MASTER EN SCIENCES INFORMATIQUES

Aspects Méthodologiques des Environnements de Quatrième Génération

Léonard, Véronique; Laurent, Didier

Award date:
1989

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix
NAMUR
Institut d'Informatique

Aspects Méthodologiques des Environnements de Quatrième Génération

Véronique LÉONARD
& Didier LAURENT

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

1988-1989

Nos premiers remerciements s'adressent à Messieurs François Bodart et Jean-Luc Hainaut pour les conseils judicieux qu'ils nous ont prodigués au long de l'élaboration de ce mémoire.

Nous souhaitons également remercier le Professeur Silvio Munari et Claude Stricker pour leur accueil et le temps qu'ils ont bien voulu nous consacrer lors de notre stage à Lausanne.

Nous désirons exprimer notre reconnaissance à Monsieur Yves Pigneur pour l'intérêt qu'il a manifesté à l'égard de nos travaux.

Qu'il nous soit enfin permis de remercier tous ceux et celles qui ont, de près ou de loin, contribué à l'aboutissement de ce travail.

Résumé

Des premières expériences en matière de développement d'applications complexes à l'aide d'outils de quatrième génération modernes (SQL*Forms, Rally, Quatrième Dimension, Magic, ...), il apparaît que les modèles et les démarches de conception traditionnels se révèlent particulièrement inadéquats. Les aspects qui sont le plus souvent remis en question concernent d'une part la découpe horizontale *analyse conceptuelle - mise en oeuvre* et d'autre part la découpe verticale *base de données - dialogues - traitements*. En effet, les concepts concrets offerts par ces environnements se rapprochent plus du domaine de la spécification que de celui de la réalisation. D'autre part, les rapports entre les composants BD, dialogues et traitements s'y touvent bouleversés : les traitements y apparaissent comme subordonnés aux composants de la base de données et des dialogues.

Ce travail a consisté à :

- relever et modéliser les concepts fondamentaux des nouveaux environnements de 4ème génération;
- choisir et réaliser une application représentative dans ces environnements;
- relever les inadéquations entre les démarches traditionnelles et ces environnements;
- proposer un cadre méthodologique adapté au développement à l'aide de ces environnements.

Abstract

The first experiments in developing complex applications with recent fourth generation tools (SQL*Forms, Rally, Quatrième Dimension, Magic, ...) have shown that the traditional models and design approaches are partially inadequate. The aspects that are most frequently questioned concern, on the one hand, the multi level design hierarchy (*conceptual - logical - physical*) and on the other hand, the splitting into loosely coupled subsystem such as *data base, dialogues and processing*.
Indeed, the concrete concepts offered by these environments seem to be closer to the specification domain than to the implementation domain. On the other hand, the classical relations between the DB components, dialogues and processing are disrupted : processing specification seems to depend on the Data Base components and on the dialogues instead of the contrary.

Our work has consisted of :

- pointing out and modelizing the basic concepts of the new fourth generation environments.
- choosing and realizing a representative application in these environments.
- pointing out the inadequacies between the traditional design approaches and the new environments.
- proposing a methodological framework suitable for development based on these environments.

Table des matières

TABLE DES FIGURES V

Chapitre I INTRODUCTION 1

I.1. **Les outils de développement d'applications : état de l'art** 1
I.2. **Position du problème** 2
I.3. **Objectif du memoire** 4

Chapitre II LES ENVIRONNEMENTS DE TROISIEME GÉNÉRATION 5

II.1. **Introduction** 5
II.2. **Le Niveau technique de l'environnement et les compétences requises** 6
II.3. **Les Fonctions indépendantes** 6
II.4. **Les Méthodes** 7
II.4.1. *Les modèles* 7
II.4.2. *Les outils logiciels* 8
II.4.3. *La démarche* 8
II.5. **Les Domaines d'application** 9
II.6. **Les problèmes** 9

Chapitre III LES ENVIRONNEMENTS DE QUATRIEME GÉNÉRATION 11

III.1. **Introduction** 11
III.2. **Rappel du formalisme du modèle Entité/Association** 11
III.3. **Les fonctions élémentaires communes à tous les E4Gs** 12
III.3.1. *Gestion et accès aux données* 13
III.3.2. *Saisie et présentation des données* 16
III.3.3. *Traitement des données* 21
III.3.4. *Echange avec l'environnement* 24
III.4. **L'intégration des fonctions** 25
III.5. **Classifications des E4Gs** 26

III.5.1. *Les E4Gs orientés utilisateurs finals et les E4Gs orientés développeurs* 26
III.5.2. *Les E4Gs orientés applications opérationnelles et les E4Gs orientés aide à la décision* 28
III.5.3. *Les E4Gs traditionnels et les E4Gs avancés* 29
III.6. **Conclusion** 29

Chapitre IV **LES CARACTÉRISTIQUES SPÉCIFIQUES DES E4GS AVANCÉS** 30

IV.1. **Introduction** 30
IV.2. **La dynamique des interactions** 30
IV.3. **Synthèse de la modélisation des E4Gs avancés** 35
IV.4. **Spécification de quatre E4Gs dans le modèle descriptif** 36
IV.4.1. *Magic* 36
IV.4.2. *Quatrième Dimension* 39
IV.4.3. *SQL*Forms* 41
IV.4.4. *Rally* 42
IV.5. **Conclusions** 44

Chapitre V **UNE DÉMARCHE DE CONCEPTION** 47

V.1. **Introduction** 47
V.2. **Le Niveau Conceptuel** 47
V.2.1. *Les données* 48
V.2.2. *Les traitements* 48
V.2.3. *Le dialogue* 50
V.3. **Le niveau Logique** 50
V.3.1. *Les données* 50
V.3.2. *Les traitements* 52
V.3.3. *Le dialogue* 57
V.4. **Le niveau physique** 60
V.4.1. *Les données* 60
V.4.2. *Les traitements* 61
V.4.3. *Le dialogue* 61

Chapitre VI **EXPÉRIMENTATION** 63

VI.1. **Choix d'une démarche expérimentale** 63
VI.2. **Spécifications informelles de l'exemple** 64
VI.3. **Analyse fonctionnelle** 65
VI.3.1. *Les Données : Structuration des informations mémorisées* 66
VI.3.2. *La statique des traitements : Spécification des fonctions et des messages fonctionnels* 71
VI.3.3. *La dynamique des traitements : Spécification de l'enchaînement des fonctions* 79
VI.4. **La conception logique** 81
VI.4.1. *Les Données : Le schéma relationnel* 81
VI.4.2. *Les Traitements : La Machine 'Phase'* 83
VI.4.3. *Le Dialogue : Spécification de la tâche* 92
VI.5. **La conception physique** 115
VI.5.1. *Les données : la structuration de la base de données* 115
VI.5.2. *Le dialogue : la description des Formulaires* 117
VI.5.3. *Les traitements : les actions et les expressions* 123
VI.5.4. *Le pilotage de l'application : les déclenchements d'actions* 126

Chapitre VII **EVALUATION CRITIQUE DE LA DÉMARCHE DE CONCEPTION** ... 130

VII.1. **Introduction** 130
VII.2. **Les données** 130
VII.3. **Les traitements** 131
VII.4. **Le dialogue** 135
VII.5. **Le pilotage** 138
VII.5.1. *L'utilisation de la BD par les traitements / la non-utilisation directe de la BD par le dialogue* 139
VII.5.2. *Indépendance Statique/Dynamique* 140
VII.6. **Conclusion** 140

Chapitre VIII **SUGGESTIONS POUR UNE DÉMARCHE DE CONCEPTION "ORIENTÉE E4GS"** 142

VIII.1. **Introduction** 142

VIII.2. **Le niveau conceptuel** 142
VIII.2.1. *Les données* 142
VIII.2.2. *Les traitements* 143
VIII.3. **Le niveau logique** 146
VIII.3.1. *Les données* 146
VIII.3.2. *Les objets du dialogue* 146
VIII.3.3. *Les traitements* 152
VIII.4. **Le niveau physique** 154

CONCLUSION 155

BIBLIOGRAPHIE 157

Table des figures

Figure III.1. : Modélisation du composant "Base de données" 15
Figure III.2. : Modélisation du composant "Interface" 20
Figure III.3. : Modélisation du composant "Traitement des données" 24
Figure IV.1. : Modélisation du composant "Pilotage" 34
Figure IV.2. : Modélisation globale des E4Gs avancés 35
Figure IV.3. : Modélisation de Magic 37
Figure IV.4. : Modélisation de Quatrième Dimension 40
Figure IV.5. : Modélisation de SQL*Forms 42
Figure IV.6. : Modélisation de Rally 43
Figure VI.1. : Schéma Entité/Association 71
Figure VI.2. : Schéma de la dynamique des fonctions de la phase 80
Figure VI.3. : Schéma MAG conforme au relationnel 81
Figure VI.4. : Dynamique d'implémentation 114
Figure VIII.1. : Démarche de spécification conceptuelle des traitements 146
Figure VIII.2. : Démarche de spécification des F/R 151
Figure VIII.3. : Démarche de spécification des B/G 151
Figure VIII.4. : Démarche de spécification logique des traitements 153

Chapitre I

Introduction

I.1. LES OUTILS DE DÉVELOPPEMENT D'APPLICATIONS : ÉTAT DE L'ART

Contrairement à ce que son nom pourrait le laisser suggérer, la quatrième génération des langages de l'informatique ne constitue pas un continuum des trois premières en ce sens qu'elle ne poursuit pas les mêmes objectifs que celles qui la précèdent.

"La 4ème génération n'est pas l'empilage d'une nouvelle couche d'outils au-dessus des langages dits machine (1ère génération), d'assemblage (2ème génération) et évolués (genre Cobol, Fortran, Basic : 3ème génération), mais une rupture totale avec l'objectif et l'environnement de la programmation classique" [Bou 84].

Cette rupture est motivée par les problèmes organisationnels croissants que rencontre l'informatique au cours de sa percée dans l'entreprise. D'une part, la productivité des développeurs d'applications apparaît très faible, en raison notamment du temps qu'ils consacrent à la maintenance des systèmes existants (environ 80 %); d'autre part, leur offre s'avère souvent inadaptée en regard de la demande des utilisateurs.

Les efforts consacrés à l'amélioration de la programmation et de l'analyse, notamment par les techniques de programmation structurée, l'ajout de macro-instructions et pré-processeurs, l'effort de normalisation, qui caractérisent la 3ème génération, laissent place à une approche moins technique et plus fonctionnelle du développement des applications. Cette approche suit deux axes parallèles qui épousent les préoccupations principales de la 4ème génération : *l'octroi d'une certaine autonomie aux utilisateurs finals* dans le développement de leurs applications, qui se traduit par la floraison d'outils de manipulation d'informations destinés à préparer une décision, outils caractérisés par un haut niveau de convivialité et une suppression importante de la programmation; *l'amélioration de la productivité des développeurs et de la qualité du développement des applications*, qui est favorisée par l'apparition d'outils d'auto-développement d'applications lourdes, outils caractérisés par un niveau d'abstraction relativement élevé et une certaine guidance du développement. Il nous semble judicieux de centrer notre étude sur ce second type d'outils, destinés aux développeurs, car il nécessite une approche méthodologique plus globale que le premier et nous paraît dès lors plus intéressant à analyser.

Voici donc le développeur confronté à deux environnements-cibles distincts, aux caractéristiques différentes :

d'une part un environnement qu'il connaît et dont il possède une certaine maîtrise : l'*Environnement de 3ème Génération (E3G),* composé d'un langage de programmation (ex: Cobol), d'un gestionnaire de base de données (ex: RDB), d'un gestionnaire d'écrans (ex: TDMS) et d'un gestionnaire de communications (ex: CICS), ces quatre fonctions étant indépendantes les unes des autres. Associées à cet environnement, des méthodes de conception proposent un ensemble coordonné de modèles, démarche et outils destiné à produire la description conceptuelle, logique et physique d'un système d'information en suivant un processus de développement allant de l'étude d'opportunité à la mise au point du système, en passant par l'analyse conceptuelle, la conception et la réalisation;

d'autre part un environnement qu'il découvre et qu'il doit apprendre à maîtriser : l'*Environnement de 4ème Génération (E4G)* possédant les mêmes fonctions de gestion, de saisie et de présentation, de traitement des données, ainsi que la fonction de communication, mais intégrées dans un même formalisme. Ici encore deux tendances se dégagent : d'un côté, une intégration simple de ces fonctions accompagnée d'une simplification du développement grâce à l'intervention d'outils et de langages de programmation de haut niveau (nous parlerons de **E4Gs traditionnels**); de l'autre côté, une intégration des mêmes fonctions combinée à l'introduction d'une fonction supplémentaire, chargée de gérer la dynamique des interactions entre les autres composants et constituant le pivot de l'environnement : la fonction de *pilotage* (nous parlerons de **E4Gs "avancés"**). Ici également, nous privilégierons au cours de notre étude ces E4Gs avancés desquels nous tenterons une modélisation descriptive, car ils se distinguent des E3Gs par leur structure et méritent davantage que l'on s'y attarde.

I.2. POSITION DU PROBLEME

L'apparition de cette nouvelle fonction gérant le pilotage au sein de l'environnement oblige à considérer les démarches de conception sous un angle différent. La gestion des données représente les fondements d'un E3G traditionnel. Sur celles-ci viennent se greffer les traitements, qui utilisent ces données et traduisent les fonctionnalités de l'application. Ces traitements sont réalisés en quelque sorte sous la forme d'une librairie offrant les services de l'application à l'interface-utilisateur; c'est à ce dernier qu'incombe la tâche de collecter les données auprès de l'utilisateur et de gérer l'enchaînement de ces services (la dynamique de l'application).

Les méthodes de conception classiques proposent un cycle de développement de l'application se décomposant en trois phases :

- *la spécification conceptuelle des données et des traitements.*
  L'analyste dégage d'abord l'ensemble des données de l'application en construisant un schéma conceptuel. Dans un second temps, il modélise les fonctionnalités de l'application en définissant les traitements à opérer sur les données et l'ordre d'enchaînement de ces traitements. Il détermine ainsi la statique et la dynamique des traitements de l'application.

- *la conception logique des données, des traitements et de l'interface.*
  Le schéma conceptuel des données est transformé par le concepteur en un schéma logique, conforme au type de système de gestion de base de données utilisé (hiérarchique, en réseau, relationnel). La statique et la dynamique des traitements servent de base, d'une part, à une décomposition en modules, d'autre part à la conception de l'interface-utilisateur. La décomposition modulaire traduit le fait que les traitements utilisent, pour réaliser leur fonctionnalité, des traitements élémentaires communs (tris, gestion de liste, ...) ainsi que des primitives de gestion de l'interface et de gestion de la base de données. Les traitements sont donc hiérarchisés selon une *relation d'utilisation.*
  La conception de l'interface-utilisateur, quant à elle, est réalisée en parallèle de la conception des traitements. La seule contrainte qu'elle doive respecter est celle de ne pas aller à l'encontre des spécifications fonctionnelles, et, notamment, de la dynamique des traitements définie au niveau conceptuel.

- *la réalisation physique des données, des traitements et de l'interface.*
  Il s'agit, d'une part d'implémenter le modèle logique des données dans le formalisme du SGBD cible, d'autre part de réaliser le codage des primitives de traitements dans le langage de programmation de l'E3G utilisé, de construire l'interface et d'établir le dialogue entre interface et traitements.

La fonction nouvelle apparue dans les E4Gs, que nous avons nommée la fonction de pilotage, modifie sensiblement les liens entre les composants de l'application (données, traitements, interface). C'est elle qui déclenche l'accès aux données de l'application, qui gère le déclenchement des procédures de traitements, qui déclenche la présentation des données à l'utilisateur. Elle prend en charge, en fait, une partie de la dynamique de l'application, définie dans les E3Gs par l'enchaînement des traitements. De plus, cette dynamique se retrouve éclatée au travers des différents objets de cette application.

Par conséquent, intuitivement, les démarches de conception classiques n'apparaissent pas totalement adéquates à ces nouveaux environnements puisqu'elles envisagent une spécification de la dynamique de l'application autonome de la spécification des objets de

l'application.

La seconde restriction portant sur les méthodes de conception classiques concerne le processus de développement. Les E4Gs se situent apparemment à un niveau d'abstraction plus élevé que les E3Gs : ils disposent généralement d'un SGBD relationnel et réduisent la part algorithmique de la conception des traitements et de l'interface. Ils ne nécessitent donc pas, semble-t-il, une conception physique telle que définie dans les méthodes actuelles. La conception physique d'une application dans un E4G se rapprocherait davantage de la conception logique axée vers un E3G.

I.3. OBJECTIF DU MEMOIRE

L'objectif de ce mémoire est de rechercher les concepts méthodologiques qui nous paraissent les mieux adaptés à une conception d'application dans un E4G avancé. Cet objectif, nous devrions l'atteindre au chapitre VIII. Pour arriver à nos fins, nous rappellerons d'abord les concepts principaux des E3Gs (chapitre II) et nous tenterons de déterminer, en les modélisant, ce qui constitue la spécificité des E4Gs par rapport aux E3Gs (chapitre III). Nous étudierons alors la classe particulière d'E4Gs que constituent les E4Gs avancés (chapitre IV) pour ensuite, après l'exposé d'une démarche de conception classique (chapitre V), expérimenter quelques-uns de ces environnements (Rally, SQL*Forms, Quatrième Dimension et Magic) à l'aide de cette démarche (chapitre VI). Cette expérimentation nous permettra d'aborder une vue critique de la démarche adoptée, appliquée aux E4Gs (chapitre VII), critique qui débouchera sur la recherche des mécanismes de conception adéquats à ces environnements, recherche qui constitue notre objectif.

Chapitre II

Les Environnements de troisième génération (E3Gs)

II.1. INTRODUCTION

La **première génération** des langages informatiques était le règne du **langage machine**. Le langage machine est le répertoire des instructions possibles pour un processeur particulier; chaque instruction est constituée d'une suite de symboles binaires. Les instructions, opérateurs et opérandes sont tous représentés par leur emplacement physique dans la machine.

Avec la **deuxième génération** est venu le règne de l'**assembleur**. Les suites de symboles binaires ont été remplacés par des symboles mnémoniques, plus proches de l'esprit du programmeur; celui-ci s'est vu dès lors déchargé du travail fastidieux et périlleux d'allocation d'emplacements physiques des instructions, opérateurs et opérandes, cette tâche incombant aux "assembleurs". Ceux-ci permettent, de plus, le partage de sous-programmes réalisant des fonctions standards, qu'il suffit de mettre au point une fois pour toutes, d'où un gain en temps et en sécurité.

L'écriture de programmes en assembleur présente néanmoins bon nombre d'inconvénients : les instructions permises sont intimement liées à la machine; les programmes doivent être écrits en termes de séquences fastidieuses d'opérations extrêmement primitives; ils sont donc fortement sujets à erreurs, difficiles à comprendre et à modifier, et, enfin, non transférables d'une machine à l'autre.

Toutes ces raisons sont à l'origine du développement de **langages de haut niveau**, dits **langages de troisième génération**. Chaque instruction d'un tel type de langage exprime d'une manière compacte une opération qui nécessiterait une suite parfois assez longue d'instructions si elle était codée en langage machine ou en assembleur. Pour que ces instructions puissent être exécutées par le processeur, un compilateur ou un interpréteur se charge d'effectuer la transformation de ces instructions de haut niveau en séries d'instructions du niveau acceptable pour un assembleur. En principe, le programmeur ne doit donc plus connaître la structure de la machine, et ses programmes deviennent transférables.

II.2. LE NIVEAU TECHNIQUE DE L'ENVIRONNEMENT ET LES COMPÉTENCES REQUISES

Les E3Gs possèdent un niveau d'abstraction assez faible. La programmation en E3Gs est donc souvent éloignée du niveau conceptuel et nécessite un passage par les niveaux logique (où l'on élabore une solution automatisée "abstraite", c'est-à-dire une solution indépendante de l'environnement final) et physique (où l'on transforme la solution "abstraite" en solution "concrète", c'est-à-dire une solution qui est conçue pour une configuration particulière matériel/logiciel donnée).

En conséquence, la programmation en E3Gs nécessite expertise et expérience en programmation. Ces environnements ne sont pas à la portée des analystes ou des utilisateurs finals, mais à la portée des programmeurs, ni experts du domaine d'application comme l'utilisateur, ni experts de l'analyse conceptuelle comme les analystes.

De plus, cette expertise se limite souvent à un langage particulier ou une fonction particulière : un expert en Cobol ne l'est pas nécessairement en C; un expert en base de données ne l'est pas nécessairement en communications.

II.3. LES FONCTIONS INDÉPENDANTES

Les langages de troisième génération ont évolué de manière importante au cours du temps pour devenir ce qu'ils sont actuellement. Les efforts de recherche sont encore aujourd'hui orientés vers l'augmentation de la productivité du programmeur. Celle-ci se traduit par l'amélioration des langages de programmation et des compilateurs (ajouts de macro-instructions, de pré-processeurs, ...) et la normalisation de ces langages, en vue d'obtenir une meilleur portabilité.

Ces langages sont devenus de réels environnements dès que sont venus s'adjoindre au langage de programmation proprement dit des outils facilitant et améliorant la programmation. Ces outils prennent désormais en charge plusieurs des fonctions associées à un système d'information : la gestion des données (gestionnaire de base de données), la gestion de saisie et présentation de ces données (gestionnaire d'écrans), et la gestion des communications du système avec son environnement (gestionnaire de communications), laissant au langage de programmation la fonction de traitements des données et la gestion des interactions entre les différents outils.

La raison d'être de ces nouveaux outils se situe dans leur aptitude à décharger le programmeur d'une partie de son travail, notamment de la programmation des accès aux

données, de la saisie et présentation des données, ...

Pourtant, s'ils possèdent cette propriété avantageuse pour le programmeur, ces outils comportent également un inconvénient majeur : ils n'utilisent pas la même syntaxe, pas les mêmes symboles mnémoniques. En bref, ils ne sont pas écrits dans le même langage. La programmation requiert alors des compétences spéciales de la part du programmeur. Celui-ci doit désormais maîtriser non seulement la syntaxe du langage de programmation, mais aussi la syntaxe du gestionnaire d'écran, du SGBD, ... Il doit également gérer les interfaces entre les différents composants de l'environnement, ces interfaces n'étant pas toujours élémentaires à réaliser.

II.4. LES MÉTHODES

Pour les raisons que nous venons d'invoquer, il est évident qu'on ne s'improvise pas programmeur, et que la démarche de programmation d'un système informatique doit s'insérer dans une démarche plus globale de conception d'un système d'information en terme de données, de traitements sur ces données et de transfert de ces données.

"Les systèmes d'information assurant l'automatisation des tâches administratives et de gestion routinière exigent des ressources méthodologiques importantes.

Pour sortir de l'empirisme et pour promouvoir une véritable approche industrielle tout au long du cycle de vie des SI, il est nécessaire de recourir à des procédés et des moyens qui assurent une définition précise des tâches à entreprendre, une exécution coordonnée et efficace de celles-ci ainsi qu'une exploitation productive des ressources affectées à ces tâches.

Un effort considérable a donc été fait au cours de la décennie actuelle dans la mise au point de **méthodes** de développement des systèmes d'information.

Toute méthode doit proposer une **démarche** fondée sur des **modèles** et mise en oeuvre à l'aide d'**outils logiciels**" [Bod 88a].

II.4.1. Les modèles

Le SI d'une organisation est chargé de représenter son fonctionnement. Il semble nécessaire de modéliser ce fonctionnement avant de construire le SI, en s'appuyant sur des concepts précis. Cela suppose dès lors l'utilisation de modèles aux niveaux conceptuels et logiques du développement du SI.

Il est acquis que les modèles soutenant la conception des applications doivent porter

sur la spécification des données du SI et des traitements associés à ces données. Les dernières recherches en matière de modélisation font apparaître une dimension supplémentaire à la conception : la modélisation de l'interface Homme/Machine. Ce domaine semble effectivement de plus en plus essentiel, étant donné la prolifération des gestionnaires d'interface (Windows, ...) et le degré croissant d'importance qu'ils prennent au sein de l'environnement de programmation.

Il faut souligner que l'apparition de ces différents modèles a amené une certaine standardisation de la conception, accompagnée d'une préoccupation de documentation, donc de clarification de cette conception. Cette clarification s'avère précieuse dans l'optique de la maintenance des systèmes d'information.

II.4.2. Les outils logiciels

"Face aux difficultés rencontrées dans la conception des SI, seul l'emploi d'outils automatisés peut **aider** efficacement l'analyste à vérifier que les modèles qu'il construit sont communicables, complets, cohérents, réalisables et conformes aux besoins.

Le recours à des outils automatisés est d'autant plus indispensable qu'un objectif à long terme est d'automatiser les étapes de réalisation des solutions conceptuelles et logiques retenues.

Les environnements logiciels d'aide au développement des SI, présentés actuellement sous le nom de Computer Aided Software Engineering (CASE), tendent à fournir un ensemble d'outils intégrés passifs (éditeurs de spécifications, bases de spécifications, rapports documentaires, etc.) et actifs (maquettes, prototypes, outils de simulation, générateurs de plans de tests, générateurs de code, gestionnaire de projet, etc.)" [Bod 88a].

II.4.3. La démarche

"La démarche, qui constitue le 3ème pôle de toute méthodologie de conception, doit être vue comme un ensemble de **règles et de propositions générales** qui précisent notamment comment mettre en oeuvre les modèles et outils automatisés en vue de maîtriser les étapes du cycle de développement du SI : l'étude d'opportunité, l'analyse conceptuelle, la conception, la réalisation et la mise au point" [Bod 88a].

La démarche, les modèles et les outils utilisés pour le développement des applications peuvent différer d'une organisation à l'autre. Là n'est pas l'essentiel. Il se situe au niveau de la prise de conscience de la nécessité, à court et à long terme, d'envisager une approche méthodologique de la conception.

II.5. LES DOMAINES D'APPLICATION

Toutes les applications opérationnelles peuvent être couvertes par les E3Gs. La seule restriction pourrait être les applications d'aide à la décision, qui sont plus facilement mises en oeuvre grâce aux outils de la cinquième génération, plus assertionnels qu'algorithmiques.

Il serait difficile d'énumérer de manière exhaustive l'ensemble des langages de troisième génération disponibles actuellement sur le marché. Notons cependant que la plupart d'entre eux sont orientés vers des familles de problèmes, la perte de généralité étant compensée par une plus grande efficacité dans le domaine de spécialisation et par des performances supérieures à la compilation.

On distingue essentiellement les langages "scientifiques" et les langages "de gestion".

Comme leur nom l'indique, les premiers sont destinés aux applications scientifiques, c'est-à-dire à des traitements où les entrées/sorties sont en général peu nombreuses, mais où les calculs internes sont complexes. Fortran (FORmula TRANslator), Pascal, C, notamment, appartiennent à cette première famille.

Les langages de gestion sont, quant à eux, destinés aux traitements pour lesquels les calculs sont relativement élémentaires, mais où les volumes d'entrées/sorties sont importants. Le langage le plus couramment utilisé dans ce domaine est le Cobol (COmmon Business Oriented Language).

II.6. LES PROBLEMES

"Durant le cycle de vie d'une application, divers problèmes peuvent apparaître. Parmi les difficultés les plus connues, le glissement des investissements du développement vers la maintenance vient certainement en tête. L'informatique se maintient mais ne progresse plus. Les demandes de développement de nouvelles applications et d'adaptation des anciennes se heurtent à l'impuissance progressive des équipes informatiques. La priorité à l'informatique opérationnelle, lourde par nature, étouffe des demandes de développement d'applications légères, urgentes, ponctuelles et non plannifiées. Or, celles-ci émanent typiquement des niveaux tactiques et stratégiques de l'organisation, c'est-à-dire qu'elles constituent ce qu'on a nommé l'informatique décisionnelle. Cette lourdeur croissante dans le développement d'applications traditionnelles et cette viscosité des structures de développement vis à vis de telles demandes, considérées de plus en plus critiques, entrainent frustration et démotivation tant chez les utilisateurs que chez les développeurs. La notion de rentabilité est remise en question, non seulement parmi les équipes de développement, mais également chez l'utilisateur, qui attend de l'outil informatique une amplification de sa productivité personnelle" [Hai 88a].

L'une des solutions envisageables à ce problème consisterait à améliorer et surtout utiliser les démarches de conception classiques dans les organisations. Cela clarifierait le processus de développement et faciliterait la maintenance des systèmes, à condition que les développeurs s'efforcent de documenter précisément leur analyse.

Cette solution n'est sans doute pas suffisante. Aussi, les études récentes se tournent-elles vers d'autres environnements de développement d'applications, susceptibles à la fois d'améliorer la productivité du développeur et d'intégrer l'utilisateur final au processus de développement.

Nous examinerons, dans le chapitre III, les caractéristiques principales d'environnements issus de ces recherches : les Environnements de 4ème Génération.

Chapitre III

Les Environnements de quatrième génération (E4Gs)

III.1. INTRODUCTION

"Les environnements de quatrième génération sont souvent présentés comme la solution privilégiée aux problèmes de productivité dans le développement des applications informatiques, ainsi qu'aux problèmes de l'accès par l'utilisateur final au système d'information de l'organisation. L'E4G constitue en effet l'une des réponses technologiques à ces problèmes.

Son objectif est l'accélération et l'amélioration du processus de développement d'applications, soit en offrant au développeur des outils de développement de haut niveau, soit en offrant à l'utilisateur final les moyens de réaliser lui-même certaines applications" [Hai 88a].

Notre propos, dans ce chapitre, est de caractériser les Environnements de 4ème Génération, d'une part en modélisant leurs concepts et fonctionnalités communs, d'autre part en dégageant certains critères permettant de classifier ces environnements.

La modélisation des concepts et fonctionnalités emprunte son formalisme au modèle de structuration des informations de la mémoire d'un SI, le modèle Entité/Association de Messieurs Bodart et Pigneur [Bod 88a]. Le modèle que nous présentons ici est un modèle conceptuel descriptif. Il jouera le rôle de grille d'analyse, de modèle de référence pour l'étude de quatre E4Gs spécifiques : Magic, Quatrième Dimension, SQL*Forms et Rally (chapitre IV). Ces E4Gs particuliers offrent des similitudes importantes au regard des différents critères dégagés dans ce chapitre : ils possèdent un degré élevé d'intégration; ils sont destinés aux développeurs plutôt qu'aux utilisateurs finals; enfin, ils sont adaptés à la conception d'applications opérationnelles plutôt qu'au développement d'applications d'aide à la décision.

III.2. RAPPEL DU FORMALISME DU MODELE ENTITÉ/ASSOCIATION

Cette section reprend les principaux concepts qui sous-tendent le modèle Entité/Association [Bod 88a]. Nous les utiliserons dans la modélisation des E4Gs.

Le modèle Entité/association met en évidence la notion d'**Entité**, chose concrète ou abstraite du réel perçu à propos de laquelle on veut mémoriser de l'information. Les entités sont réparties en classes appelées **Types d'Entités**. Un type d'Entité est caractérisé par un

nom et par une définition.

Une **association** est une mise en correspondance de deux ou plusieurs entités où chacune assure un rôle donné. Les Associations sont elles aussi classées en **Types d'Associations**. Un Type d'Association est caractérisé par un nom, par le rôle des entités reliées et par une définition.
Une contrainte de **connectivité** précise, pour chaque rôle, un couple de nombre min-max d'associations dans lesquelles toute entité doit (min) ou peut (max) assurer un rôle.
Les entités et associations possèdent des propriétés représentées par des attributs. Un **attribut** est une caractéristique ou qualité d'une entité ou association; il peut être **simple** ou **répétitif, élémentaire** ou **décomposable, obligatoire** ou **facultatif**. Il est décrit par un nom, le type d'entité ou d'association qu'il caractérise, une définition et un domaine de valeurs.

Un certain nombre de **contraintes d'intégrité** peuvent être ajoutées pour préciser certaines propriétés invariantes de la structure des données : contraintes d'intégrité statiques définissant les états valides des données et dynamiques définissant les changements d'états valides des données.

Il est à signaler que nous n'avons pas la prétention de vouloir spécifier de manière exhaustive dans le formalisme du modèle Entité/Association l'ensemble des concepts décrivant l'E4G. L'objectif que nous attachons au modèle descriptif global est de servir de base, d'une part, à la description de E4Gs spécifiques et d'autre part à une méthodologie de conception applicable à ces environnements. En cela, nous décrirons les structures utilisées lors de la conception d'une application (l'approche du développeur) et non lors de son exécution (l'approche de l'utilisateur).

III.3. LES FONCTIONS ÉLÉMENTAIRES COMMUNES À TOUS LES E4GS

"Tout E4G est articulé autour d'une BD, gérée par un SGBD. La première famille de fonctions est donc liée à la définition et à la gestion des données, ainsi qu'à l'accès à celles-ci; une deuxième famille offre divers modes de présentation des données; une troisième famille de fonctions permet de traiter les données, tandis qu'une quatrième est dédiée aux échanges avec l'environnement de l'E4G" [Hai 88a].

Nous décrivons à présent ces familles de fonctions. Cette description nous conduira à construire, par incrémentations successives, le modèle descriptif global des E4Gs.

III.3.1. Gestion et accès aux données

III.3.1.1. Description

Dans cette première famille, nous retrouvons les fonctions traditionnelles des environnements de bases de données, et en particulier,

- la définition, la gestion et la consultation des structures de données et des autres composants du système d'information. Cette fonction est assurée par un dictionnaire de données.
- la sélection, la consultation et l'extraction de données selon différents modes : langage interactif de commandes, langage naturel, par formulaire, par langage de programmation, ... La puissance du langage de sélection, son degré de procéduralité et son indépendance par rapport aux structures techniques des données constituent des caractéristiques distinctives de ces langages.
- la gestion des données : modification, création, suppression de données.

"En ce qui concerne la gestion et l'accès aux données, le modèle relationnel s'est rapidement imposé, et ceci d'autant plus que les SGBD de ce type offrent des langages d'interrogation puissants et souvent indépendants des structures techniques des données. Parmi ces langages, SQL est certainement celui qui tend à s'imposer, non pas (seulement) en raison de ses qualités intrinsèques, mais de par son statut actuel de standard. On observe en effet un ajustement progressif des langages de 4ème génération au langage de désignation des données SQL" [Hai 88a].

Un SGBD relationnel représente les données sous forme de tables (relations) organisées en lignes (n-uplets) et en colonnes (attributs). Dans un tel modèle, les structures de données utilisées sont très simples, et aucune référence à la façon d'accéder aux données ne doit être mentionnée, contrairement aux autres modèles (hiérarchique et réseau) qui définissent de façon explicite les chemins d'accès aux données. On parle ici de navigation automatique, car c'est au système de la base de données qu'il appartient de rechercher la solution conduisant aux performances les meilleures.

La plupart des SGBD relationnels offrent en outre des langages de manipulation de données à la fois simples et puissants. "Dans cette optique, un SGBD relationnel constitue très souvent la clef de voûte d'un environnement de quatrième génération" [Hen 87].

III.3.1.2. Modélisation

Les ENTITÉS sur lesquelles repose le modèle descriptif de la Base de Données des

E4Gs reprennent par conséquent la plupart des concepts appartenant aux SGBD relationnels : la **Base de Données**, la **Relation** de la base de données, le **Champ** de la relation, le **Type** du champ.

Chacune de ces entités possède au minimum un **ATTRIBUT** : son nom. Le Champ d'une relation possède en outre un attribut déterminant s'il constitue ou non une clé d'accès aux n-uplets de la relation à laquelle il appartient. Il est, de plus, lié à certaines contraintes : obligatoire (o/n), valeurs minimum et maximum, appartenance à une liste de valeurs, ... Ces contraintes constituent des attributs[1] supplémentaires de l'entité 'Champ de la relation' puisqu'elles sont spécifiées par le développeur pour chaque champ particulier.

Le 'Champ de la relation' est associé de manière univoque à un Type. Les **ASSOCIATIONS** liant les entités 'Base de données', 'Relation' et 'Champ de la relation' sont des *associations d'appartenance*. Elles lient la base de données à ses relations et la relation à ses champs. Ces associations sont des associations de maître à esclave, de propriétaire à membre, en sorte qu'elles possèdent des connectivités 1-1 pour le membre et 1-N pour le propriétaire.

Le nom de l'entité fait partie de son **IDENTIFIANT**. Les identifiants des entités 'Type' et 'Base de Données' se limitent à cet attribut. L'entité-membre d'une association d'appartenance est identifiée par son nom et l'identifiant de l'entité à laquelle elle est associée (le champ d'une relation est identifié par son nom et le nom de la relation auquel il appartient; la relation est identifiée par son nom et le nom de la base de données à laquelle elle appartient)[2].

La figure III.1 représente graphiquement cette modélisation de la Base de Données :

---

1 L'attribut liste de valeurs est un attribut répétitif. Au cours de la modélisation des autres composants de l'E4G, nous en rencontrerons d'autres. Nous ne procéderons pas à la désagrégation des entités ou associations concernées; en effet, ces attributs ne représentent pas des concepts autonomes du réel perçu.

2 Cette spécification de l'identifiant est un écart par rapport à la spécification conforme au formalisme E/A : l'entité est en réalité identifiée par son nom et le rôle que joue l'entité-maître au sein de l'association d'appartenance.

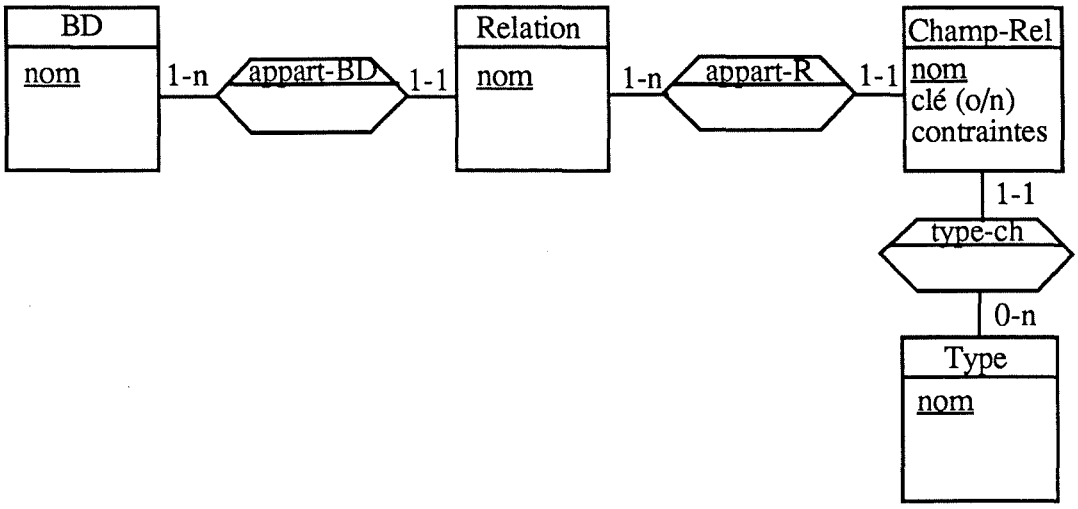


Figure III.1. : Modélisation du composant "Base de données"

Le modèle descriptif ne serait pas complet s'il ne spécifiait les différents types d'actions autorisés sur les objets du modèle, les différentes **PRIMITIVES** fournies par l'environnement au développeur d'application.

Outre les primitives de gestion et de consultation des structures de données assurées par un dictionnaire de données, elles comprennent les primitives d'accès aux données :

- accès aux records d'une relation dans une base de données (éventuellement par clé ou sur base d'un filtre) et à leurs valeurs de champs.

Elles comprennent également les primitives de modification de données :

- création, suppression d'un record;
- modification des valeurs de champs d'un record;
- suppression des records qui vérifient un filtre;
- modification des valeurs de champs des records qui vérifient un filtre.

Elles comprennent enfin les primitives de contrôle des données :

- validation de l'enregistrement d'un champ par rapport à des valeurs minimum et maximum;
- validation de l'enregistrement d'un champ par rapport à une liste de valeurs;
- validation de l'insertion, la modification ou la suppression d'un record en fonction des autorisations d'accès (autorisations d'insérer, de modifier ou de supprimer).

III.3.2. Saisie et présentation des données

III.3.2.1. Description

Les E4Gs sont fondamentalement destinés aux applications interactives. Ils comprennent une fonction importante de saisie et de présentation des données de l'application.

Cette fonction est soit réalisée par des composants spécialisés, soit intégrée à des composants multi-fonctions. Outre la production de rapports simples[1] (cfr SQL par exemple) ou complexes (tableaux à deux dimensions par exemple), nous citerons la présentation de formulaires simples (grilles d'écrans statiques) ou complexes (à zones répétitives, déroulantes, etc.) et la présentation graphique (courbes, histogrammes, camemberts, etc.). La création des rapports ou des formulaires peut être automatisée par des générateurs, outils permettant d'extraire des données de la BD et de les formater de manière standard.

Les formulaires/rapports (F/R) peuvent être composés de différents blocs ou groupes. En général, un bloc/groupe correspond à une relation de la BD et contient un ou plusieurs champs de la relation spécifiée. Il permet d'effectuer des opérations sur les records de cette relation. La correspondance entre les champs d'un record du B/G et les champs du record de la relation est valable dans les deux sens :

- les champs du record du Bloc/Groupe prennent les valeurs des champs du record de la relation avec lesquels ils sont mis en correspondance;
- les modifications effectuées sur les champs du record du B/G sont répercutées en BD lors de la validation de ce record. En effet, la notion de Bloc/Groupe est liée à la notion de transaction. Pour l'environnement, une transaction se termine à la validation d'un record du B/G. La fin de la transaction déclenche l'enregistrement du record en base de données ('commit'). Néanmoins, le développeur peut spécifier qu'une transaction englobe l'enregistrement de records de B/G différents.

Les B/G peuvent également contenir des variables non enregistrées en BD (pour présenter des valeurs calculées, par exemple) ou des champs appartenant à une autre relation. Ces champs sont liés à la relation du B/G par la correspondance de deux champs de même type. Ceci permet de créer une certaine vue[2] sur les relations de la base de données.

---

1 Nous parlerons de "rapport" pour la présentation des données et de "formulaire" pour la saisie et la présentation des données.

2 La vue est la 'fenêtre' par laquelle l'utilisateur accède aux relations de la base de données. Sa représentation est indépendante de l'organisation de cette base de données.

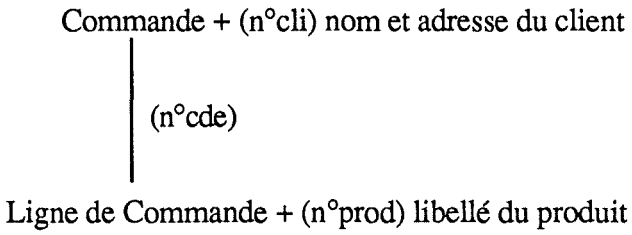
Cependant, ce n'est pas une véritable vue au sens relationnel, car il interdit de pouvoir saisir ou modifier les valeurs des champs de la relation secondaire. Ces champs constituent donc uniquement des champs d'affichage.

Les B/G sont hiérarchisés au sein du F/R. Un B/G fils est lié à son B/G père par la mise en correspondance de deux champs de même type appartenant aux deux B/G liés. Il est à remarquer que ces champs, utiles au lien, ne doivent pas être nécessairement affichés.

Cela signifie qu'un F/R permet indirectement de hiérarchiser des relations de la base de données pour constituer une vue sur une donnée agrégée et pouvoir effectuer des opérations sur cette donnée.

Prenons l'exemple d'un formulaire d'enregistrement d'une commande-client. Ce formulaire représente une vue sur la donnée agrégée *Commande*. Cette vue est le fruit de plusieurs jointures et projections portant sur les relations Commande, Client, Ligne de Commande et Produit. Les deux relations Commande et Ligne de Commande peuvent être représentées, chacune, au sein du formulaire, par un Bloc/groupe. Le B/G Ligne de Commande (fils) est lié au B/G Commande (père) par le champ 'numéro de commande'. Ce champ fait partie du B/G Ligne de Commande sans être pour autant affiché.
Le nom et l'adresse du client passant la commande peuvent être insérés dans le B/G Commande, par l'intermédiaire du champ 'numéro de client'. Le libellé du produit commandé peut être inséré dans le B/G Ligne de Commande par l'intermédiaire du champ 'numéro de produit'. Ces champs ne faisant pas partie de la relation principale du B/G ne peuvent être ni saisis, ni modifiés.

En voici l'illustration :



Il est à noter qu'une donnée agrégée doit être enregistrée en base de données d'un seul tenant. Dans cet exemple, une commande ne peut pas être enregistrée sans ses lignes (et vice versa). La validation des records du B/G Ligne ne doit donc pas déclencher l'enregistrement immédiat de ces records en BD. Le développeur doit spécifier que la transaction sur la donnée agrégée Commande porte conjointement sur la commande et ses lignes.

Les E4Gs permettent que le pilotage, le contrôle de l'exécution d'une application interactive soit partagé entre l'homme et l'ordinateur. Ils mettent à la disposition des développeurs une gestion de menus leur permettant de guider l'utilisateur à travers l'application. Ils fournissent également à l'utilisateur la possibilité de déclencher à tout moment l'une ou l'autre commande lui permettant de piloter lui-même l'exécution de l'application. Ce peut être une commande pré-définie d'insertion d'un record dans un formulaire, de query, ... (pendant la saisie d'une commande-client, par exemple, l'utilisateur peut décider d'insérer une nouvelle ligne de commande, d'annuler la commande, ...) ou une commande définie par le développeur, d'appel d'un formulaire ou d'exécution d'une procédure (par exemple, consulter la liste des produits disponibles).

III.3.2.2. Modélisation

La fonction de saisie et présentation des données, l'interface, réunit les concepts évoqués ci-dessus au sein des ENTITÉS suivantes : le **Formulaire/Rapport** (F/R), le **Bloc/Groupe** (B/G) du F/R, le **Champ** du bloc/groupe, la **Variable**, le **Menu**, la **Commande**.

Ces entités possèdent, comme les entités de la base de données, au minimum un ATTRIBUT : leur nom. Comme le Champ d'une relation, le Champ du bloc/groupe possède des contraintes (format, valeurs minimum et maximum, appartenance à une liste de valeurs, valeur obligatoire (o/n), champ affiché (o/n), modification autorisée (o/n), ...) constituant des attributs de l'entité. Ces contraintes ne peuvent évidemment pas contredire les contraintes associées au champ de la relation lié à ce champ du B/G.

Certaines entités de l'interface possèdent également des attributs représentant les paramètres qui leur sont associés : les paramètres d'un Menu représentent les choix associés au menu; ceux d'un F/R ou d'un B/G comprennent les modes d'utilisation autorisés du F/R (création, modification, query, ...), ou du B/G (création, modification, suppression d'un record dans le bloc/groupe) respectivement et ceux d'un Champ d'un bloc/groupe les autorisations d'accès (champ d'affichage, champ de saisie/affichage, ...).

Enfin, le Bloc/Groupe comprend l'attribut Transaction (o/n). Cet attribut permet de spécifier si les records du B/G doivent être enregistrés en BD directement après leur validation ou non.

La Variable est obligatoirement associée à un Type. Les entités 'F/R', 'Bloc/Groupe' et 'Champ d'un B/G' sont liées par des ASSOCIATIONS *d'appartenance*, selon la définition donnée en III.3.1.2 : le F/R a des blocs/groupes, le Bloc/Groupe possède des champs.

Ces mêmes entités sont liées aux entités de la base de données par des *associations de correspondance*. Un bloc/groupe correspond à aucune ou plusieurs relations; une relation peut être associée à plusieurs bloc/groupes. Un champ d'une relation peut être mis en correspondance avec plusieurs champs d'un bloc/groupe; un champ d'un bloc/groupe correspond au plus à un champ d'une relation.

Il faut préciser que s'il ne correspond pas à un champ d'une relation, le champ d'un B/G représente une variable.

Enfin, les Blocs/Groupes sont liés entre eux par une *association de lien*, afin traduire leur arborescence au sein d'un Formulaire/Rapport. La (les) clé(s) de lien constitue(nt) l'attribut de cette association.

L'adjonction de ces entités et associations au modèle descriptif des E4Gs nous permet de dégager deux **CONTRAINTES D'INTÉGRITÉ** :

- une *contrainte d'inclusion* liée aux associations de correspondance. Si un champ d'un bloc/groupe correspond à un champ d'une relation, alors ce bloc/groupe doit être lié par une association de correspondance à cette relation.
- une *contrainte d'exclusion* assortie d'une *contrainte d'existence* liée aux associations de correspondance dont fait partie le Champ de l'interface : un tel Champ doit représenter un Champ de la base de données ou une Variable, l'un des deux seulement.

L'**IDENTIFIANT** des entités-membres d'une association d'appartenance est composé de leur nom et l'identifiant de l'entité à laquelle elles sont respectivement associées (ex: le champ d'un B/G est identifié par son nom et le nom du B/G auquel il appartient)[1]. Les identifiants d'une commande et d'un menu se limite à leur nom respectif.

Les associations, quant à elles, sont identifiées par les rôles qu'y jouent les entités qu'elles relient.

La figure III.2 représente graphiquement cette modélisation.

---

1 Cette spécification de l'identifiant représente, ici encore, un écart par rapport à la spécification conforme au formalisme E/A : l'entité est en réalité identifiée par son nom et le rôle que joue l'entité-maître au sein de l'association d'appartenance.

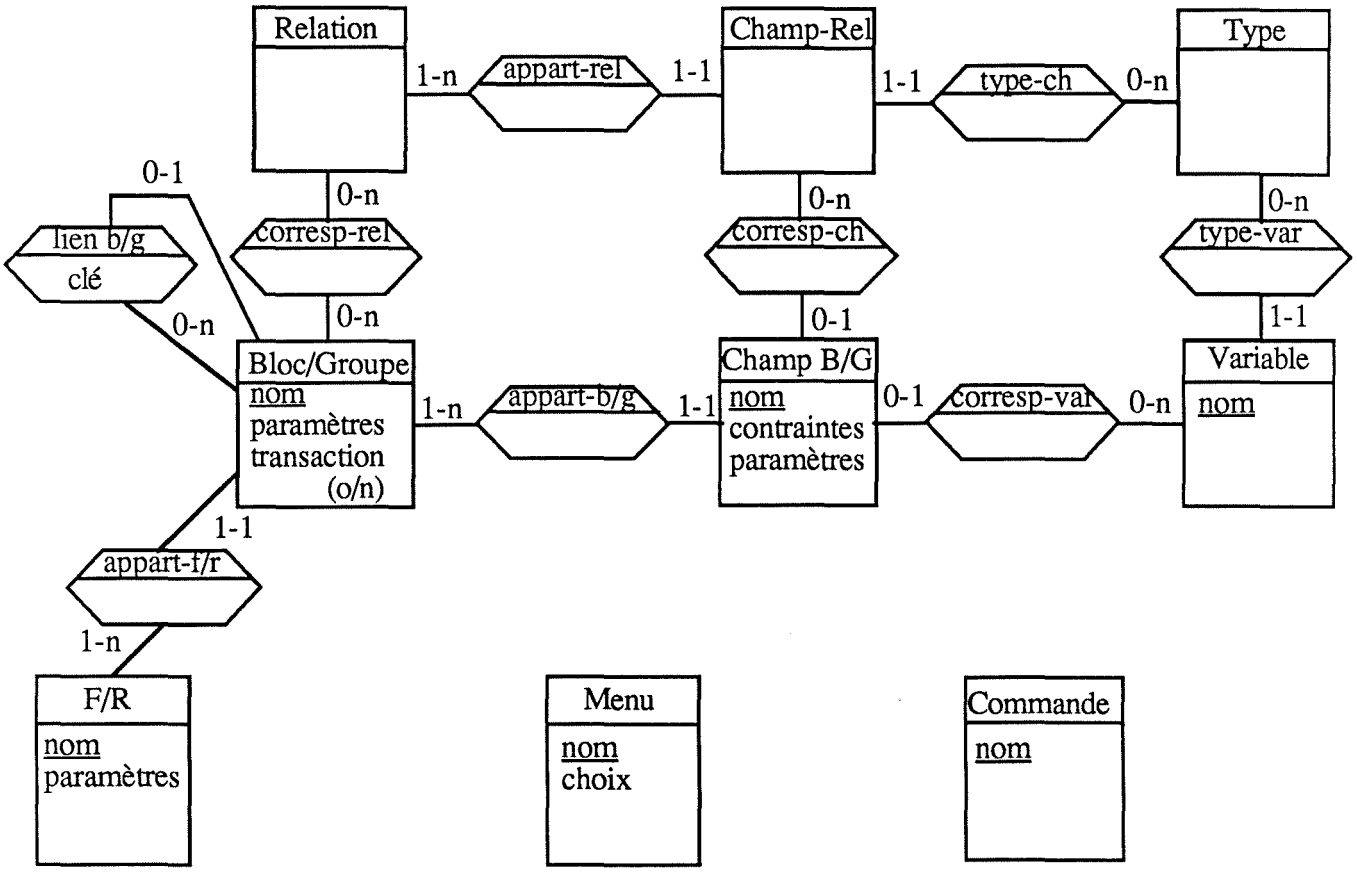


Figure III.2. : Modélisation du composant "Interface"

Il reste à énoncer les **PRIMITIVES** offertes par la fonction de saisie et présentation des données, la fonction d'Interface. L'E4G propose des primitives de structuration de l'interface et de correspondance entre l'interface et la base de données :

- hiérarchisation des blocs/groupes d'un F/R;
- liaison entre un bloc/groupe d'un F/R et une ou plusieurs relations d'une base de données;
- liaison entre un champ d'un F/R et un champ d'une relation d'une base de données.

Nous avons déjà cité l'exemple du formulaire d'enregistrement d'une commande-client, assorti de ses blocs/groupes Commande et Client, pour illustrer ces primitives.

Les autres primitives sont des primitives de contrôle de la saisie des données :

- validation de la saisie d'un champ par rapport à un format de saisie;
- validation de la saisie d'un champ par rapport à des valeurs minimum et maximum;
- validation de la saisie d'un champ par rapport à une liste de valeurs;
- validation de la saisie d'un champ en fonction des autorisations d'accès associées à ce champ (autorisation de lecture uniquement, ...);

- validation de l'insertion, la modification ou la suppression d'un record dans un bloc/groupe en fonction des autorisations d'accès associées à ce bloc/groupe (autorisation d'insérer, de modifier ou de supprimer).

Ces primitives se distinguent de celles appartenant à la base de données par le fait qu'elles s'appliquent aux champs ou aux records d'un bloc/groupe et non plus aux champs ou aux records d'une relation. Illustrons cela par un exemple : soit la valeur d'un Champ d'une relation limitée au minimum à 2 et au maximum à 6. Les Champs des blocs/groupes liés à ce champ de la relation sont tenus de respecter cette contrainte, mais peuvent la renforcer : dans un B/G, la valeur minimum du champ peut être 4 et la valeur maximum 6; dans un autre B/G, minimum 3 et maximum 5.

Il en est de même pour les autorisations d'accès : si le développeur permet d'insérer un nouveau record à une relation, il peut permettre de le faire via un certain formulaire mais non via un autre. L'exemple de l'insertion d'un nouveau produit dans le catalogue des produits est significatif : un produit peut être introduit lors d'une opération de modification du catalogue mais ne peut être introduit lors de l'enregistrement d'une commande-client (les produits peuvent y être, dans ce cas, uniquement consultés).

Les primitives de l'interface sont également les primitives plus classiques de présentation des données :

- génération d'un formulaire sur base d'un format d'écran;
- génération d'un rapport sur base d'un format d'écran;
- génération d'un graphique.

III.3.3. Traitement des données

III.3.3.1. Description

Tous les E4Gs offrent des fonctions de traitement des données extraites de la base de données. D'une manière générale, sélection et traitement des données constituent les fonctions minimales de ce que l'on appelle un environnement de 4ème génération. Ici encore, ces environnements se comparent selon leur puissance et leur degré de procéduralité. La 3ème génération était procédurale en ce sens que l'on devait spécifier pas à pas, par des instructions détaillées et convenablement mises en séquence, comment chaque chose devait s'exécuter. L'efficacité d'un pas d'exécution dépendait fortement du point où l'on était arrivé par le pas précédent.

Par opposition, les Environnements de 4ème génération possèdent une grande part de traitement non-procédural, part qui varie d'un E4G à l'autre. "Il suffit d'indiquer au système

ce qui doit être exécuté, sans avoir besoin de détailler le cheminement du processus opératoire. Le mode non procédural est à la fois plus global et plus autonome : les informations y sont regroupées en agrégats auto-suffisants, dont la taille est généralement plus importante que la simple instruction ou ligne de texte (macro-instructions). En ce sens, le mode non procédural est plus modulaire et moins linéaire que le mode procédural" [Bou 84].

"L'image de la course en taxi de l'homme d'affaire parisien qui veut se rendre rapidement à Marseille à partir de Paris est classique mais très parlante pour illustrer la non-procéduralité :

- mode procédural (E3G) : "Je vous guide : tout droit jusqu'à l'entrée du périphérique extérieur; entrez sur le périphérique par la Porte Maillot, direction Sud; (...) etc., jusqu'à Orly-Ouest, niveau départ";
- mode non-procédural (E4G) : "Menez-moi à Orly-Ouest, départ" " [Bou 84].

Les E4Gs utilisent une forme particulière de mode non-procédural : le mode déclaratif. Au niveau de l'exécution des traitements d'une application, cela signifie qu'il examine à tout moment l'ensemble des opérations spécifiées de façon déclarative et qu'il effectue l'(les) opération(s) se rapportant à l'état dans lequel se trouve le système. Ces opérations sont généralement liées à un Champ d'un Bloc/Groupe. Elles font donc partie de la définition de ce champ. Par exemple, le champ Montant de ligne d'un B/G peut être défini comme étant le produit des champs Quantité commandée et Prix d'achat du Produit commandé.

Outre cette possibilité, les traitements peuvent être exécutés au sein de procédures. Ces procédures peuvent faire appel à des F/R ou d'autres procédures; elles peuvent manipuler des relations, des champs d'une relation ou d'un B/G ou encore des variables.

Parmi les outils qui sous-tendent cette fonction de traitement des données, nous citerons principalement [Hai 88a] :

- le processeur de procédures, qui exécute (par interprétation ou compilation) les procédures de traitement de données,
- le générateur d'application, qui permet la construction d'applications complètes à partir de spécifications de haut niveau, généralement déclaratives. Il produit des programmes compilables ou interprétables à partir de descriptions externes des données, des états, des écrans et des traitements à effectuer.
- les processeurs de modèles (tableurs, processeurs de modèles financiers), qui admettent une programmation non algorithmique,
- les processeurs statistiques et d'analyse des données,

- les processeurs de règles logiques (moteur d'inférence).

III.3.3.2. Modélisation

Les Traitements comprennent une seule **ENTITÉ** : la **Procédure.**

Comme les autres entités de l'environnement, l'entité Procédure possèdent un nom. Elle peut également posséder un **ATTRIBUT** supplémentaire : ses paramètres formels.

Les **ASSOCIATIONS** à mentionner pour cette nouvelle entité sont :

- des *associations de manipulation* d'une Relation (création, modification, suppression d'un n-uplet), d'un Champ d'une relation ou d'un B/G (lecture ou modification de la valeur du champ) ou d'une Variable par la Procédure;
- des *associations d'appel* d'un F/R ou d'une autre procédure.

L'association 'appel-pr' possède un attribut : les paramètres actuels de la procédure appelée. Cet attribut est lié à la contrainte suivante : le nombre et le type des paramètres actuels doivent être identiques à ceux des parmètres formels de la procédure appelée.

Nous avons, jusqu'à présent, modélisé l'aspect procédural du traitement des données. Afin d'en modéliser l'aspect non-procédural, nous avons ajouté l'attribut non-obligatoire 'Expression' à l'entité 'Champ d'un bloc/groupe'. Cet attribut permet de spécifier le calcul déclaratif d'un champ.

La figure III.3 représente graphiquement la modélisation du composant de traitement des données.

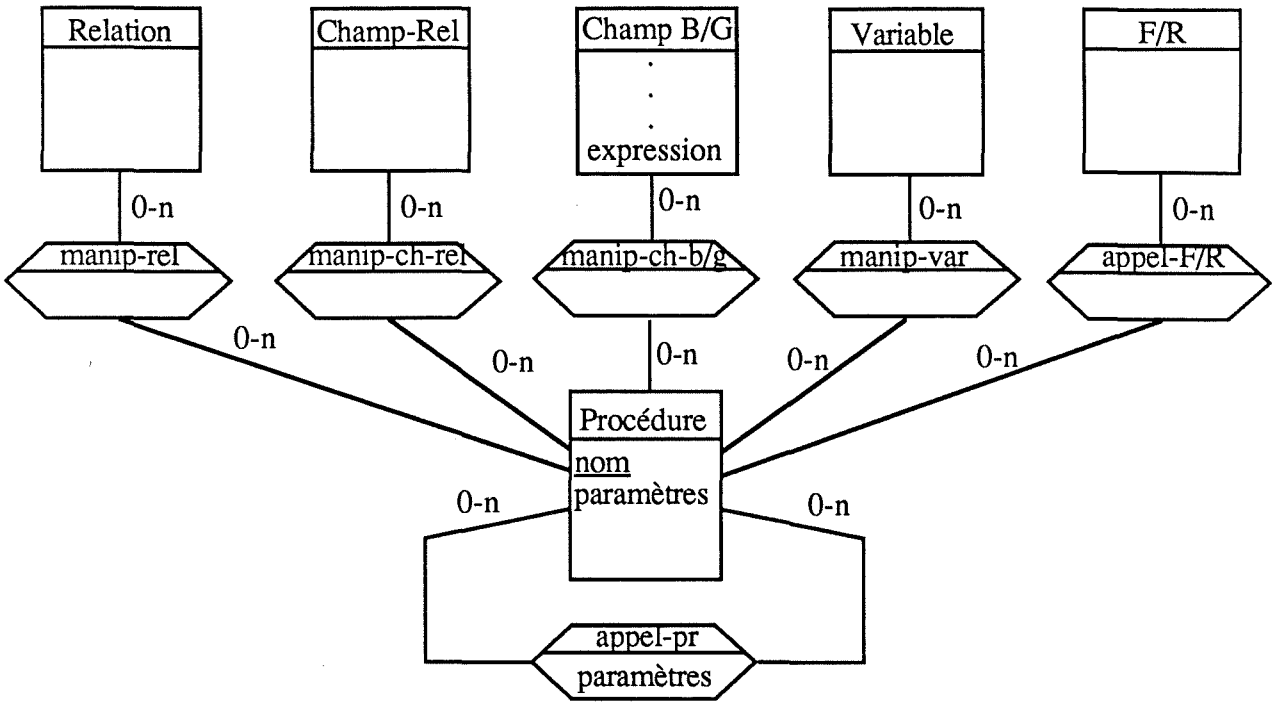


Figure III.3. : Modélisation du composant "Traitement des données"

Outre l'exécution de procédures, permettant d'effectuer des traitements algorithmiques sur les champs d'un F/R ou de la BD grâce à un langage procédural, l'E4G offre un langage non-procédural proposant les **PRIMITIVES** suivantes :

- aggrégation d'un champ d'un F/R sur base d'autres champs (somme, moyenne, maximum, ...);
- expression déclarative de calcul d'un champ d'un F/R sur base d'autres champs du F/R (cette expression est recalculée à chaque changement de valeur de l'un de ces champs). Cette expression peut éventuellement être associée à une condition.

Il possède également une fonction de plus haut niveau, que nous avons déjà évoquée, qui est la génération d'applications. Elle permet, rappelons-le, la construction de l'application complète à partir de spécifications de haut niveau.

III.3.4. Echange avec l'environnement

III.3.4.1. Description

L'E4G n'est souvent qu'une ressource des moyens informatiques de l'organisation. Il est indispensable d'établir des passerelles entre ses fonctions et celles qui ne sont pas sous son contrôle. Les E4Gs offrent des moyens d'échange avec leur environnement dans les

domaines suivants :

- communications (réseaux locaux ou externes) avec d'autres sites;
- accès à des BD externes;
- accès à (ou communication avec) des processeurs externes tels que des tableurs, autres SGBD, etc;
- exécution de programmes externes rédigés en langage de 3ème génération.

III.3.4.2. Modélisation

Nous ne modéliserons pas les concepts appartenant à ce composant puisque notre objectif est, avant tout, de cerner les composants internes de l'E4G. Nous ne nous attarderons donc pas sur ses interactions avec l'environnement.

III.4. L'INTÉGRATION DES FONCTIONS

"Il apparaît rapidement que certaines fonctions ne sont pas spécifiques aux environnements dits de 4ème génération, mais peuvent aussi être rattachées aux outils de la bureautique, aux infocentres, à l'informatique individuelle et au génie logiciel. La caractéristique essentielle des E4Gs réside en fait dans **l'intégration de ces fonctions et de ces composants**" [Hai 88a].

Cette intégration consiste, pour le développeur, à pouvoir utiliser, à tout endroit d'une application, les primitives d'accès, de saisie, de présentation, de traitement et de communication des données, énoncées précédemment.

Les E4Gs comportent des degrés divers d'intégration et de 'complétude' des différentes fonctions requises par l'utilisateur [Bou 84] :

- Certains environnements prétendent être des E4Gs. En fait, ils ne sont que des **boîte à outils**. Ils se parent des avantages de la modularité, mais ceux-ci ne tiennent guère face aux inconvénients majeurs de l'incohérence, des redondances et de la spécificité de nombreuses interfaces, sans parler de la difficulté pour l'utilisateur d'apprendre alors un nombre élevé de syntaxes différentes.
- On parle réellement d'E4G quand l'environnement vise l'**intégration et l'unicité du langage**. Le coeur de cette intégration est évidemment le dictionnaire des objets avec son système de gestion. Il assume la fonction de gestionnaire de toutes les entités constitutives des applications prises en charge par le système et modélisées dans ce chapitre : relations, champs d'une relation, formulaires, rapports, blocs/groupes, champs d'un bloc/groupe, procédures, menus, commandes, etc.

Qu'il soit gestionnaire de ces entités, cela signifie qu'il possède toutes les caractéristiques décrites dans la modélisation des E4Gs :

- il constitue le descripteur des entités, de leur structure, de leur organisation non seulement physique mais également logique;
- il joue un véritable rôle actif, puisqu'il gère la modification de ces structures, de cette organisation en mettant à la disposition du développeur l'ensemble des primitives de structuration des entités du modèle.

La seule syntaxe à utiliser est alors celle requise pour la création, la mise à jour ou la consultation de ce dictionnaire.

- Le degré ultime d'intégration et de cohérence est atteint lorsque l'**E4G lui-même est écrit dans son propre langage**, c'est-à-dire que les fonctions utilitaires de création, mise à jour, consultation, etc. sont décrites dans le même dictionnaire que les objets destinés à l'utilisateur. Cela veut dire que tout est géré dans une base d'information unique : les mécanismes de base du langage aussi bien que les informations proprement dites de l'utilisateur.

III.5. CLASSIFICATIONS DES E4GS

Une multitude d'E4Gs foisonnent sur le marché. Leur degré d'intégration est un élément de classification de ces environnements. Mais il n'est pas le seul. Les E4Gs ne sont pas tous destinés au même type d'utilisateur, ni utilisés pour développer le même type d'application. Ce sont là aussi deux critères de classification.

III.5.1. Les E4Gs orientés utilisateurs finals et les E4Gs orientés développeurs

Les E4Gs visent deux types de destinataires distincts : l'utilisateur final et le développeur. D'une manière générale, la plupart d'entre eux privilégient assez nettement l'un ou l'autre.

La caractéristique essentielle d'un **E4G orienté utilisateurs finals** est la convivialité. A la génération précédente, le développement des applications était toujours monologué (par le développeur, souvent informaticien), et seule l'exécution était éventuellement dialoguée (avec l'utilisateur, dans le cas des applications interactives). Dorénavant, le premier contact s'établit au travers d'une structure d'accueil qui oriente et informe l'usager sur l'ensemble le plus général des possibilités offertes par le système, qu'il s'agisse de développement ou d'exploitation. Le développement est guidé par un langage interactif, généralement déclaratif, à base de formulaires. L'utilisateur évolue dans un

environnement de multi-fenêtrage, il y complète des formulaires, il y navigue par l'intermédiaire de menus et il peut à tout moment consulter des écrans d'aide. Avec les E4Gs orientés utilisateurs finals, l'outil de développement est lui-même une application, et les fonctions dialoguées de développement peuvent se mêler plus ou moins fortement et immédiatement aux fonctions dialoguées d'exécution. Ces E4Gs ont donc un niveau d'intégration élevé, se rapprochant du niveau ultime défini en III.4; le développeur et l'usager peuvent être la même et unique personne en contact avec le système informatique. Pour qu'un tel mariage soit réussi, le système doit fournir à l'utilisateur le maximum de résultats "dans le plus grand confort" et selon "la loi du moindre effort" [Mar 85] :

- *travail minimal* : le langage doit être concis, fournir le maximum d'options par défaut[1] (formats préétablis, proposition automatique des options les plus probables sur menu qu'il suffit de valider, enchaînements et pilotage automatique, etc.); il doit aussi assurer le maximum de 'transparence', qualité qui consiste à cacher à l'utilisateur toute information relative aux niveaux d'abstraction inférieurs au sien (structure et emplacement des informations, vues logiques intermédiaires, etc.);
- *temps minimal* : le temps de réaction de l'utilisateur à son poste de travail doit être mis à profit par le logiciel pour faire un maximum de travail utile pour l'information et le confort de l'utilisateur (contrôles en tout genre, décodages, préventions d'erreurs, etc.);
- *résultats maximaux* : l'E4G orienté utilisateurs finals doit fournir des résultats aussi élaborés (synthétiques) et utiles que possible, proches des besoins réels de l'usager (prise de décision), faciles et agréables à exploiter (graphiques, couleurs, ...).

Un autre aspect fondamental de la convivialité est la rapidité d'apprentissage et la facilité d'emploi. Pour apprécier si ces critères sont vraiment satisfaisants dans un E4G, autrement dit s'il est orienté utilisateurs finals, on peut leur appliquer le test de J. Martin, dit "test des deux jours" [Mar 85].

Les **E4Gs orientés développeurs** sont moins axés sur la convivialité de l'interface que sur la performance de l'outil de développement, la rapidité mais aussi la souplesse du développement d'application. Leur degré d'intégration élevé accélère le processus de développement. En effet, le concepteur peut exécuter et tester son application presqu'instantanément, puisque développement et exécution utilisent les mêmes structures d'objets définies dans le même dictionnaire.

La présence d'un grand nombre d'options par défaut caractérise également ces

---

[1] Le principe de l'option par défaut est le suivant : si l'utilisateur a un choix d'options à réaliser et qu'il n'exerce pas ce choix, l'environnement le fait pour lui. Autrement dit, l'utilisateur confie la prise de décision à l'environnement.

environnements. Plus l'environnement opère ses propres choix, moins l'effort est important pour le développeur.

A un extrême, ce principe donne lieu à la pré-détermination par le système des formats, des dispositions d'écrans, des mécanismes d'accès, etc. Cela s'avère rapide mais peu flexible.

A l'autre extrême, le développeur doit tout expliquer au logiciel, comme un programme Cobol classique. Cela se révèle souple mais lent.

Les E4Gs orientés développeurs n'atteignent à la fois rapidité et souplesse que parce qu'ils opèrent des choix tout en permettant au développeur de les modifier. Cependant, la marge entre rapidité et lenteur, rigidité et souplesse demeure large. Heureusement, certains invariants s'imposent. C'est le cas de la génération de rapports : le logiciel formate le rapport d'une manière standard. Les options choisies peuvent s'avérer satisfaisantes ou nécessiter certaines adaptations. Le développeur se voit toujours autorisé à changer ces options, à modifier le format par défaut du rapport.

Ces caractéristiques ne différencient pas véritablement un E4G orienté développeurs d'un E4G orienté utilisateurs finals. Leur particularité essentielle réside dans le recours à des sous-langages procéduraux pour la description des traitements complexes et le pilotage de l'exécution. Ces sous-langages sont soit des langages de 3ème génération interfacés avec l'E4G, soit un langage procédural propre à l'environnement. Il est à noter que le développeur peut s'astreindre une certaine part de programmation alors qu'un utilisateur final ne pourrait se le permettre puisqu'il ne maîtrise pas la syntaxe des langages de programmation.

III.5.2. Les E4Gs orientés applications opérationnelles et les E4Gs orientés aide à la décision

La deuxième classification des E4Gs concerne les applications vers lesquelles ils sont orientés. La première catégorie, les **E4Gs orientés applications opérationnelles**, sont adaptés au développement des applications nouvelles; ils sont souvent plus axés sur la prise en charge d'événements (saisie, contrôle et mise à jour de données), donc sur la gestion des entrées; tandis que la seconde catégorie, les **E4Gs orientés aide à la décision**, sont destinés à de nouvelles utilisations des applications existantes et visent plutôt l'accès aux informations déjà stockées et la mise en forme de nouveaux résultats (interrogateurs, éditeurs, graphiques), donc la gestion des sorties.

III.5.3. Les E4Gs traditionnels et les E4Gs avancés

La dernière distinction que nous pourrions faire au sein des E4Gs s'opère dans le temps. Les premiers E4Gs se limitaient à l'intégration des composants Base de données, Interface, Traitement et Communication, telle que nous l'avons décrite en III.4 (nous les appelons **E4Gs traditionnels**). Par la suite sont venus s'ajouter des E4Gs intégrant toujours les mêmes composants mais contenant une fonction supplémentaire de gestion de la dynamique des interactions entre ces composants, fonction pivot de l'environnement : la fonction de pilotage. Nous nommons ces nouveaux environnements les **E4Gs avancés**.

Le passage des E4Gs traditionnels aux E4Gs avancés peut être comparé à celui des systèmes de gestion de fichiers aux systèmes de gestion de base de données. Dans le cadre d'une application bâtie sur un système de gestion de fichiers, le programmeur intègre la gestion des liens entre fichiers pour chaque application particulière alors qu'il en est déchargé avec un système de gestion de base de données. De la même manière, un E4G avancé lui permet de se décharger d'une partie de la programmation des interactions entre BD, interface, traitements et communications.

III.6. CONCLUSION

Dans ce chapitre, nous avons mis en évidence les fonctions et composants des environnements dits de quatrième génération : la gestion, la saisie et la présentation, le traitement des données ainsi que la fonction de communication. Ces fonctions ne sont pas spécifiques aux E4Gs. La caractéristique essentielle de ces environnements réside dans l'intégration de ces fonctions et de ces composants.

Nous avons également tenté d'établir une classification des E4Gs : E4Gs orientés utilisateurs finals et E4Gs orientés développeurs; E4Gs orientés applications opérationnelles et E4Gs orientés aide à la décision; E4Gs traditionnels et E4Gs avancés.

Cette dernière classification est à la base du chapitre IV : dans ce chapitre, nous analysons et modélisons les E4Gs avancés, et, plus particulièrement, la fonction de pilotage gérant la dynamique des interactions entre les composants de ces environnements. Les E4Gs spécifiques que nous étudions ensuite et sur lesquels nous nous basons dans les parties ultérieures de ce mémoire appartiennent à cette catégorie. Ils sont également orientés développeurs et orientés applications opérationnelles. Ce sont les E4Gs Magic, Quatrième Dimension, SQL*Forms et Rally.

Chapitre IV

Les caractéristiques spécifiques des E4Gs avancés

IV.1. INTRODUCTION

Le concept majeur des environnements traditionnels, qu'ils soient de troisième ou de quatrième génération, est la procédure. C'est elle qui gère le déroulement de l'application en utilisant les primitives de traitements, les primitives et objets de la base de données, les primitives et objets de l'interface. Ce qui différencie troisième et quatrième génération est d'une part l'indépendance (E3G) ou l'intégration (E4G) des composants de l'environnement et, d'autre part, le caractère procédural (E3G) ou non-procédural (E4G) de la description des objets de l'application et de leurs interactions.

L'élément central des environnements récents, les environnements de 4ème génération avancés, n'est plus la procédure mais bien les objets de l'application eux-mêmes. Les objets des E4Gs avancés sont les mêmes que ceux des environnements traditionnels : on y décrit aussi bien des relations, des formulaires, des rapports, des menus et des sous-ensembles plus ou moins agrégés de procédures. La différence entre ces nouveaux environnements et les environnements traditionnels réside dans le fait que la dynamique de l'application n'est plus gérée au sein d'une procédure ou d'un programme, mais au sein de chaque objet composant l'application. En effet, dans un E4G avancé, l'objet (la relation, le formulaire/rapport, le menu, ...) est susceptible de pouvoir déclencher lui-même une action, qui peut être une procédure ou un autre objet de l'environnement (un F/R ou un menu).
Dans un E4G avancé, le développeur se charge donc, d'abord, de décrire, dans un langage de haut niveau, les objets qui constituent son application, en les centralisant et les intégrant au sein du dictionnaire des objets. Il traduit ensuite la dynamique de l'application en l'éclatant au sein de ces objets, c'est-à-dire qu'il définit les enchaînements d'actions propres à chacun d'eux.

Si la description des enchaînements d'actions au sein d'une application est à charge du concepteur, la gestion de ces enchaînements et le contrôle de leur validité est, quant à elle, à charge de l'environnement. Cela constitue le rôle de la **fonction de pilotage**.

IV.2. LA DYNAMIQUE DES INTERACTIONS

IV.2.1. Description

Le module pivot de l'E4G possède deux fonctions imbriquées : une fonction

d'enchaînement d'actions et une fonction de contrôle de la validité de ces enchaînements.

Le formulaire, le rapport et le menu constituent chacun, au même titre que la procédure, une *action* à part entière susceptible de traduire un traitement, une fonctionnalité de l'application. C'est la particularité première des E4Gs avancés, par rapport aux E4Gs traditionnels.

Toute action peut manipuler l'ensemble des champs des F/R actifs au moment de son déclenchement. Dans l'exemple de l'enregistrement d'une commande, toute procédure déclenchée au sein du B/G Ligne de commande a accès non seulement aux champs de ce B/G, mais également aux champs du B/G père Commande. Cela signifie que les champs du B/G père constituent, en quelque sorte, des variables globales pour les actions dépendant de ce B/G.

Le *déclenchement d'une action* est guidé par la survenance d'un **événement**. Il peut être spécifié par le développeur. L'événement déclencheur peut être lié non seulement à l'interface (clic-souris, frappe d'une touche-fonction ou survenance d'un message interactif) ou aux traitements (terminaison d'une procédure, ...) comme il l'est dans les E3Gs et les E4Gs traditionnels mais également à la dynamique des données (insertion, modification ou suppression d'un record). La plupart des objets de l'E4G peuvent abriter le déclenchement d'une action. Une action est déclenchée lors d'un changement d'état de l'objet. Les différents changements d'état de ces objets (les 'action sites') sont plus nombreux dans les E4Gs que dans les environnements classiques. C'est leur second trait caractéristique. Ils peuvent être répertoriés de la manière suivante :

- liés à la Base de Données :
  - avant, après l'insertion d'un record;
  - avant, après la modification d'un record;
  - avant, après la suppression d'un record;

- liés à l'Interface :
  - à l'activation[1], à la désactivation d'un formulaire ou un rapport;
  - à l'activation, à la désactivation d'un bloc/groupe;
  - à l'activation, à la désactivation d'un record d'un bloc/groupe;
  - à l'activation, à la désactivation d'un record;
  - à l'activation, à la désactivation d'un champ;

---

[1] L'activation s'effectue au moment ou le curseur se positionne sur l'objet considéré; la désactivation coïncide avec le moment où le curseur quitte l'objet. Un champ d'affichage n'est donc pas activé.

- à l'activation d'un menu;
- au choix d'un menu;
- avant, après un query;
- à la survenance d'une commande-utilisateur;

- liés aux Traitements :
  - au sein d'une procédure.

Un Formulaire peut par exemple être déclenché avant l'activation d'un champ ou consécutivement au choix d'un menu. De même, une procédure peut être déclenchée après la modification d'un record en BD ou à la survenance d'une commande-utilisateur.

Le *contrôle du déclenchement de l'action* s'effectue, en fonction du type d'événement reçu et de l'état courant du système, sur base des règles d'enchaînement associées à l'application. Ces régles peuvent spécifier des enchaînements automatiques (s'ils dépendent de l'ordinateur) ou optionnels (s'ils dépendent de l'utilisateur).

Un changement d'état d'un objet peut en cacher un autre. Par exemple, l'activation d'un F/R entraîne l'activation du premier B/G de ce F/R, l'activation du premier record de ce B/G et l'activation du premier champ de ce record. De même, la désactivation du F/R sera précédée de la désactivation du champ, du record et du B/G actifs.

IV.2.2. Modélisation

La modélisation de la fonction de pilotage vient se greffer sur la modélisation que nous avons réalisée des autres composants d'un E4G, la base de données (cfr III.3.1.2), l'interface (cfr III.3.2.2) et les traitements (cfr III.3.3.2).

Le pilotage comprend une **ENTITÉ** : l'**Action**. Elle constitue une entité générique, dont les entités spécifiques sont la Procédure, le F/R et le Menu. En effet, chacune de ces entités représente une spécialisation, un type d'action. L'action possède un nom, le nom de l'entité spécifique qu'elle représente[1].

En traduisant ceci dans le formalisme du modèle Entité/Association, nous pouvons dire que l'entité Action est liée aux entités Procédure, F/R et Menu par des **ASSOCIATIONS** *de généralisation-spécialisation (is-a)*. L'Action constitue le type générique et la Procédure, le F/R et le Menu, les types spécifiques. Ces associations sont liées à une *contrainte d'exclusion* assortie d'une *contrainte d'existence* : une action ne peut être qu'un objet à la

[1] Les entités spécifiques voient donc leur nom migrer vers l'entité générique.

fois (une procédure, un formulaire/rapport ou un menu), mais doit être au moins l'un d'entre eux.

Les autres associations qui traduisent le lien entre l'Action et les autres entités de l'environnement sont les *associations de déclenchement.* Elles représentent la dynamique de l'environnement. Elles lient l'action et l'objet déclenchant l'action. Cet objet peut être une Relation, un F/R, un Bloc/Groupe, un Champ d'un bloc/groupe, une Procédure, une Commande ou un Menu. Ces associations possèdent des attributs :

- le changement d'état de l'objet auquel le déclenchement est associé (ex: avant insertion, après modification d'un record en BD, ...). La liste de ces changements d'état est reprise en IV.2.1;
- les paramètres actuels de procédures;
- les autorisations d'accès au F/R (la première valeur désignant le mode d'utilisation initial du F/R).

Ces deux derniers attributs sont répétitifs et non obligatoires. Il leur est associé les contraintes d'intégrité suivantes :

- si l'action déclenchée est une procédure, l'attribut 'autorisations d'accès' doit posséder la valeur nulle;
- si l'action déclenchée est un F/R, l'attribut 'paramètres' doit posséder la valeur nulle;
- si l'action déclenchée est un menu, les attributs 'paramètres' et 'autorisations d'accès' doivent tous deux posséder la valeur nulle.

L'entité Action est identifiée par son nom et le F/R, la procédure ou le menu qu'elle représente.

L'**IDENTIFIANT** des associations de déclenchement est constitué du rôle assumé, au sein de l'association, par l'objet déclenchant l'action et de l'attribut 'changement d'état de l'objet' de l'association. Cela signifie que deux actions ne peuvent être déclenchées au même changement d'état d'un objet.

Le schéma de la figure IV.1 constitue la représentation des actions et des objets pouvant les déclencher :

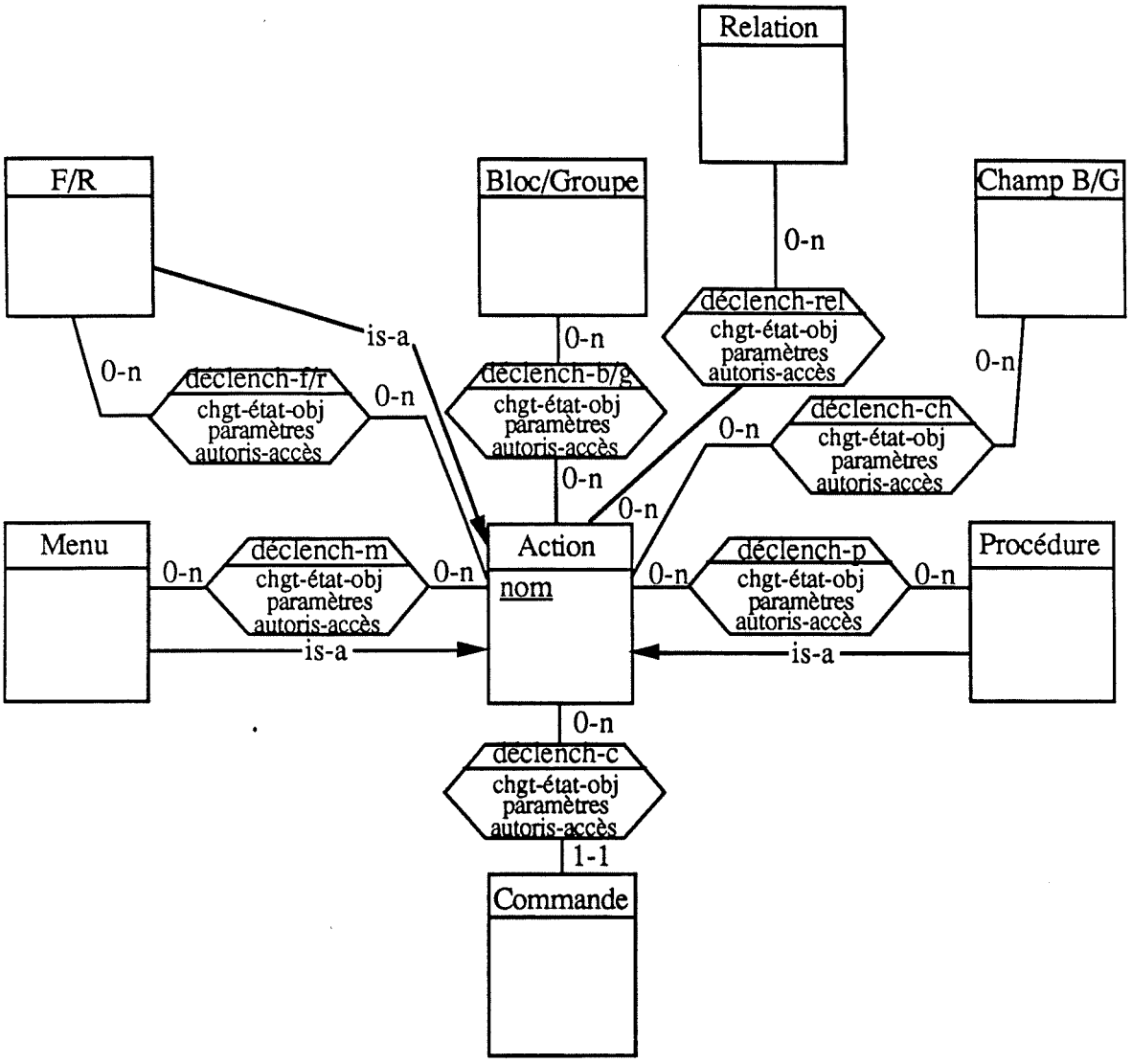


Figure IV.1. : Modélisation du composant "Pilotage"

Les **PRIMITIVES** de gestion du pilotage permettent de définir les interactions entre Base de données et Interface, entre Base de données et Traitements et entre Interface et Traitements :

- déclenchement (éventuellement conditionnel) d'une action
  - via une Relation de la base de données (insertion, modification, suppression d'un record d'une relation);
  - via un F/R (à l'activation, à la désactivation d'un F/R; avant, après query);
  - via un bloc/groupe d'un F/R (à l'activation, à la désactivation d'un bloc/goupe);
  - via un record d'un B/G d'un F/R (à l'activation, à la désactivation d'un record);

- via un champ d'un bloc/groupe d'un F/R (à l'activation, à la désactivation d'un champ);
- via un menu (à l'activation d'un menu; au choix d'un menu);
- via une commande-utilisateur;
- via une procédure.

IV.3. SYNTHESE DE LA MODÉLISATION DES E4GS AVANCÉS

Le schéma global de la figure IV.2 réunit en synthèse les principaux objets dont dispose le développeur dans un E4G avancé, orienté développeurs et applications opérationnelles. Il correspond à la construction d'un schéma brut en prenant compte des éléments nouveaux contenus dans chaque sous-schéma des 4 composants que nous avons décrit : la BD (cfr III.3.1), l'Interface (cfr III.3.2), les Traitements (cfr III.3.3) et le Pilotage (cfr IV.2). Nous avons cependant commencé par le sous-schéma du composant Pilotage. L'entié 'Action' semble en effet être l'élément central de notre modélisation globale, puisque reliée par des associations à des entités de tous les autres composants d'un E4G.

Le sous-schéma du composant Pilotage comprend donc l'entité **'Action'**. Une action peut être un Menu, un F/R ou une Procédure. Une action peut être déclenchée via la BD (avant/après l'insertion, la modification ou la suppression d'un record), via l'Interface (à l'activation/désactivation du F/R, du B/G ou du Champ d'un B/G; à l'activation ou au choix d'un Menu; à la survenance d'une commande-utilisateur) ou via les Traitements (au sein d'une procédure).

Le sous-schéma correspondant au composant BD comprend les entités **'BD'**, **'Relation'** et **'Champ d'une relation'**. La relation peut déclencher une action avant/après l'insertion, la modification ou la suppression d'un record.

Le sous-schéma correspondant au composant Interface comprend les entités **'Formulaire/Rapport'**, **'Bloc/Groupe'**, **'Champ d'un B/G'**, **'Menu'** et **'Commande'**. Le B/G d'un F/R correspond à aucune ou plusieurs Relations du composant BD; le Champ d'un B/G correspond au plus à un Champ d'une relation. Un B/G peut être lié à d'autres B/G. Chacune des entités de l'interface est susceptible de déclencher une action : le F/R, le B/G ou le Champ d'un B/G, à leur activation/désactivation; le Menu, à son activation ou au choix d'une de ses options; et, enfin, à la survenance d'une commande-utilisateur. Le changement d'état d'une entité de l'interface entraîne souvent le changement d'état d'autres entités de l'interface. Par exemple, l'activation d'un F/R précède l'activation de son premier B/G et du premier Champ de ce B/G. Les entités Menu et F/R peuvent elles-

mêmes constituer une action.

Le sous-schéma correspondant au composant Traitement des données comprend l'entité **'Procédure'**. La procédure manipule les entités Relation, Champ d'une relation et Champ d'un B/G. La procédure peut déclencher en son sein des actions. Les associations 'appel-pr' et 'appel-F/R' de la figure III.3 n'ont donc plus de raison d'être (la procédure déclenche une action qui peut être une autre procédure ou un F/R). Elle-même constitue une action. Ceci permet de modéliser l'aspect procédural du traitement des données. Afin d'en modéliser l'aspect non-procédural, nous avons ajouté l'attribut 'Expression' à l'entité 'Champ d'un bloc/groupe'. Cet attribut permet de spécifier le calcul déclaratif d'un champ.

IV.4. SPÉCIFICATION DE QUATRE E4GS DANS LE MODELE DESCRIPTIF

A présent que les concepts et fonctionnalités communs aux E4Gs avancés, orientés développeurs et applications opérationnelles ont été recensés au sein d'un modèle descriptif, nous pouvons décrire les environnements spécifiques qui serviront de base à la suite de notre étude : Magic, Quatrième Dimension, SQL*Forms et Rally.

Si ces quatre environnements appartiennent bien à la classe d'E4Gs modélisée, s'ils mettent en oeuvre les mêmes concepts et les mêmes fonctionnalités, ils se distinguent toutefois les uns des autres par la manière de combiner ces concepts, de les assembler pour réaliser les fonctionnalités de l'environnement, par les primitives qu'ils offrent au développeur. En cela, ils imposent chacun leurs restrictions (ou offrent des enrichissements) vis-à-vis de la modélisation globale. Ce sont ces restrictions (ou enrichissements) que nous étudions à présent pour chacun d'eux.

Le modèle descriptif réalise ici sa finalité. Il joue le rôle de grille d'analyse, de modèle de référence pour décrire les caractéristiques et les fonctions des E4Gs particuliers. Nous présentons successivement Magic, Quatrième Dimension, SQL*Forms et Rally.

IV.4.1. Magic

La Base de Données de l'environnement Magic met en oeuvre la plupart des objets des Bases de Données Relationnelles : la Relation, le Record (ligne) de la relation, le Champ (colonne) de la relation, la Clé. La terminologie utilisée ('Fichier' au lieu de 'Relation', 'Enregistrement' au lieu de 'Record') ne doit pas porter à confusion; les accès autorisés aux données sont bien ceux d'un système de gestion de base de données relationnel : accès par

clé à un record ou une séquence (éventuellement filtrée) de records.

Les concepts de Formulaire/Rapport, de Bloc/Groupe, de Procédure et de Menu du schéma global sont réunis au sein d'un concept central : la **Tâche**.

La Tâche abrite des **Opérations** sur une sélection de données de la base de données. L'ensemble des données manipulées par la tâche peut faire éventuellement l'objet d'une présentation à l'utilisateur (les 'Ecrans' de la tâche); un écran est toujours lié à une tâche. A cette fin, l'environnement Magic offre un éditeur d'écran, outil de formatage, permettant de localiser avec précision les champs sur l'écran. Magic offre également une gestion des états de sortie, avec ruptures, etc.

La philosophie de l'environnement Magic est claire : *le développement d'une tâche ne nécessite aucun langage de programmation.* Les opérations de la tâche sont toutes exprimées en mode déclaratif. Elles sont spécifiées une seule fois et sont exécutées pour chaque record de la tâche. Elles rejoignent les primitives du modèle descriptif global des E4Gs énumérées en III.3.1.2, III.3.2.2, III.3.3.2, III.3.4.2 et IV.2.2 :

- *Sélection d'un champ.* Cette opération met un champ à disposition de la tâche.
  Ce champ peut être le champ d'une relation. Dans ce cas, la sélection de ce champ permet l'accès à ce champ et la modification éventuelle de la valeur de ce champ. Il faut signaler qu'il n'existe aucune autre possibilité de manipuler un champ d'une relation; tout accès aux champs d'une relation doit s'effectuer via les champs d'une tâche.
  Le champ d'une tâche peut être également une Variable (un 'Champ Virtuel'), dont la durée de vie est limitée à la tâche.
  L'un et l'autre, champ réel et champ virtuel, peuvent apparaître dans les Ecrans liés à la tâche. Ils peuvent donc être saisis et présentés. Il peut leur être associé une expression de Sélection ou des expressions Minimum et Maximum limitant (tel un filtre) les records participant à la tâche.
- *Réalisation d'un lien.* Le lien est la concrétisation dynamique de la jointure entre deux relations. Une tâche opère sur une relation principale et sur autant de relations liées qu'il est nécessaire. Le lien s'effectue grâce à la mise en correspondance de deux champs de même type appartenant, l'un à la relation principale, l'autre à la relation liée. Chaque champ sélectionné au sein d'un lien peut être manipulé comme tout champ de la relation principale. Il faut noter que Magic offre, en sus, une primitive de recalcul des liens, c'est-à-dire que le lien établi est automatiquement recalculé dès modification d'une variable d'une des expressions du lien. Cela signifie que Magic permet la modification de tous les champs sélectionnés dans le F/R. Il offre par conséquent une gestion totale des vues, ce qui constitue un enrichissement du modèle descriptif global. Rappelons que la majorité des E4Gs n'offre pas cette possibilité (cfr III.3.2.1).

- *Validation d'un champ*. Cette opération permet de valider les manipulations effectuées sur ce champ (saisie, calcul, ...). Elle effectue un contrôle sur la valeur du champ uniquement.
- *Calcul d'un champ*. La puissance de Magic réside dans l'opération de calcul déclaratif d'un champ. Elle vaut principalement par les deux propriétés qui lui sont associées :

  Le recalcul automatique. En cours d'exécution de la tâche, toute modification d'un des paramètres de l'expression entraîne immédiatement le recalcul du champ concerné par cette expression.

  Le calcul "incrémentiel". L'opération 'Calcul d'un champ' peut s'effectuer sous un mode "incrémentiel". Le champ soumis à l'opération (le champ 'calculé') est alors incrémenté algébriquement en fonction de l'expression.
  Dans ce cas, le champ calculé est également soumis au recalcul automatique, en ce sens qu'il reflète sur le champ toute modification apportée à l'expression associée à ce champ.
- *Exécution d'une tâche.* Une tâche peut commander l'exécution d'une autre tâche. Cette nouvelle tâche est une sous-tâche de la première. Elle est elle-même liée à une relation principale. Les différentes tâches et sous-tâches s'identifient en quelque sorte à ce que nous avons nommé, dans le modèle général, les Blocs ou Groupes. Les sous-tâches (les tâche 'filles') sont hiérarchisées au sein de la tâche mère, de la même manière que les blocs/groupes au sein d'un F/R. Comme le spécifie le modèle descriptif global, une sous-tâche peut manipuler l'ensemble des champs de la tâche dont elle dépend.
- *Exécution d'un programme.* Une tâche peut commander l'exécution d'un autre programme Magic. Ce programme est une tâche indépendante de la première. Etant donné que cette tâche peut être 'batch' ou interactive, cette opération correspond, dans le modèle général, respectivement aux primitives de déclenchement d'une procédure et de déclenchement d'un formulaire/rapport.

Ces opérations peuvent être déclenchées pendant la saisie d'un record participant à la tâche (Corps d'Enregistrement); elles sont alors liées à un changement d'état d'un champ de la tâche : la validation est liée à la désactivation du champ, le recalcul automatique à la désactivation d'un champ modifié, etc. Elles peuvent également être déclenchées après la saisie du record (Suffixe d'Enregistrement), ainsi qu'avant ou après l'exécution de la tâche (Préfixe ou Suffixe de tâche). Ces opportunités équivalent à celles d'un déclenchement d'action du modèle général, respectivement après un Bloc ou Groupe et avant ou après un F/R (tâche interactive) ou une procédure (tâche batch).

Aux opérations de la tâche sont liées des paramètres. Tous, nous l'avons vu, sont des **expressions**; qui de lien, qui de calcul, qui encore de validation. L'opération est

éventuellement soumise à une autre expression, qui traduit la condition de déclenchement de l'opération. Elle permet notamment de déterminer l'autorisation d'accès ou la condition de calcul associée à un champ de la tâche (lecture seulement, en mode modification uniquement...), ou encore de permettre le déclenchement optionnel d'une procédure ou d'une autre tâche.

Toutes ces expressions sont réunies au sein d'une table, la Table des expressions, accessible à tout endroit de la tâche. Il suffit de remplir les tables d'exécution (table des opérations, table des expressions, ...).

Chaque tâche effectue donc des opérations sur une sélection de données, record par record, ces données faisant éventuellement l'objet d'une saisie ou d'une présentation à l'utilisateur.

De plus, Magic gére les paramètres de tâche que sont les autorisations d'accès ainsi que le mode initial d'utilisation de la tâche.

Enfin, l'environnement offre une gestion des menus principaux, c'est-à-dire ceux qui lancent les tâches initiales. Le menu ne constitue un objet de l'environnement que dans ce cas précis. Si le développeur veut déclencher un menu en cours de tâche, il doit le simuler au sein d'une tâche dédiée à cet effet.

Cette limite par rapport au modèle général est accompagnée de deux **restrictions** supplémentaires :

- l'impossibilité pour le développeur de définir des commandes d'application. L'utilisateur a toutefois à sa disposition des commandes prédéfinies de déclenchement de tâche ('Zoom'), d'effacement, d'annulation, etc.
- l'impossibilité aussi de déclencher une action directement, via une relation de la base de données. Le déclenchement doit, dans ce cas, être transféré au sein de toutes les tâches liées à cette relation.

En rapprochant ses concepts de ceux du modèle descriptif global, l'environnement Magic est représenté graphiquement à la figure IV.3.

IV.4.2. Quatrième Dimension

La base de données de Quatrième Dimension (la "structure") est constituée de "fichiers" comportant des "rubriques". Une rubrique peut être de type alphanumérique, texte, numérique, entier, entier long, date, image et racine. Ce dernier type correspond à une rubrique répétitive et décomposable. C'est la raison pour laquelle nous avons défini l'association "sous-fichier" : à un fichier peut correspondre un sous-fichier (la rubrique de type racine); à un sous-fichier ne correspond qu'un (et un seul) fichier. Une rubrique de

type racine sera donc décrite dans notre schéma dans l'entité "fichier"; elle sera liée à son fichier "père" via l'association "sous-fichier".

Une rubrique peut être énumérée (la liste des valeurs possibles que peut prendre la rubrique est définie par le développeur; l'utilisateur devra choisir parmi cette liste lors de la saisie ou la modification de la rubrique), indexée, indexée unique, saisissable ou non, modifiable ou non, obligatoire ou non.

On peut associer à chaque fichier des "formats" ("formats page" ou "formats liste") qui seront utilisés pour la saisie, la modification ou la visualisation des "fiches". Les "formats page" affichent une seule "fiche" par écran, tandis que les "formats liste" affichent une liste de "fiches" par écran.

Outre un générateur automatique de formats, un éditeur graphique permet de créer les formats manuellement par manipulation directe.

Un format correspond au plus à un fichier, mais il peut comporter des rubriques d'autres fichiers. Ces vues ne permettent néanmoins que de visualiser ces rubriques : elles ne peuvent être saisies ou modifiées que via leur propre format.

Les formats peuvent également contenir des objets interactifs tels que boîtes de dialogue et d'alerte, boutons (boutons de contrôle, boutons radios ou boutons commutateurs), zones de défilement, menus, etc.
La notion de "format" consiste en la réunion du "Formulaire/Rapport" et du "Bloc/Groupe" modélisés en III.3.2.2. Il constitue une action et peut être lié à des sous-formats; ces derniers sont soit des formats à part entière définis sur d'autres fichiers soit des formats définis sur les rubriques de type "racine" du fichier correspondant au format même.

A chaque format et à chaque fichier est associée une (et une seule) "formule" (la "formule format" et la "formule fichier").

La "formule format" est exécutée à l'activation/désactivation du format et à chaque fois que l'utilisateur modifie une rubrique du format, qu'il clique sur un bouton ou qu'il sélectionne une ligne de menu. Ces "formules format" ne sont pas directement liées à un changement d'état d'un objet particulier comme décrit en IV.2.1. Toutes les actions susceptibles d'être déclenchées par les objets du format ou par le format lui-même s'y retrouvent. Il faut donc, à l'intérieur de la formule, identifier l'événement déclencheur à l'aide de fonctions de contrôle de saisie. Par exemple, la fonction "avant saisie" répond "vrai" à l'activation du format; la fonction "après saisie" répondra "vrai" à sa désactivation; la fonction "pendant saisie" répond "vrai" à chaque modification d'une rubrique du format ou à chaque action effectuée sur un objet interactif. Pour connaître quelle est la rubrique modifiée, la fonction "modifie (nom de la rubrique)" répond "vrai" si la rubrique "nom de la rubrique" a été modifiée; pour identifier l'objet interactif sur lequel l'utilisateur a effectué une

action, il faut tester l'état de l'objet.

Les actions de Quatrième Dimension sont les mêmes que celles définies en IV.2.1 (F/R, Menu, Procédure). Ces actions, comme nous venons de l'expliquer, ne sont pas directement associées à la survenance d'un événement particulier; elles ne peuvent être déclenchées qu'au sein des formules. Il incombe donc au développeur de définir l'ensemble des tests permettant d'exécuter la bonne partie de la formule en réponse à l'événement déclencheur, c'est-à-dire définir lui-même la gestion du déclenchement d'action.

La "formule fichier" est exécutée aux mêmes changements d'état que ceux de la "formule format". Elle est définie pour tous les formats correspondant à ce fichier. Il ne s'agit donc pas, comme on pourrait le penser, d'une formule exécutée à l'insertion, modification ou suppression d'un record : la formule fichier est utilisée pour réunir les actions qui doivent être déclenchées pour tous les formats correspondant à ce fichier. Les "formules fichier" doivent être structurées à l'aide des mêmes fonctions de contrôle de saisie que celles décrites pour les "formules format". Elles sont toujours exécutées avant les "formules format".

Par rapport à notre modélisation faite en IV.2.1, les changements d'état des objets sont :

- liés à l'interface :
    - à l'activation/désactivation d'un format (qui regroupe, rappelons-le, la notion de F/R et de B/G du schéma global)
    - à la modification de la valeur d'une rubrique
    - au choix d'un menu
    - à la survenance d'une commande-utilisateur
- liés aux traitements :
    - au sein d'une formule ou d'une procédure.

Il n'y a aucun changement d'état lié à la BD.

Les concepts de Quatrième Dimension sont synthétisés à la figure IV.4.

IV.4.3. SQL*Forms

SQL*Forms n'est pas à proprement parler un E4G : il n'est en fait qu'un outil s'intégrant au SGBD Oracle et à une série d'autres outils : SQL*Plus (interrogations), SQL*Calc (tableur), SQL*Report (rapports), Pro*C, Pro*Cobol, Pro*Fortran, Pro*Pascal,...(interfaces C, Cobol, Fortran, PL/1, Pascal,...).
Cet ensemble d'outils possède l'ensemble des fonctions décrites en III.3 et peut donc être

considérée comme E4G.

Nous avons néanmoins étudié plus particulièrement SQL*Forms car c'est cet outil qui permet de donner au tout les caractéristiques spécifiques d'un "E4G avancé" évoquées en IV.2.

Nous avons relevé très peu de différences entre SQL*Forms et la modélisation des E4Gs avancés.

Par rapport à la saisie et la présentation des données, signalons seulement que SQL*Forms ne possède pas d'entité 'Menu'. Cette restriction peut néanmoins être facilement levée en utilisant un Formulaire ('Form') comprenant un "Control block", c'est-à-dire un 'Block' qui n'est associé à aucune relation de la BD et qui comporte des variables; ces dernières permettront de saisir le choix de l'utilisateur.

Par rapport au traitement des données, les procédures ("triggers") ne possèdent pas de paramètres.

Outre les changements d'état d'objets décrits en IV.2.1, SQL*Forms permet de déclencher une action à la modification d'un champ de l'interface (c'est-à-dire si la valeur du champ a été modifiée en valeur non nulle).

Les concepts de SQL*Forms sont synthétisés à la figure IV.5.

IV.4.4. Rally

L'E4G Rally propose des structures assez proches de celles du modèle global. La terminologie lui est bien entendu propre : Un champ d'une relation est un Relation Field, un Formulaire/Rapport est un Form/Report, un bloc/groupe est un Group, un champ d'un F/R est un F/R Field.

Rally permet au développeur de manipuler des données issues de différentes bases de données, notamment Rdb/VMS. Le développeur dispose de types standards simples (word signed, longword signed, f floating, character string, ...) ou dérivés (integer, real, text, money, date) pour la définition des champs de ses relations. Il peut également spécifier lui-même des types particuliers ('global field definition'). Il définira par exemple une seule fois le type 'Numéro', qui sera le type des champs 'Numéro de client', 'Numéro de produit' et 'Numéro de commande' de sa BD.

Conformément à la modélisation d'un E4G, le Formulaire/Rapport ('Form/Report') comprend des blocs/groupes ('groups') qui peuvent être hiérarchisés pour former une vue sur les relations de la base de données. Le champ d'un 'group' peut être par définition :

- un 'Data field' : il correspond alors à un champ d'une relation,
- un 'Aggregate Field' : il constitue alors l'agrégation d'autres champs du F/R, c'est-à-dire

la somme, la moyenne, le nombre, le minimum ou le maximum de ces champs,
- un 'Computed Field' : il est alors lié à une procédure de calcul, ou
- un 'Variable Field' : il constitue une variable.

Cette définition d'un champ relève totalement d'un mode de traitement déclaratif. Toujours en mode déclaratif, la validation d'un champ d'un B/G peut s'effectuer sur base d'une liste de valeurs ou par rapport à des valeurs minimum ou maximum. Pour des manipulations (validation, calcul, ...) plus complexes des champs d'un 'group', Rally contient un langage procédural (Application Development Language) qui possède les caractéristiques d'un langage de programmation classique, tel Pascal (procédures, fonctions, structures de contrôle, ...). Rally permet également de manipuler, directement, les relations de la base de données, sans passer par les champs d'un F/R, grâce à un sous-langage (Data Manipulation Language) constitué de primitives pré-définies de manipulation de données (DB_QUERY, DB_OPEN, DB_GET_FIRST, ...).

La restriction structurelle la plus importante de Rally vis-à-vis du modèle descriptif global se situe au niveau de la hiérarchisation des 'groups' : si un B/G constitue la jointure de deux relations d'une base de données, il n'est accessible qu'en lecture. Aucun de ses champs ne peut être saisi ou modifié. Si l'on désire que ces manipulations soient permises, il faut constituer deux B/G distincts (pour chacune des relations) et les lier alors au sein du F/R.

Les autres différences structurelles retenues par rapport au modèle descriptif global des E4Gs constituent davantage des enrichissements que des restrictions vis-à-vis de ce modèle. Les structures nouvelles proposées par l'environnement Rally englobent généralement les anciennes dans une optique de réutilisabilité :
- le *Data Source Definition (DSD).* Ce concept nouveau s'intercale entre les concept de relation et de formulaire/rapport. Le DSD extrait de la relation à laquelle il est associé les champs qui vont être utilisés par le F/R.
  Une relation peut avoir plusieurs DSD. Cela permet de définir, par exemple, différents formats, différentes autorisations d'accès (share write, read only, ...), différents ordres de tri pour les champs d'une relation.
  Un DSD peut être lié à plusieures relations et définir en quelque sorte une vue basée sur ces relations.
- le *Form/Report Packet.* Cet objet encadre le formulaire ou le rapport. Il permet de définir différents paramètres associés à ce F/R : mode initial d'utilisation (mode créaction, modification, recherche, suppression), autorisation d'accès (pas de query, insertion uniquement,...). Un F/R peut avoir plusieurs Form/Report Packet. Le déclenchement d'un F/R équivaut en fait au déclenchement d'un F/R packet particulier.
- le *Parameter Packet.* Dans cette même optique de réutilisabilité qui caractérise les objets

de l'environnement Magic, une procédure peut s'adjoindre des paramètres. La correspondance entre paramètres formels et paramètres actuels s'établit au sein du Parameter packet.

Les principales primitives offertes par l'environnement Rally sont semblables à celles du modèle général. La nuance se situe dans les possibilités de déclenchement d'action que permet Rally :

- la relation ne peut déclencher aucune action; les déclenchements éventuels (avant, après insertion d'un record, ...) doivent être associés à tous les groupes des F/R liés à la relation);
- les déclenchements liés à l'activation et la désactivation d'un F/R sont, dans Rally, liés à l'activation et la désactivation du F/R Packet;
- les changements d'état d'objets pouvant déclencher une action sont également plus nombreux dans Rally :
  - au sein des F/R : avant, après commit des records dans la BD;
    avant, après le changement de valeur d'un champ;
    avant, après un rollback;
  - au niveau des menus : avant, après le choix d'une option du menu.

Enfin, Rally utilise pleinement le principe des options par défaut afin d'aider le développeur à construire son application, à décrire les différents objets qui la constituent. Des générateurs de Formulaire/Rapport, de DSD, de Relation (Building Tools) se chargent de générer l'application sur base de ces descriptions. Le développeur peut toutefois contrecarrer les choix de l'environnement, notamment en termes de formats d'écrans ou d'états, en refusant d'utiliser les outils mis à sa disposition ou en modifiant a posteriori les entités créées.

Les concepts de l'environnement Rally sont synthétisés à la figure IV.6.

IV.5. CONCLUSIONS

L'ensemble des considérations énoncées dans ce chapitre concernant l'aspect dynamique de l'E4G nous suggère plusieurs conclusions quant à la particularité de ces environnements par rapport aux environnements traditionnels.

D'abord, cette particularité des E4Gs tient en quelques caractéristiques essentielles : interactivité, convivialité de l'utilisation et du développement, souplesse accrue du développement et du pilotage de l'application, ...

L'apport essentiel de tels environnements se situe, en fait, au niveau de cette

diversification d'actions déclenchables. Rappelons-le : une fonctionnalité de l'application peut être supportée par un formulaire ou un rapport, sans nécessiter l'intervention d'une procédure. Ceci nous suggère la constatation suivante : il semble désormais impossible d'isoler l'interface-utilisateur des fonctionnalités d'une application interactive, la dynamique de l'application (les déclenchements d'actions) étant intégrée au sein de chaque objet de l'application. Ce propos entre, semble-t-il, en contradiction avec celui des concepteurs de méthodologies de développement qui basent leurs travaux sur la séparation modulaire entre les fonctionnalités de l'application et son interface-utilisateur. Nous approfondirons cette réflexion après avoir expérimenté l'une de ces démarches dans les environnements considérés.

Enfin, il reste à évoquer les performances d'exploitation liées aux E4Gs, notamment au niveau de deux facteurs, le temps de réponse et la consommation CPU. Il est bien connu que l'efficacité optimale s'obtient par un bon compromis entre la puissance de traitement des matériels mis à disposition et la simplicité du logiciel qui va généralement en sens inverse de la puissance du matériel.

Les E4Gs les plus perfectionnés, notamment ceux que nous avons étudiés dans ce chapitre, ne sont généralement pas les plus performants. En effet, les qualités de portabilité et d'ouverture qui sont recherchées par ces environnements ainsi que le choix d'interactivité vont presque toujours de pair avec une sous-optimisation de leurs performances d'exploitation. Il s'agit d'un choix : favoriser la souplesse d'emploi et la convivialité de l'interface avec l'utilisateur aux dépens des performances d'exploitation. Toutefois, ce problème d'efficacité ne doit pas être sous-estimé : dès qu'il s'agit, au-delà d'un simple prototypage d'application, de réaliser et d'exploiter en temps réel des applications lourdes sur des volumes de données très importants, traditionnellement réservées au traitement par lots, les performances des E4Gs chutent.

"Cependant, faisons attention de ne pas fausser le problème : c'est la performance globale de la 4ème génération qui peut faire son succès. Il faut savoir apprécier l'efficacité globale sur l'ensemble des ressources mises en oeuvre, y compris humaines, et sur l'ensemble du cycle de vie des applications, y compris leur maintenance et leur évolution. Sur ce plan, il semble que l'efficacité des E4Gs soit infiniment meilleure que celle des outils traditionnels" [Bou 84].

Voilà qui clôture l'étude des E4Gs avancés, orientés développeurs et applications opérationnelles. Ces environnements, nous les avons expérimentés, du moins trois d'entre eux (Magic, Quatrième Dimension et Rally), en appliquant une méthode de conception classique, à savoir la méthode IDA (analyse conceptuelle), étendue à la conception logique et physique de l'application. Le compte-rendu de cette expérience est présenté au chapitre VI,

non sans qu'auparavant aient été rappelés les principaux concepts soutenant la méthode de conception utilisée (chapitre V). Cette expérience donnera lieu à des critiques ou des réflexions au sujet de l'(in)adéquation d'une méthode de conception classique et le développement d'une application interactive dans un E4G (chapitre VII). Sur base de notre expérience, sur base des critiques énoncées, mais également à l'aide du modèle descriptif global décrit aux chapitres III et IV, nous tenterons de dégager les mécanismes, les concepts méthodologiques susceptibles de soutenir une démarche de conception "orientée E4Gs" (chapitre VIII).

Chapitre V

Une Démarche de conception

V.1. INTRODUCTION

Dans ce chapitre, nous allons présenter une démarche méthodologique classique de conception d'applications interactives. Cette démarche, nous l'avons suivie lors de l'expérimentation des E4Gs envisagés (cfr Chapitre VI).

Elle s'étend sur 3 niveaux d'abstraction :

- le niveau conceptuel pour lequel elle épouse la démarche de la méthode IDA (Interactive Design Approach);
- le niveau logique pour lequel elle marie la conception d'un schéma relationnel pour les données de l'application à la conception d'une machine 'phase' (machine abstaite indépendante de l'environnement envisagé) gérant les traitements. Elle s'attache aussi à la conception du dialogue homme/machine;
- le niveau physique pour lequel elle traduit la solution abstraite en solution concrète, dans l'environnement de programmation envisagé.

V.2. LE NIVEAU CONCEPTUEL

Le niveau conceptuel a pour but de mettre en évidence les besoins exprimés dans l'organisation et de présenter une solution indépendante des moyens de réalisation.

Les fonctionnalités de l'application peuvent se représenter par différents niveaux de détails. La démarche IDA privilègie la décomposition fonctionnelle d'un SI pour raffinements successifs.

Le concept de "phase" constitue le repère central de cette décomposition par la signification qu'il présente sur le plan informationnel. "La phase est un lien d'identification de structures homogènes de données et de règles de traitements" [Bod 88a]. Nous nous limiterons, dans la démarche de conception, à la description de la phase.

Dans la méthode IDA, l'analyse conceptuelle de la phase comprend les étapes d'élaboration du schéma conceptuel des informations et d'élaboration du schéma conceptuel des traitements.

Il est à noter que la spécification conceptuelle du dialogue est englobée dans la spécification des traitements.

V.2.1. Les données

V.2.1.1. Elaboration du schéma conceptuel des informations

On élabore d'abord le contenu informationnel de la phase sous la forme d'un texte structuré composé de phrases élémentaires[1] exprimées sous une forme abstraite[2].

Ensuite, on produit un modèle Entité/Association par transformation du texte structuré, en s'appuyant éventuellement sur une étape d'interview des utilisateurs pour obtenir les informations manquantes, notamment certaines contraintes de connectivité et d'identification.

Le formalisme du modèle E/A a été rappelé en III.2.

V.2.1.2. Mise sous forme canonique

"La cohérence du modèle E/A est définie par l'absence de contradiction dans les spécifications et par la conformité de celles-ci à une **forme canonique** du schéma. Cette forme canonique est caractérisée par l'**élimination** ou le **contrôle de la redondance** et par l'élimination des **ambiguïtés**; elle a pour but la production d'un schéma conceptuel des informations aussi significatif et aussi stable que possible" [Bod 88a].

V.2.2. Les traitements

V.2.2.1. Décomposition de la phase en fonctions et spécification des fonctions

"L'objectif de cette étape est de modéliser, de façon détaillée, les règles de traitement d'une phase. Cela revient à mettre progressivement en évidence les fonctions de la phase, à décrire les messages qu'elles reçoivent et génèrent, enfin à préciser les actions élémentaires de chacune d'entre elles" [Bod 88a].
Cette décomposition se fait selon une démarche guidée par les données, démarche constructive et déductive : à partir des résultats à atteindre, on dégage de proche en proche les

---

1 "Une phrase élémentaire est un système de propositions et de mots qui ne peut pas être décomposé en constructions plus courtes sans perte de sens : elle est sémantiquement irréductible" [Bod 88a].

2 Une phrase élémentaire est exprimée sous forme abstraite si "elle se rapporte à des classes de faits" [Bod 88a].

fonctions en se rapprochant des données de la phase. Lorsqu'une fonction est mise en évidence, on précise ses résultats, on décrit son comportement interne et on dégage ce qu'elle suppose recevoir en entrée (données et contraintes). Le processus se termine quand les entrées des fonctions correspondent à celles de la phase.

V.2.2.2. Schéma de la dynamique des fonctions

Après la description statique des fonctions de la phase, il s'agit de spécifier l'enchaînement de ces fonctions afin de fournir une compréhension rigoureuse du comportement précis et détaillé de la phase.
Cette spécification s'effectue en construisant un schéma basé sur le modèle de la dynamique des traitements, "modèle causal dans lequel un **événement** est un stimulus auquel le système d'information réagit, le plus souvent, par le déclenchement de **processus** ou traitements qui peuvent à leur tour causer la survenance d'autres événements" [Bod 88a].

La survenance d'un événement peut correspondre :
- à la génération d'un message
- à la terminaison d'un processus
- à la réalisation d'un point de synchronisation[1].

Ce modèle permet de spécifier les conditions d'exécution et d'enchaînement des traitements. Ces différents enchaînements peuvent être :
- l'enchaînement **séquentiel**, lorsque chaque processus d'un type donné précède systématiquement un processus d'un autre type;
- l'enchaînement **parallèle**, lorsque chaque processus d'un type donné provoque le déclenchement simultané de plusieurs processus de types différents;
- l'enchaînement **convergent**, lorsque des processus de plusieurs types provoquent chacun et séparément le déclenchement d'un autre processus;
- l'enchaînement **synchronisé**, lorsque le déclenchement d'un processus suppose une attente préalable résultant de la survenance d'au moins deux événements, issus de traitements parallèles en amont;
- l'enchaînement **groupé**, lorsqu'un ensemble de processus de même type doivent être terminés avant de déclencher un autre traitement;
- l'enchaînement **conditionnel**, lorsqu'un processus provoque sélectivement le

---

[1] "Le mécanisme de synchronisation est utilisé pour représenter une situation dans laquelle il faut attendre la survenance d'une combinaison de deux ou plusieurs événements avant de provoquer une réaction particulière dans le système d'information" [Bod 88a].

déclenchement d'un autre processus en fonction d'une condition;
- l'enchaînement **multiple**, lorsque chaque processus d'un type donné provoque le déclenchement simultané de plusieurs processus de même type.

V.2.3. Le dialogue

Une règle fondamentale respectée dans la décomposition d'une phase prévoit qu'une fonction se déroule sans interaction avec l'utilisateur. Cela signifie qu'à condition de lui fournir les informations adéquates en entrée, une fonction doit pouvoir s'exécuter sans interaction supplémentaire avec l'utilisateur et lui fournir, au terme de son exécution, les informations souhaitées. Les fonctions sont alors vues comme des services à disposition de l'interface. C'est à ce dernier qu'incombe la tâche de collecter les données auprès de l'utilisateur et de gérer l'enchaînement des fonctions, c'est-à-dire le dialogue de l'application.

Au niveau conceptuel, le dialogue est donc spécifié par la statique et la dynamique des traitements. La statique permet de connaître les messages fonctionnels qui ont été définis et l'objectif des différents traitements; la dynamique les conditions de déclenchement et d'enchaînement de ces traitements.

La conception du dialogue n'est abordée qu'au niveau logique, consécutivement aux spécifications statique et dynamique de l'analyse fonctionnelle, qu'elle doit respecter.

V.3. LE NIVEAU LOGIQUE

La conception logique conduit à la définition d'une solution qui soit "exécutable" par une machine abstraite, strictement indépendante des machines réelles.

V.3.1. Les données

En ce qui concerne la spécification logique et physique des structures de données, la démarche de conception présentée dans ce chapitre est dérivée de la démarche proposée dans [Hai 86]. Elle s'en différencie par le fait qu'elle concerne uniquement la conception d'une base de données relationnelle, eu égard aux caractéristiques intrinsèques du composant base de données des E4Gs. La spécification relationnelle de la base de données relève donc, dans notre démarche, du niveau logique, et non du niveau physique, puisqu'il n'est pas question, dans un E4G, de base de données hiérarchique ou en réseau.

La démarche de conception logique fait usage d'un modèle spécifique, apte à la fois à

représenter la sémantique d'un schéma conceptuel et à décrire avec précision l'organisation des données du point de vue des accès techniques : le Modèle d'Accès Généralisé (MAG).

"Ce modèle propose un jeu de concepts proche de la pratique des fichiers et des bases de données sans cependant être spécifique d'aucun d'entre eux. (...) Les concepts essentiels du MAG sont ceux d'article et de type d'articles, d'item d'un article, de (type de) chemins d'accès inter-articles, de clé d'accès et d'ordre d'une séquence d'articles (chronologique, anti-chronologique, programmé, trié, ...)" [Hai 86].

La démarche comprend d'abord la dérivation d'un schéma MAG à partir du schéma conceptuel. Cette dérivation s'effectue selon les règles suivantes [Hai 86] :

- une entité est représentée par un article, un type d'entités par un type d'articles;
- une valeur d'attribut est représentée par une valeur d'item et un attribut par un item associé au type d'articles représentant le type d'entités; la répétitivité, la décomposabilité et le caractère facultatif de l'item découlent immédiatement des propriétés de l'attribut;
- un type d'associations binaire sans attribut est représenté par un type de chemins entre les types d'articles correspondants; sa classe fonctionnelle[1] et les contraintes d'existence se déduisent de la connectivité du type d'associations;
- un type d'associations au moins ternaire, sans attribut, donne lieu à un type d'articles. Le type d'articles sera lié à chacun des types d'articles correspondant aux membres du type d'associations par un type de chemins N-1, obligatoire pour le nouveau type d'articles;
- un type d'associations possédant des attributs, même s'il est binaire, est transformé en un type d'articles. Chaque attribut est représenté par un item attaché au nouveau type d'articles. Si le type d'associatons est binaire et de connectivité (i-1, k-*), le nouveau type de chemins concernant le type d'articles du côté i-1 sera de classe fonctionnelle 1-1;
- les identifiants éventuels de chaque type d'articles sont déterminés en fonction des propriétés des types d'entités et d'associations qu'il représentent. En particulier, si le type d'associations transformé en type d'articles a pour identifiant l'ensemble des rôles que jouent les entités membres, alors le type d'articles a pour identifiant les types d'articles représentant ces membres, via les nouveaux types de chemins;
- l'expression MAG des autres contraintes d'intégrité du schéma E/A sera établie de manière semblable.

Cette dérivation débouche sur un Schéma des Accès Possibles (SAP). La démarche prévoit, à ce stade, une optimisation de la stratégie d'accès du point de vue des accès à la

---

1 "La classe fonctionnelle d'un type de chemins caractérise le nombre maximum d'articles d'un membre que l'on peut associer à chaque article de l'autre membre. Les classes fonctionnelles possibles sont : 1-N (un à plusieurs), N-1 (plusieurs à un), 1-1 (un à un) et N-N (plusieurs à plusieurs)" [Hai 86].

base de données, selon le critère du nombre d'opérations logiques réalisées sur les données par les algorithmes de traitement. Cette optimisation ne nous intéresse pas ici puisque, d'une part, "la plupart des SGBD relationnels des E4Gs sont à même d'assurer une certaine optimisation d'accès relatifs à une requête relationnelle" [Hai 86] et, d'autre part, l'optimisation ne constitue pas une préoccupation au niveau de notre expérimentation. Nous nous en tiendrons donc au schéma des accès possibles. Le lecteur intéressé par de plus amples informations au sujet de l'optimisation des structures d'accès aux données pourra se référer à [Hai 86].

Le schéma des accès possibles n'est pas conforme au SGBD relationnel. Les restrictions des structures relationnelles par rapport à celles du MAG sont importantes : pas de type de chemins, items simples (non répétitifs) et élémentaires (non décomposables), items obligatoires, pas de contraintes d'intégrité autres que sur le type de valeurs d'un item, etc. Le stade suivant de la conception logique consiste, par conséquent, à rendre le schéma MAG conforme au relationnel, en supprimant notamment les types de chemins, les items répétitifs, les items décomposables, etc. Cette transformation passe par l'adjonction d'un nouvel item à un type d'articles pour référencer le type d'articles auquel il est lié, par la création d'un nouveau type d'articles pour supprimer la répétitivité d'un item, ou par l'éclatement d'un item décomposable en ses items constituants, ... Il est à noter que le concepteur sera parfois amené à créer un identifiant artificiel pour un type d'articles sans identifiant qui doit être référencé par un autre type d'articles. Il devra également veiller à relever soigneusement les contraintes d'intégrité (dites "référentielles" selon la terminologie relationnelle) engendrées par les transformations réalisées.

La traduction du schéma MAG en un schéma relationnel est alors, pour l'essentiel, immédiate : définition des tables (ou relations), de leurs colonnes (ou attributs) et du type des attributs (ou domaines) et spécification des contraintes référentielles ainsi que des identifiants. Nous maintenons, au niveau logique, la spécification des contraintes d'intégrité non prises en charge par le SGBD relationnel (contraintes dynamiques); ces contraintes doivent être assurées par les procédures logiques de mise à jour des données.
Les contraintes sont exprimées dans l'algèbre relationnelle. Les opérateurs essentiels de cette algèbre sont : l'union, l'intersection, la soustraction, le produit de relations, la projection, la sélection, la jointure de relations, ...

V.3.2. Les traitements

Il s'agit ici d'assurer une conversion des fonctions spécifiées au niveau conceptuel en procédures, exécutables sur machine "abstraite". La démarche présentée ici est issue de [Pig 89]. Elle tente de réaliser une conversion la plus systématique possible et une

architecture du logiciel la plus fidèlement calquée sur la décomposition des fonctions. Ceci nous amène à définir des règles nous permettant d'y parvenir.
La phase décrite dans l'analyse fonctionnelle est ainsi transformée en machine 'phase', dont l'état est décrit à l'aide d'une mémoire et les opérations autorisées sur cette mémoire sont représentées par des procédures et leurs arguments résultats.
La description de la machine 'phase' peut s'effectuer dans le formalisme de l'algèbre relationnelle.

"Cette première architecture pourrait alors faire l'objet d'un certain nombre d'ajustements ou de raffinements pour des motifs techniques de programmtion (réutilisation, préparation, optimisation, etc.). (...) Les services offerts par le module fonctionnel correspondraient toujours aux fonctions de la phase mais leur programmation pourrait varier car elle resterait invisible à toute utilisation du module fonctionnel, notamment par l'interface" [Bod 88a].

V.3.2.1. La technique de base

La transformation du schéma Entité/Association défini en V.2.1 en mémoire de la machine 'phase' (nous avons choisi la représentation relationnelle) a déjà été décrite en V.3.1.
Il nous reste à transformer les fonctions et leurs messages en entrée/sortie en procédures et leurs arguments/résultats. Cette transformation est simple et systématique. De plus, elle est conforme à notre objectif de continuité.

Chaque fonction nous donne une procédure; ses messages en entrée sont transformés en arguments de la procédure et ceux en sortie les résultats. Les arguments/résultats sont de type atomique si le contenu du message correspondant est limité à un seul attribut, de type 'record' si le contenu du message est constitué de plusieurs attributs, ou encore de type 'array' en cas d'attribut répétitif. Les contraintes d'intégrité liées aux messages deviennent les prérequis des procédures supposés être vérifiés avant leur exécution.
Enfin, l'objectif de la fonction et ses règles de traitement permettent de définir l'objectif de la procédure.

Exemple :

La fonction ***VALIDATION D'UNE PROVENANCE***

a pour objectif de garantir que le client n'est pas sur la 'liste noire';

génère une ***Provenance valide***

si *[la Catégorie du Client n'est pas 'douteux']*;

reçoit un ***Client valide***.

Un message ***CLIENT VALIDE***
comprend un ***Client***
[qui existe dans la mémoire];

est transformée en :

```
procédure   ValidationDuneProvenance (

{arg}       Cval     : ClientValide;
{rés}       var      Proval   : ProvenanceValide;
            var      cond     : boolean
            );
{pré        Client (Numero de client = Cval) ≠ ∅
            }
{obj        si Cval.Categorie ≠ 'douteux'
            alors
                     cond := VRAI
                     &
                     Proval:=Cval
            sinon
                     cond := FAUX
            }
```

Cette première transformation comporte néanmoins quelques inconvénients :

- la structure des arguments est très riche (Cval est de type record et comprend le numéro, le nom, l'adresse, la limite d'achat et la catégorie du client), alors que peu d'informations sont nécessaires à la procédure;
- les échanges de paramètres sont importants entre la machine 'phase' et l'interface;
- la recopie des données échangées est fréquente (Proval := Cval).

V.3.2.2. Première variante

La première variante à apporter à la technique de base consiste à limiter le contenu des arguments/résultats à l'identifiant de l'information véhiculée par le message (par exemple,

limiter le contenu de Cval à l'identifiant du client, à savoir le numéro de client) et à supprimer les résultats inutiles, c'est-à-dire identiques à un argument (par exemple, Proval).

Exemple :

```
procédure    ValidationDuneProvenance (

{arg}        Cval     : NuméroDeClient;
{rés}        var      cond    : boolean
             );
{pré         Client (Numéro de client = Cval) ≠ ∅
             }
{obj         si Client (Numéro de client = Cval) [Catégorie] ≠ 'douteux'
             alors
                      cond := VRAI
             sinon
                      cond := FAUX
             }
```

Cette variante nous oblige à introduire des primitives de consultation du contenu des messages (à raison d'une primitive par attribut du message) afin de fournir à l'interface les éléments dont elle a besoin .

Exemple :

```
procédure    NomDe(

{arg}        numC     : NuméroDeClient;
{rés}        var      nomC    : nomDeClient
             );
{pré         Client (Numéro de client = numC) ≠ ∅
             }
{obj         nomC : = Client (Numéro de client = numC) [Nom de client]
             }
```

V.3.2.3. Deuxième variante

La deuxième variante consiste à introduire une mémoire d'exécution dans la machine

'phase'. Cette mémoire permettra de mémoriser temporairement des données qui, bien qu'elles doivent être mémorisées avec d'autres (par exemple, *provenance* et *lignes* en même temps que *commande*), sont connues avant elles et que l'on est presque sûr de devoir mémoriser à terme (par exemple, *provenance* avant *commande*).

Une fois l'information mémorisée, elle ne devra plus être fournie comme argument d'une procédure ultérieure, puisque les informations nécessaires à cette procédure auront été préalablement et temporairement mémorisées par d'autres procédures de la machine phase.

Exemple :

```
procédure   ValidationDuneProvenance (

{arg}       Cval     : NuméroDeClient;
{rés}       var      cond    : boolean
            );
{pré        Client (Numéro de client = Cval) ≠ ∅
            }
{obj        si Client (Numéro de client = Cval) [Catégorie] ≠ 'douteux'
            alors
                     cond := VRAI
                     &
                     insérer (CommandeTemporaire; ... = ⊥, Client émetteur = Cval)
            sinon
                     cond := FAUX
            }


procédure   MémorisationDuneCommande (

{rés}       var      Cenr :   NuméroDeCommande;
            );
{pré        ...
            }
{obj        ...
            Commande := Commande U CommandeTemporaire
            ...
            }
```

V.3.3. Le dialogue

La conception du dialogue d'une application interactive s'effectue sur base des spécifications conceptuelles de la statique et de la dynamique des traitements, au travers du concept de tâche.

"La tâche est l'expression fonctionnelle des actions que l'utilisateur devra réaliser pour mener à bien l'exécution de l'application interactive. Une tâche est composée de messages interactifs sur lesquels l'utilisateur peut effectuer des opérations sémantiques à propos desquelles on impose une série de contraintes d'enchaînement" [Bod 88b].

Tout message de l'application que l'utilisateur voit apparaître à son poste de travail est un message interactif. Le message interactif est une unité de dialogue qui a un sens pour l'utilisateur. Il peut être de saisie et d'affichage ou d'affichage seulement.
Les messages interactifs peuvent être classés en 4 catégories [Bod 88a] :

- les messages interactifs fonctionnels sont des informations d'entrée ou de sortie des fonctions de l'application. Parmi ces messages, on trouve donc les informations à fournir aux fonctions, les résultats produits par celles-ci, ainsi que les messages d'erreur sémantiques.
- les messages d'erreur du dialogue sont chargés de signaler à l'utilisateur les erreurs syntaxiques qu'il a faites.
- les messages d'aide guident l'utilisateur dans sa manipulation de l'application
- les messages de contrôle du dialogue permettent d'orienter l'exécution de l'application. Ils offrent à l'utilisateur un choix entre différentes options ou opérations et se présentent en général sous la forme de menus.

V.3.3.1. Spécification des messages interactifs fonctionnels

Les informations de saisie et d'affichage présentées par le dialogue d'une application interactive (messages interactifs fonctionnels) sont déterminées par la spécification de la statique des traitements de l'application qui fournit la description des messages d'entrée et de sortie nécessaires à la réalisation des fonctionnalités de l'application (messages fonctionnels). Un message interactif fonctionnel peut correspondre à un regroupement ou un éclatement de messages fonctionnels.

La spécification d'un message interactif fonctionnel comporte les éléments suivants :

- un **nom**,
- une **définition**,
- un **type** (message interactif fonctionnel de saisie/affichage ou d'affichage seulement),

- une **justification** du regroupement de données choisi pour constituer le message, c'est-à-dire un exposé des raisons pour lesquelles on peut considérer ce message comme une unité de dialogue pour l'utilisateur,
- un **contenu**, décrivant les attributs du message,
- les **contraintes syntaxiques** portant sur les attributs du message,
- les **messages d'erreur sémantique** associés au message interactif,
- les **opérations de manipulation** associées au message, c'est-à-dire les opérations que l'utilisateur pourra effectuer sur ce message,
- un **schéma de la conversation** interne au message interactif, décrivant l'enchaînement des actions permises sur les attributs du message interactif (NB : si le type du message est d'affichage seulement, il n'y a pas lieu de définir un schéma de la conversation interne : l'opérateur ne peut effectuer aucune opération sur les attributs du message, la seule opération permise étant la clôture du message).

Un attribut d'un message interactif fonctionnel peut être un autre message interactif fonctionnel ou une structure de données élémentaire. A chaque message interactif fonctionnel est associé un schéma de la conversation. Ceci nous permet de définir plusieurs niveaux de conversation, chacun d'entre eux étant autonome. Par exemple, dans le cas de l'Enregistrement d'un client, on peut définir le message interactif fonctionnel Client, comprenant les messages interactifs Nom et Adresse (Rue, Numéro, Code postal et Localité). La conversation interne du message Client peut être : le Nom suivi de l'Adresse; celle du message Adresse : la Rue, suivie du Numéro, du Code postal et de la Localité. Les deux schémas ainsi définis sont autonomes : une modification de l'enchaînement au sein du message Client ne provoque pas de changement au sein du message Adresse (et vice versa).

Les opérations de manipulation peuvent [Bod 88b]
soit permettre à l'utilisateur de manipuler les éléments constitutifs d'une occurence de message :
- sélectionner un attribut d'un message,
- affecter une valeur à un attribut d'un message,
- supprimer la valeur d'un attribut d'un message,
- corriger la valeur d'un attribut d'un message,
- activer un contrôle portant sur un ou plusieurs attributs;

soit permettre de manipuler l'entièreté d'un message appartenant à une collection de messages :
- créer une occurence d'un message,
- supprimer une occurence d'un message,
- sélectionner une occurence de message,

- ranger une occurence de message (la mettre en attente),
- clôturer une occurence de message (confirmer son contenu).

Les différentes structures d'enchaînement des attributs sont le parallélisme, l'enchaînement séquentiel, la structure convergente, la synchronisation de messages (conjonction ou accumulation), l'itération et la sélection conditionnelle.

V.3.3.2. Le schéma de la conversation global

Ce schéma représente la conversation globale de la tâche. Il définit les règles d'enchaînement entre les messages interactifs fonctionnels du plus haut niveau (c'est-à-dire ceux qui ne sont pas attributs d'autres messages interactifs fonctionnels). Dans le cas où un seul message interactif fonctionnel englobe toutes les informations concernant la tâche, la conversation se réduit à la seule expression de ce message.

V.3.3.3. Spécification des messages purement interactifs

On peut ensuite spécifier les messages purement interactifs. Un message purement interactif est spécifié de la même manière suivante :

- un **nom**,
- une **définition**,
- un **type** (message de contrôle, d'erreur syntaxique ou d'aide),
- l'expression de son **déclenchement** sous forme de précondition,
- une **justification**,
- un **contenu**,
- des **contraintes sur la présentation**, et
- des **opérations de manipulation**.

V.3.3.4. La validation de l'analyse de la tâche par rapport à la dynamique des traitements.

Les messages fonctionnels interactifs proviennent d'un regroupement ou d'un éclatement des messages fonctionnels. Chaque information contenue dans un message externe fonctionnel d'entrée doit se retrouver dans un message interactif fonctionnel de saisie. Quant aux informations contenues dans les messages interactifs fonctionnels d'affichage, elles doivent appartenir à des messages fonctionnels de sortie, excepté si elles

sont calculées par l'interface à partir d'autres informations contenues dans ces messages.

V.3.3.5. La dynamique d'implémentation

A partir des différents schémas de la conversation des messages interactifs fonctionnels et de la dynamique des traitements, le concepteur d'application peut réaliser un schéma intégré, appelé schéma de la dynamique d'implémentation. Ce schéma présente non seulement l'enchaînement des messages interactifs au poste de travail (conversation) mais également les traitements que les informations fournies par ces messages permettent de réaliser.

C'est donc un schéma de la dynamique dans lequel les messages externes fonctionnels sont remplacés par des messages fonctionnels interactifs (ou des attributs de ces messages) et qui respecte les contraintes d'enchaînement décrites par les deux autres types de schémas, en donnant la priorité aux contraintes issues des schémas de la conversation.

"Ce schéma permet au concepteur d'analyser son application en terme d'efficacité. Le déclenchement d'une fonction ne se fera en effet que lorsque toutes les informations dont elle a besoin ont été introduites. La rapidité d'exécution des fonctions dépendra dès lors de l'ordre dans lequel les données dont elle a besoin seront saisies. Si le déclenchement de la première fonction de l'application proprement dite demande une information qui ne sera saisie par l'opérateur qu'en dernier lieu, l'exécution des fonctions ne débutera qu'après son introduction. Cela reste tout à fait admissible si l'utilisateur perçoit cette donnée comme la dernière à communiquer" [Ruc 88].

V.4. LE NIVEAU PHYSIQUE

La conception physique est la traduction de la solution sur une machine réelle caractérisée par ses composants matériels et logiciels (système d'exploitation, langage de programmation, système de gestion des données, gestionnaires d'écran, de communication, de processus, etc.).

V.4.1. Les données

La démarche à appliquer au niveau des données consiste à transcrire le schéma relationnel (cfr V.3.1) dans la syntaxe concrète de l'environnement envisagé.

Nous l'avons signalé en V.3.1 : l'une des propriétés de la plupart des SGBD

relationnels est qu'ils sont à même d'assurer une certaine optimisation d'accès relatifs à une requête relationnelle. En principe, le programmeur ne doit pas se préoccuper de cette optimisation, en quelque sorte implicite des accès. Ce propos est cependant à nuancer puisque tous les SGBD ne possèdent pas les mêmes potentialités à ce niveau. Une optimisation explicite des accès aux données est parfois à conseiller (voir à ce sujet [Hai 86]). Nous nous en tiendrons pour notre part à la conception basée sur une optimisation implicite, pour la raison déjà évoquée que la performance ne constitue pas l'un des objectifs de cette expérimentation.

La conception physique de la base de données consiste donc en la production du schéma interne des données. Il est dérivé du schéma relationnel : les relations restent des relations, les attributs des relations deviennent les champs des relations et les domaines de attributs les types des champs. Le schéma est déterminé principalement par le choix des clés d'accès à attribuer à chaque table ou relation. Ces clés peuvent être déduites des identifiants des relations mais l'on choisit également les clés qui paraissent utiles (fréquence d'utilisation, taille des tables) au regard des accès à effectuer sur les données.

"Outre la présence et les composants des clés d'accès, on précisera également, selon les SGBD, les techniques de réalisation de ces clés, (...), le mode de rangement des lignes d'une table (en vrac, par agrégats selon un index, par agrégats selon une expression de jointure entre deux tables, ...)" [Hai 86].

V.4.2. Les traitements

La démarche de conception physique des traitements se résume à l'implémentation (c'est-à-dire le codage) des procédures logiques dans la syntaxe du langage de programmation utilisé.
Un module physique doit représenter une unité d'exécution dans ce langage. Cela peut nécessiter l'éclatement ou la réunion de plusieurs modules logiques (procédures), ces manipulations demeurant invisibles au niveau d'abstraction supérieur.

V.4.3. Le dialogue

Nous avons vu (cfr V.3.3) que la tâche est l'expression fonctionnelle des actions que l'opérateur doit réaliser pour mener à bien l'exécution de l'application interactive.

"L'interface est la concrétisation visuelle de la tâche. Elle est composée d'une série d'objets interactifs (fenêtres, icônes, menus,...) présentant les messages interactifs, sur lesquels l'utilisateur effectuera des actions physiques (déplacement d'une fenêtre au moyen

de la souris,...). Des contraintes d'enchaînement de ces actions doivent également être émises afin de traduire, au niveau des manipulations physiques de l'utilisateur, les contraintes exprimées dans la tâche" [Bod 88b].

Chapitre VI
Expérimentation

VI.1. CHOIX D'UNE DÉMARCHE EXPÉRIMENTALE

VI.1.1. Objectif de l'expérience

L'expérimentation que nous présentons ici porte sur le développement et la réalisation d'une application dans un E4G.

Nous avions des a priori, des idées pré-conçues, à la fois sur les démarches de conception, notamment celle que nous venons de décrire au chapitre V, et sur les E4Gs, que nous avons décrits au chapitre IV. L'objectif que nous poursuivions durant cette expérimentation était de déceler les inadéquations éventuelles des méthodes de conception classiques que nous utilisions, par rapport à cette nouvelle génération.

Cet objectif associé à l'expérimentation rejoint en fait l'objectif final de ce mémoire qui est de dégager les mécanismes, les concepts méthodologiques les mieux adaptés à une implémentation dans un E4G.

Nous avons choisi une application interactive simple : l'application de Gestion du Circuit de la Commande. Nous n'en avons retenu que la phase initiale d'Enregistrement d'une Commande-Client, déjà rodée aux méthodes de conception classiques. C'est une phase conversationnelle par excellence. Elle semble donc convenir idéalement aux E4Gs. Nous aurions pu présenter ici une autre application, la Gestion de Réservation dans un Hôtel. Nous l'avons également développée et implémentée dans l'un des E4Gs étudiés (Magic). Cependant, elle comporte davantage de traitements 'batch' et apparaît plus complexe. Or, il ne faut pas perdre de vue notre préoccupation première, l'analyse de l'adéquation des méthodologies de conception aux E4Gs. La conception de la phase d'Enregistrement d'une Commande-Client comporte suffisamment d'éléments à prendre en compte dans cette analyse pour que nous ne nous dispersions pas. L'implémentation des traitements 'batch' dans un E4G pourrait faire l'objet d'une étude plus approfondie, mais sort en tout cas du cadre de notre propos.

VI.1.2. Démarche expérimentale choisie

Notre démarche s'est déroulée en deux temps : d'abord le développement de la phase suivant les étapes de la méthode de conception jusqu'au niveau logique, pour les données et

les traitements, indépendamment de tout E4G; ensuite, l'implémentation de la phase dans les E4Gs étudiés. Cette seconde étape fut plus longue que la première puisqu'il s'agissait avant tout de s'initier aux E4Gs mis à notre disposition (seuls trois des quatre E4Gs décrits précédemment nous furent accessibles en pratique : Magic, Quatrième Dimension et Rally). Durant cette étape, l'implémentation fut en grande partie intuitive. Cependant, nous nous sommes efforcés de respecter les résultats des étapes précédentes de la conception; nous nous basions uniquement sur nos antécédents de programmation (orientés E3Gs uniquement) et sur la connaissance que nous avions des E4Gs, c'est-à-dire sur les concepts du modèle descriptif spécifié aux chapitres III et IV. Il est à remarquer que nous ne présentons pas, dans ce chapitre, l'implémentation de la phase dans chaque E4G particulier. Cela n'a pas d'objet dans la mesure où chacun d'eux se rapproche du modèle général, et cela vous épargne à vous, lecteur, un apprentissage préalable de ces environnements.

Nous ferons part, dans le chapitre VII, de nos réflexions concernant l'adéquation ou la non-adéquation de la démarche de conception suivie.

VI.2. SPÉCIFICATIONS INFORMELLES DE L'EXEMPLE

L'enregistrement d'une commande client se décompose en plusieurs étapes :

- l'utilisateur peut fournir soit le nom, soit le numéro du client passant la commande. L'introduction du nom du client signifie qu'il ne possède pas encore de numéro. Il s'agit dès lors d'un nouveau client. Le système devra lui attribuer un numéro et mémoriser ses coordonnées. Le numéro d'un client permet de l'identifier parmi l'ensemble des personnes achetant des produits à l'entreprise; il constitue l'information qui devra être saisie lorsqu'un client passe une commande. Il y aura une vérification de ce numéro pour qu'il corresponde effectivement à une personne enregistrée dans la base de données de la firme et on vérifiera également qu'il n'est pas sur la liste noire (les mauvais payeurs).

- l'utilisateur devra spécifier l'adresse d'un nouveau client pour que l'entreprise sache où acheminer les marchandises et les prospectus qu'elle pourrait envoyer. Il pourra également modifier l'adresse d'un ancien client. Une procédure d'enregistrement de l'adresse permettra d'y parvenir

- les lignes de la commande seront constituées d'un numéro de produit et de la quantité désirée. Chaque numéro de produit devra correspondre à un produit vendu par la firme et disponible en stock (à moins qu'il n'existe un produit de substitution disponible).

- si les références du client ont été introduites avec succès et qu'au moins une ligne de

commande a pu être validée, la commande sera alors enregistrée, à moins que le montant de la commande ne dépasse la limite d'achat du client.

VI.3. ANALYSE FONCTIONNELLE

Nous reprenons de [Pig 89] la spécification conceptuelle de la phase.
Une première description de la phase ***Enregistrement d'une commande client*** a été établie lors de l'étude d'opportunité de l'application ***Gestion du circuit de la commande***. Celle-ci signale notamment que :

La phase ***Enregistrement d'une commande-client***

a pour objectif de vérifier une commande-client à enregister et, lorsqu'elle est valide, de la mémoriser. De plus, elle doit permettre de mémoriser un client qui commande pour la première fois et d'enregistrer un changement d'adresse communiqué par un ancien client;

est effectuée dans le ***service de réception des commandes***

génère une ***commande enregistrée***, si la mémorisation a été possible

utilise et/ou modifie une mémoire qui comprend :

- des ***commandes***, leur ***provenance*** et leurs ***lignes***;
- des ***clients*** et
- des ***produits***

reçoit une ***commande à enregistrer*** qui peut comprendre :

- un ***numéro de client***,
- un ***nom de client (nom, prénom et titre)***,
- une ***adresse de domicile (rue, Numéro, code postal et localité)***,
- des ***numéros de produit et des quantités commandées***.

VI.3.1. Les Données : Structuration des informations mémorisées

VI.3.1.1. Description de la structure des données et des contraintes d'intégrité

Dans un premier temps, il s'agit de décrire (à l'aide du modèle *Entité-Association*) la structure des données de la phase et ses contraintes d'intégrité. Ainsi a-t-il été décidé de représenter la mémoire de la phase ***Enregistrement d'une commande-client*** par les ensembles suivants :

- l'ensemble des entités ***CLIENT(s)***,
  qui correspondent aux acheteurs, susceptibles de passer des commandes;

  Normalement, une exécution de la phase consulte cet ensemble pour reconnaître un client qui passe une commande, mais elle peut aussi être amenée à y ajouter un nouveau client ou y modifier l'adresse d'un client existant.

- l'ensemble des entités ***PRODUIT(s)***,
  qui correspondent aux articles repris dans le catalogue et que les clients peuvent commander;

  l'ensemble des associations ***SUBSTITUTION(s)***,
  qui relient certains produits épuisés à leur produit analogue;

  Une exécution de la phase suppose que les deux ensembles précédents existent et ne fait que consulter les informations qui y sont reprises.

- l'ensemble des entités ***COMMANDE(s)***,
  qui correspondent aux ordres de commande passés par les clients pour des articles présentés dans le catalogue;

  l'ensemble des associations ***PROVENANCE(s)***,
  qui relient les commandes à leur client;

  l'ensemble des associations ***LIGNE(s)***,
  qui relient les commandes à leurs produits;

  Une exécution de la phase se traduit généralement par l'insertion dans les trois ensembles précédents, respectivement d'une nouvelle commande, de sa provenance et de ses lignes.

Une entité ***CLIENT***

représente une personne, physique ou morale, appartenant au marché des acheteurs potentiels ou réels de la société;

est caractérisée par :

un ***Numéro de client***
numéro attribué par compostage à l'enregistrement d'un nouveau client,
un ***Nom de client (Nom, Prénom, Titre)***
renseignements fournis par le client lors de son premier contact,
une ***Adresse de domicile (Rue, Numéro, Code postal, Localité)***
dernier domicile connu, donné par le client,
une ***Limite d'achat***
montant du crédit maximal attribué au client
[est initialement de 2500 FB à l'enregistrement d'un 'nouveau' client] et
une ***Catégorie***
indication sur l'état des relations entre le client et la société :
['régulier' s'il a acheté dans les 12 derniers mois,
'nouveau' s'il s'agit de son premier contact avec la société,
'ancien' s'il n'a plus acheté depuis 12 mois,
'douteux' s'il est repris sur la "liste noire" de la société, et
'collaborateur' s'il est un employé de la société];

joue le rôle de ***Client (Emetteur)*** dans *aucune ou plusieurs* ***Provenance(s)***;

est identifié par son *Numéro de client.*

Une entité ***PRODUIT***

représente un article figurant au dernier catalogue diffusé par la firme;

est caractérisée par :

un ***Numéro de produit***
référence de l'article telle qu'elle apparaît dans le catalogue
[les 2 premiers chiffres représentent le "rayon d'achat",

les 5 chiffres suivants représentent l' "article taille/couleur" et
le dernier chiffre représente le "check digit"
(le reste de la division entière des 7 premiers chiffres par 7)],

un ***Libellé de produit***
descriptif de l'article, repris dans le catalogue,

une ***Unité d'achat***
nombre d'articles faisant l'objet d'un lot non partionnable,

un ***Prix unitaire d'achat***
prix courant pour une Unité d'achat,

un ***Etat d'approvisionnement***
indicateur signalant la situation du produit en stock
[peut être 'en stock', 'en avis d'attente' ou 'épuisé'] et

une ***Date de réapprovisionnement*** *(facultative)*
indication sur la date probable de réassortiment
[pour un Produit dont l'Etat d'approvisionnement est 'en avis d'attente'];

joue le rôle de ***Produit (Commandé)*** dans *aucune ou plusieurs* ***Ligne(s)***,
de ***Produit (Analogue)*** dans *aucune ou plusieurs* ***Substitution(s)*** et
de ***Produit (Remplacé)*** dans *aucune ou une seule* ***Substitution;***

est identifiée par son *Numéro de produit.*

Une association ***SUBSTITUTION***

représente l'existence, pour un produit épuisé, d'une autre produit analogue qui peut lui être substitué dans toute commande;

relie un ***Produit (Remplacé)***
[dont l'Etat d'approvisionnement est 'épuisé'] et
son ***Produit (Analogue)***
[dont l'Etat d'approvisionnement n'est pas 'épuisé'];

est identifié par le *Produit (Remplacé).*

Une entité ***COMMANDE***

représente un ordre de commande passé par un client auprès de la société pour un ou plusieurs produits;

est caractérisée par :

un ***Numéro de commande***
numéro attribué par compostage lors de l'enregistrement de la Commande,
une ***Date de commande***
date du jour où la Commande a été enregistrée,
un ***Montant total de commande***
Somme de Montant(s) de ligne pour toutes les Ligne(s) de la Commande (Concernée) et
un ***Rabais de commande*** *(facultatif)*
réduction de 20% du Montant total de commande
[accordé à un Client (Emetteur) dont la Catégorie est 'collaborateur'];

joue le rôle de ***Commande (Emise)*** dans *une et une seule* ***Provenance*** et
de ***Commande (Concernée)*** dans *une ou plusieurs* ***Ligne(s)***;

est identifiée par son *Numéro de commande,*

[ne peut être ajoutée que si le Montant total de la commande, réduit du Rabais de commande éventuel, est inférieur à la Limite d'achat du Client (Emetteur)].

Une association ***PROVENANCE***

représente l'engagement contractuel entre un client et la société pour une commande donnée;

relie une ***Commande (Emise)*** et
son ***Client (Emetteur)***
[dont la catégorie n'est pas 'douteux'];

est identifiée par la *Commande (Emise).*

Une association ***LIGNE***

représente la promesse de livraison d'un article commandé;

relie une ***Commande (Concernée)*** et
un ***Produit (Commandé)***
[dont l'Etat d'approvisionnement n'est pas 'épuisé'];

est caractérisée par :

une ***Quantité commandée***
nombre d'Unités d'achat commandées du Produit (Commandé) et
un ***Montant de ligne***
produit de la Quantité commandée par le Prix unitaire d'achat du Produit (Commandé);

est identifiée par la *Commande (Concernée)* et le *Produit (Commandé).*

VI.3.1.2. Schéma Entité/Association

Le Schéma correspondant, dans le formalisme graphique du modèle Entité/Association, à cette structuration conceptuelle des données de la phase Enregistrement d'une Commande-Client est représenté à la figure VI.1 :

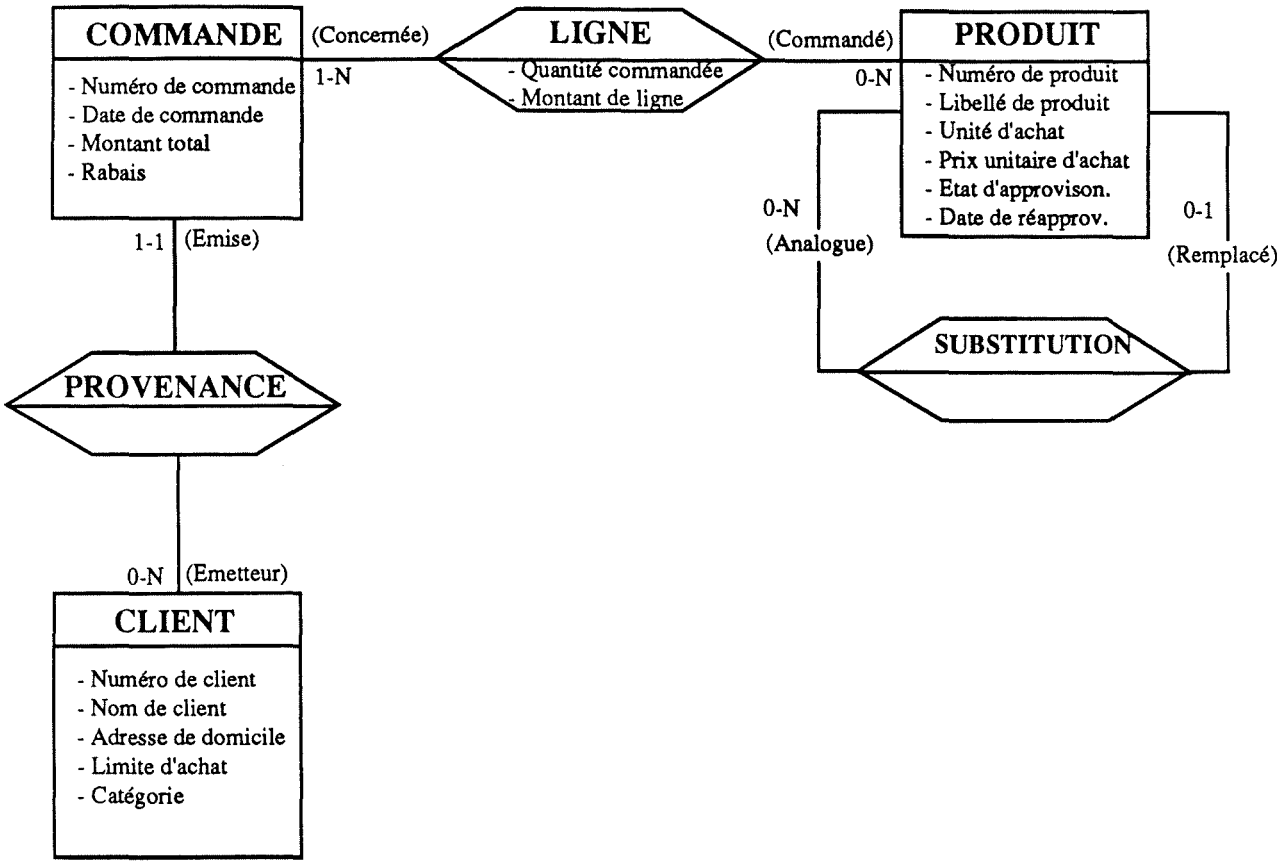


Figure VI.1. : Schéma Entité/Association

VI.3.2. La statique des traitements : Spécification des fonctions et des messages fonctionnels

Dans un second temps, il s'agit de décrire les traitements nécessaires pour accomplir les objectifs assignés à la phase. Dans le cas de l'***Enregistrement d'une commande-client***, on distingue les 3 objectifs suivants :

- ***mémorisation de la commande***,
- ***mémorisation du client***, s'il commande pour la première fois, et
- ***modification de l'adresse du client***, s'il signale un changement d'adresse.

En fonction de la structure des données adoptée, il a été décidé de décrire les traitements associés au premier objectif à l'aide des fonctions suivantes :

- la fonction *MEMORISATION D'UNE COMMANDE*
  a pour objectif de créer et de mémoriser une nouvelle commande-client valide, avec sa provenance et ses lignes de commande

Ces différentes opérations ne peuvent être dissociées, étant données les contraintes d'intégrité qui signalent qu'une commande ne peut pas être mémorisée sans sa provenance et au moins une ligne de commande.

Cette mémorisation **suppose** que les informations nécessaires à la création de la commande sont correctes; c'est-à-dire dans un état tel qu'elles permettront une mémorisation respectant toutes les contraintes d'intégrité spécifiées. Cela signifie donc qu'il existe d'autres fonctions qui vérifient la correction des informations fournies :

- la fonction ***VALIDATION D'UNE COMMANDE***
  a pour objectif de garantir, à partir des lignes de commande et du client, que les lignes portent sur des produits différents, que le montant de la commande est inférieur à la limite d'achat du client et enfin que le rabais de commande est établi s'il s'agit d'un employé de la société

- la fonction ***VALIDATION D'UNE PROVENANCE***
  a pour objectif de garantir que le client n'est pas sur la 'liste noire'

- la fonction ***VALIDATION D'UNE LIGNE***
  a pour objectif de garantir que la ligne porte sur un produit non épuisé et que, pour un tel produit, la quantité commandée est fixée

- la fonction ***VALIDATION D'UN CLIENT***
  a pour objectif d'identifier un client, existant dans la mémoire, à partir de son numéro de client

- la fonction ***VALIDATION D'UN PRODUIT***
  a pour objectif d'identifier un produit, existant dans la mémoire, à partir de son numéro de produit.

La fonction ***Mémorisation d'une commande-client*** doit ajouter une nouvelle ***Commande*** dans la mémoire, mais également sa ***Provenance*** et ses ***Lignes***, qui lui sont obligatoiremenent rattachées étant données les contraintes de connectivité spécifiées.

Le point de départ de la description de la fonction est donc la partie de la structure de données concernée par ces ajouts dans la mémoire et qui est rappelée ci-dessous. Les chiffres entre [

et] constituent les contraintes qui, supposées vérifiées avant la mémorisation, devront être préservées après l'exécution de la fonction ***Mémorisation d'une commande***.

Une entité ***Commande***
est caractérisée par :
un ***Numéro de commande***
une ***Date de commande***
un ***Montant total de commande***
[1] *[Somme des Montant(s) de ligne pour toutes les Ligne(s) de la Commande]*
et
un ***Rabais de commande*** *(facultatif)*
réduction de 20% du Montant total de commande
[2] *[accordée à un Client (Emetteur) dont la catégorie est 'collaborateur'];*
[3] joue le rôle de ***Commande (Emise)*** dans *UNE et une seule* ***Provenance*** et
[4] de ***Commande (Concernée)*** dans *UNE ou plusieurs* ***Ligne(s)***;
[5] est identifiée par son *Numéro de commande;*

[6] *[ne peut être ajoutée que si le Montant total de commande est inférieur à la Limite d'achat du Client (Emetteur)].*

où une association ***Provenance***
relie une ***Commande (Emise)*** et
[7] son ***Client (Emetteur)***
[8] *[dont la Catégorie n'est pas 'douteux'];*

où une association ***Ligne***
relie une ***Commande (Concernée)*** et
[9] un ***Produit (Commandé)***
[10] *[dont l'Etat d'approvisionnement n'est pas 'épuisé'];*
est caractérisée par :
une ***Quantité commandée***
un ***Montant de ligne***
[11] *[produit de la Quantité commandée par le Prix unitaire d'achat du Produit (Commandé)];*
[12] est identifiée par la *Commande (Concernée)* et le *Produit (Commandé).*

A partir de cette structure et de ses contraintes, il s'agit de décrire la fonction ***Mémorisation d'une commande***. Les chiffres entre [et] correspondent aux contraintes qui ont été

intégrées dans le traitement.

Les contraintes non intégrées dans la fonction sont donc **supposées** avoir été vérifiées avant l'exécution de celle-ci. A cet effet, le message ***Commande valide*** reprend la structure des *données véhiculées* sur la commande à enregistrer. Cette structure est accompagnée des contraintes supposées avoir été vérifiées.

On constatera que les contraintes non intégrées dans la fonction et celles associées au message reçu correspondent en genre et en nombre aux contraintes de la structure des données rappelées ci-dessus.

La fonction ***MEMORISATION D'UNE COMMANDE***

a pour objectif de créer et de mémoriser une nouvelle commande-client valide, avec sa provenance et ses lignes de commande;

génère une ***Commande enregistrée***

ajoute une ***COMMANDE***

[5] *[dont le Numéro de commande est attribué par compostage*
la Date de commande est celle du jour
le Montant total de commande est celui de la Commande valide
le Rabais de commande éventuel est celui de la Commande valide],

[3] une ***PROVENANCE***
[entre cette Commande et le Client de la Provenance valide],

[4] des ***LIGNE(s)***, autant que de ***Ligne(s) valide(s)***
[entre cette Commande et le Produit de la Ligne valide
dont la Quantité commandée est celle de la Ligne valide
le Montant de ligne est celui de la Ligne valide];

modifie la ***Limite d'achat*** du ***Client (Emetteur)***
[en lui soustrayant le Montant total de commande,
réduit du Rabais de commande éventuel de la Commande valide];

reçoit une ***Commande valide.***

Un message ***COMMANDE VALIDE***
comprend une ***Provenance valide,***

des ***Ligne(s) valide(s)***, au moins une,
[12] *[qui portent sur des produits différents],*
un ***Montant total de commande***
[1] *[somme des Montant(s) de ligne des Ligne(s) valide(s)]* et
[6] *[qui, réduit du Rabais de commande éventuel, est inférieur à la Limite d'achat du Client de la Provenance valide]* et
un ***Rabais de commande*** *(facultatif)*
[2] *[si la Catégorie du Client valide est 'collaborateur'];*

où une ***PROVENANCE VALIDE***
comprend un ***Client valide***
[8] *[dont la Catégorie n'est pas 'douteux'];*

où un ***CLIENT VALIDE***
comprend un ***Client***
[7] *[qui existe dans la mémoire];*

où une ***LIGNE VALIDE***
comprend un ***Produit valide***
[10] *[dont l'Etat d'approvisionnement n'est pas 'épuisé'];*
une ***Quantité commandée*** et
un ***Montant de ligne***
[11] *[produit de la Quantité commandée par le Prix unitaire d'achat du Produit valide];*

où un ***PRODUIT VALIDE***
comprend un ***Produit***
[9] *[qui existe dans la mémoire];*

La fonction ***VALIDATION D'UNE COMMANDE***

a pour objectif de garantir, à partir des lignes de commande et du client, que :
- *les lignes portent sur des produits différents,*
- *le montant de la commande est inférieur à la limite d'achat du client et*
- *le rabais de commande est établi s'il s'agit d'un employé de la société;*

génère une ***Commande valide***

[6] si *[le Montant total de commande, réduit du Rabais de commande éventuel, est inférieur à la Limite d'achat du Client valide];*

applique les règles :

[12] 1. *si plusieurs ligne(s) valide(s) portent sur un même produit, alors il faut les remplacer par une seule Ligne valide dont :*
- *la Quantité commandée est la somme des Quantité(s) commandée(s) initiales*
- *le Montant de ligne est la somme des Montant(s) de ligne initiaux*

[11] 2. *le Montant total de commande est la somme des Montant(s) de ligne pour toutes les Ligne(s) valide(s), après application de la règle 1,*

[2] 3. *si la Catégorie du Client valide est 'collaborateur', alors le Rabais de commande est établi (20% du montant total de commande).*

reçoit une ***Provenance valide*** et
des ***Ligne(s) valide(s)***, au moins une;

Un message ***PROVENANCE VALIDE***
comprend un ***Client valide***,
[8] *[dont la Catégorie n'est pas 'douteux'];*

où un ***CLIENT VALIDE***
comprend un ***Client***
[7] *[qui existe dans la mémoire];*

Un message ***LIGNE VALIDE***
comprend un ***Produit valide***,
[10] *[dont l'Etat d'approvisionnement n'est pas 'épuisé'],*
une ***Quantité commandée*** et
un ***Montant de ligne***
[11] *[produit de la Quantité commandée par le Prix unitaire d'achat du Produit valide];*

où une ***PRODUIT VALIDE***

comprend un ***Produit***
[9] *[qui existe dans la mémoire];*

La fonction ***VALIDATION D'UNE PROVENANCE***

a pour objectif de garantir que le client n'est pas sur la 'liste noire';

génère une ***Provenance valide***
[8] si *[la Catégorie du Client n'est pas 'douteux'];*

reçoit un ***Client valide***.

Un message ***CLIENT VALIDE***
comprend un ***Client***
[7] *[qui existe dans la mémoire];*

La fonction ***VALIDATION D'UNE LIGNE***

a pour objectif de garantir que la ligne porte sur un produit non épuisé et que, pour un tel produit, la quantité commandée est fixée;

génère une ***Ligne valide***
[10] si *[l'Etat d'approvisionnement du Produit valide n'est pas 'épuisé'];*

applique les règles :

[10] *1. si l'Etat d'approvisionnement du Produit est 'épuisé' et si celui-ci peut être substitué par un Produit (Analogue), alors il faut remplacer le Produit valide par son Produit (Analogue)*

[11] *2. si la Quantité commandée n'est pas explicitement fournie, alors elle est d'UNE Unité d'achat*

[11] *3. le Montant de ligne est le produit de la Quantité commandée par le Prix unitaire d'achat du Produit, après application des règles 1 et 2;*

reçoit un ***Produit valide*** et
une ***Quantité commandée*** *(facultative).*

Un message ***PRODUIT VALIDE***
comprend un ***Produit***
[9] *[qui existe dans la mémoire];*

La fonction ***VALIDATION D'UN CLIENT***

a pour objectif d'identifier un client, existant dans la mémoire, à partir de son seul numéro de client;

génère un ***Client valide***
[7] si *[il existe dans la mémoire un client avec ce Numéro de client];*

reçoit un ***Numéro de client.***

La fonction ***VALIDATION D'UN PRODUIT***

a pour objectif d'identifier un produit, existant dans la mémoire, à partir de son seul numéro de produit;

génère un ***Produit valide***
[9] si *[il existe dans la mémoire un produit avec ce Numéro de produit];*

reçoit un ***Numéro de produit.***

La fonction ***MODIFICATION DE L'ADRESSE D'UN CLIENT***

a pour objectif d'enregistrer un changement d'adresse communiqué par un client, qui existe déjà dans la mémoire;

modifie l' ***Adresse de domicile*** du ***CLIENT***
[en lui substituant l'Adresse de (nouveau) domicile fournie];

reçoit un ***Client valide*** et
une ***Adresse de (nouveau) domicile (Rue, Numéro, Code postal, Localité).***

La fonction ***MEMORISATION D'UN CLIENT***

a pour objectif de créer et mémoriser un nouveau client;

génère une ***Provenance valide;***

ajoute un ***CLIENT***
[dont le Numéro de client est attribué par compostage
le Nom de client est celui donné en entrée
l'Adresse de domicile est celle donnée en entrée
la Limite d'achat est de '2500 FB'
la Catégorie est 'nouveau']

reçoit un ***Nom de client (Nom, Prénom, Titre)*** et
une ***Adresse de domicile (Rue, Numéro, Code postal, Localité)***

VI.3.3. La dynamique des traitements : Spécification de l'enchaînement des fonctions

Le dialogue de la phase d'Enregistrement d'une Commmande-client est spécifié, au niveau conceptuel, par l'enchaînement des fonctions de la phase. Le Schéma de la dynamique de ces fonctions est représenté par la figure VI.2.

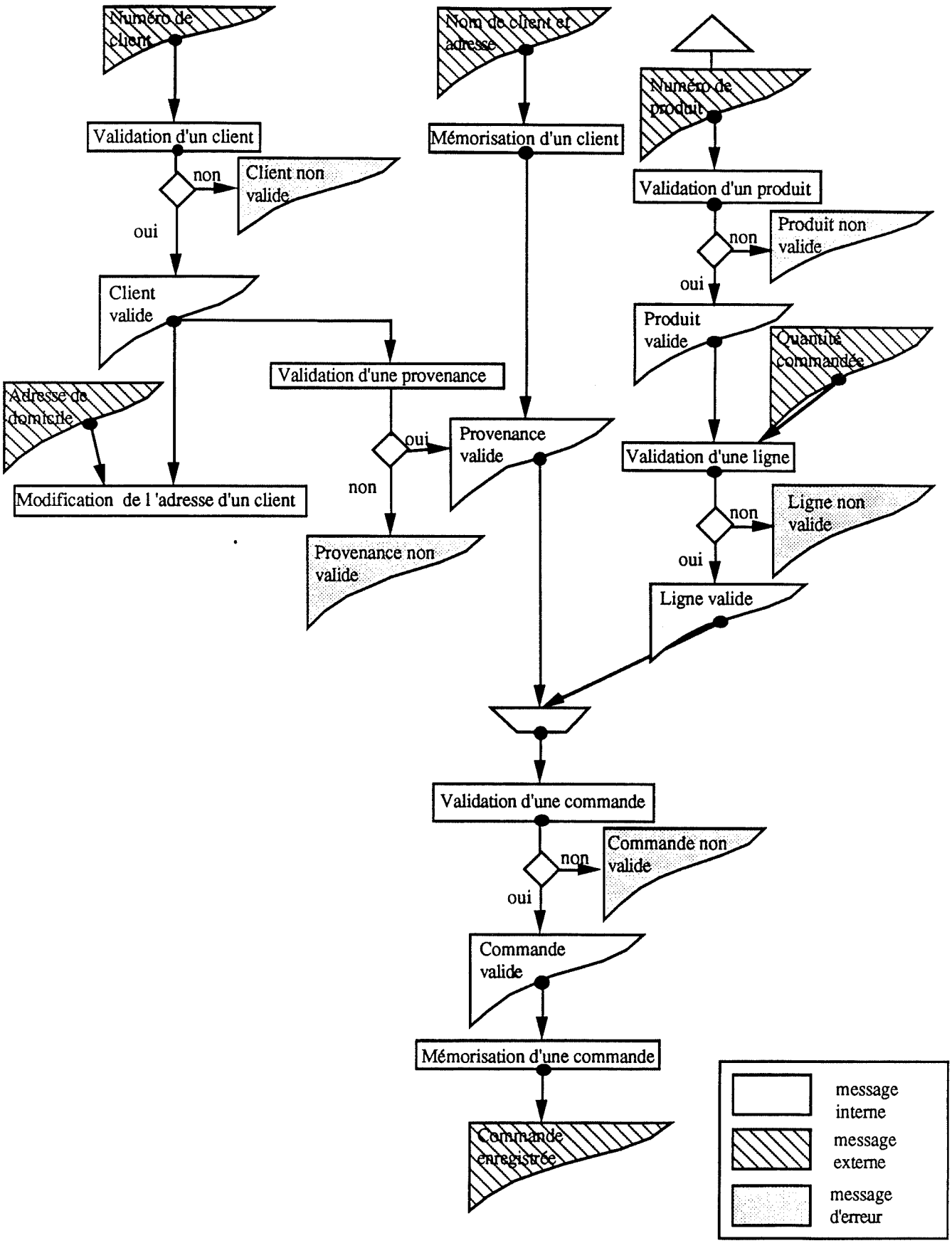


Figure VI.2. : Schéma de la dynamique des fonctions de la phase

VI.4. LA CONCEPTION LOGIQUE

VI.4.1. Les Données : Le schéma relationnel

La modélisation relationnelle des données de la phase d'Enregistrement d'une Commande-client est déduite de leur modélisation conceptuelle (cfr VI.3.1). Les règles de transformation appliquées sont celles qui ont été décrites en V.3.1 (passage du schéma E/A au schéma MAG des accès possibles, ensuite au schéma conforme au relationnel et enfin, production du schéma relationnel). La seule restriction que nous y avons apportée concerne le maintien, à ce niveau, des attributs décomposables. Elle est justifiée par un souci de clarté et de concision dans la description du schéma relationnel.

La figure VI.3 représente le schéma MAG conforme au relationnel.

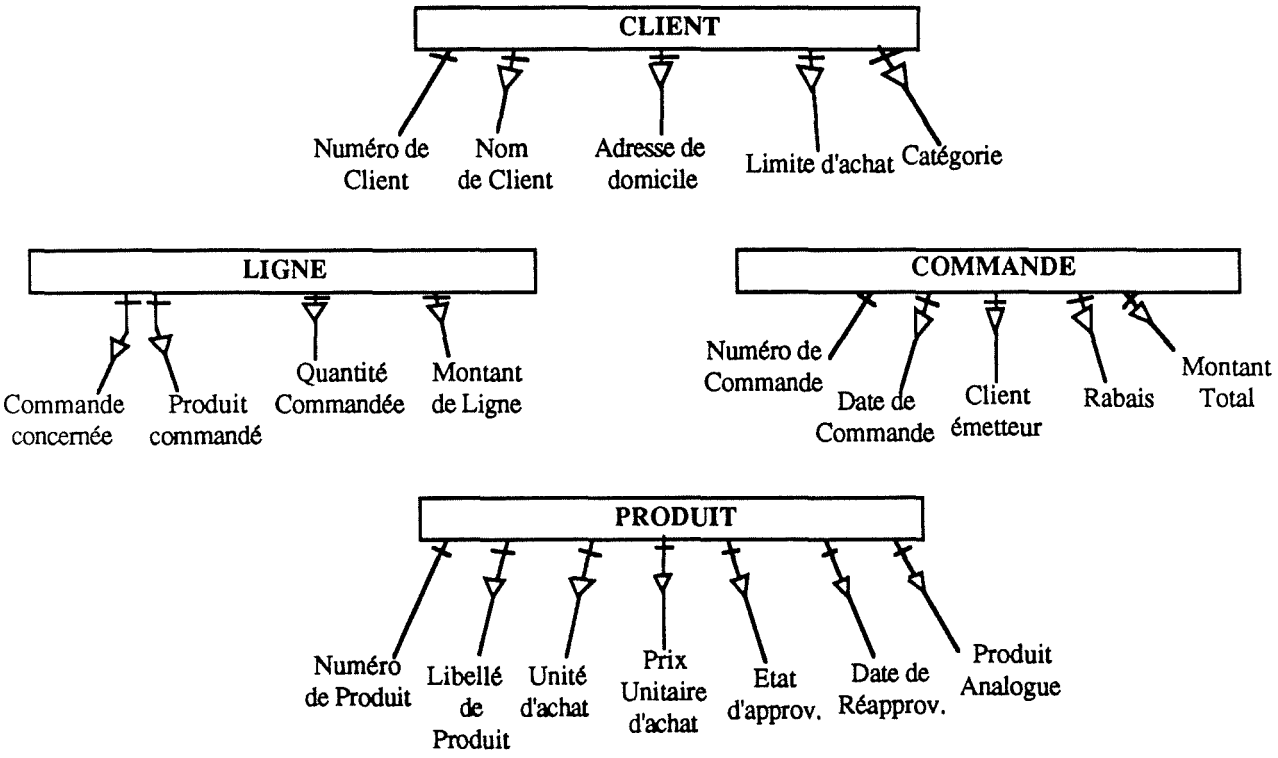


Figure VI.3. : Schéma MAG conforme au relationnel

Les tables ou relations appartenant au schéma relationnel déduit de ce schéma peuvent être décrites de la manière suivante :

Client (Numéro de client, Nom de client, Adresse de domicile, Limite d'achat, Catégorie) avec :

- **Numéro de client** : integer;
- **Nom de client** : record
  Nom : array [1..40] of char;
  Prénom : array [1..40] of char;
  Titre : array [1..40] of char
  end;
- **Limite d'achat** : real;
- **Adresse de client** : record
  Rue : array [1..40] of char;
  Numéro : array [1..6] of char;
  Code postal : array [1..4] of char;
  Localité : array [1..40] of char;
  end;

et **Catégorie** = ('REGULIER', 'NOUVEAU', 'ANCIEN', 'DOUTEUX', 'COLLABORATEUR')

Produit (Numéro de produit, Libellé de produit, Unité d'achat, Prix unitaire d'achat, Etat d'approvisionnement, Produit analogue)

avec :

- **Numéro de produit** : integer;
- **Libellé de produit** : array [1..50] of char;
- **Unité d'achat** : integer;
- **Prix unitaire d'achat** : real;
- **Etat d'approvisionnement** = ('EN STOCK', 'EN AVIS D'ATTENTE', 'EPUISE')

et **Produit analogue** : integer;

Commande (Numéro de commande, Date de commande, Montant total de commande, Rabais de Commande, Client émetteur)

avec :

- **Numéro de commande** : integer;
- **Date de commande** : date;
- **Client émetteur** : integer;
- **Montant total de commande** : real;

et **Rabais de commande** : real;

Ligne (Commande concernée, Produit commandé, Quantité commandée, Montant de ligne) avec :

- **Commande concernée** : integer;
- **Produit commandé** : integer;
- **Quantité commandée** : integer;

et **Montant de ligne** : real;

Les contraintes d'intégrité associées à ce schéma sont la traduction des contraintes liées au modèle E/A, complétées des contraintes référentielles induites par la transformation du schéma MAG en un schéma conforme au relationnel. Elles sont exprimées dans le formalisme de l'algèbre relationnelle :

- Produit [Produit analogue] in Produit [Numéro de produit]

- $\forall$ n $\in$ à Commande [Numéro de commande], $\exists$ Ligne (Commande concernée = n)
- $\forall$ n $\in$ à Commande [Numéro de commande],
  Commande (Numéro de commande = n) [Montant total de commande] := $\sum$ Ligne (Commande concernée = n) [Montant de ligne]
- $\forall$ n $\in$ à Commande [Numéro de commande],
  Commande (Numéro de commande = n) [Rabais de commande]
  = Commande (Numéro de commande = n) [Montant total de commande] *
  20% si Client (Numéro de client = Commande (Numéro de commande = n) [Client émetteur]) [Catégorie] = 'COLLABORATEUR'
  = 0 sinon
- Commande [Client émetteur] in Client [Numéro de client]

- Ligne [Commande concernée] in Commande [Numéro de commande]
- Ligne [Produit commandé] in Produit [Numéro de produit]
- $\forall$ {n,m} $\in$ à Ligne [Commande concernée, Produit commandé],
  Ligne (Commande concernée =n , Produit commandé = m) [Montant de ligne]
  = Ligne (Commande concernée =n , Produit commandé = m) [Quantité commandée] *
  Produit (Numéro de produit = m) [Prix unitaire d'achat]

VI.4.2. Les Traitements : La Machine 'Phase'

Nous présentons ici les services offerts par la machine 'phase'. Ce sont les procédures déduites des fonctions de l'analyse fonctionnelle selon les règles de transformation décrites en V.3.2 :

La phase décrite en V.3 a été transformée en machine 'phase' suivant les règles énoncées en V.3.2.1. Le résultat obtenu a subi ensuite un certain nombre de transformations (cfr les deux variantes décrites aux sections sections V.3.2.2 et V.3.2.3). La seconde variante a nécessité l'adjonction de deux relations temporaires :

CommandeTemporaire (Numéro de commande, Date de commande, Montant total de commande, Rabais de commande, Client émetteur)
LigneTemporaire (Numéro de ligne[1], Produit commandé, Quantité commandée, Montant de ligne)

Voici les opérations permises sur la machine 'phase' :

```
procédure   MémorisationDunClient (

{arg}       nomC   : nomDeClient;
            adrC   : adresseDeClient;

{rés}       var    numC  :  NuméroDeClient
            );

{obj        pour n un Numéro ∉ Client [Numéro de client]
                   ajouter (client; n, nomC, adrC, 2500, 'NOUVEAU')
            &
            numC := n
            }


procédure   ModificationDeLadresseDunClient (

{arg}       numC   : numéroDeClient;
            adrC   : adresseDeClient;

{pré}       Client (Numéro de client = numC) ≠ ∅
            );
```

[1] Nous avons du introduire un identifiant artificiel Numéro de ligne afin de pouvoir enregistrer temporairement des lignes avec un même Numéro de produit (Produit Commandé).

{obj modifier (client; Numéro de client = numC; Adresse de domicile = adrC)
}

procédure ***ValidationDunProduit*** (

{arg} numP : numéroDeProduit;

{rés} var cond : boolean
);

{obj cond := (Produit (Numéro de produit = numP) ≠ ∅)
}

procédure ***ValidationDunClient*** (

{arg} numC : numéroDeClient;

{rés} var cond : boolean
);

{obj cond := (Client (Numéro de client = numC) ≠ ∅)
}

procédure ***ValidationDuneLigne*** (

{arg} Pval : numéroDeProduit;
quC : entier;

{rés} var Pval : numéroDeProduit;
var quC : entier;
var cond : booléen
);

{pré Produit (Numéro de produit = Pval) ≠ ∅
}

```
{obj    si Produit (Numéro de produit = Pval) [Etat d'approvisionnement] ≠ 'épuisé'
        alors
                si quC = 0
                alors quC:= 1
                sinon ---
                &
                Montant := quC * Produit (Numéro de produit = Pval) [Prix Unitaire
                        d'achat]
                &
                pour n un Numéro ∉ LigneTemporaire [Numéro de ligne],
                ajouter (LigneTemporaire; n, Pval, quC, Montant)
                &
                cond := Vrai
        sinon
                si Produit (Numéro de produit = Produit (Numéro de produit = Pval)
                        [Produit analogue]) [Etat d'approvisionnement] ≠ 'épuisé'
                alors
                        si quC = 0
                        alors quC:= 1
                        sinon ---
                        &
                        Montant := quC * Produit (Numéro de produit =
                           Produit (Numéro de produit = Pval) [Produit analogue])
                           [Prix Unitaire d'achat]
                        &
                        pour n un Numéro ∉ LigneTemporaire [Numéro de ligne],
                           ajouter (LigneTemporaire; n, Produit (Numéro de produit
                           = Pval) [Produit analogue], quC, Montant)
                        &
                        Pval := Produit (Numéro de produit = Pval) [Produit
                                Analogue])
                        &
                        cond := Vrai
                sinon
                        cond := Faux
        }
```

```
procédure   ValidationDuneProvenance (

{arg}       Cval     : numéroDeClient;

{rés}       var      cond     : boolean
            );

{pré        Client (Numéro de client = Cval) ≠ ∅
            }

{obj        si Client (Numéro de client = Cval) [Catégorie] ≠ 'douteux'
            alors
                     cond := VRAI
                     &
                     insérer (CommandeTemporaire; ... = ⊥, Client émetteur = Cval)
            sinon
                     cond := FAUX
            }


procédure   ValidationDuneCommande  (

{rés}       var      cond     : boolean
            );

{pré        Client (Numéro de client = CommandeTemporaire [Client émetteur])  ≠ ∅

            Client (Numéro de client = CommandeTemporaire [Client émetteur])
            [Catégorie] ≠ 'douteux'

            Count (LigneTemporaire [Numéro de ligne]) > 0

            ∀ n ∈ à LigneTemporaire [Produit commandé],
                     Produit (Numéro de produit = n) ≠ ∅

            ∀ n ∈ à LigneTemporaire [Produit commandé],
                     Produit (Numéro de produit = n) [Etat d'approvisionnement] ≠
                     'EPUISE'
```

```
                ∀ n ∈ à LigneTemporaire [Produit commandé],
                        LigneTemporaire (Produit commandé= n) [Montant de ligne]
                        = Produit (Numéro de produit = n) [Prix unitaire d'achat] *
                                LigneTemporaire (Produit commandé = n) [Quantité
                                commandée]
                }

{obj            ∀ n ∈ à LigneTemporaire [Produit commandé],
                        si Count (LigneTemporaire (Produit commandé = n)) > 1
                        alors
                                QtéIntermédiaire := Σ LigneTemporaire
                                   (Produit commandé = n) [Quantité commandée]
                                &
                                MontantIntermédiaire := Σ LigneTemporaire
                                   (Produit commandé = n) [Montant de ligne]
                                &
                                supprimer (LigneTemporaire; Produit commandé = n)
                                &
                                ajouter (LigneTemporaire; n, QtéIntermédiaire,
                                   MontantIntermédiaire)
                        sinon   ---
                &
                montant := Σ LigneTemporaire [Montant de ligne]
                &
                si Client (Numéro de client = CommandeTemporaire [Client émetteur])
                        [Catégorie] ='COLLABORATEUR'
                alors  rabais := montant * 20%
                sinon  ---
                &
                si (montant  - rabais) ≤ Client (Numéro de client = CommandeTemporaire
                        [Client émetteur]) [Limite d'achat]
                alors
                        cond := VRAI
                        &
                        insérer (CommandeTemporaire; ... = ⊥, Montant total de commande
                                := montant, Rabais de commande := rabais)
                sinon
```

```
                cond := FAUX
        }

procédure   MémorisationDuneCommande (

{rés}       var     Cenr :  NuméroDeCommande;
        );

{pré        Client (Numéro de client = CommandeTemporaire [Client émetteur])  ≠ ∅

            Client (Numéro de client = CommandeTemporaire [Client émetteur])
            [Catégorie] ≠ 'DOUTEUX'

            Count (LigneTemporaire [Numéro de ligne]) > 0

            ∀ n ∈ à LigneTemporaire [Produit commandé], Produit (Numéro de produit
            = n) ≠ ∅

            ∀ n ∈ à LigneTemporaire [Produit commandé], Produit (Numéro de produit
            = n) [Etat d'approvisionnement] ≠ 'EPUISE'

            ∀ n ∈ à LigneTemporaire [Produit commandé], LigneTemporaire (Numéro de
            produit = n) [Montant de ligne] = Produit (Numéro de produit = n) [Prix
            unitaire d'achat] * LigneTemporaire (Produit commandé = n) [Quantité
            Commandée]

            Il n'existe pas {n,m} (n ≠ m) ∈ LigneTemporaire [ProduitCommandé] tel que
            LigneTemporaire (Produit commandé = n) = LigneTemporaire (Produit
            commandé = m)

            CommandeTemporaire [Montant total de commande] := Σ LigneTemporaire
            [Montant de ligne]

            si Client (Numéro de client = CommandeTemporaire [ClientEmetteur])
                    [Catégorie] = 'COLLABORATEUR'
            alors CommandeTemporaire [Rabais de commande]
                    = CommandeTemporaire [Montant total de commande] * 20%
```

CommandeTemporaire [Montant total de commande] - CommandeTemporaire [Rabais de commande] ≤ Client (Numéro de client = CommandeTemporaire [Client émetteur]) [Limite d'achat]
}

{obj pour n un Numéro ∉ Commande [Numéro de commande],
insérer (CommandeTemporaire; ... = ⊥, Numéro de commande := n, Date de commande := DateDuJour)
&
Commande := Commande U CommandeTemporaire
&
∀ m ∈ à LigneTemporaire [Numéro de ligne],
Ligne := Ligne U {{n} U LigneTemporaire (Numéro de ligne = m) [Produit commandé, Quantité commandée, Montant de ligne]}
&
Client (Numéro de client = CommandeTemporaire [Client émetteur]) [Limite d'achat] := Client (Numéro de client = CommandeTemporaire [Client émetteur]) [Limite d'achat] - (Commande (Numéro de commande = n) [Montant total de commande] - Commande (Numéro de commande = n) [Rabais de commande])
}

Il nous reste à définir les procédures de consultation des relations du schéma relationnel. Il en existe une pour chaque attribut non identifiant.

procédure ***NomDe(***

{arg} numC : NuméroDeClient;
{rés} var nomC : nomDeClient
);
{pré Client (Numéro de client = numC) ≠ ∅
}
{obj nomC : = Client (Numéro de client = numC) [Nom de client]
}

```
procédure    AdresseDe(

{arg}        numC    : NumeroDeClient;
{rés}        var     adrC    : adresseDeClient
             );
{pré         Client (Numéro de client = numC) ≠ ∅
             }
{obj         adrC : = Client (Numéro de client = numC) [Adresse de domicile]
             }

procédure    LibelléDe(

{arg}        numP    : NumeroDeProduit;
{rés}        var     libelléP  : libelléDeProduit
             );
{pré         Produit (Numéro de produit = numP) ≠ ∅
             }
{obj         libelléP : = Produit (Numéro de produit = numP) [Libellé de produit]
             }
```

Les procédures de consultation des autres attributs des relations du schéma relationnel se présentent sous le même modèle que les trois procédures décrites ici. Nous ne les reproduisons pas.

VI.4.3. Le Dialogue : Spécification de la tâche

La séparation modulaire entre les fonctions et l'interface permet de spécifier plusieurs interfaces pour une même application. [Ruc 88] a mis en évidence deux interfaces différents pour la phase Enregistrement d'une commande-client : l'une pour la saisie de bons de commande, l'autre pour la saisie de commandes téléphonées.
Une commande téléphonée devra avant tout fournir un maximum d'informations en un minimum de temps pour que l'utilisateur puisse les communiquer au client. Celui-ci pourra dès lors modifier sa commande en fonction des éléments qui lui sont fournis.
La saisie d'un bon de commande pourra se faire de façon plus souple. L'utilisateur ne sera plus contraint par une marche à suivre imposée par le client et il ne sera également pas pressé de la même manière par le temps.

Nous avons choisi de développer l'interface de la commande téléphonée car son interactivité est plus importante que celle de la saisie d'un bon de commande.

Au contraire d'une commande écrite, l'enregistrement d'une commande téléphonée met en interaction l'opérateur et le client qui passe la commande. "Ce dernier est l'utilisateur final de l'application interactive, le destinataire des informations diffusées par l'interface. Dès lors, il est indispensable d'améliorer la qualité des informations qui seront communiquées pour permettre au client de passer une commande valide" [Ruc 88]. L'interface devra donc communiquer les éléments suivants :

- l'affichage du **nom et de l'adresse du client** après l'introduction du numéro de client : il permet la validation du numéro introduit ("Vous appelez-vous bien Pierre Durant ?") et la vérification de l'adresse (le client peut avoir déménagé sans en avoir averti la firme).

- l'affichage du **numéro du client** après l'introduction de son nom et son adresse : ceci est très important car il permet de communiquer au client le numéro par lequel il devra dorénavant passer des commandes.

- l'affichage du **total** et éventuellement du **rabais** de la commande : le client sera naturellement désireux de connaître la somme qu'il devra payer à la réception de sa facture et également du rabais qui lui est offert s'il est membre du personnel;

- l'affichage du **libellé** de chaque produit commandé : pour la saisie d'une commande téléphonée, il est nécessaire d'avoir un "feedback" immédiat (le libellé) après la saisie du

numéro de produit et avant de spécifier la quantité (l'opérateur peut alors dialoguer avec le client pour savoir si le produit est bien celui qu'il désire);

- l'affichage du **montant de ligne** pour chaque ligne commandée : il permet à l'utlisateur de voir la contribution de chaque ligne au montant total de commande et, éventuellement, choisir la (les) ligne(s) à supprimer en cas de dépassement de la limite d'achat.

L'interface devra également veiller à la qualité des informations fournies à l'utilisateur en ce qui concerne les raisons de la non validation d'une commande. Un message (Commande_non_valide) contiendra la limite d'achat du client qui a été dépassée. Cette information n'intéresse pas l'opérateur qui saisit une commande écrite car il n'est pas en communication avec le client. Ce dernier est par contre intéressé de la connaître pour pouvoir, par exemple, supprimer une ligne de commande et ne plus dépasser la limite d'achat.

VI.4.3.1. Spécification des messages interactifs fonctionnels

Les informations contenues dans les messages interactifs fonctionnels proviennent des messages fonctionnels. Leur regroupement doit non plus se faire en fonction des traitements mais de l'utilisateur.

La saisie d'une commande téléphonée correspond à la saisie du client passant la commande, la saisie des lignes de commande et l'affichage du montant à payer pour cette commande. Cela représente trois unités de dialogue pour l'utilisateur.

La saisie d'un client peut elle aussi se diviser en unités de saisie plus élémentaires : le numéro de client, son nom et son adresse. De même pour la saisie d'une ligne : le numéro de produit et la quantité.

Il reste enfin les messages d'erreur sémantique associés à la saisie de la commande, du client et des lignes : client non valide et provenance non valide pour le client, produit non valide et ligne non valide pour la ligne et commande non valide pour la commande.

Cette décomposition en messages interactifs fonctionnels représente un choix de conception. Nous aurions pu en envisager une autre dans laquelle nous aurions uniquement défini, par exemple, des messages interactifs fonctionnels atomiques. Il nous semblé cependant que la commande, le client et la ligne représentaient de réelles unités de dialogue pour l'utilisateur.

Les messages interactifs fonctionnels résultant de notre découpe sont donc :

COMMANDE

CLIENT

Num_client
Nom_client
Adr_domicile
Client_non_valide
Provenance_non_valide

LIGNE

Num_produit
Qu_produit
Produit_non_valide
Ligne_non_valide

SOLDE_À_PAYER
COMMANDE_NON_VALIDE

COMMANDE

Nom : commande;
Définition : une commande est passée par un client; elle comprend des lignes; lors de son enregistrement, on fournit à l'opérateur le numéro de la commande, la date de commande, le montant total, le rabais et le solde à payer (montant total - rabais);
Type : message de saisie/affichage;
Justification : ce message constitue une unité de dialogue car il représente toutes les informations concernant une commande (la commande elle-même, le client émetteur et les lignes de commande);
Contenu :

- Client : message;
- Ligne : message;
- Solde_à_payer : message;
- num_commande : entier;
- date_commande : date;

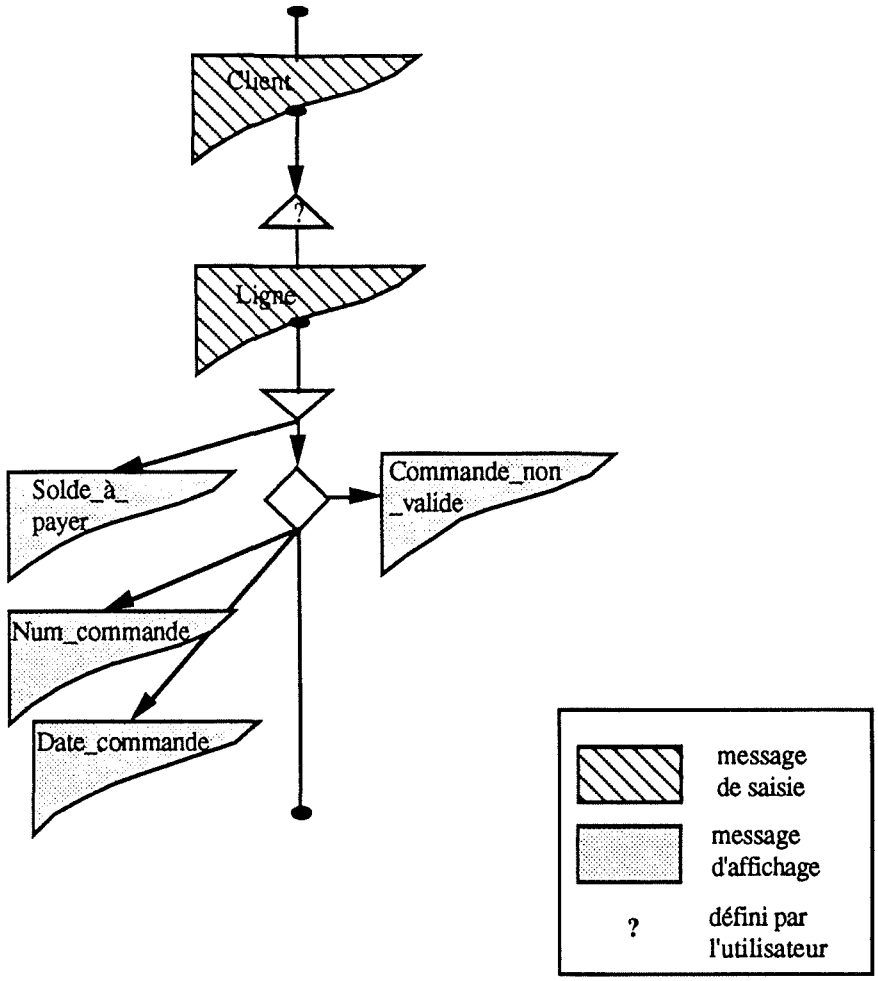
Contraintes syntaxiques : ---
Messages d'erreur sémantique :

- Commande_non_valide

Contraintes sur la présentation : - non définies -;
Opérations de manipulation :

- l'utilisateur peut créer et clôturer le message;

Schéma de la conversation :



CLIENT

Nom : client;

Définition : un client est une personne physique ou morale susceptible de passer des commandes; le message est composé d'un numéro de client, attribué par compostage lors de la création du client, d'un nom et d'une adresse de domicile;

Type : message de saisie/affichage;

Justification : ce message constitue une unité de dialogue car il représente toutes les informations concernant un client (numéro, nom et adresse);

Contenu :

- Num_client : message;
- Nom_client : message;
- Adr_domicile : message;

Contraintes syntaxiques : ---

Messages d'erreur sémantique :

- Client_non_valide;

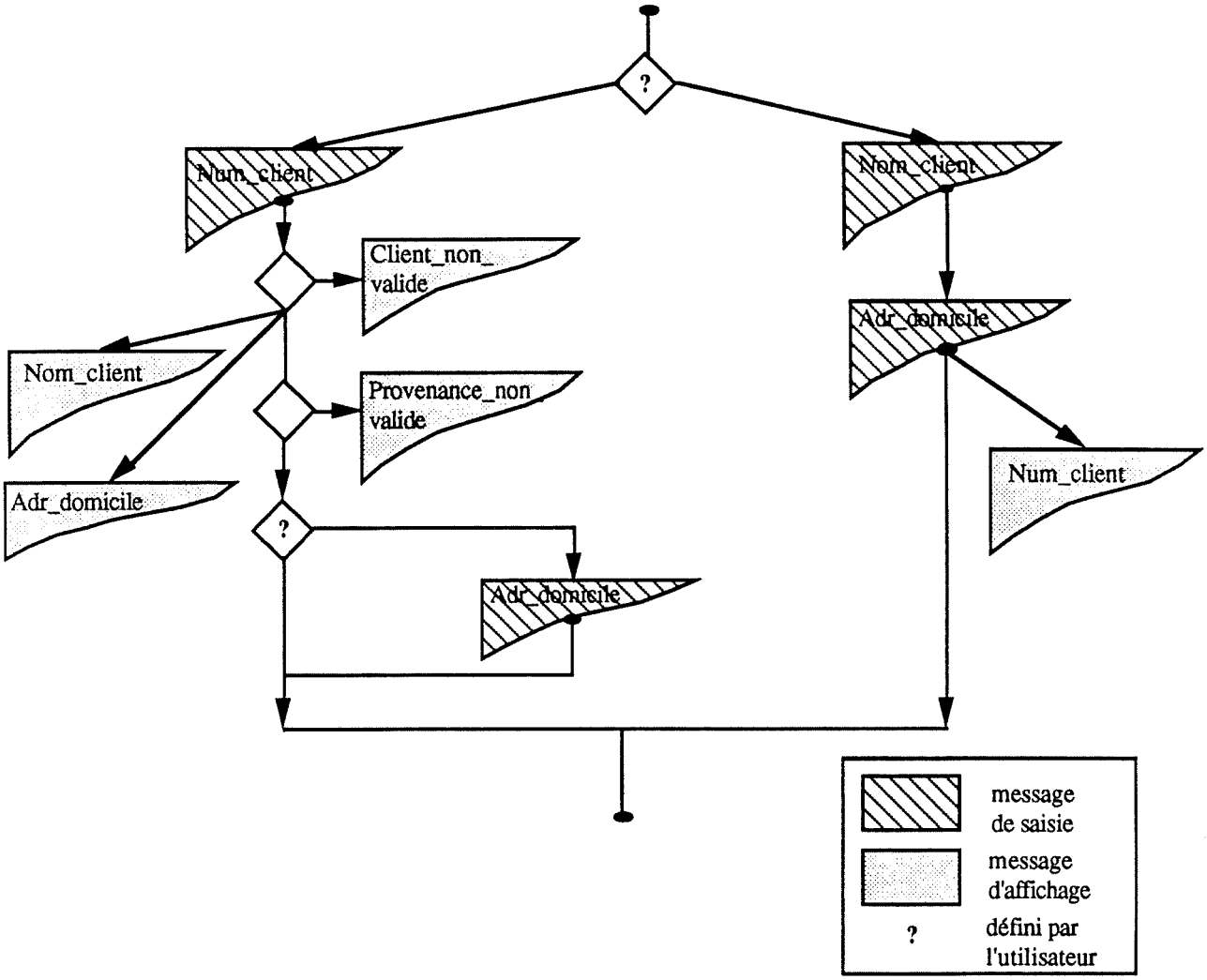
- Provenance_non_valide;

Contraintes sur la présentation : - non définies -;

Opérations de manipulation :

- l'utilisateur peut sélectionner le message Num_client ou sélectionner le message Nom_client;
- l'utilisateur peut clôturer le message;

Schéma de la conversation :



Num_client

Nom : num_client;

Définition : un numéro_client est un numéro attribué par compostage lors de la création du client. Il est identifiant et doit être fourni par l'opérateur afin de retrouver les coordonnées d'un ancien client qui passe une commande;

Type : message de saisie/affichage;

Justification : ce message représente l'information que l'opérateur doit introduire pour retrouver les informations au sujet d'un ancien client qui passe la commande (saisie) ou l'information que le nouveau client devra dorénavant fournir pour passer de nouvelles commandes (affichage);

Contenu :

- numéro : entier;

Contraintes syntaxiques :

- numéro doit être un entier (cfr message d'erreur syntaxique Num_client_entier) ;

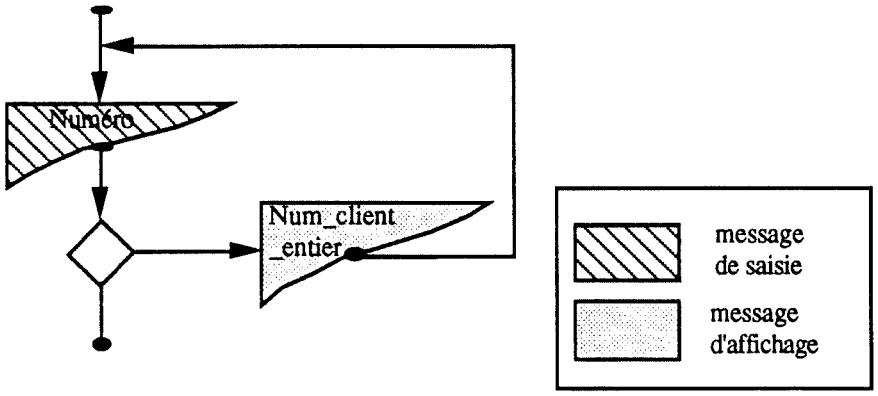
Messages d'erreur sémantique : ---

Contraintes sur la présentation : - non définies -;

Opérations de manipulation :

- l'utilisateur peut affecter une valeur à l'attribut du message ou la corriger;
- l'utilisateur peut clôturer le message lorsque son attribut a reçu une valeur (et que les contraintes syntaxiques sont vérifiées);

Schéma de la conversation :



Nom_client

Nom : nom_client;

Définition : un nom_client est composé du nom du client, son prénom et son titre;

Type : message de saisie/affichage;

Justification : ce message comprend toutes les informations que l'opérateur doit introduire sur le nom d'un client lors de sa création, ou les informations nécessaires à l'opérateur pour vérifier que le numéro de client introduit correspond effectivement au client passant la commande (le numéro n'est pas erroné);

Contenu :

- nom : char[40];
- prénom : char[40];

- titre : char[40];

Contraintes syntaxiques : ---

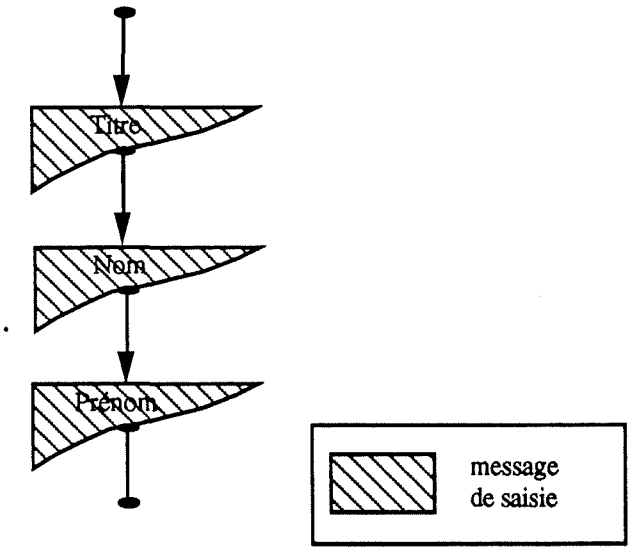
Messages d'erreur sémantique : ---

Contraintes sur la présentation : - non définies -;

Opérations de manipulation :

- l'utilisateur peut affecter une valeur aux attributs du message ou la corriger;
- l'utilisateur peut clôturer le message lorsque tous ses attributs ont reçu une valeur;

Schéma de la conversation :



Adr_domicile

Nom : adr_domicile;

Définition : le message représente l'endroit où réside le client et où devront être acheminées les commandes; une adresse d'un client est constituée d'une rue, un numéro, un code postal et d'une localité;

Type : message de saisie/affichage;

Justification : ce message constitue une unité de dialogue car il permet d'introduire l'adresse du client ou de la modifier;

Contenu :

- rue : char[40];
- numéro : char[6];
- code_postal : char[4];
- localité : char[40];

Contraintes syntaxiques :

- le code_postal doit être un entier positif (cfr message d'erreur syntaxique Code

postal_entier);

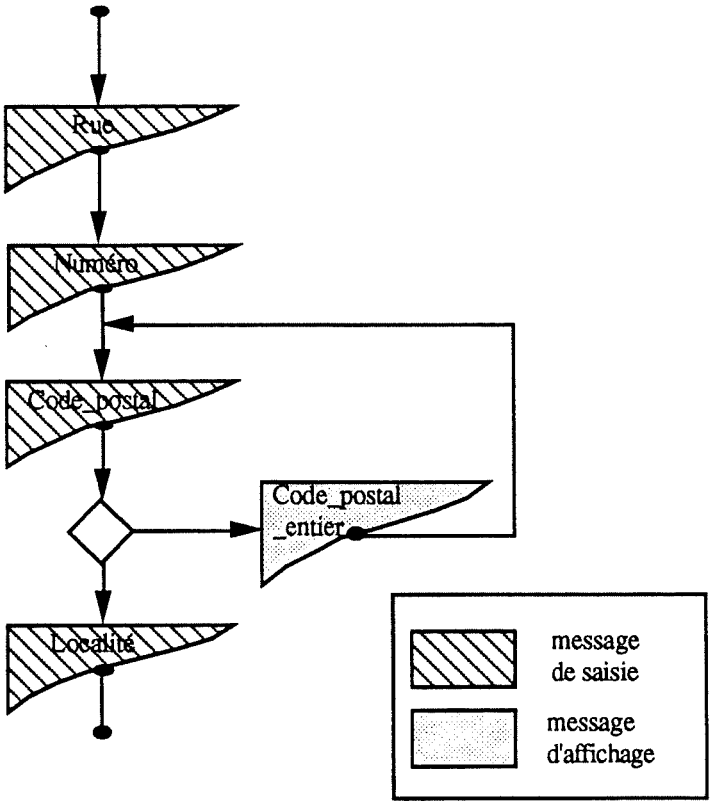
Messages d'erreur sémantique : ---

Contraintes sur la présentation : l'adresse devra figurer sur deux lignes séparées avec une première pour la rue et le numéro et une deuxième pour le code postal et la localité;

Opérations de manipulation :

- l'utilisateur peut affecter une valeur aux attributs du message ou la corriger;
- l'utilisateur peut clôturer le message lorsque tous ses attributs ont reçu une valeur (et que les contraintes syntaxiques sont vérifiées);

Schéma de la conversation :



Client_non_valide

Nom : client_non_valide;

Définition : le message client_non_valide est le texte qui apparaît lorsque l'opérateur a introduit un numéro de client inexistant;

Type : message d'affichage;

Justification : le message a pour but de faire connaître à l'utilisateur que le numéro de client introduit n'existe pas; il fait part d'une erreur commise;

Contenu :

- un texte expliquant que le numéro du client introduit n'existe pas : "Le numéro de client

que vous avez introduit n'existe pas. Veuillez essayer avec une nouvelle valeur";

Contraintes syntaxiques : ---

Messages d'erreur sémantique : ---

Contraintes sur la présentation : la présentation et la position du message devront être similaires aux autres messages d'erreur;

Opérations de manipulation :

- l'utilisateur ne peut que clôturer ce message;

Schéma de la conversation : ---

Provenance_non_valide

Nom : provenance_non_valide;

Définition : le message provenance_non_valide signale à l'opérateur que le client dont il a introduit les références est de catégorie "douteux" pour l'entreprise;

Type : message d'affichage;

Justification : le message a pour but de faire connaître à l'utilisateur que le client est "douteux"; il lui fait part de l'erreur commise;

Contenu :

- un texte signalant que client introduit est "douteux" : "Le numéro que vous avez introduit correspond à un client faisant partie de la liste noire. Il ne peut plus effectuer de commande";

Contraintes syntaxiques : ---

Messages d'erreur sémantique : ---

Contraintes sur la présentation : la présentation et la position du message devront être similaires aux autres messages d'erreur;

Opérations de manipulation :

- l'utilisateur ne peut que clôturer ce message;

Schéma de la conversation : ---

LIGNE

Nom : ligne;

Définition : une ligne représente la promesse de livraison d'un produit commandé en une certaine quantité; elle est composée d'un numéro de produit, dun libellé de produit, d'une quantité et d'un montant de ligne;

Type : message de saisie/affichage;

Justification : ce message comprend toutes les informations concernant une ligne de commande (numéro de produit, libellé, quantité et montant de ligne);

Contenu :

- Num_produit : message;
- libellé_produit : char[50];
- Qu_produit : message;
- montant_de_ligne : réel;

Contraintes syntaxiques :

- montant_de_ligne = prix unitaire d'achat du produit commandé * Qu_produit

Messages d'erreur sémantique :

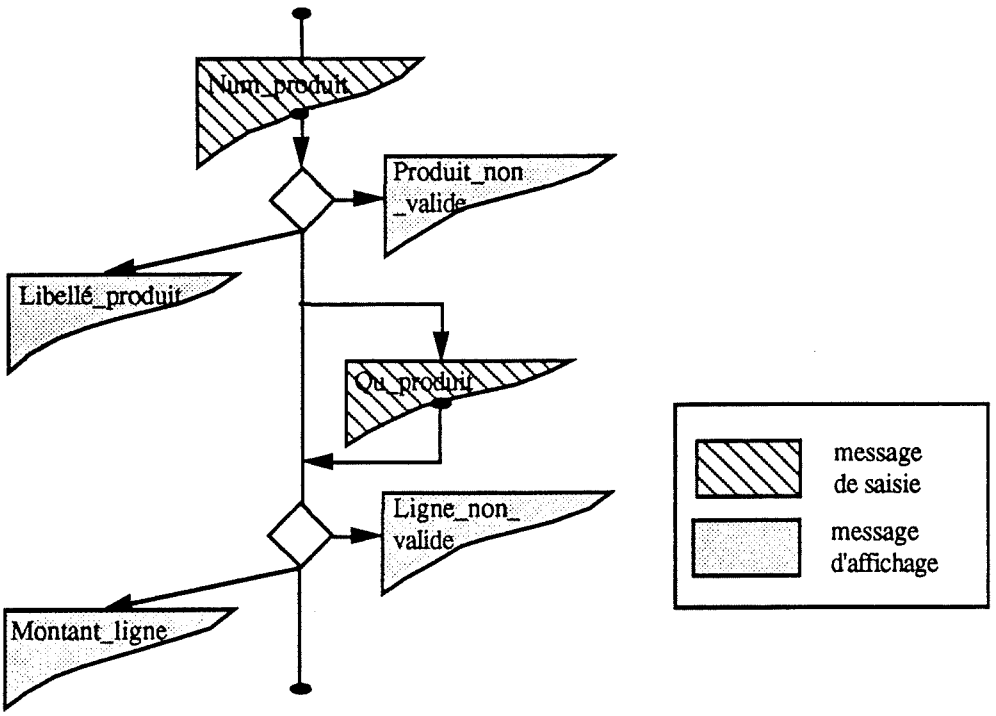
- Produit_non_valide;
- Ligne_non_valide;

Contraintes sur la présentation : libellé_produit devra être affiché bien en correspondance avec le numéro de produit num_produit pour éviter à l'utilisateur un effort de visualisation;

Opérations de manipulation :

- l'utilisateur peut créer le message Qu_produit;
- l'utilisateur peut clôturer le message;

Schéma de la conversation :



Num_produit

Nom : Num_produit;
Définition : le message est constitué d'un numéro à l'aide duquel l'entreprise répertorie ses produits;
Type : message de saisie/affichage;
Justification : ce message a pour but de saisir le numéro du produit que le client désire commander; il constitue une unité de saisie pour l'opérateur car celui-ci doit être capable de vérifier que le numéro de produit correspond bien à la volonté du client par l'affichage du libellé;
Contenu :

- numéro : entier;

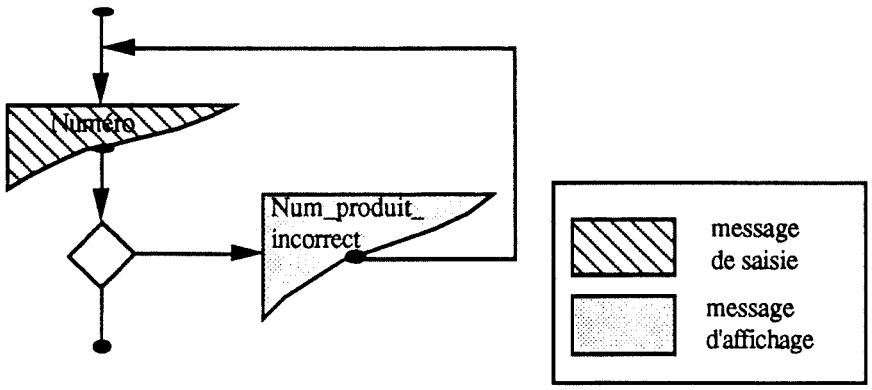
Contraintes syntaxiques :

- le numéro de produit doit être un entier positif dont le dernier chiffre doit être égal au reste de la division des 7 premiers chiffres par 7 (cfr message d'erreur syntaxique Num_produit_incorrect);

Messages d'erreur sémantique : ---
Contraintes sur la présentation : -non définies-;
Opérations de manipulation :

- l'opérateur peut affecter une valeur à son attribut, la corriger;
- l'opérateur peut clôturer le message lorsque son attribut a reçu une valeur (et que les contraintes syntaxiques sont vérifiées);

Schéma de la conversation :



Qu_produit

Nom : qu_produit;
Définition : le message représente la quantité de produit que le client passant la commande

désire;

Type : message de saisie/affichage;

Justification : ce message a pour but de saisir la quantité du produit à commander si le client la spécifie; il constitue une unité de saisie car il permet à l'utilisateur de spécifier la quantité du produit précédemment entré et dont il connait à présent le libellé;

Contenu :

- quantité : entier strictement positif;

Contraintes syntaxiques :

- la quantité doit être un entier strictement positif (cfr message d'erreur syntaxique Quantité_entière positive);

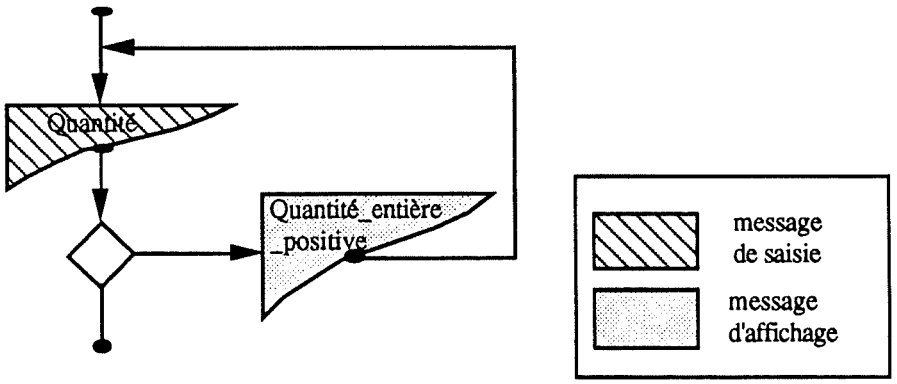
Messages d'erreur sémantique : ---

Contraintes sur la présentation : - non définies -;

Opérations de manipulation :

- l'utilisateur peut affecter une valeur à l'attribut du message, la corriger ou la supprimer;
- l'utilisateur peut clôturer le message à tout moment si son attribut n'a pas de valeur, lorsque les contraintes syntaxiques sont vérifiées sinon;

Schéma de la conversation :



Produit_non_valide

Nom : produit_non_valide;

Définition : le message produit_non_valide signale à l'opérateur qu'il a introduit un numéro de produit inexistant;

Type : message d'affichage;

Justification : le message a pour but de faire connaître à l'utilisateur que le numéro de produit introduit n'existe pas; il lui fait part de l'erreur commise;

Contenu :

- un texte signalant que le numéro de produit introduit n'existe pas : "Le numéro que

vous avez introduit ne correspond à aucun produit existant. Veuillez essayer une nouvelle valeur";

Contraintes syntaxiques : ---

Messages d'erreur sémantique : ---

Contrainte sur la présentation : la présentation et la position du message devront être similaires aux autres messages d'erreur;

Opérations de manipulation :

- l'utilisateur ne peut que clôturer ce message;

Schéma de la conversation : ---

Ligne_non_valide

Nom : ligne_non_valide;

Définition : le message ligne_non_valide signale à l'opérateur que le produit commandé est épuisé et qu'il ne peut être remplacé par un produit substitut;

Type : message d'affichage;

Justification : le message a pour but de faire connaître à l'utilisateur que le produit de la ligne est épuisé; il lui fait part de l'erreur commise;

Contenu :

- un texte signalant que le produit est épuisé : "Le produit que vous avez introduit ne peut être commandé; il est épuisé et ne peut être substitué";

Contraintes syntaxiques : ---

Messages d'erreur sémantique : ---

Contrainte sur la présentation : la présentation et la position du message devront être similaires aux autres messages d'erreur;

Opérations de manipulation :

- l'utilisateur ne peut que clôturer ce message;

Schéma de la conversation : ---

SOLDE_À_PAYER

Nom : solde_à_payer;

Définition : le message représente le solde à payer, c'est-à-dire soit le montant de la commande si le client passant la commande n'est pas 'COLLABORATEUR', soit le montant de la commande réduit du rabais si le client est 'COLLABORATEUR';

Type : message d'affichage;

Justification : le message a pour but de faire connaître le montant total de la commande, éventuellement réduit du rabais, il constitue une unité d'affichage pour l'utilisateur puisqu'il correspond au montant que le client devra effectivement débourser pour payer la commande;
Contenu :

- montant_total : réel
- rabais : réel
- solde : réel

Contraintes syntaxiques :

- solde = montant_total -rabais

Messages d'erreur sémantique : ---
Contraintes sur la présentation : le montant_total, le rabais et le solde devront apparaître dans cet ordre sous les montants de ligne;
Opérations de manipulation:

- l'opérateur ne peut pas manipuler ce message car il est entièrement géré par l'interface;

Schéma de la conversation : ---

COMMANDE_NON_VALIDE

Nom : commande_non_valide;
Définition : le message commande_non_valide signale à l'opérateur que le solde à payer (montant total - rabais de commande) dépasse la limite d'achat du client;
Type : message d'affichage;
Justification : le message a pour but de faire connaître à l'utilisateur que le montant total de commande, réduit du rabais éventuel, est supérieur à la limite d'achat du client; il constitue une unité d'affichage pour l'opérateur puisqu'il lui fait part de l'erreur commise;
Contenu :

- un texte signalant que le montant total de la commande est supérieur à la limite d'achat du client en lui communiquant cette limite;

Contrainte sur la présentation : le texte du message devra expliquer clairement l'erreur commise en affichant la limite d'achat; la présentation et la position du message devront être similaires aux autres messages d'erreur;
Opérations de manipulation :

- l'utilisateur ne peut que clôturer ce message;

VI.4.3.2. Spécification du schéma de la conversation global

Dans notre cas, étant donné que le message interactif fonctionnel Commande englobe

toutes les informations concernant la tâche, la conversation se réduit à la seule opération de création du message.

VI.4.3.3. Spécification des messages purement interactifs

Un message purement interactif ne concerne que l'interface; les fonctions n'en n'ont pas connaissance. Voici les messages que nous avons spécifiés pour la saisie d'une commande téléphonée:

a) Messages de contrôle

Saisie_Ancien_ou_Nouveau

Nom : saisie_ancien_ou_nouveau;
Définition : ce message permet à l'opérateur de choisir d'introduire les coordonnées d'un ancien client (numéro de client) ou celles d'un nouveau (nom de client);
Type : message de contrôle;
Est déclenché : à la création du message Client;
Justification : ce message permet à l'utilisateur de choisir le message qu'il désire saisir selon que le client soit nouveau ou ancien. Rappelons qu'il sera inutile dans un environnement tel que la manipulation directe;
Contenu : les alternatives offertes (choix de la saisie du Nom_client ou Num_client);
Contraintes sur la présentation : il serait souhaitable d'offrir à l'opérateur la possibilité d'effectuer son choix sans l'aide de ce message. L'utilisateur pourrait passer d'un message à l'autre à l'aide d'une fonction;
Opérations de manipulation :

- l'utilisateur doit sélectionner l'alternative choisie;

Saisie_Changement_Adresse

Nom : saisie_changement_adresse;
Définition : ce message permet à l'opérateur de déterminer si l'adresse du client (ancien) doit être modifiée ou non;
Type : message de contrôle;
Est déclenché : lorsque l'opérateur a introduit un ancien client valide et non "douteux";
Justification : ce message donne à l'opérateur la possibilité de modifier l'adresse d'un ancien client;

Contenu : les alternatives offertes (modifier l'adresse du client ou non);
Contraintes sur la présentation : il serait souhaitable d'offrir à l'utilisateur la possibilité d'effectuer son choix sans l'aide de ce message;
Opérations de manipulation :

- l'utilisateur doit sélectionner l'alternative choisie;

Saisie_terminée

Nom : saisie_terminée;
Définition : ce message permet à l'utilisateur de signaler à l'interface qu'il a terminé l'introduction des lignes de la commande;
Type : message de contrôle;
Est déclenché : il peut l'être lorsque l'opérateur a introduit une ligne de commande valide;
Justification : ce message est nécessaire pour pouvoir satisfaire le point de synchronisation de la structure itérative avant de valider la commande; c'est en effet l'opérateur qui signale la fin de la saisie des lignes de commande (il détermine leur nombre);
Contenu : un texte explicatif;
Contraintes sur la présentation : le message devra être présenté par une icône, un menu ou une commande, manipulable dès qu'une ligne de commande a été validée;
Opérations de manipulation :

- l'utilisateur peut sélectionner et clôturer le message;

Saisie_annulée

Nom : saisie_annulée;
Définition : ce message permet à l'opérateur de signaler au dialogue l'annulation de la commande qu'il était en train d'introduire;
Type : message de contrôle;
Est déclenché : à n'importe quel moment lorsque l'utilisateur le désire;
Justification : Ce message donne la possibilité à l'utilisateur d'annuler des saisies erronées; il constitue une unité de saisie pour l'opérateur car il lui permet de signaler à l'interface son désir d'annuler la saisie de la commande en cours;
Contenu : un texte explicatif;
Contraintes sur la présentation : le message devra être présenté par une icône visible en permanence, un menu ou une commande disponible à tout moment;
Opérations de manipulation :

- l'utilisateur peut sélectionner et clôturer le message;

Quitter_saisie

Nom : quitter_saisie;
Définition : ce message permet à l'opérateur de signaler au dialogue qu'il annule la commande qu'il était en train d'introduire et qu'il désire quitter l'application interactive;
Type : message de contrôle;
Est déclenché : à n'importe quel moment lorsque l'utilisateur le désire;
Justification : ce message donne la possibilité à l'utilisateur de quitter l'application interactive quelque soit sa position dans le schéma de la conversation;
Contenu : un texte explicatif;
Contraintes sur la présentation : le message devra être présenté par une icône visible en permanence, un menu ou une commande activable à tout moment;
Opérations de manipulation :
- l'utilisateur peut sélectionner et clôturer le message

b) Messages d'erreur syntaxique

Nous avons rassemblé ici les messages d'erreur syntaxique consécutifs aux contraintes syntaxiques définies dans les messages interactifs fonctionels :

Num_client_entier

Nom : num_client_entier;
Définition : ce message a pour but de faire connaître à l'opérateur qu'il a introduit un numéro de client non entier;
Type : message d'erreur syntaxique;
Est déclenché : après la saisie du message Num_client si le numéro du client n'est pas un entier;
Justification : ce message constitue une unité d'affichage pour l'utilisateur puisqu'il l'informe d'une erreur commise;
Contenu : un texte : "le numéro de client que vous avez entré doit être un entier";
Contraintes sur la présentation : ce message pourra être remplacé par une gestion des caractères introduits pour Num_client de sorte que seuls les chiffres soient acceptés. De plus, il pourrait être supprimé en considérant qu'il s'agit alors d'un numéro de client inexistant (affichage du message Client_non_valide);

Opérations de manipulation :

- l'utilisateur ne peut que clôturer le message;

Code_postal_entier

Nom : adr_code postal_entier;
Définition : ce message a pour but de faire connaître à l'opérateur qu'il a introduit un code postal non entier;
Type : message d'erreur syntaxique;
Est déclenché : après la saisie du message Adr_domicile si le code postal de l'adresse n'est pas un entier;
Justification : ce message constitue une unité d'affichage pour l'utilisateur puisqu'il l'informe d'une erreur commise;
Contenu : un texte signalant que le code postal introduit doit être un entier positif "le code postal de l'adresse doit être un nombre. Veuillez introduire une nouvelle valeur.";
Contraintes sur la présentation : le message devra expliquer clairement l'erreur commise;
Opérations de manipulation :

- l'utilisateur ne peut que clôturer le message;

Quantité_entière_positive

Nom : quantité_entière_positive;
Définition : ce message a pour but de faire connaître à l'opérateur qu'il a introduit une quantité à commander non entière positive;
Type : message d'erreur syntaxique;
Est déclenché : après la saisie du message qu_produit si la quantité à commander du produit n'est pas un entier ou n'est pas positive;
Justification : ce message constitue une unité d'affichage pour l'utilisateur puisqu'il l'informe de l'erreur commise;
Contenu : un texte signalant que la quantité à commander doit être un entier positif: "Veuillez introduire une quantité entière positive";
Contraintes sur la présentation : ce message pourra être remplacé par une gestion des caractères introduits pour la quantité à commander de sorte que seuls les chiffres soient acceptés;
Opérations de manipulation :

- l'utilisateur ne peut que clôturer le message;

Num_produit_incorrect

Nom : num_produit_incorrect;
Définition : ce message a pour but de faire connaître à l'opérateur qu'il a introduit un numéro de produit incorrect car il n'est pas entier ou le reste de la division des 7 premiers par 7 n'est pas égale au huitième;
Type : message d'erreur syntaxique;
Est déclenché : après la saisie du message Num_produit si le numéro de produit n'est pas un entier ou si le dernier chiffre n'est pas égal au reste de la division des 7 premiers chiffres par 7;
Justification : ce message constitue une unité d'affichage pour l'utilisateur car il l'informe de l'erreur commise;
Contenu : un texte signalant que le numéro du produit introduit est incorrect : "Le numéro de produit que vous avez introduit est incorrect. Veuillez taper un nouveau numéro";
Contraintes sur la présentation : le message pourrait être supprimé en considérant qu'il s'agit d'un numéro de produit inexistant et en affichant le message Produit_invalide;
Opérations de manipulation :

- l'utilisateur ne peut que clôturer le message;

c) Messages d'aide

Nous avons décidé de nous contenter d'offrir à l'utilisateur un minimum d'aide car notre but n'est pas de créer une application interactive de haut niveau. La spécification de messages d'aide incluant la gestion du contexte, d'un tableau de bord... se fera de la même manière que le message d'aide que nous spécifions ci-dessous. Ils contiendront néanmoins des informations manipulées par les fonctions.

Aide_sémantique

Nom : aide_sémantique;
Définition : ce message affiche à l'opérateur des informations susceptibles de mieux lui faire comprendre la sémantique de l'application interactive;
Type : message d'aide;
Est déclenché : à n'importe quel moment, lorsque l'opérateur le désire. Pour cela, il devra sélectionner le message;

Justification : ce message constitue une unité d'affichage car il présente à l'utilisateur un ensemble d'informations concernant les fonctionnalités de l'application interactive;
Contenu : un texte expliquant les mécanismes qui régissent l'application proprement dite;
Contraintes sur la présentation : le message pourra être décomposé en plusieurs parties, offrant à l'utilisateur le choix de poursuivre la lecture ou de revenir à l'exécution de l'application.
Opérations de manipulation :

- l'utilisateur peut créer, sélectionner, ranger ou clôturer le message;

VI.4.3.4. Validation de la tâche par rapport à la dynamique des traitements

Cette phase consiste en la vérification de la correspondance entre les informations contenues dans les messages fonctionnels et les messages interactifs fonctionnels.

D'une part, chaque information contenue dans un message externe fonctionnel d'entrée doit se retrouver dans un message interactif fonctionnel de saisie :

Messages fonctionnels d'entrée	**Messages interactifs fonctionnels (saisie)**
Numéro de client	Num_client
Nom de client	Nom_client
Adresse de domicile	Adr_domicile
Numéro de produit	Num_produit
Quantité commandée	Qu_produit

D'autre part, les informations contenues dans les messages interactifs fonctionnels d'affichage doivent appartenir à des messages fonctionnels de sortie :

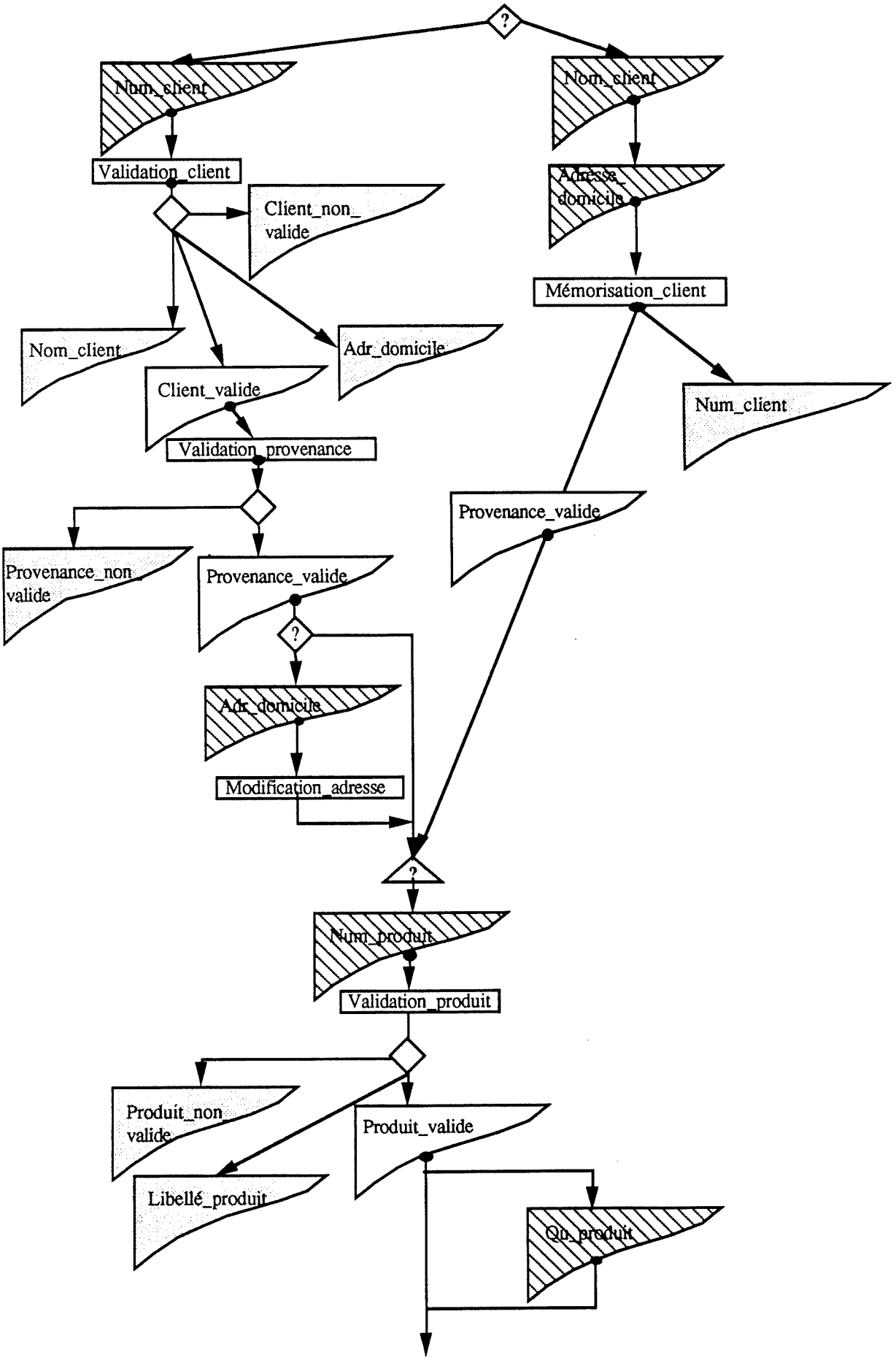
Messages interactifs fonctionnels (affichage)	**Messages fonctionnels de sortie**
Solde_à_payer	Commande valide
Num_commande	Commande valide
Date_commande	Commande valide
Num_client	Provenance valide
Nom_client	Client valide
Adr_domicile	Client valide

Libellé_produit	Produit valide
Montant_de_ligne	Ligne valide

Notons que le message interactif fonctionnel Solde_à_payer comprend une information (Solde) n'appartenant à aucun message fonctionnel de sortie. Ce message est néanmoins valide puisque cette information est calculée à partir du Montant total et du Rabais, informations qui appartiennent, elles, à un message fonctionnel de sortie (Commande valide).

VI.4.3.5. Dynamique d'implémentation

Le schéma de la dynamique d'implémentation constitue l'intégration des schémas de la conversation et de la dynamique des traitements, en donnant la priorité aux contraintes issues de la conversation. L'interface de la commande téléphonée requiert des feed-back d'informations très rapides (nom, adresse du client, libellé de produit, ...). Les traitements sont par conséquent déclenchés après la saisie de messages interactifs fonctionnels de bas niveau (num_client déclenche la validation du client, num_produit la validation du produit, ...) Le schéma de la dynamique d'implémentation que nous obtenons suite à ces considérations est présenté à la figure VI.4.


?
Num_client
Validation_client
Client_non_valide
Nom_client
Client_valide
Adr_domicile
Validation_provenance
Provenance_non_valide
Provenance_valide
?
Adr_domicile
Modification_adresse
?
Num_produit
Validation_produit
Produit_non_valide
Libellé_produit
Produit_valide
Qu_produit
Nom_client
Adresse_domicile
Mémorisation_client
Num_client
Provenance_valide

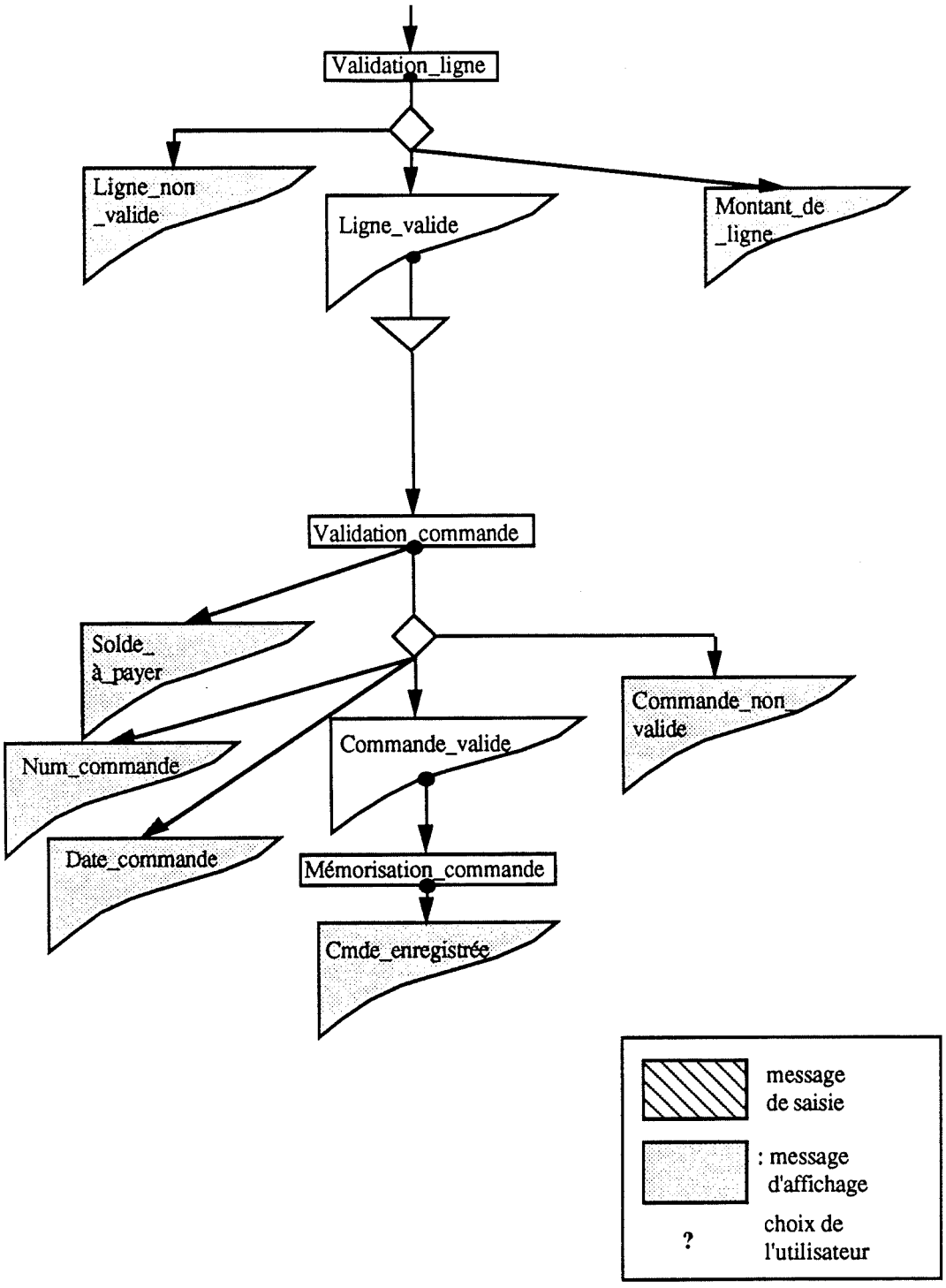


Figure VI.4. : Dynamique d'implémentation

VI.5. LA CONCEPTION PHYSIQUE

L'implémentation de la phase d'Enregistrement d'une Commande-client a été réalisée de manière intuitive, c'est-à-dire qu'aucune méthode précise n'a été suivie (ni architecture physique des traitements, ni spécification des objets interactifs du dialogue, ...). Notre objectif était double. D'une part, nous avons tenté, pour les données, les traitements et le dialogue de respecter, dans la mesure du possible, les résultats de la conception logique et, par là-même, les spécifications conceptuelles de la phase; d'autre part, nous avons voulu utiliser au mieux les potentialités offertes par les E4Gs.

Nous ne vous présentons pas, ici, les résultats de la réalisation physique de la phase dans les trois E4Gs utilisés pour celle-ci (Magic, Quatrième Dimension et Rally); cela nous paraît peu opportun dans la mesure où la syntaxe de ces langages vous est sans doute étrangère ou, du moins, non familière. Par contre, afin d'offrir une vue plus synthétique et générale de cette réalisation, nous en présentons les résultats en nous référant aux concepts du modèle descriptif global des E4Gs, concepts que nous avons décrits aux chapitres III et IV.

Nous structurons la présentation des résultats de l'implémentation physique de la phase de la même manière qu'ont été structurées son analyse conceptuelle et sa conception logique, à savoir en terme de données, de traitements et de dialogue. Cependant, nous y intégrons l'implémentation distincte de la fonction de pilotage, fonction nouvelle offerte par les E4Gs étudiés. Intuitivement, nous avons ordonnancé sa réalisation de manière suivante : données, dialogue, traitements et pilotage. Ce choix intuitif semble se justifier au regard de la modélisation des E4Gs avancés (cfr figure IV.2).

VI.5.1. Les données : la structuration de la base de données

La définition des structures de données dans les E4Gs envisagés n'a pas posé de problème. Les tables ou relations de la base de données sont presque identiques aux tables ou relations du schéma relationnel, à l'exception des attributs décomposables qui ont été remplacés par les attributs qui les composaient :

Client (Numéro de client, Nom de client, Prénom de client, Titre, Rue, Numéro de rue, Code postal, Localité, Limite d'achat, Catégorie)

Produit (Numéro de produit, Libellé de produit, Unite d'achat, Prix unitaire d'achat, Etat

d'approvisionnement, Produit analogue)

Commande (Numéro de commande, Date de commande, Montant total de commande, Rabais de Commande, Client émetteur)

Ligne (Commande concernée, Produit Commandé, Quantité commandée, Montant de ligne)

Notons que les relations temporaires (LigneTemporaire et CommandeTemporaire) ne doivent pas être implémentées par le développeur car elles sont prises en charge automatiquement par l'environnement.

Les types des champs des relations décrites dans le dictionnaire de données sont ceux définis en VI.4.1. Les clés d'accès à ces relations sont déduites des identifiants des relations du schéma relationnel : Numéro de client (relation Client), Numéro de commande (relation Commande), Numéro de Produit (relation Produit) et Numéro de commande/Numéro de produit (relation Ligne).
Ces clés d'accès constituent des clés d'accès uniques (primary key) pour les relations Client, Produit et Commande. Elles traduisent donc également l'identifiant de ces relations. Par contre, la clé d'accès de la relation Ligne (Numéro de commande/Numéro de produit) n'est pas unique. L'identifiant de cette relation est vérifié dynamiquement par les traitements. La raison en est que l'interface est directement lié à la base de données et que la mémoire temporaire est gérée par l'environnement au sein de la base de données et non de l'interface. Les lignes de commande sont donc enregistrées temporairement dans la base de données, deux n-uplets Ligne d'une même commande pouvant temporairement concerner le même produit. Ce n'est qu'avant l'enregistrement définitif de la commande, et donc de ses lignes, qu'un traitement de concaténation des lignes portant sur le même produit, vérifiera dynamiquement l'identifiant de la relation Ligne.

Les contraintes d'intégrité portant sur les valeurs de certains champs des relations de l'application peuvent être prises en charge par le dictionnaire des données, grâce aux primitives de contrôle sur les données (cfr III.3.1.2); ce sont les contraintes du type 'appartenance à une liste de valeurs' :

- dom (Catégorie) = {'REGULIER', 'NOUVEAU', 'ANCIEN', 'DOUTEUX', 'COLLABORATEUR'}
- dom (Etat d'approvisionnement) = {'EN STOCK', 'EN AVIS D'ATTENTE', 'EPUISE'}

Les différentes autorisations d'accès liées aux records des relations de la BD sont les suivantes :

- autorisation d'insérer un Client, une Commande, une Ligne de Commande;
- autorisation de modifier un Client;
- autorisation de modifier la rue, le numéro de rue, le code postal, la localité, la limite d'achat et la catégorie d'un Client.

VI.5.2. Le dialogue : la description des Formulaires

La conception physique du dialogue comprend la traduction des messages interactifs en objets interactifs. Cette traduction nous donne :

VI.5.2.1. Les objets interactifs correspondant aux messages interactifs fonctionnels

Dans notre exemple, les messages interactifs fonctionnels (cfr III.4.3) sont intégrés au sein de deux formulaires représentant, l'un, la transaction d'Enregistrement de la commande, l'autre, les transactions de Création ou de Mise à jour effectuées sur le client.

LE FORMULAIRE D'ENREGISTREMENT D'UNE COMMANDE

Il représente le support de la commande téléphonée et comprend les Blocs/Groupes suivants :

Un **Bloc/Groupe principal** (père) **Commande** correspondant à la **Relation Commande.** Cet objet interactif représente le message interactif fonctionnel Commande (Client, Lignes et attributs à afficher). Néanmoins, il ne comprend qu'une partie du message interactif fonctionnel Client, celle qui ne correspond pas à une transaction sur le client (c'est-à-dire la saisie d'un ancien client sans changement d'adresse). Le reste du message est représenté au sein de l'objet interactif Formulaire Création/Mise à jour d'un Client.
Ce B/G contient une seule occurence de l'entité Commande, au travers des champs suivants : le Numéro de commande, le Numéro de client (Client émetteur), la Date de commande, le Rabais de commande et le Montant total de commande. Ce Bloc/Groupe contient également les champs Nom de client, Prénom de client, Titre, Rue, Numéro de rue, Code postal, Localité de la Relation Client. Enfin, il comprend un champ représentant une variable : le Solde à payer.

Chacun de ces champs possède un format. Ce format doit être compatible avec le type du champ de la relation auquel est lié le champ du Bloc/Groupe. Décrire ici ces formats ne comporte pas un grand intérêt dans l'optique de notre expérimentation.

Les autorisations d'accès à ces champs sont les suivantes :

- le *Numéro de commande*, la *Date de commande*, le *Rabais de commande*, le *Montant total de commande* et le *Solde à payer* : champs d'affichage[1];
- le *Numéro de client* : autorisation d'insérer et de modifier;
- le *Nom du client*, le *Prénom du client*, le *Titre*, la *Rue*, le *Numéro de rue*, le *Code postal*, la *Localité* : champs d'affichage[2].

Le champ Numéro de client n'est pas obligatoire[3].

Le B/G Commande possède un B/G /fils :

un **Bloc/Groupe Ligne** correspondant à la **Relation Ligne**. Cet objet interactif représente le message interactif fonctionnel Ligne. Il comprend les occurences des lignes de la commande, composées chacune des champs Numéro de commande (Commande concernée), Numéro de produit (Produit commandé), Quantité commandée et Montant de ligne. Le lien entre les B/G père et fils s'effectue via les champs 'Numéro de commande' des deux B/G. Ce B/G comprend également le champ Libellé de produit de la Relation Produit.
La remarque exprimée pour le Bloc/Groupe Commande concernant les formats reste d'application pour les champs de ce B/G.

Les autorisations d'accès liées aux champs de ce B/G sont les suivantes :

- le *Numéro de produit* : autorisation d'insérer et de modifier;
- la *Quantité commandée* : autorisation d'insérer, de modifier et de supprimer;
- le *Libellé de produit* et le *Montant de ligne* : champs d'affichage.

Le champ Numéro de commande est destiné à faire la liaison entre les 2 B/G (Commande et Ligne); il ne doit pas être affiché dans le B/G Ligne puisqu'il l'est déjà

---

1 Un champ d'affichage ne peut être saisi par l'utilisateur (interdiction d'insérer, de modifier et de supprimer).

2 Ces champs sont des champs d'affichage car ils ne correspondent pas à la relation principale du B/G et ne peuvent être saisis, modifiés ou supprimés (cfr III.3.2.1).

3 Nous avons voulu avoir un même formulaire pour saisir une commande passée par un ancien ou un nouveau client. L'utilisateur signifie qu'il veut saisir un nouveau client en quittant le champ Numéro de client sans lui avoir attribué de valeur.

dans le B/G Commande.

Il nous reste encore à définir le séquencement des objets interactifs de saisie composant le B/G. Ce séquencement correspond à l'ordre dans lequel ces objets seront activés lors de l'exécution du F/R. Cet ordre doit respecter les schémas de la conversation interne associés au message interactif Commande (cfr VI.4.3.1). Le séquencement que nous proposons est le suivant : le Numéro de client, suivi du B/G Ligne. Au sein de ce B/G Ligne, le séquencement choisi est : le Numéro de produit (Produit commandé) suivi de la Quantité commandée.

De même, nous avons choisi une disposition des objets interactifs sur l'écran la plus agréable possible pour l'utilisateur, tout en respectant les contraintes sur la présentation définies au sein des messages interactifs fonctionnels correspondants (cfr VI.4.3.1). Cette disposition coïncide notamment avec l'ordre de saisie normal des informations constituant la commande.

Elle est représentée à la figure VI.5.

Enregistrement d'une Commande-Client

Numéro de Commande : _____ Date de Commande : __/__/____

Numéro du Client : _____

Nom du Client : _____ _____ _____

Rue : _____ n° : ___

Code Postal : _____ Localité : _____

Numéro du Produit	Libellé du Produit	Quantité Commandée	Montant de Ligne
____	_____	_____	_____
____	_____	_____	_____
____	_____	_____	_____
____	_____	_____	_____

Montant Total : _____

Rabais : _____

Solde à payer : _____

Figure VI.5. : Formulaire d'Enregistrement d'une Commande

LE FORMULAIRE DE CRÉATION / MISE À JOUR D'UN CLIENT

Ce formulaire abrite les transactions d'insertion d'un nouveau client ou de mise à jour de l'adresse d'un client et comprend un seul Bloc/Groupe :

Le **Bloc/Groupe Client** correspondant à la **Relation Client.** Cet objet interactif représente les attributs du message interactif fonctionnel Client qui abritent une transaction

sur le client. Il contient une seule occurence de l'entité Client, au travers des champs suivants : le Numéro de Client, le Titre du client, le Nom du client, le Prénom du client, la Rue, le Numéro, le Code postal et la Localité.

Les autorisations d'accès à ces champs sont les suivants :

- le *Numéro de client* : champ d'affichage;
- le *Nom de client* , le *Prénom* et *le Titre* : autorisation d'insérer (mais pas de modifier)[1];
- la *Rue*, le *Numéro de rue*, le *Code postal*, la *Localité* : autorisation d'insérer et de modifier;

Le séquencement des champs que nous avons choisi respecte les schémas de la conversation internes au message client : le Numéro de Client, le Titre, le Nom du client, le Prénom du client, la Rue, le Numéro, le Code postal et la Localité.

Les remarques, énoncées pour le F/R d'Enregistrement de la Commande, sur les formats des champs et la disposition de l'écran sont valables pour ce F/R également.

La disposition à l'écran est représentée à la figure VI.6.

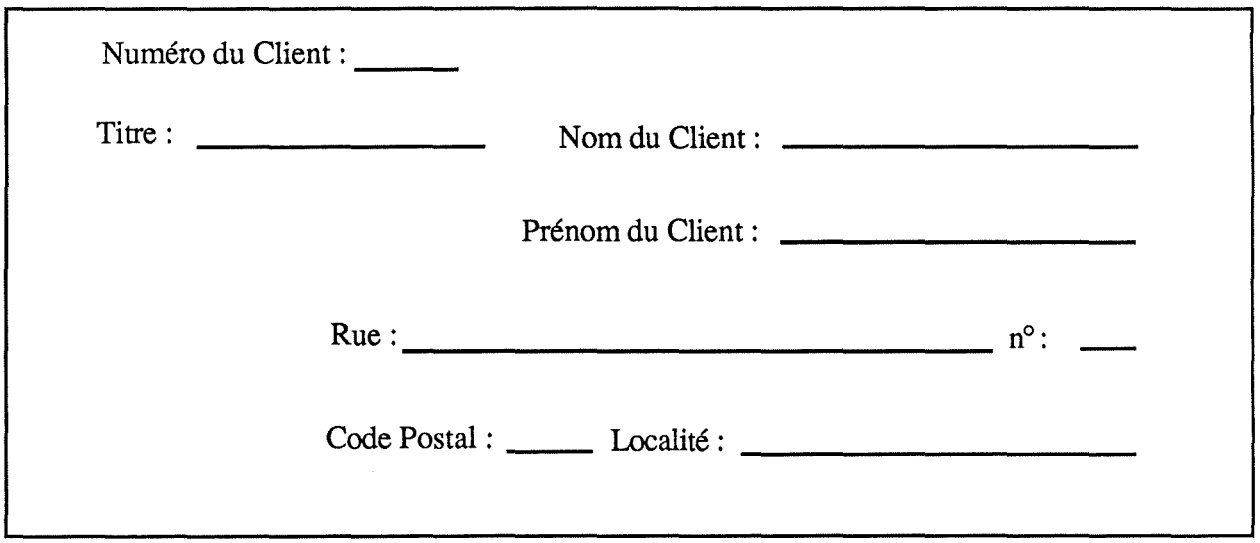


Figure VI.6. : Formulaire de Création/Mise à jour d'un Client

[1] Cela signifie que ces champs ne pourront être modifiés s'il s'agit d'un ancien client. Par contre, ils pourront toujours l'être tant que la saisie d'un nouveau client n'aura pas été validée.

Les messages d'erreur sémantique doivent, comme les autres messages interactifs fonctionnels, être traduits en objets interactifs à ce niveau. Les E4Gs étudiés possèdent tous une gestion des messages d'erreur gérant leur présentation et leur position à l'écran. Les contraintes sur la présentation de ces messages (cfr VI.4.3.1) sont donc prises en charge par le système. Le développeur doit juste définir le contenu (texte explicatif) de ces messages.

VI.5.2.2. Les objets interactifs correspondant aux messages purement interactifs

LES MESSAGES DE CONTROLE

En ce qui concerne les messages de contrôle (cfr VI.4.3.3), nous avons essayé de respecter les contraintes sur la présentation :

- le message Saisie_Ancien_ou_Nouveau ne donne pas lieu à un objet interactif. L'utilisateur signifie qu'il veut saisir un nouveau client en quittant le champ Numéro de client du B/G Commande sans lui attribuer de valeur.
- Le message Saisie_Changement_Adresse correspond à une commande-utilisateur pouvant être déclenchée après l'introduction d'un ancien client valide et non 'douteux'. Nous avons appelé cette commande-utilisateur "Chgt_Adresse".
- les messages Saisie_terminée, Saisie_annulée et Quitter_saisie correspondent à des commandes-utilisateur prédéfinies dans l'environnement : valider (commit), annuler (roll-back[1]) et quitter (quit). Ces commandes-utilisateur sont normalement déclenchables à tout moment. La validation de la commande ne peut cependant l'être que lorsqu'une ligne valide a été saisie. Cela signifie que le développeur doit associer cette contrainte au 'commit'.

LES MESSAGES D'ERREUR SYNTAXIQUE

Tous les messages d'erreur syntaxique (à l'exception du message Num_Produit_Incorrect) sont pris en charge par le système grâce à la définition du type et des contraintes associées à chaque champ : le type permet de définir les caractères permis à la saisie; les contraintes permettent de vérifier la longueur du champ saisi, de définir les valeurs minimum et maximum autorisées pour le champ saisi, de vérifier le caractère obligatoire du

---

[1] Le roll-back correspond à l'annulation d'une transaction. Suite à cette annulation, l'environnement rétablit la base de données dans l'état où elle se trouvait avant la transaction.

champ, ...

En ce qui concerne le message Num_Produit_Incorrect, la vérification du check digit ne peut se faire que via une procédure.

LES MESSAGES D'AIDE

Tous les E4Gs étudiés fournissent une gestion des messages d'aide. Ces derniers peuvent être associés à différents objets de l'E4G : le Formulaire/Rapport, le Bloc/Groupe, le Champ du B/G et le menu. Pour ce faire, une commande-utilisateur, qui peut être activée à tout moment, leur est dédicacée.

VI.5.3. Les traitements : les actions et les expressions

En ce qui concerne les traitements, nous avons choisi de suivre la spécification logique des procédures décrites en VI.4.2 tout en nous pliant aux contraintes imposées par les E4Gs étudiés. De tels environnements contraignent le déroulement de l'application : celui-ci, en effet, n'est pas uniquement explicité par l'enchaînement des procédures, mais également par celui d'autres objets de l'environnement. Ces objets peuvent non seulement constituer une action mais aussi en déclencher eux-mêmes (cfr IV.2.2).

D'autre part, nous avons tenu à profiter des potentialités des environnements qui nous étaient proposées. Cela nous a amené à intégrer à certains B/G, des champs non affichés. Ces champs constituent, en quelque sorte, des variables globales pour les actions dépendant du B/G auquel ils appartiennent. Ils peuvent être intégrés à l'expression déclarative du calcul d'un champ ou encore au sein d'une procédure. Ainsi a-t-on introduit dans les B/G du F/R Commande :

- la Catégorie du client et la Limite d'achat dans le B/G Commande;
- l'Etat d'approvisionnement, le Produit analogue et le Prix unitaire d'achat dans le B/G Ligne.

Les actions déduites des procédures liées à la mémorisation d'une commande sont groupées au sein du Formulaire Enregistrement d'une commande et celles déduites des procédures liées à la mémorisation ou à la modification de l'adresse d'un client au sein du Formulaire Création/Mise à jour d'un Client. Il est à noter cependant que nous ne retrouvons pas de correspondance univoque (1-1) entre les procédures logiques et les actions physiques.

VI.5.3.1. Les traitements liés au Formulaire Enregistrement d'une commande

La **ValidationDunClient** est réalisée au sein d'une procédure (au sens du modèle descriptif global) par une primitive de sélection sur la relation Client de la base de données. La procédure renvoit un booléen pour exprimer la réussite ou l'échec de la sélection du client.

De même pour la **ValidationDunProduit**. Une procédure se charge de la sélection du produit en BD.

La **ValidationDuneProvenance** est effectuée par une Procédure (au sens du modèle descriptif global) qui lit la Catégorie du record courant du B/G Commande. La procédure renvoit le booléen VRAI si cette Catégorie n'est pas douteuse.

La **ValidationDuneLigne** est éclatée en deux traitements distincts.
D'une part la Procédure *ValidationDuneLigne* réalisant les opérations suivantes :

- Si l'Etat d'approvisionnement du record courant du B/G Ligne n'est pas épuisé,
  ALORS
  - Mettre la Quantité commandée à 1 si elle ne contient pas de valeur

  SINON
  - Lecture du champ Etat d'approvisionnement du record de la relation Produit identifié par le Produit analogue
  - Si l'Etat d'approvisionnement du Produit analogue n'est pas épuisé,
    ALORS
    - Mettre la Quantité commandée à 1 si elle ne contient pas de valeur
    - Numéro de produit du B/G Ligne := Produit analogue

    SINON

La procédure renvoit un booléen FAUX si le produit et son produit analogue (s'il existe) sont tous deux épuisés.

D'autre part, le calcul du champ Montant de ligne est exprimé de manière déclarative : l'expression Quantité commandée * Prix Unitaire d'achat du record courant du B/G Ligne est associée au Montant de ligne.

Cet éclatement de la procédure ValidationDuneLigne reflète un choix de conception.

L'expression déclarative du calcul du Montant de ligne est définie une fois pour toutes et est recalculée à chaque changement de valeur de l'un de ses champs : soit lorsque l'utilisateur modifie la quantité commandée ou le numéro de produit, soit lorsque la procédure ValidationDuneLigne remplace le produit épuisé par son produit analogue.

La **ValidationDuneCommande** est également éclatée en différents traitements :

- à la désactivation du B/G Ligne, une procédure de concaténation de lignes (*ConcaténationLignes*) et une procédure (*LimiteDachatOK*) qui renvoit un booléen VRAI si le (Montant total - Rabais) ≤ Limite d'achat du record courant du B/G Commande.
- le calcul des champs Montant total et Rabais de commande, exprimé de manière déclarative : l'expression Σ Montant de ligne est associée au champ Montant total et l'expression Montant total * 20% est associée au Rabais de commande, sous la condition que la catégorie du client courant soit 'collaborateur'. Chacune de ces expressions est recalculée au changement de valeur d'un de ses champs.

De même pour la **MémorisationDuneCommande** :

- le Numéro de commande est nécessaire pour effectuer le lien entre le B/G père Commande et le B/G fils Ligne. Il doit donc posséder une valeur au moment de la réalisation du lien, c'est-à-dire à l'activation du B/G Ligne de commande. Nous avons choisi d'effectuer le calcul du Numéro de Commande par compostage (*CalculNuméroCommande*) à l'activation du record du B/G Commande;
- le calcul de la Date de commande est exprimé de manière déclarative : l'expression Date du jour (variable système disponible par chaque E4G étudié) est associée à la Date de commande du B/G Commande;
- le calcul de la limite d'achat du record courant du B/G Commande (*RéductionLimiteDachat*) est effectué à la désactivation du record B/G Commande;
- l'enregistrement de la commande et de ses lignes est prise en charge de manière automatique par l'environnement, à la désactivation du record du B/G Commande. Le développeur aura néanmoins spécifié auparavant que la transaction portait sur la commande et ses lignes, et interdit le commit après chaque record du B/G Ligne.

VI.5.3.2. Les traitements liés au Formulaire Création/Mise à jour d'un Client

La **ModificationDeLadresseDunClient** est prise en charge de manière automatique par l'environnement grâce à la correspondance des champs Rue, Numéro, Code

postal et Localité du B/G Client avec les champs respectifs de la Relation Client.

La **MémorisationDunClient** est réalisée :

- à l'activation du record du B/G Client, par une procédure de calcul du Numéro de client (*CalculNuméroClient*)
- avant l'insertion du record Client en BD, par une procédure (*InitNouveauClient*) qui affecte les valeurs '2500' et 'nouveau' aux champs Limite d'achat et Catégorie de la relation Client.
- à la désactivation du record, l'enregistrement du nouveau client en BD est pris en charge de manière automatique par l'environnement.

Voilà qui clôture la description de la réalisation physique des traitements de la phase d'Enregistrement d'une Commande-client. Il nous reste à présent à décrire le pilotage de l'application.

VI.5.4. Le pilotage de l'application : les déclenchements d'actions

VI.5.4.1. Les actions associées au Formulaire Enregistrement d'une Commande

L'action initiale de la phase est l'affichage du Formulaire d'Enregistrement de la Commande-Client. Au sein de ce formulaire, la dynamique d'implémentation peut être réalisée de la manière suivante :

NB : les procédures en italique font référence aux procédures décrites en VI.5.3. Elles traduisent les fonctionnalités de la phase. Les autres sont liées à la gestion du dialogue (validation syntaxique d'un champ, calcul d'un champ d'affichage, ...)

A l'**activation du record du B/G Commande :**

ACTION : Procédure *CalculNuméroCommande*

A la **désactivation du champ Numéro de client du B/G Commande :**

ACTION : Procédure ValiderClientEmetteur

- SI la valeur du Numéro de client du B/G Commande n'est pas nulle
  ALORS
  - SI **ACTION** : Procédure *ValidationDunClient*
    ALORS
    - SI **ACTION** : Procédure *ValidationDuneProvenance*
      ALORS
      --- [1]
      SINON
      - Message d'erreur "Provenance_non_valide"
      - Refuser le champ Numéro de client du B/G Commande

    SINON
    - Message d'erreur "Client_non_valide"
    - Refuser le champ Numéro de client du B/G Commande

  SINON
  - **ACTION** : Formulaire Création/Mise à jour d'un Client (mode création)

A l'**activation de la commande-utilisateur "Chgt_adresse" :**

ACTION : Formulaire Création/Mise à jour d'un Client (mode modification)[2]

A la **désactivation du champ Numéro de produit du B/G Ligne :**

ACTION : Procédure ValiderProduitCommandé

- SI **ACTION** : Procédure CheckDigitCorrect
  ALORS
  - SI **ACTION** : Procédure *ValidationDunProduit*
    ALORS

    SINON

---

[1] Dans ce cas, le curseur se positionne sur le champ suivant, respectant ainsi le séquencement défini en VI.5.2.

[2] Lorsque le F/R Création/Mise à jour d'un Client s'ouvre en mode Modification, les champs Nom, Prénom et Titre du B/G ne peuvent être modifiés. Le curseur se positionne directement sur le champ Rue du B/G.

- Message d'erreur "Produit_non_valide"
- Refuser le champ Numéro de Produit du B/G Ligne

SINON

- Message d'erreur "Num_produit_incorrect"
- Refuser le champ Numéro de produit du B/G Ligne

A la **désactivation d'un record du B/G Ligne :**

ACTION : Procédure ValiderLigne

- SI **ACTION** : Procédure *ValidationDuneLigne*
  ALORS

  SINON
  - Message d'erreur "Ligne_non_valide"
  - Refuser le champ Numéro de Produit du B/G Ligne

A la **désactivation du B/G Ligne:**

ACTION : Procédure ValiderCommande

- **ACTION** : Procédure *ConcaténationLignes*
- **ACTION** : Procédure CalculerSoldeAPayer
- SI **ACTION** : Procédure *LimiteDachatOK*
  ALORS

  SINON
  - Message d'erreur ("Commande_non_valide")

A la **désactivation du record du B/G Commande :**

ACTION : Procédure *RéductionLimiteDachat*

VI.5.4.2. Les actions associées au Formulaire Création/Mise à jour d'un Client

A l'activation du record du B/G Client :

SI Mode Création
ALORS
- **ACTION** : Procédure *CalculNuméroClient*

SINON

A l'insertion du record en BD :

ACTION : *InitNouveauClient*

Voici qui clôture la présentation des résultats de la conception physique de la phase Enregistrement d'une Commande-client, et plus globalement de l'expérimentation des E4Gs. Nous avons tenté, lors de la conception physique, de suivre les spécifications logique et conceptuelle de la phase. Toutefois, nous nous en sommes parfois écartés afin d'utiliser davantage les potentialités nouvelles des E4Gs. Nous développons dans le prochain chapitre, le chapitre VII, les considérations que nous avons à exprimer vis-à-vis de la méthode de conception d'application utilisée pour l'implémentation dans un E4G.

Chapitre VII

Evaluation critique de la démarche de conception

VII.1. INTRODUCTION

L'expérimentation réalisée sur la phase Enregistrement d'une commande-client a donné deux résultats distincts : d'une part, le résultat de la conception logique, à savoir un modèle relationnel des données, une spécification des procédures de traitement et une spécification de la tâche; d'autre part, le résultat de la conception physique intuitive en terme de relations de la base de données, de F/R, d'actions et d'enchaînement d'actions.

L'objectif de ce chapitre est d'évaluer la compatibilité des résultats de la conception logique avec ceux de la conception physique, et, plus largement, l'adéquation d'une démarche de conception semblable à celle utilisée lors de l'expérimentation, dans l'optique des E4Gs. Cette évaluation sera suivie, au chapitre VIII, d'une synthèse des concepts méthodologiques qu'il nous paraît judicieux d'utiliser au cours de la conception d'une application dans un E4G.

La pertinence de la démarche utilisée lors de l'expérimentation peut être envisagée sous deux aspects complémentaires. D'une part, de manière autonome au niveau de chacun de ses composants (données, traitements, dialogue); d'autre part, sur le plan des interactions qu'elle établit entre ces différents composants (le pilotage). C'est par cette seconde approche que nous serons amenés à cerner et remettre en cause, dans une certaine mesure, les principes qui encadrent ces démarches de conception classiques (indépendance traitements-dialogue, comportement statique et dynamique, ...).

Nous commençons donc par la critique séparée des différents composants de la démarche de conception.

VII.2. LES DONNÉES

L'utilisation, au niveau logique de la conception, du modèle relationnel nous paraît judicieuse dans l'optique des E4Gs. En effet, le composant 'base de données' de ces environnements recèle les principales caractéristiques d'un SGBD relationnel.

Au niveau conceptuel, le modèle de structuration des informations, par son approche Entité/Association, "offre, quant à lui, de grandes qualités de communication et une bonne capacité de représentation des informations du réel perçu" [Bod 88a]. Il constitue, semble-

t-il, une étape préalable indispensable dans l'élaboration des applications; ceci d'autant plus que le passage du schéma E/A au schéma relationnel est facilité par l'existence de règles de transformation quasi systématiques (cfr V.3.1).

VII.3. LES TRAITEMENTS

Pour concevoir les traitements de l'application, nous avons utilisé lors de notre expérimentation une démarche proposant la conversion systématique de la décomposition conceptuelle de la phase en fonctions à l'architecture logique des traitements. Nous espérions que le passage de l'architecture logique à l'architecture physique soit aussi direct. Tel n'a pas été le cas ! Nous pouvons avancer deux raisons à cet échec :

D'une part, les procédures logiques peuvent être éclatées, à la conception physique, en plusieurs **actions** correspondant à des changements d'états d'objets différents de l'application. C'est le cas dans notre exemple de la procédure MémorisationDuneCommande :

Conception logique :
Procédure *MémorisationDuneCommande*
{obj Calcul par compostage du Numéro de commande
&
...
&
Réduction de la Limite d'achat du Client émetteur
&
...
}

Conception physique :
A l'*activation du record du B/G Commande*
ACTION : Calcul par compostage du Numéro de commande
...
A la *désactivation du record du B/G Commande*
ACTION : Réduction de la Limite d'achat du Client émetteur
...

Il faut souligner le caractère impératif de cette manière de travailler. Le calcul du Numéro de commande devait nécessairement s'opérer avant d'effectuer le lien entre le B/G père Commande et le B/G fils Ligne de commande, et non après. La transformation de la

procédure MémorisationDuneCommande en une seule unité d'exécution était donc impossible.

D'autre part, même si elle constitue le reflet d'une procédure logique, une procédure physique (une action) peut être amputée de l'un de ses traitements :

- *si celui-ci représente un calcul pouvant être exprimé de manière déclarative*; le développeur ne l'inclut pas, alors, dans la procédure physique. Il est à remarquer que ceci constitue un choix de conception mû par le désir d'utiliser les potentialités de l'E4G et de rendre la conception la plus simple possible.
  C'est le cas, dans notre exemple, de la procédure ValidationDuneLigne :

  Conception logique :
  procédure *ValidationDuneLigne*
  {obj Remplacer le produit, s'il est épuisé, par son produit analogue
  Mettre la quantité à 1 si elle est vaut 0
  Calculer le Montant de ligne (Prix unitaire d'achat * Quantité)
  }

  Conception physique :
  ACTION : procédure *ValiderLigne*
  - Remplacer le produit, s'il est épuisé, par son produit analogue
  - Mettre la quantité à 1 si elle vaut 0

  EXPRESSION associée au champ Montant de ligne du B/G Ligne :
  Prix unitaire d'achat * Quantité.
  (Ce calcul est déclaratif en ce sens que l'expression est automatiquement recalculée lors d'un changement de valeur de l'une des variables de l'expression).

- *si celui-ci correspond à l'enregistrement d'un record d'un B/G lié à une relation BD.*
  Dans notre exemple, c'est le cas de l'enregistrement du B/G Client du F/R 'Création/Maj Client' dans la relation Client.
  Cet enregistrement est pris en charge automatiquement par l'environnement, en fin de transaction, via l'action 'commit'.
  Le développeur ne prend donc plus en compte cet enregistrement au sein des actions qu'il décrit. Toutefois, il doit veiller au maintien de la cohérence de la base de données en définissant lui-même, dans les cas opportuns, les bornes de la transaction.
  Dans notre exemple, la transaction portant sur l'enregistrement de la commande requiert l'enregistrement conjoint du record Commande et des records du B/G Ligne de

commande se rapportant à cette commande. Aucun 'commit' ne peut s'effectuer pour les Lignes seules. Le développeur doit donc définir que le B/G Ligne ne constitue pas, seul, une transaction.

Ces constatations diverses montrent que l'architecture physique des traitements de la phase ne peut être déduite de manière systématique (correspondance 1-1) de l'architecture logique. Cela signifie, entre autres, que les caractéristiques des procédures de la conception logique ne sont pas semblables à celles des actions de la conception physique. Ces procédures logiques sont, en fait, issues des deux variantes proposées dans la conception logique. Il nous reste à examiner si les procédures que donne la technique de base ou la première variante sont davantage compatibles avec les propriétés des actions physiques. Rien n'est moins sûr !

La première adaptation de la technique de base consistait à limiter le contenu des arguments/résultats des procédures logiques, dérivées des fonctions de la phase, à l'identifiant des relations véhiculées en paramètres et à supprimer les résultats inutiles (c'est-à-dire identiques à un argument).

Dans les E4Gs étudiés, toute action a accès aux éléments des objets (F/R, B/G) actifs au moment de son déclenchement. L'adaptation effectuée par la première variante s'éloigne de cette caractéristique. La technique de base semble au contraire s'en rapprocher, puisque la procédure logique qu'elle définit reçoit en argument l'entièreté de la relation manipulée (tout comme l'action d'un E4G a accès à tous les champs du record d'un bloc/groupe).

La seconde variante de la technique de base consistait à introduire une mémoire temporaire. Comme son nom l'indique, cette mémoire permet de mémoriser temporairement des données que l'on est presque sûr de devoir mémoriser à terme. Elle rend inutile le passage de certains paramètres entre les procédures logiques.

Les actions d'un E4G ne nécessitent pas, non plus, de paramètres pour les raisons évoquées ci-dessus. La seconde variante se rapproche donc, sur ce point, des caractéristiques de l'E4G. La nuance réside dans le fait que la mémoire temporaire est gérée, dans le cas d'un E4G, automatiquement par l'environnement grâce à la mise en correspondance des B/G avec les relations de la base de données; le concepteur ne doit donc pas la définir.

En conclusion de l'étude des différentes variantes proposées par la démarche de conception, il semble que l'architecture logique dégagée par la seconde variante, amputée de la première adaptation effectuée sur la technique de base, soit plus proche (ou moins éloignée) de l'architecture physique que les autres architectures proposées par la démarche.

Il demeure néanmoins qu'elle s'avère inadaptée à une dérivation systématique des procédures logiques dans la syntaxe des E4Gs, comme nous l'avons montré lors de cette évaluation.

Puisque l'architecture logique est dérivée presque directement de la décomposition de la phase en fonctions, c'est cette décomposition-même qu'il faut évaluer et sans doute remettre en question.

La démarche proposée pour la description conceptuelle des traitements de la phase, de l'événement d'entrée 'Commande à enregistrer' jusqu'à l'événement de sortie 'Commande enregistrée' relève de l'approche orientée-but, qui consiste à obtenir les différents objectifs participant à la phase (Mémorisation d'une commande, Mémorisation d'un client et Màj d'un client) et d'extraire ensuite les sous-objectifs à atteindre pour réaliser ces objectifs. "Cette technique procède d'une logique d'utilisation, chaque objectif correspondant à un résultat que veut atteindre l'utilisateur" [Bar 88].

Il faut remarquer que ces objectifs sont en fait des transactions (d'enregistrement, de mise-à-jour ou de consultation) sur une donnée de la BD. Ils correspondent donc aux F/R de l'implémentation physique de l'E4G. Cette décomposition permet en fait d'associer chaque objectif et sous-objectif à un objet interactif (F/R, B/G, Champ d'un B/G) : Validation d'une ligne est associé au B/G Ligne et Validation d'un produit au champ Num_produit.

Cette technique de décomposition semble donc pertinente dans le chef d'une démarche qui serait "orientée E4G". Néanmoins, les traitements spécifiés pour atteindre chaque objectif sont éclatés, dans un E4G, au sein de l'objet interactif associé. Si l'objet est élémentaire, le traitement se traduit systématiquement en une expression ou une action directement liée au champ du B/G. Par contre, si l'objet est un B/G ou un F/R, le traitement associé est éclaté au sein de cet objet. On pourrait envisager que ce traitement soit décomposé en traitements plus atomiques qui pourraient être associés à des champs du B/G. Dans notre exemple, les objectifs et sous-objectifs constituent uniquement des validations ou des enregistrements. Le calcul d'un champ ne pourrait-il pas constituer également un objectif à lui seul ?

Une autre remarque peut être avancée concernant la dynamique des traitements de l'application, indépendamment de la considération que nous venons de faire. La dynamique ne prend en charge que le déroulement prescrit de la phase, c'est-à-dire son déroulement standard ou recommandé. Dans le cadre d'E4Gs disposant d'une gestion automatisée et plus souple du pilotage de l'application et possèdant les caractéristiques d'un logiciel interactif (déclenchements optionnels, ...), une démarche de conception devrait envisager de décrire le déroulement minimal de la phase, c'est-à-dire les opérations et enchaînements minimaux nécessaires pour que le but de la phase puisse être considéré comme atteint, et d'y intégrer

les déclenchements de traitements optionnels.
C'est le cas, dans la phase Enregistrement d'une commande-client, de la fonction Modification de l'adresse d'un client, ou ce serait le cas de fonctions de consultation des clients ou des produits.

Le problème que nous venons de soulever, qui concerne la dynamique des traitements, semble constituer l'un des problèmes cruciaux rencontré par les démarches de conception classiques face aux E4Gs. Nous appronfondirons cette question lorsque nous aborderons la critique plus spécifique du type de pilotage d'application prôné par ces démarches.

VII.4. LE DIALOGUE

Afin de pouvoir évaluer la démarche utilisée pour la conception du dialogue d'une application interactive, nous rappelons les caractéristiques principales du composant 'interface' des E4Gs avancés :

- un F/R réunit les **transactions** portant sur une donnée élémentaire ou agrégée. Une transaction est une opération de consultation ou de mise à jour d'une donnée de la base de données. Cette donnée peut englober plusieurs relations (une relation principale et des relations secondaires). Les manipulations liées à chaque relation font l'objet d'un B/G particulier. Les B/G sont hiérarchisés pour représenter les dépendances entre relations principale et secondaires.

  L'Enregistrement d'une commande-client constitue l'illustration de ce concept de transaction :
  Le F/R 'Enregistrement d'une commande' représente la transaction de Création d'une commande. La donnée manipulée est la Commande; les relations composant cette donnée sont les relations 'Commande' et 'Ligne'. Au sein du F/R, le B/G père Commande se rapporte à la relation Commande et le B/G fils Ligne à la relation Ligne. Ces deux B/G sont liés par leur Numéro de commande respectif.
  Le F/R 'Création/Mise à jour d'un client' représente les transactions de création d'un client et de Mise à jour d'un client, se rapportant toutes deux à la donnée client; cette donnée est liée à la seule relation Client. Le F/R est donc constitué du seul B/G Client.

- chaque B/G est constitué de champs dont le séquencement doit être défini par le concepteur.

Ces différentes notions étant rappelées, nous pouvons évaluer, à présent, la démarche de conception du dialogue, et plus précisément la spécification de la tâche qu'elle préconise,

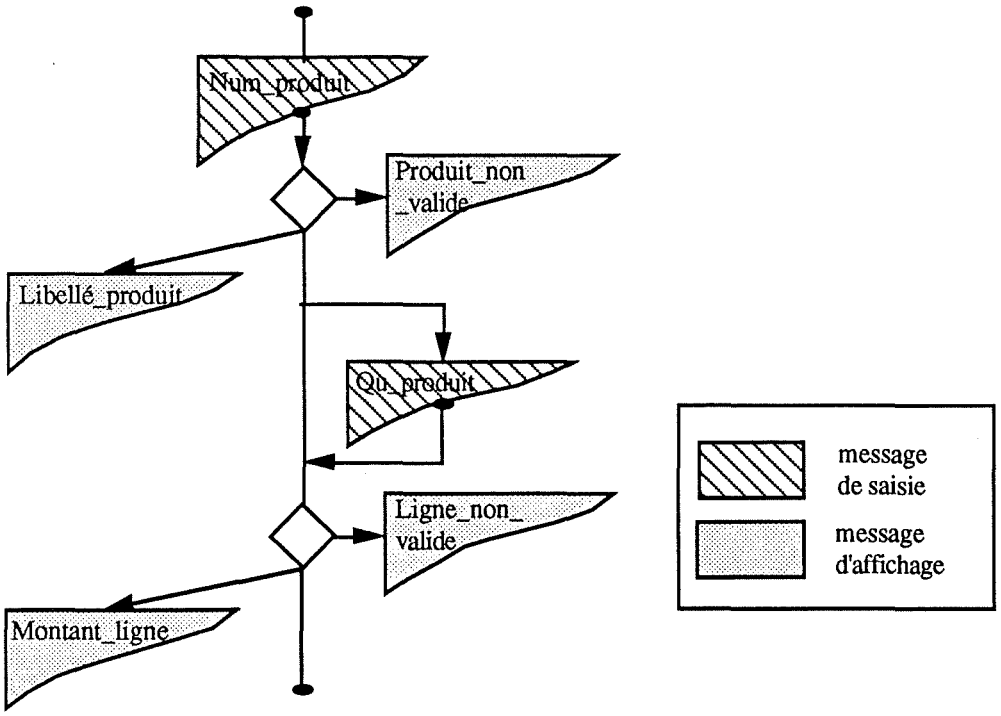
en étudiant la "pertinence" des éléments dont elle se compose.

VII.4.1. La spécification des messages interactifs

VII.4.1.1. messages interactifs fonctionnels

Une spécification de messages interactifs fonctionnels semble nécessaire pour dériver la conception d'objets interactifs, qu'ils soient F/R, B/G ou champs d'un B/G ou encore messages d'erreur sémantique. Néanmoins, cette dérivation n'est pas systématique : si le message interactif fonctionnel Commande correspond bien à un F/R, si le message interactif fonctionnel Ligne correspond à un B/G, il n'en est pas de même pour le message interactif fonctionnel Client. Celui-ci est représenté au sein du F/R 'Enregistrement d'une Commande' lorsqu'aucune transaction sur le client n'est effectuée et au sein du F/R 'Création/Mise à jour d'un client' dans le cas contraire. Cela s'explique par le fait que le concept de transaction ne constitue pas un critère de décomposition en messages interactifs; le seul critère utilisé à ce niveau est l'unité de dialogue. Cela signifie que les critères de décomposition retenus au niveau logique (portant sur les messages interactifs) ne correspondent pas totalement à ceux du niveau physique (portant sur les objets interactifs). Cela rend par conséquent impossible une conversion univoque des messages interactifs en objets interactifs.

Ces considérations concernent le critère de décomposition attaché aux messages interactifs fonctionnels; en ce qui concerne la spécification-même de ces messages, il semble que les éléments qui la composent soient pertinents, notamment le schéma de la conversation interne. Ce schéma définit l'enchaînement des différents attributs du message. Il peut être traduit systématiquement en un séquencement des éléments de l'objet interactif correspondant à saisir. Prenons l'exemple du message interactif fonctionnel Ligne et de l'objet interactif correspondant, le B/G Ligne :
Dans le message interactif fonctionnel Ligne, le schéma de la conversation interne est le suivant :



Au sein du B/G Ligne, le séquencement choisi est : le Numéro de produit (Produit commandé) suivi de la Quantité commandée. Il est déduit de l'enchaînement des messages de saisie du message Ligne.

VII.4.1.2. les messages purement interactifs

MESSAGES DE CONTROLE

La spécification des messages de contrôle est indispensable même si ceux-ci correspondent, au niveau physique, à une commande-utilisateur gérée par l'environnement (commit, rollback). En effet, même dans le cas d'un 'commit', le déclenchement du message est conditionné par l'état du système. L'état dans lequel le système doit se trouver pour que le message puisse être activé doit être décrit dans la spécification de ce message.

Il est à remarquer que ce sont ces messages de contrôle qui nous permettent de réaliser le lien entre la transaction d'Enregistrement d'une Commande et les transactions effectuées sur le client passant la commande : la transaction d'Enregistrement d'un client sera déclenchée suite à l'un des choix du message 'Saisie_Ancien_ou_Nouveau' et la transaction de Mise à jour du client consécutivement à l'un des choix du message 'Saisie_Changement_Adresse'.

MESSAGES D'ERREUR SYNTAXIQUE

La spécification des messages d'erreur syntaxique n'est pas utile dans la mesure où ceux-ci sont pris en charge par l'E4G. L'environnement se base, pour ce faire, sur les types définis pour les champs de l'interface. Ces champs étant déduits des messages interactifs fonctionnels, c'est au sein de ces messages que doit apparaître la spécification complète des types des composants du message, en terme de format, de valeur minimum et maximum, d'appartenance à une liste de valeurs, d'obligation ou d'interdiction de saisie, etc. puisque ceux-ci sont pris en charge par le système.

Une nuance cependant : les messages découlant d'un contrôle complexe de la valeur d'un champ doivent, eux, être spécifiés. Dans notre exemple, c'est le cas du message Num_produit_incorrect, qui est consécutif au contrôle du 'check digit' du Numéro de produit.

MESSAGES D'AIDE

Dans un E4G, chaque message est associé à un objet de l'application (un F/R, un B/G, un champ d'un B/G ou un menu).

La spécification logique d'un message d'aide doit donc contenir, outre la définition du message et de son contenu, le nom de l'objet auquel il est lié.

Voici qui clôture l'évaluation de la spécification des messages interactifs. Le dernier élément constitutif de la conception logique du dialogue est le schéma de la dynamique d'implémentation de la tâche. Il supporte la validation de la tâche par rapport à la dynamique des traitements de la phase. Il illustre les interactions entre le dialogue et les traitements. Il est l'équivalent logique du pilotage de l'application. Sa conception doit par conséquent être mise en correspondance avec la conception physique du pilotage, décrite dans notre expérimentation, en VI.5.4.

VII.5. LE PILOTAGE

Au travers de la dynamique d'implémentation de la phase, c'est aussi la dynamique des traitements qu'il faut évaluer. Rappelons qu'en VII.3, nous avons critiqué la pertinence de la décomposition de la phase en fonctions; par là, nous avons mis en cause la dynamique de ces fonctions. Notre objectif, à présent, est d'envisager cette dynamique sous un angle plus général.

Cette dynamique illustre parfaitement la vision qu'ont les méthodes de conception classiques, des interactions entre les différents composants d'un environnement.

VII.5.1. L'utilisation de la BD par les traitements / la non-utilisation directe de la BD par le dialogue

La démarche de conception classique stipule que, "de manière générale, une application interactive est composée de deux parties principales : le dialogue et les fonctions de l'application. La partie dialogue se charge de communiquer à l'utilisateur toutes les données qui lui sont nécessaires et demande à ce dernier certaines informations utiles à l'application. Quant aux fonctions de l'application, elles traitent les données fournies par l'opérateur, accèdent aux informations stockées dans une BD, produisent des résultats, ... (...) Le critère retenu pour déterminer la limite entre les fonctions et le dialogue de l'application interactive est le suivant : tout traitement ne nécessitant pas d'accès à la BD peut très bien être effectué par la partie dialogue de l'application" [Bod 88b]. Le corollaire de ce critère est "Le dialogue ne prend par en charge les traitements sur les données de la base de données".

La prise en compte de cette séparation modulaire permet notamment de spécifier plusieurs interfaces tout en gardant les mêmes fonctionnalités de l'application.

A première vue, les E4Gs n'offrent pas cette indépendance : l'interface (via les B/G) est mis en correspondance avec les relations de la BD. Néanmoins, cette mise en correspondance ne constitue qu'une expression non-procédurale d'un ensemble d'accès à la BD et ne compromet en rien l'indépendance Interface/Fonctionnalités. Cependant, le concepteur de l'interface devrait normalement connaître l'organisation de la base de données afin de pouvoir lier les B/G aux relations de celle-ci. L'intégration inhérente aux E4Gs, et dans le cas présent l'intégration des composants BD et Interface, lui offre un support logiciel (le dictionnaire des objets) permettant d'effectuer cette correspondance.

On pourrait également se demander si l'expression déclarative du calcul d'un champ d'un B/G ne contredit pas le critère de distinction défini par la démarche; en effet, ce calcul est spécifié dans l'interface alors qu'il porte parfois sur un champ non affiché. Il s'avère que le critère est toujours valable à ce niveau puisque le champ non affiché fait totalement partie de l'interface. Celui-ci ne réalise donc pas d'accès à la BD.

Cependant, le corollaire du critère n'est plus respecté. En effet, le calcul pris en charge par l'interface représente un traitement sur une donnée de la base de données (ex : Montant de ligne). L'apport des E4Gs dans ce domaine consiste en la réalisation automatique du lien entre le champ calculé de l'interface et le champ BD correspondant.

Quant aux traitements réalisant les autres fonctionnalités de l'application, ils sont eux aussi liés, dans un E4G, aux objets de l'interface (via les changements d'état de ces objets).

On peut donc considérer que, dans un E4G, les F/R prennent en charge une partie des fonctionnalités de l'application. Il semble donc que cette potentialité offerte nécessite un changement d'optique dans la démarche de conception des applications. Ce changement se traduirait au niveau logique par une spécification des objets de l'interface intégrant la spécification des traitements leur étant associés.

VII.5.2. Indépendance Statique/Dynamique

La démarche classique conduit à une séparation modulaire entre la dynamique et la statique (tant de l'interface que des traitements) de l'application.
La dynamique a pour rôle de gérer le déroulement de l'application. Elle contient l'enchaînement des messages interactifs et des procédures. Les Messages interactifs sont chargés de la saisie des informations nécessaires à l'exécution des procédures et l'affichage des résultats fournis par ces dernières. Les procédures implémentent chacune une des fonctions décrites lors de l'analyse fonctionnelle de l'application interactive.

Dans les E4Gs, la dynamique n'apparaît pas en tant que telle. Elle n'est plus gérée explicitement au sein d'un module autonome. Elle ne correspond plus à un enchaînement linéaire de messages et de fonctions mais se retrouve dispersée au sein des différents objets (F/R, B/G, ...) de l'application. Il n'est donc plus possible de l'implémenter indépendamment de ces objets, comme le préconise la démarche classique de conception.

VII.6. CONCLUSION

Cette évaluation critique nous a permis de mettre en évidence les inadéquations entre les démarches classiques de conception et les E4Gs.

Les démarches de conception classiques n'intègrent pas la conception des différents objets de l'application. Elles sont, au contraire, motivées par un souci de modularité (différenciation des Données, Dialogue et Traitements au travers de modules) dans la conception. Or, les E4Gs tendent à regrouper la description des objets de l'application autour de concepts centraux (l'Objet et l'Action, la Donnée agrégée et la Transaction) qui intègrent fonctionnalités et dialogue, dialogue et données.

Dans le même ordre d'idée, de telles démarches ne tiennent pas compte de l'existence d'une fonction de pilotage au sein des E4Gs gérant automatiquement les interactions entre les objets de l'application (Relation, F/R, B/G, Champ, Procédure, Menu, ...).

En résumé, les démarches de conception classiques sont, semble-t-il, inadaptées à la prise en charge des concepts de la 4ème génération.

Le chapitre suivant tentera de proposer une démarche de conception plus adaptée à cette génération en tenant compte des critiques évoquées dans ce chapitre.

Chapitre VIII

Suggestions pour une démarche de conception "orientée E4Gs"

VIII.1. INTRODUCTION

Nous proposons, dans ce chapitre, une démarche de conception d'application qui nous semble compatible avec une implémentation dans un E4G. Nous nous basons, pour cela, à la fois sur la modélisation des E4Gs effectuée aux chapitres III et IV et sur la critique d'une démarche de conception classique, exprimée au chapitre précédent.

Dans ce chapitre déjà, nous avions laissé entrevoir les caractéristiques que pourrait avoir une démarche de conception adaptée aux E4Gs. Elle devrait en tout cas prendre en compte, au niveau conceptuel, les notions de donnée agrégée, de transaction sur une donnée agrégée, et au niveau logique les notions d'objets (F/R, B/G, Champ d'un B/G, ...), d'état d'objet, de calcul déclaratif et d'actions associées à des changements d'état d'objets.

Comme nous l'avons déjà signalé, les concepts communs aux E4Gs avancés relèvent d'un niveau d'abstraction relativement élevé. La démarche de conception "orientée E4Gs" que nous proposons comprend une étape de spécification conceptuelle et une étape de conception logique aboutissant à la spécification des objets constituants du modèle descriptif global défini en IV.3. La conception physique consiste, quant à elle, à transcrire la description de ces objets dans la syntaxe concrète de l'environnement considéré.

VIII.2. LE NIVEAU CONCEPTUEL

VIII.2.1. Les données

La modélisation conceptuelle des données selon l'approche E/A paraît adéquate pour les raisons explicitées en VII.2. Les principes de cette modélisation ont été énoncés en V.2.1 et implémentés sur la phase d'Enregistrement d'une commande-client en VI.3.1. Nous n'avons pas envisagé, à ce niveau, de modélisation de la notion de donnée agrégée (ex : la Commande et ses Lignes). Nous avons préféré prendre en compte ce concept au sein de la spécification des traitements de l'application interactive.

VIII.2.2. Les traitements

La spécification de la **phase** reste le repère central de la structuration des fonctionnalités de l'application, au niveau conceptuel. Elle comprend :

- la description des **objectifs** de la phase,
- les **performances** souhaitées,
- les **informations** en entrée et en sortie,
- les **actions** qu'elle effectue sur la mémoire du SI.

Les objectifs d'une phase d'une application interactive coïncident généralement avec des transactions de Création, Mise à jour ou Consultation d'une donnée (élémentaire ou agrégée) de la base de données. La spécification de ces **transactions** se compose des éléments suivants :

- un **nom**;
- une description de l'**objectif** de la transaction; cet objectif peut être de type 'consultation', 'création' ou 'mise à jour' d'une donnée en BD;
- une description de la **donnée agrégée** qu'elle manipule et des contraintes éventuelles à respecter vis-à-vis de cette donnée, avant son enregistrement en BD; une donnée agrégée est constituée d'une entité et des associations auxquelles cette entité est liée par une connectivité de type 1-*.
- une description des **règles de traitement** à réaliser pour respecter certaines contraintes d'intégrité associées à la donnée manipulée par la transaction, eu égard aux performances souhaitées pour la phase.
- une description des **actions** effectuées en BD sur d'autres données que la donnée agrégée.

Dans l'exemple de l'Enregistrement d'une commande-client, les transactions sont : la Création d'une commande, la Création d'un client et la Mise à jour d'un client. La spécification de la transaction 'Création d'une commande' se présente de la manière suivante :

La transaction *'Création d'une Commande'*
a pour objectif l'enregistrement d'une commande valide

ajoute une entité ***Commande***

caractérisée par :
un ***Numéro de commande***

[1] *[qui l'identifie et est calculé par compostage]*
une ***Date de commande***
[2] *[égale à la date du jour]*
un ***Montant total de commande***
[3] *[Somme des Montant(s) de ligne pour toutes les Ligne(s) de la Commande]* et
un ***Rabais de commande*** *(facultatif)*
réduction de 20% du Montant total de commande
[4] *[accordé à un Client (Emetteur) dont la catégorie est 'collaborateur'];*

une et une seule ***Provenance***
émise par un client
[5] *[qui existe en mémoire]* et
[6] *[dont la catégorie n'est pas 'douteux']*

une ou plusieurs ***Ligne(s)***, chacune
[7] *concernant un produit différent*
[8] [*qui existe en mémoire*] et
[9] [*dont l'état d'approvisionnement n'est pas 'épuisé'*] et
caractérisée par :
une ***Quantité commandée***
un ***Montant de ligne***
[10] *[produit de la Quantité commandée par le Prix unitaire d'achat du Produit (Commandé)]*;

[11] *[ne peut être ajoutée que si le Montant total de commande est inférieur à la Limite d'achat du Client (Emetteur)]*.

modifie la ***Limite d'achat*** *du client (émetteur)*
[12] *[en lui soustrayant le Montant total de commande, réduit du rabais éventuel de la commande valide]*

Les performances souhaitées pour la phase (refuser le moins de bons de commande possible) suggèrent d'associer les règles de traitement suivantes à la transaction de Création d'une commande :

Afin de respecter la contrainte [7]
si plusieurs ligne(s) valide(s) portent sur un même produit,

alors il faut les remplacer par une seule Ligne valide dont :
- *la Quantité commandée est la somme des Quantité(s) commandée(s) initiales*
- *le Montant de ligne est la somme des Montant(s) de ligne initiaux*

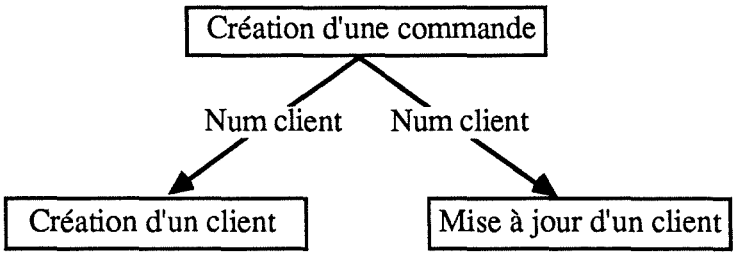
Afin de respecter la contrainte [9]

si l'Etat d'approvisionnement du Produit est 'épuisé' et
si celui-ci peut être substitué par un Produit (Analogue),
alors il faut remplacer le Produit valide par son Produit (Analogue)

Afin de respecter la contrainte [10]

si la Quantité commandée n'est pas explicitement fournie,
alors elle est d'UNE Unité d'achat

Les transactions constituant les objectifs d'une phase s'unissent pour atteindre son objectif global. Nous pouvons les représenter sous forme d'une arborescence. Les noeuds de l'arbre constituent les transactions et les arcs la relation 'déclenche'. Le déclenchement est toujours associé à l'identifiant de la donnée manipulée par la transaction fille, identifiant qui est repris dans la transaction mère[1]. Dans l'exemple de l'enregistrement d'une commande-client, les trois transactions se hiérarchisent de la manière suivante :



Il s'agit de la seule dynamique de traitement établie au niveau conceptuel.

Nous synthétisons la démarche à suivre pour spécifier les traitements au niveau conceptuel, à la figure VIII.1 :

1 Il est à remarquer que cette arborescence est comparable à celle des blocs/groupes au sein d'un F/R.

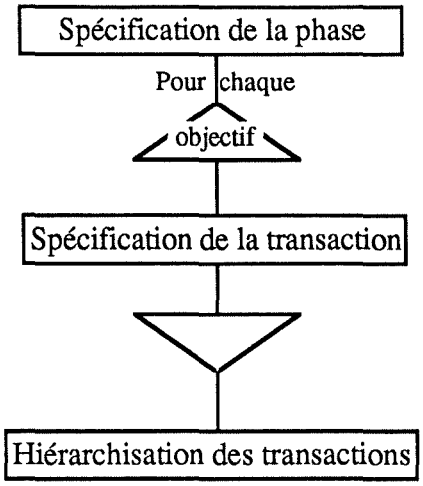


Figure VIII.1. : Démarche de spécification conceptuelle des traitements

VIII.3. LE NIVEAU LOGIQUE

La démarche de conception logique se base sur le modèle descriptif global des E4Gs. Elle tente de décrire l'ensemble des objets constituant l'application, objets de la base de données (relations, champs d'une relation), objets du dialogue (F/R, B/G, Champs d'un B/G), ainsi que les opérations associées à ces objets (expression déclarative de calcul d'un champ, procédure de traitement) et les déclenchements d'actions (F/R, Menu ou Procédure) associés aux changements d'état des objets.

VIII.3.1. Les données

La démarche proposée en V.3.1 demeure valable pour les E4Gs, étant donné les propriétés relationnelles de leur composant 'base de données'. Le schéma E/A est transformé en un schéma MAG conforme au relationnel, ensuite en un schéma relationnel. Le schéma relationnel obtenu pour l'Enregistrement d'une commande-client est décrit en VI.4.1.

VIII.3.2. Les objets du dialogue

La transaction conceptuelle effectue des opérations sur une donnée agrégée. Elle est représentée, au niveau conceptuel, par l'objet **Formulaire/Rapport**. Les autorisations d'accès associées à cet objet sont déduites du type de transaction dont il s'agissait. Dans notre exemple, la transaction Création d'une commande s'abrite au sein du F/R

Enregistrement d'une commande, tandis que les transactions de Création et Mise à jour d'un client s'intègrent au sein du F/R Création/Mise à jour d'un client.

Les objets **Blocs/Groupes** constituant le F/R se déduisent de la donnée à laquelle est associée le F/R. Si cette donnée est élémentaire (par exemple, le client), le F/R ne comprendra qu'un seul B/G (le B/G Client pour le F/R Création/Mise à jour d'un client). Si la donnée est agrégée, elle regroupe plusieurs relations du schéma relationnel. Chacune de ces relations est mise en correspondance avec un B/G au sein du F/R; les B/G sont hiérarchisés sur base des contraintes référentielles existant entre les relations. Dans le F/R Enregistrement d'une commande, le B/G fils Ligne de commande, lié à la relation Ligne, est associé à son B/G père Commande, lié à la relation Commande par l'intermédiaire du numéro de commande.

Les objets **Champs des B/G** sont déduits des attributs manipulés par la transaction (qui sont devenus des champs d'une relation). Chaque *champ d'une relation manipulé* est mis en correspondance avec un champ du B/G lié à cette relation. Dans notre exemple, un champ numéro de commande est inséré au B/G Commande et mis en correspondance avec le champ de même nom de la relation Commande; un champ Quantité est ajouté au B/G Ligne de commande et mis en correspondance avec le champ Quantité de la relation Ligne. En fonction de l'interface envisagé, d'autres champs peuvent également être introduits aux B/G, alors qu'ils n'appartiennent pas à la relation principale du B/G. Ce sont des champs d'affichage. Il peuvent correspondre à des *champs d'une autre relation* de la base de données (associée indirectement à la transaction); c'est le cas du Nom de client (relation Client) ou du Libellé de produit (de la relation Produit). Ils peuvent correspondre à des variables et abriter des calculs sur les données du F/R; c'est le cas du Solde_à_payer (Montant total - Rabais).

Les Blocs/Groupes d'un F/R requièrent une attention toute particulière et une spécification détaillée. En effet, ce sont eux qui abritent les opérations effectuées sur chaque record d'une relation de la base de données. Nous avons choisi de les spécifier de la manière suivante :

- un **nom**,
- le nom de la **relation** principale à laquelle le Bloc/Groupe correspond,
- les **autorisations d'accès** liées au Bloc/Groupe (autorisation d'insérer, de modifier un record ...),
- la description du **contenu** du Bloc/Groupe (des champs et/ou d'autres blocs/groupes),
- l'expression des **contraintes syntaxiques** associées aux champs du Bloc/Groupe (format, valeurs minimum et maximum, appartenance à une liste de valeurs, affiché (o/n), obligatoire (o/n), ...) et les **messages d'erreur syntaxique** associés à ces contraintes (dans le cas d'une contrainte syntaxique complexe),
- les **messages d'erreur sémantique** associés au bloc/groupe,

- les **messages d'aide** associés au bloc/groupe,
- des **contraintes sur la présentation** des éléments du bloc/groupe,
- un **schéma de la conversation** interne au bloc/groupe, décrivant l'enchaînement des actions permises sur les champs du bloc/groupe (NB : si les autorisations d'accès au bloc/groupe sont inexistantes, il n'y a pas lieu de définir un schéma de la conversation interne : l'utilisateur ne peut effectuer aucune opération sur les champs du bloc/groupe).

Cette spécification de l'objet Bloc/Groupe est laissée à l'initiative du développeur (notamment en ce qui concerne le schéma de la conversation et les contraintes sur la présentation). Elle doit toutefois subir une validation sur base de la description de la transaction associée au B/G. Les B/G doivent être spécifiés de manière à permettre à l'utilisateur de saisir toutes les données nécessaires à la réalisation de la transaction et doivent afficher toutes les informations utiles à l'utilisateur, dans l'optique de l'interface envisagé.

Les messages interactifs d'erreur syntaxique, d'erreur sémantique et d'aide doivent être spécifiés de la manière suivante :

- un **nom**,
- une **définition**,
- un **type** (message interactif d'aide, d'erreur syntaxique ou d'erreur sémantique),
- un **contenu**, décrivant le texte du message,
- un **objet** (champ ou bloc/groupe) auquel le message est associé.

Voici un exemple de spécification d'un Bloc/Groupe, le B/G Ligne :

Ligne

Nom : ligne;
Relation : Ligne;
Autorisations d'accès : autorisation d'insérer et de modifier;
Contenu :

- Num_produit : entier;
- libellé_produit : char[50];
- Qu_produit : entier;
- montant_de_ligne : réel;

Contraintes syntaxiques :

- le numéro de produit doit être un entier positif dont le dernier chiffre doit être égal au reste de la division des 7 premiers chiffres par 7 (message d'erreur syntaxique Num_produit_incorrect);

- la quantité doit être un entier strictement positif;

Messages d'erreur sémantique :

- Produit_non_valide;
- Ligne_non_valide;

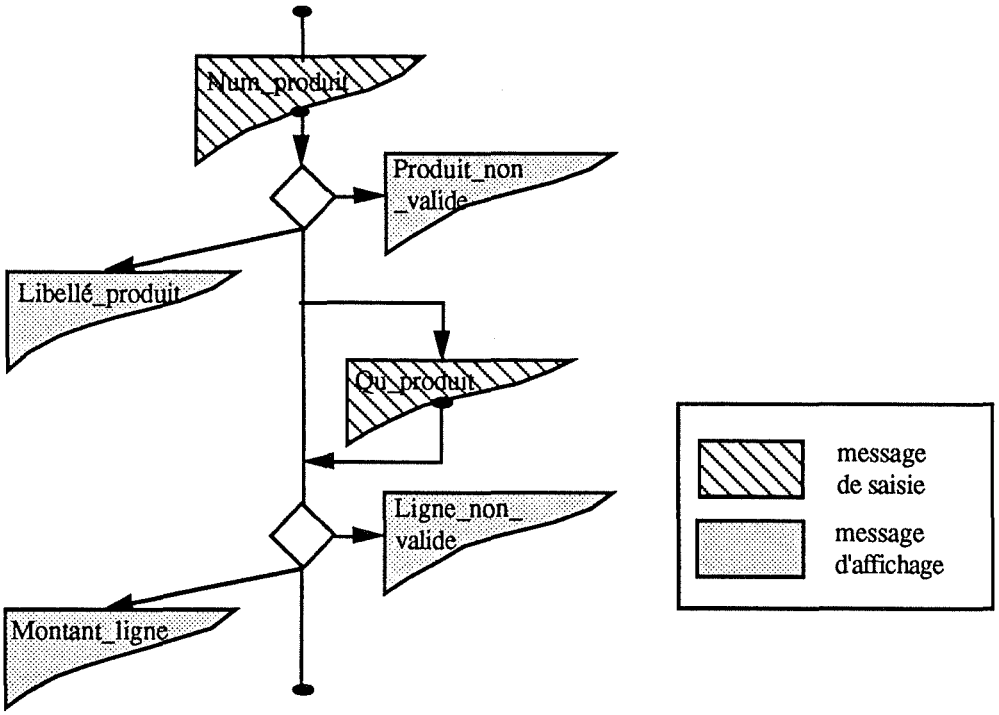
Messages d'aide : - non définis -

Contraintes sur la présentation : libellé_produit devra être affiché bien en correspondance avec le numéro de produit num_produit pour éviter à l'utilisateur un effort de visualisation;

Opérations de manipulation :

- l'utilisateur peut créer le message Qu_produit;
- l'utilisateur peut clôturer le message;

Schéma de la conversation :



Produit_non_valide

Nom : produit_non_valide;

Définition : le message produit_non_valide signale à l'opérateur qu'il a introduit un numéro de produit inexistant;

Type : message d'erreur sémantique;

Contenu :

- un texte signalant que le numéro de produit introduit n'existe pas : "Le numéro que

vous avez introduit ne correspond à aucun produit existant. Veuillez essayer une nouvelle valeur";

Objet : Num_produit.

Ligne_non_valide

Nom : ligne_non_valide;

Définition : le message ligne_non_valide signale à l'opérateur que le produit commandé est épuisé et qu'il ne peut être remplacé par un produit substitut;

Type : message d'erreur sémantique;

Contenu :

- un texte signalant que le produit est épuisé : "Le produit que vous avez introduit ne peut être commandé; il est épuisé et ne peut être substitué";

Objet : Num_produit.

Num_produit_incorrect

Nom : num_produit_incorrect;

Définition : ce message a pour but de faire connaître à l'opérateur qu'il a introduit un numéro de produit incorrect car il n'est pas entier ou le reste de la division des 7 premiers par 7 n'est pas égale au huitième;

Type : message d'erreur syntaxique;

Contenu : un texte signalant que le numéro du produit introduit est incorrect : "Le numéro de produit que vous avez introduit est incorrect. Veuillez taper un nouveau numéro";

Objet : num_produit.

Nous synthétisons la démarche à suivre pour spécifier les F/R, à la figure VIII.2 :

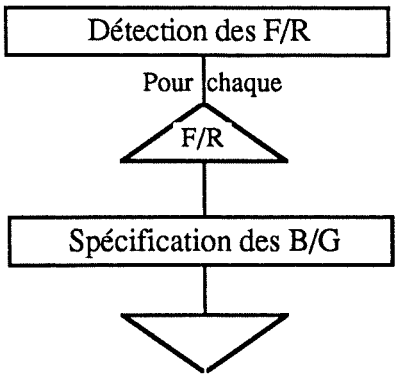


Figure VIII.2. : Démarche de spécification des F/R

La spécification des B/G est synthétisée à la figure VIII.3 :

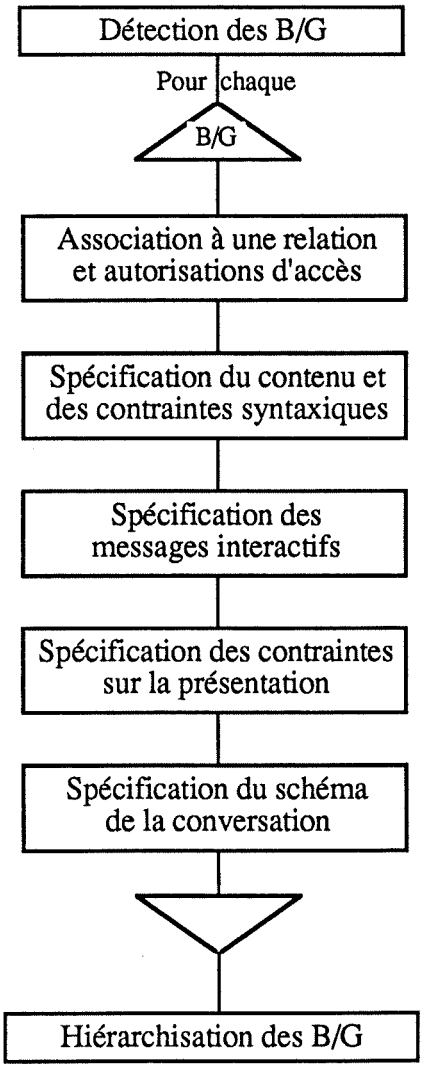


Figure VIII.3. : Démarche de spécification des B/G

VIII.3.3. Les traitements

Au niveau du modèle descriptif global des E4Gs, un traitement peut être soit une **expression déclarative** du calcul d'un champ d'un B/G, soit une **action** déclenchée lors d'un changement d'état d'un objet de l'application. La liste de ces changements d'état est reprise en IV.2.1.
La spécification d'une *expression déclarative de calcul d'un champ* se réalise dans la définition du champ du B/G auquel elle est associée. Elle doit contenir :
- une **description** déclarative de l'expression.

La spécification d'une *action* doit, elle, être constituée :
- du **nom** de l'action déclenchée;
- de son **type** (procédure, menu, F/R);
- de sa **description** (pour une procédure ou un menu),
- de l'**objet** qui abrite son déclenchement (F/R, B/G, Champ d'un B/G, Commande, Menu),
- du **changement d'état de l'objet** auquel le déclenchement est associé (ex: avant insertion, après modification d'un record en BD, ...);
- d'une **condition de déclenchement** éventuellement;
- des **paramètres actuels** *de procédure* si l'action déclenchée est une procédure avec paramètres;
- les **autorisations d'accès au F/R** (la première valeur désignant le mode d'utilisation initial du F/R) si l'action déclenchée est un F/R.

Il reste à déterminer comment déduire les expressions déclaratives ou les actions à partir de la spécification d'une phase ou d'une transaction.

Le premier traitement issu de la description conceptuelle des transactions d'une phase est celui de déclenchement d'un F/R. Dans l'exemple de l'enregistrement d'une commande-client, le déclenchement du F/R Enregistrement d'une commande est sans doute consécutif à un menu initial et le déclenchement du F/R Création/Mise à jour d'un client est consécutif à un choix de l'utilisateur (Saisir un nouveau client ou modifier l'adresse d'un client). Le développeur doit permettre à l'utilisateur d'effectuer ce choix (cela correspond aux messages de contrôle envisagé dans la démarche classique). Au niveau des E4Gs, le déclenchement d'un F/R peut être lié à un changement d'état d'un objet de l'application. Il semble donc inutile de spécifier un message de contrôle. Il est plus judicieux que la première étape de la démarche de spécification logique des traitements consiste à localiser directement l'objet dont le changement d'état déclenche le F/R (éventuellement un déclenchement optionnel). Dans l'exemple de l'enregistrement d'une commande-client, le déclenchement du F/R

Création/Mise à jour d'un client doit être associé à la désactivation du Numéro de client du B/G Commande du F/R Enregistrement d'une commande, si celui-ci n'a pas reçu de valeur (il est alors activé en mode 'création'); il peut être associé à une commande-utilisateur ou un menu déclenchable si le numéro de client est valide (il est alors activé en mode 'modification').

Les autres traitements (expression ou action) sont déduits des contraintes associées, au niveau conceptuel, aux données de la transaction (numérotées entre [ ]). Il n'existe pas de règles systématiques de transformation des contraintes d'intégrité ou des règles de traitement, associées à la transaction, en actions ou expressions; ces contraintes ou règles de traitement peuvent être très variée. Cependant, nous pouvons énoncer quelques règles à titre indicatif :

- une contrainte de validation portant sur un attribut élémentaire de la donnée agrégée sera associée à la désactivation du champ correspondant au sein du F/R;
- une contrainte de validation portant sur plusieurs attributs d'une même entité (au sein de la transaction) sera associée à la désactivation du record du B/G correspondant à la relation issue de l'entité; .
- une contrainte de validation portant sur les valeurs d'attributs d'entités différentes au sein de la donnée agrégée sera associée à la désactivation du record du B/G le plus élevé dans la hiérarchie des B/G concernés;
- le calcul d'un attribut élémentaire sera généralement exprimé de manière déclarative dans la description-même du champ correspondant du F/R. Le calcul du Montant de ligne sera exprimé de manière déclarative et associé au champ Montant de ligne du F/R Enregistrement d'une commande; cette règle n'est pas contraignante puisque le calcul du numéro de commande, identifiant de la commande, peut s'effectuer à l'activation du record du B/G Commande.

Nous synthétisons la démarche à suivre pour spécifier les traitements, à la figure VIII.4 :

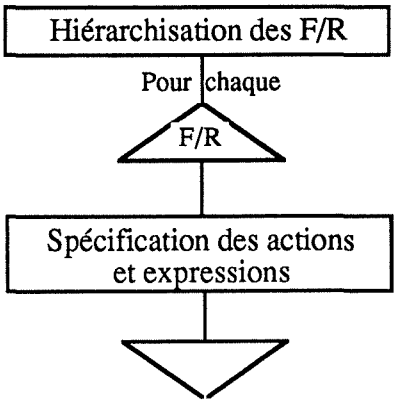


Figure VIII.4. : Démarche de spécification logique des traitements

VIII.4. LE NIVEAU PHYSIQUE

La conception physique consiste en la traduction des spécifications des objets logiques composant l'application, dans la syntaxe de l'environnement considéré. Cette traduction pourrait s'effectuer de manière plus ou moins systématique en utilisant des règles de transformation propres à chaque E4G particulier. Notre objectif, dans ce chapitre, était de proposer un cadre méthodologique global adapté au développement d'applications à l'aide de ces environnements. Dégager une démarche de conception physique particulière à chaque E4G n'entre donc pas dans nos préoccupations.

Conclusion

En débutant ce mémoire, nous avons signalé les problèmes inhérents aux Environnements de Troisième Génération : le manque de productivité des développeurs, l'isolement des utilisateurs, etc. La Quatrième Génération est présentée comme une solution technique à ces différents problèmes. D'une part, elle permet d'intégrer l'utilisateur final au processus de conception des applications et, d'autre part, elle fournit au développeur la possibilité d'améliorer sa productivité ainsi que la qualité du développement de ses applications.

Notre premier objectif était de déterminer en quoi, par leur structure, ces nouveaux environnements différaient des environnements de la génération précédente. Ils n'en diffèrent pas par les fonctions qu'ils offrent : ils possèdent principalement, tout comme les E3Gs, une fonction de gestion et d'accès aux données, une fonction de saisie et présentation des données et une fonction de traitement de ces données. Leur caractéristique propre réside, cependant, dans l'intégration de ces fonctions. Cette intégration est réalisée grâce à la présence d'un dictionnaire unique centralisant la description des objets d'une application. Ce dictionnaire gère non seulement la description des objets mais également les relations entre ces objets : les objets de l'interface y sont mis en correspondance directe avec les objets de la base de données; ils représentent le lieu de réalisation des différentes transactions sur ces données.

Les E4Gs dits traditionnels maintiennent la dynamique de l'application au sein du composant de traitement des données, c'est-à-dire au sein des procédures ou primitives. D'autres E4Gs, que nous avons nommés les E4Gs avancés, envisagent une approche différente de cette dynamique. Cette approche est caractérisée par deux notions nouvelles : l'action et le changement d'état d'un objet. Tout changement d'état d'un objet peut déclencher une action. Les actions activables sont plus diversifiées dans les E4Gs avancés que dans les environnements traditionnels : un formulaire, un rapport et un menu constituent des actions potentiellement déclenchables à l'un des changements d'état d'un objet de l'application. La dynamique de l'application est alors éclatée au sein des différents objets qui composent l'application, et principalement au sein des objets de l'interface. Nous avons également remarqué, lors de l'expérimentation de certains E4Gs, que les actions étaient plus élémentaires que dans les environnements traditionnels puisqu'elles pouvaient être associées à un objet élémentaire de l'application (un champ de l'interface par exemple).

Cette conception nouvelle des actions et des enchaînements d'actions au sein d'une application nécessite une approche différente de la part du développeur. Dans un

environnement traditionnel, la dynamique de l'application est intégrée au sein d'un module qui s'exécute de façon linéaire. Dans les nouveaux environnements que nous considérons ici, la dynamique n'est plus gérée explicitement. Le développeur la décrit au sein des différents objets de l'application. Il n'est donc plus possible de l'implémenter indépendamment de ces objets, comme le préconisent les démarches classiques de conception.

Cette conception autonome de la dynamique de l'application ne constitue pas la seule inadéquation des démarches de conception classiques vis-à-vis des E4Gs avancés. Le fait qu'un objet de l'interface, le formulaire ou le rapport, intègre l'ensemble des actions liées aux changements d'état de ses objets constitutifs nous a également amené à remettre en question l'indépendance entre l'interface et les fonctionnalités de l'application. Un formulaire ou un rapport semble bien réaliser une fonctionnalité de l'application en abritant une transaction sur une donnée de la base de données. Les procédures de traitement associées à la transaction ne contribuent qu'à la réalisation de sous-objectifs, d'objectifs élémentaires nécessaires à la réussite de la transaction. Elles sont, en quelque sorte, subordonnées à la donnée et à l'objet interactif associé à cette donnée. Une démarche de conception "orientée E4Gs" doit renverser la hiérarchisation des composants base de données, traitements et dialogues proposée initialement par les démarches de conception classiques.

C'est ce que nous avons tenté de réaliser, à l'issue de ce travail, en suggérant une démarche de conception dans laquelle les données conservent une place primordiale, mais où leur conception et celle du dialogue supplantent la conception des traitements. Les traitements spécifiés dans notre démarche ne constituent, pour la plupart, que des validations de contraintes sémantiques, portant sur une donnée de la base de données. Mais, même s'ils constituent un traitement de calcul, ils sont néanmoins toujours associés à un objet de l'interface, qu'il soit élémentaire (un champ d'un B/G) ou agrégé (un B/G ou un F/R).

Il nous semble donc avoir atteint l'objectif de ce mémoire qui consistait à rechercher des concepts méthodologiques adaptés à un développement d'application dans un E4G avancé. Nous sommes cependant conscients de certaines limites de la démarche proposée (spécification d'une seule phase, restriction aux applications interactives constituées uniquement de transactions sur une donnée de la base de données, ...). Lever ces limites pourrait constituer un des objectifs de recherches ultérieures dans le domaine des Environnements de Quatrième Génération.

Bibliographie

[Bar 88] BARTHET M.F., *Logiciels interactifs et ergonomie*, Dunod, 1988.

[Bod 88a] BODART F., PIGNEUR Y., *Conception assistée des applications informatiques, Tome 1, Etude d'opportunité et analyse conceptuelle*, Masson, 1988.

[Bod 88b] BODART F., WARNANT G., *Notes du cours Interfaces Homme-Machine*, Institut d'Informatique, FUNDP, Namur, 1988.

[Bou 84] BOUTEL J.S., PIGNON J.P., *Problématique et typologie des nouveaux langages de l'informatique*, 01 Informatique, n°178, Avril 1984; *Concepts fondamentaux des nouveaux langages de l'informatique*, 01 Informatique, n°180, Juin-Juillet 1984; *Langages de quatrième génération : les produits disponibles*, 01 Informatique, n°183, Septembre-Octobre 1984; *Langages de quatrième génération : une tentative de classification*, 01 Informatique, n°184, Novembre 1984.

[Hai 86] HAINAUT J.L., *Conception assistée des applications informatiques, Tome 2, Conception de la base de données*, Masson, 1986.

[Hai 88a] HAINAUT J.L., *Les environnements de quatrième génération*, Journ'Almin, n°8, Septembre 1988.

[Hai 88b] HAINAUT J.L., *Notes de cours Introduction à la théorie relationnelle des bases de données*, Institut d'Informatique, FUNDP, Namur, 1988.

[Hen 87] HENNEAUX Th., SMETS P., *Les outils de quatrième génération : concepts et enquête*, Mémoire de fin d'études, Institut d'Informatique, FUNDP, Namur, 1987.

[Lbd 88] Groupe LBD4G (AFCET), *Langages de quatrième génération*, Dunod, 1988.

[Mar 85] MARTIN J., *Fourth generation languages, vol.1 : Principles*, Prentice-Hall, 1985.

[Mar 86a] MARTIN J., *Fourth generation languages, vol.2 : Reprresentative 4GLs*, Prentice-Hall, 1986.

[Mar 86b] MARTIN J., *Fourth generation languages, vol.3 : 4GLs from IBM*, Prentice-Hall, 1986.

[Pig 89] PIGNEUR Y., *Construction d'une application informatique à partir de la spécification d'une phase IDA*, Mémorandum technique M-31, Institut d'Informatique et d'Organisation, HEC, Lausanne, Avril 1989.

[Ruc 88] RUCHE J.C., *Méthodologie de spécification d'une interface homme-machine*, Mémoire de fin d'études, Institut d'Informatique, FUNDP, Namur, 1988.

IMPRESSION : SPRL Centre de Reprographie - Tél. : 081/22.87.72