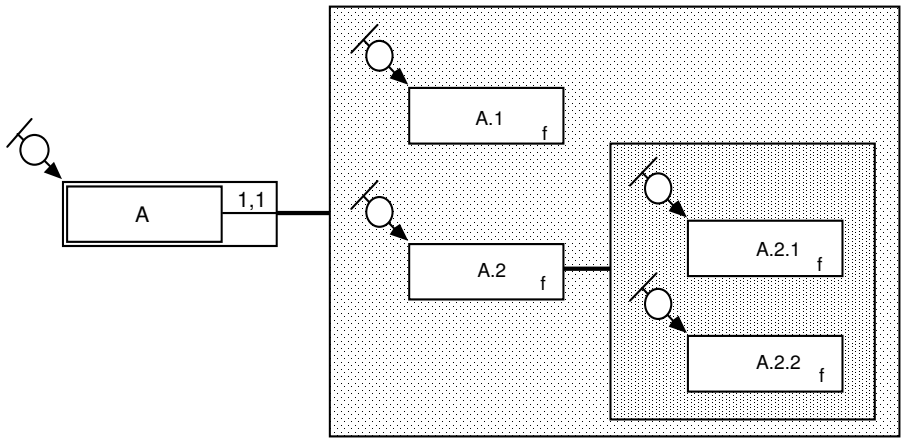


Figure 5.13 : Représentation d’opérations avec des contraintes sur les sous-opérations facultatives

Remarque : la contrainte “une et une seule” implique l’exclusion mutuelle entre les opérations concernées. Supposons que l’opération “A” de la figure 5.13 contienne trois opérations filles. La figure 5.14 représente deux exclusions mutuelles différentes. Dans le cas du haut, il y a exclusion mutuelle entre toutes les opérations filles puisque la contrainte “1,1” de “A” indique qu’on ne peut exécuter qu’une seule opération fille. Plus précisément, dès qu’une opération fille est déclenchée (par exemple “A.2”), le contrôleur de dialogue va regarder si le nombre maximal de déclenchement est atteint (ici 1). Comme cela est vrai, il rend inaccessible les opérations filles restantes (“A.1” et “A.3”). Dans le cas du bas, l’exclusion mutuelle ne s’applique que sur “A.1” et “A.2”. Dès que l’utilisateur déclenche “A.1”, le contrôleur rend inaccessible “A.2”, donc également “A.2.1” et “A.2.2” grâce à l’autogestion des opérations (détaillée dans le Chapitre 6). Inversement si l’utilisateur déclenche “A.2.1” ou “A.2.2”, le contrôleur rend inaccessible “A.1”, par contre il est toujours possible d’exécuter “A.2.1” et “A.2.2” en même temps puisqu’il n’y a pas de contrainte dans “A.2”.



Exclusion mutuelle entre A.1, A.2, A.3



Exclusion mutuelle entre A.1 et (A.2.1, A.2.2)

Figure 5.14 : Deux cas d'exclusion mutuelle gérés directement par la contrainte "une et une seule"

Il reste une dernière contrainte qui peut être intéressante à la fois pour les imbrications d'opérations et pour les opérations elles-mêmes : le nombre de déclenchements. On peut effectivement avoir besoin de borner le nombre de déclenchements d'une opération. Ce nombre (ou plutôt l'intervalle) étant fixé une fois pour toutes, le contrôleur de dialogue sera à même de gérer le nombre de déclenchements des opérations. Nous représentons une opération de ce type avec un cadre double contenant le nombre de déclenchements dans la partie inférieure (Figure 5.15). Un exemple de ce type de contrainte est le nombre maximum d'essais (en général trois) dont on dispose pour entrer le code d'une carte bancaire. Dans ce cas l'intervalle sera [1,3].

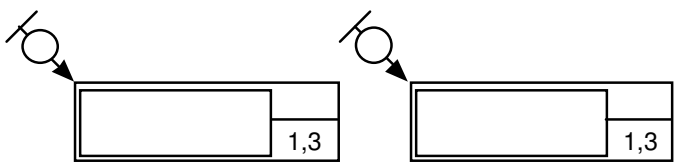


Figure 5.15 : Deux opérations avec un nombre de déclenchements à choisir dans l'intervalle [1,3]

Il est important de bien faire la différence entre le nombre de déclenchements d'une opération et le nombre d'opérations à exécuter dans une opération mère. Ainsi l'opération "A" de la figure 5.16

comporte trois sous-opérations avec la contrainte “au plus une”. Elle subit également la contrainte de n’être déclenchable qu’une seule fois. Cela signifie que si l’utilisateur la déclenche, il ne pourra exécuter ensuite que “A.1” ou “A.2” ou “A.3” (les deux “ou” sont exclusifs). Dans les deux premiers cas, il n’aura qu’une seule exécution possible de l’opération fille, alors qu’avec “A.3” il aura droit à autant d’exécutions de “A.3” qu’il veut (0,n).

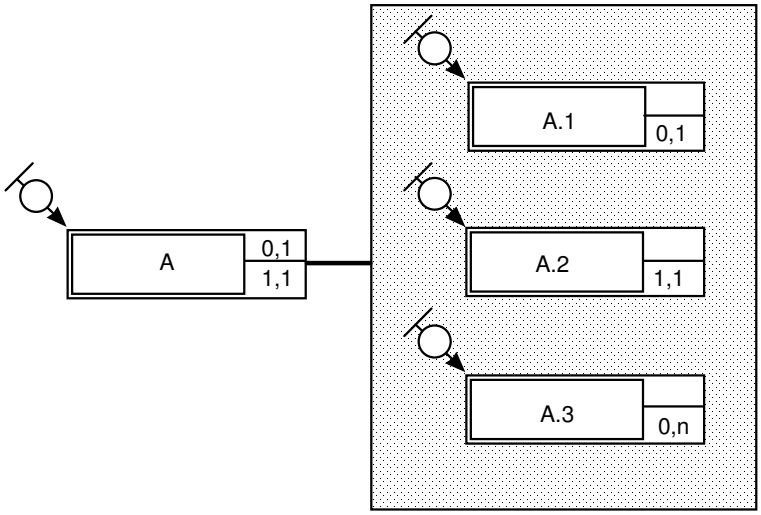


Figure 5.16 : Exemple d’une opération mère avec ses filles et subissant diverses contraintes

La représentation d’une opération subissant des contraintes sur ses filles ou sur elle-même est donc la suivante (Figure 5.17) :

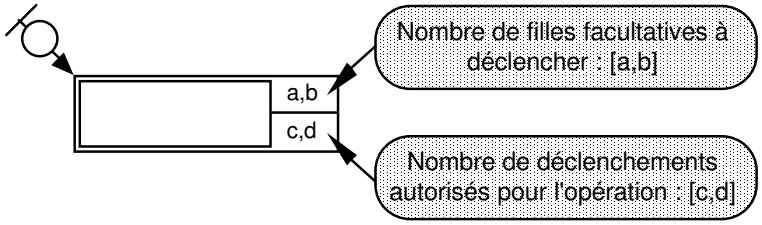


Figure 5.17 : Représentation d’une opération avec contraintes sur ses filles et sur elle-même

Par défaut une opération qui ne possède pas de double cadre est une opération qui soit ne subit pas de contrainte, soit subit des contraintes de façon implicite. Par exemple sur la figure 5.18.a, les opérations “A.1”, “A.2” et “A.3” doivent obligatoirement être déclenchées puisque “A” est obligatoire, et le nombre de déclenchements de ces trois opérations est laissé au choix de l’utilisateur puisqu’il n’y a pas de contrainte sur “A” (contrainte implicite 1-n). On emploiera donc indifféremment les deux types de représentation (avec cadre de contrainte comme sur la figure 5.18.a ou sans cadre de contrainte comme sur la figure 5.18.b).

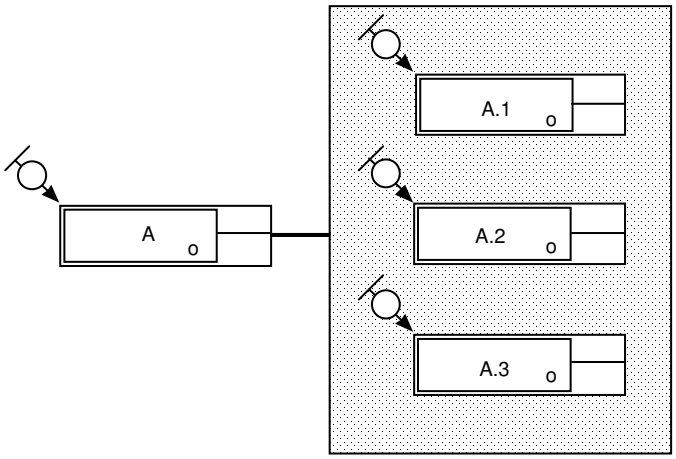


Figure 5.18.a : Une opération mère avec ses filles, ne subissant pas de contrainte explicite

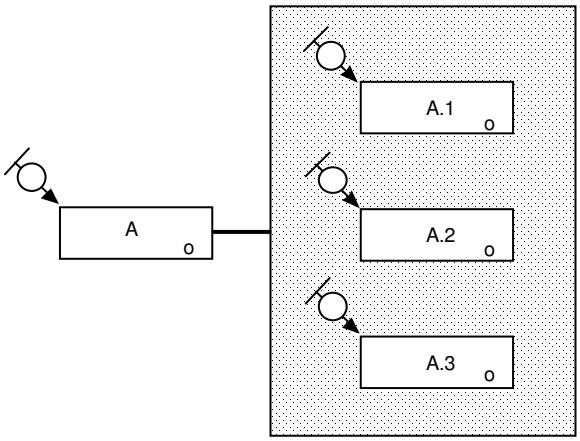


Figure 5.18.b : Représentation implicite de l'opération A de la figure 5.18.a

Tous les cas précédemment cités sont des restrictions de cas réels. En effet il est fréquent de trouver des opérations qui mêlent à la fois des opérations obligatoires et des opérations facultatives. Ceci ne remet nullement en cause les règles sur les contraintes énoncées avant puisque celles-ci ne s'appliquent que sur les opérations filles facultatives. Pour les opérations filles obligatoires, seules les contraintes basées sur le nombre de déclenchements seront possibles (elles le sont également pour les opérations filles facultatives). Prenons un exemple pour illustrer ce propos. La figure 5.19 implique les conséquences suivantes :

- l'opération "A" peut être déclenchée autant de fois que l'on veut et à chaque déclenchement on ne pourra exécuter au plus qu'une seule des deux opérations facultatives "A.2" et "A.5".

- à chaque déclenchement de l'opération "A", on devra obligatoirement exécuter les opérations filles "A.1", "A.3" et "A.4". D'après les règles vues au début de ce chapitre, cela implique que l'on peut déclencher ces trois opérations dans divers ordres : (1, 3, 4) - (1, 4 ,3) et (4, 1, 3). Ces trois opérations n'ayant pas de contrainte de déclenchement, l'utilisateur peut donc les déclencher à volonté.

- l'opération "A" sera considérée comme terminée à partir du moment où "A.1", "A.3" et "A.4" sont terminées.

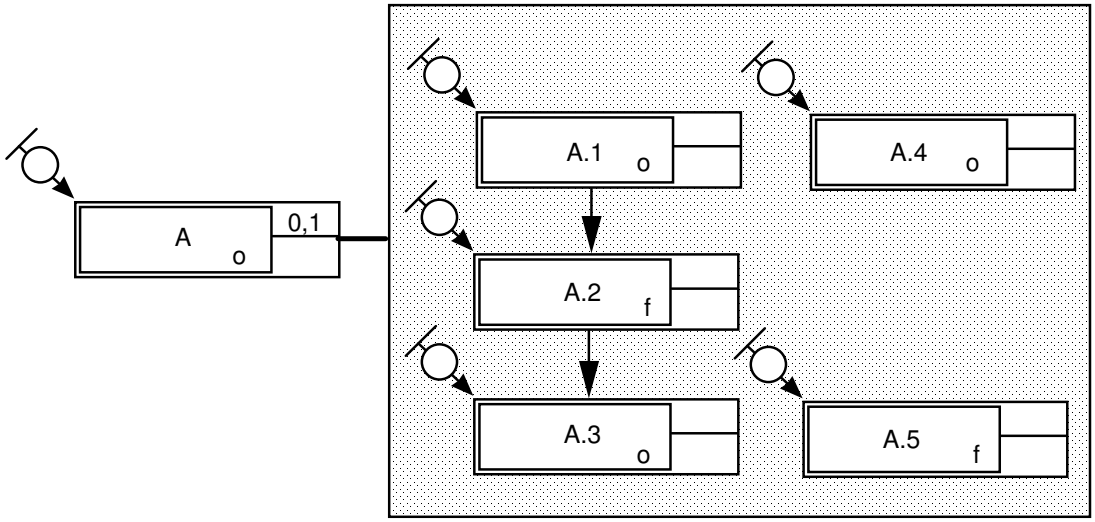


Figure 5.19 : Une opération mère et avec des opérations filles obligatoires et facultatives

Avec les contraintes sur les opérations filles facultatives, nous recréons les synchronisations ET, OU et OU EXCLUSIF. Il est donc possible d'utiliser l'une ou l'autre de ces représentations ou bien encore de passer de l'une à l'autre. Les figures suivantes (Figure 5.20.a, b et c) donnent les représentations pour le ET, le OU et le OU EXCLUSIF.

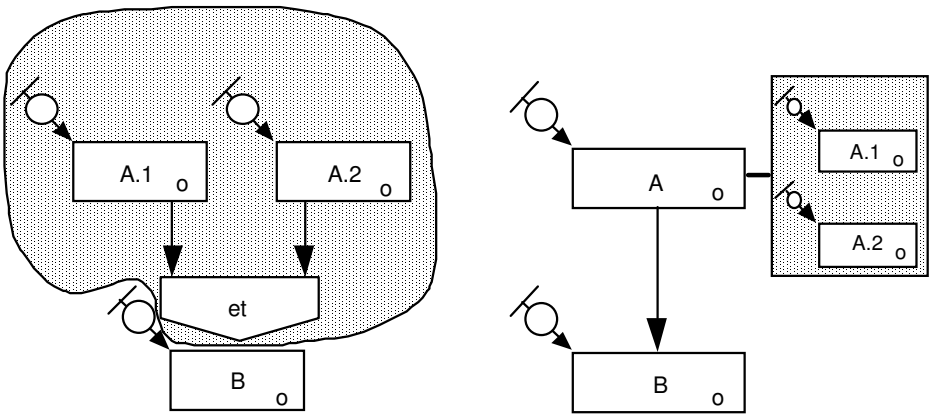


Figure 5.20.a : Représentation du ET avec Diane+. Le dessin de gauche montre une synchronisation avec un ET. Dans le dessin de droite (qui est équivalent à celui de gauche), une opération "A" contient les deux opérations filles "A.1" et "A.2" . Elle ne possède pas de contrainte et son type obligatoire implique l'exécution des deux opérations filles pour que l'opération mère soit considérée comme terminée.

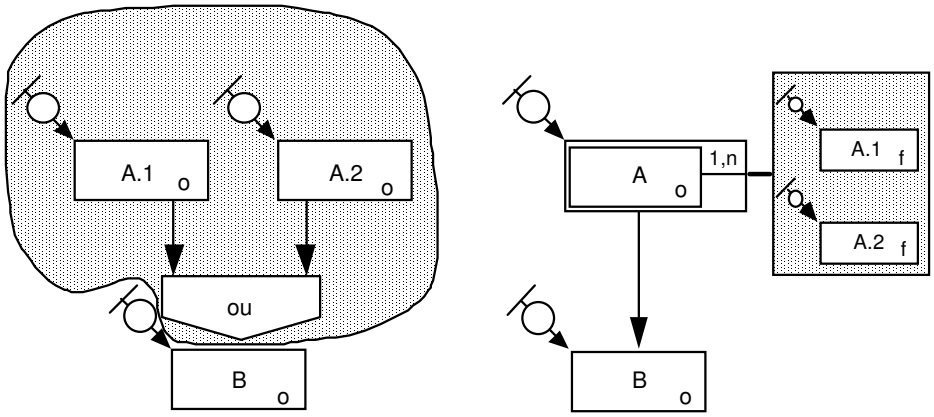


Figure 5.20.b : Représentation du OU avec Diane+. Le dessin de droite donne l'équivalent du "OU" représenté à gauche. L'opération "A" contient les deux opérations filles dont le type a été changé pour que l'on puisse appliquer une contrainte sur "A". Cette contrainte ("au moins une" opération fille) doit être validée pour que "A" soit considérée comme terminée.

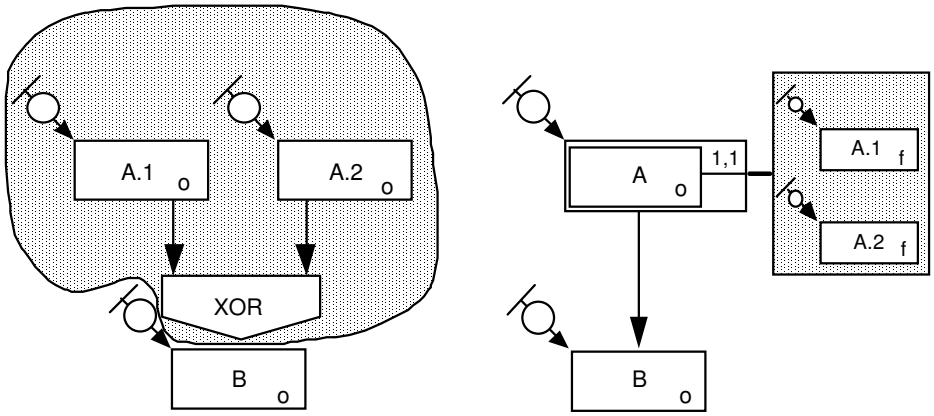


Figure 5.20.c : Représentation du XOR avec Diane+. Le dessin de droite donne l'équivalent du "OU EXCLUSIF" représenté à gauche. L'opération "A" contient les deux opérations filles dont le type a été changé pour que l'on puisse appliquer une contrainte sur "A". Cette contrainte ("une et une seule" opération fille) doit être validée pour que "A" soit considérée comme terminée.

Jusqu'à présent nous n'avons utilisé que des opérations à déclenchement utilisateur (interactives ou non). Il reste cependant possible d'utiliser des opérations à déclenchement automatique comme opérations filles. Ainsi la figure 5.21 implique que dès qu'on a lancé l'exécution de l'opération "A", le système exécute automatiquement les opérations "A.1" et "A.4" ce qui pourra provoquer par exemple l'affichage d'une boîte de dialogue et d'une fenêtre. Lorsque "A.1" est terminée, l'utilisateur a le choix entre déclencher "A.2" ou "A.5" (ou exclusif à cause de la contrainte 0,1). Dans les deux cas l'opération "A.3" lui est accessible. Utiliser des opérations automatiques comme opérations filles permet donc d'expliciter le travail concurrent entre des opérations. Le système général est bien entendu prévu pour fonctionner dans un environnement multi-tâches. Ceci signifie que l'on peut exécuter non seulement plusieurs opérations simultanément comme nous venons de le voir, mais également plusieurs procédures à la fois.

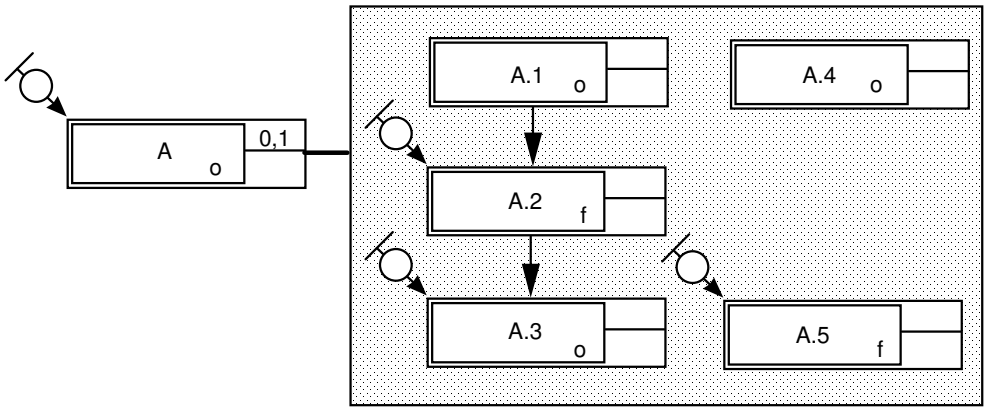


Figure 5.21 : Une opération mère avec des opérations filles travaillant concurremment

2.6. Intégration de l'utilisateur

Les caractéristiques de Diane+ que nous venons de voir prouve qu'elle intègre à la fois l'utilisateur et la répartition du contrôle entre l'homme et la machine contrairement à des méthodes telles que MAD. Cette intégration se retrouve à deux niveaux :

- *l'interaction de l'utilisateur sur l'application* par l'intermédiaire :

  - *du mode d'interaction des opérations*. Si on précise qu'une opération est interactive, cela implique que l'utilisateur devra intervenir au cours de l'exécution pour que l'opération puisse se dérouler. Au contraire, si une opération est automatique, cela signifie qu'on interdit à l'utilisateur d'intervenir au cours de l'exécution.

  - *du mode de déclenchement des opérations*. Si une opération possède un déclenchement utilisateur, cela implique que seul l'utilisateur peut la déclencher. Au contraire, si une opération ne possède pas de déclencheur, cela signifie qu'on interdit à l'utilisateur le déclenchement de l'opération.

  Dans les deux cas, on spécifie à la fois le permis (autorisation de déclenchement et d'interaction) et l'interdit (interdiction de déclenchement et d'interaction) ce qui donne des spécifications plus rigoureuses et plus faciles à gérer.

- *la latitude décisionnelle de l'utilisateur* par l'intermédiaire :

  - *du type des opérations*. Si une opération est facultative, on laisse à l'utilisateur le choix de l'exécuter ou non (aux contraintes près). Les opérations facultatives augmentent donc la latitude décisionnelle alors que les opérations obligatoires la restreignent. Là encore les deux types sont nécessaires pour exprimer l'intégration de l'utilisateur en explicitant le permis et l'interdit.

  - *des liens de précédence sur les opérations*. Relier deux opérations obligatoires par un lien de précédence permanente implique que l'utilisateur devra impérativement exécuter ces opérations dans l'ordre donné par le lien. A l'inverse si deux opérations obligatoires ne sont pas liées par une précédence, cela implique que l'utilisateur choisit l'ordre de déclenchement. Par ailleurs nous avons vu qu'il est possible de lier des opérations obligatoires avec des opérations facultatives. Ceci constitue une restriction

pour les opérations facultatives qui ne peuvent alors être déclenchées qu'entre les deux opérations qui l'encadrent. La présence de liens de précédence est donc synonyme de restriction de la latitude décisionnelle alors que l'absence de liens implique une large latitude décisionnelle.

- *des contraintes* qui peuvent s'exprimer sur le nombre de déclenchements autorisés et sur le nombre de sous-opérations facultatives à exécuter. L'absence de contraintes signifie que l'utilisateur a un choix total sur ces deux plans, donc forte latitude décisionnelle. Au contraire la présence de contraintes restreint de manière explicite la latitude décisionnelle.

Diane+ permet donc de spécifier explicitement la latitude décisionnelle **maximale** dont dispose l'utilisateur. Ceci nous permet de contrôler de façon sûre le dialogue homme-machine comme nous le verrons tout au long de cette thèse.

3. Gestion des tâches par poste de travail

La conception (au sens large) d'une application interactive est généralement orientée en fonction du public visé. Cela suppose que l'on a une connaissance précise des types d'utilisateurs existants ou susceptibles d'utiliser l'application. Cela suppose également que l'on connaît les postes de travail pour lesquels l'application est conçue. A chacun de ces postes correspond une procédure minimale ainsi qu'un ou plusieurs types d'utilisateurs pour lesquels nous aurons une procédure prévue et plusieurs procédures effectives. Par exemple le poste de travail correspondant à une secrétaire aura une seule procédure minimale, mais les personnes qui travailleront à ce poste pourront avoir des niveaux différents allant par exemple de "novice" à "expérimenté". Les procédures prévues permettront de représenter les niveaux prédéfinis (expert, débutant, occasionnel, etc.) et les procédures effectives permettront de représenter les niveaux intermédiaires ou non prévus. Nous allons à présent détailler ces différentes procédures.

3.1. Procédure prévue

La procédure prévue correspond au déroulement attendu pour un poste de travail et un type d'utilisateur pour atteindre un but de manière satisfaisante. Elle est généralement exprimée par les responsables des postes de travail lors de l'analyse de l'existant ou des spécifications[69]. Elle est constituée d'un ensemble d'opérations qui sont nécessaires ou utiles pour atteindre le but. Par ailleurs elle contient un nombre généralement important de précédences permanentes.

La procédure prévue est surtout utile pour l'aide et l'apprentissage. Dans le premier cas elle fournit la liste ordonnée des opérations à réaliser pour mener à bien une tâche, dans le second cas elle permet de libérer l'utilisateur du problème du choix de l'ordre d'exécution des opérations. L'exemple de la figure 5.22 pourrait fournir l'aide suivante (on suppose que l'utilisateur n'a encore exécuté aucune opération et que le but à atteindre est de gérer les stocks) :

> Pour gérer les stocks, vous devez d'abord exécuter l'opération "A". Celle-ci terminée, vous avez la possibilité d'exécuter l'opération "B". Dans tous les cas vous devrez ensuite exécuter l'opération "C". Vous devrez également exécuter l'opération "F" que le système déclenchera pour vous. Au cours de ces exécutions, le système vous

[69] La procédure prévue peut également être le résultat de l'interprétation de recueils auprès des responsables si ceux-ci ne sont pas capables de l'exprimer directement.

demandera des informations pour mener à bien les opérations “A”, “C” et “F”. Lorsque ces trois opérations seront terminées, vous aurez achevé la gestion des stocks. Notez également que pendant que vous travaillerez, le système exécutera les opérations “D” et “E” et que vous avez la possibilité d’exécuter l’opération “G” à tout moment.

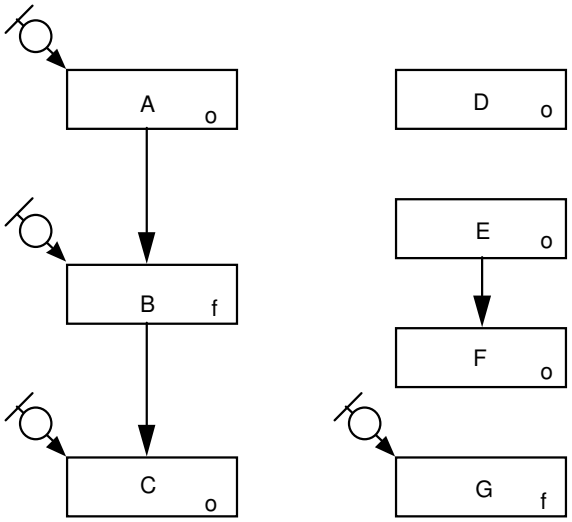


Figure 5.22 : Exemple de procédure prévue

Au niveau de l’apprentissage, on peut imaginer un didacticiel qui expliquerait d’abord ce que l’on cherche à réaliser (ici gérer les stocks), puis comment le réaliser, c’est-à-dire le séquencement des opérations. Il pourrait également préciser que certaines opérations ne nécessitent pas l’intervention de l’utilisateur alors que d’autres sont toujours à sa disposition. Ceci étant expliqué, le didacticiel simule l’exécution de “D” et “E”, puis demande à l’utilisateur de déclencher “A”. Si tout se passe bien, il pourra lui proposer d’abord de déclencher “C”, et ensuite de déclencher “B” puis “C” pour montrer les deux choix possibles. On peut imaginer également que le didacticiel rappelle, à intervalles réguliers ou non, que l’utilisateur peut déclencher “G” quand il le souhaite. Certains didacticiels tels que ceux d’Excel ou de Works fonctionnent sur un principe semblable.

3.2. Procédure effective

On peut différencier deux types de procédures effectives :

- *les procédures effectives pour un type prédéfini d’utilisateur*. Puisque Diane+ est bâtie sur la planification hiérarchique et que celle-ci est fonction des types d’utilisateurs et des postes de travail, il est possible de prévoir des séquencements correspondant à ces types. Ainsi on bâtira une procédure effective pour des utilisateurs occasionnels (pour des utilisateurs débutants on utilisera plutôt la procédure prévue) ou bien encore pour des utilisateurs qui ont un niveau de confidentialité restreint. Ces procédures dépendent également des postes de travail ce qui signifie que pour un même but et pour des utilisateurs de même niveau, deux postes de travail distincts (par exemple le secrétariat et la direction d’une société) présenteront deux procédures effectives différentes.
  Ce type de procédure effective, bien que techniquement réalisable, n’est pourtant pas très intéressante car elle se rapproche énormément des procédures prévues qui sont également basées sur les types d’utilisateurs et sur les postes de travail. Dorénavant nous emploierons le terme “effective” uniquement dans le sens donné ci-dessous.

- *les procédures effectives pour un utilisateur particulier*. Elles sont utiles pour optimiser des sessions de travail. Deux cas peuvent se présenter :

  - un utilisateur régulier exécute généralement la même séquence d'opérations. Afin de gagner du temps et de réduire sa charge de travail, il peut souhaiter enregistrer cette séquence. Ceci se produit déjà dans des logiciels tels que Excel ou Word où ce type d'enregistrement s'appelle une *macro*. Avec Diane+, il est tout à fait possible de recréer ces macros soit à la demande de l'utilisateur (celui-ci construit lui-même ses macros de la même façon qu'avec Excel) soit de manière automatique (le système est capable de déduire qu'un utilisateur exécute toujours les mêmes enchaînements). Ce deuxième cas est déjà à l'étude depuis quelques années et des maquettes de démonstration tels que Eager [CYPHER 91] utilisent ce principe.

  - un utilisateur occasionnel n'est satisfait ni de la procédure prévue ni de la procédure minimale. Il peut également souhaiter enregistrer son propre séquencement qui pourra lui être proposé par la suite automatiquement lors de sa connexion avec le système. Dans ce cas l'enregistrement se fera obligatoirement après la demande explicite de l'utilisateur et il sera créé entièrement par ce dernier.

Si l'on reprend la procédure prévue de l'exemple précédent (Figure 5.22), une des procédures effectives possibles serait la suivante (Figure 5.23) :

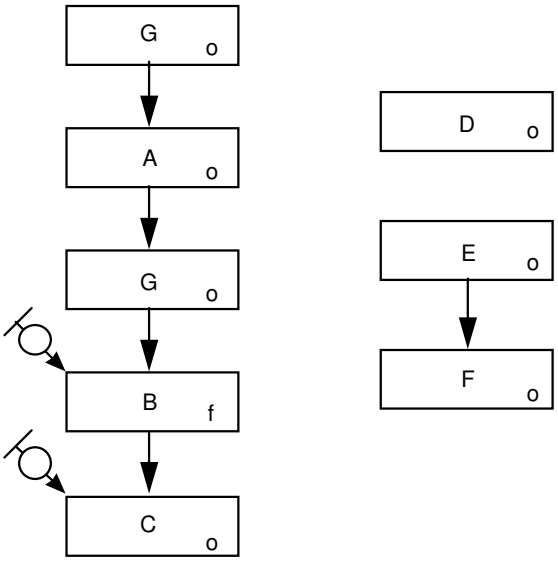


Figure 5.23 : Une procédure effective basée sur la procédure prévue de la figure 5.22.

On constate que l'utilisateur a choisi d'utiliser plusieurs fois l'opération "G" de manière obligatoire[70] et avec un déclenchement automatique par la machine. Il a donc restreint sa latitude décisionnelle sur cette opération de même que sur l'opération "A" à laquelle il a retiré le déclenchement utilisateur. Ainsi lorsque cette procédure sera exécutée, le système lancera l'exécution des opérations "D", "E", et "G" automatiquement. Lorsque "E" sera terminée, il démarrera "F", de même qu'à la fin de la première exécution de "G" il exécutera "A", puis de

[70] S'il avait laissé le type facultatif, l'opération "G" n'avait plus de sens car elle ne possède plus de déclencheur.

nouveau “G” avant de redonner la main à l’utilisateur pour qu’il choisisse entre exécuter “B” puis “C” ou exécuter “C” directement.

3.3. Procédure minimale

La procédure minimale correspond au noyau dur des procédures effectives et prévues. Elle est en général extraite de ces dernières car il est rare que les utilisateurs ou les responsables du futur système soient capables de la donner immédiatement. La procédure minimale contient d’une part le contrôle minimum obligatoire sur les opérations et d’autre part le maximum d’opérations dont l’utilisateur dispose pour le but associé. Elle fait apparaître les précédences correspondant aux règles de gestion et d’organisation, contrairement aux procédures prévues et effectives qui utilisent également les précédences pour augmenter l’utilisabilité de l’application.

Stocker dans une procédure particulière le contrôle minimal sur le dialogue est intéressant sur deux points. Tout d’abord cela permet de visualiser ce contrôle minimal d’une façon plus précise que s’il était directement intégré aux procédures effectives et prévues. On peut donc à tout moment, dans la spécification et dans la conception, modifier ce contrôle minimal. Le deuxième point intéressant est que ce noyau dur du contrôle nous permet de valider les procédures effectives et prévues. En effet celles-ci doivent être cohérentes avec la procédure minimale ce qui signifie qu’elles peuvent restreindre la latitude décisionnelle, mais **jamais** l’étendre. Les conséquences immédiates sont résumées dans le tableau suivant :

Si dans la procédure minimale on a :	**Alors dans les procédures prévue et effective on a :**
une opération obligatoire	une opération obligatoire
une opération facultative	une opération facultative ou obligatoire
une opération automatique	une opération automatique
une opération interactive	une opération automatique ou interactive[71]
une opération sans déclencheur	une opération sans déclencheur
une opération avec déclencheur	une opération avec ou sans déclencheur
une précédence permanente entre deux opérations	une précédence permanente entre deux opérations
pas de lien entre deux opérations	pas de lien entre les deux opérations ou une précédence permanente entre les deux opérations

Soit la procédure minimale ci-dessous (Figure 5.24) :

[71] Le passage d’une opération interactive en opération automatique sera exceptionnel. En effet si le concepteur a prévu une interactivité (par exemple dans une saisie), il paraît difficile d’exécuter cette opération de manière équivalente sans cette interactivité.

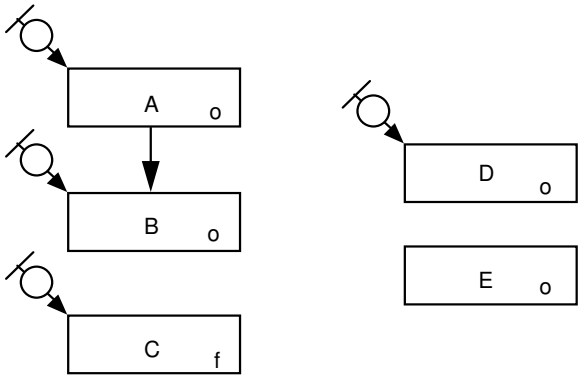


Figure 5.24 : Un exemple de procédure minimale

- la figure 5.25 donne un exemple de procédure (prévue ou effective) qui respecte cette procédure minimale. L'utilisateur a décidé de restreindre sa latitude décisionnelle en ajoutant une contrainte de déclenchement sur "A", en supprimant le déclenchement utilisateur sur "B" et "D", et en insérant de manière obligatoire et automatique l'opération "C".

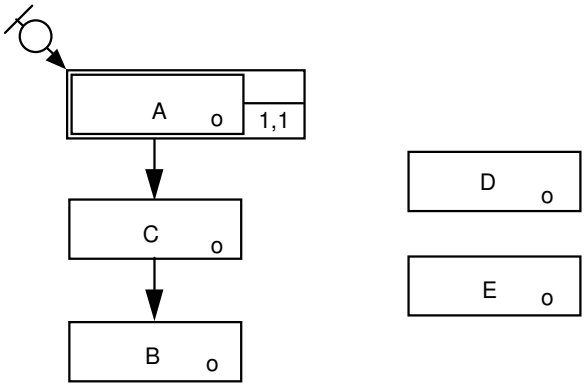


Figure 5.25 : Un exemple de procédure respectant la procédure minimale de la figure 5.24 précédente

- la figure 5.26 représente une procédure qui ne respecte pas cette procédure minimale car l'utilisateur a transformé une opération obligatoire en facultative ("D"), a supprimé la précédence permanente entre "A" et "B", et a ajouté un déclenchement utilisateur sur une opération ("E").

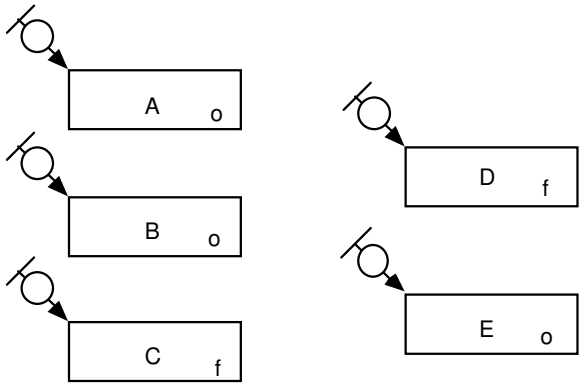


Figure 5.26 : Un exemple de procédure ne respectant pas la procédure minimale de la figure 5.24

Intégrer dans la procédure minimale toutes les opérations facultatives dont l'utilisateur peut avoir besoin pour réaliser une tâche nous permet de limiter son choix lors de la session de travail. De même que précédemment, l'utilisateur a la possibilité de réduire ce choix sur le nombre d'opérations facultatives disponibles et sur leur exécution (il peut les rendre obligatoires ou les lier à d'autres opérations). En procédant de cette façon, nous sommes certains que l'utilisateur ne pourra pas sortir des limites qui lui ont été ainsi imposées. Il est bien évident que le problème majeur est de lui fournir effectivement les bonnes opérations.

La figure 5.27.a montre une procédure minimale avec un choix relativement large au niveau des opérations facultatives. La figure 5.27.b montre une procédure effective, respectant cette procédure minimale, où l'utilisateur a restreint sa latitude décisionnelle.

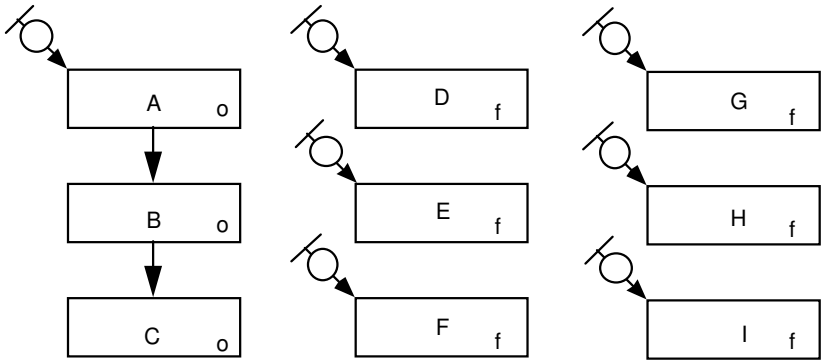


Figure 5.27.a : Une procédure minimale avec un choix important au niveau des opérations facultatives

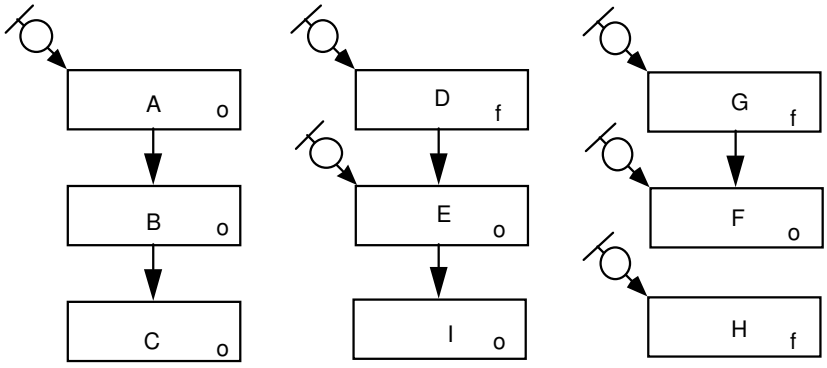


Figure 5.27.b : Une procédure respectant la procédure minimale de la figure 5.27.a et restreignant la latitude décisionnelle au niveau des opérations facultatives

Ces deux approches dans la procédure minimale nous permettent de borner la latitude décisionnelle. Au minimum l'utilisateur devra exécuter toutes les opérations obligatoires en respectant les contraintes d'enchaînement, de déclenchement, etc., d'autre part l'utilisateur ne pourra jamais exécuter des opérations autres que celles qui se trouvent dans la procédure minimale.

4. Exemples de spécifications

Nous présentons dans ce paragraphe deux exemples de procédures Diane+ : le premier reprend l'exemple de la messagerie électronique traité dans [NORMAND 92a] et le second concerne la gestion d'une bibliothèque. Nous voulons, par ces deux exemples, montrer que le formalisme Diane+ d'une part peut s'appliquer aussi bien à un niveau général (exemple de la messagerie) qu'à un niveau de détail fin (exemple de la bibliothèque), et d'autre part procure certains avantages que ne fournissent pas la plupart des modèles (par exemple qui déclenche les opérations, quelle est la liberté d'action minimale et maximale de l'utilisateur, etc.).

4.1. Une messagerie électronique

Outre la notion conceptuelle des données, le travail de V. Normand [NORMAND 92a] manipule le concept de tâche. Pour cela, V. Normand utilise un formalisme particulier qui présente les tâches sous forme d'arbre. La figure 5.28 représente l'arbre des tâches d'une messagerie électronique. Cet arbre montre les choix (par exemple entre Traiter Boîte Aux Lettres et Envoyer Message) ainsi que les contraintes d'enchaînement proposées à l'utilisateur (par exemple Entrer Nom *puis* Entrer Password). Sur cet arbre de tâches se superpose la notion de "Workspace" (ou Espace de Travail[72]) qui permet de définir "des activités virtuelles offrant les éléments nécessaires à la réalisation d'une ou plusieurs tâches utilisateur". Ces espaces de travail sont en relation avec des "Perspectives" qui "définissent des vues restrictives sur les données et les opérations d'un concept de l'application". Dans le workspace Ecriture de la messagerie, le concept manipulé est le message et la perspective correspondante autorise la modification du sujet et du texte du message.

[72] V. Normand associe les Espaces de travail aux procédures effectives de Diane en précisant que les Espaces découlent des procédures effectives. En effet lors de l'analyse de ces procédures, on extrait les tâches qui sont exécutées le plus souvent en parallèle. Chacune de ces tâches donne lieu ensuite à un Espace de travail distinct.

Un arbre de tâches tel que celui de la messagerie peut être traduit sous la forme de procédures et d'opérations Diane+. La figure 5.29 montre un exemple de traduction. Si l'on compare les deux approches, on constate qu'entre la première et la seconde :

- la *séquencialité* est traduite par les précédences. Par exemple lorsque l'utilisateur veut utiliser la messagerie, il doit se connecter. Pour cela il doit d'abord lancer le système *puis* s'identifier. Avec Diane+, nous représentons cet enchaînement grâce à une précédence entre ces deux opérations.

- la *suite non ordonnée* est représentée par l'absence de précédence et par l'utilisation d'opérations obligatoires. Par exemple pour réaliser la tâche Définir un message, l'utilisateur doit entrer le texte et le sujet du message. Ces deux opérations devant être réalisées, nous leur donnons un type obligatoire. Étant donné que l'arbre ne spécifie pas d'enchaînement entre ces deux opérations, nous n'ajoutons donc pas de précédence.

- la *boucle* est représentée de manière implicite. Nous avons vu précédemment (cf § 2.4.1) qu'une opération Diane+ qui possède un déclencheur et qui ne subit pas de contrainte de déclenchement pour elle-même, signifie que l'utilisateur peut l'exécuter autant de fois qu'il le veut. Traiter un message en est un exemple.

- l'*alternative* est représentée par des opérations filles facultatives et une opération mère avec contraintes sur les opérations filles. La tâche Traiter un message par exemple comporte deux opérations en séquence, chacune d'entre elles comportant une alternative. Prenons le cas de la première alternative Choisir un message ou Prendre le premier message non lu. Avec Diane+, nous regroupons ces deux opérations dans une opération mère Choix de message qui correspond à la racine de l'alternative dans l'arbre des tâches. La contrainte de déclenchement ("une et une seule" opération fille) signifie bien d'une part que l'on devra choisir entre l'une des deux opérations, et d'autre part que ce choix est obligatoire à cause du type obligatoire de l'opération mère.

- l'*alternative non stricte* est représentée par des opérations facultatives. Par exemple la tâche Travailler permet de choisir entre Traiter boîte aux lettres et Envoyer message, sans que ce choix soit imposé. Cela dénote que l'utilisateur peut faire l'une ou l'autre des tâches, ou bien les deux ou encore aucune de ces tâches. Cette caractéristique correspond exactement au type facultatif des opérations Diane et Diane+. L'opération mère Travailler contient donc deux opérations filles facultatives sur lesquelles elle n'impose aucune contrainte.

- le *parallélisme*, géré dans l'arbre des tâches grâce aux boucles, est traduit par l'intermédiaire des déclencheurs et de l'absence de contrainte. Les boucles permettent en effet d'exécuter plusieurs fois la même opération en parallèle, mais il est également possible d'exécuter en parallèle plusieurs boucles. Dans la traduction avec Diane+, nous retrouvons ce parallélisme dans les opérations à déclencheur. Par exemple l'arbre des tâches fournit un parallélisme pour Traiter un message. Nous retrouvons cette caractéristique dans l'opération de même nom de la procédure Diane+. Cette opération étant à déclenchement utilisateur et sans contrainte de déclenchement, on peut l'exécuter autant de fois que l'on veut. De même on peut lire ou encore répondre à plusieurs messages en même temps.

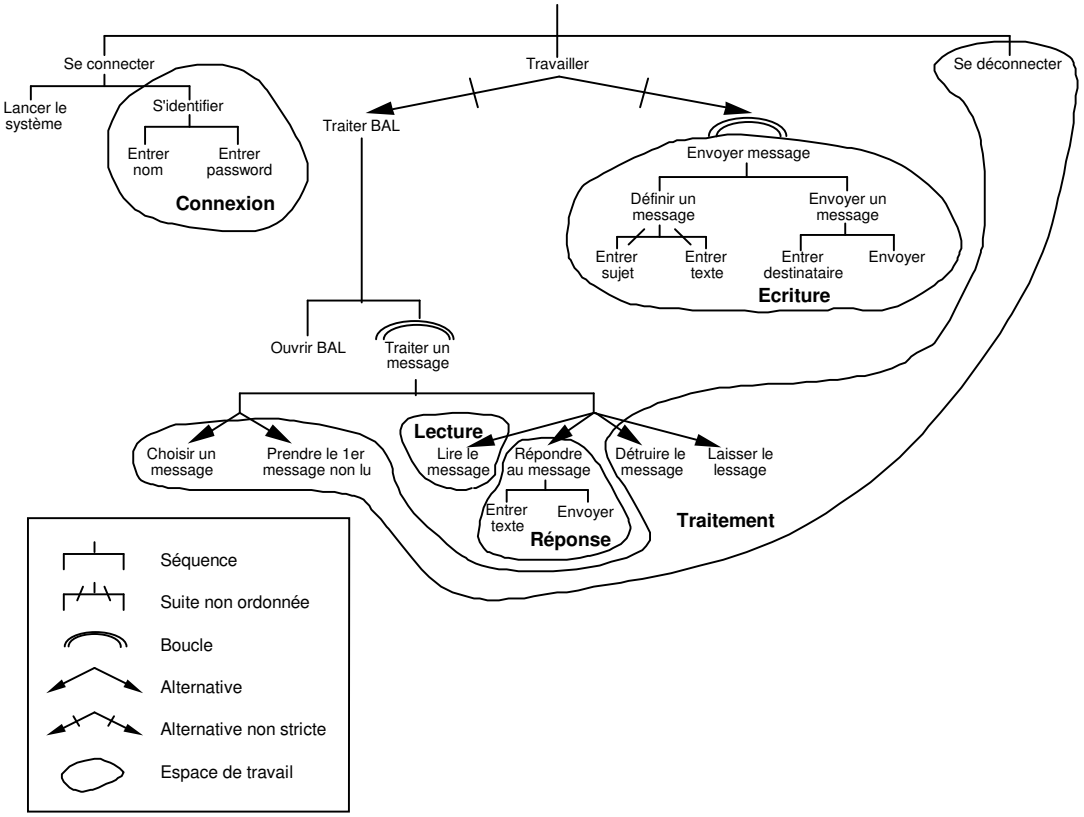


Figure 5.28 : L'arbre des tâches pour la messagerie électronique (extrait de [NORMAND 92a])

Le formalisme Diane+ permet également de traduire :

- *l'imbrication des opérations*. La tâche Se connecter contient une séquence de deux opérations qui sont Lancer le système et S'identifier. Cette dernière contient également une séquence de deux opérations (Entrer nom et Entrer password). Nous représentons ces imbrications par des imbrications d'opérations Diane+. Toutes ces opérations devant être exécutées, elles ont donc toutes un type obligatoire. Ainsi pour que Se connecter soit terminée, il faut que Lancer le système et S'identifier le soient, et pour que S'identifier soit terminée, il faut que Entrer nom et Entrer password le soient également.
  On peut remarquer que l'arbre des tâches ne fait qu'une distinction sommaire entre les tâches déclenchées par l'utilisateur et celles déclenchées par la machine. Ainsi toute tâche terminale qui n'est pas dans un Espace de travail (par exemple Ouvrir BAL) est exécutée automatiquement par le système. Dans la traduction Diane+, nous avons fait figurer un exemple de cette distinction (Lancer le système, S'identifier, etc., n'ont pas de déclencheur). Cela signifie que la machine lance seule l'exécution et qu'ainsi, lorsque l'utilisateur déclenche Se connecter, le contrôleur de dialogue déclenche automatiquement Lancer le système qui, une fois terminée, lance automatiquement (par l'intermédiaire du contrôleur) S'identifier.

- *les opérations par défaut*. Ce type d'opération n'existe pas dans l'arbre des tâches. Avec Diane+, dans l'opération Choix de message, l'opération Prendre le premier message non lu est l'opération par défaut. Cela signifie que si l'utilisateur appuie sur la touche Entrée après le déclenchement de Choix de message, l'opération Prendre le premier message non lu sera automatiquement exécutée. Au niveau externe, cela se traduira par exemple par la mise en vidéo inverse du premier élément de la liste des messages.

- *le nombre de déclenchements* d'une opération ou celui de ses opérations filles. Nous avons déjà vu plus haut que les contraintes de déclenchement permettent de traduire les choix dans le formalisme Diane+. Nous parlons ici d'un autre type de contraintes. C'est le cas par exemple pour l'opération S'identifier ou bien encore pour l'opération Envoyer un message qui stipule que l'on ne doit entrer qu'un seul destinataire pour envoyer un message. Plus précisément, on ne peut effectuer qu'une seule fois l'opération Entrer destinataire. Cela n'interdit pas que cette opération puisse permettre la saisie d'une liste de destinataires au lieu d'un seul, auquel cas cette liste sera assimilée par la suite à un seul destinataire. Ainsi l'opération suivante (Envoyer) devra être à même de traiter cette liste, c'est-à-dire d'envoyer le même message à chacun des destinataires. Les contraintes de déclenchement sur des opérations n'existent pas dans le travail de V. Normand. L'emploi des contraintes avec Diane+ reste facultatif ; il permet une restriction de la latitude décisionnelle de l'utilisateur, ce que n'autorise pas l'arbre des tâches.

- *la "nature" des opérations*. L'arbre des tâches ne fait la distinction qu'entre les tâches exécutées par l'homme et celles exécutées par la machine. Ceci correspond aux opérations sans déclencheur de Diane+. Par contre il semble que l'arbre des tâches, contrairement à Diane+, ne distingue pas les tâches interactives des tâches non-interactives. De même rien n'est dit en ce qui concerne l'obligation d'exécuter les tâches. Doit-on toutes les exécuter ? L'alternative non stricte remédie localement à ce problème puisque l'on peut exécuter zéro, une ou plusieurs des tâches proposées, ce qui équivaut à avoir des opérations Diane+ facultatives sans contrainte. Diane+ est donc plus puissante à ce niveau puisqu'elle permet des combinaisons plus nombreuses (opérations obligatoires ou facultatives et pouvant contenir des contraintes d'exécution de n opérations parmi m).

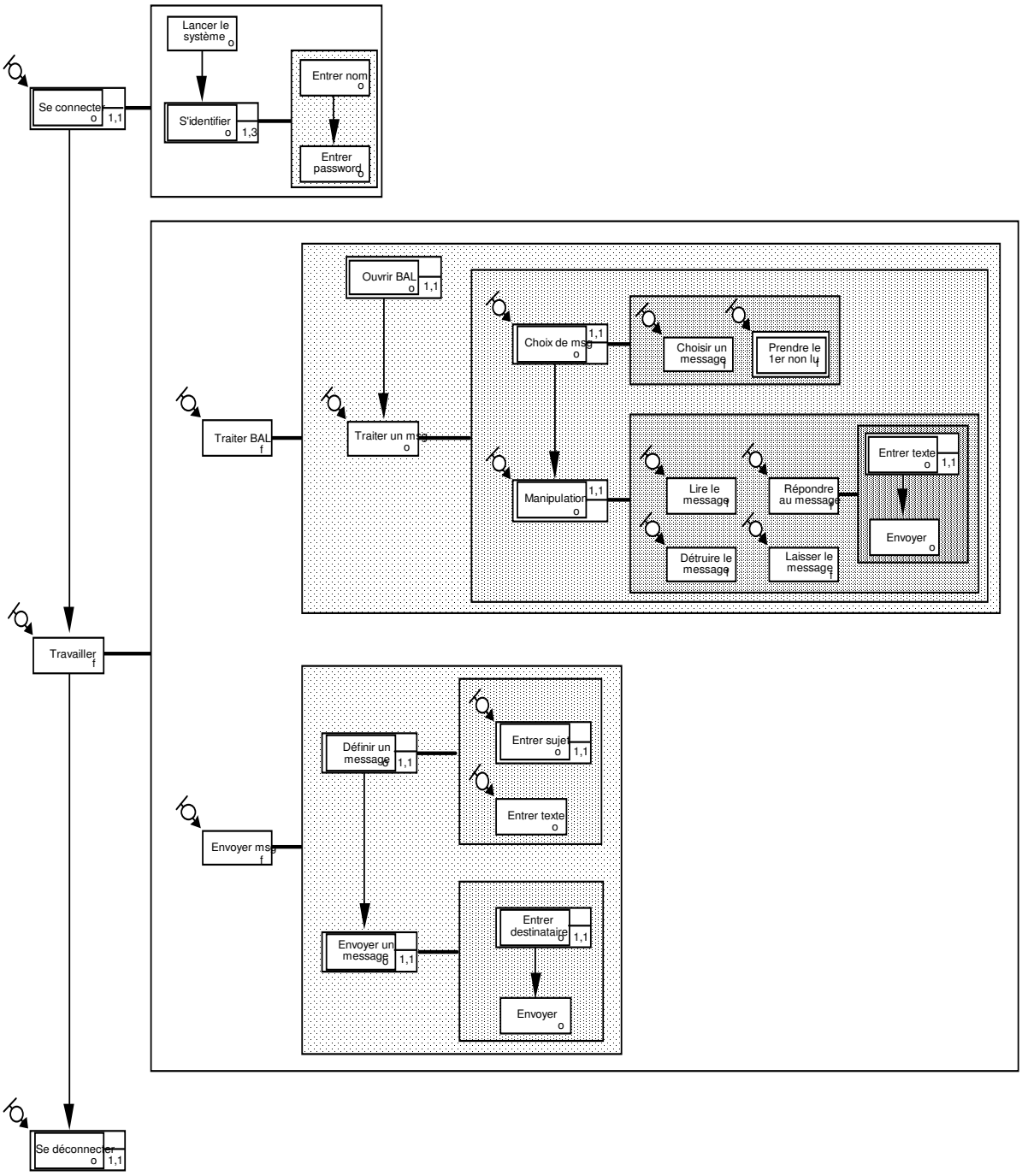


Figure 5.29 : Représentation de la messagerie électronique avec Diane+

La figure 5.30 propose une autre représentation pour cette messagerie. Dans cette spécification, l'utilisateur dispose des mêmes opérations que précédemment, mais nous les avons réparties directement dans l'optique Diane+. Par ailleurs la boîte aux lettres n'est ouverte qu'une fois au début de la connexion plutôt qu'à chaque traitement sur la boîte aux lettres comme précédemment (cette ouverture est exécutée automatiquement par le système). Enfin la connexion est faite automatiquement à l'entrée de la procédure.

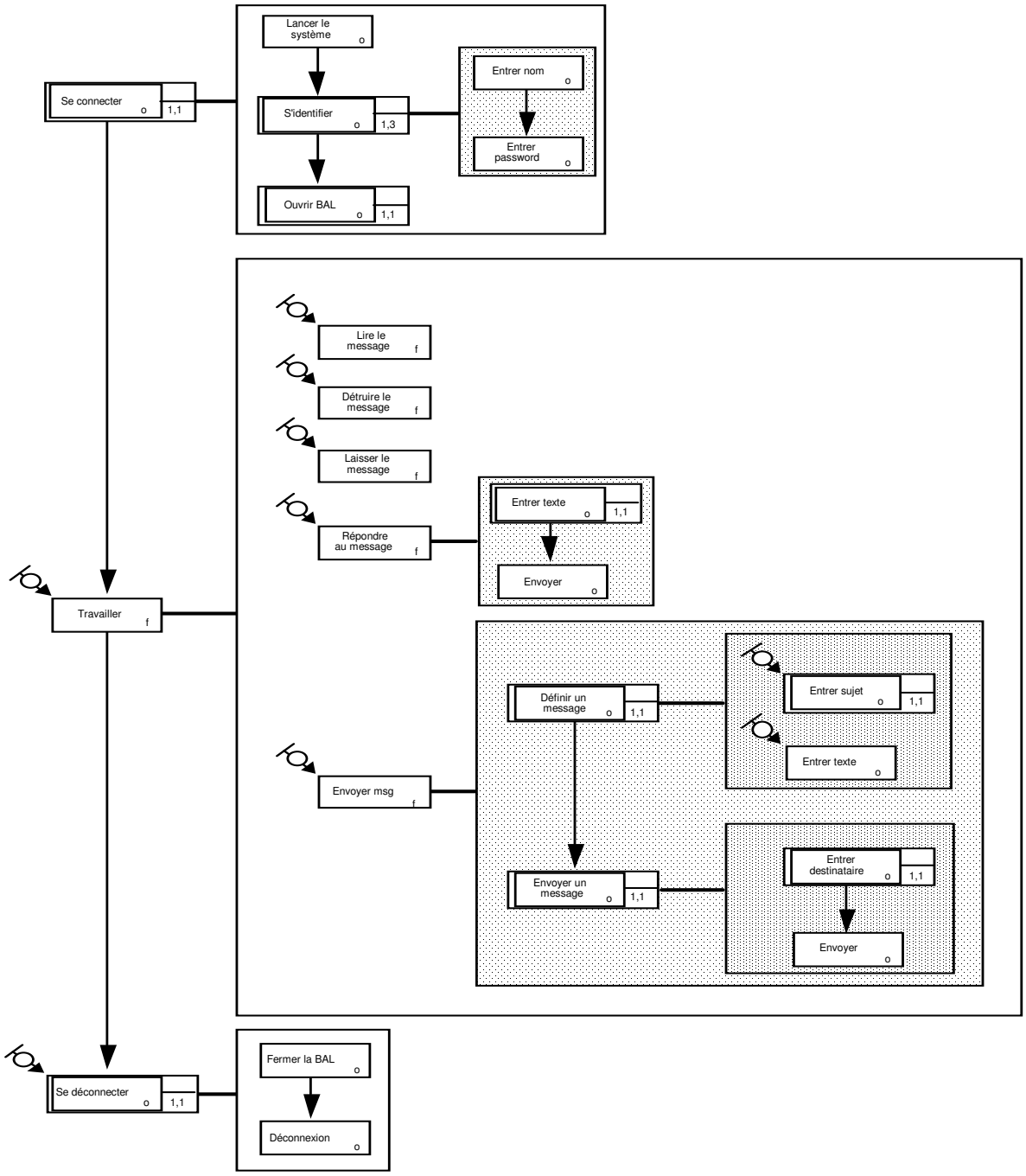


Figure 5.30 : Représentation simplifiée de la messagerie électronique de la figure 5.29

4.2. Gestion d'une bibliothèque

Nous nous intéressons ici aux procédures de prêts et de retours de prêts dans une bibliothèque. Notre objectif est de montrer que Diane+ permet de préciser finement le comportement de l'application vis-à-vis de l'utilisateur. Nous affinerons donc les opérations jusqu'à un niveau quasi élémentaire.

a. Le Prêt

Pour qu'un prêt soit possible, l'utilisateur (par exemple le documentaliste) doit avoir en sa possession la référence ou le code du livre, ainsi que le code de la personne qui désire faire le prêt (l'abonné). Ces deux conditions étant réunies, le documentaliste devra saisir la date du prêt ainsi que celle du retour prévu.
La figure 5.31 représente la procédure minimale Prêt. Le détail de l'opération Enregistrer un prêt est présenté plus loin (Figure 5.33).

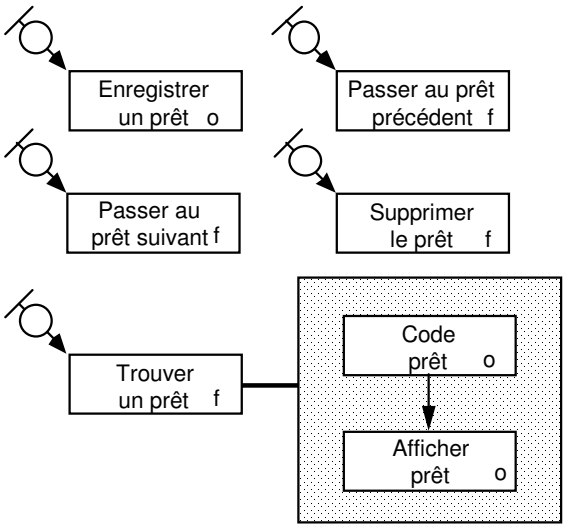


Figure 5.31 : Procédure minimale de Prêt

La caractéristique principale de cette procédure est de ne pas être obligatoire[73]. Le prêt est en effet à la disposition du documentaliste, mais rien ne l'oblige à l'exécuter. Dans le cas où cela se produit, seule l'opération Enregistrer un prêt devra au minimum être exécutée.
On peut remarquer que la procédure minimale contient des opérations complémentaires à l'opération Enregistrer un prêt. Comme nous l'avons vu précédemment (cf § 3.3), la procédure minimale doit contenir toutes les opérations qui peuvent s'avérer nécessaires ou utiles à l'utilisateur pour mener à bien sa tâche. Nous avons donc ajouté des opérations permettant de supprimer le prêt affiché, de trouver un prêt particulier, et d'afficher le prêt suivant ou précédant celui affiché. Toutes ces opérations sont facultatives et sans contrainte ; elles sont donc constamment à la disposition de l'utilisateur.

Nous avons fait figurer le détail de Trouver un prêt au même niveau que l'opération elle-même à titre d'exemple. Nous aurions pu tout aussi bien la détailler à part comme nous l'avons fait pour Enregistrer un prêt. Le détail de cette opération nous apprend que lorsque l'utilisateur veut trouver un prêt (c'est-à-dire lorsqu'il déclenche Trouver un prêt), le système l'oblige à saisir le code du prêt qu'il cherche, puis ceci étant terminé, le système affiche le prêt correspondant. Si nous voulions que la recherche se fasse aussi bien sur le code du prêt, que sur celui du livre ou de l'abonné, l'opération Trouver un prêt aurait été représentée comme sur la figure 5.32. Sur cette figure, les critères de recherche sont exclusifs. Si l'on désire qu'ils soient complémentaires, il suffit de modifier la contrainte au niveau de l'opération Saisie code (par exemple "au plus" deux critères).

[73] Cette caractéristique n'apparaît pas sur la figure, mais apparaîtrait sur le dessin représentant la procédure ou le but associé Gérer les prêts. Il faut faire attention à ne pas confondre le type de la procédure (facultatif ici) avec celui de l'opération Enregistrer un prêt (obligatoire ici).

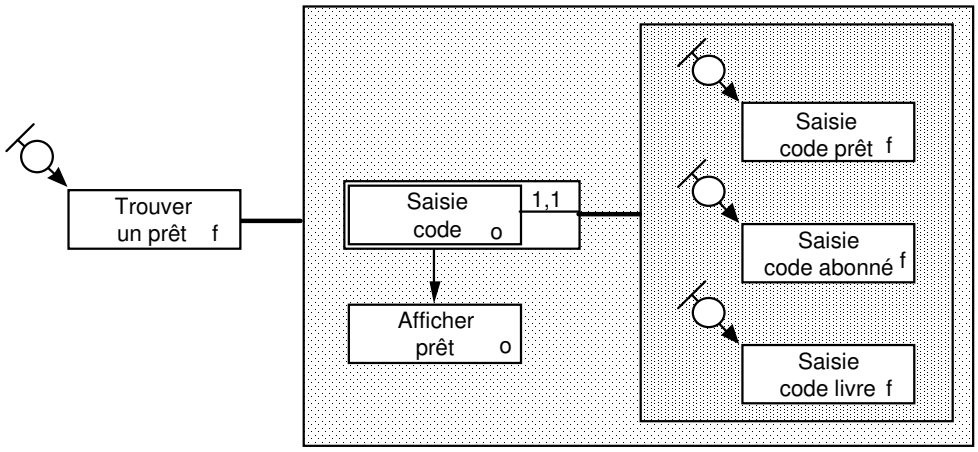


Figure 5.32 : Détail de l'opération "Trouver un prêt " fonctionnant sur trois critères exclusifs de recherche

La figure 5.33 détaille l'opération Enregistrer un prêt de la procédure minimale. Cette décomposition nous révèle que :

- l'utilisateur peut indifféremment commencer par saisir le code de prêt, le code de l'abonné ou l'ouvrage, et de plus qu'il n'existe pas d'enchaînement entre ces trois opérations,
- la saisie du code de l'abonné implique l'affichage des renseignements sur l'abonné (nom et téléphone),
- le choix de l'ouvrage peut se faire par la saisie de son code ou (exclusif) de son titre. Lorsque l'utilisateur choisit cette dernière alternative, le système lui propose automatiquement la liste des titres présents dans la bibliothèque ; l'utilisateur n'a plus alors qu'à choisir un de ces titres et le système referme alors automatiquement la fenêtre avant de "rendre la main" à l'opération Saisie ouvrage,
- le choix d'un ouvrage implique d'une part son affichage, c'est-à-dire l'affichage de son code, puis de son titre, de sa langue et de sa nouveauté (i.e. si ce livre est dans la bibliothèque depuis moins de 15 jours), et d'autre part l'affichage de la date du prêt. Cette date est supposée être celle du jour de la saisie. Nous aurions pu également présenter l'opération Afficher date du prêt de telle sorte qu'on affiche automatiquement la date du jour et que l'utilisateur puisse corriger cette date par défaut (Figure 5.34),

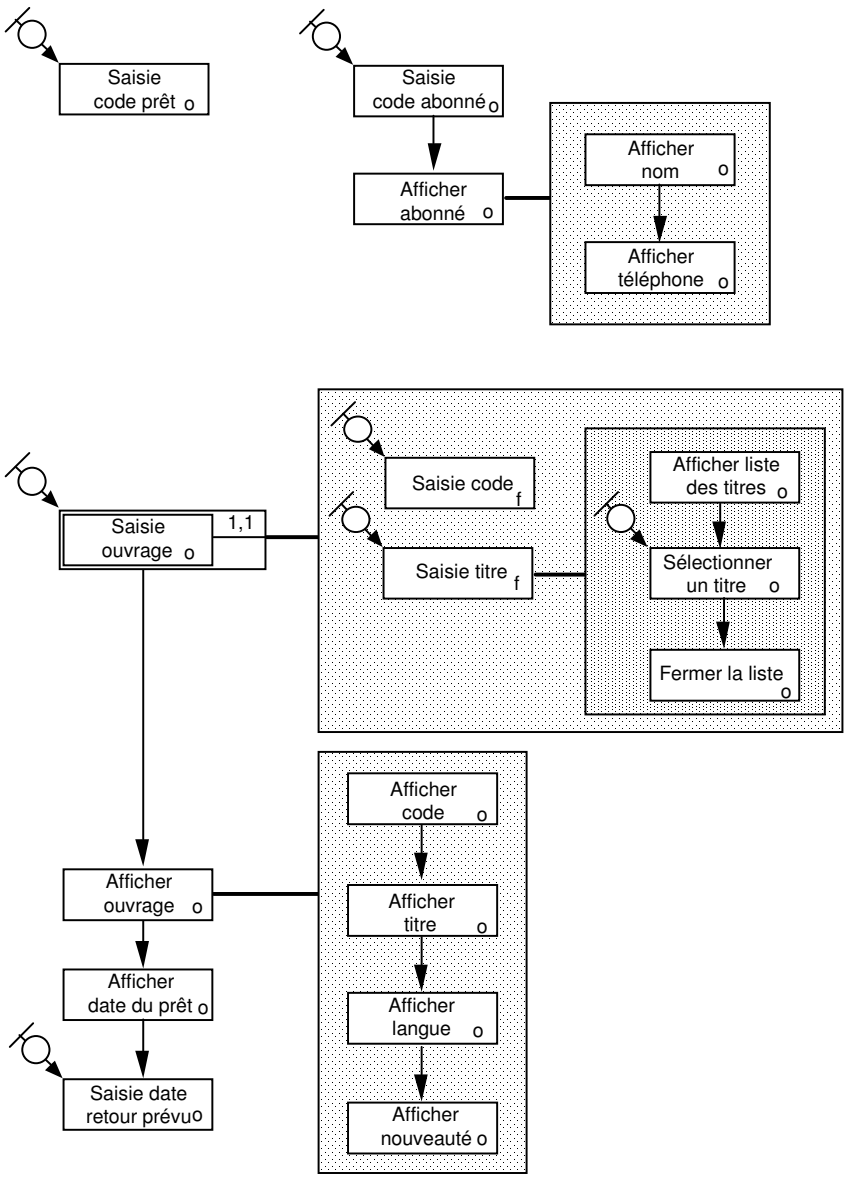


Figure 5.33 : Détail de l'opération Enregistrer un prêt de la figure 5.31

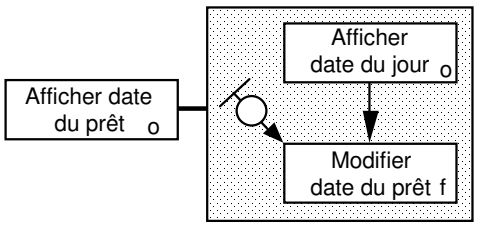


Figure 5.34 : Affichage de la date de prêt (par défaut la date du jour) avec modification éventuelle après la saisie

- la saisie de la date de retour prévu ne peut se faire qu'après l'affichage de la date de prêt. Telle qu'elle est représentée sur la figure 5.33, cette opération laisse entièrement la main à l'utilisateur (on suppose que cette opération contrôle la validité de la date saisie). On pourrait également faire afficher une date par défaut, par exemple la date du jour plus deux semaines,

tout en permettant une correction éventuelle. La représentation de cette opération serait donc similaire à celle de l'affichage de la date de prêt,

- lorsque toutes les opérations obligatoires de niveau 1 (Saisie code prêt, Saisie code abonné, Saisie ouvrage, etc.) ont été exécutées, l'opération mère Enregistrer un prêt reprend la main. Elle permet alors la validation de ce prêt ou son annulation (qui est restée disponible depuis le début de la saisie).

b. Retour de prêt

Pour enregistrer le retour d'un livre, c'est-à-dire la fin d'un prêt, le documentaliste doit connaître le code du prêt avant de pouvoir saisir la date de retour effectif. La figure 5.35 montre le détail de l'opération Retour de prêt.

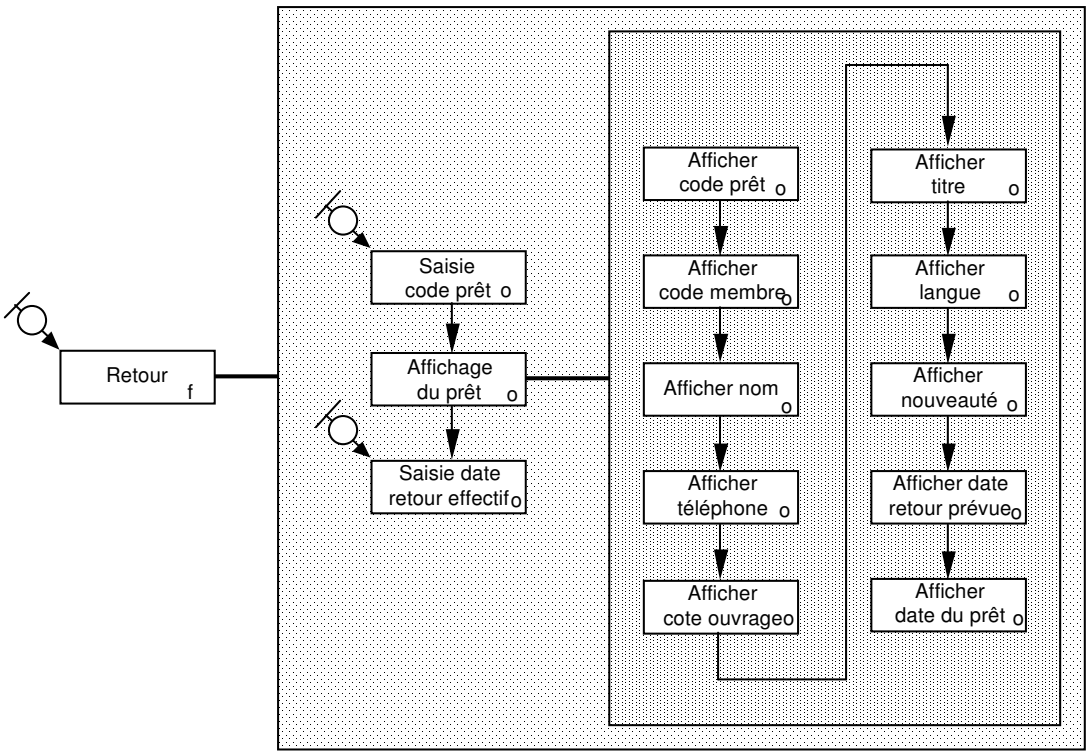


Figure 5.35 : Détail de l'opération Retour de prêt

On constate que la latitude décisionnelle est très faible dans cette opération ; toutes les opérations sont obligatoires et elles sont toutes reliées par des précédences permanentes. Si l'on voulait étendre la latitude décisionnelle, on pourrait par exemple supprimer les précédences entre Affichage du prêt et Saisie date retour effectif. De même que pour la procédure minimale du prêt, il est possible d'avoir dans la procédure minimale de retour des prêts les opérations Trouver un prêt, Passer au prêt suivant, Passer au prêt précédent, Supprimer le prêt. La duplication d'opérations telles que Supprimer le prêt est intéressante sur le plan de la logique de fonctionnement. Il est très pratique en effet pour le documentaliste de pouvoir supprimer un prêt sans être obligé de changer d'espace de travail (enregistrer les prêts ou enregistrer les retours). Si l'on considère que la gestion des prêts et celle des retours sont indépendantes, on aura donc deux procédures minimales distinctes. Cependant ces deux tâches sont regroupées de la façon suivante :

But : Gestion de la bibliothèque

Sous-but 1 : Gérer les prêts

Tâche 1 : Enregistrer un prêt

Opération 1 : Enregistrer un prêt
Opération 2 : Trouver un prêt
Opération 3 : Trouver le prêt suivant
Opération 4 : Trouver le prêt précédent
…
Tâche 2 : Enregistrer un retour
Tâche 3 : Faire une relance
…
Sous-but 2 : Gérer le fond
…

Ce type de décomposition nous permettra la génération de l'interface dans le chapitre suivant.

5. Diane+ et les autres méthodes

Différentes approches peuvent être envisagées pour le développement d'applications interactives. V. Normand [NORMAND 92a] en propose trois qui sont : l'approche psychologique (par exemple avec des méthodes telles que CLG [MORAN 81]), l'approche type génie logiciel (par exemple avec les travaux de Green [GREEN 85]) et l'approche ergonomique par exemple avec Diane+. Ces trois approches utilisent la notion de tâches et reconnaissent à des degrés divers l'importance de l'étape d'analyse des tâches. Certaines notions sont propres à chaque méthode (par exemple les procédures effectives ou prévues dans Diane+) alors que d'autres sont communes (par exemple l'utilisateur ou la tâche). Bien que toutes ces méthodes proposent en général un bon support pour l'analyse conceptuelle des applications interactives, elles ne donnent que très peu d'indications pour les spécifications externes ainsi que pour celles du dialogue homme-machine. Étant donné l'orientation actuelle des interfaces, nous nous intéresserons uniquement à des méthodes qui permettent le développement d'applications interactives. Pour cela nous étudierons d'abord des méthodes orienté-tâche qui prennent en compte les aspects du dialogue homme-machine, la décomposition fonctionnelle et la notion de tâches et de buts, puis des méthodes orienté-objet qui sont basées principalement sur la décomposition en fonction des données.

5.1. Diane+ et les méthodes orienté-buts

5.1.1. La notion de tâche

La tâche est une notion fondamentale que l'on retrouve dans beaucoup de travaux sur l'analyse du travail. Plus ou moins complète dans sa structure, elle repose souvent sur des attributs tels que l'action à exécuter ou les entrées-sorties. Nous donnons ici quelques exemples de travaux basés sur les tâches (on notera au passage que presque tous ces travaux utilisent cette notion conjointement avec celle d'objet).

B. Rousseau et C. Pierret-Golbreich [ROUSSEAU 88] décomposent une tâche en un niveau fonctionnel et un niveau opérationnel. Le premier contient les conditions d'exécution et les effets de la tâche, le second donne l'action ou les tâches à exécuter en réponse à un contexte fonctionnel

donné. C. Pierret-Golbreich [PIERRET 88] affine cette définition de tâche en ajoutant des attributs qui définissent l'état initial, l'état final et l'opérateur de la tâche[74].
M. Montalban [MONTALBAN 87] reprend la décomposition en niveau opérationnel et fonctionnel[75]. Une tâche y est constituée d'un ensemble de règles de production ordonnées, d'une tâche d'exception[76], d'un ensemble d'indicateurs, d'une nature et en supplément lors de sa manipulation d'un environnement d'exécution[77] ainsi que de l'auteur de son exécution.
Dans l'outil 3DKAT [DIENG 88], une tâche possède un nom, une classe dont elle est une instance, des entrées-sorties, des buts, des pré-conditions, des actions[78] à exécuter et des déclarations constituées de la liste des types d'objets qu'elle manipule. Les tâches peuvent être des tâches de base ou des tâches composées. Dans le premier cas ce sont des tâches procédurales et des tâches à base de règles[79], dans le second cas ce sont des tâches de base assemblées par des opérateurs temporels tels que AND, OR, THEN[80].
H. R. Hartson [HARTSON 92] utilise une notation orienté-utilisateur et orienté-tâche (User Action Notation) pour décrire le comportement physique de l'utilisateur et de l'interface pendant l'exécution d'une tâche. Avec cette notation, une tâche est décomposée récursivement en sous-tâches, le dernier niveau correspondant à des actions bas-niveaux (bouton de la souris enfoncé, déplacement du curseur sur une icône, etc.). UAN utilise une notation simple pour décrire ces actions élémentaires et les relations temporelles liant les tâches (séquentialité, concurrence, interruptabilité mutuelle, etc.). Une des originalités de UAN est la généricité des actions. Par exemple on peut écrire que lorsque l'on clique sur le contexte[81] d'un objet, celui-ci passe en inverse-vidéo. On peut alors appliquer cette action aux objets que l'on veut (icône, forme géométrique quelconque, nœud dans un graphe, etc.).

Nous présentons à présent trois méthodes orienté-tâches qui sont : Merise, MAD et la représentation par blocs. Merise est à l'origine de Diane et est très utilisée dans le milieu industriel pour la spécification, MAD est une méthode utilisée plutôt par les ergonomes pour analyser des tâches existantes alors que la représentation par blocs s'oriente plus vers une implémentation de tâches dans des systèmes d'assistance à l'opérateur.

5.1.2. Merise

Parmi toutes les caractéristiques de Merise, trois seulement nous intéressent ici :

- elle permet l'analyse et le développement de tâches procédurales et de systèmes d'informations collectifs en se focalisant sur les parties informatives et procédurales de ces systèmes. Elle ne peut donc pas être utilisée pour des applications de type créatif ou d'aide à la décision ce qui restreint son intérêt pour les IHM.

---

[74] L'opérateur décrit le niveau opérationnel de la tâche. Ce niveau est constitué d'entrées, de sorties, de conditions d'utilisation et d'un corps qui fait appel à une action primitive, une tâche de haut niveau ou bien encore une liste de tâches.

[75] Ce niveau fonctionnel est représenté hiérarchiquement par des fonctions de haut niveau, chaque fonction étant composée de sa fonction mère, de ses filles, de ses paramètres, de pré-conditions, de post-conditions, d'une fonction d'évaluation et d'une tâche si l'ensemble des fonctions filles est vide.

[76] Cette idée existe également dans [PLECZON 92] qui utilise le critère d'exception dans la structure de tâches (cf § 5.1.4).

[77] On retrouve cette notion de contexte dans [MICHARD 90] qui l'utilise pour gérer les interactions et les conflits éventuels entre tâches.

[78] Une action est composée d'une liste de procédures, de règles ou de tâches à exécuter.

[79] Cette capacité à intégrer à la fois des traitements procéduraux et des règles d'inférence a déjà été appréciée dans le § 6 et se retrouve par exemple dans la méthode KADS.

[80] Michard [MICHARD 90] utilise également ces opérateurs temporels ainsi qu'AVANT, APRES, PENDANT.

[81] Le "contexte" d'un objet est ce par quoi l'objet en question peut être manipulé. Ce peut être par exemple l'objet lui-même, son contour, des poignées sur son contour, etc.

- elle n'est fiable que pour des systèmes qui possèdent une grande stabilité. Or l'une des caractéristiques recherchées dans les IHM est la capacité d'un système à pouvoir facilement évoluer soit dans sa structure (par l'intermédiaire des concepteurs) soit dans son utilisation (par l'intermédiaire de l'utilisateur).
- elle utilise une analyse systémique globale du système d'information. On obtient ainsi des vues générales et détaillées des traitements à réaliser qui n'intègrent cependant pas la répartition des tâches entre l'homme et la machine. Merise décompose en effet les traitements par procédure et non pas par poste de travail. On sait donc pour chaque opération quel est le poste qui la déclenche, mais on n'a pas une vue générale des traitements par poste de travail.

Par ailleurs, Merise utilise la notion d'événement dans l'analyse des systèmes. Trois sortes d'événements sont pris en compte :

- les événements externes au système. Ils sont soit déclencheurs d'opérations soit résultats d'opérations,
- les événements internes au système et externes au processus auquel ils sont rattachés. Ce type d'événement permet de bâtir un pont entre deux processus sans être obligé d'avoir des opérations à cheval sur deux processus distincts. De façon générale ce type d'événement n'occasionne pas d'attente entre les deux processus,
- les événements internes au processus. Ce sont des résultats d'opérations qui peuvent également déclencher d'autres opérations dans le même processus.

Ces trois types d'événements sont en fait des occurrences d'événements, c'est-à-dire qu'ils sont pris en compte grâce à leur valeur. On n'écrira donc pas des événements tels que "commande" mais plutôt des événements tels que "commande passée". On retrouve ces événements dans les applications interactives. Les événements externes sont par exemple des événements en provenance du matériel informatique (fichier absent d'un disque dur, disquette protégée en écriture, etc.), le second type d'événement permet de faire un lien entre deux procédures sans toutefois les mêler (l'enregistrement d'un client rend possible la création de factures à son nom), enfin le troisième type est essentiel aux applications interactives puisque c'est grâce à des événements de ce type qu'on peut augmenter la latitude décisionnelle de l'utilisateur. Néanmoins les concepts de base de Merise ne sont pas suffisants pour bâtir des applications interactives bien qu'elle possède des notions intéressantes telles que le "mode d'automatisation" des opérations [ROCHFELD 91a]. On retrouve d'ailleurs les trois modes d'automatisation dans Diane et dans Diane+. Certaines extensions de Merise, par exemple dans Merise/2 [PANET 91], font apparaître directement les liens entre opérations et données (Figure 5.36).

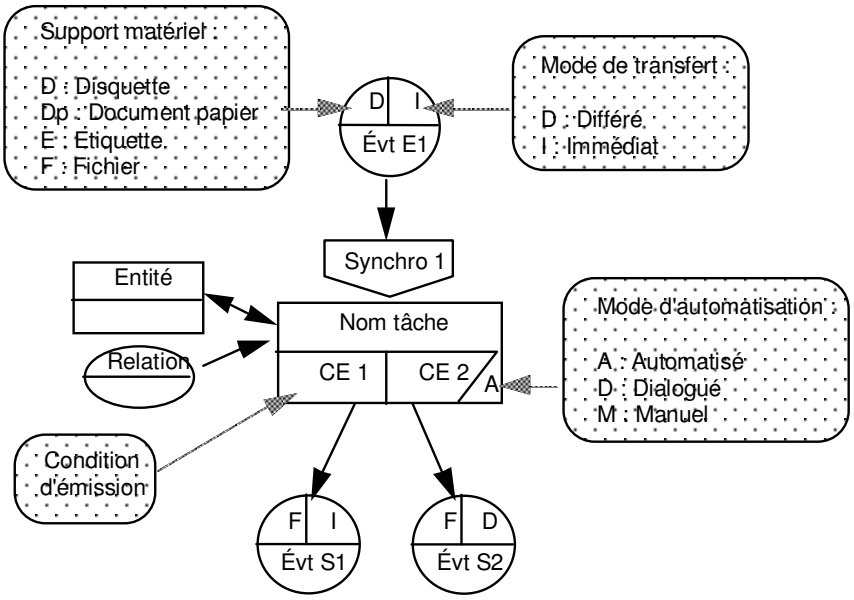


Figure 5.36 : Représentation d'une opération selon A. Rochfeld [ROCHFELD 91a]

Merise/2 propose diverses notions intéressantes telles qu'utiliser à la fois des vues centrées sur les opérations grâce au MCTA (Modèle Conceptuel des Traitements Analytique) et des vues centrées sur les objets grâce aux cycles de vie.

"Le concept de base du MCTA est l'opération analytique. Une opération analytique est déclenchée par un ou plusieurs événements et fournit un ou plusieurs résultats. Elle est constituée d'un ensemble d'actions élémentaires sur la structure de données. Une action élémentaire crée, consulte, modifie ou supprime une et une seule entité (objet ou relation). Elle peut être accompagnée d'une condition de déclenchement et/ou d'itération" [PANET 91].

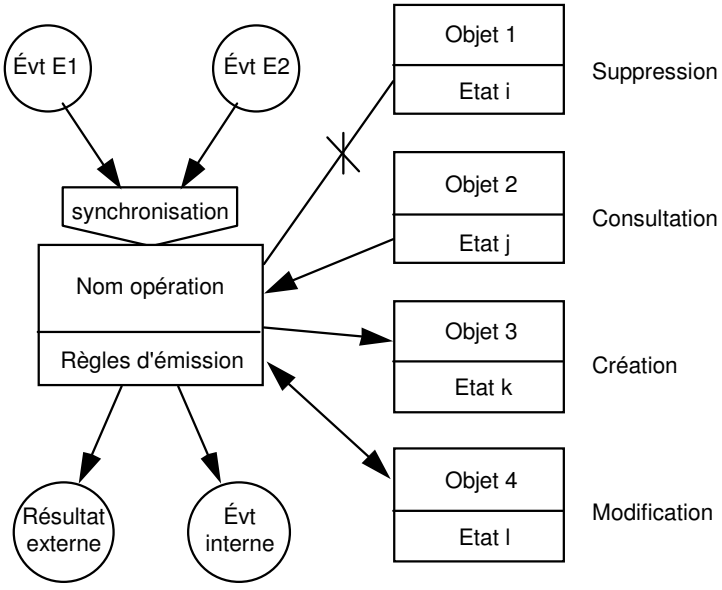


Figure 5.37 : Représentation d'opération dans Merise/2

Une opération peut être décomposée en sous-opérations. Merise/2 préconise une décomposition jusqu'à obtention de fonctions réutilisables.

“Alors que l’opération analytique donne une vision globale des conséquences d’un même événement, le cycle de vie d’un objet (CVO) met en évidence l’ensemble des états que peut prendre un objet au cours de son cycle de vie, et l’ensemble des événements qui font passer l’objet d’un état dans un autre état. Le MCTA donne une vision synthétique de la coordination des événements, le CVO donne une vision synthétique de la coordination entre états” [PANET 91].

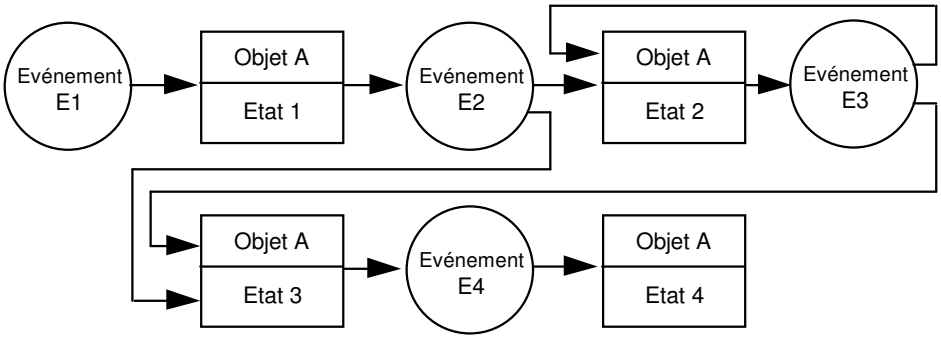


Figure 5.38 : Représentation du cycle de vie d’un objet dans Merise/2

On notera que cette notation n’est pas sans rappeler celle de la méthode présentée dans [MÄRTIN 91] (cf Chapitre 2, § 5.3.5).

Pour intégrer l’utilisateur dans les traitements, G. Panet et al. [PANET 91] proposent d’ajouter des primitives de dialogue aux primitives non interactives qui apparaissent dans le modèle logique des traitements. Le formalisme utilisé est très proche de celui du MCTA. G. Panet et al. justifient ce choix par le modèle de Seeheim qui demande une séparation entre le noyau fonctionnel et la présentation. On peut toutefois reprocher à Merise/2 d’intégrer l’interface uniquement vers la fin de l’analyse, la contraignant ainsi à se plier aux spécifications. L’intégration de l’interface depuis le début des spécifications peut faire apparaître des défauts dans les spécifications (latitude décisionnelle trop faible, critères ergonomiques non respectés, etc.).

Conclusion sur Merise et Merise/2

Les deux défauts majeurs de Merise et de Merise/2 sont de ne faire intervenir que l’utilisateur dans le déclenchement des opérations (la machine ne peut jamais prendre la relève) et de ne pas faire de différence sur le type des opérations (obligatoires ou facultatives) et sur le type des utilisateurs. Diane+ remédie à ces deux défauts en proposant une description des opérations qui prend en compte le déclencheur de l’opération (homme ou machine), et différents types de procédures et d’opérations afin de prendre en compte le plus possible le niveau et les habitudes des utilisateurs.

5.1.3. MAD

Un des objectifs généraux de MAD (Méthode Analytique de Description des tâches) est de représenter des tâches utilisateur de manière uniforme afin de poser les problèmes ergonomiques de conception [SCAPIN 89]. Elle utilise pour cela la planification hiérarchique en y intégrant des aspects de synchronisation reliant les actions à exécuter.

Une tâche est représentée par un arbre composé d’“item-tâche”. Chacun de ces items est caractérisé par un ensemble de buts et de prérequis (qui peuvent être également des items) et est décomposé en objets et en opérations (par exemple sélection (texte)). Chaque tâche contient :

- un état initial constitué de la liste des objets en entrée,
- un état final constitué de la liste des objets en sortie ou des objets directement créés ou modifiés,
- un but qui est en fait un sous-ensemble de l'état final,
- des pré-conditions qui sont constituées d'un ensemble de prédicats exprimant des contraintes sur l'état initial. Il existe des pré-conditions qui permettent le déclenchement de la tâche associée et des pré-conditions déclenchantes qui lancent elles-mêmes l'exécution.
- des post-conditions qui sont constituées d'un ensemble de prédicats exprimant des contraintes sur l'état final et qui doivent nécessairement être satisfaites après l'exécution de la tâche associée.

MAD fournit deux types de tâches :

- les tâches élémentaires qui sont des tâches indécomposables et qui correspondent à une action. Elles contiennent en plus des attributs pré-cités un descripteur de l'action à exécuter,
- les tâches composées qui comportent en plus des attributs pré-cités, un champ "structure" qui décrit le corps de la tâche. Cette structure s'exprime au moyen de constructeurs (Figure 5.39) tels que SEQ (tâches en séquence), PAR (tâches en simultané), ALT (tâches au choix), BOUCLE, FAC (tâches facultatives) et d'arguments qui sont constitués d'une liste de tâches. Il faut remarquer que les "structures" ne possèdent pas de constructeurs conditionnels (IF) ce qui signifie qu'on ne peut pas représenter des règles d'émission.

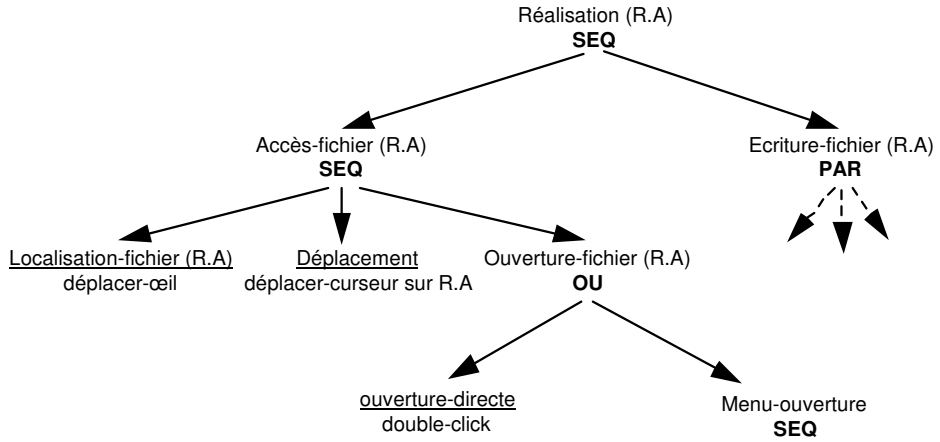


Figure 5.39 : Extrait de l'arbre de la tâche "réalisation de rapport d'activité (R.A)" [SCAPIN 89].

Conclusion sur MAD

MAD fournit une bonne représentation des tâches du point de vue de l'ergonome. Cependant elle décrit le travail indépendamment de la répartition entre l'homme et la machine et de la répartition entre le poste de travail et l'organisation générale. Il est donc difficile de l'utiliser directement pour la couverture complète du cycle de vie des applications interactives.

5.1.4. La représentation par blocs

P. Pleczon présente dans [PLECZON 92] un formalisme de représentation de tâches basée sur l'utilisation de blocs de connaissances [BOY 89] [MATHE 90]. Ce formalisme est utilisé pour des applications d'assistance à l'opérateur (conduite automobile, télémanipulation, etc.). Il décompose les tâches sous forme de blocs dont la structure générale est donnée sur la figure 5.40. Chaque bloc

contient soit des opérations à exécuter soit d'autres blocs (décomposition hiérarchique des tâches). Le bloc n'est déclenchable que lorsque les pré-conditions sont satisfaites. A cet instant les actions proposées dans le bloc deviennent activables. Si une erreur survient durant leur exécution et que cette erreur correspond à une des conditions anormales du bloc, celui-ci déroute son exécution sur le bloc en relation. Si l'exécution s'est correctement déroulée, un certain nombre de buts du bloc sont atteints, ce qui provoque également le branchement sur d'autres blocs.

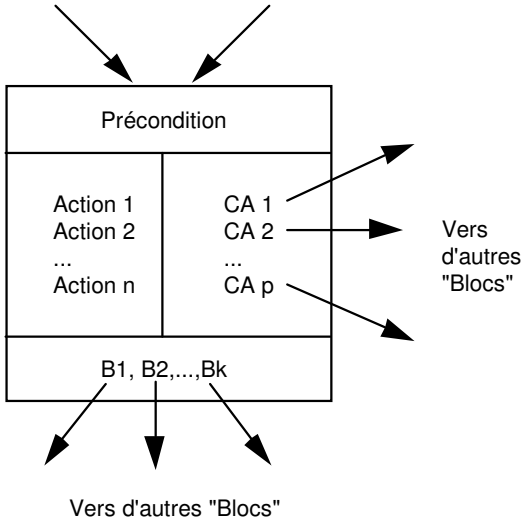


Figure 5.40 : Représentation graphique d'un bloc.
Les CA_i représentent les Conditions Anormales, les B_j représentent les buts

La figure 5.41 représente les blocs permettant la conduite d'une voiture. Dans cet exemple, le bloc Démarrer contient comme action le sous-bloc Actionner démarreur représenté sous la ligne hachurée. Par ailleurs les flèches visualisent les liens entre les blocs soit grâce aux buts (par exemple entre Démarrer et Avancer) soit par les conditions anormales (Passer la première et Mauvais rapport). La flèche grisée représente un "lien de récupération", c'est-à-dire un lien qui part d'un niveau inférieur à un niveau supérieur. Ce type de lien est utilisé pour remettre le système dans une configuration normale après une erreur.

Cette notion de blocs est intéressante sur plusieurs points :

- elle utilise une décomposition hiérarchique des tâches et des opérations,
- elle gère la notion de buts,
- elle gère automatiquement les erreurs,
- elle utilise le multi-tâche.

Le formalisme paraît donc intéressant à utiliser pour les IHM. Or toutes les actions représentées par ces blocs sont supposées interactives (l'utilisateur déclenche ces opérations par l'intermédiaire de boutons ou de molettes sur le tableau de bord par exemple). De par la nature temps réel de l'application représentée par ces blocs, tous les branchements sont donc prévus et "câblés". La latitude décisionnelle de l'utilisateur est donc relativement nulle. De plus les actions ne manipulent pas de données au sens informatique classique, mais des valeurs issues de capteurs. La représentation par blocs mériterait donc une extension permettant d'intégrer et de visualiser des données de type base de données par exemple. Par ailleurs il serait intéressant d'intégrer dans cette extension une gestion de l'aide.

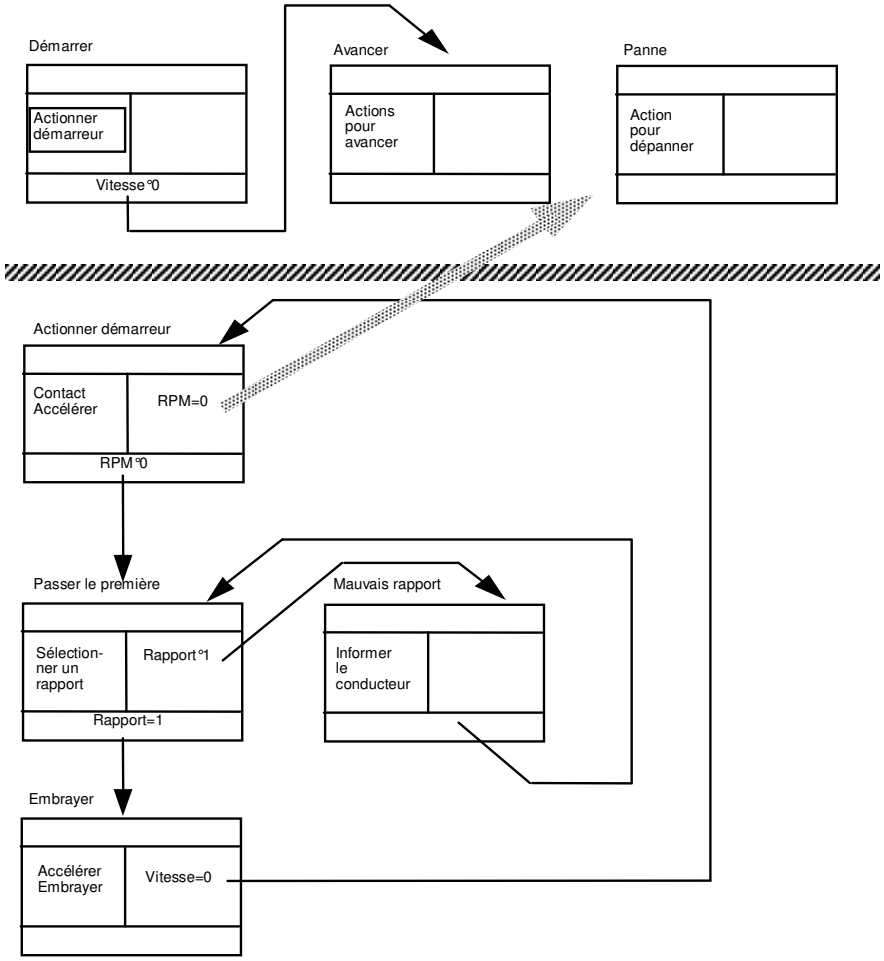


Figure 5.41 : Exemple de décomposition de tâches avec la représentation par blocs (extrait de [PLECZON 92])

Conclusion sur la représentation par blocs

La représentation par blocs est donc proche de Diane+ (décomposition et planification hiérarchiques des tâches et des opérations, gestion du multi-tâche, latitude décisionnelle en partie implicite), mais elle s'avère encore trop incomplète (pas d'intégration explicite de l'utilisateur par exemple) pour une utilisation dans les IHM bien que ses fonctionnalités et son pouvoir d'expression soient puissants [BOY 91a] [BOY 91b].

5.2. Diane+ et les méthodes orienté-objets

Nous présentons ici trois "méthodes" orienté-objet, différentes de par leur notion d'objet. La première (OOA [COAD 90]) est sans aucun doute la plus objet des trois, la seconde (OOM [ROCHFELD 91b]) est le résultat de l'adaptation de Merise au concept objet, et la troisième est une sur-couche au formalisme entité-association qui permet de créer des objets dits "naturels" [BRES 90]. Pour chacune d'entre elles, nous ferons une rapide présentation puis nous verrons en quoi elles peuvent être conjuguées avec Diane+.

5.2.1. La méthode de Coad et Yourdon : OOA

A. Coad et E. Yourdon propose dans [COAD 90] une méthode d'analyse orienté-objet. Cette méthode est donc située en amont des méthodes de conception orienté-objet. Elle se décompose en cinq étapes qui sont :

- *trouver les classes et les objets*. Pour Coad et Yourdon, le mot objet a la même signification que les instances de Smalltalk (le sens de classe est également le même que dans Smalltalk). Ils précisent qu'il ne faut pas séparer l'analyse des données de celle des traitements ce qui leur permet de construire des objets bien structurés,
- *identifier les structures*. Coad et Yourdon introduisent un formalisme qui leur permet de représenter élégamment les différents types de liens qui unissent les classes et les instances. Pour structurer les classes entre elles, ils utilisent la notion de généralisation/spécialisation ainsi que celle de composé/composant. La figure 5.42 montre une classe A sans instance qui se spécialise en deux classes avec des instances. Les liens unissent ici les classes puisqu'ils portent sur elles (ils touchent le cadre des classes et non pas le cadre des instances).

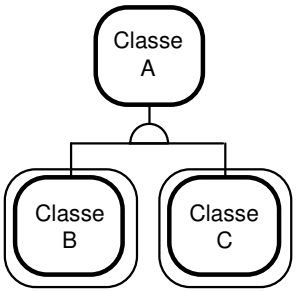


Figure 5.42 : Une classe A sans instance se spécialisant en deux sous-classes avec instances

La figure 5.43 montre une relation de composé/composant entre deux classes (un organisme est constitué d'employés). Remarquons que le lien porte sur les instances et non pas sur les classes, et qu'il supporte des contraintes de cardinalité.

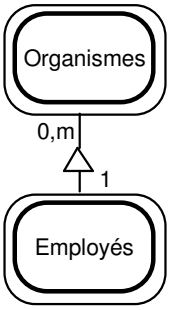


Figure 5.43 : Une relation de composé/composant entre deux classes

- *identifier les sujets*. Structurer les classes entre elles ne suffit pas. Il faut également les grouper suivant une certaine logique afin d'avoir des vues d'ensemble plus claires. Coad et Yourdon définissent la notion de sujet (Figure 5.44). Un sujet regroupe plusieurs classes qui ont un lien sémantique entre elles. Ces sujets peuvent avoir trois présentations différentes ; fermés ils ne présentent que leur nom, partiellement ouverts ils affichent les noms des classes qu'ils contiennent, enfin ouverts ils présentent les classes qu'ils englobent ainsi que leurs liens. Cette notation se rapproche des objets naturels que nous verrons plus loin et elle nous semble intéressante pour Diane+. On peut envisager en effet de présenter

sous un aspect similaire les procédures ; fermées elles n'afficheraient que leur nom, partiellement ouvertes elles afficheraient le nom des opérations qui les composent et enfin ouvertes elles s'afficheraient telles que nous les avons représentées jusqu'à présent.

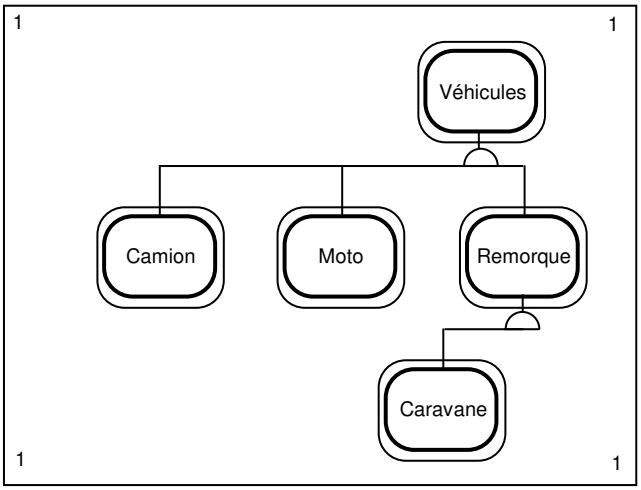


Figure 5.44 : La cadre numéroté 1 correspond au sujet numéro 1 du système. Il réunit toutes les classes qui ont un rapport avec le transport. Un sujet peut contenir des classes qui, bien qu'en rapport, ne sont pas nécessairement liées par un lien de type généralisation ou composant.

- *définir les attributs*. L'architecture générale étant définie, il reste à affiner les classes. Pour cela Coad et Yourdon associent à chaque classe un ensemble d'attributs dont la signification est la même que celle des attributs dans les schémas entité-association.

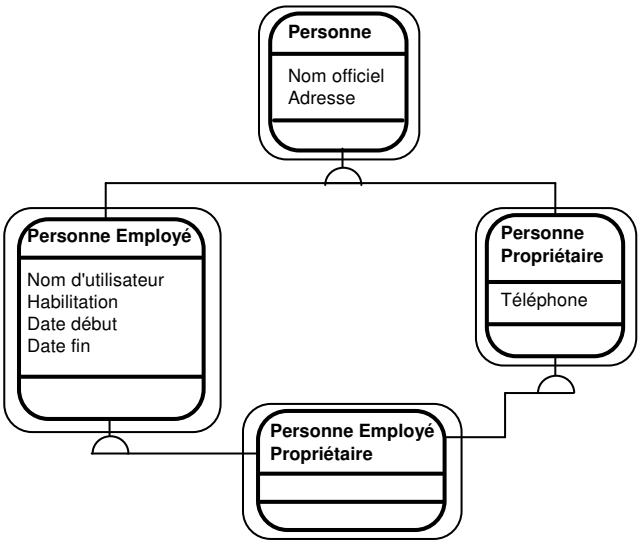


Figure 5.45 : Quatre classes avec leurs attributs (au centre des cadres) et leurs relations.

- *définir les services*. Cette dernière étape permettra aux objets d'avoir une vie propre et de pouvoir être manipulés par d'autres objets ou par l'utilisateur. Coad et Yourdon différencient deux types de services : les services à algorithme simple tel que créer et qui sont implicites, et les services à algorithme complexe tels que superviser. Les services étant définis, Coad et Yourdon proposent de relier les objets en fonction des messages qu'ils sont capables d'émettre. En fait ces liaisons relient uniquement les objets qui sont dépendants ; elles ne

permettent pas de dire quelle sera la dynamique du système final. Coad et Yourdon précisent qu'on peut associer une chronologie aux messages en ajoutant des numéros d'ordre, mais ceci permet de définir une partie seulement de la dynamique du système. Étant donné que les connexions entre objets ne mettent en évidence que les objets dépendants, le choix des enchaînements de traitements non spécifiés par ces connexions est donc laissé à l'utilisateur. Ceci implique que la latitude décisionnelle est quasiment maximale. Or certains systèmes demandent une restriction de cette latitude et la méthode OOA ne permet pas cette restriction. Nous avons vu que Diane+ répond à cette attente (cf § 3.2).

L'apparence finale des classes et des objets est donc la suivante (Figure 5.46) :

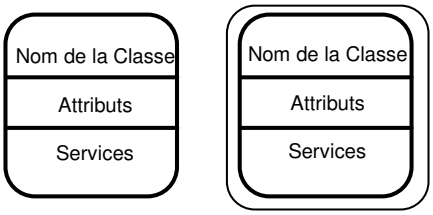


Figure 5.46 : Structure générale des classes.
La classe de gauche ne possède pas d'instance.

Cette phase d'analyse terminée, nous disposons d'une structure de classes et d'instances claire, non ambiguë et qui peut être validée par l'utilisateur. Malheureusement elle est insuffisante pour être implémentée directement, c'est pourquoi Coad et Yourdon proposent de passer à la phase de conception orienté-objet en ajoutant des composants facilitant l'implémentation. Ces composants sont par exemple une couche pour l'interaction homme-machine, une couche pour la gestion des tâches, et une couche pour la gestion des données. On peut d'ailleurs se demander si ces différents composants n'arrivent pas un peu tard dans le développement. Le dialogue homme-machine par exemple aurait tout intérêt à être incorporé depuis l'analyse ce qui permettrait une validation plus complète du système par l'utilisateur.

Conclusion sur la méthode OOA

Coad et Yourdon proposent un outil intéressant pour le développement des applications informatiques. Jusqu'à présent le concept objet n'était utilisé que pour l'implémentation et la conception. Avec une analyse orienté-objet, les objets couvrent maintenant entièrement le cycle de vie.
Le formalisme employé est très clair et facile à utiliser. Il peut être facilement validé par les informaticiens au même titre que les schémas entité-association. Cependant il a l'inconvénient de faire perdre la vue globale du système car on connaît uniquement les services par rapport aux objets. La dynamique entre les objets n'apparaît pas (sauf pour les cas où la chronologie est fixée) ce qui signifie que l'utilisateur doit décider quels objets gérer et comment le faire puisqu'il ne dispose en fait que d'un vrac d'objets qui sont groupés uniquement en fonction de leur sémantique. V. Normand [NORMAND 92a] répond à ce problème par l'intermédiaire d'arbres de tâches qui mettent en relation les objets manipulés en fonction des tâches à réaliser.
Si une analyse de type OOA doit être utilisée, il nous paraît intéressant, voire même indispensable, de chapeauter les classes issues de l'analyse et de la conception par des procédures qui seraient directement branchées sur les services des objets. En effet ces objets ont été créés dans le but de réaliser des traitements. Dans le cas des tâches créatives et des tâches expertes, le vrac des objets peut être suffisant à condition que l'utilisateur sache comment utiliser ces objets. Par contre pour les tâches de type procédural, l'utilisateur doit manipuler plusieurs objets simultanément et en

suivant des règles d'enchaînement strictes. En groupant les objets en fonction des traitements à réaliser, et en tenant compte des enchaînements et des contrôles obligatoires, la gestion des objets devient plus simple pour l'utilisateur. Par exemple la saisie d'un bon de commande fait appel à la classe Client, à la classe Bon de Commande, à la classe Articles, etc. Si on les regroupe dans une procédure Diane+ qui reprend les services offerts par ces classes, il est alors possible de contrôler les enchaînements ainsi que les déclencheurs simplement en créant une couche de dialogue qui cimente les différents objets.
La méthode d'analyse OOA possède donc de nombreux avantages, mais elle a tendance à être trop floue sur le plan du dialogue homme-machine et sur la dynamique du système produit. Utilisée conjointement avec une méthode orientée tâche telle que MAD (pour une analyse de l'existant) ou Diane+ (pour la spécification), elle devrait permettre la réalisation de systèmes bien analysés et d'une grande utilisabilité.

5.2.2. La méthode OOM

La méthode OOM (Orientation Objet dans Merise) [ROCHFELD 91b] [ROCHFELD 92] propose une orientation objet qui réunit les données et les traitements sans remettre en cause les acquis antérieurs de Merise. En cela elle pallie en grande partie aux deux limites des méthodes de conception orienté-objet qui sont [ROCHFELD 92] :

- *la couverture partielle du cycle de vie*, les méthodes orienté-objet n'intervenant bien souvent que vers la fin, juste avant la programmation,
- *la définition totalement subjective des objets*, d'où un problème pour la validation et la réutilisation.

A. Rochfeld remarque que toutes les méthodes de conception suivent le référentiel représenté sur la figure 5.47. OOM respecte donc également ce référentiel. Sa dimension fonctionnelle est basée sur l'utilisation de diagrammes de flux de données qui sont réutilisés pour les diagrammes d'activité (niveau conceptuel), les diagrammes d'acteurs pour les postes de travail (niveau organisationnel) et les diagrammes de modules (niveau physique). Nous allons détailler à présent les deux autres dimensions qui nous concernent d'avantage ici.

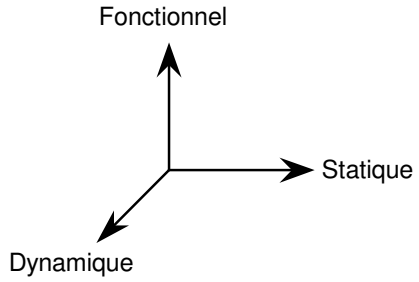


Figure 5.47 : Le référentiel des méthodes de conception

Dimension statique

OOM utilise des MCD évolués[82] qui sont beaucoup plus complets et plus proches de la réalité que ceux de Merise. Malheureusement ils n'intègrent que les données, et les traitements doivent donc

[82] Remarquons que ces MCD ne respectent plus la normalisation à cause des entités multi-valuées et des listes de valeurs qu'ils incorporent. Rochfeld ne précise malheureusement pas si sa méthode est dédiée uniquement à la spécification (auquel cas la normalisation n'est pas gênante) ou s'il désire également s'en servir pour la conception (auquel cas la normalisation est importante).

toujours être traités séparément. Pour remédier à cet inconvénient, A. Rochfeld reprend ces MCD et les transforme en objets en leur incorporant les traitements qui s'y rapportent. Pour cela il définit les méthodes des objets par rapport à un référentiel de neuf méthodes types [ROCHFELD 92] :

- acquisition ou création d'information,
- sélection d'information,
- consultation d'information,
- contrôle d'information,
- suppression d'information,
- calcul d'information dérivée,
- présentation,
- édition de résultats,
- traitements et édition des erreurs.

A. Rochfeld intègre donc plus de méthodes que Coad et Yourdon ne le font dans leur analyse orienté-objet (par exemple les méthodes de présentation). Ce référentiel de méthodes a l'avantage de fournir aux données un standard pour leur future manipulation. On est donc sûr que toutes les données pourront répondre aux mêmes types de demandes.

Pour passer d'un MCD classique à un MCD objet, A. Rochfeld intègre les méthodes énoncées plus haut dans les entités concernées. Il les répartit ensuite en fonction de leur comportement vis-à-vis de la structure des objets. Celles qui ont un comportement neutre migrent vers un objet central alors que les autres restent attachées aux objets sur lesquels elles portent. L'étape suivante consiste à définir les parties privées (attributs et méthodes) et publiques (méthodes uniquement) des objets. A. Rochfeld intègre enfin les liens d'héritage et les associations (qui sont devenues également des objets) dans les objets qui en dépendent directement. La figure 5.48 représente le passage du modèle externe de données représentant un devis à son homologue orienté-objet.

A. Rochfeld reste flou en ce qui concerne la définition des parties publiques et privées. Il semblerait que cette répartition soit faite en fonction de l'objet qui est demandeur des services. Cela signifie que ces parties publiques et privées sont susceptibles d'être différentes en fonction des autres objets et donc il est difficile d'apprécier exactement ce qui est connu des autres objets et ce qui ne l'est pas. Par ailleurs cela suppose que l'on est obligé de faire de nombreuses définitions des objets, d'où une redondance d'informations et une perte de temps. Si l'on suppose par contre que la définition des parties publiques et privées est unique, OOM est une solution élégante pour passer des schémas classiques à des schémas objet.

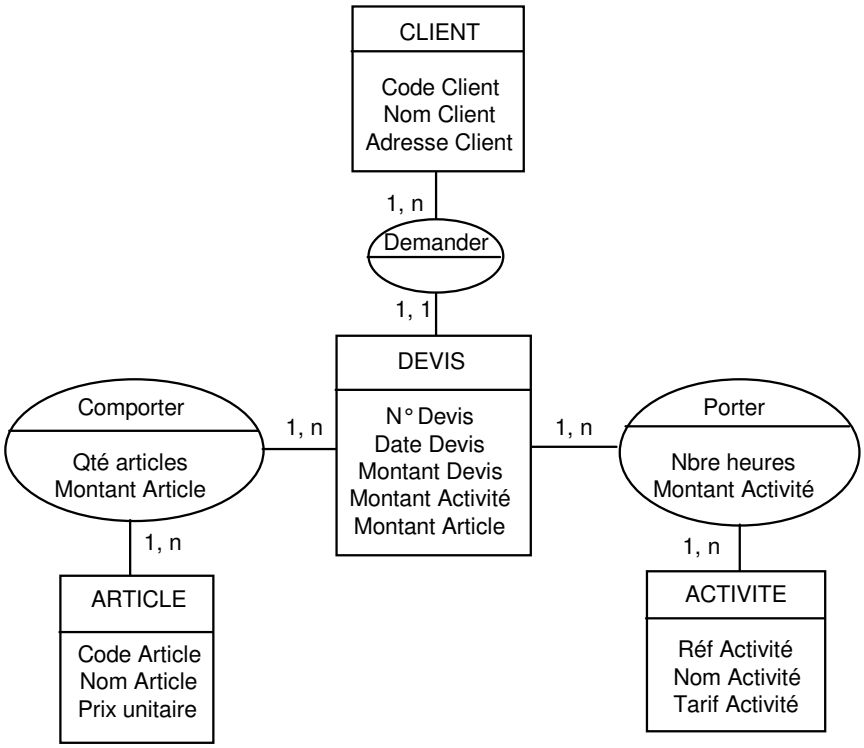


Figure 5.48.a : Modèle externe de données d'un devis

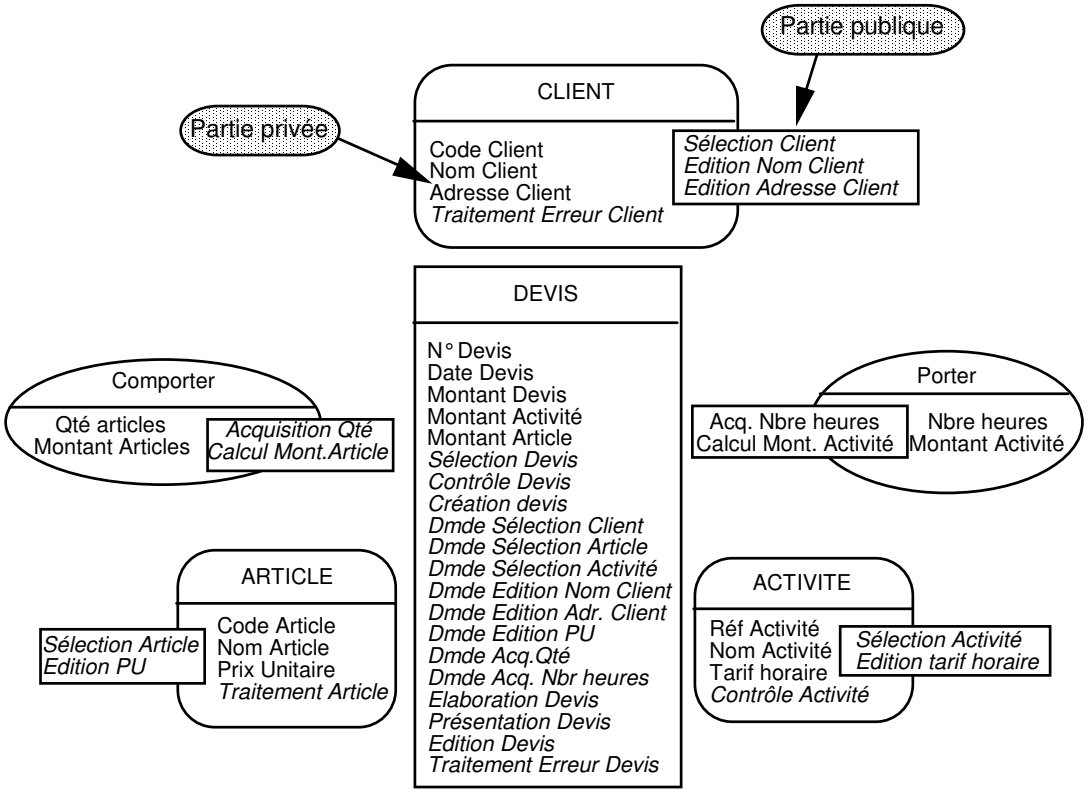


Figure 5.48.b : Modèle externe objet du même devis que celui de la figure 5.48.a.
Les libellés en italique sont des noms de méthodes.

Dimension dynamique

Cette dimension se traduit au niveau des objets et des traitements.

a. Les objets

Pour les objets, Rochfeld utilise des graphes d'agencement et des graphes de cycle de vie. Les graphes d'agencement précisent comment les objets sont utilisés conjointement, ceci s'exprimant sur les modèles externes objets par des flèches reliant les objets dépendants (Figure 5.49).

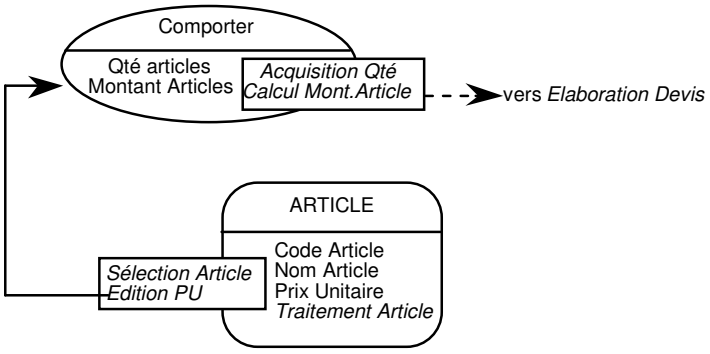


Figure 5.49 : Extrait du graphe d'agencement du modèle externe objet d'un devis. Les flèches partent des méthodes et pointent sur d'autres méthodes ou sur des attributs.

Les graphes de cycle de vie (présents également dans Merise, OOA [COAD 90] et OMT) montrent comment les objets se transforment en mettant en relief les méthodes utilisées lors des changements d'états ainsi que les conditions de transition entre ces états (Figure 5.50).

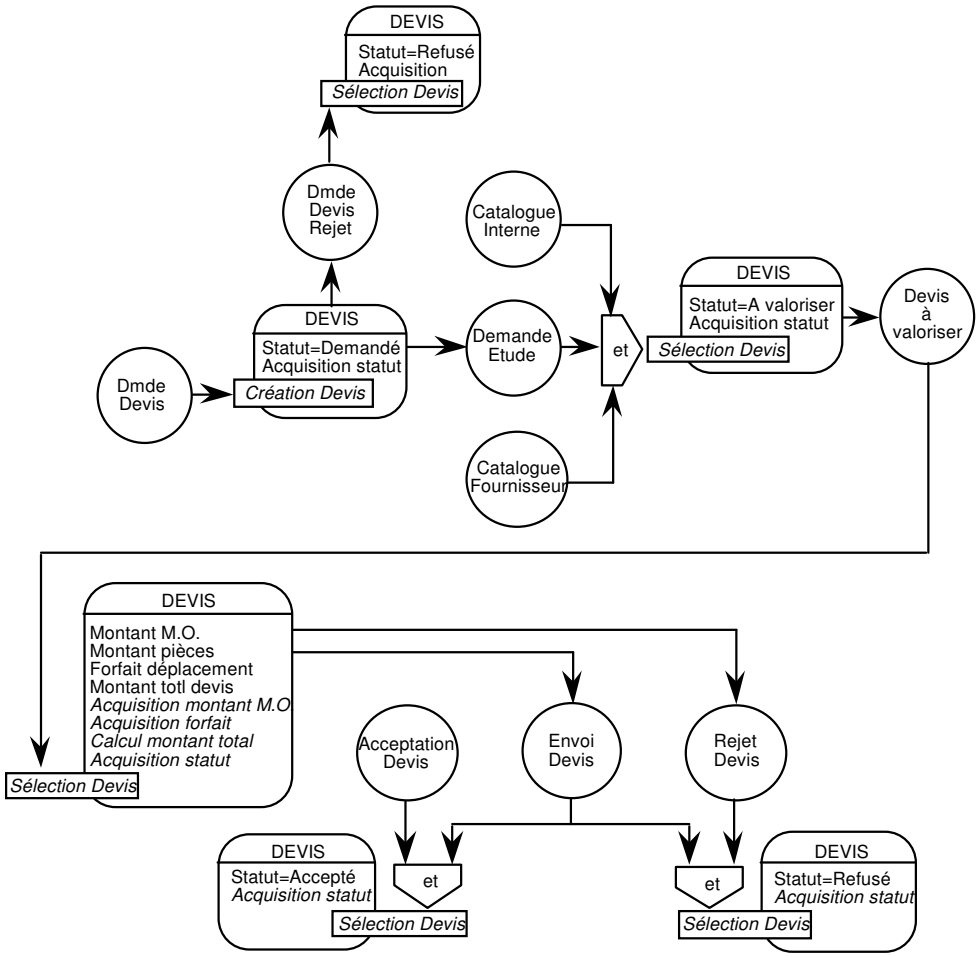


Figure 5.50 : Cycle de vie de l'objet DEVIS

b. Les traitements

La dynamique au niveau des traitements se traduit par les MCT de Merise auxquels A. Rochfeld incorpore les entités manipulées ainsi que le type des liens qui les unissent avec les traitements. La figure 5.51 énumère les différents cas de figure. Ce formalisme est très intéressant pour Diane+ car il nous permet de savoir immédiatement quels sont les objets en entrée et les objets en sortie, d'où la définition des paramètres des opérations. De plus il nous fournit le type d'intervention sur les données (création, suppression, etc.) ce qui nous permet de faire des contrôles automatiques de par la nature des opérations (par exemple une suppression implique une confirmation sans que le concepteur ait besoin de le spécifier).

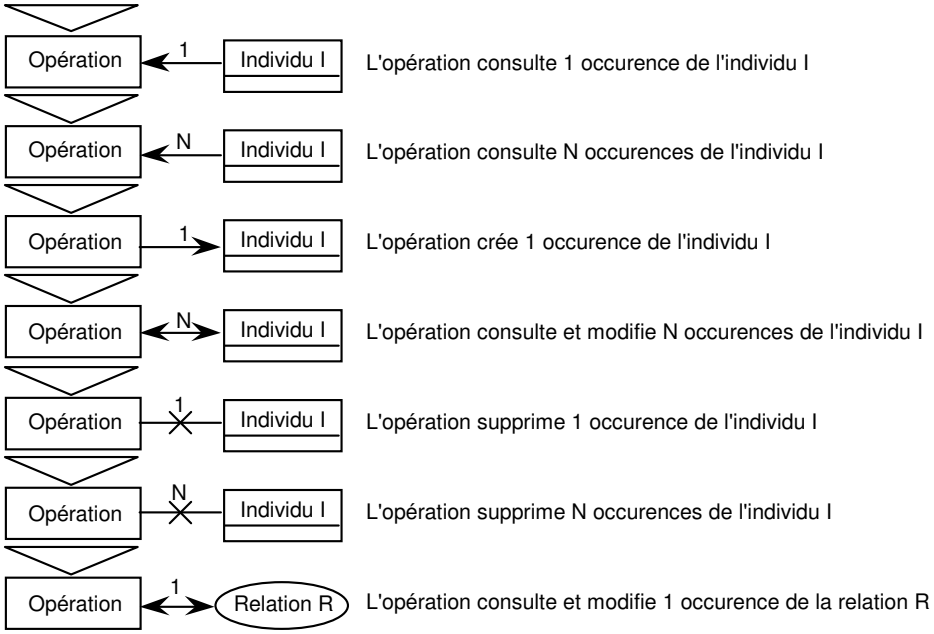


Figure 5.51 : Conventions graphiques pour les modèles conceptuels analytiques des traitements dans OOM

Conclusion sur la méthode OOM

OOM permet de préciser la dynamique du système au niveau de chaque objet (par le cycle de vie), mais elle fait perdre la vision globale de cette dynamique. Elle ne permet pas de dire QUI fait QUOI QUAND. Les cycles de vie montrent bien l'évolution de chaque objet, mais on ne perçoit pas les répercussions sur les objets dépendants. De même que la méthode OOA, l'utilisateur dispose d'un vrac d'objets qu'il est le seul à gérer. On peut envisager de compléter OOM avec Diane+ en reprenant ses résultats, en groupant les opérations publiques par postes de travail et en ajoutant des caractéristiques de Diane+ telles que les déclencheurs ou les types d'opérations. On procéderait donc de façon similaire à l'adaptation de Diane+ sur la méthode OOA.

OOM est cependant très intéressante car elle permet le passage de Merise à Merise Objet sans rien modifier des acquis. Un thème de recherche intéressant serait son utilisation pour la génération automatique de l'interface. Des travaux tels que celui de V. Normand, qui génère l'interface, sont en effet proches de OOM.

5.2.3. Les objets naturels

Le modèle des objets naturels, présenté par P-A Brès dans [BRES 90] [BRES 92], s'intègre dans l'atelier de génie logiciel Tramis. “Un objet naturel est un objet du système d'information significatif pour l'utilisateur. C'est en fait un objet dont les composants (entités et associations) sont fortement liés sémantiquement et réagissent d'une manière commune d'un point de vue fonctionnel, permettant une définition globale de leur comportement” [BRES 90].

Le modèle des objets naturels est donc bâti au-dessus des schémas entités-associations classiques. En cela il permet de réutiliser d'anciennes descriptions de systèmes d'information sans en modifier les structures. Les objets naturels sont le résultat de l'assemblage d'entités et d'associations en blocs qui ont une signification les uns par rapport aux autres et cela par rapport à l'utilisateur et non plus par rapport à l'informaticien. On peut dire que les objets naturels fournissent une logique

d'utilisation des données. Ainsi l'entité Ligne de Commande utilisée d'habitude par les informaticiens se retrouve ajoutée à l'entité Commande classique pour former l'objet naturel Commande qui est plus parlant pour l'utilisateur (Figure 5.52).

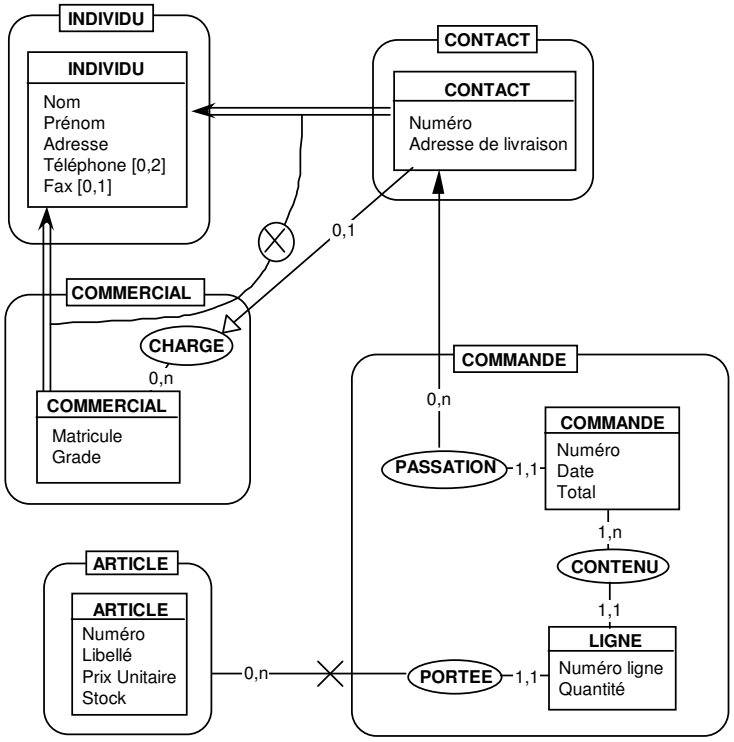
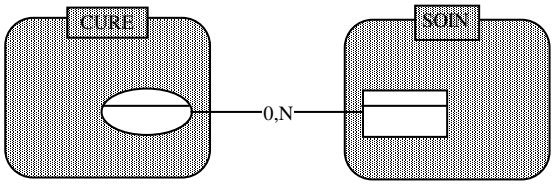


Figure 5.52 : Exemple d'objets naturels

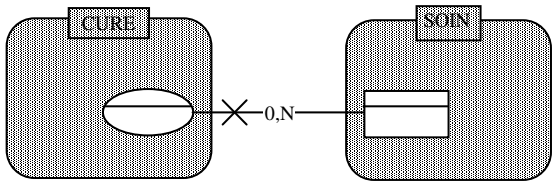
Les objets naturels apportent non seulement la notion d'assemblage sémantiquement logique, mais également des liens qui améliorent cet assemblage. On dispose ainsi de liens qui unissent les objets naturels par l'intermédiaire des entités et des associations qui les composent. Ces liens (appelés également rôles frontières) permettent d'exprimer par exemple[83] :

- *Opacité et Apparence* : l'ensemble des soins d'une cure est défini exclusivement à partir de l'objet naturel CURE. L'objet SOIN voit les occurrences de CURE auxquelles il est relié.

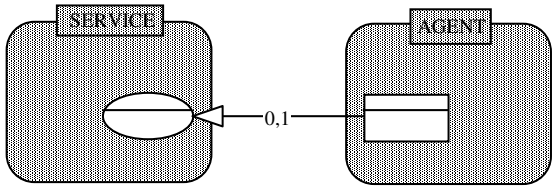


- *Opacité et Fermeture* : l'ensemble des soins d'une cure est défini exclusivement à partir de l'objet naturel CURE. L'objet SOIN ne voit pas les occurrences de CURE auxquelles il est relié.

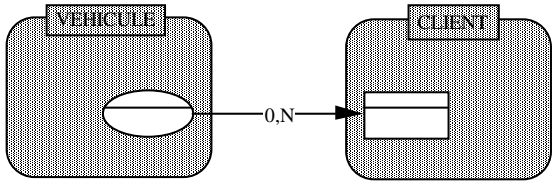
[83] Les exemples cités sont tirés de la documentation de Tramis/View [TRAMIS 92].



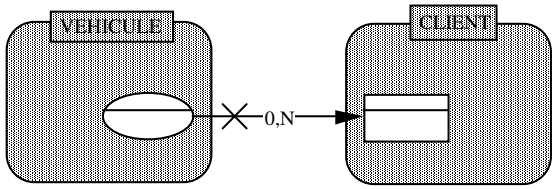
- *Opacité et Ouverture* : l'affectation des agents est définie exclusivement à partir de l'objet naturel SERVICE. Il est possible de déterminer le service d'affectation à partir de l'objet naturel AGENT.



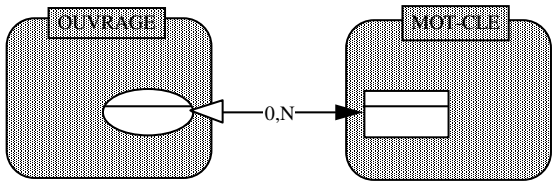
- *Gestion Dynamique et Apparence* : le propriétaire d'un véhicule est défini (avec la création possible du client) à partir de l'objet naturel VEHICULE. L'objet CLIENT voit les véhicules lui appartenant.



- *Gestion Dynamique et Fermeture* : le propriétaire d'un véhicule est défini (avec la création possible du client) à partir de l'objet naturel VEHICULE. L'objet CLIENT ne voit pas les véhicules lui appartenant.



- *Gestion Dynamique et Ouverture* : les mots-clés caractérisant un ouvrage sont définis à partir de l'objet naturel OUVRAGE (avec création possible des nouveaux mots-clés). Il est possible de relier un ouvrage avec un mot-clé à partir de l'objet naturel MOT-CLE.



La structure d'un objet naturel doit respecter trois règles fondamentales :
- tout objet naturel doit posséder une et une seule entité racine,
- toute entité ou association est accessible par un chemin unique depuis la racine,
- les objets naturels n'ont jamais de point commun, c'est-à-dire que les intersections sont toujours vides.

Ces trois règles impliquent que l'on doit construire les schémas entités-associations sans cycle et les objets naturels sous la forme d'un arbre ce qui peut parfois être contraignant. Par ailleurs il est fréquent que des objets qui paraissent naturels à l'utilisateur contiennent des entités ou des relations communes. Par exemple un bon de commande, une facture ou la gestion des clients font appel à l'entité Client. P-A Brès propose de régler ces contraintes par les liens frontière (héritage, généralisation/spécialisation, etc.) ce qui peut parfois gêner pour la validation auprès de l'utilisateur.

Ce modèle des objets naturels est intéressant sur quatre points :

- *il permet une validation plus efficace auprès de l'utilisateur* puisque les entités manipulées sont les siennes et non pas celles de l'informaticien,

- *il permet la génération du schéma de la base de données*. L'atelier Tramis peut en effet générer le schéma en SQL, Oracle, O2, etc.,

- *il permet la gestion dynamique du schéma généré*. Tous les liens qui ont été définis, de même que les contraintes de type, de cardinalité, etc., sont automatiquement gérés. Ceci signifie que les objets naturels sont non seulement un concept, mais également une réalité informatique avec leur propre comportement,

- *il permet, à partir des données, la génération de l'interface servant à la manipulation élémentaire de ces données*. Ce point est également très intéressant puisqu'il nous fournit, avec le point précédent, tout ce qu'il faut pour gérer correctement les données à la base. Notons que l'interface suit une logique d'utilisation puisque les données sont structurées suivant cette même logique. Si l'on prend l'exemple d'une gestion de bibliothèque (Figure 5.53), l'objet naturel ABONNE génère automatiquement la fenêtre de création d'abonnés représentée sur la figure 5.54.

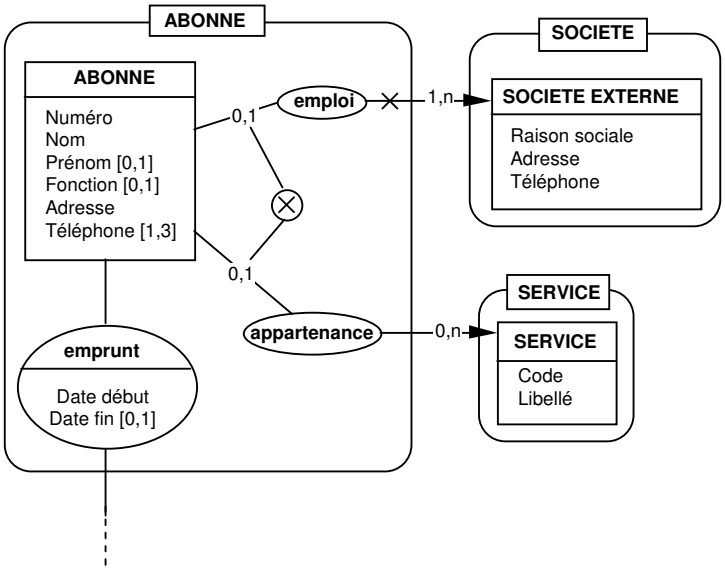


Figure 5.53 : Extrait du modèle des objets naturels d'une bibliothèque (extrait de [TRAMIS 92]). Le sigle ⊗ signifie qu'un abonné est soit dans une société soit dans un service.

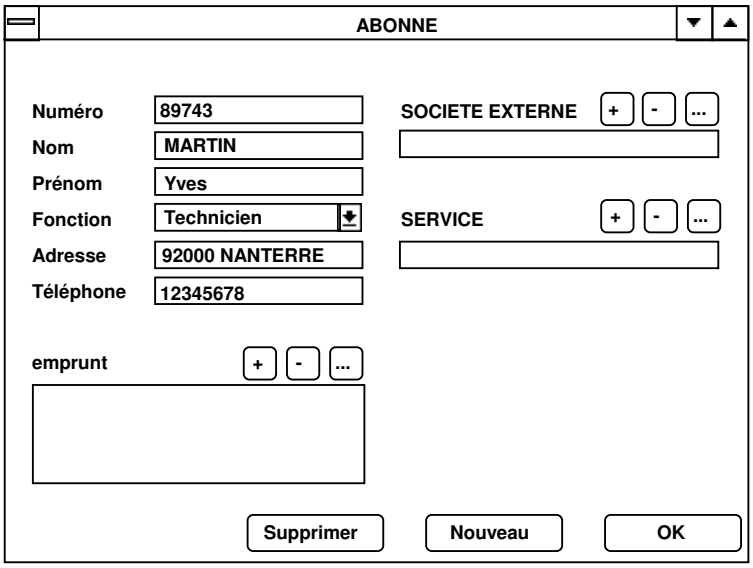


Figure 5.54 : Fenêtre de manipulation générée depuis l'objet naturel ABONNE

On constate que cette fenêtre reprend tous les attributs de l'entité ABONNE et qu'elle regroupe également tous les objets naturels qui en dépendent (SOCIETE EXTERNE et SERVICE). Étant donné les liens qui unissent ABONNE avec SOCIETE EXTERNE et SERVICE, il est possible d'ajouter (voire même de créer) dynamiquement des occurrences de ces entités directement depuis ABONNE (flèches noires). De même les fenêtres de SERVICE permettent de lister tous les abonnés pour un service donné ce qui est impossible depuis SOCIETE EXTERNE.

Le modèle des objets naturels est donc très intéressant pour la création d'applications interactives de type système d'information (on peut d'ailleurs envisager de l'étendre aux applications de type créatives ou expertes). Ce modèle nous fournit tout le nécessaire pour bâtir des traitements qui

soient robustes et modulaires. On peut en effet envisager l'architecture représentée sur la figure 5.55 :

- la couche de base est composée d'une base de données (relationnelle, objet, etc.) avec les traitements nécessaires pour les contrôles de type, d'intégrité, pour les créations, les suppressions, etc.

- la seconde couche (les objets naturels) repose sur cette base de données et intègre tous les traitements nécessaires à la gestion des objets naturels (création, consultation). Ces traitements font appel à ceux de la couche inférieure ; par exemple la création d'un abonné (objet naturel) fait appel à la création d'un abonné (dans la base de données) ainsi qu'à la création (ou la connexion) d'une société externe et d'un service (tous les deux objets naturels).

- la troisième couche contient les opérations qui manipulent un ou plusieurs objets naturels à la fois. C'est le cas par exemple pour la saisie d'une facture qui peut avoir besoin de l'objet naturel Client et de l'objet naturel Facture.

- la quatrième couche contient les procédures Diane+ telles que nous les avons décrites jusqu'à présent dans ce chapitre. Cette couche contient donc les procédures minimales, prévues et effectives qui permettent d'effectuer des traitements manipulant des objets naturels (par exemple la gestion des stocks fait appel aux articles, aux entrepôts, etc.).

- la cinquième couche est l'ensemble des buts à atteindre. Elle correspond donc à l'ensemble des tâches que l'application peut exécuter.

- la sixième et dernière couche correspond à l'ensemble des stratégies dont on dispose pour réaliser le but principal.

Cette architecture six couches est certes complexe, mais elle permet de bâtir des interfaces adaptées aux postes de travail et en fonction de la structure du système. On remarque en effet que plus un système est de type créatif, plus la frontière entre l'homme et la machine se situe bas (inversement plus un système est pré-planifié, plus la limite se situe haut). Dans cette architecture, les trois couches inférieures peuvent être vues comme l'ensemble des primitives nécessaires à l'utilisation des objets naturels sous-jacents à l'application (par exemple la gestion des stocks manipule les objets naturels Article et Lieu de Stockage dont la gestion au niveau élémentaire, pour la création, la consultation, etc., est située dans la couche de ces objets naturels). Ces trois couches sont supposées être invariantes quel que soit le poste de travail qui les utilisera. Par contre, chaque poste de travail peut manipuler différemment ces traitements (par exemple en fonction du niveau de confidentialité). Ceci est réalisé par la couche des procédures qui définit l'ensemble des opérations (avec les liens de précédence, les contraintes, etc.) nécessaires pour réaliser une tâche.
En fait cette architecture doit être vue depuis le haut. L'utilisateur veut atteindre un but. Pour cela il met en œuvre une stratégie (première couche) qui lui fera choisir tels ou tels sous-buts à réaliser (choix dans la deuxième couche). Les sous-buts étant déterminés et l'utilisateur étant connu, cela implique la mise en route des procédures correspondantes dans la troisième couche.

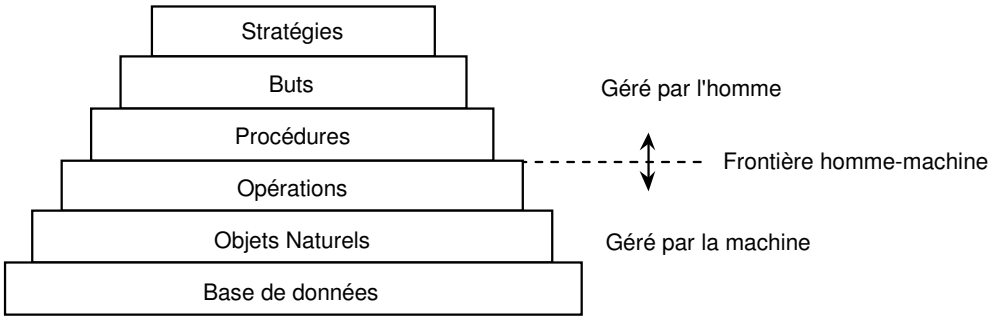


Figure 5.55 : Architecture six couches.
Cette architecture a l'avantage d'incorporer les objets naturels qui permettent une encapsulation des données et des vues différentes en fonction des personnes susceptibles d'utiliser ces données (concepteur du système ou utilisateur).

Dans les applications actuelles la couche de stratégie n'existe pas. Cela s'explique par le fait qu'il faut modéliser l'utilisateur, c'est-à-dire ses connaissances, son mode de réflexion, etc., mais on doit également pouvoir analyser le contexte dans lequel il évolue (le choix d'une stratégie dépend principalement de ces deux facteurs). A l'heure actuelle, notre système n'intègre pas encore cette couche.

Cette architecture nous ramène donc d'une certaine façon au modèle de Seeheim puisque nous avons d'un côté le noyau fonctionnel avec son interface, de l'autre nous avons la présentation et au centre le contrôle de dialogue (Figure 5.56).

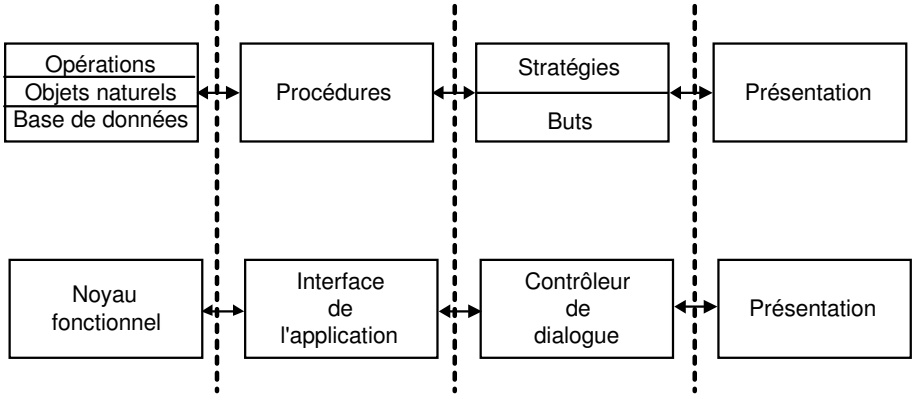


Figure 5.56 : Correspondance entre l'architecture six couches et le modèle de Seeheim.

Notons également que la possibilité d'avoir plusieurs présentations pour une même application reste vraie (Figure 5.57). Cela peut d'ailleurs se traduire par l'utilisation d'objets de présentation en relation avec les objets naturels comme nous le verrons dans le prochain chapitre.

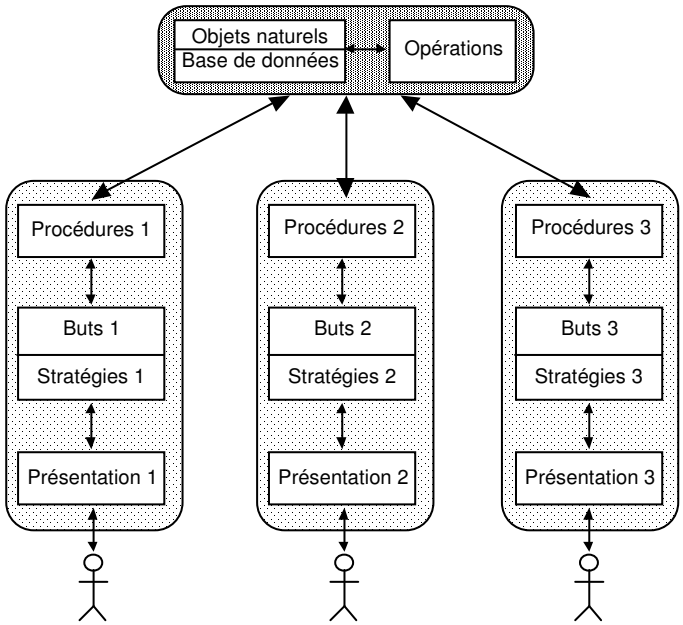


Figure 5.57 : Création de plusieurs interfaces pour le même noyau fonctionnel

5.3. Discussion

La remarque générale que l'on peut faire sur les méthodes basées sur les tâches est qu'elles n'intègrent généralement pas dans les tâches, contrairement à Diane+, la répartition du contrôle entre l'homme et la machine (par exemple dans MAD). Certaines méthodes tentent d'évoluer afin de s'adapter aux IHM, c'est le cas de Merise/2 qui apporte à Merise la notion d'objet et de cycle de vie. Cependant l'IHM n'est pas complètement intégrée et reste rudimentaire. De plus, rien n'est dit sur la représentation externe (par exemple est-il possible de la générer automatiquement ?) et sur les liens entre le conceptuel et l'externe (cohérence entre les deux ?). Merise et Merise/2 paraissent donc destinées à être exclusivement utilisées pour les systèmes d'information collectifs.
Merise est une méthode qui fournit un découpage rigoureux des tâches et des opérations par poste de travail, mais elle ne fait pas de différenciation sur les niveaux des utilisateurs. De plus elle ne permet pas d'intégrer la latitude décisionnelle de l'utilisateur contrairement à Diane+. Le dialogue homme-machine qu'elle fournit est un dialogue rigide et l'aide n'est pas intégrée. Avec Diane+, cette aide découle des spécifications du dialogue ; elle est capable de s'adapter automatiquement aux évolutions de ces spécifications.

Des trois "méthodes" orienté-objet que nous avons vues plus haut, celle des objets naturels nous semble la plus intéressante pour notre travail puisqu'elle nous permet d'une part de disposer d'un modèle de données très solide (bonne définition des liens et des structures des objets) et basé sur la logique d'utilisation, et d'autre part de permettre la génération de l'interface tout en disposant de la gestion automatique des objets et ceci également en logique d'utilisation. C'est pourquoi nous la reprenons en partie dans le chapitre suivant. La méthode d'analyse OOA fournit également des objets bien structurés, mais sans aucun lien véritable entre eux. Il serait peut-être intéressant de mélanger ces deux méthodes en gardant le concept d'objet naturel auquel on ajouterait la notion de méthodes privées et publiques. Ainsi la création d'un abonné (objet naturel) se décomposerait en une méthode publique Création et en plusieurs méthodes privées (Création d'un emprunt, Création d'un emploi, etc.).
Enfin la méthode de A. Rochfeld peut être vue en parallèle avec celle de P-A Brès puisqu'ils partent chacun de schémas entités-associations pour aboutir à un concept objet. Celle de

A. Rochfeld est plus puissante au niveau comportemental des objets (cycle de vie et graphe d'agencement) alors que celle de P-A Brès est plus orientée vers une manipulation efficace pour l'utilisateur. Ces deux méthodes peuvent bénéficier des apports de Diane+ en reprenant les méthodes des objets qu'elles contiennent et en les assemblant (avec d'autres opérations) sous forme de procédures Diane+. On a alors la possibilité de générer des interfaces qui sont à la fois efficaces au niveau des traitements procéduraux (par exemple la gestion des stocks) et au niveau de la gestion des données (par exemple la création des abonnés).

Toutes les méthodes énoncées dans ce paragraphe ont l'inconvénient commun de ne pas détailler les traitements qu'elles gèrent. Cette critique s'applique également à Diane+. La notation UAN [HARTSON 92] permet de remédier à ce problème puisqu'elle décrit précisément les interactions autorisées ainsi que le comportement que doit adopter l'application. Diane+ fournirait ainsi le comportement général de l'application (buts, procédures et enchaînements divers) et UAN décrirait le comportement à l'intérieur des opérations.

Diane+ peut donc servir de complément à des méthodes telles que Merise, MAD ou OOA. Elle peut être employée dans le développement d'une application (spécification et conception), mais elle peut également être utilisée pour la rénovation d'applications.

5.3.1. Rénovation d'applications

La rénovation consiste à créer une nouvelle application à partir d'une application existante. Pour que cela soit possible, plusieurs conditions doivent être réunies :

- *il existe une analyse complète de l'application existante*. Cette analyse fournit les fonctionnalités de l'application, le public visé, les postes de travail, l'organisation des services, la répartition des tâches entre les postes de travail ainsi qu'entre l'homme et la machine, etc. A partir de cette analyse, on va être capable de dire s'il est possible de modifier le comportement de l'application et de l'interface homme-machine. Le second cas concerne plus précisément le "face-lifting" dans lequel Diane+ n'intervient pas directement. Ce secteur est plutôt réservé aux outils de prototypage et de maquettage ou bien encore aux UIMS. Par contre, modifier le comportement de l'application est directement lié aux méthodes de conception. Cette modification peut être pratiquée à deux niveaux : interne ou externe (vis-à-vis de l'utilisateur). Ces deux niveaux sont en forte corrélation puisque le comportement interne a très souvent une répercussion à l'écran, donc vis-à-vis de l'utilisateur. Inversement, modifier les possibilités d'intervention de l'humain sur le système peut impliquer la modification du comportement interne.
  Le comportement interne correspond principalement aux enchaînements et à l'exécution des opérations. Diane+ ne peut pas intervenir dans ce dernier cas puisqu'ici seule la programmation entre en jeu (on changera par exemple un algorithme ou bien encore des structures de données). Par contre, Diane+ peut être utilisée pour modifier les séquencements des opérations. De manière générale, on ne pourra que restreindre la latitude décisionnelle de l'utilisateur.

- *l'application existante possède une latitude décisionnelle non nulle*. Dans le cas contraire, cela signifie que l'utilisateur est esclave de l'application. Il est donc probable que les traitements soient implémentés de telle sorte qu'ils s'appellent directement les uns les autres. Modifier ces enchaînements risque d'entraîner des perturbations importantes dans le fonctionnement de l'application. Dans le cas où l'application possède une latitude décisionnelle non nulle, il est toujours possible de la restreindre (si l'on veut l'augmenter, on retrouve le même problème que précédemment, c'est-à-dire des risques d'effets de bord). On

pourra par exemple enchaîner des opérations qui ne l'étaient pas ou encore rendre obligatoires des opérations facultatives.

- *l'application existante possède un noyau fonctionnel à base de primitives*. Cette décomposition en primitives implique une indépendance entre les primitives et donc permet de bâtir des procédures qui font appel à ces primitives tout en restant indépendant de l'implémentation de ces dernières. Il est alors possible de choisir le degré de latitude décisionnelle de la nouvelle application. Cette situation est la plus intéressante pour la mise en œuvre de Diane+. Le regroupement des primitives en procédures correspond en effet en partie aux caractéristiques de Diane+ ; il est ainsi possible de créer des procédures prévues et minimales en fonction du niveau des utilisateurs sans se soucier du comportement interne des primitives. Ceci suppose que ces dernières sont implémentées de telle sorte qu'elles n'aient pas de feed-back à l'écran et qu'elles ne tiennent pas compte d'un éventuel niveau de l'utilisateur (débutant, occasionnel ou encore de haute confidentialité). Dans le cas contraire, la rénovation sera beaucoup plus difficile à réaliser puisque l'on devra différencier les primitives qui sont indépendantes de ces feed-backs et de ces intégrations de niveaux, de celles qui ne le sont pas. On pourra alors ajouter de nouvelles primitives qui viendront combler ces lacunes tout en faisant appel aux primitives sans feed-back et sans intégration de niveau quelconque.

Un travail antérieur [BARTHET 92] a montré comment il était possible de rénover une application de France Télécom. L'objectif de ce travail était de voir comment on peut, à partir d'une application existante, implantée sur un serveur et fonctionnant sur des terminaux et des PCs utilisés comme terminaux alphanumériques, parvenir à faire fonctionner cette même application sur ces PCs avec une interface graphique. Plus précisément France Télécom voulait à la fois disposer de l'ancienne application, mais cette fois sous environnement graphique, et de plus implémenter de nouvelles fonctionnalités qui soient disponibles uniquement sous cet environnement. L'étude de l'application a permis de voir qu'elle était bâtie en partie sur des primitives qu'il a ensuite été possible de réutiliser pour l'environnement graphique. L'étude finale n'a porté que sur une partie de l'application et la latitude décisionnelle dans cette étude a été décrite par l'intermédiaire de Diane.

5.3.2. Développement d'applications

L'utilisation la plus pertinente de Diane+ intervient dans le développement complet d'une application. De par sa nature, Diane+ ne peut évidemment pas couvrir complètement le cycle de vie de l'application ; elle intervient cependant à tous les stades du développement, plus précisément :

- dans la spécification où elle permet de spécifier la répartition du dialogue entre l'homme et la machine, en fonction des postes de travail et du niveau des utilisateurs,
- dans la conception où elle affine ce dialogue par l'intermédiaire de contraintes sur les opérations ou bien encore d'attributs tels que la nature ou le type des opérations,
- dans la réalisation où elle prépare une génération de l'interface ainsi que celle des procédures et des opérations décrites dans les phases précédentes.

Diane+ peut être utilisée avec une approche montante ou une approche descendante. Dans le premier cas elle s'appuie sur une couche d'objets telle que les OPACs que nous décrirons plus loin ou bien encore sur des objets issus d'une méthode telle que celle OOA. Diane+ ajoute à cette couche d'objets les notions de procédures et de buts qui permettent d'agencer ces objets en fonction des postes de travail. Avec l'approche descendante, Diane+ décrit l'application par

l'intermédiaire de buts et de procédures qu'elle affine jusqu'à rejoindre la structure des données manipulées, cette structure pouvant être implémentée ou non en objet.
Quelle que soit l'approche employée, Diane+ intervient donc dans les trois représentations d'une application (cf Chapitre 1, § 2), c'est-à-dire :

- dans la représentation conceptuelle où elle définit :
  - les modes de dialogue grâce aux opérations et aux procédures,
  - la connexion entre les opérations et les données grâce à l'utilisation des objets OPACs.
- dans la représentation externe lors de la génération de l'interface,
- dans la représentation interne grâce à la génération des objets nécessaires au fonctionnement de l'application (principalement les opérations, les procédures, les fenêtres et les règles d'enchaînement).

Pour que le développement d'une application soit réalisable avec Diane+, il faut que l'application soit basée sur un ensemble de primitives constituant le noyau fonctionnel afin de rendre possible la création des procédures. Deux choix sont alors possibles pour la constitution de ce noyau fonctionnel :

- les primitives ne font pas appel à la représentation externe et n'exécutent que des calculs sur les valeurs des données. Le concepteur a alors la charge de gérer lui-même la représentation externe (affichage, rafraîchissement, etc.). Cette gestion peut être facilitée par des primitives fournies par l'environnement de travail (gestion des menus, du scrolling, des sélections dans les listes, etc.).
- les primitives découlent des objets qui constituent une part du noyau fonctionnel et qui sont capables de gérer eux-mêmes leur représentation externe. Dans ce cas le concepteur ne s'occupe que de capturer leur valeur ou de leur envoyer une valeur. Prenons l'exemple d'une donnée constituée d'une liste. Cette liste se représentera à l'écran par une "listbox". Pour ajouter une valeur à cette liste, le concepteur n'aura qu'à écrire "liste add: élément" ce qui provoquera automatiquement l'ajout de l'élément à la liste ainsi que le rafraîchissement de la listbox. Ce principe de gestion automatique de la valeur et de sa représentation externe (en maintenant la cohérence) n'est autre que le modèle PAC. Nous aborderons plus loin ce point en détail.

6. Des tâches aux objets

L'approche objet intervient à présent à tous les stades dans le développement des applications, c'est-à-dire depuis la spécification jusqu'à l'implémentation, ceci étant facilité par les nombreuses méthodes orienté-objet associées [BAILIN 89], [COLBERT 89], [COAD 90], [GIBSON 90], [BOOCH 91], [SCHLAER 91], [RUMBAUGH 91]... Cette couverture du cycle de vie existe depuis longtemps avec l'approche orientée tâche, mais on s'est aperçu que celle-ci n'était pas toujours facile à mettre en œuvre pour l'implémentation surtout depuis que l'on cherche à réaliser des applications de plus en plus interactives. Or l'approche orienté-tâche (et but) a fait ses preuves. Les ergonomes savent bien qu'il est plus facile de décrire un travail en termes de tâches que l'on exécute et de buts à atteindre plutôt qu'en termes d'objets que l'on manipule. Par ailleurs décrire un travail par des tâches permet de mieux appréhender et surtout de valider le modèle cognitif de l'utilisateur. Le modèle tâche est donc avantageux pour les premières phases dans la création d'applications informatiques. La planification hiérarchique permet par exemple de bien décomposer les traitements à réaliser et de les agencer en fonction des buts à atteindre. Ce type de décomposition est aisément réalisable dans le cadre de tâches pré-planifiées telles que les tâches bureautiques [SEBILLOTTE 88] et également dans le cadre de tâches expertes. On peut ainsi obtenir des décompositions en fonction des postes de travail ou encore en fonction des types d'utilisateurs. Par ailleurs les méthodes de systèmes à base de connaissance comme KADS

[HICKMANN 89] [BRUNET 91] permettent d'extraire ces décompositions à la suite d'échanges avec les experts (interviews, relevés, questionnaires, etc.). L'orientation tâche de ces méthodes provient du fait qu'elles "collent" aux représentations cognitives des utilisateurs.

De l'autre côté de la frontière qui sépare la spécification de la conception (Figure 5.58), les informaticiens conçoivent et implémentent les applications en utilisant l'approche objet. La raison de cette orientation est double. Tout d'abord il est plus facile de concevoir et d'implémenter en termes d'objets plutôt qu'en termes de procédures. L'approche objet possède en effet des caractéristiques qui sont très intéressantes sur le plan du génie logiciel, principalement pour les interfaces graphiques (réutilisation, encapsulation, robustesse, etc.). Ces mêmes caractéristiques sont plus difficiles à mettre en œuvre avec une programmation de type procédural. La deuxième raison est que les outils informatiques disponibles actuellement convergent vers une utilisation intensive et totale de l'approche objet ce qui conforte les informaticiens dans leur choix. Les UIMS par exemple utilisent les environnements de travail qui sont basés eux-mêmes sur l'approche objet, les langages de programmation sont entièrement objet ou possèdent des extensions objet, etc.

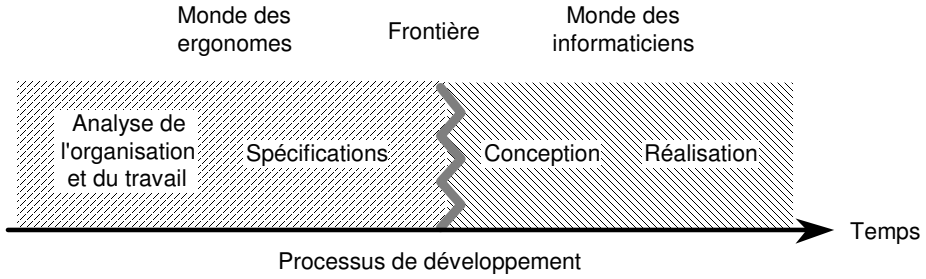


Figure 5.58 : Processus de développement d'une application et démarcation des deux mondes qui le composent.

Chacun de ces deux mondes (ergonomes et informaticiens) a donc ses méthodes de travail. Certaines de ces méthodes ont fait leurs preuves depuis longtemps alors que d'autres n'en sont qu'à leur début. Notre propos ici n'est pas de débattre sur leur validité mais de voir s'il est possible de les coordonner. La disparité apparente de ces deux mondes oblige en effet les ergonomes à acquérir une bonne connaissance technique de l'informatique s'ils veulent réaliser des applications en utilisant leur approche orienté-tâche. A l'opposé les informaticiens désireux de créer des applications sans l'aide des ergonomes ont tendance à négliger l'aspect ergonomique dans les spécifications et dans la conception des applications. Nous cherchons donc ici à voir s'il est possible de traduire le travail des ergonomes dans l'approche objet utilisée par les informaticiens. Pour cela nous nous concentrons sur le passage du modèle tâche-but au modèle objet en comparant ces modèles au niveau de la statique et de la dynamique.

6.1. Comparaison de la statique

Le modèle tâche-but regroupe deux notions fondamentales, les buts et les tâches. Le modèle tâche fournit une décomposition fonctionnelle de l'application sans se préoccuper des buts. Les seuls liens qu'il autorise permettent de structurer en macro-opérations les opérations qui composent les tâches. A l'opposé le modèle but reprend la décomposition fonctionnelle du modèle tâche et assemble les opérations en fonction des buts à atteindre sans se préoccuper du comportement des opérations. La majorité des concepts composant la statique du modèle tâche-but peuvent se traduire dans le modèle objet. Plus précisément :

- *les buts* sont des objectifs à atteindre ou des états dans lesquels le système doit se trouver à la fin des opérations. Nous pouvons différencier deux types de buts : ceux qui sont évaluables par la machine et ceux qui ne le sont pas. Les seconds sont des buts d'ordre général tels que écrire un document ou faire un dessin. Leur signification est difficile à représenter en machine car déjà difficile à appréhender sur un plan cognitif. Que signifie en effet "faire un dessin" ? Est-ce dessiner un trait, un carré, ou un ensemble de symboles représentant quelque chose de concret tel un animal ou un paysage ? De même "écrire un document" peut être considéré atteint dès que l'on a tapé un caractère ou une ligne si l'on considère qu'un document est un ensemble de caractères. Mais on peut également voir un document comme un texte syntaxiquement et sémantiquement correct. Dans ce cas il est très difficile d'évaluer ce but automatiquement par la machine.
  Le second type de buts est plus appréhendable par la machine et peut être représenté par l'intermédiaire de variables dans le modèle objet. Ces buts correspondent en général à des changements d'état des données. Par exemple le but "enregistrer un client" peut se traduire par un booléen ClientEnregistré dont la mise à jour se fera par l'intermédiaire d'opérations choisies au préalable et qui seront les seules à pouvoir le modifier. Les buts évaluables sont plus faciles à représenter pour les tâches pré-planifiées que pour les tâches créatives puisque pour ces dernières les séquences d'opérations ne sont pas connues à l'avance.
  En résumé, les tâches pré-planifiées sont composées d'une part de buts généraux difficilement représentables en machine, et d'autre part de sous-buts dont l'évaluabilité augmente au fur et à mesure de la décomposition.

- *les tâches* regroupent les opérations nécessaires à la réalisation d'un but. La première idée que l'on peut avoir est de traduire ces opérations par l'intermédiaire de méthodes dans le modèle objet. Cette idée est intéressante car elle permet d'utiliser les aspects d'héritage et de polymorphisme de la programmation objet, mais elle a l'énorme inconvénient de figer la répartition des opérations dans des classes représentant les tâches. Or une opération peut être utilisée par plusieurs tâches et la façon de procéder que nous venons d'évoquer nous oblige à une redondance dans l'écriture des méthodes. Par ailleurs la représentation des tâches sous forme d'objet n'est pas aussi simple. Comment traduire en effet les tâches et les liens qui les unissent ? On peut songer à définir une classe par tâche[84] et grouper les opérations (méthodes) par classe. La capacité de hiérarchisation des langages objet peut être utilisée pour traduire la hiérarchie entre les tâches, mais cette solution implique également une rigidité complète au niveau de cette hiérarchisation. L'arbre qui en résulte ne peut donc pas évoluer au cours des sessions. De plus, à chaque nouvelle application, le concepteur est obligé de créer un nouvel arbre, c'est-à-dire un nouvel ensemble de classes et de sous-classes. Une idée qui paraît plus judicieuse est d'utiliser une classe Tâche générique[85] qui contiendrait la structure générale des tâches ainsi que les méthodes nécessaires à leur gestion (création, modification des liens, etc.). Cette seconde solution permet une plus grande évolubilité puisque les instances de cette classe peuvent être modifiées à volonté, y compris durant l'exécution de l'application. Ceci permet par exemple à l'utilisateur de créer une nouvelle tâche à partir d'opérations. Dans notre travail, une tâche contient une procédure minimale, une procédure prévue et plusieurs procédures effectives. La classe Tâche doit donc être capable de gérer cette caractéristique.

[84] L'outil 3DKAT utilise ce principe en définissant comme une instance de classe.

[85] Plusieurs travaux utilisent déjà cette notion de tâche générique. C. Pierret-Golbreich [PIERRET 88] et D. Scapin [SCAPIN 89] définissent par exemple une tâche comme un objet générique caractérisé par ses attributs "état initial", "état final" et "opérateur".

- le contenu *des opérations* diffère suivant la nature des tâches. Dans le cas de tâches simples, les opérations contiennent des traitements procéduraux (qui s'appliquent sur un ensemble de données) et des actions élémentaires (actèmes[86]) pour chacune de ces données. Dans le cas de tâches complexes, les opérations ne contiennent que les actèmes puisque les inférences sont laissées à la charge de l'utilisateur. Dans les deux cas, les traitement procéduraux peuvent être traduits par l'intermédiaire des méthodes des langages objets, ceux-ci possédant toujours des structures de contrôle (itérations, branchements conditionnels, etc.). La décomposition fonctionnelle d'une application fait apparaître des notions additionnelles telles que les conditions de déclenchement ou les synchronisations (qui concernent surtout la dynamique). Ces conditions peuvent se traduire sous une forme booléenne dans le modèle objet. D'après ce que nous avons dit précédemment, la question est de savoir où stocker ces opérations. Si l'on reprend l'idée d'une classe Tâche générique, les opérations pourraient être stockées par l'intermédiaire de variables dédiées à leur manipulation. On peut envisager des variables contenant le texte d'une méthode sous une forme ASCII ou semi-compilée (suite de codes hexadécimaux par exemple), ce texte étant par la suite interprété et évalué par l'environnement (comme avec Smalltalk par exemple).
  Des méthodes telles que Diane+ permettent d'associer des types ou des natures aux opérations. Les classes permettraient de représenter ces natures et ces types en associant une classe à chaque nature et à chaque type, ceci ne posant pas de problème puisque le nombre de natures et de types est connu à l'avance et ne risque pas de changer durant les sessions de travail. Ainsi la classe Saisie regrouperait toutes les saisies et pour chaque méthode, on ferait apparaître les types des données manipulées. Or le nombre de classes risquant de croître très rapidement à cause du nombre de combinaisons possibles, il est nécessaire de définir un critère de classification. On peut choisir de créer une classe par nature (suppression, création, consultation, etc.) et définir les types par l'intermédiaire de variables.
  Une telle utilisation des classes nous ramène au problème précédemment énoncé avec les tâches, c'est-à-dire la rigidité. En effet classer les opérations suivant tel ou tel critère nous oblige à les écrire sous la forme de méthodes attachées à des classes, c'est-à-dire qu'il est possible de les lister pour chaque classe. Or cela peut s'avérer illogique puisqu'une opération peut exécuter le même traitement dans des conditions différentes. Par exemple une opération peut être facultative ou obligatoire suivant le contexte, de même qu'elle peut être déclenchée par la machine ou par l'utilisateur. Or si elle a été écrite dans l'une ou l'autre des classes, il devient impossible de changer ces caractéristiques. Pour remédier à ce genre de problème, on peut envisager l'utilisation d'une classe Opération générique[87], qui de façon similaire à la classe Tâche, permettrait de créer des instances d'opérations. La classe Opération comporterait alors les méthodes nécessaires à la création d'une opération, au changement de ses caractéristiques, à son comportement en fonction de ses caractéristiques, etc., et chaque instance pourrait alors être rattachée à une procédure ou une autre opération. On peut imaginer par exemple que chaque instance tient à jour une liste de pointeurs qui lui permet de savoir à quels autres objets elle est reliée (opération, procédure, etc.) et pour chacun d'eux on dispose de la liste des caractéristiques de l'opération. Ainsi il n'y a pas de redondance dans le code pour les traitements et chaque opération peut être utilisée autant de fois qu'on le souhaite (Figure 5.59).

[86] Nous reprenons ici le terme employé par KOD [VOGEL 88] qui appelle également les traitements des "inférences".

[87] Merise/2 [PANET 91] utilise déjà la notion d'opération générique par exemple en définissant une opération générique Déclaration de sinistre qu'elle spécialise en plusieurs opérations suivant le montant du sinistre. La notion de généricité est ici utilisée pour factoriser des caractéristiques sur les opérations et sur les données que ces opérations génériques manipulent.

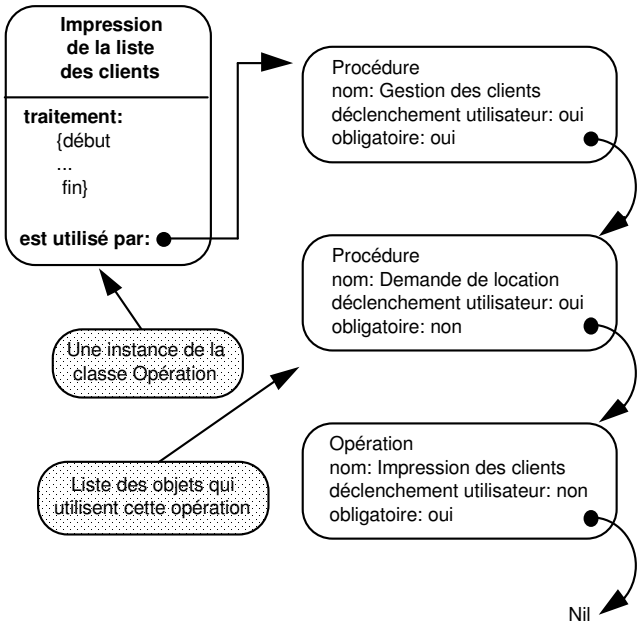


Figure 5.59 : Représentation d'une opération dans le modèle objet

- *les données* sont le cœur du problème. Alors que dans le modèle tâche-but on associe des données aux traitements (la facturation utilise un client et des articles), dans le modèle objet l'inverse se produit puisque les données encapsulent les traitements. Or le résultat attendu est identique : on veut créer une facture avec un client et des articles. Certains chercheurs tentent de passer d'un modèle entité-association à un modèle objet (cf § 5.2.2) [MAKSAY 91], [ROCHFELD 91b], [ROCHFELD 92]. Pour cela ils utilisent des règles qui leur permettent d'une part de transformer les associations en méthodes ou en attributs et d'autre part d'associer ces méthodes et ces attributs à certains objets (par exemple en fonction de la nature de la méthode). Cette voie est intéressante mais des choix peuvent s'avérer difficiles. Ainsi la création d'une facture doit-elle être rattachée à la classe Client ou à la classe Facture ? D'après [MAKSAY 91] l'objet principal est la facture et c'est donc elle qui bénéficie des méthodes associées à la facturation. Par ailleurs la transformation en méthodes et en objets des relations entre les entités gomme les liens qui unissent les données. Une autre solution est apportée par Rochfeld [ROCHFELD 92] qui transforme les relations en objets et qui les incorpore dans d'autres objets. Par exemple l'objet Devis comporte deux objets relations qui concernent les heures et les articles utilisés dans le devis.
  Les données sont également souvent accompagnées de notions diverses telles que les contraintes d'intégrité. Celles-ci peuvent être traduites par des contraintes sur les types et par des méthodes associées aux classes de données. On retrouve d'ailleurs cette notion de gestion autonome des données dans les objets naturels (cf § 5.2.3).

6.2. Comparaison de la dynamique

La dynamique du modèle tâche-but passe principalement par les notions de temps, d'enchaînements et d'événements .

- *le temps* est peut-être la notion la plus difficile à traduire dans le modèle objet. Elle apparaît dans le modèle tâche à la fois sous la forme de synchronisations (l'opération A et l'opération B doivent être achevées pour que l'opération C soit disponible), de temporisations

(déclencher l'opération A tous les 15 jours ou bien encore déclencher l'opération C une semaine après l'opération B) et de délais d'exécution liés aux temps de calcul. La représentation du temps dans le modèle objet oblige à le dissoudre à travers des messages et des conditions de déclenchement rendant ainsi difficile un éventuel retour du modèle objet au modèle tâche.

- *les enchaînements* entre les opérations ainsi qu'entre les tâches peuvent être traduits directement par l'intermédiaire des messages. Cette situation correspond par exemple aux enchaînements câblés entre opérations dans les procédures minimales. Les enchaînements peuvent également être traduits indépendamment des messages. Cette situation a plusieurs solutions. Nous n'en retiendrons qu'une seule ici : les règles d'enchaînement. Elles permettent de représenter les liens entre les buts, entre les sous-buts et entre les opérations. Le formalisme MAD [SCAPIN 89] par exemple peut très bien être représenté sous forme de règles de même que le formalisme Diane+. L'avantage des règles qui nous intéresse ici est leur capacité à évoluer dans le temps. Cette évolution peut apparaître dans les séquencements entre les opérations et au niveau de la stratégie. En fonction du contexte on peut envisager de disposer de règles qui permettraient d'affecter des poids aux stratégies existantes en fonction de l'évolution du contexte ou de l'utilisateur.

- le modèle tâche intègre la notion *d'événement.* Or l'une des caractéristiques du modèle objet est la communication par envois de messages qui sont proches du concept d'événement. Il est donc possible de représenter les événements par l'intermédiaire de messages. Il faut alors différencier deux types d'événements :

  - les événements internes, comme par exemple l'enregistrement d'un client, qui sont en fait des messages que s'envoient des objets. Ce type d'événement peut déclencher des opérations ou bien mettre à jour des valeurs. Dans le premier cas nous trouvons (Figure 5.60) :

    i. des événements subjectifs[88] tels que Demande abonné. En général ces événements ne seront pas explicitement implémentés car ils sont en forte corrélation avec les opérations. Par exemple l'événement Demande abonné est intégré dans le déclencheur de l'opération Demande de prêt ce qui permet à l'utilisateur de déclencher effectivement l'opération quand un abonné se présente.
    ii. des événements qui testent une donnée, par exemple livre non rendu consiste à regarder si le livre qui est demandé est déjà prêté ou non. Dans l'affirmative l'utilisateur déclenche une relance auprès de la personne qui a emprunté le livre.

  Dans le second cas nous trouvons des événements qui résultent d'opérations et qui modifient une donnée. C'est le cas par exemple avec livre prêté en sortie de l'opération enregistrement prêt.

[88] Ces événements sont porteurs d'informations qui ne sont pas en relation directe avec les données. Ce peut être par exemple une date qui permettra de déclencher une opération calendaire.

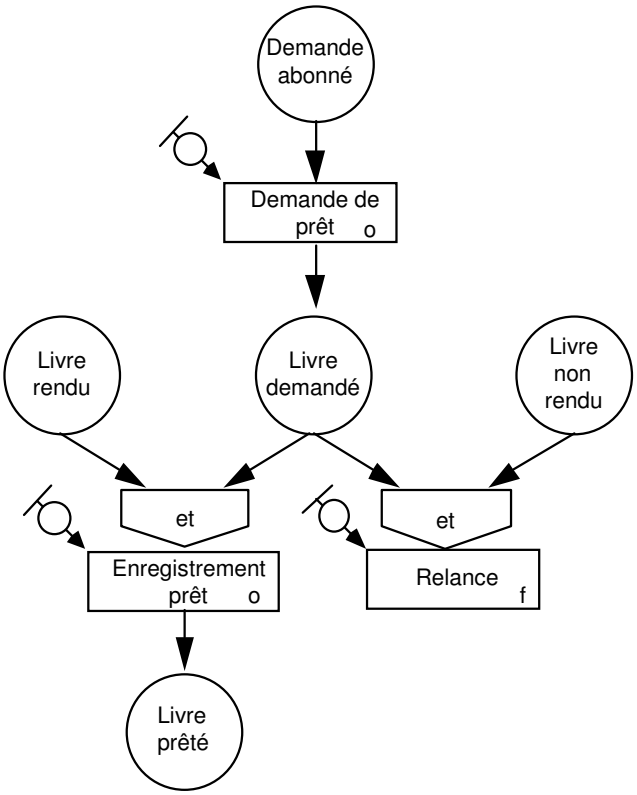


Figure 5.60 : Événements et opérations

- les événements externes (par exemple un clic souris de la part de l'utilisateur) qui doivent être traduits en messages interprétables par les opérations. Cette traduction peut revenir au gestionnaire de l'interface ou au gestionnaire du dialogue. Dans le premier cas nous trouvons des événements tels que des clics sur une fenêtre à l'arrière-plan ou un redimensionnement de fenêtre, dans le second cas nous trouvons par exemple des ouvertures de menus, des clics dans des listes ou sur des boutons, etc. Tous ces événements devant pouvoir être interprétés par l'application, il est intéressant de les exprimer avec un niveau d'abstraction élevé. On les retrouvera par exemple sous une forme semblable à sélection de la commande C dans le menu M ou encore sélection de l'élément E dans la liste L.

Conclusion sur la comparaison tâche-objet

Le modèle tâche et le modèle objet ne sont donc pas incompatibles. Bien que leur utilisation dépende des objectifs à atteindre (par exemple une validation du modèle cognitif ou une implémentation robuste) et que chacun d'eux dispose de caractéristiques qui lui sont propres (par exemple la modélisation cognitive de l'utilisateur pour le modèle tâche et le respect des critères du génie logiciel pour le modèle objet [MEYER 90]), nous avons vu que leur pouvoir d'expression est très voisin et qu'il paraît possible de traduire le modèle tâche dans le modèle objet sans perdre de ce pouvoir d'expression. V. Normand utilise une autre solution dans [NORMAND 92a]. Son système manipule à la fois les objets et les tâches, ainsi que la dimension de fonctionnement et la dimension d'utilisation. Les objets représentent la première dimension, les tâches avec les "espaces de travail" et les "perspectives" représentent la seconde, tous ces concepts étant réunis et en étroite relation au sein du même système.

La traduction dont nous avons parlé ici se fait chronologiquement, mais il est également possible d'envisager un autre type de traduction. En effet il est intéressant de considérer une équivalence entre le modèle tâche et le modèle objet tout au long du cycle de développement d'une application puisqu'on peut effectivement développer entièrement une application dans l'un ou l'autre de ces deux modèles. Cette équivalence permettrait d'utiliser au mieux les acquis informatiques tant sur le plan des méthodes que sur celui des outils et des personnes. Ceci pourrait faire l'objet d'une étude particulière.

Diane+, quant à elle, mélange le modèle tâche-but et le modèle objet. D'une part, elle intègre dans les spécifications à la fois des données structurées en objets (voir le modèle OPAC dans le paragraphe suivant) et une description de tâches, d'autre part, lors de la génération, les concepts issus du modèle tâche sont traduits sous forme d'objets.

7. Modélisation des données avec OPAC

Nous avons vu que le passage des tâches aux objets n'est pas sans difficulté (par exemple la dilution de la notion de temps). Parmi les problèmes qu'il pose, deux nous intéressent plus particulièrement ici : la répartition des traitements et la gestion de la dynamique du système. La solution que nous adoptons dans cette thèse concerne la modélisation des données par l'intermédiaire d'un modèle dérivant du modèle PAC [COUTAZ 88]. Ce nouveau modèle, appelé OPAC, a pour particularité de regrouper les données avec les traitements élémentaires qu'elles autorisent (création, suppression, affichage, etc.). Ainsi la gestion des tâches (c'est-à-dire la dynamique et l'aide principalement) se retrouve déportée dans les opérations et les procédures Diane+ comme nous le verrons plus loin. Le modèle OPAC permet de représenter efficacement les données sur le plan externe. Il rend possible la création d'une couche de base constituée d'objets capables de gérer de manière élémentaire à la fois leur valeur (Abstraction) et leur représentation externe (Présentation). Cette gestion passe par le Contrôle qui s'occupe également de maintenir la cohérence entre l'Abstraction et la Présentation.

Notre modèle OPAC utilise le modèle PAC conjointement avec le modèle des objets naturels. La raison de cette union est double :

- les objets naturels fournissent une vue sur les données avec une logique d'utilisation par rapport à l'utilisateur. En cela leur utilisation est très intéressante. Pourtant, dans leur version actuelle, ils possèdent les inconvénients suivants :

    - ils sont basés essentiellement sur le modèle entité-association. Nous avons vu qu'ils obligent le concepteur à découper le système en objets naturels qui ne possèdent pas d'intersection et qui sont issus d'une seule racine.

    - ils ne sont pas assez puissants quant aux contrôles qu'ils fournissent par rapport aux interactions de l'utilisateur. Lors de la génération de leur représentation externe, ils fournissent des fenêtres ne permettant que des manipulations élémentaires (création, consultation, suppression, etc.), et sans possibilité de contrôle[89]. Les widgets qui représentent les attributs des données sont par exemple des zones toujours accessibles à l'utilisateur. Les objets naturels ne font donc pas de différence quant aux droits d'accès

[89] Les objets naturels gèrent toutefois les contraintes d'intégrité et les vues qu'ils offrent quant aux autres objets naturels.

en fonction des utilisateurs (procédure minimale, prévue et effective avec Diane+), des contextes d'exécution (opération obligatoire ou non par exemple) et de la nature des opérations (saisie, consultation,...).

- PAC fournit des objets autonomes et bien structurés. Pourtant on peut lui reprocher :

  • de ne pas fournir des objets indépendants du contexte de travail. Prenons le cas d'une application qui contrôle un ensemble complexe de données (par exemple un réseau de circulation de rames de métro ou bien encore une centrale hydraulique). L'état de l'application est représentée à l'utilisateur par l'intermédiaire d'un panneau de contrôle (Figure 5.61). Chaque élément de ce panneau (jauge, interrupteur, aiguillage, feu de signalisation) peut être représenté par un agent PAC (Figure 5.62.a et 5.62.b).

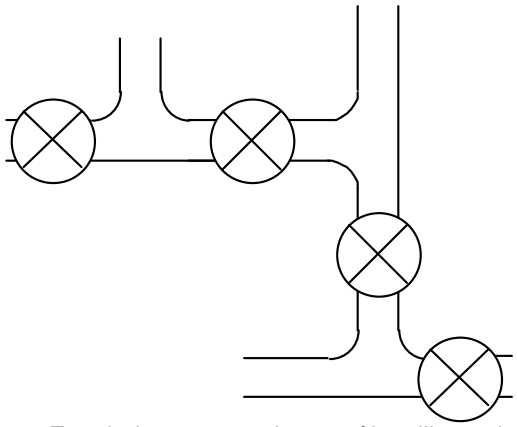


Figure 5.61 : Extrait de panneau de contrôle utilisant des vannes

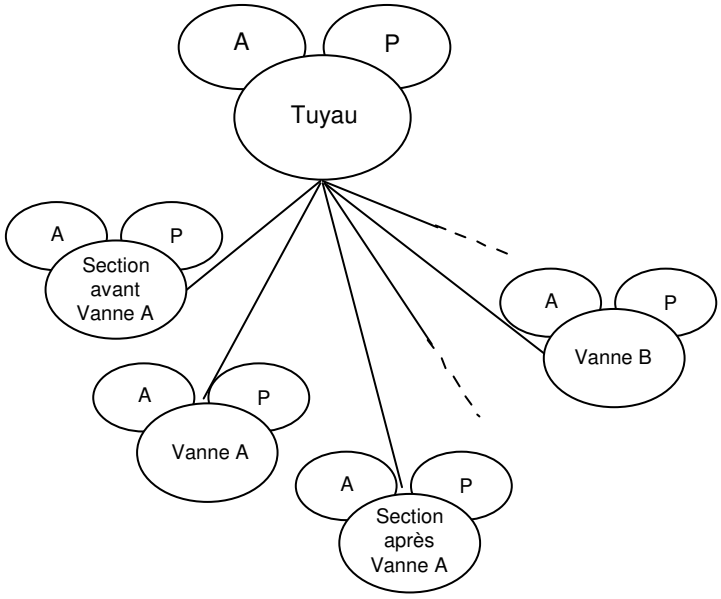


Figure 5.62.a : Représentation avec PAC des éléments intervenant dans la figure 5.62.b

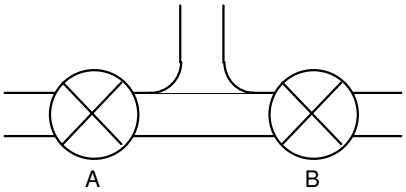


Figure 5.62.b : Représentation des agents PAC de la figure 5.62.a.
Les deux vannes et le tuyau sont représentés chacun par un agent PAC.

Si l'on suppose que ce panneau est disponible sur chaque poste de travail (les différentes personnes qui contrôlent l'application sont par exemple géographiquement dispersées), l'utilisation d'agents PAC ne possédant chacun qu'une Présentation, implique que le panneau sera le même pour chaque poste[90]. Or il serait intéressant de représenter ce panneau en fonction du poste de travail où il apparaît ainsi qu'en fonction du niveau de l'utilisateur. Ainsi chaque utilisateur n'aurait accès qu'aux agents PAC qu'il peut manipuler. On pourrait par exemple visualiser les agents PAC inaccessibles tout en interdisant leur manipulation ou bien encore ne pas les représenter du tout. Il est bien sûr possible de représenter ces situations avec le modèle PAC, mais cela implique la dilution du contexte de travail à travers les méthodes fournies par les agents PAC[91].

La solution que nous proposons pour ce genre de situation est la suivante : les opérations manipulent des agents PAC qui proposent quant à eux des traitements indépendants du contexte de travail. Si une opération est présente dans une procédure, les agents PAC associés seront représentés à l'écran et disponibles pour l'utilisateur (sous réserve de modification d'accès par l'opération). Les agents PAC non présents dans une procédure ne seront pas représentés à l'écran[92], ils sont donc inaccessibles pour l'utilisateur. Si l'on reprend l'exemple de la figure 5.62.b, les vannes proposent deux méthodes : ouvrir et fermer. La procédure de la figure 5.63.a contient ces deux opérations. Les vannes apparaîtront donc sur le panneau (Figure 5.63.b). La procédure de la figure 5.64.a ne contient pas ces deux opérations pour la vanne A (le poste de travail associé n'a pas le doit de manipuler cette vanne). La vanne A n'apparaîtra donc pas sur le panneau[93] (Figure 5.64.b).

[90] Il reste cependant possible d'utiliser des agents PAC qui fournissent plusieurs Présentations pour un même concept.

[91] Pour éviter la dilution, on peut utiliser un agent qui modélise le contexte de travail et qui contrôle l'accès et l'affichage de ses sous-agents PAC. Il existerait alors un tel agent par contexte.

[92] Pour connaître les agents PAC qui ne sont pas présents dans une procédure, il suffit de faire la différence entre la procédure en question et la procédure minimale. En effet cette dernière correspond au niveau maximal d'accès aux agents PAC. Elles contient donc toutes les opérations et tous les agents PAC disponibles.

[93] On peut également la faire apparaître sous une forme différente (par exemple en pointillé) pour montrer qu'elle existe mais qu'elle n'est pas accessible à ce poste. Ce choix est à discuter lors des spécifications.

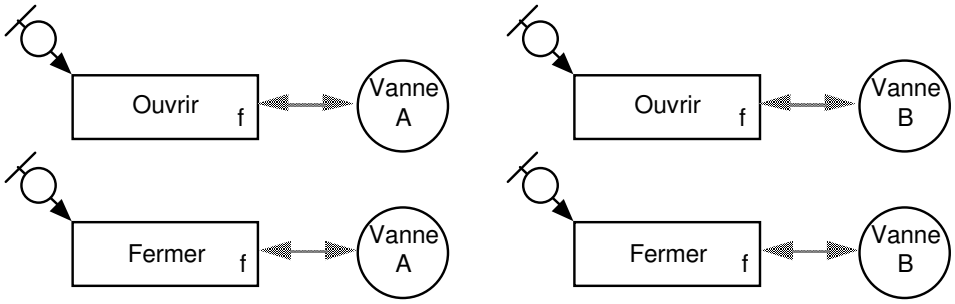


Figure 5.63.a : Procédure faisant appel aux opérations des deux vannes

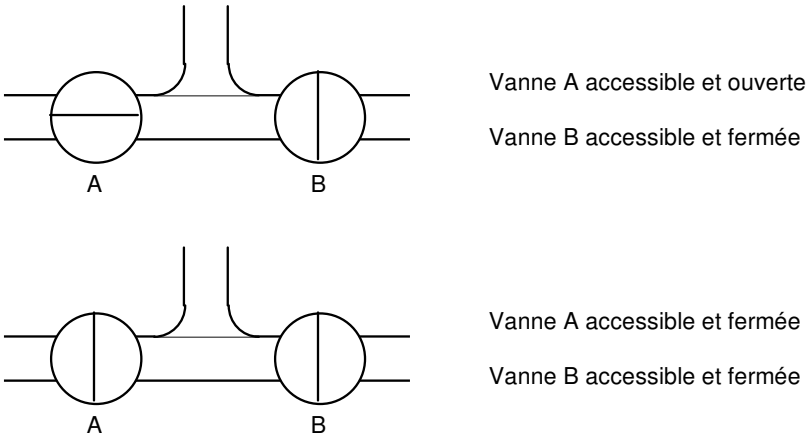


Figure 5.63.b : Résultat de la figure 5.63.a sur le panneau de contrôle

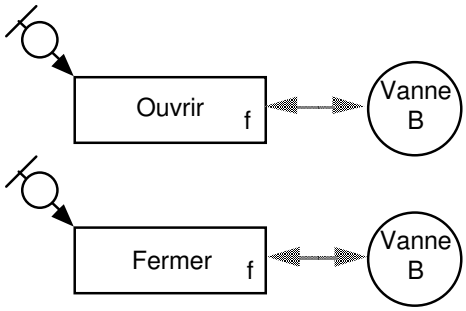


Figure 5.64.a : Procédure ne faisant appel qu'aux opérations de la vanne B

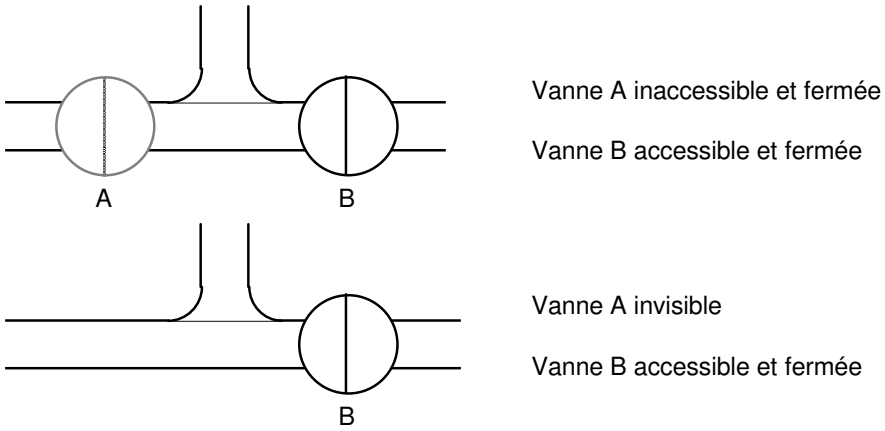


Figure 5.64.b : Résultats possibles de la figure 5.64.a sur le panneau de contrôle

C'est pourquoi nous utilisons le concept de logique d'utilisation des objets naturels en liaison avec celui d'auto-gestion de la représentation externe des agents PAC. Pour cela, nous associons chaque attribut présent dans un objet naturel[94] à un agent PAC, l'Abstraction représentant cet attribut[95] et la Présentation étant fonction de la nature de l'attribut. Le modèle résultant sera dorénavant appelé OPAC (Objet naturel PAC).

Avec OPAC, nous construisons des objets qui fournissent, en plus d'une gestion de leur représentation externe, un ensemble de primitives permettant de les manipuler (ces primitives seront toujours fournies par le Contrôle). Prenons un exemple (Figure 5.65) : une donnée de type liste (par exemple un annuaire) peut être représentée par une listbox (Présentation). L'Abstraction correspond à la liste des abonnés et le Contrôle permet de faire afficher la liste ou encore est capable de déterminer quel est l'élément sélectionné dans la listbox. Si une opération manipule cette liste, tous les échanges d'informations devront transiter par le Contrôle. Ainsi il devra être possible d'ajouter un élément à la liste (qui mettra automatiquement à jour son affichage), ou encore de supprimer l'élément sélectionné. Quant aux traitements à exécuter par les opérations, ils se réduisent à des traitements du type :

> Liste ajoute: élément — on suppose que élément est correct pour la liste. L'Abstraction reçoit par le Contrôle l'ordre d'ajouter élément à sa valeur actuelle, puis la Présentation reçoit l'ordre de se mettre à jour.
>
> Liste supprime: (Liste sélection)[96] — Liste sélection renvoie la position de l'élément sélectionné dans la listbox. Lorsque l'utilisateur a cliqué sur la listbox, l'Abstraction a pris connaissance de l'élément sélectionné par l'intermédiaire du Contrôle qui lui a fourni la position de l'élément sélectionné (sélection). Le Contrôle peut ainsi fournir différents résultats sur la sélection (position, libellé de la sélection, élément complet de la liste...)

[94] Nous supposons dorénavant que le concept d'objet naturel n'est pas aussi restrictif que dans sa version actuelle. Nous supposons par exemple qu'il est possible d'avoir des objets naturels avec une intersection non nulle.

[95] L'Abstraction contient des données dont elle gère elle-même la sémantique par ses propres méthodes.

[96] On aurait pu écrire également Liste supprime: (Liste selectedItem) où selectedItem renvoie la valeur sélectionnée.

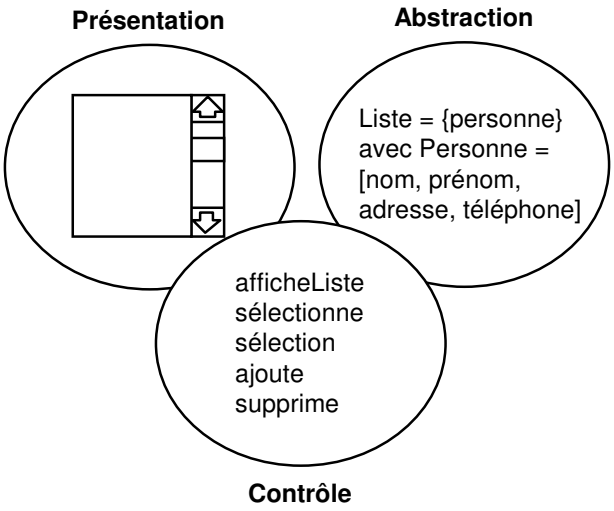


Figure 5.65 : Un objet PAC représentant une liste d'éléments.

On peut également utiliser des OPACs complexes. Par exemple la saisie d'un employé peut faire appel à la saisie de son nom, de son titre (Mme, Mlle, M), de son numéro de sécurité sociale, etc., qui sont autant d'éléments d'apparence et de comportement différents. Pour cela on utilisera un OPAC comme celui représenté sur la figure 5.66. L'OPAC Employé a connaissance de la Présentation, de l'Abstraction et du Contrôle de chacun de ses composants. Il est capable de gérer la totalité de la Présentation par l'intermédiaire des Présentations et des Contrôles de chacun de ses composants. Le Contrôle de Employé permet d'accéder à chaque Abstraction ce qui implique qu'une opération qui a besoin de manipuler l'adresse d'un employé aura recours au Contrôle de Employé et non pas au Contrôle de Adresse. L'OPAC Employé devra donc fournir des méthodes permettant la saisie, la modification, la recherche ou la suppression d'employés. Ces méthodes devront porter sur l'ensemble des attributs composant un employé. On préférera donc une méthode du type SaisieEmployé qui renvoie un objet composite plutôt que des méthodes de saisie élémentaires pour chaque attribut (SaisieNom, SaisieTitre, SaisieNuméroSécu, etc.). Par contre ces méthodes seront disponibles dans les Contrôles correspondant à chacun de ces attributs, ceci permettant éventuellement de changer des ordres de saisie (par exemple d'abord le titre puis le nom). Les méthodes fournies par les Contrôles devront respecter un standard de nom et de comportement pour une totale réutilisation. Par exemple les saisies correspondront au message getContents, la suppression à remove, etc.

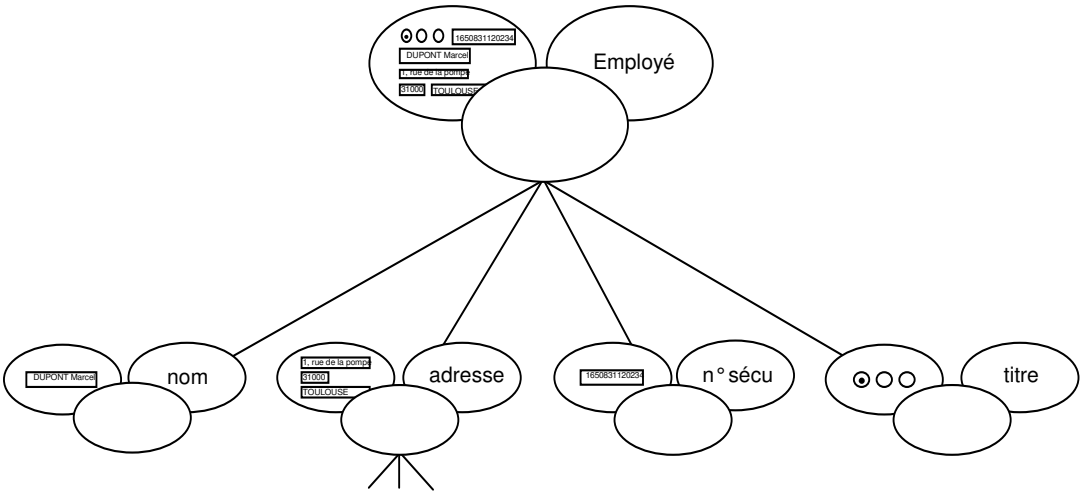


Figure 5.66 : Un OPAC complexe représentant un employé. L'adresse est elle-même un OPAC complexe qui contient la rue, le code postal et la ville.

Pour construire les procédures et les opérations Diane+, nous devrons disposer au départ d'un ensemble d'OPACs. Cet ensemble d'objets compose les deux couches de base de notre architecture six couches (cf § 5.2.3). Par ailleurs il n'est pas nécessaire d'avoir toujours accès à tous les composants d'un OPAC. Par exemple certaines opérations n'auront besoin que du nom de l'employé. C'est pourquoi nous définirons des vues externes sur ces OPACs en précisant pour chaque opération les Abstractions qui nous intéressent. Par ailleurs ces Abstractions pourront être visibles différemment, ceci étant géré directement par l'OPAC. Si l'on demande par exemple à un OPAC Employé d'entrer en mode de recherche (l'utilisateur veut trouver un employé particulier), l'OPAC Employé mettra ses composants correspondant aux clés d'index en lecture/écriture (ex: nom et numéro de sécurité sociale) et les autres en lecture seulement (affichage de l'adresse, du titre de la personne, etc.). Cette notion de vues externes sur les OPACs peut être rapprochée des "Perspectives" de V. Normand. Une Perspective permet de choisir les données manipulées ainsi que le type de manipulation qu'elles peuvent subir. Diane+ éclate cet aspect en trois points :

- le choix des données (c'est-à-dire les OPACs utilisés) est issu des procédures qui font apparaître les OPACs manipulés,
- la vue que l'on a sur les données (par exemple ne prendre que le nom d'un client) est représentée dans les vues externes des OPACs (choix des attributs de l'Abstraction),
- les manipulations possibles sur les données sont représentées par l'intermédiaire des connexions entre les opérations et les OPACs (par exemple une flèche qui va d'un OPAC à une opération signifie que l'OPAC est en lecture par rapport à l'opération).

Cette décomposition en trois parties est certes fastidieuse, mais elle expose clairement les trois points pré-cités alors que les Perspectives nécessitent une lecture approfondie pour arriver au même point.

Nous étudierons dans le chapitre suivant la connexion entre les opérations et les OPACs ainsi que les répercussions sur la génération de l'interface.

8. Conclusion

Diane+ permet de spécifier la répartition des tâches entre l'homme et la machine. Nous avons vu qu'il est possible de l'utiliser à la suite des méthodes (orienté-tâche ou orienté-objet) qui ne possèdent pas cette caractéristique. Le formalisme Diane+ permet d'exprimer de manière simple et puissante la latitude décisionnelle de l'utilisateur qui devient alors aisément vérifiable. Cependant

il est important de noter que toutes les caractéristiques de Diane+ que nous avons vues ne font pas intervenir directement les données. Les spécifications des opérations permettent en effet de valider le dialogue homme-machine uniquement sur l'éventail des possibilités d'enchaînements. Il est donc probable que des enchaînements soient réalisables par rapport aux traitements et impossibles au niveau des données. Le chapitre suivant montrera comment nous réalisons l'implémentation des concepts Diane+ en faisant le lien avec les OPACs.

Chapitre 6
Génération et Gestion Automatique à partir de Diane+

1. Introduction

Ce chapitre a pour objectif de montrer comment les concepts et les processus que nous avons présentés jusqu'ici peuvent être implémentés. Ces concepts sont principalement les tâches, les procédures, les opérations, les contraintes et l'aide, les processus étant la génération automatique et la gestion automatique. Cette implémentation est basée sur Diane+ et utilise fortement l'approche objet.
Le paragraphe 2 décrit les processus de génération et de gestion automatique à partir des spécifications Diane+. Il détaille également les objectifs et le fonctionnement général de notre outil. Une maquette est actuellement en cours d'implémentation. Sa représentation interne est exposée au § 3 dont la dernière partie effectue un bilan quant aux objectifs visés.

2. Génération et gestion automatique à partir des spécifications Diane+

Nous allons détailler dans ce paragraphe les processus de génération et de gestion automatique de l'interface, de la dynamique et de l'aide. Pour cela nous nous fixons comme objectif de spécifier un outil permettant de développer une application avec Diane+. Nous verrons également comment utiliser les règles d'enchaînement [TARBY 91a] [TARBY 93] et comment les objets, représentant les concepts Diane+, permettent une implémentation plus efficace.

L'outil que nous nous proposons de spécifier devra permettre :

- *la saisie de la spécification du dialogue homme-machine*. Ce dialogue est décrit par l'intermédiaire des procédures Diane+ telles que nous les avons présentées dans le chapitre précédent. L'outil doit permettre toutes les commandes élémentaires telles que sauvegarder, retrouver ou supprimer une spécification. La saisie de la spécification devra se faire de manière graphique et interactive, c'est-à-dire en utilisant une boîte à outils qui contiendra l'ensemble des éléments Diane+ (opérations, procédures, déclencheurs, contraintes, etc.). L'outil devra être capable de gérer lui-même cette représentation graphique de la spécification ; par exemple, si on supprime ou si on déplace une opération, tous les liens seront mis à jour aussi bien du point de vue externe (à l'écran) que du point de vue interne (liens entre les objets représentant les opérations).
  La présentation de cet outil pourrait s'inspirer de celle du Class Hierarchy Browser de Smalltalk (Figure 6.1). Dans ce cas nous aurions une zone réservée à l'affichage des procédures Diane+ (zone 1), cette zone ne pouvant accueillir qu'une procédure à la fois. Le choix de cette procédure se fera par l'intermédiaire de la zone 2. Un choix dans cette zone provoque l'affichage de la procédure dans la zone 1, ainsi que celui des noms des opérations (zone 3) et des données (zone 4) qu'elle utilise. Lorsqu'on choisit une opération dans la zone 3, la zone 4 ne contient alors que les données manipulées par cette opération et la zone 5 contient le code du traitement réalisé par cette opération. Il est possible à cet instant de connaître toutes les opérations de même nom, de même que toutes les opérations ou

procédures qui utilisent l'opération sélectionnée[97]. Ces possibilités sont un grand atout pour le déboggage.

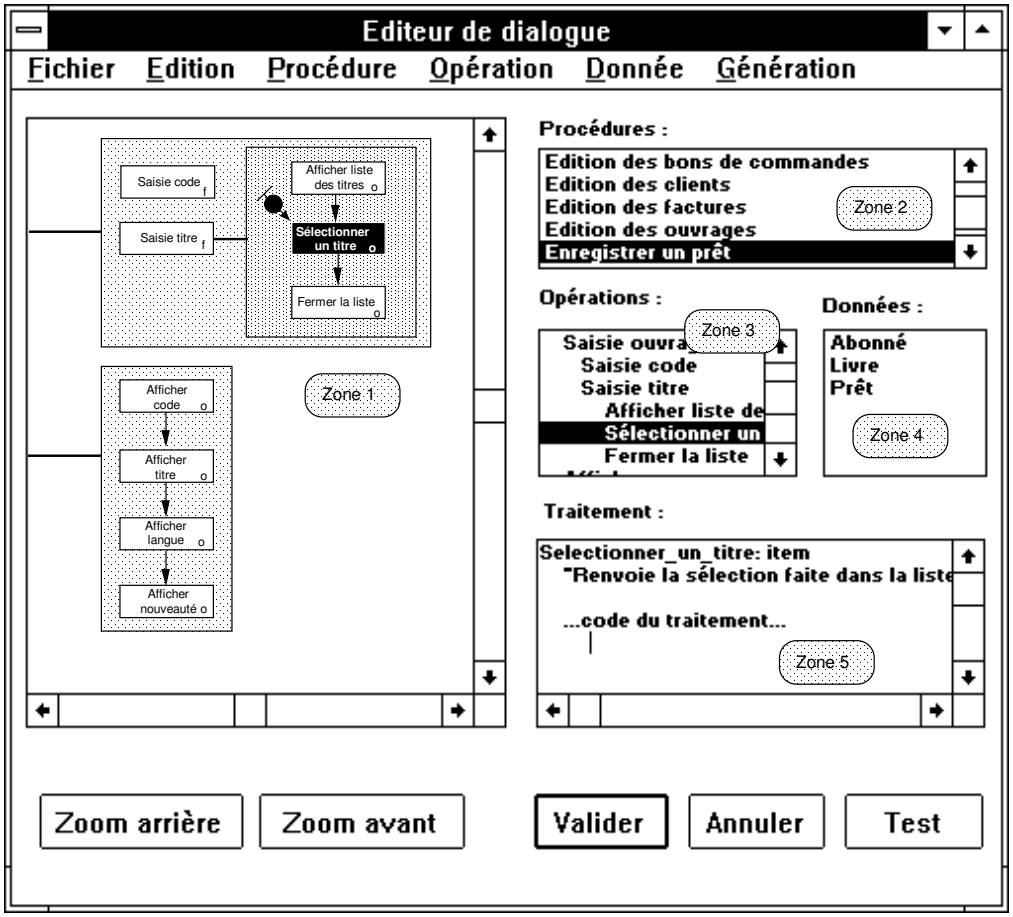


Figure 6.1 : Exemple d'éditeur pour la méthode Diane+

- *des tests* [98]*à tous les niveaux depuis la spécification jusqu'à la génération*. Ces tests permettent :
    - de vérifier la syntaxe des procédures. Ils sont capables de détecter par exemple qu'il y a conflit sur les attributs d'une opération. Ils peuvent également détecter des blocages éventuels par exemple à cause de contraintes illégales.
    - de contrôler l'interface générée. Après la phase de spécification, on peut déjà procéder à la génération d'un prototype de l'application. Il doit alors être possible de vérifier que les opérations décrites dans les procédures et les éléments de l'interface générée coïncident parfaitement. Ces tests porteront par exemple sur l'enchaînement des fenêtres, la gestion des menus, sur l'accessibilité réelle des opérations ou bien encore sur leur simultanéité.

- *la génération de l'interface*. L'objectif de cette génération n'est pas d'obtenir de manière automatique une interface parfaite, mais de fournir une interface de base qui soit capable de se gérer elle-même sur un plan purement externe et qui contienne tous les éléments nécessaires au bon fonctionnement de l'application. Cette interface respectera donc des critères ergonomiques généraux et il sera possible de récupérer son code afin la modifier par

[97] Ceci est respectivement similaire aux commandes "Senders of" et "Implementors of" de Smalltalk.
[98] Cette partie n'est pas implémentée pour l'instant.

l'intermédiaire d'un UIMS ou plus simplement d'un éditeur de ressources. On pourra par exemple disposer différemment des zones de saisie ou bien encore réorganiser les menus ou les commandes dans ces menus. Dire que l'interface se gère elle-même au niveau externe pur signifie qu'entre le moment où l'utilisateur provoque un stimulus sur cette interface et le moment où l'interface envoie l'événement correspondant à l'application, seule l'interface est intervenue. Par exemple si l'utilisateur choisit une commande dans un menu, l'interception du clic, l'ouverture du menu, le choix de la commande dans ce menu et l'envoi de la demande d'exécution de cette commande à l'application, sont entièrement à la charge de l'interface. De même, si une opération devient invalide suite à certaines conditions, le contrôleur de dialogue en avise l'interface qui se charge alors de rendre cette invalidité visible pour l'utilisateur. Ces différents aspects seront abordés plus en détail dans le § 2.3.1 et le § 3.5.2.

- *la génération de certains objets de l'application*. Puisque l'on a décrit les opérations et les procédures, il est possible de générer des objets qui vont les représenter. Par exemple à chaque opération va correspondre une instance de la classe Opération (idem pour une procédure). De même, la génération de l'interface va créer des objets utilisés dans cette interface (fenêtres, menus, boutons, etc.). La génération des objets représentant les données n'est pas abordée dans ce rapport puisque nous supposons que nous partons d'une couche d'objets OPAC qui représentent ces données. Nous approfondirons la génération des opérations dans le § 2.2.1.

- *la gestion automatique de la dynamique*. Des liens sont créés lors de la génération de l'interface et des objets de l'application. Ces liens établissent le contact par exemple entre une opération et sa représentation externe. Le contrôleur de dialogue, lors des sessions de travail, sait toujours si une opération est accessible ou non pour l'utilisateur. A chaque changement d'état de l'opération, le contrôleur met à jour sa représentation externe de manière automatique. Ainsi un bouton ou un menu deviendra grisé ou bien encore une listbox deviendra inaccessible. Les liens entre le contrôleur de dialogue, l'interface et le noyau fonctionnel seront détaillés dans le paragraphe 3.

- *la gestion automatique de l'aide*. L'utilisation de règles d'enchaînement dans le contrôleur de dialogue va nous permettre de gérer de manière automatique, et presque totale, l'aide dans l'application. L'avantage de l'utilisation de ces règles a déjà été présenté dans le Chapitre 2 (cf § 5.3.2) et dans le Chapitre 4 (cf § 2.3). Le paragraphe 3.5 précisera la représentation interne de ces règles ainsi que leur gestion pour l'aide.

2.1. Notion d'opérations de base

Une application, quelle qu'elle soit, doit respecter aujourd'hui un certain nombre de normes dans la représentation externe, dans son comportement, etc. Ces différentes normes peuvent être classées en deux grandes catégories : les normes communes à l'ensemble des applications (ou du moins celles d'un environnement particulier, Windows par exemple), et les normes à l'intérieur d'une même application. Les applications que nous générons doivent donc respecter ces deux types de normes. Nous allons tout d'abord détailler ces dernières et voir lesquelles sont effectivement réalisables, puis nous aborderons le thème des opérations de base, c'est-à-dire des opérations fournies automatiquement lors de la génération et dont l'utilité n'est plus à démontrer (OK, Annuler, Quitter, etc.).

2.1.1. Paramètres constants et paramètres variables

M-F Barthet décrit dans [BARTHET 88] la notion de paramètres constants et de paramètres variables. Ces paramètres s'appliquent au niveau conceptuel et au niveau externe des applications.

Les paramètres variables concernent des choix qui sont faits pour l'application et qui ne sont pas valables a priori pour d'autres applications. Au niveau externe ce sont par exemple :
- les couleurs,
- les tailles des fenêtres,
- la disposition des zones dans les fenêtres qui peut être voulue compatibles avec les formulaires existants,
- le vocabulaire employé dans les fenêtres.

Au niveau conceptuel ces paramètres correspondent au découpage en buts, procédures et opérations, qui dépend de l'application à développer.
Ce type de paramètres nous concernent peu puisqu'ils sont fixés par les concepteurs et ne subissent donc pas de règle générale. On peut cependant prévoir leur intégration partielle dans notre outil en proposant aux concepteurs des grilles de paramètres variables, les valeurs saisies dans ces grilles servant alors de référence lors de la génération de l'interface ou de l'aide. Ces grilles concerneraient surtout la représentation externe.

Les paramètres constants sont (ou devraient être), quant à eux, valables quelle que soit l'application. Au niveau conceptuel on trouve par exemple :
- les aides à l'apprentissage avec :
  - le guidage fonctionnel,
  - le guidage d'utilisation.
- l'aide au travail avec :
  - la suspension de traitement,
  - le transfert de données (par exemple le couper/coller),
  - le différé,
  - l'annulation (Undo),
  - la répétition (Redo).
- l'évolution du pilotage, c'est-à-dire pouvoir modifier la répartition des tâches entre l'homme et la machine[99].

Au niveau externe, les paramètres constants concernent par exemple :
- la structure des fenêtres (barre de menus, barre d'état, titre, mise en icône, etc.),
- la répartition des commandes et des menus (par exemple le menu Fichier et Edition sont toujours les deux premiers menus),
- le comportement de l'interface vis-à-vis de l'utilisateur (un bouton = un clic, une icône = un double clic, etc.),
- l'utilisation des couleurs (pas plus de cinq couleurs dans une fenêtre),
- l'utilisation des widgets [FOLEY 92] [JANSSEN 93].

Certains paramètres sont proposés directement par les environnements de travail. Par exemple Windows fournit les procédures nécessaires au couper/coller. Pour l'utiliser, le programmeur doit employer un des formats proposés par Windows ou bien encore créer son propre format. Le travail demande donc ici quelques efforts de la part du développeur. Par contre Windows crée des fenêtres qui ont toutes la même structure et qui se comportent toutes de la même façon. On est donc certain de conserver une parfaite homogénéité dans l'application et entre les applications.

[99] Cette modification de la répartition aura toujours pour effet de restreindre la latitude décisionnelle de l'utilisateur.

Deux types de paramètres constants sont plus difficiles à gérer. Ce sont :

- la répartition des fonctions proposées par l'application. Dans notre outil lors de la génération élémentaire de l'interface que nous verrons plus loin, les commandes sont réparties de la façon suivante :
  - toutes les opérations facultatives isolées sont groupées à part (à la fin de chaque menu ou sur une ligne séparée pour les boutons),
  - les autres opérations apparaissent par bloc et dans l'ordre donné par les précédences.

  Or si deux blocs distincts apparaissent dans une procédure, comment doit-on les classer ? Rien ne dit que l'un ou l'autre sera plus utilisé et donc on ne peut pas choisir leur ordre d'apparition par exemple dans les menus. Il reste cependant la solution arbitraire d'utiliser l'ordre alphabétique.

- la syntaxe des fonctions. Si le concepteur a écrit une fois Afficher les clients et une autre fois Affichage des clients, il est quasiment impossible de détecter cette erreur. Pourtant cela nuit fortement à l'homogénéité et à l'apprentissage.

Grâce à Diane+, deux paramètres constants ne posent pas de problème. Ce sont :

- *l'évolution du pilotage* puisque l'une des caractéristiques de Diane+ est de permettre des répartitions différentes en fonction des utilisateurs. Nous avons vu au Chapitre 5 que les procédures effectives permettent de répondre à ce besoin.
- *l'aide à l'apprentissage* puisque les procédures Diane+ fournissent directement l'aide d'utilisation (l'aide de fonctionnement étant implémentée directement pour chaque opération par le concepteur).

2.1.2. Opérations de base

La plupart des applications actuelles respectent des règles d'ergonomie générale que l'on peut classer dans les paramètres constants. Par exemple le fait de quitter une application sans avoir sauvegardé au préalable les documents ouverts ou encore supprimer un élément, provoquent toujours une confirmation. Notre outil propose des opérations de base qui produisent automatiquement ces confirmations ou ces vérifications sans que le concepteur ait besoin de les spécifier. Le concepteur manipulera donc des opérations du type Quitter l'application, Suspendre l'opération en cours, etc.
Pour qu'elles soient à même de se gérer individuellement, nous représentons ces opérations par l'intermédiaire d'objets comme nous le verrons plus loin. Ces opérations sont des instances d'une classe Opération qui fournit des demandes de confirmation, ou encore des messages d'alerte en fonction de la nature de l'opération (irréversible, dangereuse, etc.).

Nous allons détailler dans un premier temps les opérations de base Quitter l'application, Interrompre une opération, OK et Annuler. Trois autres opérations réservées plutôt à la manipulation de bases de données sont ensuite abordées. Ces opérations sont : Trouver, Précédent et Suivant (elles sont générées à la demande du concepteur).

Opération Quitter l'application

Lorsque l'utilisateur veut quitter l'application en cours, celle-ci doit vérifier que cela est possible. Dans la négative, l'application propose des "portes de sortie" à l'utilisateur. Certaines applications interdiront par exemple de quitter l'application tant qu'on n'a pas sauvegardé le travail en cours, d'autres proposeront de sauvegarder avant de sortir. Ce dernier cas est le plus fréquent et répond également à la norme CUA [IBM 89]. Nous le présentons à présent.

Quitter l'application est une opération obligatoire et à déclenchement utilisateur. A priori elle est toujours disponible, mais on peut envisager également de restreindre ses conditions d'activabilité, par exemple ne pas quitter tant qu'une connexion est en cours. Elle est également générée sans que le concepteur le demande. Dans le cas où il désire complexifier le processus de sortie, il a la possibilité soit de coupler l'opération générée avec un de ses processus (par exemple une opération ou une procédure) soit de la remplacer par son propre processus de sortie. Dans tous les cas, elle est générée au niveau le plus haut de l'application, c'est-à-dire au niveau des procédures. On peut effectivement considérer que Quitter l'application est un but en soi, avec pour procédure l'opération elle-même.

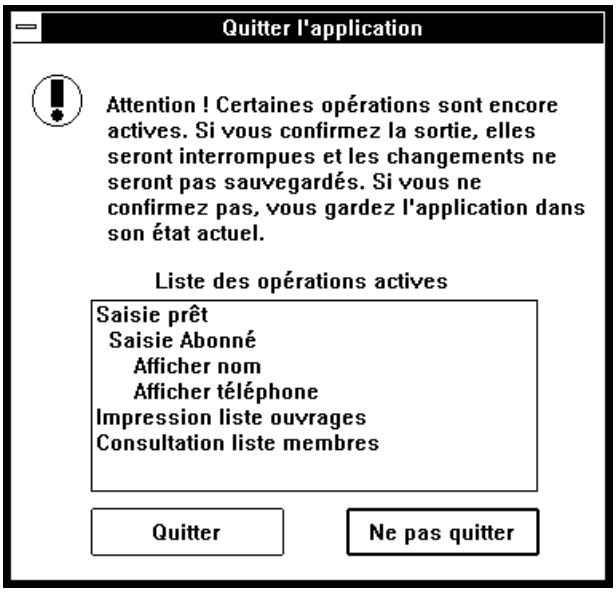


Figure 6.2 : Résultat du déclenchement de l'opération Quitter l'application

Lorsque l'utilisateur déclenche Quitter l'application, le contrôleur de dialogue inspecte alors l'application afin de chercher s'il existe des opérations en cours d'exécution. Dans l'affirmative, il affiche un message à l'utilisateur en lui stipulant qu'il ne peut pas quitter directement l'application. Ce message contient également la liste de toutes les opérations en cours[100] ; si l'utilisateur choisit, après ce message, de quitter l'application, toutes les opérations en cours sont interrompues et les changements intervenus depuis la dernière sauvegarde sont définitivement perdus (Figure 6.2).

Opération Interrompre

Dans notre système, chaque opération est supposée interruptible. Dans le cas contraire, le concepteur doit demander explicitement la non-interruptabilité en choisissant la valeur non interruptible pour l'opération en question.
Le déclenchement d'une opération interruptible provoque la création d'un processus correspondant au traitement demandé et possédant une certaine priorité. Ce processus est identifié entre autres par le nom de l'opération. Si l'utilisateur désire interrompre l'opération, le contrôleur de dialogue envoie alors au gestionnaire de processus l'ordre de détruire le processus correspondant. Par

[100] Cette inspection des opérations est facilement réalisée grâce à l'implémentation en objet. Nous verrons plus loin comment la gestion des opérations est traitée de manière récursive et individuelle par les opérations.

contre, le retour à l'état précédent de l'application n'est pas pris en compte de manière automatique.

Opération OK

Cette opération est parfois appelée Valider ou Enregistrer suivant les applications. Dans notre outil, l'opération générera toujours un bouton OK, le concepteur ayant la possibilité de changer le libellé du bouton par la suite.

Le fonctionnement général de cette opération est le suivant : l'opération est disponible à partir du moment où tout ce qui la précède est terminé. Cela signifie donc que l'opération OK dépend directement des précédences permanentes qui la lie avec les autres opérations obligatoires. Les liens avec les opérations facultatives ne nous intéressent pas directement, cependant ils sont pris en compte soit par l'intermédiaire des opérations obligatoires, soit par l'intermédiaire des contraintes sur ces opérations facultatives.

Remarque importante sur OK : l'opération OK ne doit effectuer aucun traitement. Elle sert uniquement à déclencher d'autres opérations (voir plus loin opération Trouver).

La figure 6.3.a présente une procédure qui demande que la saisie soit terminée pour qu'on puisse la valider. Par exemple la saisie ne sera terminée qu'après que l'on ait saisi un nom et un numéro de client, et la validation consistera à ajouter ce nouveau client à la base de données correspondante. L'opération Trouver qui permet de trouver un client par exemple à partir de son nom ou de son code est facultative. Lorsque l'utilisateur la déclenche, OK reste activable. Si le concepteur désire qu'OK ne soit activable qu'à partir du moment où il y a suffisamment d'informations saisies, il devra l'écrire explicitement dans l'opération Trouver. Des cas de ce genre seront abordés plus loin.

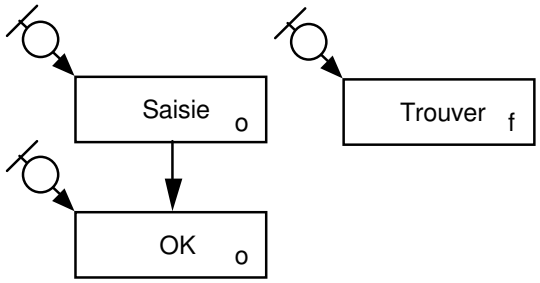


Figure 6.3.a : Un exemple d'utilisation de l'opération Valider

Remarque importante sur les opérations facultatives : déclencher Trouver provoque l'inactivabilité de OK tant que Trouver n'est pas terminée. Les liens avec les opérations facultatives sont donc gérés.

L'utilisation de OK montrée ici n'est pas d'un grand intérêt ici puisqu'elle correspond à une utilisation classique des opérations Diane+. La figure 6.3.b donne une utilisation plus intéressante de l'opération OK. Dans cette figure l'opération OK n'apparaît pas, mais lors de la génération de l'interface, le système place sur la fenêtre un bouton la représentant. Ce bouton OK ne deviendra activable que lorsque les deux opérations Saisie nom et Saisie code seront terminées. Si cela se produit et que pour une raison quelconque l'une des opérations est de nouveau déclenchée (par exemple l'utilisateur efface le nom qu'il a saisi), OK devient inactivable. Pour que cela soit possible, le bouton OK est associé à l'ensemble de ces trois opérations (ceci peut être une opération

mère ou une procédure). Or le Chapitre 5 a montré qu'un bloc n'est terminé qu'à partir du moment où toutes les opérations obligatoires (ainsi que les opérations facultatives avec leurs contraintes) sont terminées. Il suffit donc d'associer le bloc général au bouton OK.

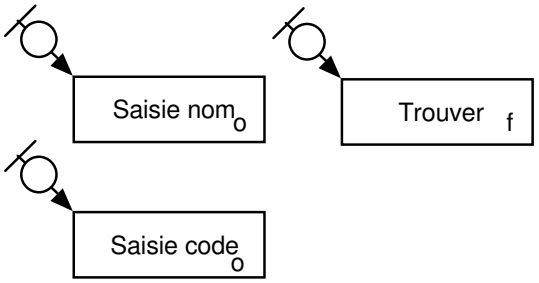


Figure 6.3.b : Cas normal d'utilisation de l'opération OK. Celle-ci est associée en fait à la procédure composée de ces trois opérations et n'apparaît donc pas dans la figure.

La figure 6.4 montre un cas plus habituel où la gestion de OK par le système est intéressante. Dans cette situation, OK dépend directement de Saisie code prêt, Afficher abonné et Saisie date retour prévu. Ces deux dernières opérations dépendent à leur tour d'autres opérations, par exemple Afficher abonné dépend de Saisie code abonné, mais également de Afficher téléphone qui dépend à son tour de Afficher nom, etc. Si le concepteur devait gérer toutes ces dépendances pour rendre activable OK, cela l'obligerait à écrire de nombreux tests pour savoir si telle et telle opération sont terminées. Or nous venons de dire que OK dépend en fait du bloc général et qu'il est activable lorsque toutes les opérations obligatoires du premier niveau sont terminées. Ceci implique que OK sera activable lorsque Saisie code prêt, Saisie code abonné, Afficher abonné, Saisie ouvrage, Afficher ouvrage, Afficher date du prêt et Saisie date retour prévu seront terminées. Ceci étant géré directement par le contrôleur de dialogue, le concepteur n'a rien à écrire pour la gestion de OK. De plus si l'utilisateur choisit Saisie titre dans Saisie ouvrage, cela implique que OK ne sera activable que si les trois sous-opérations de Saisie titre sont également achevées (en plus de celles précédemment énoncées). La gestion de OK est donc d'une grande simplicité.

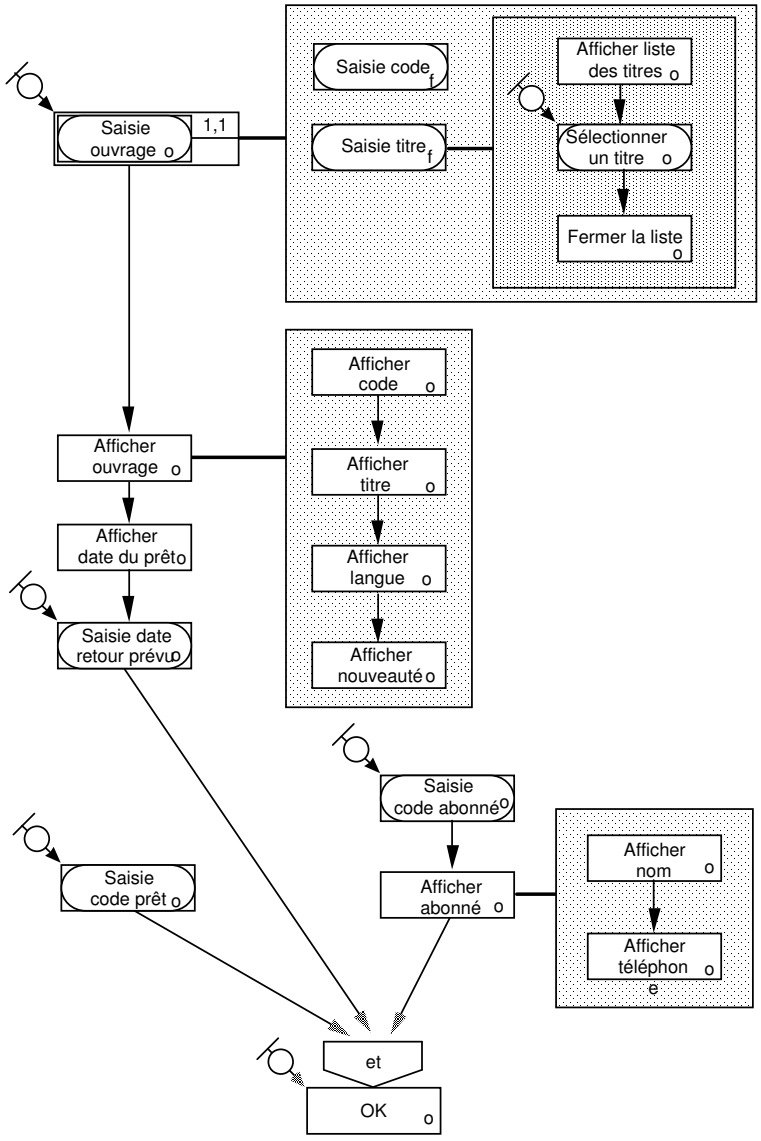


Figure 6.4 : Utilisation et gestion de OK dans le cas d'une opération complexe.
Les précédences grisées sont automatiquement gérées par le contrôleur de dialogue.

Opération Annuler

L'opération Annuler que nous présentons ici correspond au bouton Annuler que l'on trouve dans les applications pour Macintosh (on la trouve par exemple dans la fenêtre d'ouverture de fichiers) et dans certaines applications pour PC (par exemple dans Omnis). Cette fonction Annuler ne défait pas la dernière action, mais termine l'opération en cours sans enregistrer les changements survenus depuis la dernière validation. Elle se contente en fait la plupart du temps de fermer la fenêtre.

Annuler doit donc toujours être disponible dans la mesure où les traitements associés peuvent effectivement être annulés. Annuler correspond ainsi à une opération Diane+ facultative qui est automatiquement ajoutée au contenu d'une procédure ou d'une opération. Par défaut sa génération produit le code permettant de fermer la fenêtre à laquelle elle appartient. Si le concepteur veut

changer son comportement (par exemple en interrompant les traitements en cours), il doit, pour l'instant, l'écrire lui-même après la génération de code[101].

Si l'on reprend l'exemple de la figure 6.3.b, l'opération Annuler est donc ajoutée automatiquement par le système aux opérations du premier niveau spécifiées par le concepteur. Elle n'apparaît cependant pas dans le schéma (nous représentons cette présence virtuelle dans la figure par un grisé). Lors de la génération, le système transforme Annuler en un bouton du même nom et qui, par défaut, sera donc toujours activable.

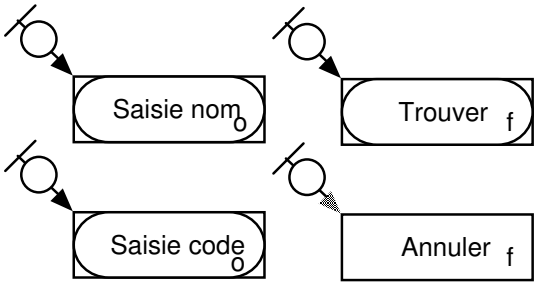


Figure 6.5 : Un exemple de procédure incorporant l'opération Annuler

Opération Trouver

Cette opération permet de rechercher un élément à partir des attributs qui servent de clé dans sa description. Cette opération s'applique donc principalement aux fichiers.

Si le concepteur utilise un OPAC de type fichier, celui-ci doit être capable de lui fournir la liste des clés qu'il utilise. L'opération Trouver peut alors être sollicitée par le concepteur afin de l'intégrer dans sa spécification. Le concepteur ne doit surtout pas détailler le fonctionnement de cette opération. Trouver doit être capable de mettre automatiquement en lecture/écriture les zones correspondant aux clés, et en lecture seulement les autres zones. La figure 6.6 donne un exemple de cette situation. Sur la figure 6.6.b, l'opération Trouver fait appel à l'OPAC Fichier des Clients (Figure 6.6.a) dont la clé est constituée de l'Abstraction Code abonné. Trouver demande à l'OPAC Fichier des clients le nom de ses clés. Le résultat est Code abonné que Trouver met alors en accès libre pour l'utilisateur alors qu'il rend inactivable les autres zones. Le code correspondant à Trouver serait donc :

Trouver :

- Saisie clés :
  Pré-actions :

  listeCle := Fichier_des_Clients getKeys "listeCle est une variable locale. Le résultat de cette ligne est l'attribut Code abonné"

  listeAttribut := Ficher_des_Clients getAttributs "listeAttribut est une variable locale"

[101] La solution que nous envisageons pour l'instant à cette question est d'associer chaque traitement automatique (par exemple une impression) à un processus particulier identifié par un nom et une priorité (Smalltalk gère directement ce genre de processus). Lors d'une session de travail sur une fenêtre, on peut imaginer stocker les processus qui s'y rapportent dans une liste. Ainsi lorsque l'utilisateur choisit d'annuler cette fenêtre, on envoie un message d'arrêt à chacun des processus de la liste. Tous les problèmes ne sont pas pour autant résolus puisqu'il peut arriver qu'un processus qui est ainsi interrompu ne remette pas le système dans un état correct.

listeCle do: [:cle | cle enable]. "Pour chaque élément de la liste des clés, rendre la zone correspondante activable"

(listeAttribut reject: listeCle) do: [:attribut | attribut disable]. "Pour chaque élément de la liste des attributs qui ne sont pas des clés, rendre la zone correspondante inactivable"

Action :

fin := (Fichier_des_Clients getContents) "L'OPAC récupère les valeurs saisies dans les zones correspondant aux clés"

Post-conditions :

fin "L'opération est terminée si fin est vrai"

- Rechercher :

Action :

fin := (Fichier_des_Clients find: (Fichier_des_Clients getContents))

Post-conditions :

fin

Lors de la génération, Trouver donnera naissance à un bouton du même nom[102]. Ce bouton permet de déclencher l'opération de recherche. Après un clic sur ce bouton, OK est activable de même que Saisie clés (l'utilisateur peut saisir les valeurs pour les clés, ceci étant supposé géré par l'OPAC). Dès que l'utilisateur clique sur OK, l'opération OK est terminée et le système déclenche automatiquement Rechercher à partir des valeurs saisies par l'utilisateur. La recherche est à la charge de l'OPAC qui retournera soit la valeur recherchée, soit un message d'erreur puis "faux".

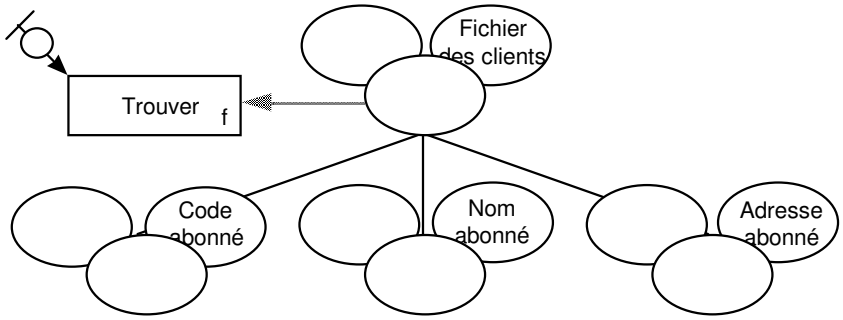


Figure 6.6.a : Représentation de l'opération Trouver avec l'OPAC Fichier des clients. La clé est l'Abstraction Code Abonné. Cette représentation est celle que voit le concepteur.

[102] Cette génération n'a pas besoin d'être demandée par le concepteur puisque Trouver est considérée comme une opération de base. En général une opération interactive à déclenchement utilisateur, et qui n'est pas de base, ne donne pas naissance à un bouton sauf si le concepteur le demande explicitement.

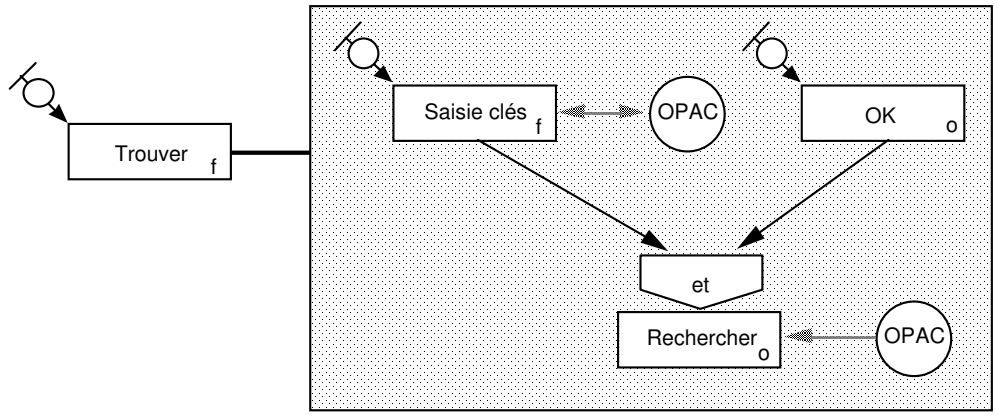


Figure 6.6.b : Structure de l'opération Trouver

Opérations Suivant et Précédent

Le fonctionnement de ces deux opérations est similaire à celui de Trouver. Le concepteur connecte ces opérations à un OPAC qui est capable de trouver et d'afficher l'élément qui suit ou qui précède celui déjà affiché.

Le code correspondant à ces opérations est le suivant :

- Suivant :
  Pré-conditions :
  OPAC getContents"s'il n'y a pas de valeur affichée, l'opération n'est pas activable"
  Action :
  fin := OPAC findNext: (OPAC getContents)
  Post-conditions :
  fin
- Précédent :
  Pré-conditions :
  OPAC getContents"s'il n'y a pas de valeur affichée, l'opération n'est pas activable"
  Action :
  fin := OPAC findPrevious: (OPAC getContents)
  Post-conditions :
  fin

La génération de ces opérations produit des boutons de même nom qui lancent automatiquement ces opérations lorsque l'utilisateur clique dessus.

Opérations de base et natures d'opérations

La nature des opérations permet également de générer des comportements de base. Ainsi une opération de nature "dangereuse" produit automatiquement l'ajout d'une demande de confirmation dans son code (par défaut au début du bloc Action, le concepteur pouvant par la suite la déplacer dans un autre bloc). De même une opération de nature "saisie" implique la génération d'une ligne de code qui renvoie le résultat attendu, par exemple la connexion d'un OPAC sur une opération saisie produit le code fin := OPAC getContents. En suivant un raisonnement similaire, l'opération Supprimer donne fin := OPAC_X remove: OPAC_Y (Figure 6.7).

Le type des OPACs peut également intervenir. Par exemple une opération Consultation qui utilise un OPAC de type fichier implique automatiquement :

- le placement des champs sur la fenêtre de consultation,
- la nature des zones affichées à l'écran (en entrée/sortie pour les index, en entrée seulement pour les autres champs),
- la gestion des différents widgets de la fenêtre de consultation (boutons OK, Annuler ...).

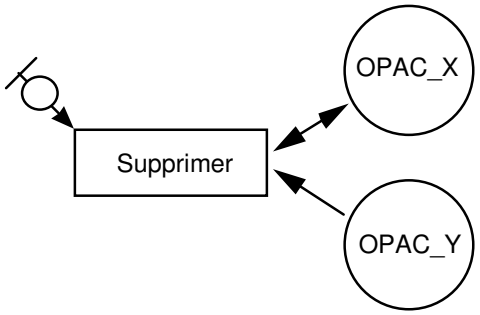


Figure 6.7 : Une opération de nature saisie avec ses OPACs

Notre réflexion n'a pas été poussée plus avant pour l'instant, mais ceci pourrait faire l'objet d'une étude dont les objectifs serait de trouver quelles sont les opérations de base et les natures d'opérations dont on a besoin (création, suppression, saisie, consultation, etc.), comment générer leur code en fonction des données qu'elles manipulent et comment les gérer ?

2.2. Génération automatique

Nous allons voir dans ce paragraphe quels sont les éléments de l'application que l'on peut générer, les moyens employés et les limites de cette génération. Nous allons aborder la génération des opérations, des fenêtres, des menus et des règles.

2.2.1. Génération des opérations

Lorsque nous avons présenté Diane+, nous nous sommes focalisés sur les traitements. Or il est bien évident que ces traitements manipulent des données. Le chapitre 5 avait pour objectif de présenter les aspects fonctionnels de Diane+ dans l'optique de la gestion du dialogue homme-machine (rappelons que Diane+ est une méthode basée sur les traitements et non pas sur les données). Il est cependant possible de faire figurer les données dans les schémas Diane+. Nous utiliserons une représentation similaire à celle de Merise/2 (cf Chapitre 5, § 5.1.2) et celle de OOM (cf Chapitre 5, § 5.2.2) pour faire le lien entre les données et les opérations (Figure 6.8).

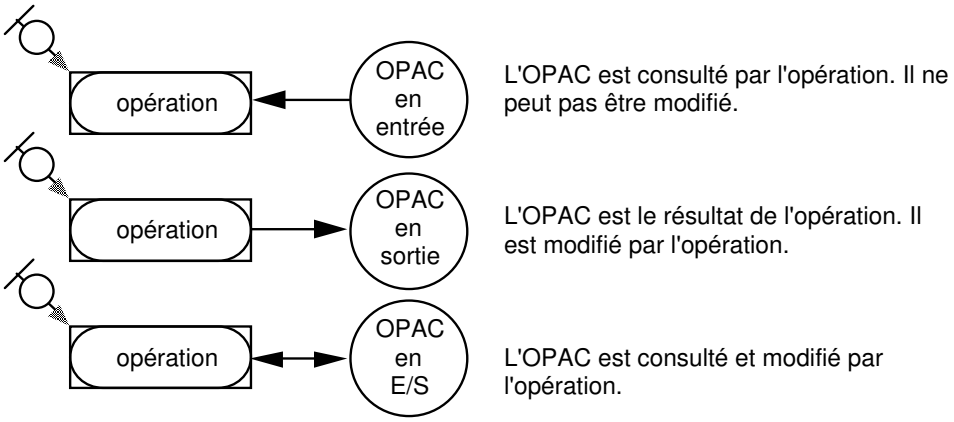


Figure 6.8 : Lien entre les opérations et les données.
Les flèches indiquent le type de manipulation par rapport à l'opération.

Si une opération contient des opérations filles, celles-ci "héritent" des OPACs[103] de leur mère avec au plus les mêmes droits d'accès (par exemple "entrée/sortie" permet "entrée" alors que le contraire est faux). Ainsi sur la figure 6.9, Saisie code, Saisie titre ainsi que ses sous-opérations, ont accès à l'OPAC Ouvrage de l'opération Saisie ouvrage. De même les opérations filles de Saisie titre ont accès à l'OPAC Fichier des ouvrages.

Une opération ne peut exécuter que trois types de traitements outre des calculs locaux :

- des actions sur les données (OPACs), par exemple ajouter des valeurs, consulter une valeur, etc.,
- des actions sur l'interface par l'intermédiaire des Présentations des OPACs, par exemple rendre activable une combo box[104],
- des appels de primitives du noyau fonctionnel.

[103] Nous prenons ici le terme OPAC dans un sens non restrictif. Ces OPACs pourraient également être des entités comme dans OOM ou bien encore des objets naturels. Nous supposerons cependant dans le reste de cette thèse que les données seront toujours des OPACs.

[104] Une combo box est une listbox qui se présente sous la forme suivante Langue: Français et qui permet de choisir une valeur parmi celle proposée dans la listbox par le système.

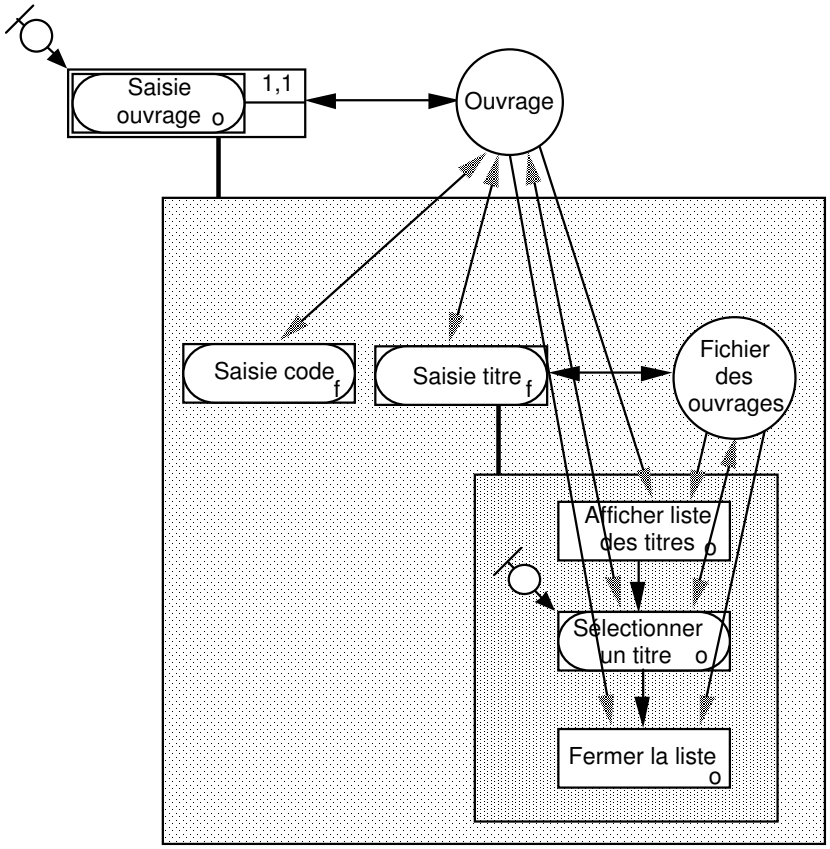


Figure 6.9 : Lien avec les données entre une opération mère et ses filles

Faire apparaître les données manipulées avec les opérations permet de générer les paramètres des opérations. La figure 6.10 fournira par exemple l'en-tête suivante :

Prêt
type : facultatif
mode : interactif
déclencheur : true
paramètres : In (Abonné), Out (Prêt), InOut (Ouvrage)

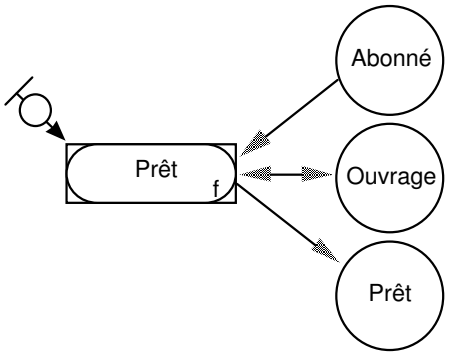


Figure 6.10 : Une opération avec ses paramètres

On remarque qu'il est possible de générer également la description de l'opération (mode d'interaction, type, déclencheur). Nous verrons plus loin comment sont représentées ces opérations sous forme d'objets.

Remarque importante : on ne doit faire apparaître dans les spécifications Diane+ que les opérations nécessaires à la gestion du dialogue. Toutes les opérations qui sont prises en compte par les OPACs ne doivent pas apparaître dans le détail à moins que cela soit important pour les enchaînements. Dans la figure 6.9 par exemple, si on suppose que l'OPAC Ouvrage gère lui-même l'opération Saisie titre, le bloc des opérations filles de cette opération disparaît. De même, si Ouvrage gérait lui-même l'opération Saisie ouvrage, toutes les opérations filles disparaîtraient. Dans cette même figure, on aurait pu faire apparaître les opérations Saisie code et Saisie titre comme des opérations obligatoires et enchaînées dans cet ordre. Ceci constituerait alors une restriction de la latitude décisionnelle, d'où la nécessité de l'expliciter complètement.

2.2.2. Génération des fenêtres

Cette génération fait intervenir à la fois les données et les traitements. En fait, nous envisageons deux types de générations : une génération élémentaire afin de valider les spécifications du dialogue homme-machine et une génération plus complète destinée à être réutilisée pour l'application finale.

Génération élémentaire

Ce premier type de génération intervient durant la spécification et la conception. Elle n'est utile qu'au concepteur et n'a pas pour objectif de servir de validation auprès de l'utilisateur final, mais plutôt de tester la cohérence entre les spécifications et l'interface générée.

Après avoir décrit les procédures Diane+, on peut souhaiter vérifier que l'application réagit tel qu'on l'a prévu. Il doit donc être possible de passer des procédures Diane+ à une simulation simple du comportement de l'interface. Pour obtenir cette génération, le concepteur a dû au préalable définir le but de l'application, les sous-buts et les procédures associées (Figure 6.11). A chaque but terminal (1.1, 1.2, 1.3 et 2) correspond au moins une procédure (minimale, prévue ou effective). Il est possible de représenter les buts et les sous-buts par l'intermédiaire du formalisme Diane+ comme cela est montré sur la figure 6.11 ou bien en utilisant d'autres formalismes (graphes et/ou, arbres, etc.).

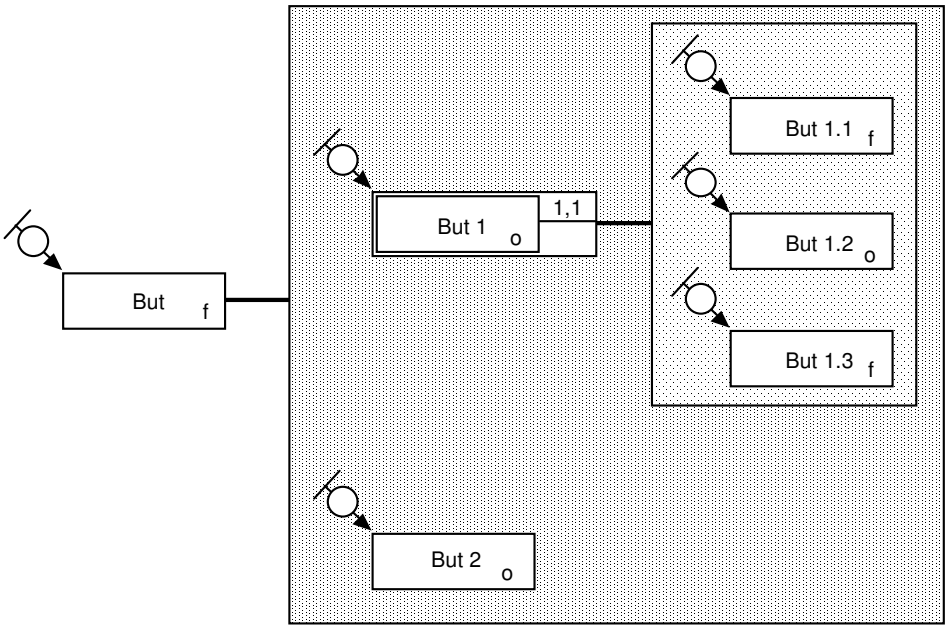


Figure 6.11 : Représentation des buts et des sous-buts avec Diane+

Les buts et les procédures (avec les opérations, mais sans les données) étant définis, on génère :

- une fenêtre correspondant au but principal. Cette fenêtre est la fenêtre principale de l'application. Toutes les autres fenêtres seront ouvertes à l'intérieur de celle-ci. Sa barre de menu est composée de menus qui portent le nom des buts de niveau 1 (buts $_i$) et les commandes de ces menus sont composées des niveaux de buts suivants (buts $_{i.j.k...}$). Les buts terminaux correspondent aux déclenchements de procédures.

- une fenêtre secondaire vide par procédure[105] et par opération. Cette fenêtre aura pour titre le nom du but pour une procédure ou de l'opération pour une opération ; dans le cas où une fenêtre est issue d'une opération, son affichage dépendra de la nature de celle-ci. Ainsi une opération sans déclencheur fera afficher automatiquement la fenêtre correspondante alors qu'une opération qui possède un déclencheur sera affichée après que la personne qui teste l'interface l'ait explicitement demandé. La figure 6.12.a représente la procédure Diane+ correspondant au sous-but 1.3 de la figure 6.11 et la figure 6.12.b représente les fenêtres correspondant à cette procédure dans l'application.

[105] Pour l'instant, nous nous proposons de générer par défaut une boîte de dialogue par procédure, mais le concepteur a la possibilité de demander une fenêtre véritable (bords variables, bouton de plein écran, etc.).

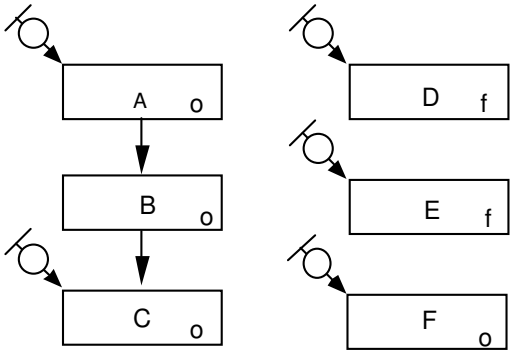


Figure 6.12.a : La procédure associée au sous-but 1.3

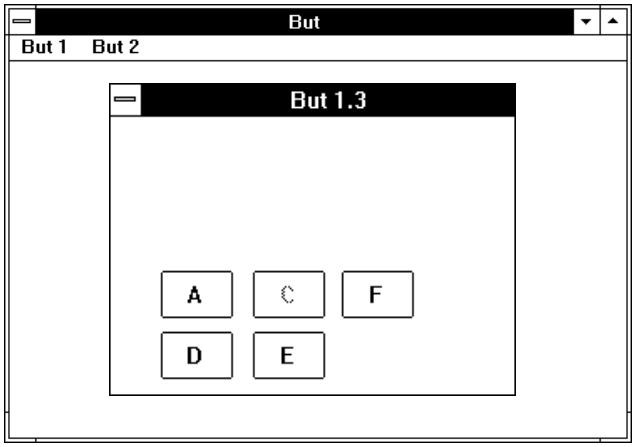


Figure 6.12.b : La fenêtre correspondant à la procédure associée au sous-but 1.3

On constate que la fenêtre a pour nom celui du but associé, et que toutes les opérations qui possèdent un déclencheur sont présentes sur cette fenêtre[106] sous forme de boutons dont la disposition peut être redéfinie par le concepteur. Les opérations obligatoires sont sur la ligne du haut et les opérations facultatives **isolées** sont sur la ligne du bas. La fenêtre principale contient comme prévu des menus correspondants aux buts de niveau 1 ; ces menus seront examinés plus loin (cf § 2.2.3).
En se basant sur la procédure Diane+, le contrôleur de dialogue rend inaccessible l'opération "C" puisque celle-ci ne peut être exécutée qu'après "A". On vérifie donc déjà à ce niveau la cohérence avec la procédure Diane+. Puis lorsque le concepteur clique sur l'un des boutons accessibles, le système ouvre une nouvelle fenêtre par dessus celle(s) existante(s). Ainsi lorsque le concepteur clique sur le bouton A, on obtient l'écran suivant (Figure 6.13) :

[106] Les opérations qui ne possèdent pas de déclencheur étant activées automatiquement par le système, il n'est donc pas nécessaire de les faire figurer sur la fenêtre (nous verrons plus loin comment elles sont prises en compte).

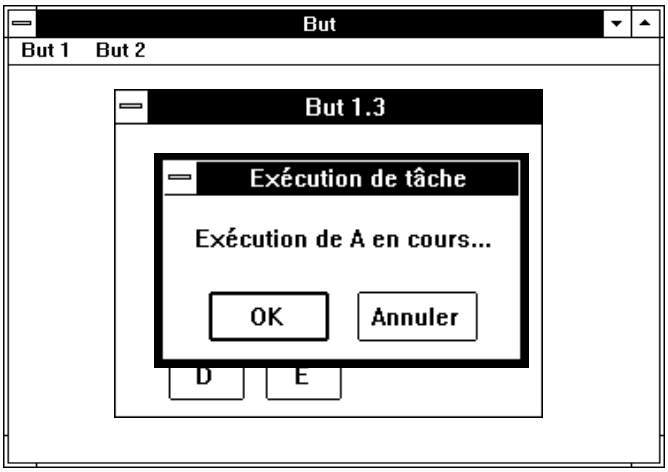


Figure 6.13 : Simulation d'exécution d'une opération interactive

Étant donné que cette opération "A" est interactive, la fenêtre affichée contient un bouton OK qui permet au concepteur de dire à partir de quel instant elle est considérée comme terminée. Le bouton Annuler permet de déclarer l'opération comme non exécutée afin de vérifier que l'interface revient bien dans l'état initial. Si l'opération "A" avait été automatique, la boîte de dialogue n'aurait pas eu de bouton OK, mais elle aurait conservé son bouton Annuler. Les opérations automatiques possèdent par ailleurs un temps d'affichage fixé à l'avance et destiné à simuler leur exécution. A la fin de ce temps, la fenêtre se ferme et le système continue sa progression en suivant la logique de la procédure. Dans notre cas, dès que le concepteur ferme la fenêtre "A" avec OK, la fenêtre correspondant à "B" s'affiche (Figure 6.14). A la fermeture de celle-ci, le bouton "C" devient accessible sur la fenêtre de départ (But 1.3). Le concepteur a alors le choix entre déclencher "C" ou "F" (qu'il pouvait également déclencher au début). De par leurs caractéristiques similaires à celles de "A", leur déclenchement provoquera donc une fenêtre semblable à celle de "A". Notons que tout au long de ces étapes, le concepteur avait le loisir de cliquer sur la fenêtre But 1.3 afin de déclencher les opérations "D" et "E". Chacune d'elles aurait eu pour conséquence l'affichage d'une fenêtre similaire à celle de "A".

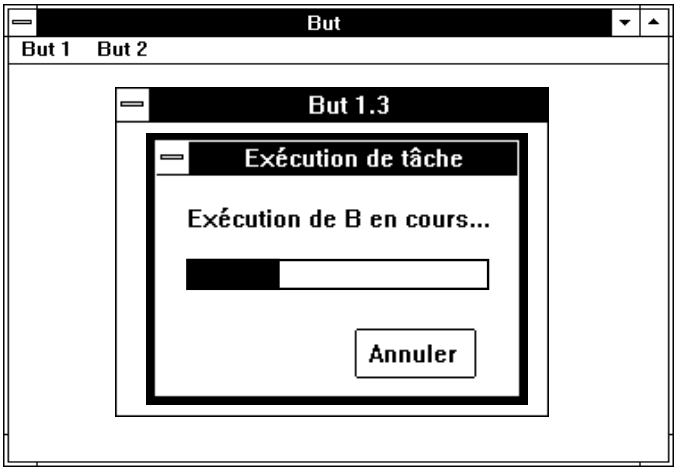


Figure 6.14 : Simulation d'exécution d'une opération automatique

S'il existe des contraintes sur les opérations (par exemple le fait qu'une opération α ne soit déclenchable qu'une seule fois), le contrôleur invalide alors automatiquement le bouton correspondant après le premier déclenchement de α.

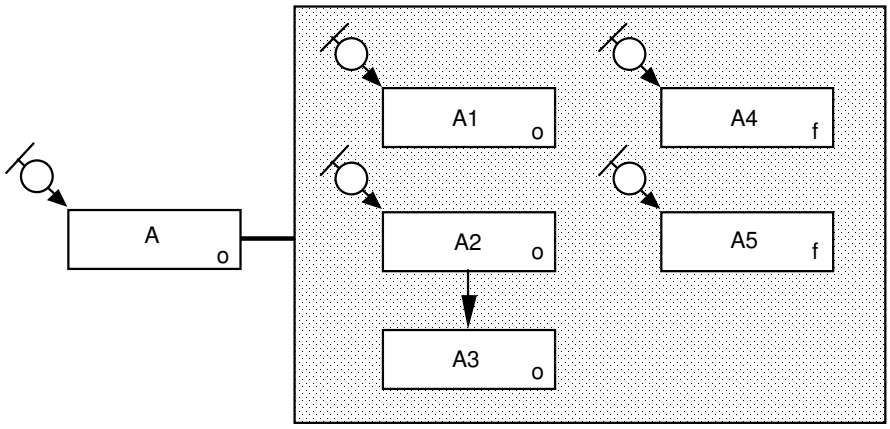


Figure 6.15.a : Représentation Diane+ de l'opération "A" de la figure 6.12.a et permettant la génération de la fenêtre de la figure 6.15.b

Si une opération contient des sous-opérations, le principe reste identique à celui des procédures. Si l'on suppose que l'opération "A" précédente se présente comme cela est montré sur la figure 6.15.a, alors la fenêtre générée à son déclenchement est celle présentée sur la figure 6.15.b. On constate donc que la génération est récursive et que son principe est invariable.

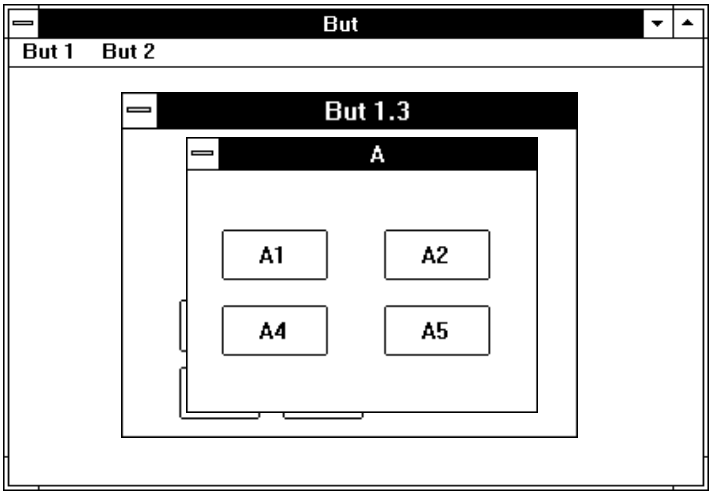


Figure 6.15.b : Fenêtre générée pour une opération interactive avec des sous-opérations

Génération finale

Le second type de génération intègre à la fois les opérations Diane+ et les données de l'application. Alors que le premier type de génération peut intervenir même si les opérations ne sont pas encore complètement définies (par exemple on ne leur a pas encore associé de traitement à exécuter), la génération que nous allons voir à présent ne peut se faire qu'en phase finale de conception. Son objectif principal est de permettre une validation auprès de l'utilisateur final.

Après avoir décrit les procédures, le concepteur doit associer à chaque opération les données qu'elle manipule. En effet, bien que le concepteur connaisse à l'avance les relations entre données et traitements, la répartition du dialogue avec Diane+ ne nécessite pas l'intervention des données. Il est donc possible de concevoir et de tester cette répartition sans manipuler les données. Par contre en phase finale de conception, il est primordial de faire le lien entre les opérations et les données. Pour cela, le concepteur associe à chaque procédure des OPACs en entrée et des OPACs en sortie. Une fois cette association terminée, chaque opération à l'intérieur de la procédure a connaissance de ces objets. Il est également possible d'ajouter localement pour certaines opérations des objets en entrée et en sortie (cf § 2.2.1).

Les règles de génération que nous utilisons sont les suivantes :

- une opération interactive (avec ou sans déclenchement utilisateur) qui fait appel à un OPAC implique que tous les attributs de l'OPAC utilisés dans l'opération sont placés en colonne sur la fenêtre. Ces attributs sont choisis au moyen des vues externes sur les OPACs (par défaut tous les attributs). L'opération "A" de la figure 6.16 fournira par exemple tous les attributs de l'OPAC Employé alors que l'opération "B" ne fournira que le nom et le numéro de sécurité sociale. L'aspect externe des attributs est laissé à la charge de la Présentation des OPACs.

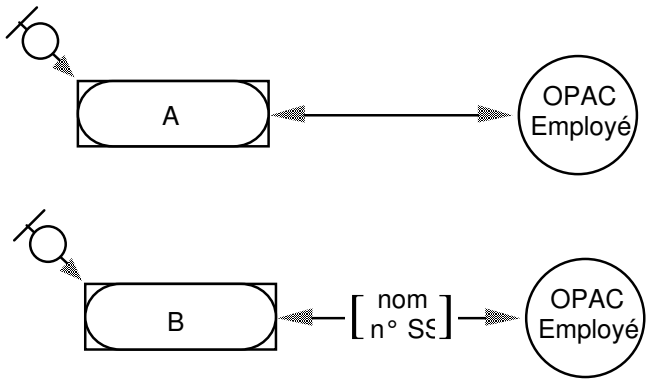


Figure 6.16 : Lien entre opération et données avec ou sans vue externe

- une opération automatique à déclenchement utilisateur qui fait appel à un OPAC implique sa représentation par un bouton dans la fenêtre.

Dans le cas où le concepteur n'est pas d'accord avec ces choix par défaut, il a la possibilité de choisir lui-même la représentation externe des OPACs et des opérations. Par exemple une opération interactive pourra être représentée par un bouton (Figures 6.17.a et 6.17.b).

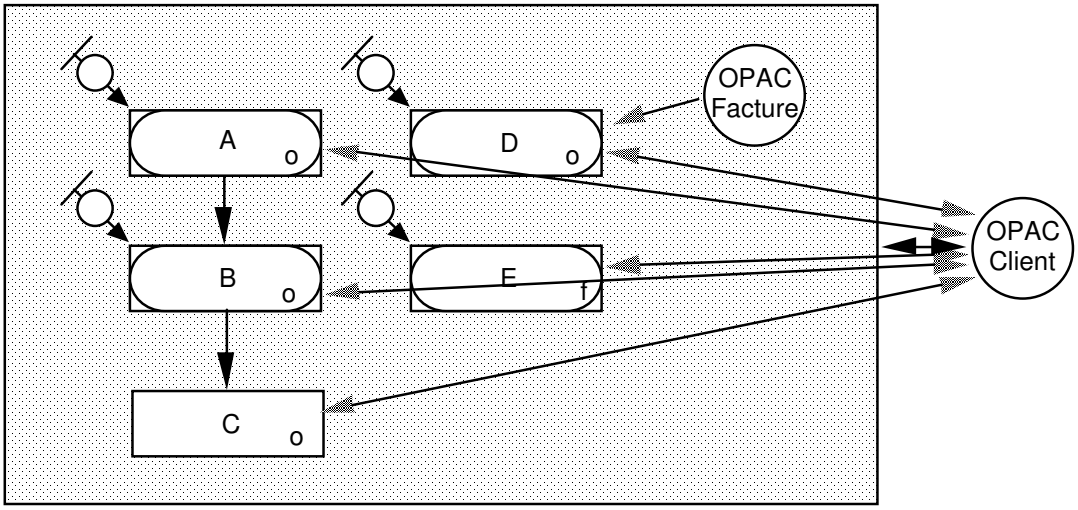


Figure 6.17.a : Détail d'une procédure ayant pour paramètre l'OPAC Client et permettant la génération de la fenêtre présentée dans la figure 6.17.b

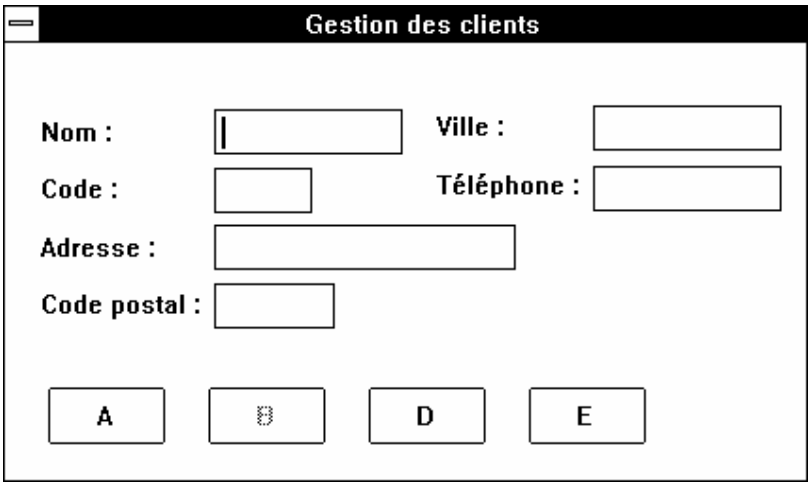


Figure 6.17.b : Fenêtre générée à partir d'opérations et de données

Cette génération terminée, le concepteur a la possibilité de modifier la fenêtre en supprimant les attributs dont éventuellement il n'a pas besoin ou en réorganisant les autres. En aucun cas il ne peut en ajouter, ceci afin de respecter la cohérence entre les spécifications et la réalisation[107] (Figure 6.17.c).

[107] Il reste possible d'ajouter ou de modifier des widgets afin d'utiliser plus efficacement une fenêtre, par exemple transformer des listes en cases à cocher.

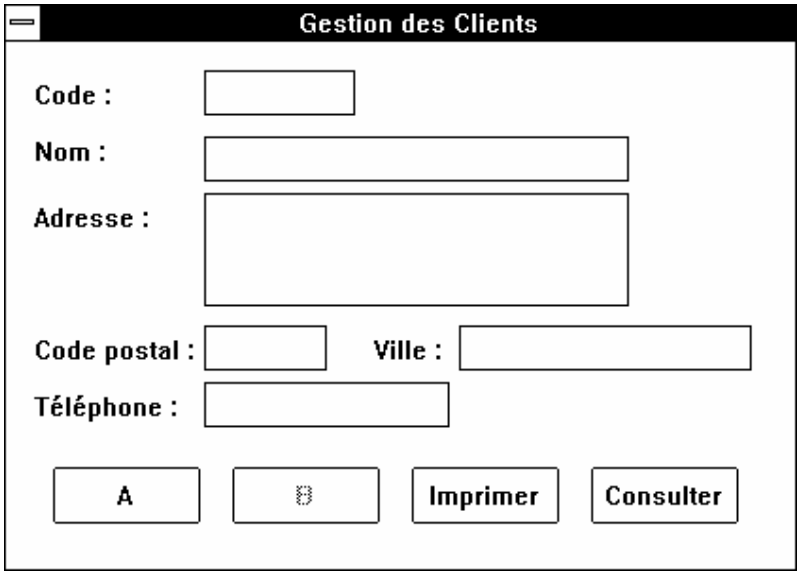


Figure 6.17.c : Exemple de réorganisation d'une fenêtre générée

Étant donné l'association entre une opération, la procédure qui la contient et les données qui s'y rapportent, l'opération récupère automatiquement comme paramètres les données de la procédure (ou de l'opération) qui la contient. Ainsi dans la figure 6.17.a, si l'on déclenche "A", "A" aura pour paramètre le client affiché à cet instant dans la fenêtre principale. De même "D" reprendra ces paramètres en plus de ceux qui lui sont propres, c'est-à-dire les attributs de Facture. La figure 6.18.b donne un autre exemple de fenêtre générée puis réorganisée. Elle est issue de la procédure de prêt vu précédemment sur laquelle on a greffé les OPACs correspondants (Figure 6.18.a).

Nous avons dit précédemment que la génération fait intervenir des OPACs. La figure 6.19.a donne un exemple de procédure qui manipule l'OPAC Liste des Employés (qui manipule lui-même l'OPAC Employé vu précédemment) et la figure 6.19.b montre la fenêtre générée. Dans cette fenêtre, les boutons Valider[108] et Annuler ont été placés automatiquement par la procédure (leur fonctionnement a été présenté au § 2.1.2). Les opérations Créer, Supprimer et Trouver ont été transformées en boutons suite à la demande du concepteur. Les méthodes associées font appel directement aux méthodes fournies par les composants Contrôles des OPACs.

[108] Valider correspond au bouton OK dont seul le libellé a été changé.

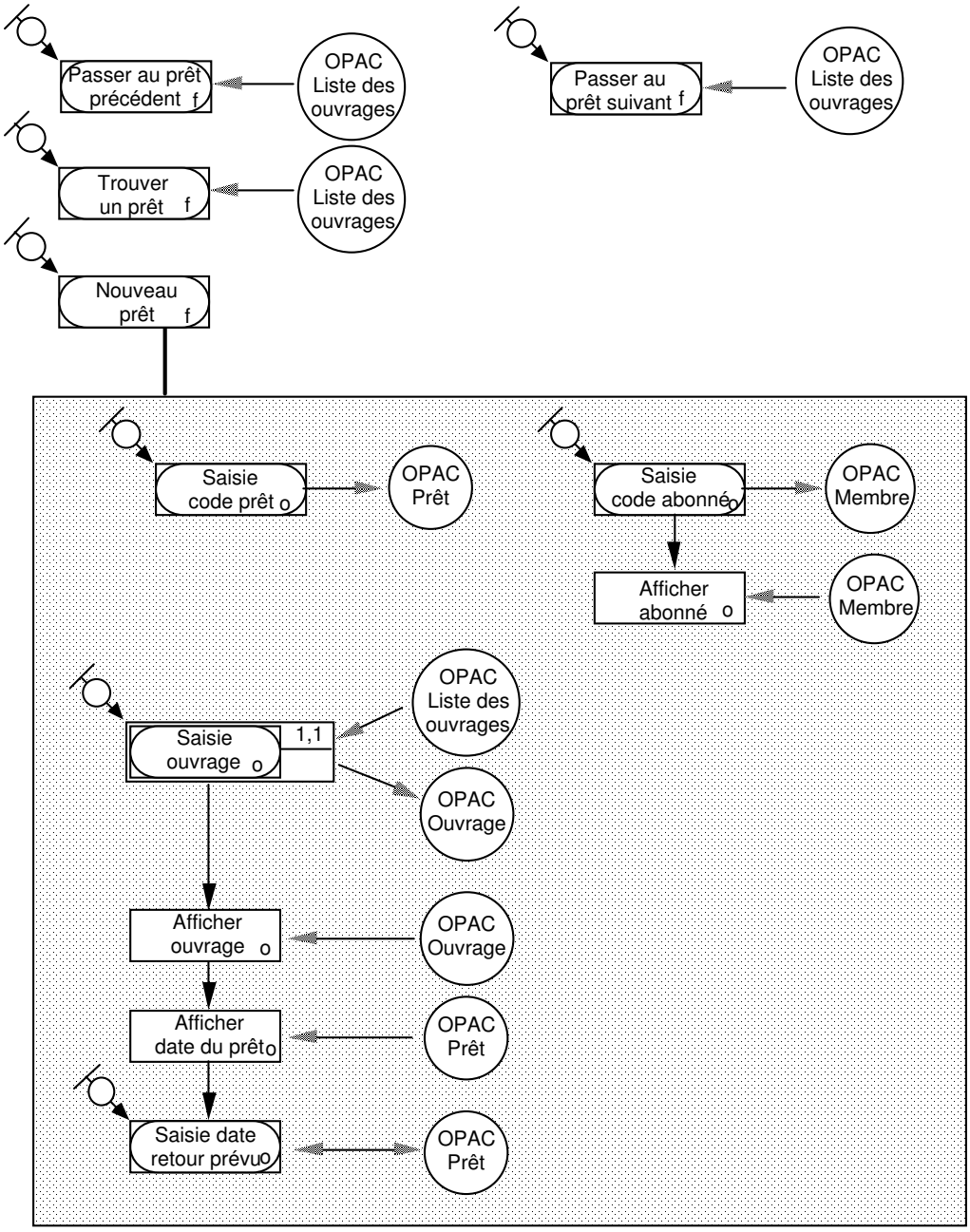


Figure 6.18.a : La procédure de Prêt avec les OPACs associés.
Pour des raisons de clarté, nous n'avons fait figurer que les niveaux de détails importants.

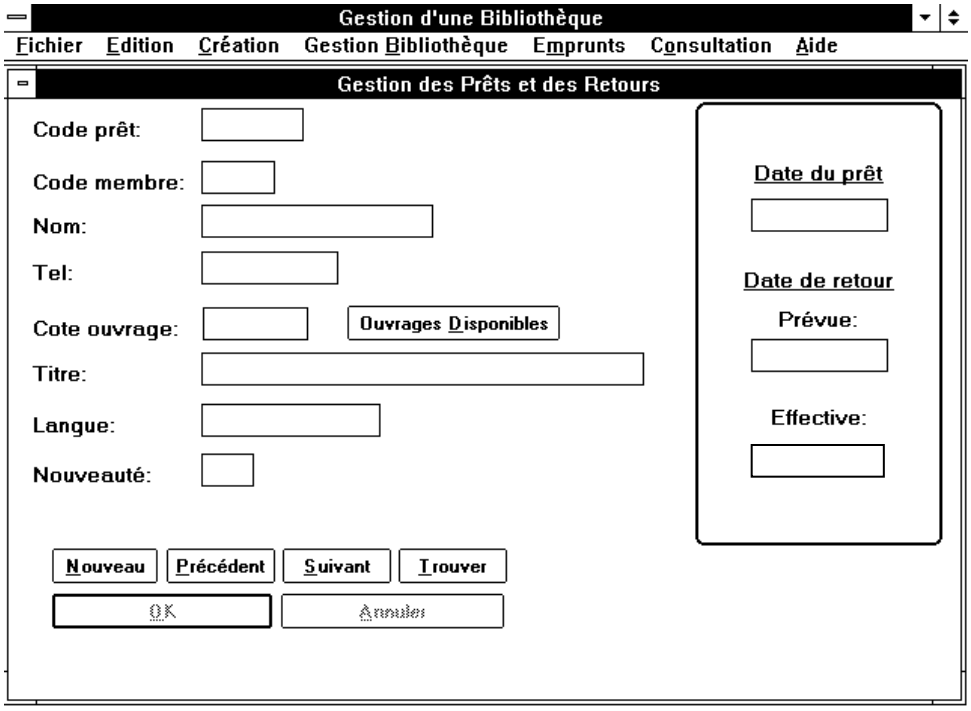


Figure 6.18.b : La fenêtre des prêts après génération et mise en forme. Le bouton “Ouvrages disponibles” correspond à la sous-opération “Saisie titre” de l’opération “Saisie ouvrage”

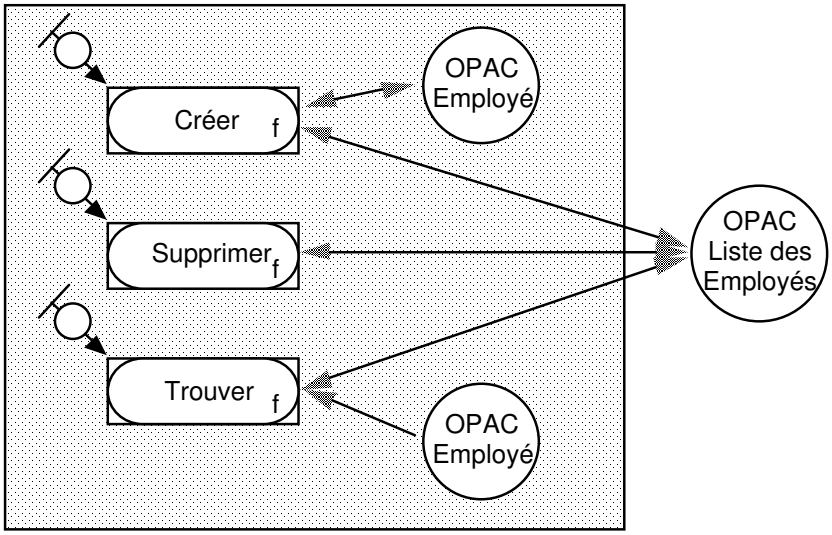


Figure 6.19.a : Exemple de procédure minimale de gestion des Employés utilisée pour la génération de la fenêtre de la figure 6.19.b

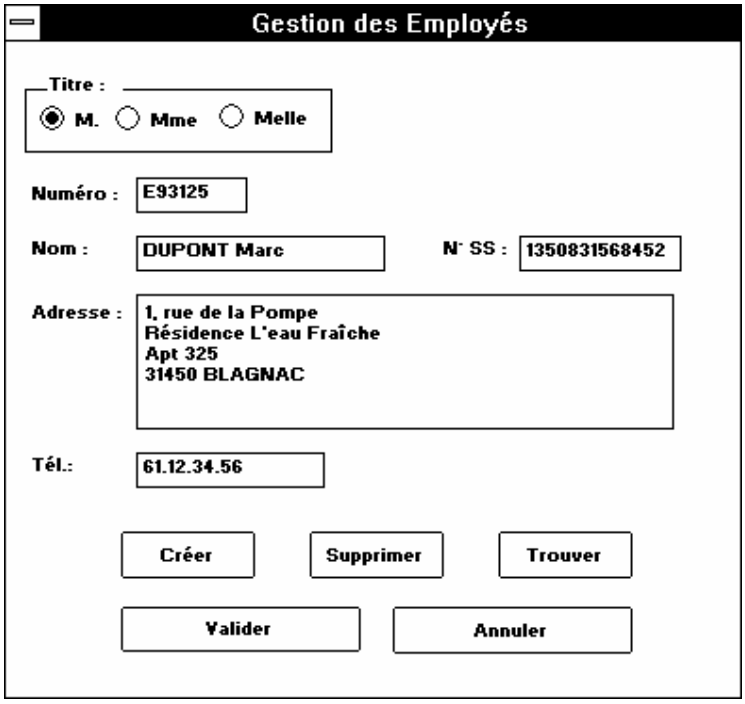


Figure 6.19.b : Fenêtre générée à partir la procédure minimale de la figure précédente

Le concepteur peut associer par exemple les traitements suivants aux opérations :

- Créer
  Liste_des_Employés add: (Employé getContents)
- Supprimer
  Liste_des_Employés remove: (Employé getContents)
- Trouver
  Liste_des_Employés find: (Employé getContents)

On suppose, dans ce cas, que l'OPAC Liste des Employés est capable de comprendre et de gérer des messages tels que add: (Employé getContents)[109] ou find: (Employé getContents). Pour cela, l'OPAC Employé gère lui-même les différentes zones qui le représentent à l'écran (nom, numéro, adresse, etc.). Il permet par exemple à l'utilisateur de saisir les rubriques dans un ordre quelconque ou bien encore il contrôle la validité syntaxique du numéro de sécurité sociale. L'OPAC Liste des Employés, quant à lui, est capable de trouver (find:) l'OPAC Employé correspondant à la valeur retournée par le message getContents. S'il n'existe aucune occurrence à partir des valeurs entrées, l'OPAC affiche un message d'erreur.
Ce cas est un cas extrême où l'OPAC est très puissant. La plupart du temps nous disposerons uniquement d'OPACs plus simple, c'est-à-dire qui fournissent des traitements plus élémentaires. La figure 6.20 représente ce que devient la procédure de la figure 6.19.a dans un cas semblable[110]. Sur cette figure on précise les enchaînements que l'on attend de la part de l'application à partir des OPACs. Ainsi l'opération d'ajout stipule que l'on doit obligatoirement saisir un employé (c'est-à-

[109] Le message getContents renvoie un OPAC Employé qui contient les valeurs des zones de saisie (nom, numéro, etc.). Dans le cas où les valeurs saisies ne sont pas correctes syntaxiquement, getContents renvoie un message d'erreur, puis la valeur "faux" une fois que le message a été lu (c'est-à-dire après la fermeture de la boîte de dialogue du message d'erreur).

[110] Dans un souci de clarté, nous avons fait apparaître les OPACs en face des opérations et non pas à côté de la procédure ce qui impliquait un grand nombre de flèches vers les opérations, surchargeant ainsi le dessin. Les OPACs Liste des Employés et Employé sont donc locaux à la procédure et non pas aux opérations.

dire remplir les zones correspondantes) avant de pouvoir activer l'opération d'ajout. On obtiendrait alors les opérations suivantes :

- Créer :
  - Saisie :

    Action : fin := Employé getContents "Employé récupère les valeurs des zones de saisie. S'il n'y a pas eu de valeur saisie, fin prend la valeur 'faux'"

    Post-conditions : fin "La post-condition permet de vérifier que l'opération s'est terminée correctement"
  - Création :

    Action : fin := (Liste_des_Employés add: Employé) "Si l'ajout ne se passe pas correctement, l'OPAC renvoie un message d'erreur puis 'faux'"

    Post-conditions : fin "la valeur de fin (vrai ou faux) indique que l'opération est terminée ou non"
- Supprimer :
  - Saisie :

    Action : fin := Employé getContents

    Post-conditions : fin
  - Suppression :

    Action : fin := (Liste_des_Employés remove: Employé)

    Post-conditions : fin

- Trouver :
  - Saisie :

    Action : fin := Employé getContents

    Post-conditions : fin
  - Recherche :

    Action : fin := (Liste_des_Employés find: Employé)

    Post-conditions : fin

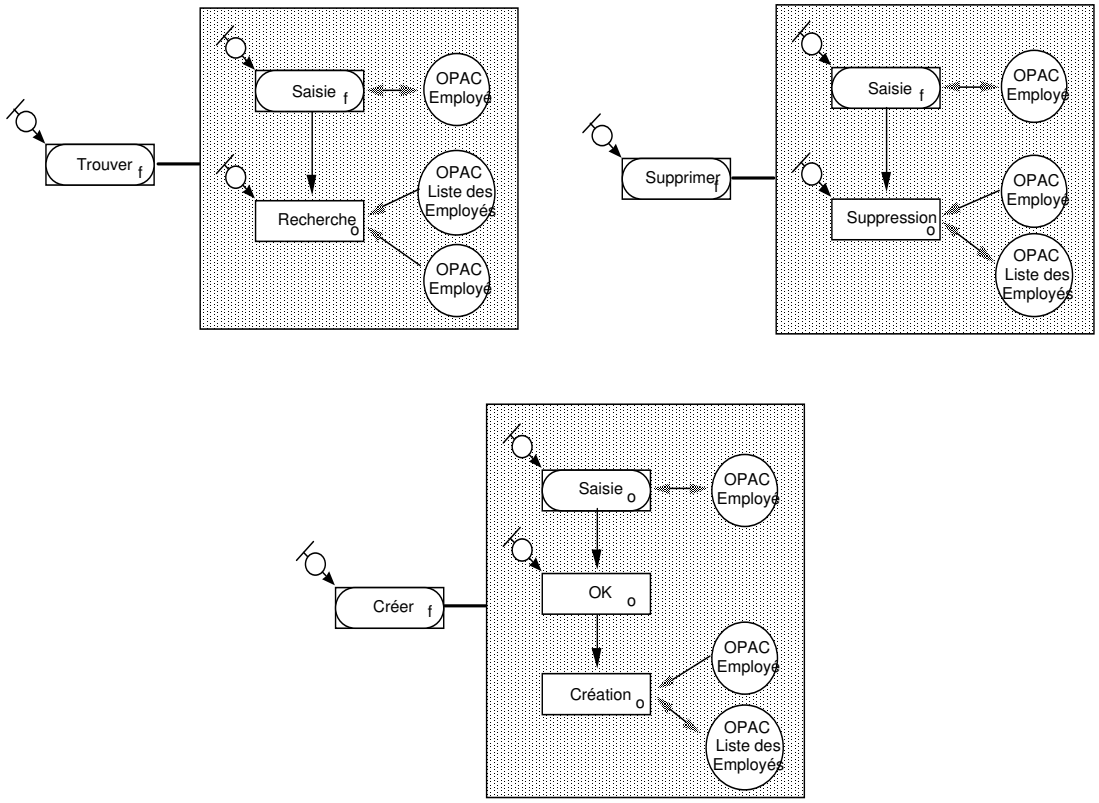


Figure 6.20 : Détail de la procédure de la figure 6.19.a dans le cas où les OPACs ne fournissent que des traitements élémentaires

Grâce à la spécification de la procédure, l'opération Création ne sera disponible que lorsque la saisie sera terminée. On voit ici l'importance des OPACs puisque ce sont eux qui vont dire à quel moment cette opération sera effectivement terminée. Par exemple l'OPAC peut décider que la saisie n'est terminée qu'à partir du moment où il y a un nom et un numéro d'employé. Le contrôleur de dialogue se contente de scruter régulièrement l'opération Saisie pour déterminer si elle est terminée ou non. Lorsque c'est le cas, le contrôleur rend alors activable l'opération OK à condition que ses pré-conditions (si elles existent) soient vérifiées. Cette activabilité implique alors que le bouton OK devient activable pour l'utilisateur. Un clic de sa part sur ce bouton déclenche l'exécution automatique de l'opération Création qui provoque l'ajout à l'OPAC Liste des Employés de l'OPAC Employé précédemment saisi. On retrouve un fonctionnement identique pour les deux autres opérations, mis à part pour l'opération Saisie qui est facultative, ce qui signifie que les boutons Suppression et Recherche sont toujours disponibles. Si le concepteur désire que l'entrée en mode saisie interdise Supprimer et Trouver, il doit explicitement le faire apparaître dans l'opération par l'intermédiaire des pré- et des post-actions, par exemple :

- Créer :
  - Saisie :
    Pré-actions : Supprimer disable; Trouver disable
    "ceci implique que Suppression et Recherche deviennent inaccessibles pour l'utilisateur[111] (mise en grisé géré automatiquement par l'intermédiaire de ces opérations et de leurs représentants externes)"
    Action : fin := Employé getContents
    Post-conditions : fin
    Post-actions : Supprimer enable; Trouver enable

[111] En fait seule l'opération Recherche devient inactivable étant donné son type obligatoire.

- Valider :
  Pré-actions :
  Action : fin := (Liste_des_Employés add: Employé)
  Post-conditions : fin
  Post-actions :

Nous donnons ci-dessous un autre exemple de fenêtre générée à partir d'un OPAC. Celui-ci gère une liste de dates. L'OPAC est capable d'ajouter, de supprimer et de tester une date. Chaque date ajoutée est automatiquement classée suivant un ordre chronologique. La procédure minimale associée est représentée sur la figure 6.21.a. Elle indique que l'ajout d'une date exige sa saisie au préalable et que la suppression d'une date exige sa sélection dans la liste. L'OPAC Liste de dates est supposé gérer lui-même l'OPAC Date, c'est-à-dire qu'une sélection dans la liste (OPAC Liste) provoque l'affichage de la sélection dans la zone de saisie des dates (OPAC Date). Si cela n'était pas le cas, il suffirait de détailler l'opération Sélection en la divisant en deux opérations (Sélection dans la liste et Affichage de la sélection). Inversement l'OPAC Date est en relation avec l'OPAC Liste, c'est-à-dire qu'un clic dans la zone de saisie rend la liste inaccessible et que la sortie du curseur de la zone de saisie rend la liste accessible.

Les traitements saisis par le concepteur seront les suivants :

- Ajout :
  - Saisie :
    Action : fin := Date getContents
    Post-conditions : fin
  - Ajouter :
    Action : fin := (Liste_des_dates add: Date)
    Post-conditions : fin
- Suppression :
  - Sélection :
    Action : fin := (Liste_des_dates selection)
    Post-conditions : fin
  - Supprimer :
    Action : fin := (Liste_des_Dates remove: Date) "Date a automatiquement été mis à jour par l'OPAC Liste des Dates"
    Post-conditions : fin

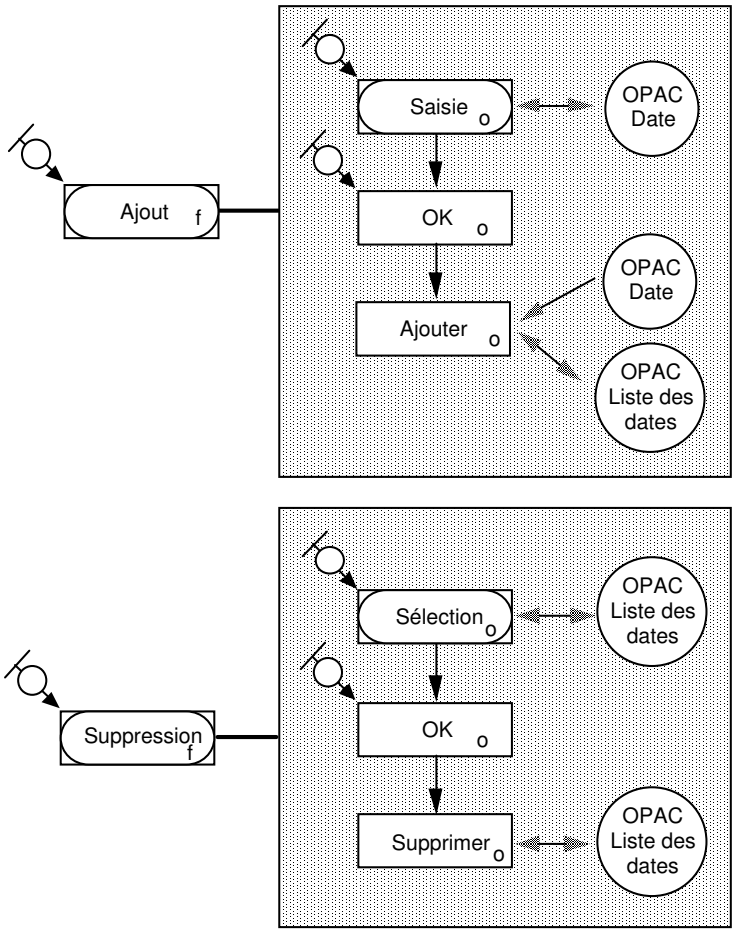


Figure 6.21.a : Procédure minimale pour la gestion de l'OPAC Liste de dates

Le bouton OK est activable lorsque :
- l'opération Ajout est active et que Saisie est terminée,
- l'opération Suppression est active et que Sélection est terminée,
- la fenêtre s'ouvre puisque les deux opérations mères sont facultatives.

Si le concepteur souhaite invalider le bouton Supprimer lorsqu'on clique sur la zone de saisie (la liste est automatiquement grisée par l'OPAC Liste qui en reçoit l'ordre par l'OPAC Date), il doit alors modifier l'opération Saisie comme suit :

- Saisie :

  Pré-actions : Suppression disable[112]

  Action : fin := Date getContents

  Post-conditions : fin

  Post-actions : Suppression enable

[112] On aurait pu également écrire Supprimer disable ce qui aurait eu pour effet de griser uniquement le bouton, mais en désactivant l'opération mère on est sûr de rendre inaccessible toutes les opérations obligatoires qu'elle contient. La robustesse est donc plus forte.

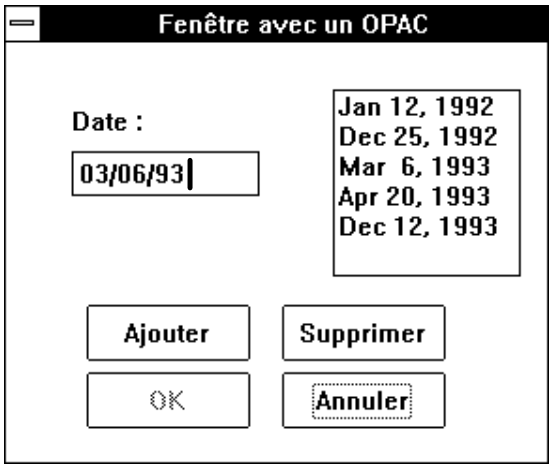


Figure 6.21.b : Fenêtre générée à partir de la procédure minimale de la figure 6.21.a

2.2.3. Génération des menus

La génération de menu n'intervient par défaut que pour les buts. Le principe que nous allons énoncer ici est également valable pour les procédures, ceci permettant d'obtenir des commandes de menu au lieu de boutons (il est tout à fait possible cependant de générer à la fois des menus et des boutons).

Nous avons vu que les procédures sont les feuilles terminales de l'arbre des buts d'une application (cf § 2.2.2). Toutes les couches Buts, Sous-buts, Sous-sous-buts, etc., vont ainsi être utilisées pour la génération des menus qui suit les règles suivantes (Figure 6.22) :

- dans le premier niveau de buts (c'est-à-dire les sous-buts puisque le but principal est considéré comme le niveau 0), chaque but donne naissance à un menu. Dans la figure 6.22, le menu But 4 est grisé car il dépend du But 3 qui doit être terminé au préalable.

- à l'intérieur d'un sous-but (quel que soit son niveau supérieur à 1), chaque paquet de buts donne naissance à une subdivision dans le menu (par exemple "ABC", "DE" et "FGH" dans la figure 6.23), et tous les buts facultatifs isolés sont regroupés à la fin du menu. Par ailleurs, si un but contient des sous-buts (But A dans la figure 6.23), la commande associée dans le menu donne naissance à un sous-menu. Ce processus pouvant se répéter autant de fois que l'on veut, il est intéressant de faire intervenir ici les règles d'ergonomie générale et de fixer à trois le nombre d'imbrications des sous-menus.

- le dernier niveau de buts (qui peut être différent suivant les menus) est directement connecté à l'activation de la procédure correspondante.

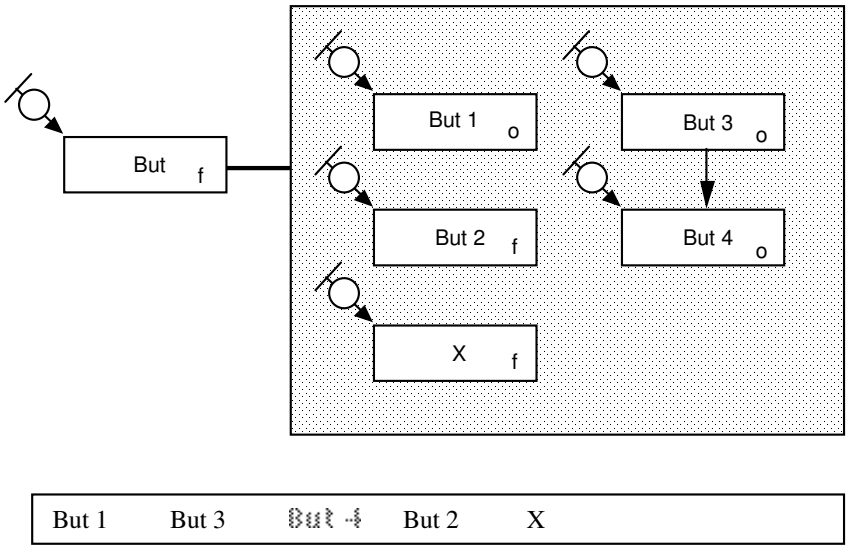


Figure 6.22 : Passage de but principal avec ses sous-buts à la barre des menus.

Les menus ainsi générés sont réutilisés pour la version finale de l'application où le concepteur peut, de même que pour les fenêtres, modifier à loisir leur apparence en regroupant des commandes, en les classant différemment, ou bien encore en renommant des commandes.

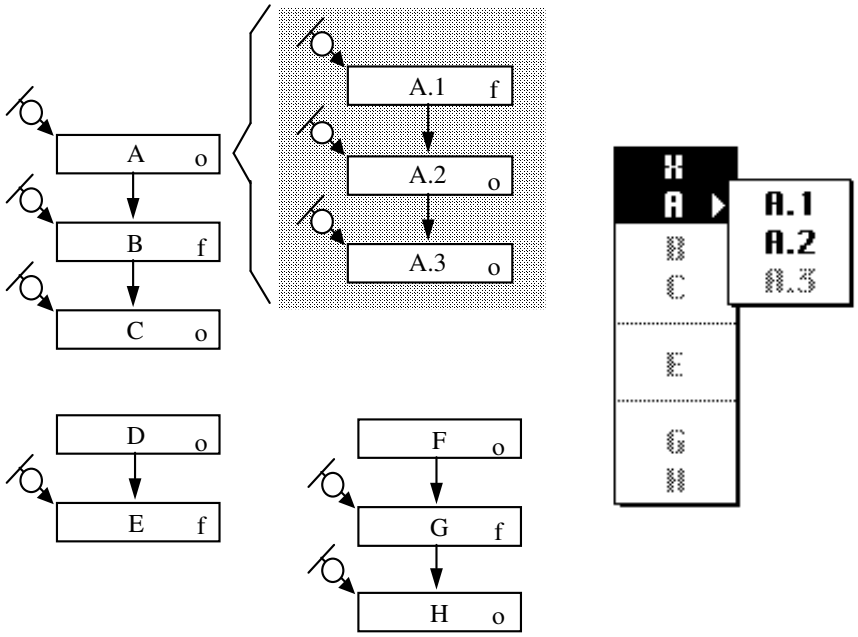


Figure 6.23 : Exemple de passage d'une description Diane+ à un menu.
Cette figure représente la décomposition du sous-but X de la figure 6.22.

La gestion des menus est assurée par le contrôleur de dialogue de la même manière que pour les fenêtres ; un but qui devient inaccessible provoque donc la mise en grisé de la commande associée dans le menu (exemple : "B" et "C" dans la figure 6.23).

2.2.4. Discussion

La génération automatique que nous employons ici correspond à la définition que nous en avons donnée au chapitre 3, c'est-à-dire une création et non pas une traduction. Cette création s'applique

principalement lorsque nous générons les fenêtres et les menus à partir des spécifications Diane+ (la génération des opérations est quant à elle une traduction du formalisme Diane+ en langage objet). Ce type de génération a déjà été appliqué dans des travaux tels que ceux de V. Normand [NORMAND 92a] et de J. Foley [FOLEY 91].
V. Normand utilise ainsi les "Workspaces" pour créer les fenêtres, les "Perspectives" pour les sous-fenêtres, les "Concepts" pour les zones de saisie et d'affichage, et les opérations de ces différents intervenants pour les menus. V. Normand utilise donc un principe voisin du nôtre puisqu'elle part de la spécification des tâches pour la présentation générale de l'application, et de celles des données pour le détail de cette présentation. Par ailleurs grâce à la structure en arbre de tâches, la génération de la Présentation suit une logique d'utilisation de la même façon que Diane+.
J. Foley suit une démarche similaire puisqu'il part du modèle conceptuel de l'application (constitué de la description des objets et des actions de l'application) pour en déduire la Présentation [FOLEY 90a] [FOLEY 93a]. Il utilise pour cela d'une part des règles de sélection qui permettent de choisir une catégorie d'objets de Présentation en fonction de la description des données, d'autre part des règles d'organisation et de disposition permettant de disposer de façon cohérente et agréable les différents objets d'interaction ainsi créés. Nous n'utilisons pas, quant à nous, de règles de disposition automatique comme le fait J. Foley. Nous organisons seulement les menus en fonction de la spécification des buts (et de leurs composants) et nous regroupons les différentes zones de saisie et d'affichage par OPAC et toujours en colonne. L'intégration de règles de disposition et d'organisation de J. Foley serait intéressante pour la génération des menus et des fenêtres. Les règles de sélection pourrait quant à elle être réparties entre le module de génération et les OPACs. En effet un OPAC est capable de fournir plusieurs Présentations ; le choix de la Présentation la mieux adaptée serait alors traité par le module de génération qui demanderait ensuite aux OPACs de fournir la (ou les) Présentations correspondante à son choix.

2.3. Gestion Automatique

Les spécifications Diane+ donnent naissance à des règles d'enchaînement qui sont présentées sous la forme $X \Rightarrow \alpha$, où X et α sont de même nature (opération ou but), et X est l'ensemble des opérations (ou des buts) qui précèdent directement α dans la procédure (ou le but). La figure 6.24 donne un exemple de règles. Les règles d'enchaînement sont la représentation figée de la dynamique de l'application. Elles fournissent en effet tous les enchaînements élémentaires obligatoires ($A \Rightarrow B$, $B \Rightarrow C$, $E \Rightarrow F$, etc.) entre les opérations à l'intérieur d'une même procédure ou d'une même opération, et entre les buts à l'intérieur d'un but plus général. Une procédure peut contenir plusieurs niveaux d'imbrication d'opérations, et le passage d'un niveau à l'autre est géré entièrement par le contrôleur de dialogue.

Nous allons voir de façon générale dans ce paragraphe comment les règles peuvent être utilisées pour la gestion automatique de la dynamique et de l'aide (la gestion interne des règles pour ces deux utilisations sera abordée plus loin).

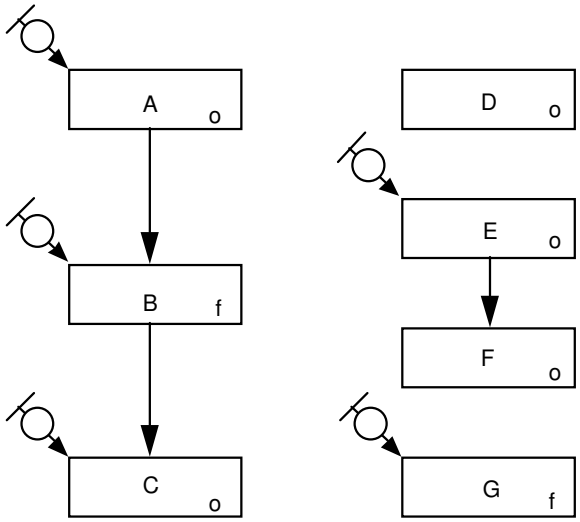


Figure 6.24 : Exemple de procédure Diane+. Les règles générées correspondantes sont (A ⇒ B), (B ⇒ C), (E ⇒ F), (e ⇒ A), (e ⇒ D), (e ⇒ E), (e ⇒ G), (C D F G ⇒ s)[113]

2.3.1. Gestion Automatique de la dynamique

La gestion de la dynamique est basée sur l'utilisation conjointe des règles d'enchaînement et de l'implémentation objet des concepts Diane+. Le cœur du contrôleur de dialogue est un moteur d'inférence qui scrute perpétuellement les éventuels changements d'état dans les concepts Diane+ (par exemple une opération dont la pré-condition devient vraie) et qui répercute ces changements par l'intermédiaire des règles d'enchaînements. Lorsqu'un tel changement survient, le moteur envoie des messages aux objets (opérations, procédures et buts) dépendant de ce changement pour les informer de l'évolution. Les objets en question prennent alors en charge la gestion de leur état et les éventuelles modifications qu'ils doivent exécuter.
Les règles prennent ici toute leur importance. En effet il est possible de modifier les précédences entre les opérations sans pour cela être obligé de rebâtir complètement l'application comme cela serait nécessaire si l'on avait implémenté les précédences dans le code des traitements. Grâce à l'utilisation des règles, on peut donc ajouter, modifier ou retirer des précédences sans aucun problème, même lorsque l'application est en cours d'utilisation. Par ailleurs, comme les règles et les objets représentant les concept, Diane+ (de tels objets seront dorénavant appelés *objet Diane+*) sont dissociés, il est possible de modifier les caractéristiques des objets Diane+ sans perturber les règles d'enchaînement. On peut par exemple modifier les contraintes ou le type d'une opération (à condition de ne pas enfreindre les règles syntaxiques). Par contre, on ne peut modifier le déclencheur et le mode des opérations puisque ceux-ci interviennent directement dans la génération de la Présentation.

Chaque objet Diane+ gère ses représentants externes en plus de son état. Ainsi tout changement de son état est immédiatement reflété sur l'interface. Par exemple une opération qui devient accessible à la suite de l'application d'une règle d'enchaînement, rend ses représentants externes accessibles pour l'utilisateur (qui pourra alors agir dessus). A l'inverse, une opération qui devient inaccessible rend ses représentants externes inaccessibles pour l'utilisateur (par exemple en grisant les widgets associés). De cette façon nous sommes sûrs que les représentants externes ne pourront pas émettre d'événement parasite durant la période d'inaccessibilité de l'objet Diane+.

[113] "e" correspond au point d'entrée de la procédure et "s" correspond au point de sortie. Tous les deux sont reliés aux opérations qui en dépendent (respectivement (A, D, F, G) et (C, D, F, G)).

Le Chapitre 5 a montré que les concepts Diane+ sont imbriqués. Ainsi un but contient des procédures qui contiennent à leur tour des opérations pouvant elles-mêmes contenir plusieurs niveaux de sous-opérations. Cette imbrication est prise en compte pendant la gestion de la dynamique.

- gestion des imbrications de l'extérieur vers l'intérieur : lorsqu'un objet Diane+ change d'état, il informe tous les objets Diane+ qu'il contient du changement à adopter. Par exemple, un but qui devient accessible demande aux procédures concernées de se rendre activables ; à leur tour, les procédures vont répercuter ce changement d'état sur leurs opérations et plus particulièrement sur les premières opérations des blocs qui les composent.
- gestion des imbrications de l'intérieur vers l'extérieur : les objets Diane+ peuvent modifier l'état des objets qui les contiennent. Ainsi une opération fille facultative qui se termine informe sa mère de sa terminaison, ce qui a pour effet d'incrémenter le compteur de contraintes. Si la contrainte devient vraie (c'est-à-dire si le nombre maximal de déclenchements de filles facultatives est atteint), l'opération mère demande alors à ses filles facultatives de se rendre inaccessibles (ce message peut éventuellement se propager si les filles facultatives sont composées d'autres opérations). Par ailleurs, si l'opération mère ne contient que des filles facultatives, la vérification de la contrainte implique sa terminaison.

Remarquons pour terminer que la gestion de la dynamique est simplifiée grâce à la capacité d'auto-gestion des widgets. Par exemple l'ouverture d'un menu et la sélection de l'une de ses commandes est prise en compte par le menu lui-même et non pas par le contrôleur de dialogue. Par contre si l'objet Diane+ associé au menu devient inaccessible, c'est le contrôleur qui demandera au menu de se griser.

2.3.2. Gestion Automatique de l'aide

Les règles ne fournissent des renseignements que sur les traitements. Toute l'aide que l'on pourra en retirer sera donc basée sur les traitements et pas sur les données. Nous verrons un peu plus loin que les aides dépendent toutefois du contexte d'exécution, donc également un peu des données.

Nous avons vu auparavant qu'une application contient deux types d'aides : l'aide fonctionnelle et l'aide d'utilisation, ces deux aides pouvant être ou non contextuelles. Dans notre cas, l'aide fonctionnelle est statique. Pour cela, lors de la description d'une opération, nous écrivons un texte (qui sera affiché lors de la demande de l'aide fonctionnelle) qui explique le fonctionnement de l'opération. Par exemple la commande Imprimer expliquera la signification des paramètres affichés dans la boîte de dialogue (nombre de copies, numéros des pages à imprimer, etc.). Les règles ne sont donc d'aucune utilité ici. Pour l'aide d'utilisation par contre, elles se révèlent d'un apport précieux. Si nous reprenons la procédure de la figure 6.24, l'utilisateur pourra demander par exemple ce qui se passera s'il déclenche l'opération "A". Quatre types de questions peuvent être posées :

- *Que se passe-t-il si on déclenche l'opération α ?* Dans ce cas, on fournit au contrôleur de dialogue le fait que l'opération α est supposée terminée. Le contrôleur, qui fonctionne ici comme un moteur d'inférence en chaînage avant, propage ce fait dans les règles et en ressort tous les nouveaux faits qui en découlent, c'est-à-dire l'état des opérations à cet instant (déclenchable, déclenchée, active, terminée, bloquée, non initialisable, etc.). On remarque donc que l'aide résultante dépend du contexte dans lequel on l'a appelée. Il est également possible de fournir une aide indépendante de ce contexte. Pour cela il suffit, lors des inférences, de ne pas faire intervenir les états courants des opérations.

Si l'on reprend la question "Que se passe-t-il si on déclenche 'A' ?", le moteur fonctionne alors de la façon suivante :

- on envoie un message à "A" pour lui dire de se terminer ce qui a pour effet de mettre son état à "terminée"[114]. Étant donné que chaque opération se gère elle-même, elle détecte sa fin d'exécution et envoie un message à ses successeurs (c'est-à-dire à "B") par l'intermédiaire du contrôleur de dialogue pour les informer de cette terminaison. Or "B" se gère également elle-même ce qui a pour effet de la rendre accessible pour l'utilisateur. Le type facultatif de "B" l'oblige à répercuter ce message (par l'intermédiaire du contrôleur de dialogue) à ses successeurs (c'est-à-dire "C"). "C" étant obligatoire, elle ne transmet pas ce message.

- le contrôleur a continué d'inférer pendant ce temps et à chaque inférence il a regardé quelles sont les opérations qui sont devenues accessibles dans la procédure en cours (ainsi que dans les sous-opérations de cette procédure s'il y en a). Il a donc détecté d'abord "B", puis "C" qu'il affiche dans une liste à l'écran. Si l'utilisateur désire aller plus en avant, il choisit soit de simuler la terminaison d'une des opérations affichées dans la liste afin de voir la prochaine étape réalisable (bouton Exécuter la sélection dans la figure 6.25.a), soit de simuler le plus loin possible la terminaison de l'opération sélectionnée. Dans les deux cas il relance l'aide, et le moteur infère de nouveau. Dans le second cas, chaque fois qu'une nouvelle opération est rencontrée, le moteur la déclare terminée et propage ce fait. Ce type d'aide correspond à une aide en profondeur.

Les figures 6.25.a et 6.25.b donnent un exemple de ce type d'aide. La fenêtre de la figure 6.25.a correspond à la demande et la fenêtre de la figure 6.25.b à la réponse. L'indentation dans la liste réponse reflète l'imbrication des opérations, c'est-à-dire que l'affichage de l'abonné correspond à l'affichage de son nom et de son téléphone. Pour chaque opération on fait apparaître son état lors du résultat (exécution signifie qu'elle est déclenchée automatiquement par le système, activable signifie que l'utilisateur peut la déclencher, etc.).

[114] Pour que cette aide soit réalisable, nous sommes obligés de travailler sur des copies des opérations sinon la propagation des faits "opération α terminée" peut être irrémédiable. Les langages objet tels que Smalltalk fournissent toutes les méthodes nécessaires à la duplication d'objets. Par ailleurs cette duplication n'est que temporaire, et là encore les langages objets se révèlent très intéressant puisqu'ils gèrent dynamiquement les objets (allocation de mémoire et récupération de cette mémoire à la destruction des objets).

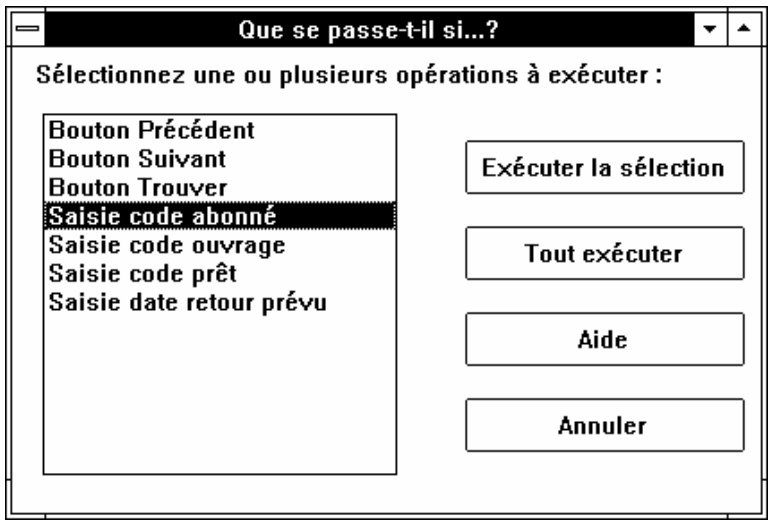


Figure 6.25.a : Fenêtre d'aide permettant de connaître les conséquences d'une action. Le bouton "Tout exécuter" permet de donner l'aide avec une recherche maximale en profondeur, c'est-à-dire que l'on cherche toutes les conséquences possibles de l'action choisie et non pas seulement les conséquences directes.

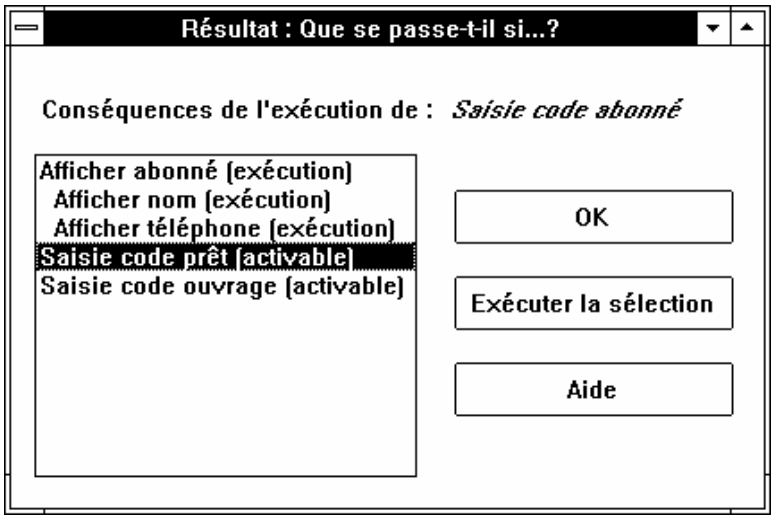


Figure 6.25.b : Exemple de fenêtre résultat pour la question "Que se passe-t-il si...?". L'utilisateur a choisi "Exécuter la sélection" dans la fenêtre précédente. Le bouton "Exécuter la sélection" de cette fenêtre permet de redemander de l'aide à partir du résultat.

- *Comment faire pour atteindre α ?* Suite à cette demande de la part de l'utilisateur, le contrôleur affiche la liste des opérations et des buts en rapport avec la procédure et l'application en cours. Il faut en effet différencier ces deux types d'aides.

  - dans le premier cas (bouton radio Opérations dans la figure 6.26.a), une commande est inaccessible (par exemple un bouton grisé) et l'utilisateur désire savoir comment la rendre valide. On demande donc au moteur d'inférer en chaînage arrière avec comme but à atteindre l'opération fournie par l'utilisateur. Si l'on reprend la procédure de la figure 6.24, l'utilisateur peut demander par exemple comment exécuter l'opération "F". Le moteur lui répond alors qu'il doit d'abord exécuter "E" et que lorsque cela sera fait, l'opération "F" sera lancée automatiquement. De même que précédemment, cette aide dépend du contexte en cours. Si l'on veut s'en abstraire, il suffit de ne pas tenir compte

des états des opérations. Il est également possible de repartir de la solution fournie par le moteur et de remonter encore plus loin si cela est possible. On peut là aussi reculer à chaque fois d'une étape ou bien encore remonter directement jusqu'au point le plus éloigné permettant d'atteindre l'opération voulue. On remonte ainsi d'abord jusqu'à la procédure, puis successivement dans tous les niveaux de sous-buts pour terminer sur le premier niveau de buts (niveau 1). On recueille ainsi à la fois des buts et des opérations. Pour les premiers on affiche Sélectionner (but), pour les seconds on affiche Exécuter (opération).
Le chaînage arrière fonctionne de la façon suivante : on donne au moteur comme but l'opération que l'utilisateur veut atteindre. Puis le moteur extrait de la procédure en cours toutes les règles qui ont pour terminaison cette opération. On regroupe alors, dans une liste qu'on affiche à l'écran, les opérations présentes dans ces règles. Si l'on désire tenir compte du contexte, on n'affichera que les opérations qui ne sont pas terminées et on précisera celles qui sont en cours d'exécution (Figure 6.26.a et 6.26.b).

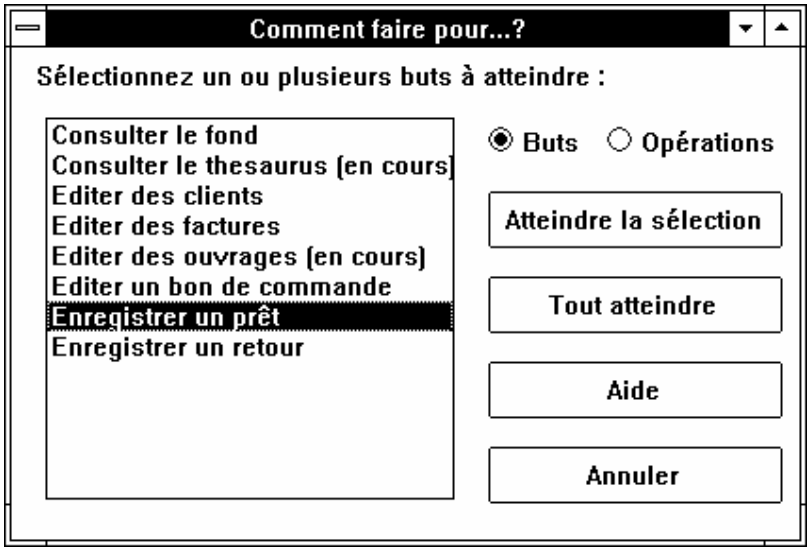


Figure 6.26.a : Fenêtre d'aide permettant de trouver le chemin pour atteindre un but ou une opération

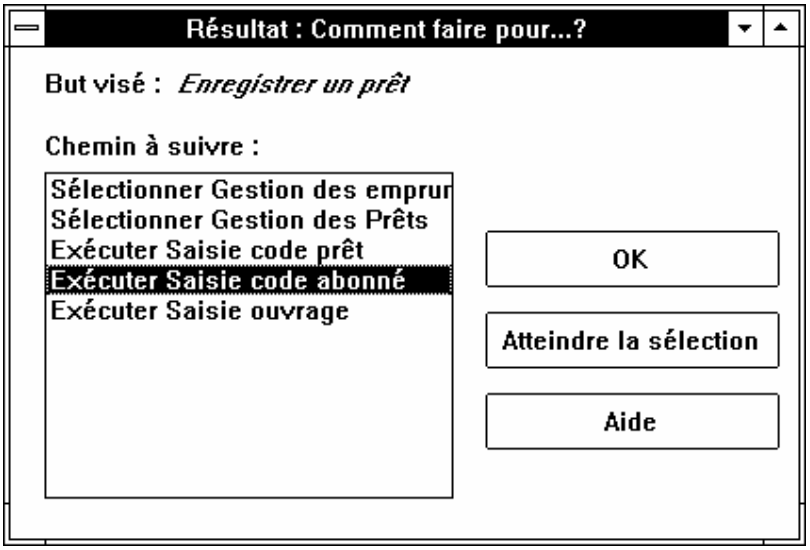


Figure 6.26.b : Exemple de fenêtre résultat pour la question “Comment faire pour...?”. L’utilisateur a choisi “Tout atteindre” ce qui signifie que l’on remonte jusqu’au niveau des buts de niveau 1. L’ordre affiché correspond à la séquence à suivre pour atteindre le but recherché.

- dans le second cas (bouton radio Buts dans la figure 6.26.a), l’utilisateur demande une aide plus générale. Dans ce cas, le contrôleur affiche tous les buts qu’il est possible de réaliser dans l’application. Ces buts ont été saisis par le concepteur et ils ont été reliés aux procédures et aux opérations adéquates. Ainsi lorsque l’utilisateur choisit un but particulier, cela revient à demander quelles sont les procédures et/ou les opérations à réaliser pour atteindre ce but. Il suffit alors de lister l’ensemble des opérations obligatoires présentes dans la procédure associée. En fait l’ensemble des opérations est plus important car il inclut également les opérations facultatives sur lesquelles reposent des contraintes de déclenchement. Il est également important de faire apparaître les précédences entre les opérations. Comme précédemment, on ne se contentera pas de lister les opérations, mais on les groupera suivant l’ordre de déclenchement. De même que pour l’aide précédente, il est possible de tenir compte ou non du contexte en cours.

Dans les deux cas que nous venons de voir, il doit être possible de rechercher un but ou une opération grâce à un mot-clé (Figure 6.27). Cette demande est formulée par l’intermédiaire de la commande Rechercher un but... du menu de la figure 6.28.

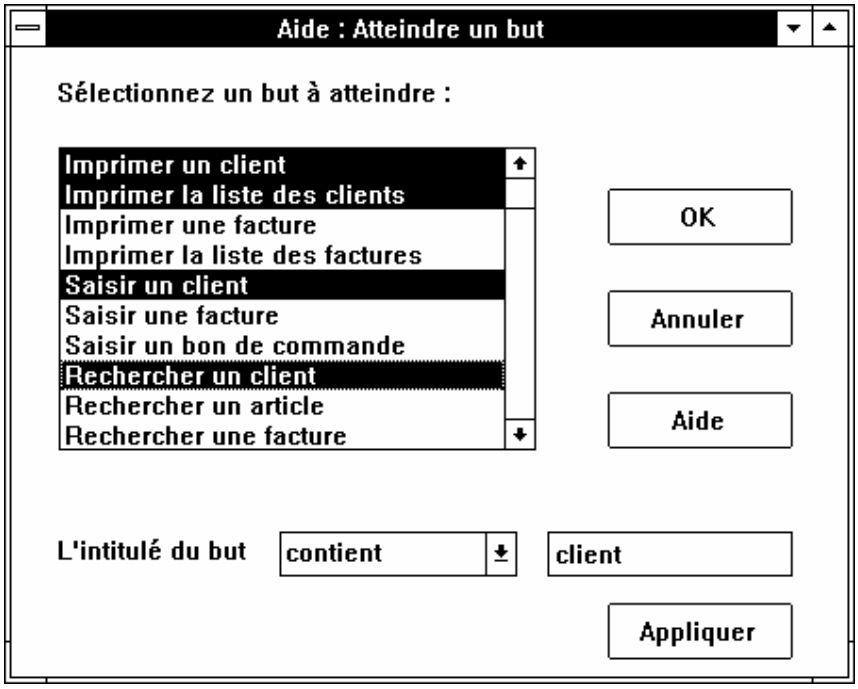


Figure 6.27 : Exemple de recherche de buts par mot-clé

- *Pourquoi une commande est-elle grisée ?* L'utilisateur peut en effet être surpris de voir qu'un bouton ou une commande dans un menu soit grisé[115]. Il décide donc de demander une explication sur cette inaccessibilité. Celle-ci peut avoir cinq motifs :
  - la pré-condition de l'opération n'est pas vérifiée,
  - les opérations obligatoires précédentes ne sont pas terminées,
  - l'opération est facultative et la contrainte de l'opération mère est vérifiée,
  - l'opération mère n'est pas active,
  - l'opération est non initialisable ou pas encore initialisée (la définition des états des opérations est donnée au § 3.2).

  Ainsi dans la figure 6.24, si l'utilisateur demande pourquoi "C" n'est pas accessible, le système lui répondra, en fonction du contexte de la demande, que :
  - la pré-condition de "C" n'est pas vérifiée,
  - "C" est non initialisable,
  - "A" n'est pas terminée ou qu'elle n'a pas encore été exécutée,
  - la procédure associée n'est pas active.

  Pour les deux dernières réponses, l'utilisateur pourra demander un complément d'information (Pourquoi "A" n'est pas terminée ?, Pourquoi la procédure n'est pas active ?, Comment terminer "A" ?). S'il désire savoir pourquoi les opérations précédentes ne sont pas actives, il ne pourra le faire qu'au pas à pas seulement, car ce type aide dépend fortement du contexte. On ne peut donc pas remonter directement jusqu'au point le plus haut comme précédemment, ce qui n'a aucun sens ici puisque correspondant en fait à l'aide "Comment faire pour...".

- *Comment terminer α ?* Cette aide est utilisée en complément de l'aide "Pourquoi α est grisée ?". En effet suite à une question de ce type, le système peut répondre qu'une des

[115] Le widget en question doit correspondre à une opération, à une procédure ou à un but. Nous supposons dans ce paragraphe que le widget est associé à une opération.

opérations précédentes n'est pas terminée. L'utilisateur voudra donc savoir comment achever cette opération.

Une autre utilisation de cette aide est lorsque l'utilisateur veut quitter le système et que le contrôleur de dialogue lui signale qu'il y a encore des opérations actives. Si l'utilisateur veut quitter proprement l'application, il pourra grâce a cette aide déterminer les actions à exécuter afin d'achever ces opérations.

Dans les deux cas, le contrôleur liste la suite des actions à suivre, par exemple "terminer α" si α est une opération élémentaire, "déclencher β" (β étant un but ou une opération", ou bien encore "faire α,β, χ puis δ" dans le cas d'une opération mère, d'une procédure ou d'un but. Pour obtenir ce résultat, le contrôleur de dialogue se base essentiellement sur les règles d'enchaînement et sur la composition des opérations (vérification des pré- et des post-conditions).

Toutes ces aides seront disponibles par l'intermédiaire d'un menu général (Figure 6.28). Ce menu présentera en plus d'une aide sur l'aide et de l'aide fonctionnelle classique, les commandes :

- "Comment atteindre..." qui correspond à la question "Comment faire pour atteindre α ?". Un clic sur cette commande provoque l'ouverture d'une fenêtre contenant la liste des buts et des opérations.
- "Comment terminer..." qui correspond à la question "Comment terminer α ?". Cette commande provoque l'ouverture d'une fenêtre contenant la liste des buts ou des opérations en cours (sélection par des boutons radio).
- "Que se passe-t-il si...?" Avec cette commande, la souris se transforme en un point d'interrogation que l'utilisateur déplace sur la commande ou le widget qui l'intéresse, puis il clique sur cette commande ou ce widget, et le système liste alors comme nous l'avons vu précédemment toutes les conséquences de cette action.
- "Expliquer le grisé" qui correspond à "Pourquoi une commande est-elle grisée ?". Lorsque l'utilisateur sélectionne cette commande, il obtient également un point d'interrogation, mais d'un aspect grisé pour lui dire qu'il n'est utilisable que pour les zones inaccessibles.
- "Rechercher un but" qui permet de sélectionner un objectif (but ou opération) parmi la liste de tous ceux existants.

Pour que ces aides fonctionnent, il est nécessaire que les opérations et leur représentation externe soient en étroite relation. Ceci est réalisé lors de la génération de l'interface. A chaque widget correspond une opération, mais à une opération peut correspondre plusieurs widgets (par exemple une commande dans un menu et un bouton sur la fenêtre). Chaque widget connaît donc l'opération qu'elle représente. Ainsi lorsque l'utilisateur demande une aide fonctionnelle sur une commande particulière, le système est capable de déterminer quelle est cette commande, et donc quelle aide il faut fournir.

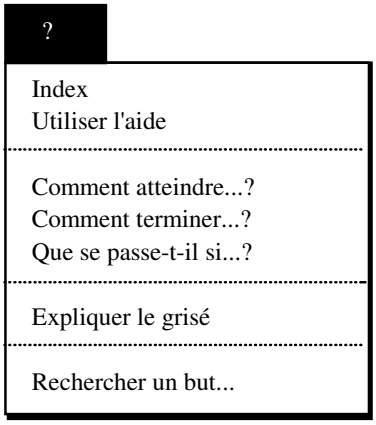


Figure 6.28 : Le menu d'aide.
Les deux premières commandes sont l'aide fonctionnelle et l'aide sur l'aide, les autres sont pour l'aide d'utilisation.

2.3.3. Discussion

Grâce aux extensions de Diane+, il est donc possible de gérer automatiquement la dynamique et l'aide d'utilisation. En ce qui concerne la gestion de la dynamique à partir de spécifications conceptuelles, l'utilisation des pré-conditions est la plus courante. Elle permet de décomposer et de diluer les séquencements d'opérations à travers les différents intervenants du dialogue. Cette pratique des pré-conditions se retrouve par exemple dans Siroco-Guide [NORMAND 92a] et dans UIDE [FOLEY 91]. Par ailleurs, Siroco-Guide utilise des classes de contrôle pour générer les modules chargés de gérer la dynamique. On trouve ainsi des contrôleurs associés aux "Sessions", aux "Workspaces", aux opérations et aux "Concepts". A l'inverse, UIDE regroupe la gestion de la dynamique dans un gestionnaire de dialogue qui est chargé d'inspecter les objets Actions de l'application pour y détecter des changements d'états qu'il se charge de faire répercuter sur l'interface. Notre façon de gérer la dynamique est donc très proche de celle de UIDE.
La gestion automatique de l'aide est quasi inexistante dans Siroco-Guide. Elle apparaît en effet sous la forme de commentaires courts ou longs saisis par le concepteur, et dépeignant les entités manipulés, ainsi que sous forme de textes générés à partir des structures décrites dans les spécifications (par exemple les "Workspaces"). Au contraire J. Foley tire pleinement parti de l'utilisation des pré- et des post-conditions [GIESKENS 92]. Grâce à celles-ci, le gestionnaire de dialogue est capable de répondre à deux questions [FOLEY 90b] :

- *"Pourquoi un objet d'interaction est inaccessible ?"*. L'explication est fournie directement à partir des pré-conditions.
- *"Comment faire une action ?"*. L'aide résultante est une animation qui reflète fidèlement ce que devra faire l'utilisateur pour exécuter l'opération choisie. Pour créer cette animation, J. Foley utilise d'une part des scénarios d'animation (par exemple le clic est représenté par trois images successives de la souris, avant, pendant et après le clic), et d'autre part les "Techniques d'Interaction" qui sont attachées aux objets d'interaction, eux-mêmes attachés aux objets et aux actions de l'application. Il est donc possible de partir de l'action désirée et de remonter jusqu'à l'objet d'interaction et son mode de fonctionnement.

Notre travail incorpore ces deux types de question, mais il ne gère pas l'animation. Celle-ci est toutefois possible. Étant donné que l'on est capable de retrouver le chemin à suivre pour atteindre un but ("*Comment faire pour atteindre...?"*), et que ce chemin fait appel à différents objets Diane+, il est donc possible de connaître les représentants externes associés à ces objets et donc la

façon de les utiliser. Certes le travail n'est pas sans difficulté technique, mais il ne demande que peu de modifications quant à la gestion de l'aide. Par ailleurs l'animation pourrait également être appliquée aux questions "Comment terminer...?", et "Que se passe-t-il si...?".

2.4. Évaluation automatique

Ce paragraphe propose une discussion sur la possibilité qu'offrent Diane+ et notre outil d'incorporer une évaluation automatique des spécifications et de l'ergonomie. Notre réflexion n'a pas été pour l'instant plus approfondie, mais nous la présentons dans le but d'ouvrir des horizons pour l'évaluation automatique des spécifications et de l'ergonomie ainsi que pour l'intégration automatique de cette dernière.

2.4.1. Évaluation de l'ergonomie

Outre les règles d'enchaînement, il peut être intéressant d'incorporer dans notre outil des règles d'ergonomie générale. Ces deux bases de règles sont bien sûr implémentées indépendamment l'une de l'autre, mais le contrôleur de dialogue les utilise conjointement. Ces règles d'ergonomie peuvent être utilisées lors des sessions de travail, mais également au cours du développement, par exemple lors de la génération de l'interface. Trois natures de règles ergonomiques existent :

- les règles facilement contrôlables. Elle sont utilisées principalement lors de la génération de l'interface et sont accompagnées de poids (ceux-ci sont représentées ci-dessous par des chiffres compris entre 0 et 1, à titre d'exemple). Ce sont des règles du type[116] :
  - les boutons sont placés horizontalement en bas de la fenêtre (0.7),
  - les boutons sont placés verticalement à droite dans la fenêtre (0.3),
  - les zones de saisie et d'affichage sont placées verticalement dans la fenêtre (0.5),
  - la barre des menus contient entre cinq et neuf menus (0.6),
  - les menus contiennent entre cinq et neuf commandes (0.6),
  - les opérations dont la nature est de type "alerte" sont présentées dans une boîte de dialogue de type "alerte" avec une icône en forme de point d'exclamation (0.8),
  - les opérations de nature dangereuse demande une confirmation[117] (0.9),
  - les menus ne contiennent pas plus de trois niveaux de sous-menus (0.9),
  - il existe un menu par but (0.7),
  - les commandes des menus sont classées par ordre chronologique de déclenchement (0.6),
  - les opérations facultatives isolées sont groupées à part à la fin des menus (0.7),
  - chaque opération durant plus d'une seconde fait afficher le sablier pendant son exécution (0.4).

  On peut envisager deux utilisations possibles de ces règles pour l'évaluation. Soit le moteur les utilise pendant la génération de l'interface (par exemple il compte le nombre de menus à faire afficher et signale au concepteur tout débordement important de l'intervalle [5,9]), soit ces règles sont utilisées par un système expert, comme celui présenté dans [PALANQUE 92], qui évalue a posteriori l'interface générée et propose des solutions aux problèmes rencontrés. Quelques outils utilisent déjà ce type de règles. Citons par exemple

[116] La syntaxe des règles donnée ici n'est pas directement implémentable, mais elle donne une idée du contenu des règles.

[117] Ce type de règle est utilisé pendant les sessions de travail. Dès qu'une opération de nature dangereuse est déclenchée, le contrôleur de dialogue fait afficher une boîte de dialogue pour la confirmation. Le concepteur n'a donc pas à se soucier de ce type d'interaction.

Génius [JANSSEN 93] ou encore UIDE [FOLEY 92] qui utilise des règles de disposition des objets d'interactions.

- les règles difficilement contrôlables. La difficulté de ce contrôle vient d'une part de leur caractère cognitif et subjectif, d'autre part de leur caractéristique à être fortement dépendantes des tâches implémentées (ce qui n'est pas le cas des deux autres types de règles). Ce sont des règles telles que :
  - "le vocabulaire des commandes est homogène", c'est-à-dire qu'il emploie soit des verbes (consulter, imprimer,...) soit des substantifs (consultation, impression,...),
  - "les messages affichés sont courts, clairs, utilisent des mots simples et ont une tournure positive". Seule la longueur des messages est facilement contrôlable, le caractère positif et la clarté des messages sont plus difficiles à évaluer,
  - "une liste doit être affichée dans un widget de type scroll list". Il peut arriver pourtant que l'on doive faire afficher des listes dans des "combo box" ou sous forme d'un ensemble de cases à cocher. Des produits tels que Génius [JANSSEN 93] et UIDE [FOLEY 92] intègrent des règles de ce type et permettent de choisir le widget le plus approprié à une situation.

  Pour l'instant, seul l'humain est vraiment capable d'évaluer une application avec ces règles. Quand la représentation des connaissances pourra modéliser des concepts tels que la caractère positif et clair d'une expression, ces règles pourront alors être implémentées.

- les règles impossible à contrôler. Ce sont des règles qui font appel à des concepts qui sont à la fois extérieurs à l'application et fortement en relation avec elle. Ces règles sont par exemple :
  - le vocabulaire utilisé est celui de l'entreprise et il correspond à l'utilisateur,
  - les écrans générés respectent les anciens formulaires de l'entreprise.

  Des règles de ce type ne peuvent être vérifiées totalement que par un humain qui va évaluer par exemple la correspondance entre le vocabulaire de l'entreprise et celui de l'application. Le caractère fortement subjectif de ces règles empêche une implémentation et une évaluation automatique par un système expert.

2.4.2. Évaluation des spécifications

L'évaluation des spécifications doit consister en une évaluation formelle permettant l'identification d'erreurs telles que "l'opération α n'est jamais accessible" ou encore "la spécification est impossible". Nous n'avons pas approfondi plus avant cette réflexion qui se borne pour l'instant aux règles énoncées au Chapitre 5. D'autres règles peuvent cependant être envisagées. Prenons le cas d'une opération qui contient une opération obligatoire et une opération facultative, et sur laquelle on veut appliquer une contrainte. Si cette contrainte est "une et une seule", le système devra répondre : "Cette contrainte signifie que l'opération facultative est obligatoire. Vous devez changer le type de l'opération ou la contrainte". Si la contrainte est : "au moins deux", le système devra répondre : "Cette contrainte est impossible car vous n'avez qu'une seule opération facultative. Vous devez changer la contrainte ou ajouter une autre opération facultative". Ce type d'évaluation est en fait une vérification syntaxique que d'autres travaux tels que [PALANQUE 92] et [BASTIDE 92] proposent déjà. Cette vérification syntaxique peut intervenir pendant ou après les spécifications.

3. Représentation interne

Ce paragraphe expose d'une part la représentation machine des concepts Diane+ que nous avons exposés au Chapitre 5, et d'autre part l'implémentation des processus de génération et de gestion automatique énoncés au § 2.2 et 2.3. Nous commençons par présenter l'architecture générale d'une application développée avec Diane+, puis nous détaillerons la représentation des opérations, des procédures et des buts. Nous aborderons ensuite le fonctionnement du contrôleur de dialogue et nous terminerons par un bilan de la maquette déjà réalisée.

3.1. Architecture générale

D'après la figure 6.29, notre outil devrait fournir des applications décomposées en six couches. Sur le plan conceptuel, le Contrôle du Dialogue est assuré par les couches Stratégies et Buts, l'Interface du Noyau Fonctionnel est gérée par la couche des Procédures et le Noyau Fonctionnel est composé des OPACs et des données qu'ils représentent. La partie Présentation est quant à elle dispersée de la couche Buts à la couche des OPACs. Chacun des composants de ces couches gère sa Présentation (qui est facultative[118]) en fonction de son état et des ordres venant des couches supérieures.
Remarque : nous ne nous intéresserons pas ici à la couche Stratégies étant donné que nous ne pouvons pas l'intégrer pour l'instant.

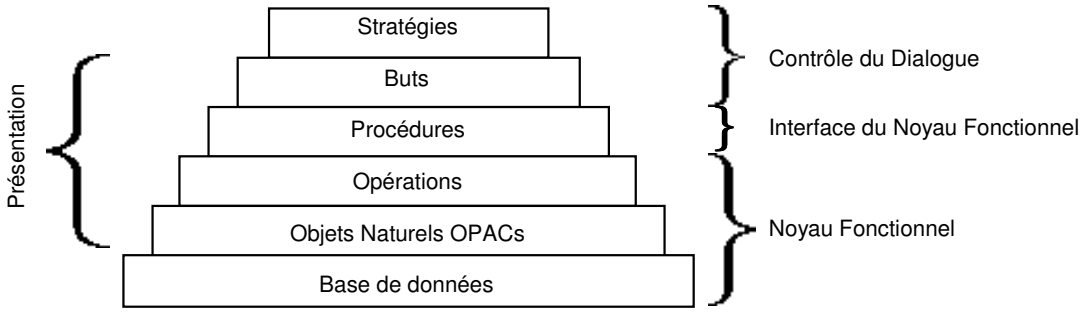


Figure 6.29 : Architecture six couches basée sur les OPACs.

La couche des données (composée des deux couches de base) ne sera pas détaillée dans ce rapport. Nous avons déjà précisé que notre travail repose sur une couche de données capables de se gérer par elle-même soit grâce à sa structure, par exemple avec les OPACs, soit grâce à une couche située juste au-dessus de celle des structures de données. Dans tous les cas, nous disposons d'un ensemble de données et d'un ensemble de traitements élémentaires sur ces données permettant une gestion individuelle pour les traitements de base (création, consultation, modification, etc.).

La couche suivante contient les opérations Diane+. Chaque opération est représentée en machine par une instance de la classe Opération que nous détaillerons plus loin (cf § 3.2). Notons déjà qu'une opération est en relation avec ses sous-opérations, avec des procédures, avec des OPACs et avec ses représentants au niveau de l'interface (widgets).

La couche Procédures représente les procédures Diane+. Chaque procédure est représentée par une instance de la classe Procédure qui sera abordée plus loin (cf § 3.3). Une procédure est en relation

[118] Un OPAC n'est pas obligé d'avoir une représentation externe. Cette caractéristique est issue du modèle PAC. De même, une opération n'a pas forcément une Présentation (c'est le cas par exemple des opérations automatiques).

avec ses opérations, avec un but, avec des OPACs ainsi qu'avec ses représentants au niveau de l'interface.

Les buts et les sous-buts seront également représentés par des instances d'une classe But (cf § 3.4). Un but connaît le poste de travail auquel il est rattaché, ainsi que les procédures qui permettent de le réaliser. Un but possède également un ou plusieurs représentants externes.

La figure 6.30 représente le méta-schéma conceptuel du système. La partie grisée à gauche contient les entités générées par le système et utilisées pour la représentation externe. La partie grisée en bas correspond à la couche des OPACs. Ils sont utilisés par le système, mais ils ne génèrent rien. Ils fournissent seulement leur représentation externe aux fenêtres qui les manipulent. Enfin la partie non grisée correspond à la représentation interne des concepts fondamentaux de Diane+.

Les buts, les procédures et les opérations sont générés à partir des spécifications Diane+ et sont représentés par des instances des classes correspondantes. De par leur nature, un but connaît ses procédures, et une procédure connaît ses opérations. Réciproquement une opération sait à quelles procédures elle appartient, de même qu'une procédure connaît le but qu'elle permet de réaliser.

Les widgets ayant également été générés à partir des spécifications (cf § 2.2.2), le système a créé des liens entre les composants Diane+ et leur représentation externe. Ainsi une opération connaît tous ses représentants externes (par exemple un bouton ou une commande dans un menu) et réciproquement un widget connaît le composant Diane+ qu'il représente (but, procédure, opération). Si le concepteur a modifié la représentation externe d'un composant (par exemple en changeant le libellé d'un bouton ou encore en changeant un menu de place), le lien reste inchangé. S'il supprime un widget, le lien disparaît.

L'intérêt de ces liens est d'augmenter l'efficacité des feed-backs. Nous avons vu qu'une opération est capable, lorsqu'elle change d'état, de modifier l'interface en conséquence. Sans ces liens, ces modifications devraient transiter par le contrôleur de dialogue, ce qui ralentirait considérablement les traitements.

De même que les composants Diane+, les widgets sont parfois liés entre eux. Lors de la génération de l'interface, les éléments d'une fenêtre se trouvent liés avec la fenêtre qui les contient. Par exemple les menus et les boutons sont directement dépendants de la fenêtre qui les affiche, et un menu contient différentes commandes, chacune étant issue d'une opération ou d'un sous-but. Ces liens entre les widgets permettent de gérer plus efficacement les événements produits par l'utilisateur. Tous les événements ne concernant que l'interface restent à l'intérieur de celle-ci. Tous les événements relevant de l'application sont d'abord traités par l'interface avant d'être envoyés à l'application. Par ailleurs ces liens ne doivent pas à être révélés au concepteur, mais ils doivent être gérés implicitement par le système lors de la génération de l'interface et lors des sessions de travail.

Le contrôleur de dialogue fait le lien entre la représentation externe et le noyau fonctionnel (procédures Diane+ et OPACs) en interceptant les interactions de l'utilisateur et en mettant à jour l'interface suite aux changements d'états du noyau fonctionnel.

- lorsque l'utilisateur travaille, il ne peut interagir qu'avec les éléments accessibles de l'interface. En effet tous les widgets qui sont inaccessibles d'après les spécifications Diane+, sont automatiquement invalidés. L'événement produit à la suite d'une interaction est intercepté par l'interface, puis envoyé à l'élément correspondant sous forme de message. L'élément exécute alors le traitement correspondant.

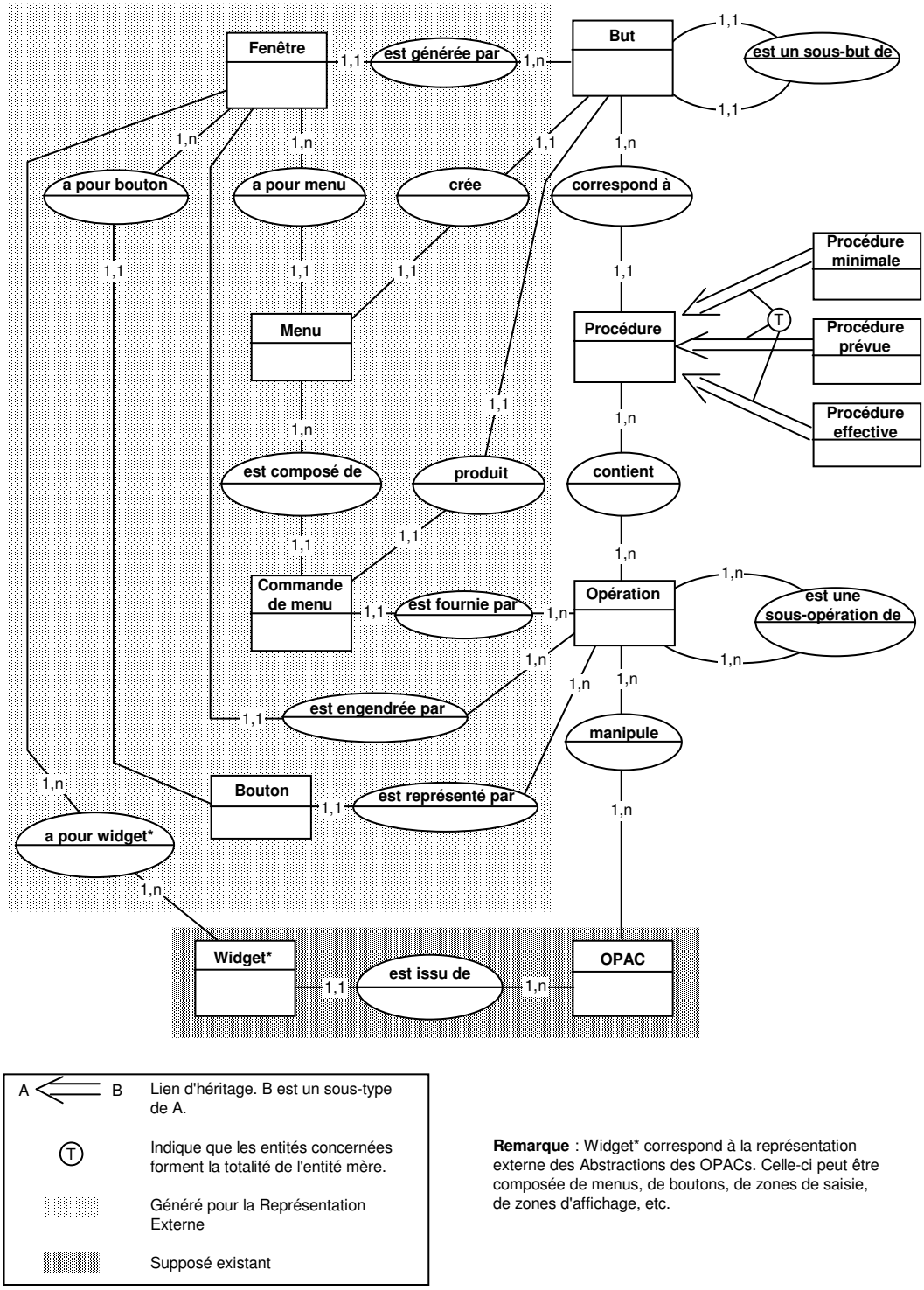


Figure 6.30 : Méta-schéma conceptuel de notre outil

- lorsqu'un élément du noyau fonctionnel change d'état, il envoie un message à ses représentants externes pour les informer de ce changement. Par exemple si une opération devient accessible pour l'utilisateur, elle envoie un message à ses représentants pour enlever le grisé des widgets.

On peut remarquer que le contrôleur de dialogue n'apparaît pas explicitement dans le schéma. Notre système propose en effet un contrôle externe au niveau conceptuel, mais distribué au niveau interne.
Les objets Diane+ se chargent de mettre à jour leur représentation externe suivant leur accessibilité (principalement en fonction des pré-conditions). Cette représentation externe est également gérée en partie par les widgets. Lorsqu'une opération devient invalide, elle se contente d'émettre un message vers ses représentants pour leur dire de se griser. Ce sont ces derniers qui exécutent l'invalidation au niveau externe. De même les widgets ne génèrent pas d'événements lorsqu'ils sont grisés.
Les widgets sont également responsables du déclenchement des opérations qu'ils représentent. Ils prennent en charge l'action de l'utilisateur (clic, double clic, etc.) et l'envoi d'un message à l'opération associée pour lui demander de s'exécuter.
La partie restante du contrôle du dialogue (gestion des enchaînements, gestion de l'aide et choix des procédures en fonction des buts, des postes de travail et de l'utilisateur) est à la charge du moteur d'inférence.

3.2. Objet opération

Chaque opération sera représentée en machine par une instance de la classe Opération. D'après le Chapitre 5, une opération peut être utilisée plusieurs fois[119], c'est-à-dire dans des opérations ou des procédures différentes. La structure d'une opération est composée d'une partie commune à toutes les occurrences et d'une partie propre à chacune d'elle. La structure des opérations sera donc la suivante :

- la partie commune qui contient :

  - *le nom* de l'opération. Il est donné par le concepteur lors des spécifications Diane+. Ce nom est également utilisé lors de la génération de l'interface pour les libellés des boutons et autres widgets, et lors de la gestion de l'aide.

  - *le traitement* associé. Il constitue le corps de l'opération et il est composé de deux parties : l'en-tête et le traitement lui-même. L'en-tête est généré à partir des spécifications. Il est constitué du nom de l'opération et des paramètres nécessaires à l'opération[120]. Ces paramètres sont les données que l'on a rattachées à l'opération en phase finale de conception. Le traitement est, quant à lui, saisi entièrement par le concepteur. Il peut faire appel à des primitives ou exécuter des calculs locaux, mais en aucun cas il ne doit appeler d'autres opérations issues des spécifications.

  - *la liste des opérations filles*. Cette liste est en fait une liste de pointeurs dirigés vers des instances de la classe Opération qui correspondent aux opérations filles. Cette liste fonctionne de la même façon que la liste des opérations pour une procédure (voir plus loin).

  - *les règles d'enchaînement* sur ses sous-opérations. Nous avons vu au Chapitre 5 qu'une opération peut contenir des sous-opérations. Nous allons voir plus loin que, grâce à

[119] Il est important de noter que l'utilisation multiple d'une opération implique également celle de ses sous-opérations. Si cela n'est pas souhaité, il faut alors créer une nouvelle opération (par exemple avec un copier-coller), puis modifier cette nouvelle opération.
[120] Notre implémentation étant basée sur Smalltalk/V, cet en-tête est réduit au nom de l'opération et de celui des paramètres car Smalltalk/V ne fait pas de différence entre les paramètres en entrée et ceux en sortie.

cette liste de règles, une opération peut gérer son état ainsi que celui de ses opérations filles.

- la partie propre à chaque occurrence contenant :

  - *les pré-conditions*. Une opération peut exécuter le même traitement dans des conditions différentes suivant le contexte où elle se trouve. Les pré-conditions sont donc attachées aux occurrences plutôt qu'à l'opération elle-même. Elles sont également accompagnées d'un texte (facultatif) utilisé pour l'aide d'utilisation, par exemple pour expliquer pourquoi une commande est grisée.

  - *les post-conditions*. Pour la même raison que les pré-conditions, les post-conditions sont associées aux occurrences. De même, elles possèdent un texte (facultatif) utilisé pour l'aide d'utilisation, par exemple lorsque l'utilisateur demande ce qui se passe s'il déclenche l'opération.

  - *les pré-* et *les post-actions.* Elles permettent de gérer plus efficacement la représentation externe. Elles permettent par exemple d'invalider des zones de saisie ou des boutons qui ne pouvaient pas l'être suite aux spécifications.

  - *le déclenchement utilisateur*. Cette variable est en fait un booléen qui est vrai lorsque l'opération est à déclenchement utilisateur et faux sinon. Cette variable est très utile pour l'auto-gestion de l'opération (voir plus loin).

  - *le type* qui précise si l'opération est obligatoire ou facultative. Cette variable est également très utile pour l'auto-gestion de l'opération.

  - *le mode* indiquant si l'opération est interactive ou non. Le mode est utilisé d'une part lors de la génération de l'interface et d'autre part lors des sessions de travail pour l'auto-gestion de l'opération.

  - *l'aide fonctionnelle* de l'opération. Ce texte facultatif explique à quoi sert l'opération et comment on doit l'exécuter, par exemple comment saisir une date ou un code client, ou encore comment remplir les paramètres de la boîte de dialogue pour l'impression.

  - *la nature*. Elle est facultative, mais son utilisation permet de réduire la charge du concepteur, par exemple une opération de nature dangereuse fait demander automatiquement une confirmation.

  - *l'interruptabilité*. C'est un booléen qui indique si une opération est interruptible (le booléen a la valeur vrai) ou non (le booléen a la valeur faux).

  - *le compteur de déclenchements*. Il permet de représenter les contraintes de déclenchement sur l'opération elle-même. Dès que la valeur du compteur sort de l'intervalle autorisé, le compteur en informe immédiatement l'opération propriétaire afin qu'elle prenne les dispositions nécessaires. Par défaut l'intervalle est [0;∞[puisque chaque nouvelle opération peut être exécutée autant de fois que l'on veut.
  - *la contrainte sur les sous-opérations*. Elle est représentée également par un compteur qui totalise le nombre de sous-opérations facultatives exécutées. De même que précédemment, le compteur informe immédiatement l'opération propriétaire dès qu'il sort de l'intervalle autorisé (par défaut [0;∞[).

- *l'état* de l'opération. La figure 6.31 représente les différents états qu'une opération peut prendre. Nous préciserons plus loin comment ces états sont gérés.

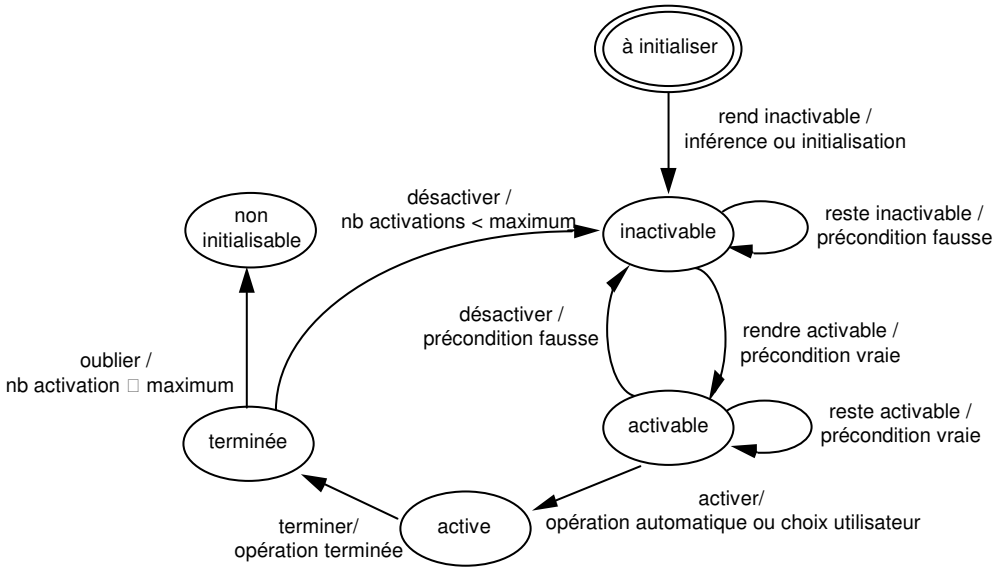


Figure 6.31 : Graphe des états d'une opération

- *la liste des représentants externes*. Cette liste est constituée de pointeurs dirigés vers les widgets qui représentent l'opération au niveau externe. L'opération ne connaît que ces pointeurs ; elle ne connaît ni le libellé, ni la position, ni le type des widgets, ni les liens qu'ils ont avec d'autres widgets.

- *le propriétaire de l'occurrence*. Il correspond au bloc où apparaît l'opération. Ce bloc peut être une opération ou une procédure. Pour désigner le propriétaire, nous utilisons un pointeur sur l'objet qui le représente. Il est ainsi possible de connaître son nom, son type, son état, etc.

Exemple 1 : Suppression d'un employé dans la liste des employés (Figure 6.32)

La représentation de cette opération sera la suivante[121] :

- Partie commune à toutes les occurrences :
  - *nom* : **'Supprimer'**
  - *traitement* :

    en-tête: **supprimer: (Employé, Liste_des_Employés)**

    corps : **nil** "le traitement est réparti dans ses trois opérations filles"
  - *liste des opérations filles* [122]: **'Saisie', 'OK', 'Suppression'**.
  - *règles de précédence*[123] :

    ((e ⇒ 'Saisie'), (e ⇒ 'OK'), ('Saisie', 'OK' ⇒ 'Suppression'), ('Suppression' ⇒ s))

[121] Le texte en gras correspond au code généré soit directement par le système soit par l'intermédiaire des saisies du concepteur (par exemple pour le type d'une opération).

[122] Nous rappelons que cette liste contient en fait des pointeurs vers les instances de ces opérations.

[123] Nous représentons de manière schématique les règles d'enchaînement. En fait les règles manipulent des pointeurs sur des opérations au lieu de manipuler les noms des opérations.

- Partie propre à cette occurrence :
  - *pré-conditions* : **nil**
  - *post-conditions* : **nil**
  - *pré-actions* : **nil**
  - *post-actions* : **nil**
  - *déclenchement utilisateur* : **true**
  - *type* : **facultative**
  - *mode* : **interactive**
  - *état* : 'à initialiser' "l'état dépend du contexte de travail"
  - *aide fonctionnelle* : 'Pour supprimer un employé, vous devez avoir affiché au préalable l'employé concerné, puis vous devez cliquer sur le bouton Supprimer'
  - *nature* : **nil** "Seule Suppression est dangereuse"
  - *interruptible* : **true** "Seule Suppression est ininterruptible"
  - *compteur de déclenchement* :

    intervalle : **[0, ∞[**

    valeur : 0 "la valeur dépend du nombre de déclenchements effectués"
  - *contrainte* :

    intervalle : **[0, ∞[**

    valeur : 0 "la valeur dépend du nombre d'opérations filles facultatives effectuées"
  - *représentants externes* : **nil**
  - *propriétaire*:

    procédure Gestion des employés "donné à titre d'exemple"

Nous donnons également la représentation de l'opération fille Suppression :

- Partie commune à toutes les occurrences :
  - *nom* : **'Suppression'**
  - *traitement* :

    en-tête : **suppression: (Employé, Liste_des_Employés)**

    corps :

    Message YesOrNo with: ('Voulez-vous vraiment supprimer', Employé getNom**)**

    ifTrue: [fin := (Liste_des_Employés remove: Employé) **]**

    ifFalse: [fin := false **].**
  - *liste des opérations filles* : **nil**
  - *règles de précédence* : **nil**

- Partie propre à cette occurrence :
  - *pré-conditions* : Employé getContents "on vérifie que le contenu est correct"
  - *post-conditions* : fin
  - *pré-actions* : Liste_des_Employés disable "donné à titre d'exemple"
  - *post-actions* : Liste_des_Employés enable "donné à titre d'exemple"
  - *déclenchement utilisateur* : **false**
  - *type* : **obligatoire**
  - *mode* : **automatique**
  - *état* : 'activable' "l'état dépend du contexte de travail"
  - *aide fonctionnelle* : 'Lorsque vous cliquez sur ce bouton, vous supprimez l'employé affiché'
  - *nature* : '**dangereuse**'
  - *interruptible* : **false**
  - *compteur de déclenchement* :

    intervalle : **[0, ∞[**

valeur : 0

- *contrainte* :

  intervalle : **[0, ∞[**

  valeur : 0

- *représentants externes* [124]:

  Button new label: 'Suppression'

  position: 100 @ 120;

  height: 30;

  width: 60;

  font: System 12 @ 10;

  when: #clicked perform: #cliquer[125]

- *propriétaire*:

  opération Supprimer[126].

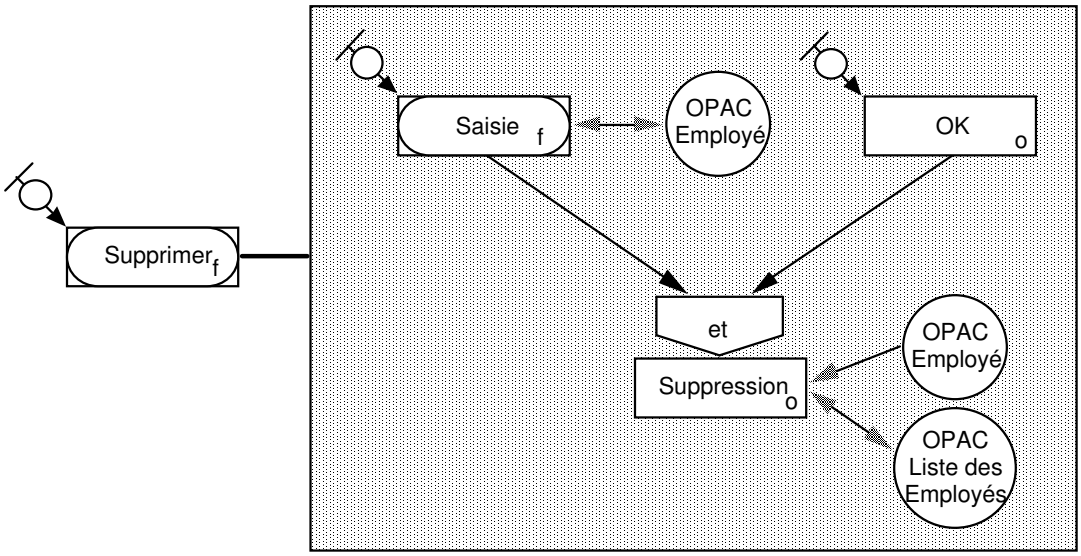


Figure 6.32 : Suppression d'un employé dans la liste des employés

[124] Nous donnons à titre d'exemple ce que serait le code du bouton associé à l'opération. En fait représentants externes contient une liste de pointeurs. Le bouton Suppression est en fait une instance de la classe Button et peut être repéré grâce à son adresse mémoire comme tous les autres objets. Le lien entre la description de l'opération et l'adresse du bouton est automatiquement géré lors de la génération.

[125] La méthode cliquer se contente d'envoyer à l'objet opération représentée par le bouton le message self exécution. Ce message est compréhensible par toutes les instances de la classe Opération et provoque la vérification des pré-conditions, puis l'exécution des pré-actions et celle du traitement associé. Il déclenche ensuite la vérification des post-conditions et enfin l'exécution des post-actions.

[126] Nous rappelons que cette variable est en fait un pointeur sur l'occurrence correspondante de l'opération Supprimer.

Exemple 2 : une opération de Saisie avec vue externe explicite (Figure 6.33)

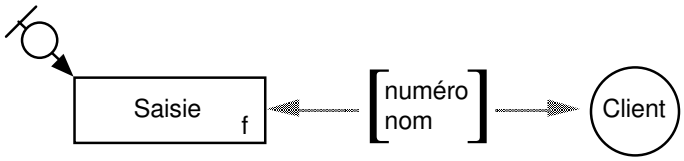


Figure 6.33 : Une opération de Saisie avec vue externe explicite

- Partie commune à toutes les occurrences :
  - *nom* : **'Saisie'**
  - *traitement* :

    en-tête : **saisie: (Client.numéro, Client.nom)**

    corps :

    fin := (Client.numéro getContents) and: (Client.nom getContents).
  - *liste des opérations filles* : **nil**
  - *règles de précédence* : **nil**
- Partie propre à cette occurrence :
  - *pré-conditions* : **nil**
  - *post-conditions* : fin
  - *pré-actions* : Client. adresse disable; Client.numéroSS disable
    "donné à titre d'exemple"
  - *post-actions* : Client. adresse enable; Client.numéroSS enable
    "donné à titre d'exemple"
  - *déclenchement utilisateur* : **true**
  - *type* : **facultative**
  - *mode* : **interactive**
  - *état* : 'activable' "l'état dépend du contexte de travail"
  - *aide fonctionnelle* : 'Cette opération vous permet de saisir uniquement le nom et le numéro d'un client'
  - *nature* : **nil**
  - *interruptible* : **true**
  - *compteur de déclenchement* :

    intervalle : **[0, ∞[**

    valeur : 0
  - *contrainte* :

    intervalle : **[0, ∞[**

    valeur : 0
  - *représentants externes* :

```
EntryField new
        owner: Client.nom;
        name: 'Nom :';
        position: (25 @15, 125 @ 40).
EntryField new
        owner: Client.numéro;
        name: 'Numéro :';
```

position: (25 @ 60, 90 @ 100). "les deux widgets sont issus des Présentations de l'OPAC concerné et sont par conséquent directement connectés sur les Abstractions associées"

- *propriétaire*:

procédure Gestion des clients "donné à titre d'exemple"

Évolution de l'état d'une opération

Le graphe des états des opérations est géré automatiquement par chaque opération qui, pour cela, s'inspecte à chaque inférence et change d'état en conséquence. Lorsque l'application démarre, toutes les opérations sont à l'état à initialiser. Puis le moteur infère pour trouver toutes les opérations que l'utilisateur peut déclencher, de même que les opérations qui doivent être déclenchées. Les opérations répondant à ces deux critères passent à l'état inactivable (leurs filles restent à l'état à initialiser). Jusqu'à présent tous les représentants externes des opérations sont également inactivables (boutons et menus grisés, listbox inaccessibles, etc.). Si les pré-conditions sont vérifiées, les opérations passent à l'état activable (leurs filles demeurent à l'état à initialiser), sinon elles restent inactivable. Dès que les opérations passent à l'état activable, les représentants externes deviennent également activables pour l'utilisateur, c'est-à-dire qu'il peut interagir avec eux.
Deux cas se présentent alors :

- soit l'utilisateur déclenche une opération. Ce déclenchement a lieu au gré de l'utilisateur tant que les pré-conditions de l'opération sont vérifiées. Si cette dernière propriété n'est plus vraie, l'opération et ses représentants retournent à l'état inactivable.
- soit l'opération se déclenche par elle-même. Ceci se produit uniquement lorsque l'opération est automatique, donc sans déclencheur. Le contrôleur n'intervient donc pas dans le déclenchement.

Une fois déclenchée[127], l'opération s'exécute (état active), puis se termine ce qui l'amène à l'état terminée. Deux cas se présentent de nouveau :

- soit les contraintes de déclenchement sur l'opération ne sont pas vérifiées auquel cas l'opération retourne à l'état "inactivable (ainsi que ses représentants), et le cycle peut recommencer (test des pré-conditions, déclenchement, etc.).
- soit les contraintes de déclenchement sur l'opération sont vérifiées et l'opération ne peut plus être déclenchée . Elle passe alors à l'état non initialisable ce qui signifie que rien ne permet de la déclencher jusqu'à la prochaine réinitialisation (ses représentants deviennent inactivables et ses filles deviennent non initialisable).

Définition de l'état terminée d'une opération

Une opération active passe à l'état terminée dans les conditions suivantes (dans tous les cas, le traitement associé à l'opération est supposé achevé et la post-condition devra être vraie) :

- opération sans fille : pas de condition supplémentaire,
- opération avec filles : (pas de filles à l'état "active") et (toutes les filles obligatoires sont terminées) et (la contrainte sur les filles est vérifiée).

[127] Le déclenchement d'une opération mère provoque le passage de l'état à initialiser à l'état inactivable pour les filles qui dépendent directement du point d'entrée (cf § 3.5.1).

Chaque fois qu'une opération change d'état, elle en informe ses filles afin qu'elles s'adaptent à ce changement si nécessaire. Ceci est vrai également pour les procédures qui informent leurs opérations et pour les buts qui informent leurs sous-buts ou leurs procédures. Les changements importants sont le passage à :

- l'état active qui provoque le passage à l'état inactivable pour les filles,
- l'état terminée qui provoque la réinitialisation des filles (compteur de déclenchement entre autres) et qui les fait passer à l'état à initialiser),
- l'état non initialisable qui fait passer également les filles à l'état non initialisable.

Une opération gère donc elle-même son état tout en gérant également en partie l'état de ses filles, qui peuvent à leur tour gérer en partie l'état de leurs filles. Ainsi la gestion des états des opérations est récursive et répartie dans les opérations. Le contrôle du dialogue est donc également géré en partie par les opérations.

3.3. Objet procédure

A la suite des spécifications Diane+, notre outil génère des objets Procédures à partir des procédures construites par le concepteur. Un objet Procédure s'identifie par :

- *son nom* (facultatif) saisi par le concepteur pour clarifier éventuellement les spécifications si le nombre de procédures est important,

- *son type* qui prend la valeur minimale, prévue ou effective. Seules les procédures minimales sont nécessaires, mais nous prévoyons également une gestion des procédures prévues, puis des procédures effectives.

- *son état* qui permet à la procédure de s'auto-gérer au même titre que les opérations (voir plus loin).

- *le poste de travail attaché*. Pour une même tâche, la procédure minimale pour une secrétaire ne sera pas la même que celle du comptable ou celle du directeur de la société. Grâce à l'identification de l'utilisateur lors de la connexion, il est possible de choisir la procédure minimale appropriée par l'intermédiaire de cet attribut.

- *la liste des opérations* qui est constituée d'un ensemble de pointeurs (Figure 6.34). Chacun de ces pointeurs est dirigé vers l'opération (en fait une instance de celle-ci) présente dans la procédure.

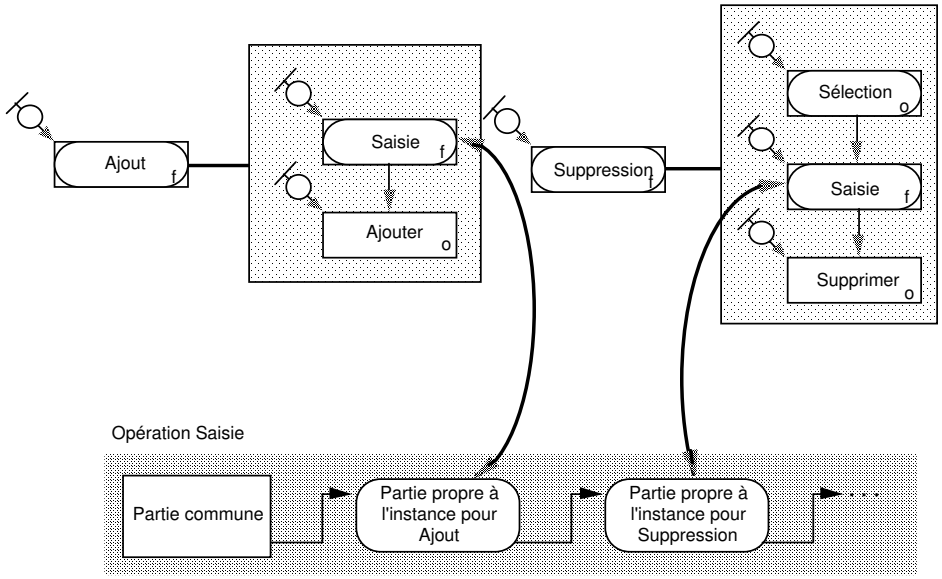


Figure 6.34 : Connexion entre les opérations d'une procédure et leurs instances de la classe Opération

- *les règles d'enchaînement* qui sont du même type que celles présentes dans les opérations. Ces règles portent sur les enchaînements entre les opérations présentes au premier niveau dans la procédure (les sous-niveaux sont traités par les règles d'enchaînement dans les opérations).

- *ses représentants externes.* En général, une procédure fournira une ou plusieurs fenêtres (boîtes de dialogue ou fenêtres classiques). De même que pour les opérations, une procédure et ses représentants se connaissent mutuellement puisque lors de la génération de l'interface, le système crée des liens entre les procédures et leurs représentants externes par l'intermédiaire de pointeurs. Les procédures ne connaissent donc pas directement leurs représentants, néanmoins elles peuvent leur demander de se rendre activables ou inactivables.

- *le but associé*. De même que pour les opérations, il doit être possible de connaître le nom, le type, l'état, etc., du but associé. Pour cela, nous utilisons un pointeur sur l'objet représentant ce but. Grâce à ce pointeur, un but peut inspecter ses procédures pour en déduire un éventuel changement d'état de sa part, tout comme une opération le fait vis-à-vis de ses filles.

Une procédure est capable de s'auto-gérer au même titre qu'une opération (elle fonctionne comme une opération mère obligatoire). Elle utilise pour cela les règles d'enchaînement qu'elle contient et qui portent sur le premier niveau d'opérations qui la composent. La figure 6.35 représente une procédure qui a reçu l'ordre de se rendre active (ceci se produit par exemple lorsque l'utilisateur choisit une commande dans un menu). A cet instant toutes les opérations de la procédure sont supposées être à l'état à initialiser. Après réception du message, la procédure redistribue cet ordre à chaque première opération de chacun de ses blocs ("A, "D", "G", "H" et "I") en leur demandant de passer à l'état inactivable. Son rôle sur les opérations s'arrête ici et les opérations prennent alors le relais ; par exemple "A" étant obligatoire, elle ne transmet pas ce message à ses successeurs contrairement à "D" qui le passe à "E" (à cause de son type facultatif) qui le passe à son tour à "F". La procédure se charge désormais :

- de demander à chacune de ses opérations si elle est terminée ou non,
- de gérer les enchaînements,

- de gérer éventuellement les contraintes sur les filles.

Lorsque toutes les opérations obligatoires, ainsi que les opérations facultatives avec contraintes sont terminées, elle passe à l'état terminée et met à jour ses filles qui à leur tour mettent à jour leurs représentants externes.

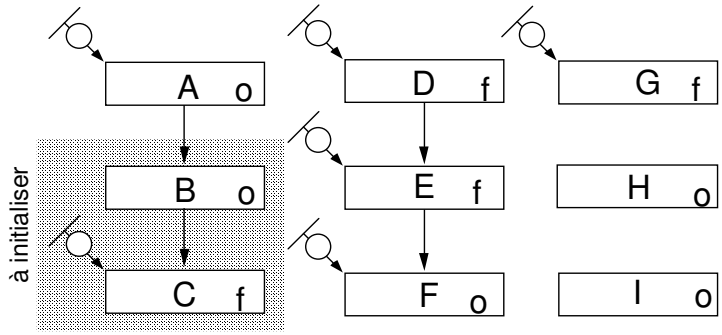


Figure 6.35 : Auto-gestion d'une procédure.
B et C restent à l'état "à initialiser" tant que A n'est pas terminée.

Si les opérations de cette procédure contenaient des opérations filles, le principe de gestion de la procédure reste inchangé puisque les opérations mères prennent en compte la gestion de leurs filles. En résumé, une procédure n'a un droit de regard que sur les opérations de premier niveau (Figure 6.36).

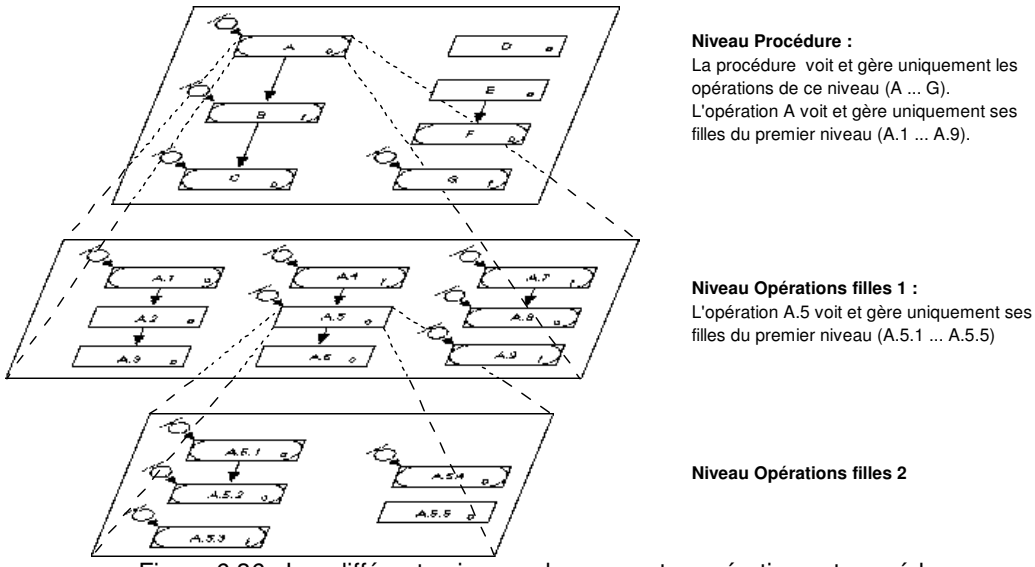


Figure 6.36 : Les différents niveaux de vues entre opérations et procédures

3.4. Objet But

Les buts seront représentés par des objets qui contiennent soit la liste de leurs sous-buts soit la liste des procédures à réaliser ainsi qu'éventuellement des règles d'enchaînement sur ces procédures et ces sous-buts (nous avons vu précédemment que les buts peuvent être décrits avec le formalisme Diane+ et donc qu'ils peuvent générer des règles d'enchaînement). Les buts sont identifiés par :

- *leur nom* qui est saisi par le concepteur et qui est utilisé lors de la génération de l'interface, par exemple pour le nom d'un menu ou d'une fenêtre.

- *leur liste de sous-buts*. Cette liste est facultative puisqu'il est possible de n'avoir qu'un seul but général pour l'application. L'imbrication des sous-buts dans un but suit le même principe que celui des opérations filles dans une opération mère.

- *leur liste de procédures.* A un but terminal (c'est-à-dire qui ne possède pas de sous-but) doit correspondre au moins une procédure (la minimale), mais plusieurs procédures (prévue et effectives) qui sont fonction du poste de travail associé peuvent lui être rattachées.

- *leur état*. Un but suit le même graphe d'état que les opérations et les procédures. En fait l'état d'un but est identique à celui de la procédure qu'il a déclenchée.

- *leur liste de représentants externes*. Nous avons vu précédemment qu'un but produit par défaut un menu ou une commande dans un menu (ceci étant modifiable par le concepteur). Lors de la génération, le système crée donc des liens entre ces représentants et les buts associés. De même que pour les opérations et les procédures, cette liste ne contient que des pointeurs.
- *le but associé*, c'est-à-dire le but où ils apparaissent. Étant donné que la décomposition en sous-buts est facultative, cette variable l'est également. Si elle existe, elle correspond à un pointeur sur l'objet représentant le but père.

3.5. Le moteur d'inférence

Nous avons précisé auparavant que nous n'implémentons pour l'instant que les procédures minimales, cependant le principe des règles que nous allons décrire juste après reste valable quel que soit le type de procédure. Nous commençons par montrer comment sont générées les règles, puis nous montrerons comment elles sont utilisées de manière interne dans la gestion de la dynamique et dans l'aide.

3.5.1. Syntaxe des règles

Parallèlement à la génération de l'interface, notre outil génère les règles d'enchaînement à partir des spécifications Diane+. Rappelons que cette génération se répète à chaque niveau, c'est-à-dire qu'il y a une génération pour les buts, une génération pour les procédures et une génération pour chaque opération qui contient des opérations filles.

La figure 6.37 donne un exemple de procédure. Lors de la génération, le système rajoute à cette procédure un point (ou place) d'entrée ("e") et un point (ou place) de sortie ("s"). La place d'entrée est reliée à chaque opération qui doit être accessible lors de l'entrée dans la procédure. Ces liaisons pointent sur chaque première opération de chaque bloc. Elles regroupent donc des opérations qui doivent être déclenchées automatiquement par le système ("C" et "F") et des opérations qui doivent ou qui peuvent (suivant leur type obligatoire ou facultatif) être déclenchées par l'utilisateur ("A" et "E").

Dans le cas de la figure 6.37, la place "e" nous donne donc les règles suivantes : e $\Rightarrow$ A, e $\Rightarrow$ C, e $\Rightarrow$ E et e $\Rightarrow$ F. Ces règles sont équivalentes à dire : "e" permet de déclencher "A", "e" permet de déclencher "C", "e" permet de déclencher "E" et "e" permet de déclencher "F".

La place "s" est reliée à chaque dernière opération de chaque bloc. Elle permet d'écrire les règles suivantes : B $\Rightarrow$ s, D $\Rightarrow$ s, E $\Rightarrow$ s et F $\Rightarrow$ s. La place "s" est surtout utilisée de manière invisible

pour l'aide, principalement dans le chaînage arrière. Il est important de noter que, dans les règles, nous ne faisons pas de différence sur les opérations facultatives ou obligatoires, de même que les règles ne font pas apparaître les contraintes de déclenchement et les déclencheurs. Ce sont les opérations qui, grâce à leur auto-gestion, aviseront le contrôleur de leur état en fonction de leurs caractéristiques.
Les autres règles générées correspondent aux précédences permanentes entre les opérations[128]. Les règles correspondantes sont donc : A $\Rightarrow$ B, C $\Rightarrow$ D.

Plutôt que d'utiliser des règles élémentaires comme celles énoncées plus haut, nous emploierons la syntaxe suivante pour les règles : (X $\Rightarrow$ x) où X est l'ensemble des opérations qui précèdent directement x, ce qui revient à écrire ($\{x_1, x_2, x_3,...x_n\} \Rightarrow x$). Si l'on reprend les règles précédentes, nous obtenons donc : (e $\Rightarrow$ A), (e $\Rightarrow$ C), (e $\Rightarrow$ E), (e $\Rightarrow$ F), (A $\Rightarrow$ B), (C $\Rightarrow$ D), (B D E F $\Rightarrow$ s).

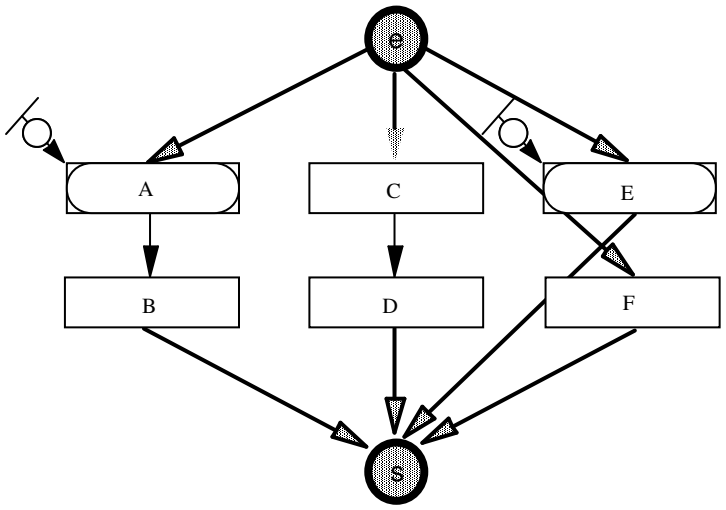


Figure 6.37 : Une procédure Diane+ avec les deux places supplémentaires utilisées pour la génération des règles.

Si certaines des opérations présentes dans cette procédure contiennent des opérations filles, on réitère le même processus pour chacune de ces opérations. La figure 6.38 montre un exemple de décomposition des opérations "A" et "C" de la procédure précédente. Après l'ajout des places "e" et "s", les règles générées sont donc :

- pour "A" : (e $\Rightarrow$A.1), (e $\Rightarrow$A.3), (A.1 $\Rightarrow$A.2), (A.3 $\Rightarrow$A.4), (A.2 A.4 $\Rightarrow$s)
- pour "C" : (e $\Rightarrow$C.1), (e $\Rightarrow$ C.4), (C.1 $\Rightarrow$C.2), (C.2 $\Rightarrow$ C.3), (C.3 C.4 $\Rightarrow$s).

[128] Nous rappelons que nous ne tenons pas compte des précédences indicatives (cf Chapitre 5, § 2.4.2).

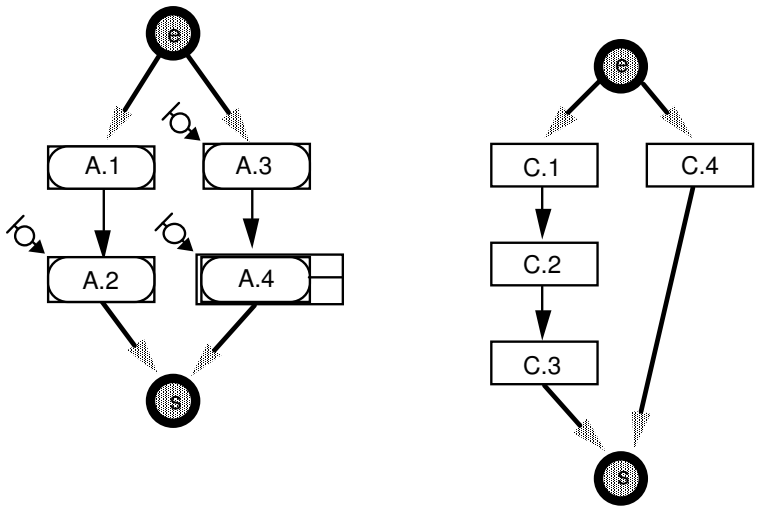


Figure 6.38 : Exemple de décomposition des opérations “A” et “C” de la figure 6.37

3.5.2. Mécanisme d'inférence pour la dynamique

Nous avons vu qu’au démarrage d’une application toutes les opérations sont à initialiser. Lorsqu’on déclenche une procédure, le contrôleur envoie à toutes les opérations α, présentes dans les règles ($e \Rightarrow \alpha$) de cette procédure, un message pour leur dire de passer à l’état inactivable. Ainsi dans la figure 6.37, les opérations “A”, “C”, “E”, “F” sont concernées. Examinons en détail le comportement de chacune d’entre elles lorsqu’elles reçoivent ce message.

- l’opération “A” sait qu’elle possède des opérations filles. Elle transmet alors ce message à toutes ses opérations filles qui dépendent de la place d’entrée locale. Les opérations “A.1” et “A.3” deviennent donc inactivables. Chacune d’elles prend alors sa gestion en main ce qui signifie que si les pré-conditions de “A.1” sont vérifiées[129], “A.1” devient alors activable (idem pour “A.3”). Si cette situation est vérifiée, “A.1” se déclenche alors automatiquement puisqu’elle ne possède pas de déclencheur. Lorsqu’elle est terminée, le contrôleur envoie un message à “A.2” pour lui dire de se rendre inactivable. De même que pour “A.1”, si les pré-conditions sont vérifiées, “A.2” passe à l’état activable. Lorsque “A” est terminée, le contrôleur de dialogue envoie un message à “B” pour qu’elle se rende inactivable. Si ses pré-conditions sont vérifiées, elle se déclenche alors automatiquement.

- l’opération “C” sait également qu’elle possède des filles. Elle suit donc le même processus que “A” et transmet le message à “C.1” et “C.4” qui deviennent alors inactivables. Si leurs pré-conditions sont vérifiées, elles passent à l’état activable puis elles se déclenchent automatiquement (absence de déclenchement utilisateur).
  On notera au passage que les règles tiennent compte des exécutions concurrentes (ici “C.1” et “C.4” s’exécutent en parallèle). Il est bien évident que le système cible devra être capable de gérer ce multi-tâche.
  Lorsque “C.1” s’achève, le contrôleur de dialogue utilise alors la règle ($C.1 \Rightarrow C.2$) et envoie un message à “C.2” pour qu’elle se rende inactivable[130]. De même que pour “C.1” et “C.4”, si les pré-conditions sont vérifiées, “C.2” s’exécute automatiquement. Le processus se

[129] Si les pré-conditions ne sont pas vérifiées, le contrôleur de dialogue vérifie à chaque inférence si un changement s’est produit dans les conditions de déclenchement des opérations.
[130] Ceci ne peut se faire que si l’opération “C2” est à initialiser. Dans la négative, cela signifie que “C2” est déjà en auto-gestion.

poursuit ainsi jusqu'à la fin de l'exécution de "C.3". Lorsque "C.3" et "C.4" sont terminées, "C" sait alors que toutes ses opérations filles sont terminées. Elle passe donc également à l'état terminée. Si elle possédait une contrainte du type "ne peut être déclenchée qu'une et une seule fois", elle passe alors automatiquement à l'état non initialisable.

- l'opération "E" se comporte de la même façon que "A" bien qu'elle ne possède pas de filles, c'est-à-dire qu'elle passe d'abord à l'état inactivable puis activable si ses pré-conditions sont vérifiées.

- l'opération "F" se comporte comme l'opération "C", c'est-à-dire qu'elle se déclenche automatiquement si ses pré-conditions sont vérifiées.

Lorsque toutes les opérations obligatoires sont terminées et que la contrainte sur les filles est vérifiée, le contrôleur de dialogue sait que la procédure associée est terminée et il lui envoie alors un message pour lui dire de passer à l'état terminée (fonctionnement similaire à celui d'une opération mère envers ses filles).

Tout au long de ce processus et à chaque changement d'état des opérations, celles-ci ont automatiquement mis à jour leurs représentations externes. Le contrôleur de dialogue n'intervient donc absolument pas dans la gestion purement externe de l'application. Celle-ci est prise en charge d'une part par les opérations et d'autre part par l'interface elle-même (par exemple un bouton grisé ne peut pas émettre d'événement suite à un clic de l'utilisateur).

En résumé le fonctionnement général du contrôleur de dialogue consiste à inférer perpétuellement sur les règles d'enchaînement. A chaque inférence il détecte toutes les règles qui sont déclenchables en inspectant l'état des opérations qui les composent[131]. Il se contente ensuite d'envoyer un message aux opérations résultantes (c'est-à-dire les opérations qui terminent une règle) pour leur dire de passer à l'état inactivable. Le reste de la gestion est réparti dans les objets représentant les opérations, les procédures et les buts.

3.5.3. Mécanisme d'inférence pour l'aide

Ce mécanisme est très simple. Détaillons les quatre types d'aide :

- si l'utilisateur demande l'aide "Que se passe-t-il si on déclenche α ?", le contrôleur commence par faire une copie de la liste des opérations, des procédures et des buts[132]. Puis il met α à l'état terminée et infère sur les copies des listes comme s'il s'agissait d'un déclenchement normal d'opération. Les inférences produisent des perturbations dans les états permettant par exemple à des opérations de se terminer ou de rendre possible des enchaînements. Pour détecter les conséquences du déclenchement, il suffit alors d'inspecter le nouvel état des opérations, des buts et des procédures et de le comparer à celui d'avant le déclenchement.

- si l'utilisateur demande l'aide "Comment faire pour atteindre α ?", le contrôleur travaille alors en chaînage arrière en partant des règles qui ont pour conséquence l'opération choisie

[131] Dans notre cas, une règle est considérée déclenchable si les opérations prémisses sont terminées et si l'opération à déclencher n'est ni en cours d'exécution ni non initialisable.

[132] Si l'on désire que les conséquences soient visibles au niveau de l'interface, la copie des listes est suffisante. Dans le cas contraire, il suffit d'affecter nil à l'ensemble des représentants externes.

par l'utilisateur[133]. De même que précédemment, en fonction du désir de tenir compte ou non du contexte, on sélectionnera respectivement dans les règles et parmi les prémisses soit l'ensemble complet des opérations soit uniquement les opérations qui sont activables. Cette première inférence nous amène soit aux opérations précédant α, soit à l'opération mère de α. On réitère ce processus avec comme nouvel α le résultat précédemment obtenu, et ceci jusqu'à parvenir au niveau de la procédure, puis du but qui contiennent α. A chaque inférence si le résultat comporte une opération qui contient l'α du départ, alors cette opération doit être terminée ; si le résultat comporte une opération qui ne contient pas l'α du départ, alors cette opération doit être rendue active.

- le troisième cas d'aide ("Pourquoi α est grisé?") revient à dire si les faits suivants sont vrais ou faux :
  - la pré-condition de α n'est pas vérifiée,
  - les opérations obligatoires précédentes ne sont pas terminées,
  - α est facultative et la contrainte de l'opération mère est vérifiée,
  - l'opération mère n'est pas active (la mère peut être également une procédure ou un but),
  - α est non initialisable ou pas encore initialisée.

- "Comment terminer α ?" revient à utiliser les règles d'enchaînement de l'opération α[134] ; par exemple dans le cas de la figure 6.38, on écrira : "pour que A soit terminée, vous devez exécuter 'A.1 puis A.2', 'A.3 puis A.4'". Cette aide tiendra compte d'une part du type des opérations (si A.3 est facultative, on écrira "vous pouvez exécuter A.3" sinon on écrira "vous devez exécuter A.3"), d'autre part de l'état des opérations (si A.1 est déjà terminée, on écrira "vous devez exécuter A.2").

3.6. Bilan de l'implémentation

Une première maquette de l'outil que nous avons spécifié dans ce chapitre a déjà été implémentée en Smalltalk/V sous Windows 3.1. Elle est composée principalement d'un éditeur Diane et du moteur d'inférence précédemment évoqué. Seule la partie concernant la gestion automatique de la dynamique et de l'aide a été implémentée pour l'instant. Nous détaillons brièvement à présent l'éditeur et le moteur.

3.6.1. L'éditeur Diane+

L'éditeur Diane+ est un éditeur textuel (Figure 6.39) qui permet de créer des procédures et des opérations Diane+ par l'intermédiaire d'éditeurs spécialisés (non présentés ici). Il prend en compte l'imbrication des opérations telle qu'elle a été présentée au Chapitre 5. La gestion des contraintes sur les filles est assurée, de même que les contraintes de déclenchement. A chaque opération mère et à chaque procédure peuvent être associées des règles d'enchaînement comme celles qui ont été présentées dans ce chapitre (les règles en rapport avec les places "e" et "s" n'ayant aucune signification pour le concepteur, elles n'apparaissent donc pas dans la liste des règles).

---

[133] Cette recherche se fait ici dans l'ensemble complet des règles d'enchaînement car l'objectif à atteindre n'est pas forcément en rapport avec le travail en cours.

[134] Si l'utilisateur désire en savoir plus, il demande un complément d'aide sur le résultat obtenu. Dans ce cas, on réitère le processus d'explication d'exécution d'opération à partir du résultat.

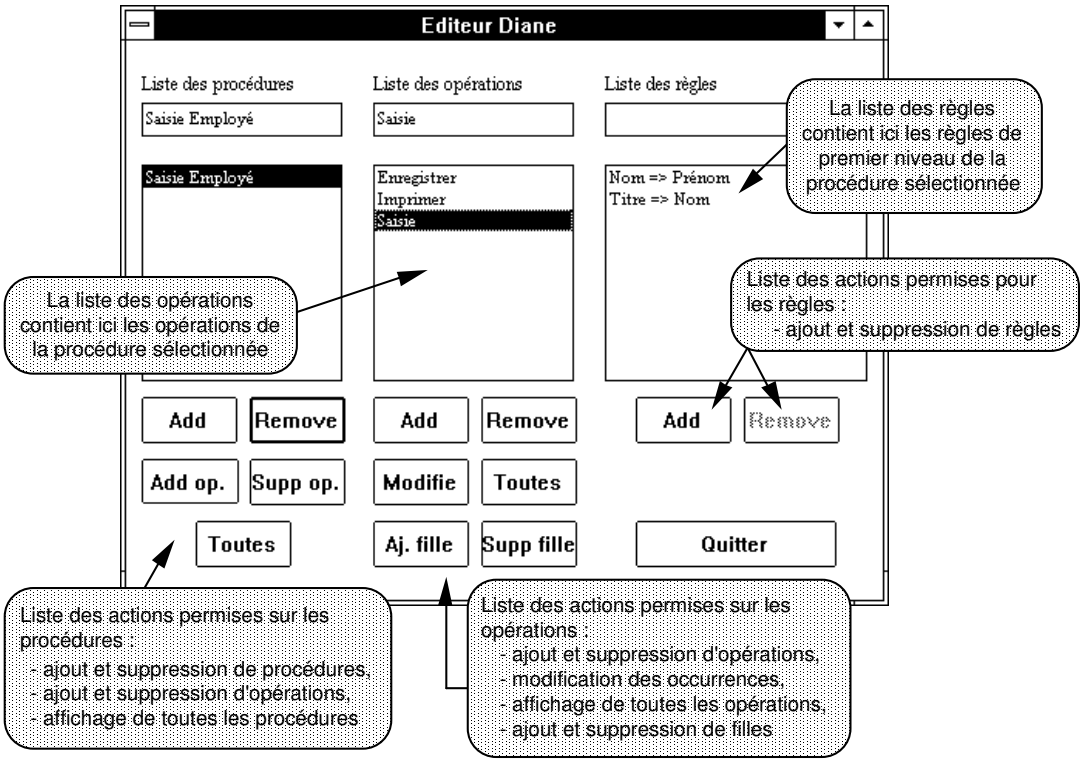


Figure 6.39 : L'éditeur Diane+

La gestion des occurrences d'opérations est également gérée par un éditeur spécialisé (Figure 6.40) qui prend en compte pour l'instant :

- les pré- et les post-conditions (écrites en Smalltalk),
- les pré- et les post-actions (également écrites en Smalltalk),
- le déclencheur,
- le mode,
- le type,
- l'état,
- la contrainte sur les filles,
- le contrainte sur le nombre de déclenchement,
- la liste des représentants externes (uniquement en consultation),
- le propriétaire de l'opération.

Grâce à cet éditeur et à l'éditeur Diane+, il est possible de modifier en temps réel les règles d'enchaînement ainsi que tous les attributs pré-cités exceptés le propriétaire et les représentants externes, ceci afin de visualiser immédiatement les répercussions sur l'application.

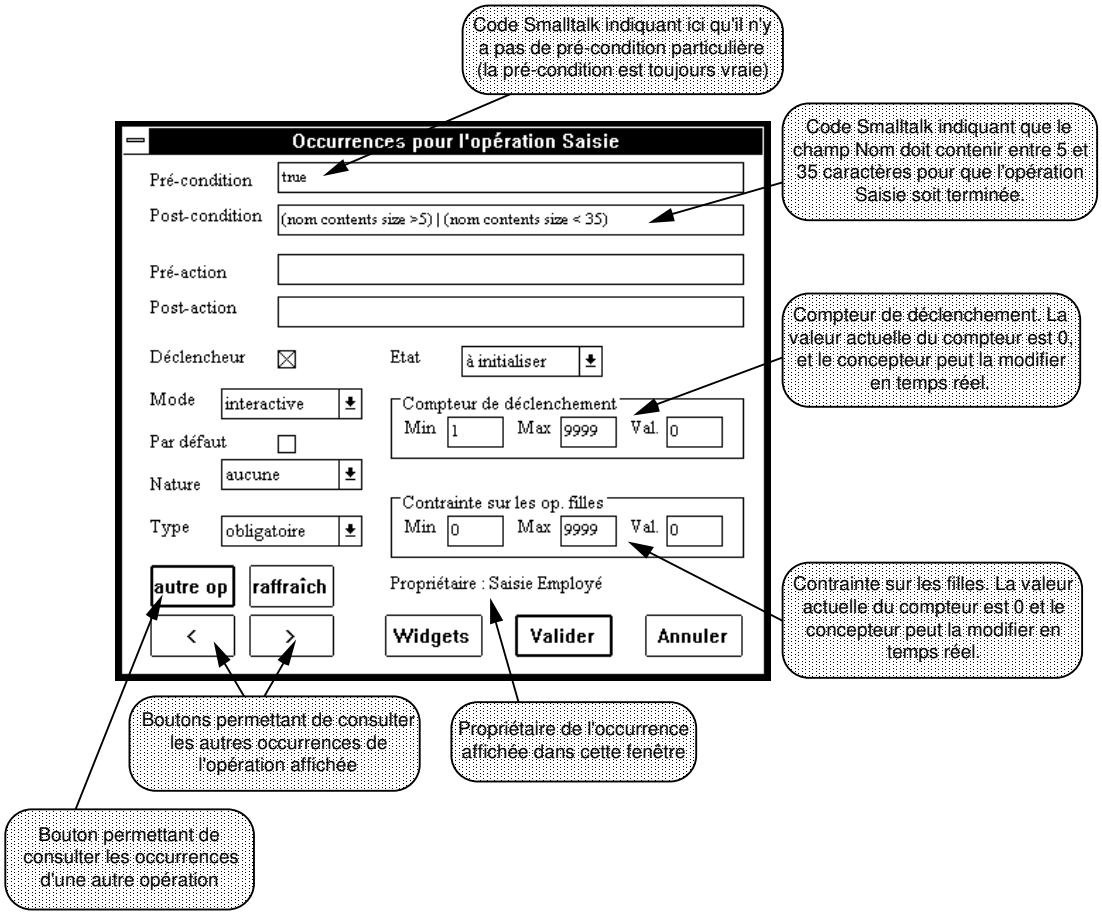


Figure 6.40 : L'éditeur d'occurrences des opérations Diane+

La maquette ne prenant pas en compte la génération de la représentation externe, nous devons pour l'instant construire complètement cette dernière. Ceci est réalisé par l'intermédiaire de Window Builder qui est l'UIMS fonctionnant avec Smalltalk/V. Cet UIMS est composé d'un éditeur de fenêtres et d'un générateur de code Smalltalk (une fenêtre = une classe). Une fois que les écrans sont réalisés, nous connectons les différents éléments interactifs avec les opérations et les procédures Diane+[135]. Le système est alors prêt à fonctionner.

3.6.2. Le moteur d'inférence

Les spécifications étant achevées et les connexions terminées, le moteur est alors capable de gérer automatiquement la dynamique et l'aide. Dans sa version actuelle, le moteur ne prend pas en compte la question "Que se passe-t-il si…?", cette limitation étant dûe uniquement à une limitation de la part de Smalltalk quant à la duplication d'objets complexes. Les trois autres questions sont, quant à elles, gérées de la même façon que celle présentée dans ce chapitre.

Les résultats validés par la maquette sont donc les suivants :

- *gestion automatique de la dynamique* pour la représentation externe et pour la représentation interne (enchaînements, changements d'états, etc.),

[135] Nous avons vu dans ce chapitre que la connexion doit être à double-sens (une opération doit pouvoir mettre à jour sa représentation externe, et une interaction sur un widget doit déclencher une opération ou une procédure).

- *gestion automatique de l'aide* pour les questions "Comment faire pour…?", "Pourquoi… est grisé ?", "Comment terminer…?",

- *cohérence* entre le conceptuel et l'aide ainsi qu'entre le conceptuel et la dynamique puisque tout changement dans les spécifications (règle d'enchaînement, description d'une opération, etc.) est immédiatement pris en compte dans la dynamique et dans l'aide.

4. Conclusion

Nous avons vu dans ce chapitre comment nous pouvons implémenter les concepts Diane+ en nous basant sur une approche objet. Nous avons montré également les avantages de cette approche (par exemple la gestion répartie du dialogue). L'utilisation du modèle PAC influe en partie sur l'implémentation que nous proposons, par exemple en déléguant une partie du dialogue aux OPACs (principalement pour leur représentation externe et la gestion de leur sémantique). D'autres modèles auraient pu être utilisés, par exemple les MCD. Les OPACs nous ont surtout permis de créer des applications dont la couche des données est directement réutilisable en ce qui concerne les traitements élémentaires (création, suppression, recherche...). Elle peut ainsi être utilisée avec d'autres formalismes (par exemple les RPO) ou d'autres méthodes (par exemple MAD). La dernière partie a exposé comment l'utilisation conjointe des objets et des règles d'enchaînement permet la gestion de la dynamique et l'aide des applications réalisées. Certains points ne sont cependant pas encore résolus. Ainsi :

- les spécifications ne peuvent pas prendre en compte toutes les interactions entre les opérations. Par exemple si une opération est mutuellement exclusive avec une opération qui se trouve dans un bloc indépendant[136] (par exemple dans une autre procédure), le concepteur doit spécifier lui-même cette contrainte dans le code des susdites opérations. Il serait intéressant de pouvoir intégrer des contraintes d'exclusion mutuelle plus puissantes dans le formalisme Diane+ (par exemple qu'une opération "A" invalide les opérations "B" et "C", alors que "B" n'invalide que "C").
- la génération de l'interface demeure encore trop rudimentaire. Certes des choix sont dores et déjà prévus (par exemple choisir entre des fenêtres et des boîtes de dialogue, de même qu'entre des boutons et des menus), par ailleurs la plupart des widgets sont issus des OPACs (zone de saisie, list box, etc.). La prochaine étape consistera à intégrer des règles ergonomiques de génération permettant par exemple de choisir automatiquement des boutons radio ou une combo box en fonction du niveau de l'utilisateur, de la place disponible à l'écran ou encore de la donnée à représenter. On peut également envisager d'ajouter des attributs aux opérations ; la fréquence d'utilisation prévue permettrait ainsi de choisir entre un bouton (fréquence élevée) et un menu (fréquence basse). Enfin il serait également intéressant de pouvoir générer l'interface dans un environnement laissé au choix de l'utilisateur (Windows, X-Window...) puisque nombre de produits proposent déjà cette caractéristique.

[136] L'exclusion mutuelle dans un même bloc est directement gérée par le contrôleur de dialogue par l'intermédiaire des contraintes sur les opérations filles (cf Chapitre 5, § 2.4.3).

Conclusion

Nous avons montré dans cette thèse comment, dans le cadre des applications interactives pré-planifiées, la méthode Diane+ permet à la fois de spécifier finement et de gérer automatiquement, après une phase de génération automatique, le dialogue homme-machine.

Outre son aspect orienté-tâche qui permet de "coller" à la représentation des utilisateurs utilisée par les ergonomes, Diane+ a l'avantage de ne pas imposer de formalisme particulier pour son implémentation. Ainsi nous avons montré au Chapitre 5 qu'elle pouvait être utilisée en complément d'autres méthodes dont les méthodes orienté-objet. Cet aspect est intéressant à deux points de vue :

- les méthodes orienté-objet fournissent une décomposition des données à la fois pertinente et efficace. Cependant elles ne sont pas très puissantes pour des traitements faisant intervenir plusieurs objets à la fois (dissolution des traitements dans les méthodes). Diane+ apporte une solution en utilisant une couche de données objet (dotée d'une capacité de gestion élémentaire) qu'elle chapeaute par ses concepts d'opérations, de procédures et de buts, permettant ainsi la création d'applications adaptées à chaque niveau d'utilisateur. Nous conservons de cette façon une séparation franche entre les données et les traitements *complexes* quel que soit le public visé.
- les méthodes orienté-objet sont tournées davantage vers l'implémentation. Elles respectent les critères du génie logiciel et permettent ainsi des implémentations robustes et réutilisables surtout en ce qui concerne les données.

Vis-à-vis de notre domaine de travail, l'apport de cette thèse est double :

- d'une part nous montrons comment *améliorer la conception du dialogue homme-machine*. Nous avons dû pour cela modifier et ajouter certains concepts à Diane (par exemple la nouvelle définition de la précédence permanente et les contraintes entre les opérations mères et les opérations filles). La méthode Diane+ contient, quant à elle, toutes les propriétés nécessaires pour spécifier n'importe quel dialogue homme-machine. Elle permet de couvrir une plage de dialogues allant des plus rigides (latitude décisionnelle nulle) au plus libres (latitude décisionnelle maximale). Cependant l'intérêt de Diane+ se situe principalement dans les dialogues intervenant entre ces deux extrémités. En cela, Diane+ répond à l'absence d'intégration de l'utilisateur présente dans tous les produits de création d'IHM. Avec Diane+, il est possible par exemple de choisir si une opération doit ou non être exécutée ou bien encore qui, entre l'homme et la machine, devra déclencher un traitement.
  Diane+ permet de formaliser de façon concise et lisible l'ensemble des choix proposés à l'utilisateur. Le formalisme employé ne fait apparaître pour cela que les contraintes du dialogue ; d'une part tout ce qui n'est pas explicité est autorisé (par exemple si deux opérations ne sont pas enchaînées, leur séquencement est laissé au bon vouloir de l'utilisateur), d'autre part tout ce qui est spécifié peut être restreint (par exemple une opération facultative peut devenir obligatoire). En cela le formalisme Diane+ est de loin beaucoup plus intéressant que des formalismes tels que celui des automates qui sont sujet à des explosions combinatoires, ce qui n'est pas le cas de Diane+.
  Remarquons pour finir qu'avec Diane+ il est possible de spécifier le dialogue homme-machine à des niveaux de détail dont la granularité peut aller de fort à faible (cf Chapitre 5, § 4).

- d'autre part nous montrons comment *gérer plus facilement le dialogue homme-machine*. Cette gestion se trouve simplifiée par la génération automatique et la gestion automatique de l'interface et de l'aide.

  i. *l'interface* :
  Nous avons montré qu'il est possible de déduire la représentation externe d'une application à partir de ses spécifications Diane+ et que cette représentation externe peut être gérée automatiquement par le contrôleur de dialogue. Diane+ implique donc une forte cohérence entre le conceptuel et le comportemental. Par ailleurs le contrôle de dialogue est réparti entre les différents composants conceptuels de l'application (opérations, procédures, buts) ce qui réduit la charge de travail du moteur d'inférence.

  ii. *l'aide* :
  Grâce à la génération automatique, nous conservons également une forte cohérence entre les spécifications du dialogue et l'aide d'utilisation de l'application. Rappelons que Diane+ est une des rares méthodes à permettre ce type d'aide à l'heure actuelle. Cette aide permet à l'utilisateur de connaître le chemin à parcourir pour aboutir à un but, de donner une explication pour une commande inaccessible ou bien encore de lister les conséquences suite à une éventuelle action de sa part.

Plusieurs suites à ce travail peuvent être envisagées. La première d'entre elles va consister en la réalisation complète de l'outil spécifié dans cette thèse[137]. Nous utiliserons Smalltalk en conjonction avec Window Builder pour l'implémentation. Le premier nous fournit un environnement complet de programmation objet, le second permet le prototypage à partir d'objets Smalltalk (le code généré des fenêtres donné dans cette thèse est issu de cet outil).
La seconde étape devra permettre de renforcer les liens entre les données et les traitements. Pour l'instant nous supposons que les OPACs sont à même de gérer des messages de création ou de suppression envoyés par les opérations, mais c'est le développeur qui doit écrire ces messages. Avec des liens plus complets du type de ceux de Merise/2 (création, suppression, consultation, etc.), ces écritures de messages seront raccourcies. Il suffira de tirer des liens prédéfinis entre les opérations et les attributs présents dans les vues externes des OPACs pour obtenir l'écriture du message correspondant (par exemple Client.nom consultation).
Une autre suite envisagée pour ce travail est la connexion de Diane+ sur les Objets Coopératifs Interactifs [PALANQUE 92]. Ces derniers fournissent des ensembles de données et de traitements d'une grande rigueur et pouvant être validés formellement. L'apport de Diane+ consisterait à ajouter la dimension d'adaptabilité et la concision qui font défaut à ces ensembles.
Enfin, à plus long terme, on peut envisager :

- une gestion multi-utilisateurs, Diane+ ayant été conçue à l'origine pour des applications mono-utilisateur,
- l'intégration de la couche Stratégie évoquée au chapitre 5 (§ 5.2.3) permettant ainsi de disposer à la fois d'interfaces adaptées (grâce aux différentes procédures Diane+) et adaptables (grâce au choix de stratégies),
- une évaluation a priori et a posteriori, ainsi qu'une validation formelle des spécifications,
- l'intégration de Diane+ dans les nouvelles interfaces (monde virtuel, interfaces multi-média et multi-modales, etc.). Dans le cas du multi-média par exemple, des traitements peuvent être déclenchés par un ordre vocal et non plus par un clic sur un bouton (Windows intègre déjà cette possibilité [WINDOWS 93]). Le concept du déclencheur reste donc le même mais il est étendu puisqu'on lui ajoute un type (vocal, gestuel, etc.).

[137] Pour jouir pleinement de l'apport de Diane+, nous devrons tout d'abord approfondir la vérification automatique de la syntaxe des opération et des procédures (cf Chapitre 5, § 2.4.2).

- la portabilité de l'IHM résultante par l'intermédiaire de bibliothèques d'objets graphiques permettant de choisir l'environnement destinataire.

Bibliographie

[ADREIT 89a] ADREIT Françoise, MAURAN Philippe, RICHY Hélène ; Les Boîtes Actives : spécification d'une interface interactive d'application ; Colloque sur l'ingénierie des interfaces homme-machine - Sophia Antipolis - Mai 1989

[ADREIT 89b] ADREIT Françoise ; Contribution à la spécification d'interfaces homme-SGBD ; Thèse de 3ème cycle informatique de l'Université des Sciences et Techniques du Languedoc - Montpellier - Novembre 1989

[ALEXANDER 87] ALEXANDER H. ; Formally-Based Tools and Techniques of Human-Computer Dialogues ; Ellis Horwod Limited - Chichester - 1987

[ALTY 89] ALTY J.L, MULLIN J. ; Dialogue Specification in the Gradient-Dialogue System ; People and Computers V - Vol. 5, N° 8 - pp. 151-169 - Septembre 1989

[APPLE 87] Apple Computer, Inc. ; Human Interface Guidelines: The Apple Desktop Interface ; Reading, MA: Addison-Wesley - 1987

[APPLE 88] Hypercard Script Language Guide: The Hypertalk Language ; Addison Wesley - New York - 1987

[APPLE 89] MacApp 2.0 Reference Manual ; Cupertino - 1989

[ARCH 92] The UIMS Developers Workshop - A Metamodel for the Runtime Architecture of An Interactive System ; SIGCHI bulletin, Vol. 24, N° 1, pp. 32-37 - 1992

[BAILEY 83] BAILEY J., PEARSON ; Development of a Tool for Measuring and Analysing Computer User Satisfaction ; Management Science, Vol. 29, N° 5 - pp. 54-67 - 1983

[BAILIN 89] BAILIN S. C. ; An Object-Oriented Requirements Specification Method ; CACM, Vol. 32, N° 5 - pp 608-623 - Mai 1989

[BALBO 92a] BALBO Sandrine, COUTAZ Joëlle ; Un pas vers l'évaluation automatique des interfaces homme-machine ; ERGO IA'92 - Biarritz - 7-9 octobre 92

[BALBO 92b] BALBO Sandrine, COUTAZ Joëlle ; Une taxinomie pour les méthodes d'évaluation des interfaces homme-machine ; IHM'92 - Quatrièmes Journées sur l'Ingénierie des Interfaces Homme-Machine - pp. 56-65 - Paris - 30 novembre-2 décembre 1992

[BANNON 83] BANNON L., CYPHER Allen, GREENSPAN S., MONTY M. ; Evaluation and Analysis of User's Activity Organization ; CHI'83 - pp. 54-57 - 1983

[BARTH 86] BARTH Paul S. ; An Object-Oriented Approach to Graphical Interfaces ; ACM Transactions on Graphics - Vol. 5, N° 2 - pp. 142-172 - April 1986

[BARTHET 86] BARTHET Marie-France ; Conception d'applications conversationnelles adaptées à l'utilisateur ; Thèse de doctorat en informatique N° 108 - Institut National Polytechnique - Toulouse - Mars 1986

[BARTHET 88] BARTHET Marie-France ; Logiciels interactifs et ergonomie : modèles et méthodes de conception ; DUNOD - 1988

[BARTHET 91] BARTHET Marie-France, PALANQUE Philippe, ALQUIER Anne-Marie ; L'ergonomie dans les applications MERISE : un complément à la méthode ; Autour et à l'Entour de MERISE - Sophia Antipolis - Mars 1991

[BARTHET 92] BARTHET Marie-France, TARBY Jean-Claude ; Spécifier, concevoir et réaliser une interface graphique ; Rapport interne - D.T.R.N - Toulouse - Janvier 92

[BASS 90] BASS Len, CLAPPPER Brian, HARDY Erik, KAZMAN Rick, SEACORD Robert ; Serpent: A User Interface Environment ; USENIX Conference - pp. 245-259 - Washington - 22-26 January 1990

[BASS 91a] BASS Len, LITTLE Reed, PELLEGRINO Robert, REED Scott, SEACORD Robert, SHEPPARD Sylvia, SZEZUR Martha R. ; The ARCH Model: SEEHEIM Revisited ; User Interface Developers' Workshop - April 1991

[BASS 91b] BASS Len, COUTAZ Joëlle ; Developing Software for the User Interface - Addison-Wesley Publishing - 1991

[BASTIDE 90] BASTIDE Rémi, PALANQUE Philippe ; Petri Nets with Objects for the Design, Validation and Prototyping of User-Driven Interfaces ; INTERACT'90 - pp. 625-631 - Elsevier Science Publisher - IFIP - August 1990

[BASTIDE 91] BASTIDE Rémi ; Modelling a Flexible Manufacturing System by Means of Cooperative Objects ; CAPE'91 - Elsevier Science Publisher - IFIP - Septembre 1991

[BASTIDE 92] BASTIDE Rémi ; Cooperative Objects: A Formalism for Modelling Concurrent Systems ; Thèse de 3ème cycle informatique - Université Toulouse I - Février 1992

[BEAUDOUIN 91] BEAUDOUIN-LAFON M. ; Interfaces homme-machine : vue d'ensemble et perspectives ; Revue Génie Logiciel et Systèmes Experts, N° 24 - pp. 4-16 - Septembre 1991

[BENBASAT 82] BENBASAT I., WAND Y. ; A Structured Approach to Designing Human Computer Dialogues ; Working paper N° 835 - A Dialogue Generator and its Use in DSS Design - Inf. Manage., 5 - pp. 231-241 - october 1982

[BENCI 76] BENCI G., BODART F., CABANNES A. ; Concepts for the design conceptual schema ; IFIP Conference: Freudenstadt ; North Holland Publishers - 1976

[BENYON 90] BENYON David, MURRAY Dianne, JENNINGS Frances ; An Adaptative System Developper's Toolkit ; INTERACT'90 - pp. 573-577 - IFIP - 1990

[BERARD 90] BERARD Christian, N'GUYEN Philippe ; Méthodes des scénarios : analyse et conception des interfaces homme-machine ; CIM'90 - pp. 383-392 - BORDEAUX - 12-14 juin 1990

[BERRADA 92] BERRADA TAZI Mohamed ; Hyperstation : un système d'intégration d'outils et de génération d'interfaces utilisateur flexibles. Conception et réalisation ; Thèse de 3ème cycle informatique - Université de Montpellier II - 1992

[BETTS 87] BETTS B., BURLINGAME D., FISHER G., FOLEY James D., GREEN Mark, KASIK D. J., KERR S., OLSEN Dan R., THOMAS J. ; Goals and Objective for User Interface Software ; Computer Graphics, Vol. 21, N° 2 - 1987

[BODART 89a] BODART F., PIGNEUR Yves ; Conception assistée des systèmes d'information ; méthode ; modèles ; outils (deuxième édition) ; Masson - Paris - 1989

[BODART 89b] BODART F., PROVOT I. ; Éléments de modélisation et d'architecture des IHM dans les SI - Une approche pragmatique ; Rencontres Lausannoises sur les Interfaces Homme-Machine - Troisième cycle romand d'informatique - Lausanne - Septembre 1989

[BOOCH 91] BOOCH Grady ; Object Oriented Design ; Addison Wesley - 1991

[BORNING 79] BORNING A. H. ; ThingLab - A Constraint-Oriented Simulation Laboratory ; Report SSL-79-3 - Xerox Parc - Palo Alto - July 1979

[BORNING 86a] BORNING A. H. ; Defining Constraints Graphically ; CHI'86 - ACM, New York - pp. 137-143 - 1986

[BORNING 86b] BORNING A. H., DUISBERG R. ; Constraint-Based Tools for Building User Interfaces ; ACM Transactions on Graphics, Vol. 5, N° 4 - pp. 345-374 - October 1986

[BORRAS 88] BORRAS P., CLEMENT Dominique ; CENTAUR: The System ; SIGSOFT'88 - pp. 14-24 - Boston - 1988

[BOS 92] BOS Edwin ; Some Virtues and Limitations of Action Inferring Interfaces ; UIST'92, pp. 79-88 - Monterey, California - 15-18 novembre 1992

[BOUSSE 89] BOUSSE Marc ; GUISE : un générateur d'interfaces utilisateurs permettant la manipulation de structures de données ; Colloque sur l'ingénierie des interfaces homme-machine - pp. 57-70 - Sophia Antipolis - 24-26 mai 1989

[BOUSSE 91] BOUSSE Marc ; Étude et réalisation d'un outil de production d'interfaces utilisateurs : le système GUISE ; Thèse de 3ème cycle informatique N° 585 - Université de Rennes I - Janvier 1991

[BOUSSINOT 91] BOUSSINOT Frédéric, DE SIMONE Robert ; The ESTEREL Language ; Rapport de recherche INRIA N° 1487 - Juillet 1991

[BOY 89] BOY Guy A., CAMINEL T. ; Situation Pattern Acquisition Improves the Control of Complex Dynamic Systems ; 3d European Workshop on Knowledge Acquisition for Knowledge-Based Systems - Paris - Juillet 1989

[BOY 91a] BOY Guy A. ; Computer Integrated Documentation ; NASA Technical Memorandum 103870 - septembre 1991

[BOY 91b] BOY Guy A. ; Indexing Hypertext Documents in Context ; Hypertext'91 Proceedings, pp. 1-11 - décembre 1991

[BRAJNIK 86] BRAJNIK Giorgio, GUIDA Giovanni, TASSO Carlo ; An Expert Interface for Effective Man-Machine Interaction - in Cooperatives Interfaces to Information Systems - pp. 259-323 - Springer-Verlag - 1986

[BRAMS 83] BRAMS G. W. (nom collectif) ; Réseaux de Petri : théorie et pratique. Tome 1 : théorie et analyse ; Tome 2 : modélisation et applications ; Masson - Paris - 1983

[BRES 90] BRES Paul-André, CARRERE Jean-Paul ; Validation d'un système d'information - Un modèle de comportement d'objets naturels ; Autour et à l'entour de Merise - pp. 239-255 - Sophia Antipolis - Septembre 1990

[BRES 92] BRES Paul-André ; Le modèle des objets naturels - Une méthode de conception orientée objet ; Forum logiciel - Août 1992

[BRISSON 90] BRISSON G., DROUIN A., FAVEAUX L. ; Le maquettage dans une démarche ergonomique de conception d'interfaces utilisateur. ; ERGO.IA'90, AFCET SELF IDLS - pp. 44-63 - Biarritz 1990

[BRUNET 91] BRUNET E. ; KADS : méthode d'ingénierie de la connaissance ; Genie Logiciel et Systemes Experts -N° 23 - Juin 1991

[BUXTON 83] BUXTON W. A. S. ; Toward a Comprehensive User Interface Management ; Computer Graphics, Vol. 17, N° 3 - pp. 35-42 - July 1983

[CARDELLI 85] CARDELLI L., PIKE R. ; Squeak : A Language for with Mice ; SIGGRAPH'85, Vol.19, N°3 - pp. 199-204 - July 1985

[CARDELLI 87] CARDELLI L. ; Building User interfaces by Direct Manipulation - Research report 22 - Digital Equipment Corporation Systems - Palo Alto - 1987

[CAREY 82] CAREY T. ; User Difference in Interface Design ; Computer - pp. 14-20 - 1982

[CARLSON 77] CARLSON E., GRACE B., SUTTON J. ; Case Studies of End User Requirements for Interactive Problem-Solving Systems ; MIS Q. - pp. 51-63 - March 1977

[CASTELLANI 91] CASTELLANI Xavier ; L'implémentation de concepts fondamentaux du modèle de la méthode MCO d'analyse et de conception des systèmes d'objets ; Le Génie Logiciel et ses Applications - pp 743-759 - Toulouse - 9-13 décembre 1991

[CHAKRAVARTY 89] CHAKRAVARTY I., KLEYN M. ; Visualisms for Describing Interactive Systems ; 4th IFIP Working Conference on User Interfaces - Nappa Valley - August 1989

[CHATTY 91] CHATTY Stéphane ; Un modèle pour définir le comportement dynamique d'une interface ; IHM'91 - Troisièmes Journées sur l'Ingénierie des Interfaces Homme-Machine - Dourdan (Essonne) - Décembre 1991

[CLEMENT 90] CLEMENT Dominique ; A Distributed Architecture for Programming Environments ; Rapport de recherche INRIA N° 1266 - Juillet 1990

[COAD 90] COAD P, YOURDON E ; Object-Oriented Analysis ; Yourdon Press - Prentice-Hall - 1990

[COLBERT 89] COLBERT E. ; The Object-Oriented Software Development Method: A Practical Approach to Object-Oriented Development ; TRI-ADA 89 - octobre 1989

[COLLET 87] COLLET C. ; Les formulaires complexes dans les bases de données multimédia ; Thèse de 3ème cycle - Université Scientifique Technologique et Médicale de Grenoble - Grenoble - 1987

[COUTAZ 87] COUTAZ Joëlle ; The Construction of User Interfaces and the Object Paradigm ; ECOOP'87 - pp. 135-144 - June 1987

[COUTAZ 88] COUTAZ Joëlle ; Interface homme-ordinateur : conception et réalisation ; Thèse de 3ème cycle informatique - Université Joseph Fourier - Grenoble I - Décembre 1988

[COUTAZ 90] COUTAZ Joëlle ; Interface homme-ordinateur : conception et réalisation ; Dunod Informatique - Avril 1990

[COUTAZ 91a] COUTAZ Joëlle ; Interfaces Homme-Machine : un regard critique ; TSI - Vol. 10, N° 1 - 1991

[COUTAZ 91b] COUTAZ Joëlle, NIGAY Laurence ; Seeheim et architecture multi-agent ; IHM'91 - Troisièmes journées sur l'Ingénierie des Interfaces Homme-Machine - pp. 89-98 - Dourdan (Essonne) - Décembre 1991

[COUTAZ 91c] COUTAZ Joëlle, KARSENTY Solange ; Architecture des systèmes interactifs ; IHM'91 - Troisièmes journées sur l'Ingénierie des Interfaces Homme-Machine - Production des participants en ateliers - Dourdan (Essonne) - Décembre 1991

[COUTAZ 91d] COUTAZ Joëlle, BASS Len ; Developing Software for the User Interface - Addison-Wesley Publishing - 1991

[COUTAZ 93] COUTAZ Joëlle ; Software Architecture Modeling for User Interfaces ; to appear in Encyclopedia of Software Engineering - Wiley & Son - 1993

[CROCHART 88] CROCHART Karl, DAMAY Jeanne, LEPROVOST Jean-François, MORVAN Michel, WALFARD Dominique ; Ergolab : un atelier de spécification et de maquettage d'interfaces usager ; Le Génie Logiciel et ses Applications - Toulouse - 5-8 décembre 1988

[CUA 89] IBM ; Common User Access Advanced Interface Design Guide ; June 1989

[CYPHER 91] CYPHER Allen ; EAGER: Programming Repetitive Tasks by Example ; CHI'91 - pp. 33-39 - New Orleans - 27 avril-2 mai 1991

[DE CHAMPEAUX 91] DE CHAMPEAUX Dennis ; Object-Oriented Analysis and Top-Down Software Development ; ECOOP' 91 - pp. 360-376 - July 91

[DEWAN 87] DEWAN Prasun, SOLOMON Marvin ; DOST: An Environment to Support Automatic Generation of User Interfaces ; SIGPLAN Notices - Vol. 2, N° 1 - pp. 150-159 - Janvier 1987

[DIENG 88] DIENG R., TROUSSE B. ; 3DKAT, A Dependency-Driven Dynamic-Knowledge Acquisition Tool ; 3th Symposium on Knowledge Engineering - Madrid - 1988

[DOMENJOZ 90] DOMENJOZ Luc, GRIZE François, MELLIER Pierre ; CONIAC : un constructeur d'interfaces permettant l'acquisition de données avec vérifications syntaxiques et sémantiques ; ERGO.IA'90, AFCET SELF IDLS - pp. 12-19 - Biarritz 1990

[DUVAL 91] DUVAL Thierry ; Génération automatique d'interfaces utilisateurs pour des applications conversationnelles ; IHM'91 - Troisièmes Journées sur l'Ingénierie des Interfaces Homme-Machine - pp. 129-134 - Dourdan (Essonne) - Décembre 1991

[EL MRABET 91] EL MRABET Hamid ; Outils de génération d'interfaces : état de l'art et classification ; Rapport technique INRIA N°126 - Février 1991

[ENVY 90] OBJECT TECHNOLOGIE International - Distribué par TAU CETI, 264 rue du Faubourg Saint-Honoré, 7508 Paris - avril 90

[FALZON 89] FALZON Pierre ; Ergonomie cognitive du dialogue ; Sciences et Technologies de la Connaissance - Presses Universitaires de Grenoble - 1989

[FEKETE 91] FEKETE Jean-Daniel ; Un modèle abstrait pour la construction d'éditeurs graphiques interactifs ; IHM'91 - Troisièmes Journées sur l'Ingénierie des Interfaces Homme-Machine - pp. 135-139 - Dourdan (Essonne) - Décembre 1991

[FISHER 92] FISHER Gene L., BUSSE Dale E., WOLBER David A. ; Adding Rule-Based Reasoning to a Demonstrational Interface Builder ; UIST'92, pp. 89-97 - Monterey, California - 15-18 novembre 1992

[FLECCHIA 87] FLECCHIA Mark A., BERGERON R. Daniel ; Specifying Complex Dialogs in ALGAE ; SIGCHI + GI'87 - pp. 229-234 - Toronto - 5-9 april 1987

[FOLEY 84] FOLEY James D., VAN DAM A. ; Fundamentals of Interactive Computer Graphics ; Addison Wesley publishing company - 1984

[FOLEY 87] FOLEY James D., KIM Won Chul, GIBBS Christina A. ; Algorithms to Transform the Formal Specification of User-Computer Interface ; INTERACT'87 - pp. 1001-1006 - IFIP - 1987

[FOLEY 88] FOLEY James D., GIBBS Christina A., KIM Won Chul, KOVACEVIC Srdjan ; A knowledge-based user interface management system - CHI'88 - pp. 67-72 - ACM - 1988

[FOLEY 89] FOLEY James, KIM Won Chul, KOVACEVIC Srdjan, MURRAY Kevin ; Defining Interfaces at a High Level of Abstraction ; IEEE Software - pp. 25-32 - January 1989

[FOLEY 90a] FOLEY James D., KIM Won Chul ; DON: User Interface Presentation Design Assistant ; UIST'90 - pp. 10-20 - octobre 1990

[FOLEY 90b] FOLEY James D., SUKAVIRIYA Piyawadee ; Coupling a UI Framework with Automatic Generation of Context-Sensitive Animated Help ; SIGGRAPH'90 - Snowbird, Utah - pp. 152-166 - octobre 1990

[FOLEY 91] FOLEY James, KIM Won Chul, KOVACEVIC Srdjan, MURRAY Kevin ; UIDE: An Intelligent Design Environment - in Architectures for Intelligent Interfaces, J. SULLIVAN and S. TYLER - Addison-Wesley - 1991

[FOLEY 92] FOLEY James D., de BAAR Dennis J. M. J., MULLET Kevin E. - Coupling Application Design and User Interface Design - CHI'92 - pp. 259-266 - 3-7 may 1992

[FOLEY 93a] FOLEY James D., KIM Won Chul ; Providing High-Level Control and Expert Assistance in the User Interface Presentation Design ; INTERCHI'93, pp. 430-437 - Amsterdam - 24-29 avril 1993

[FOLEY 93b] FOLEY James D., SUKAVIRIYA Piyawadee ; A Second Generation User Interface Design Environment: The Model and The Runtime Architecture ; INTERCHI'93, pp. 375-382 - Amsterdam - 24-29 avril 1993

[FRANCHI 89] FRANCHI-ZANETTACCI Paul ; Attribute Specifications for Graphical Interface Generation ; IFIP - 11th World Computer Congress - pp. 149-155 - San Francisco - 1989

[GIBSON 90] GIBSON E. ; Objects. Born and Bred ; BYTE - pp 245-254 - octobre 1990

[GIESKENS 92] GIESKENS Daniel F., FOLEY James D. ; Controlling User Interface Objects through Pre- and Postconditions ; CHI'92 - pp. 189-194 - 3-7 may 1992

[GOLDBERG 84] GOLDBERG A. ; Smalltalk-80. Volume 1 : the interactive programming environment ; Tome 2 : the language and its implementation ; Addison Wesley Publishing Company - 1984

[GOLDSTEIN 82] GOLDSTEIN I. P. ; Intelligent Tutoring Systems. The genetic Graph: A Representation for the Evaluation of Procedural Knowledge ; Academic Pres - London - 1982

[GREEN 84] GREEN Mark ; Report on Dialogue Specification Tools ; Computer Graphic Forum 3 - 1984

[GREEN 85] GREEN Mark ; The Design of Graphical User Interfaces ; Ph. D. Thesis - CSRI-170-85, Computer Systems Research Institute - University of Toronto - 1985

[GREEN 86] GREEN Mark ; A Survey of Three Dialogue Models ; ACM Transactions on Graphics, Vol. 5, N° 3 - pp. 244-275 - July 1986

[GRØNBÆK 91] GRØNBÆK Kaj, HVIID Anette, TRIGG Randall H. ; APPLBUILDER - An Object-Oriented Application Generator Supporting Rapid Prototyping ; Le Génie Logiciel et ses Applications - pp 257-272 - Toulouse - 9-13 décembre 1991

[GUEST 82] GUEST S. P. ; The Use of Software Tools for Dialogue Design ; International Journal of Man-Machine Studies, 16 - pp. 263-285 - 1982

[GUSTAFSON 82] GUSTAFSON M. R., KARLSON T., BUBENKO J. ; A declarative approach to conceptual modeling - IFIP - 1982

[HAEBERLI 88] HAEBERLI Paul E. ; ConMan: A Visual Programming Language for Interactive Graphics ; Computer Graphics, Vol. 22, N° 4 - pp. 103-111 - August 1988

[HAGIYA 90] HAGIYA Masami, OHTANI Kouji ; Parallel Object-Oriented UIMS with Macro and Micro Stubs ; USENIX Conference - pp. 259-274 - Washington - 22-26 January 1990

[HAREL 88] HAREL D. ; On Visual Formalisms ; Communications of the ACM, Vol. 31, N° 5 - pp. 514-530 - 1988

[HAREL 90] HAREL D. ; Statemate : A Working Environment for the Development of Complex Reactive Systems ; IEEE Transaction on Software Engineering, Vol. 16, N° 4 - pp. 403-413- Avril 1990

[HARRIS 90] HARRIS Larry R. ; User Interfaces for Inference-Based Programs ; AI Expert - pp. 42-48 - Octobre 1990

[HARTSON 89] HARTSON H. Rex, HIX Deborah ; Human-Computer Interface Development: Concepts and Systems for its Management ; ACM Computing Surveys, Vol. 21, N° 1 - pp. 5-92 - March 1989

[HARTSON 90] HARTSON H. Rex, HIX Deborah ; Developing Human-Computer Interface Models and Representation Techniques ; Software - Practice and Experience, Vol. 20, N° 5, pp. 425-457 - May 1990

[HARTSON 92] HARTSON H. Rex, GRAY Philip D. ; Temporal Aspects of Tasks in the User Action Notation ; Human-Computer Interaction, Vol. 7, pp. 1-45 - 1992

[HARVEY 88] HARVEY G. ; Understanding Hypercard for version 1.1 ; Editions Sybex - 1988

[HASHIMOTO 92] HASHIMOTO Osamu, MYERS Brad A. ; Graphical Styles for Building User Interfaces by Demonstration ; UIST'92, pp. 117-124 - Monterey, California - 15-18 novembre 1992

[HAYES 85] HAYES P. J., SZEKELY P. A., LERNER R. A. ; Design alternatives for user interface management system based on experience with COUSIN - CHI'85 - pp. 169-175 - San Francisco - April 1985

[HEITZ 87] HEITZ M. ; HOOD, une méthode de conception hiérarchisée orientée objets pour le développement des gros logiciels techniques et temps réels ; Bigre, N° 57 - Journées ADA France : le parallélisme en ADA - Décembre 87

[HENDERSON 86] HENDERSON D.A. ; The Trillium User Interface Design Environment ; SIGCHI'86 - pp. 221-227 - April 1986

[HERRMANN 89] HERRMANN M., HILL Ralph. D. ; Abstraction and declarativeness in user interface development: the methodological basis of the Composite Object Architecture - IFIP'89 - Elsevier Publishers - 1989

[HICKMANN 89] HICKMANN F. R., KILLIN J. L., LAND L., PORTER D., TAYLOR R. M. ; Analysis for Knowledge Based Systems. A Practical Guide to the KADS Methodology ; Ellis Horwood, publishers-Chichester - 1989

[HILL 86] HILL Ralph D. ; Supporting Concurrency Communication and Synchronization in Human Computer Interaction - The Sassafras UIMS ; ACM Transactions on Graphics - Vol. 5, N° 3 - pp. 179-210 - July 1986

[HILL 87a] HILL Ralph D. ; Some Important Features and Issues in User Interface Management System ; Computer Graphics, Vol. 21, N° 2 - April 1987

[HILL 87b] HILL Ralph D. ; Event-Response Systems - A Technique for Specifing Multithreaded Dialogues ; SIGCHI+GI'87 - Human Factors in Computing Systems - pp. 241-248 - Toronto - 5-9 april 1987

[HILL 92] HILL Ralph D. ; The Abstraction-Link-View paradigm: using constraints to connect user interfaces to applications ; CHI'92 - pp. 335-341 - 3-7 may 1992

[HOOD 89] EUROPEAN SPACE AGENCY ; HOOD Reference manual, issue 3.0 ; Ref. WME/89-173/JB - September 1989

[HUDSON 87] HUDSON S. E. ; UIMS Support for Direct Manipulation Intefaces ; Computer Graphics, Vol. 21, N° 2 - pp. 120-124 - April 87

[HUTCHINS 86] HUTCHINS E. L., HOLLAN J. D., NORMAN D. A. ; Direct Manipulation Interfaces ; User Centered System Design ; Lawrence Erlbaum Associates, publishers - pp. 87-124 - 1986

[IBM 89] IBM - Common User Access Advanced Interface Design Guide - june 1989

[ILOG] Aïda/Masaï - manuel de référence. ILOG, 9 rue Royale 75008 Paris

[JACOB 85] JACOB R. J. K. ; A State Transition Diagram Language for Visual Programming ; IEEE Computer, Vol. 18, N° 8 - pp. 51-59 - August 1985

[JACOBSON 89] JACOBSON I. ; The Industrial Development of Software Using an Object-Oriented Technique ; Revue Objective Systems SF AB - 1989

[JANSSEN 93] JANSSEN Christian, WEISBECKER Anette, ZIEGLER Jürgen ; Generating User Interfaces from Data Models and Dialogue Net Specifications - Interchi'93 - pp. 418-423 - Amsterdam - 24-29 april 1993

[JOHNSON 92] JOHNSON Jeff ; Selectors: Going beyond user-interface widgets ; CHI'92 - pp. 273-279 - 3-7 may 1992

[KARSENTY 87] KARSENTY Solange ; Graffiti : un outil interactif et graphique pour la construction d'interfaces homme-machine adaptables ; Thèse de 3ème cycle informatique de l'Université Paris-Sud Centre d'Orsay - Décembre 1987

[KARSENTY 91] KARSENTY Solange, WEIKART Chris ; Une extension de l'interface d'application dans le modèle de Seeheim ; IHM'91 - Troisièmes journées sur l'Ingénierie des Interfaces Homme-Machine - Dourdan (Essonne) - Décembre 1991

[KASIK 82] KASIK D. J. ; A User Interface Management System ; Computer Graphics - pp. 99-106 - March 1982

[KIM 90] KIM Won Chul, FOLEY James D. ; DON: User Interface Presentation Design Assistant - SIGGRAPH'90 - pp. 10-20 - October 1990

[KOLSKI 91] KOLSKI Christophe, MOUSSA Faouzi ; Une approche d'intégration de connaissances ergonomiques dans un atelier logiciel de création d'interfaces pour le contrôle de procédés ; Le Génie Logiciel et ses Applications - pp 181-194 - Toulouse - 9-13 décembre 1991

[KUO 85a] KUO Bob, KONSYNSKI Benn ; An Architecture for Dialog Management: Implications in User-Computer Dialog Design ; Interfaces in Computing, Vol. 3, pp. 259-275 - 1985

[KUO 85b] KUO Bob ; Dialogue Modeling for Interoperability in Micro-Based Workstations ; MIS Technical Report N° 84-12-2-001 - University of Arizona, Tucson, AZ, 1985

[KURLANDER 92] KURLANDER David, FEINER Steven ; A History-Based Macro By Example System ; UIST'92, pp. 99-106 - Monterey, California - 15-18 novembre 1992

[LIM 92] LIM K. Y., LONG J. B., SILCOCK N. ; Integrating human factors with the Jackson System Development method: an illustrated overview - Ergonomics, Vol. 35, n° 10, pp. 1135-1161 - 1992

[LINDGREEN 74] LINDGREEN P. ; Basic operations on informatics as a base for data design ; IFIP Congres: Stockholm - 1974

[LINTON 89] LINTON Mark A., CALDER Paul R., VLISSIDES John M. ; Interviews: A C++ Graphical Interface Toolkit ; Internal report - Stanford University - 1989

[MAKSAY 91] MAKSAY G., PIGNEUR Yves ; Reconciling Functional Decomposition, Conceptual Modeling, Modular Design and Object Orientation for Application Development ; IFIP 91 - pp. 233-252 - Elsevier Science Publishers - 1991

[MANDALA 84] Revue Pour la Science - N° spécial N° 85 - Novembre 1984

[MARCA 88] MARCA D. A., McGOWAN C. L. ; SADT - McGraw Hill, New York - 1988

[MÄRTIN 90] MÄRTIN Christian ; A UIMS for Knowledge Based Interface Template Generation and Interaction ; INTERACT'90 - pp. 651-657 - IFIP - 1990

[MÄRTIN 91] MÄRTIN Christian, ERGIN Bülent, KOULOUMDJIAN Jacques, PINON Jean-Marie ; Méthode de conception par objets pilotée par les événements ; Le Génie Logiciel et ses Applications - pp 761-774 - Toulouse - 9-13 décembre 1991

[MATHÉ 90] MATHÉ N. ; Assistance intelligente au contrôle de processus : application à la télémanipulation spatiale ; Thèse de doctorat informatique - École Nationale Supérieure de l'Aéronautique et de l'Espace - Toulouse - 1990

[MEYER 90] MEYER Bertrand ; Conception et programmation par objets ; InterEditions - Paris - 1990

[MIJARES 76] MIJARES I., PEEBLES R. ; Structural semantics of data bases ; In Modeling in Data Base Management Systems ; IFIP Conference: Freudenstadt ; North Holland Publishers - 1976

[MIYASHITA 92] MIYASHITA Ken, MATSUOKA Satoshi, TAKAHASHI Shin, YONEZAWA Akinori, KAMADA Tomihisa ; Declarative Programming of Graphical Interfaces by Visual Examples ; UIST'92, pp. 107-116 - Monterey, California - 15-18 novembre 1992

[MONTALBAN 87] MONTALBAN M. ; Prise en compte des spécifications en ingénierie, application aux systèmes experts de conception ; Thèse de l'Université de Nice - Novembre 1987

[MORAN 81] MORAN T. ; The Command Language Grammar: A Representation for the User Interface of Interactive Computer Systems ; International Journal of Man-Machine Studies, 15 - pp. 3-50 - 1981

[MOZEICO 82] MOZEICO H. ; A Human Computer Interface to Accommodate User Learning Stages ; Communication of ACM - pp. 100-104 - February 1982

[MYERS 87] MYERS Brad A. ; Gaining General Acceptance for SGIU ; ACM Computer Graphics, Vol.1, N°2 - April 1987

[MYERS 88a] MYERS Brad A. ; Tools for Creating User Interfaces: An Introduction and Survey ; CMU-CS-88-107 - Carnegie Mellon - January 1988

[MYERS 88b] MYERS Brad A. ; Creating User Interfaces by Demonstration ; Perspectives in Computing ; Academic Press ; Inc - New York - 1988

[MYERS 89a] MYERS Brad A. ; User-Interface Tools: Introduction and Survey ; IEEE Software - January 1989

[MYERS 89b] MYERS Brad A. et al. ; The Garnet Toolkit Reference Manuals: Support for Highly Interactive Graphical User Interfaces in Lisp ; Report CMU-CS-89-196 - November 1989

[NANARD 90] NANARD Jocelyne ; La Manipulation Directe en Interface Homme-Machine ; Thèse de 3ème cycle informatique de l'Université Montpellier II - Décembre 1990

[NEILSEN 90] NEILSEN J. ; Designing for international use ; Actes de CHI'90, ACM SIGCHI, pp. 291-294 - Seattle, WA - 1990

[NEWMAN 68] NEWMAN W. M. ; A System for Interactive Graphical Programming ; AFIPS Spring Joint Computer Conf. - Thompson Books - Washington, D.C. - 1968

[NIELSEN 91] NIELSEN Jakob, THOVTRUP Henrik ; Assessing the Usability of a User Interface Standard ; CHI'91, pp. 335-341 - New Orleans - 27 avril-2 mai 1991

[NIELSEN 92a] NIELSEN Jakob, DIELI Mary, POLTROCK Steven, ROSENBERG Daniel ; Designing Usable Systems Under Real-Word Constraints: A Practitioners Forum ; CHI'92, pp. 149-152 - 3-7 may 1992

[NIELSEN 92b] NIELSEN Jakob ; Finding Usability Problems Through Heuristic Evaluation ; CHI'92, pp. 373-380 - 3-7 may 1992

[NIELSEN 93a] NIELSEN Jakob, LANDAUER Thomas K. ; A Mathematical Model of the Finding of Usability Problems ; INTERCHI'93, pp. 206-214 - Amsterdam - 24-29 avril 1993

[NIELSEN 93b] NIELSEN Jakob, PHILLIPS Victoria L. ; Estimating the relative Usability of two Interfaces: Heuristic, Formal, and Empirical Methods Compared ; INTERCHI'93, pp. 214-221 - Amsterdam - 24-29 avril 1993

[NIGAY 91] NIGAY Laurence, COUTAZ Joëlle ; Software design rules for multi-agent architectures - Amodeus BRA RP2/WP17 - 1991

[NORMAN 86] NORMAN D. A., DRAPPER S. W. ; User Centered System Design ; Amsterdam: Lawrence Erlbaum Associates - 1986

[NORMAND 92a] NORMAND Véronique ; Le modèle SIROCO : de la spécification conceptuelle des interfaces utilisateur à leur réalisation ; Thèse de 3ème cycle informatique - Université Joseph Fourier - Grenoble I - Avril 1992

[NORMAND 92b] NORMAND Véronique, COUTAZ Joëlle ; Unifying the Design and Implementation of User Interfaces through the Object Paradigm ; ECOOP'92 - Utrecht Netherlands - 29 june-3 july 1992

[OLSEN 83] OLSEN Dan R. ; SYNGRAPH : a Graphical User Interface Generator ; ACM Computer Graphics, Vol. 23, N° 3 - pp. 43-50 - July 1983.

[OLSEN 86] OLSEN Dan R. ; Mike : The Menu Interaction Control Environment ; ACM Transaction on Graphics, Vol. 5, N°4 - pp. 318-344 - October 1986

[OLSEN 89] OLSEN Dan R. ; A Programming Language Basis for User Interface Management ; CHI'89 - pp. 171-176 - May 1989

[OSF 89] OSF/MOTIF ; Programmers's Reference Manual, Revision 1.0 ; Open Software Foundation - Cambridge - 1989

[PALANQUE 92] PALANQUE Philippe ; Modélisation par objets coopératifs interactifs d'interfaces homme-machine dirigées par l'utilisateur ; Thèse de 3ème cycle en informatique - Université Toulouse I - Septembre 92

[PALANQUE 93] PALANQUE Philippe, BASTIDE Rémi, DOURTE Louis, SIBERTIN-BLANC Christophe ; Design of User-Driven Interfaces Using Petri Nets and Objects ; CAISE'93 - Paris - Juin 93

[PANET 91] PANET G., LETOUCHE R., PEUGEOT C. ; De MERISE à MERISE/2 : techniques de "refinement" et nouvelles architectures d'applications ; Autour et à l'entour de Merise - pp. 309-328 - Sophia Antipolis - Avril 1991

[PEAUCELLE 77] PEAUCELLE Jean-Louis ; Les besoins en informatique dans les systèmes informatiques de gestion - Thèse de Doctorat d'Etat - Paris - 1977

[PELLAUMAIL 86] PELLAUMAIL P. ; La méthode AXIAL. Conception d'un système d'information ; Éditions d'Organisation - 1986

[PETOUD 89] PETOUD Isabelle, PIGNEUR Yves ; Des spécifications fonctionnelles à l'interface utilisateur sans programmation ; Colloque sur l'ingénierie des interfaces homme-machine - pp. 71-84 - Sophia Antipolis - Mai 1989

[PETOUD 90] PETOUD Isabelle ; Génération automatique de l'interface homme-machine d'une application de gestion hautement interactive ; Thèse de doctorat de sciences économiques - Université de Lausanne - École des Hautes Études Commerciales - 1990

[PETRI 62] PETRI C. ; Kommunication mit automaten ; Ph. D. Dissertation - University of Bonn - 1962

[PIERRET 88] PIERRET-GOLBREICH C. ; Vers un système à base de connaissances centrées-objets pour la modélisation de systèmes dynamiques en biologie ; Thèse de l'Université de Compiègne - Novembre 1988

[PINGAND 89] PINGAND Philippe, NANARD Marc ; Génération d'Interface Dynamiquement Adaptable ; Colloque sur l'ingénierie des interfaces homme-machine - Sophia Antipolis - 24-26 mai 1989

[PLECZON 92] PLECZON Patrick ; Éléments de méthodologie et outils pour l'assistance à l'opérateur : application à la conduite automobile ; Thèse de doctorat informatique - École Nationale Supérieure de l'Aéronautique et de l'Espace - Toulouse - Décembre 1992

[POLLIER 91] POLLIER Agnès ; Évaluation d'interface par des ergonomes : diagnostics et stratégies ; Rapport INRIA N° 1391 - Février 1991

[PORTEJOIE 91a] PORTEJOIE Philippe ; Planification en univers mono-agent (Définitions, concepts, objectifs, exemples, limitations) - Introduction à la planification en univers multi-agents ; Actes de l'Équipe de Recherche Intelligence Artificielle de Clermont-Ferrand - N° 1 - pp. 3-12 - Laboratoire d'informatique - Université Blaise Pascal - Clermont-Ferrand - Mai 1991

[PORTEJOIE 91b] PORTEJOIE Philippe ; Contribution à l'étude des systèmes d'intelligence artificielle distribuée : planification en univers multi-agents ; Actes de l'Équipe de Recherche Intelligence Artificielle de Clermont-Ferrand - N° 1 - pp. 13-25 - Laboratoire d'informatique - Université Blaise Pascal - Clermont-Ferrand - Mai 1991

[RAS 83] RASMUSSEN J. ; Skills, Rules and Knowledge, Signals, Signs and Symbols, and Others Distinctions in Human Performance ; IEEE Transactions on Systems, Men and Cybernetics, SMC-13, 3, pp. 233-257 - 1983

[REISNER 81] REISNER Phyllis ; Formal Grammar and Human Factors Design of an Interactive Graphics System ; IEEE Transactions on Software Engineering - Vol. SE-7, N° 2 - pp. 229-240 - March 1981

[RICHARD 83] RICHARD Jean-François ; Logique de fonctionnement et logique d'utilisation ; Rapport de recherche INRIA N° 202 - Avril 83

[ROCHFELD 90] ROCHFELD Arnold ; Les méthodes de conception orientées objet ; INFORSID - pp. 563-593 - 1992

[ROCHFELD 91a] ROCHFELD Arnold, MEHL Christian, MALBOSC Guy, FOLLOT Maurice ; Retombées d'une modélisation E-R sur des modèles de traitement Merise ; Génie logiciel et systèmes experts- N° 25 - Décembre 1991

[ROCHFELD 91b] ROCHFELD Arnold ; Modèle externe de données et modèle externe objet ; Le Génie Logiciel et ses Applications - pp. 775-789 - Toulouse - 9-13 décembre 1991

[ROCHFELD 92] ROCHFELD Arnold ; Les méthodes de conception orientées objet ; INFORSID 92 - pp. 563-593 - Clermont-Ferrand - Mai 1992

[ROLLAND 88] ROLLAND Colette, FOUCAUT O., BENCI G. ; Conception des systèmes d'information - La méthode REMORA ; Eyrolles - Paris - 1988

[ROLLAND 91] ROLLAND Colette, CAUVET C. ; Modélisation conceptuelle orientée objet ; Les Septièmes Journées Bases de Données Avancées - Lyon - 25-27 septembre 1991

[ROSE 86] ROSE C. et al. ; Inside Macintosh ; Addison Wesley - 1986

[ROUDAUD 90] ROUDAUD Brigitte, LAVIGNE Valérie, LAGNEAU Olivier, MINOR Earl ; SCENARIOO: A New Generation UIMS ; INTERACT'90 - pp. 607-612 - IFIP - 1990

[ROUSSEAU 88] ROUSSEAU B. ; Vers un environnement de résolution de problèmes en biométrie ; Thèse de l'université Claude Bernard - Lyon I - Février 1988

[RUMBAUGH 91] RUMBAUGH J. et al. ; Object-Oriented Modeling and Design ; Prentice-Hall - 1991

[RUSSI 90] RUSSI Vincenzo, ZOMPI Roberto ; Graphical Object-Oriented Executable Specification for an Automation Oriented Paradigm of Software Development ; Design and Implementation of Symbolic Computation Systems - Vol. 10, N° 12 - pp. 225-235 - CAPRI - Avril 1990

[SACERDOTI 74] SACERDOTI E. D. ; Planning Hierarchy of Abstraction Spaces ; Artificial Intelligence, 5 - pp. 115-135 -1974

[SACERDOTI 77] SACERDOTI E. D. ; A Structure for Plans and Behaviour ; Elsevier Computer Science Library - 1977

[SADOU 91] SADOU S. ; Une architecture semi-homogène des systèmes interactifs ; IHM'91 - Troisièmes Journées sur l'Ingénierie des Interfaces Homme-Machine - pp. 193-198 - Dourdan (Essonne) - Décembre 1991

[SCAPIN 87] SCAPIN Dominique L. ; Guide Ergonomique de Conception des Interfaces Homme-Machine - Rapport INRIA, N° 77 - 1987

[SCAPIN 89] SCAPIN Dominique L., PIERRET-GOLBREICH C. ; MAD : une méthode analytique de description des tâches ; Colloque sur l'ingénierie des interfaces homme-machine - Sophia Antipolis - 24-26 mai 1989

[SCHEIFLER 86] SCHEIFLER R.W., GETTYS J. ; The X-Window System ; ACM Transaction on Graphics - Vol. 5, N° 3 - pp. 79-109 - April 1986,

[SCHIELE 90] SCHIELE Franz, HOPPE H. Ulrich ; Inferring Tasks Structures from Interaction Protocols ; INTERACT'90 - pp. 567-572 - IFIP - 1990

[SCHLAER 91] SCHLAER S., MELLOR S. J. ; Object Life Cycles: Modeling the World in States ; Yourdon Press - 1991

[SCHMUCKER 86] SCHMUCKER K. J. ; MacApp: An Application Framework ; Byte - pp. 189-192 - August 1986

[SCHULERT 85] SCHULERT A.J., ROGERS G.T., HAMILTON J.A. ; ADM : A Dialogue Manager ; CHI'85 - pp. 177-183 - April 1985

[SEBILLOTTE 88] SEBILLOTTE Suzanne ; Hierarchical Planning as a Method for Task Analysis: The Example of Office Task Analysis ; Behaviour and Information Technology, Vol. 7, N° 3 - pp. 275-293 - 1988

[SENACH 89] SENACH B. ; Vers un éditeur d'Interfaces ergonomiques ; Colloque sur l'ingénierie des Interfaces Homme-Machine - pp. 167-180 ; Sophia-Antipolis - 24-26 mai 1989

[SENACH 90a] SENACH B. ; Évaluation ergonomique des interfaces homme-machine : une revue de la littérature ; Rapport de recherche INRIA N° 1180 - 1990

[SENACH 90b] SENACH B. ; Évaluation de l'ergonomie des interfaces homme-machine : du prototypage aux systèmes experts ; ERGO.IA'90, AFCET SELF IDLS - Biarritz - 1990

[SHAN 89] SHAN Yen-Ping ; An Event-Driven Model-View-Controller Framework for Smalltalk-80 ; OOPSLA'89 - pp. 347-352 - 1989

[SHAN 90] SHAN Yen-Ping ; An Object-Oriented UIMS for Rapid Prototyping ; INTERACT'90 - pp. 633-638 - IFIP - 1990

[SIBERT 86] SIBERT J. L., HURLEY W. D., BLESER T. W. ; An Object-Oriented User Interface Management System ; Computer graphics : SIGRAPH'86 Conference Proceedings , Vol. 20, N° 4 - pp. 259-268 - DALLAS - 18-22 août 1986

[SIBERTIN 85] SIBERTIN-BLANC Christophe ; High Level Petri Nets with Data Structure ; 6th European Workshop on Petri Nets and Applications - Espoo (Finland) - June 1985

[SINGH 89a] SINGH S., GREEN Mark ; A High-Level User Interface Management System ; CHI'89 - pp. 133-138 - May 1989

[SINGH 89b] SINGH S., GREEN M. ; Chisel: A System for Creating Highly Interactive Screen Layouts - SIGGRAPH'89 - pp. 86-94 - November 1989

[SINGH 91] SINGH S., GREEN M. ; Automating the Lexical and Syntactic Design of Graphical User Interfaces: The UofA* UIMS - ACM Transactions on Graphics, Vol. 10, N° 3, pp. 213-254 - July 1990

[SLEEMAN 82] SLEEMAN D., BROWN J. S. ; Intelligent Tutoring Systems ; Academic Pres - London - 1982

[TAHON 90] TAHON C., SOENEN R. ; Interfaces homme-machine et productique : un outil générique pour la conception de systèmes de gestion d'interfaces ; CIM'90 - Bordeaux - 12-14 juin 1990

[TALBOT 91] TALBOT Stéphane, AYEL Marc ; Programmation par objets : construction itérative des taxinomies en gérant explicitement les exceptions ; Le Génie Logiciel et ses Applications - pp 57-67 - Toulouse - 9-13 décembre 1991

[TANNER 85] TANNER P.P., BUXTON W.A.S ; Some Issues in Future User Interface Management System (UIMS) Development ; User Interface Management Systems - Springer-Verlag - pp.67-79 - Berlin - 1985

[TARBY 91a] TARBY Jean-Claude, BARTHET Marie-France ; Contrôleur de Dialogue à Base de Règles ; Le Génie Logiciel et ses Applications - pp 167-180 - Toulouse - 9-13 décembre 1991

[TARBY 91b] TARBY Jean-Claude, BARTHET Marie-France ; Productions d'interfaces : vers la génération automatique de contrôleur de dialogue ; IHM'91 - Troisièmes Journées sur l'Ingénierie des Interfaces Homme-Machine - Dourdan (Essonne) - Décembre 1991

[TARBY 92] TARBY Jean-Claude, BARTHET Marie-France ; Du modèle tâche au modèle objet ; IHM'92 - Quatrièmes Journées sur l'Ingénierie des Interfaces Homme-Machine - pp. 93-98 - Paris - 30 novembre-2 décembre 1992

[TARBY 93] TARBY Jean-Claude, BARTHET Marie-France ; Automatic Interface and Dialogue Controller Generation with the Diane Method ; Rapport Interne - LIS - Université Toulouse I

[TARDIEU 85] TARDIEU H., ROCHFELD Arnold, COLLETTI R., PANET G., VAHEE G. ; La méthode MERISE, démarches et pratiques ; Édition d'Organisation - 1985

[TARDIEU 86] TARDIEU H., ROCHFELD Arnold, COLLETTI R. ; La méthode MERISE, principes et ouils ; Édition d'Organisation - 1986

[THIMBLEBY 80] THIMBLEBY H. ; Dialogue Determination ; International Journal of Man-Machine Studies, 13 - pp. 295-304 - 1980

[TRAMIS 92] CONCIS - 37 bis, rue du Prébuard - 95100 Argenteuil ; Tramis : un atelier de conception intégré - Manuel de référence (View, Dialog, Flow)

[TREUD 76] TREUD S. ; A Framework of Characteristics Applicable to Graphical User-Computer Interaction ; In User-Oriented Design of Interactive Graphic Systems, Association for Computing Machinery - pp. 61-71- New York - 1976

[VALDEZ 89] VALDEZ Ray ; XVT: A Virtual Toolkit for Windows and the Mac ; Byte, Vol. 14, N° 3 - pp. 209-216 - 1989

[VAN ZIJL 91] VAN ZIJL L., MITTON D. ; Using Statecharts to Design and Specify a Direct-Manipulation Graphical User Interface ; Proceedings of the 6th Southern African Computer Symposium - pp. 51-68 - July 91

[VANDER 88] VANDER-ZANDEN B. T. ; Constraint Grammars in User Interface Management Systems ; Graphics Interface'88 - pp. 176-184 - 1988

[VANDER 90] VANDER-ZANDEN B. T., MYERS Brad A. ; Automatic, Look-and-Feel Independent Dialog Creation for graphical User Interfaces - CHI'90 - pp. 27-34 - April 1990

[VIGNAUD 89] VIGNAUD André, RAMES Jean-René, ROLLAND Colette ; Une méthode pour la spécification d'interfaces graphiques interactives ; Colloque sur l'ingénierie des interfaces homme-machine - pp. 149-167 - Sophia Antipolis - Mai 1989

[VOGEL 88] VOGEL C. ; Génie cognitif. ; Masson - 1988

[WASSERMAN 82a] WASSERMAN A. I., SHEWMAKE D.T. ; Rapid Prototyping of Interactive Information Systems ; ACM Software Engineering Notes, Vol. 7, N° 5 - pp.171-180 - 1982

[WASSERMAN 82b] WASSERMAN A. I. ; The user Software Engineering Methodology: An Overview - Information Systems Design Methodologies: A Comparative Review - pp. 591-628 + 647-648 - IFIP - 1982

[WASSERMAN 85] WASSERMAN A. I. ; Extending State Transition Diagrams for the Specification on Human-Computer Interaction ; IEEE Transaction on Software Engineering, Vol. 11, N° 8 - pp. 699-713 - August 1985

[WASSERMAN 87] WASSERMAN A. I., PIRCHER P., SHEWMAKE D., KERSTEN M. ; Developing Interactive Information Systems With The User Software Engineering Methodology ; Readings in Human Computer Interaction ; A Multidisciplinary Approach - Morgan Kaufmann Publishers ; Inc - Los Altos - pp. 561-575 - 1987

[WASSERMAN 91] WASSERMAN A. I., PIRCHER P. A. ; Object-Oriented Structured Design ; Tutorial Tools 91 - Paris - 1991

[WEBSTER 89] WEBSTER B. F. ; The NextBook - Addison Wesley - 1989

[WENGER 87] WENGER Étienne ; Artificial Intelligence and Tutoring Systems - Computational and Cognitive Approaches to the Communication of Knowledge ; Morgan Kaufmann Publishers - 1987

[WIECHA 89] WIECHA Charles, BENNETT William, BOIES Stephen, GOULD John ; Generating Highly Interactice User Interfaces ; CHI'89 - pp. 277-282 - May 1989

[WIECHA 90] WIECHA Charles, BENNETT William, BOISES S., GOULD J., GREEN S. ; ITS: A Tool for Rapidly Developing Interactive Applications - ACM Transactions on Information Systems, Vol. 8, N° 3, pp. 204-236 - July 1990

[WINDOWS 93] Windows Sound System : Donnez la parole à votre PC - Horizons, Magazine des utilisateurs des produits Microsoft, n° 13 - Juin, juillet, août 1993

[WORMS 91] WORMS Philippe ; ETRUSC : étude d'une interface utilisateur pour la sécurité civile ; Mémoire du diplôme d'ingénieur du CNAM - Paris - Juillet 91

[YAP 88] YAP Sue-Ken, SCOTT Michael L. ; A Grammar-Based Approach to the Automatic Generation of User Interface Dialogues ; CHI'88 - pp. 73-78 - May 1988

[YAP 90] YAP Sue-Ken, SCOTT Michael L. ; PENGUIN: A Language for Reactive Graphical User Interface Programming ; INTERACT'90 - pp. 619-624 - IFIP - 1990

Index des auteurs

Index des figures