Université de Nancy 1

UER
Sciences Mathématiques

Centre de Recherche en Informatique de Nancy

LASSIF

LAngage de Spécification des Systèmes d'InFormation

Logiciel d'Aide à la Spécification des Systèmes d'InFormation

THESE

présentée et soutenue publiquement le 8 juin 1985
devant la commission d'examen
A L'UNIVERSITE DE NANCY 1

pour l'obtention du grade de
DOCTEUR ES-SCIENCES MATHEMATIQUES
par

Odile THIERY

Président	*J.C. DERNIAME*
Rapporteurs	*J.P. FINANCE* *C. ROLLAND* *F. BODART* *J. KOULOUMDJIAN*
Examinateurs	*M. CREHANGE* *O. FOUCAUT*

La célébration est l'attitude vers laquelle tend toute l'activité humaine. Le plus grand art est de savoir remercier.

J. DE BOURBON BUSSET
"Tu ne mourras pas" Gallimard

Il est vrai que remercier est un art difficile. Il est vrai aussi que l'on ne parvient pas à la soutenance d'une thèse d'état sans beaucoup d'aide et de compréhension de ses professeurs et pairs. Aussi je tiens à remercier :

- *J.C. DERNIAME, Professeur à l'Université de Nancy 1, directeur du CRIN et président de ce jury, pour toute la connaissance qu'il m'a apportée lorsque j'étais étudiante. Je lui dois un amour immodéré pour l'assembleur et le système. Il fait beaucoup pour le développement du CRIN, tout en restant accessible et à l'écoute des chercheurs que nous sommes.*
- *J.P. FINANCE, Professeur à l'Université de Nancy 1, sans qui ce travail ne serait pas. Il m'a initiée avec patience et pédagogie aux secrets des Types Abstraits. Je tiens à lui exprimer toute ma reconnaissance pour sa gentillesse, sa compétence et ses efforts pour m'accorder du temps. Il anime de plus le groupe "programmation" au CRIN au sein duquel règnent une bonne humeur et un goût de mise en commun des idées qu'il provoque par son ouverture d'esprit.*
- *C. ROLLAND, Professeur à l'Université de Paris I, animatrice du projet REMORA dont beaucoup d'idées ont servi à l'ossature de cette thèse. Elle sait ce que je lui dois aussi bien en matière d'enseignement, où sa pédagogie a été pour moi un exemple à suivre, qu'en matière de recherche, où j'ai tout appris à son contact en tant que jeune chercheur. Qu'elle soit remerciée pour ses qualités de compétence et de rigueur mais aussi pour sa grande chaleur humaine.*

Je remercie également :

- *F. BODART, Professeur à l'Institut Universitaire Notre Dame de la Paix à Namur, pour l'intérêt qu'il a toujours porté au projet REMORA et l'effort qu'il a fait de s'intéresser à mon travail. Je le remercie d'avoir bien voulu être rapporteur.*

- *J. KOULOUMDJIAN, Professeur à l'IUT Informatique de Lyon Villeurbanne, pour toutes les remarques utiles qu'il a faites sur cette thèse auxquelles j'ai essayé de répondre pour améliorer la qualité de mon travail. Je le remercie, particulièrement, d'avoir accepté d'être rapporteur et d'avoir fait ainsi, s'il était nécessaire, une fois de plus la preuve des qualités humaines qui lui sont largement reconnues.*

Que les autres membres du jury trouvent également ici l'expression de ma gratitude :

- *M. CREHANGE, Professeur à l'Université de Nancy II, s'intéresse à ce domaine jonction des Bases de Données et de la théorie des Types Abstraits. Sa chaleur, sa compétence rendent les discussions de recherche toujours enrichissantes. Je lui sais gré de participer à ce jury.*
- *O. FOUCAUT, Maître-assistante à l'Université de Nancy 1, est en fait à l'origine de l'idée d'introduire les Types Abstraits pour formaliser les concepts du modèle de REMORA. Notre accord, voire notre complicité, aussi bien pour l'enseignement que pour la recherche sont connus de tous. Il nous est très agréable de travailler ensemble et j'espère que notre collaboration se développera encore. Je la remercie plus particulièrement d'avoir relu cette thèse avec attention et de m'avoir aidée à l'améliorer.*

J'adresse également mes remerciements à tous les membres du groupe programmation et du CRIN en général qui, d'aucuns m'ont initiée aux joies de LISP et de CEYX, d'autres aux concepts d'arbres abstraits et aux environnements de programmation, d'autres enfin aux plaisirs de UNIX.
Que tous trouvent ici l'expression de ma gratitude.
Le CRIN est devenu un lieu de rassemblement grâce à l'esprit d'entraide et à la chaleur qui y règnent.

Tous connaissent à la fois la compétence et la gentillesse de N. GAUTIER. Cent fois sur le métier remettant son ouvrage, elle a beaucoup apporté à la qualité de réalisation de ce travail, acceptant toujours avec bonne humeur les modifications et autres corrections du texte.

Enfin je voudrais remercier :

- *Ceux qui ont permis, un jour, qu'une petite étudiante de presque 19 ans puisse accéder, au sortir de l'IUT Informatique, à la maîtrise d'Informatique. Sans eux bien sûr ce travail n'aurait pu avoir lieu. Je tiens à citer tout particulièrement C. PAIR et G. TISSIER qui ont tant fait pour le développement du département Informatique.*
- *Mes parents qui m'ont poussée dans cette voie et m'ont permis de sauter successivement toutes les étapes que je m'étais fixées.*

Comment ne pas être mièvre dans les remerciements que l'on peut adresser à ses proches. Peut être vaut-il mieux ne rien dire ... J'espère simplement qu'ils n'auront pas trop pâti de tout le temps que je n'aurai pas pu leur accorder... et il est sûr que si je n'avais pas trouvé chez nous compétence scientifique, soutien moral et matériel, ce travail n'aurait pu exister.

A Jacques,

S O M M A I R E

PARTIE III. Une solution relationnelle pour le processeur de la dynamique

.<Je distingue deux moyens de cultiver les sciences : l'un d'augmenter la masse des connaissances par des découvertes ; et c'est ainsi qu'on mérite le nom d'"inventeur" ; l'autre de rapprocher les découvertes et de les ordonner entre elles, afin que plus d'hommes soient éclairés, et que chacun participe, selon sa portée, à la lumière de son siècle...>>

DIDEROT

INTRODUCTION

Ainsi, recommençant un ouvrage vingt fois,
Si j'écris quatre mots, j'en effacerai trois...

BOILEAU Satire II <<A Molière>>

1. INTRODUCTION

En dépit du nombre considérable de méthodes d'analyse proposées, de par le monde, pour concevoir des Systèmes d'Information (SI), il existe fort peu de propositions établissant un consensus utilisateurs-gestionnaires/concepteurs sur les SI qu'ils construisent.

Le principal objectif de ce travail est de développer des solutions pour la spécification abstraite de SI de qualité en Informatique de gestion ; en se plaçant au confluent de deux domaines de l'informatique : l'informatique d'organisation et la théorie de la programmation. Le travail s'appuie sur les résultats du projet REMORA [ROL 82].

2. LA CONCEPTION DES SI EN INFORMATIQUE DE GESTION

2.1. Définition du SI, du SIA

Nous retenons la définition du SI de N.C. CHURCHILL [DUF 80] :
<<combinaison formalisée des ressources humaines et informatiques résultant de la collecte, de la mémorisation, de la recherche, de la communication et de l'utilisation des données, en vue de permettre une gestion efficace des opérations au sein d'une organisation>>.

Nous nous limitons, dans ce travail, au Système d'Information Automatisé (SIA) que nous voyons comme une collection de Données et de Transactions sur les Données (figure 1).

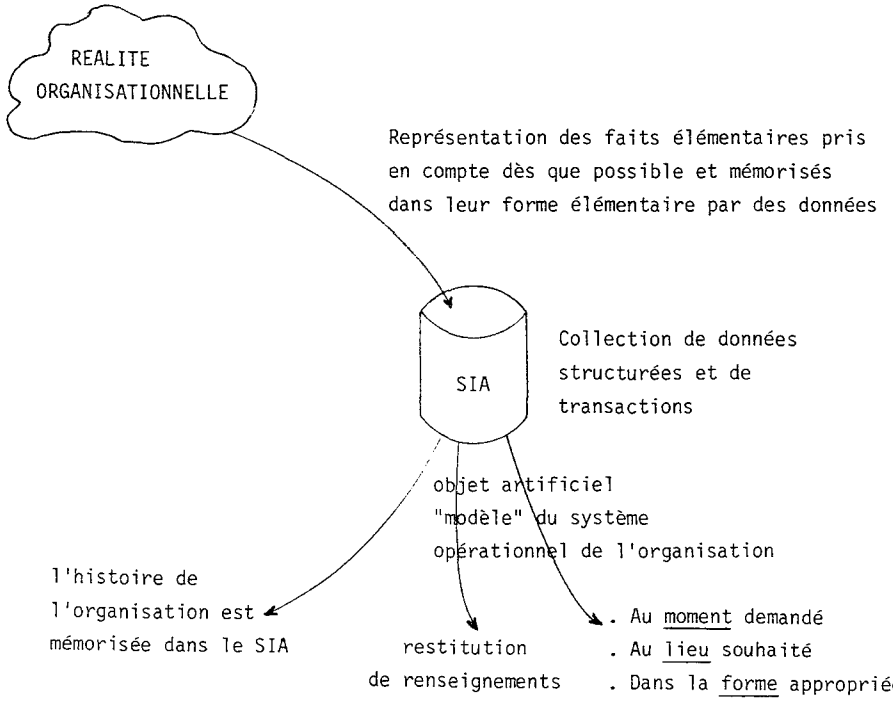


Figure 1 le SIA . ensemble de représentations
. support de décision

Le SIA est un miroir sur lequel le fonctionnement de l'"univers du discours" [ISO 81, NIJ 77], ou l'"univers du problème" [FIN 79, DER 79] est projeté avec des réductions temporelles et spatiales ; il est à la base des choix pris par les décideurs dans les entreprises.

2.2. La classe particulière des problèmes en Informatique de Gestion.

Les applications de l'Informatique de Gestion résultent d'un cadre bien particulier, où il faut envisager :

(i) le nombre de détails qu'il y est nécessaire de prendre en compte (par exemple pour résoudre l'automatisation d'un calcul de paie ou d'une comptabilité),

(ii) l'usage que l'on fait des données,

(iii) la spécificité des contraintes techniques en matière de stockage de l'information,

(iv) les structures mêmes de l'organisation relatives à ses règles de fonctionnement.

Pour répondre à cette spécificité, différentes solutions ont été apportées.

2.3. Les résultats essentiels de l'Informatique de Gestion

Trois étapes sont fondamentales dans les recherches en Informatique de gestion :

(i) Le développement des Méthodes d'Analyse, telles que CORIG [SGI], ARIANE [GAM 77], MINOS [SLIGOS], PROTEE [SIS 78] et LCP [WAR 75].
Basées sur le concept de programmation structurée, elles ont été les premières à préconiser une étape de raisonnement abstrait lors de l'élaboration d'un système.
Leurs résultats sont un ensemble de méthodes (ou canevas) associées à des outils et des langages.

(ii) Le rapport ANSI/SPARC [ANS 75, TSI 77] a introduit l'étape fondamentale de conception abstraite d'une Base de Données à partir de modèles [CHE 76, KEN 77, FLO 77, BUB 77, BEN 76, BRE 79, HSU 79, BRO 81].

(iii) Le rapport du groupe ISO [ISO 81] a introduit la notion de système d'abstraction et complété la définition des modèles conceptuels [ANT 81, FOU 82] et des outils de gestion des SI construits.

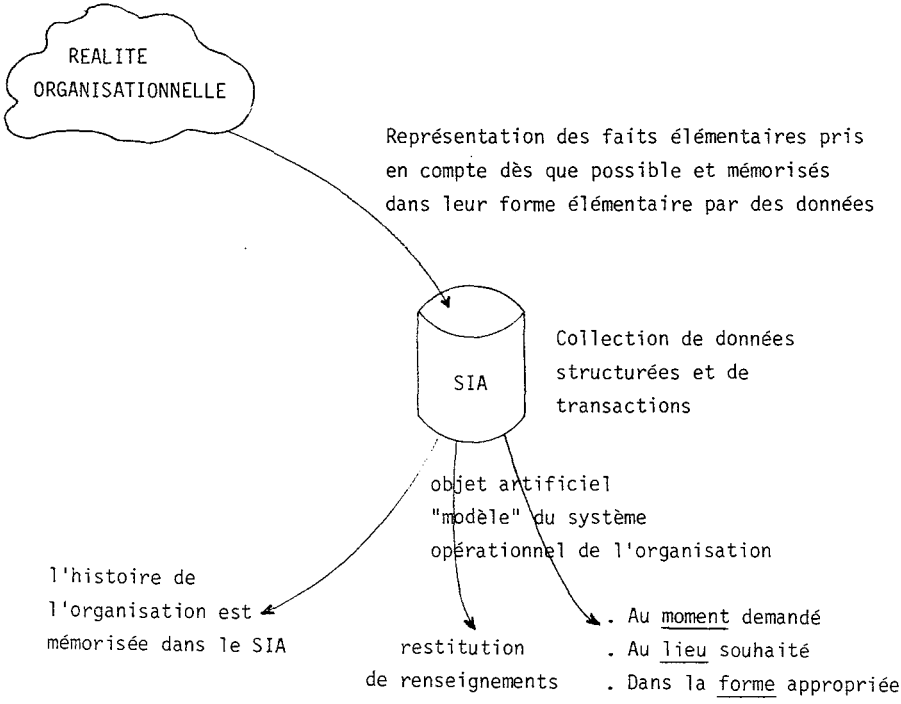


Figure 1 le SIA . ensemble de représentations
. support de décision

Le SIA est un miroir sur lequel le fonctionnement de l'"univers du discours" [ISO 81, NIJ 77], ou l'"univers du problème" [FIN 79, DER 79] est projeté avec des réductions temporelles et spatiales ; il est à la base des choix pris par les décideurs dans les entreprises.

2.2. La classe particulière des problèmes en Informatique de Gestion.

Les applications de l'Informatique de Gestion résultent d'un cadre bien particulier, où il faut envisager :

(i) le nombre de détails qu'il y est nécessaire de prendre en compte (par exemple pour résoudre l'automatisation d'un calcul de paie ou d'une comptabilité),

(ii) l'usage que l'on fait des données,

(iii) la spécificité des contraintes techniques en matière de stockage de l'information,

(iv) les structures mêmes de l'organisation relatives à ses règles de fonctionnement.

Pour répondre à cette spécificité, différentes solutions ont été apportées.

2.3. Les résultats essentiels de l'Informatique de Gestion

Trois étapes sont fondamentales dans les recherches en Informatique de gestion :

(i) Le développement des Méthodes d'Analyse, telles que CORIG [SGI], ARIANE [GAM 77], MINOS [SLIGOS], PROTEE [SIS 78] et LCP [WAR 75].
Basées sur le concept de programmation structurée, elles ont été les premières à préconiser une étape de raisonnement abstrait lors de l'élaboration d'un système.
Leurs résultats sont un ensemble de méthodes (ou canevas) associées à des outils et des langages.

(ii) Le rapport ANSI/SPARC [ANS 75, TSI 77] a introduit l'étape fondamentale de conception abstraite d'une Base de Données à partir de modèles [CHE 76, KEN 77, FLO 77, BUB 77, BEN 76, BRE 79, HSU 79, BRO 81].

(iii) Le rapport du groupe ISO [ISO 81] a introduit la notion de système d'abstraction et complété la définition des modèles conceptuels [ANT 81, FOU 82] et des outils de gestion des SI construits.

3. LA SPECIFICATION DES SI EN THEORIE DE LA PROGRAMMATION

3.1. La classe des problèmes

Le noyau principal de ce domaine de recherche est la spécification de problèmes, limités dans leur volume mais complexes dans les traitements, où tous les efforts ont concouru pour définir des spécifications de qualités

Spécifier un problème [FIN 79]

Etant donné un problème, décrit de manière imprécise, en donner une spécification précise à partir de laquelle on puisse dériver une solution.

Le but est de se définir un cadre formel pour valider la spécification obtenue et démontrer ses propriétés. Le cadre formel est, en fait, lié à la théorie du premier ordre [FIN 79] ou du deuxième ordre [DUF 80].

Définition d'un SI [FIN 79]

Un système d'information est un calcul des prédicats typé du premier ordre spécifiant l'univers d'un problème.

Depuis l'origine des recherches en théorie de la programmation, i.e. l'étude de meilleurs langages de programmation, jusqu'aux préoccupations actuelles des méthodes de construction et transformation de programmes, divers résultats ont été obtenus.

3.2. Les résultats essentiels de la théorie de la programmation

On doit à ce domaine :

- l'introduction des Types Abstraits algébriques et des systèmes de réécriture pour spécifier un problème et valider la spécification obtenue [LES 75, REM 82, FIN 79] ;
- la proposition de langages de spécification associés [FIN 79, WUL 76, LAM 77] ;
- des méthodes de conception de programmation [FIN 79, LAU 76, SOU 82, DUB 84] ;

- des environnements de programmation [DON 75, TEI 81, JAC 84] proposant un ensemble d'outils riche et intégré, proche des développements des ateliers construits en génie logiciel [BIGRE 80-82, AND 84, MIN 84, KRA 82].

4. LES POINTS COMMUNS ENTRE CES DEUX DOMAINES

Ils résultent de la problématique de la conception d'un SI et des moyens que l'on peut proposer pour la résoudre.

4.1. La problématique de la conception

L'accroissement énorme des besoins, le coût de développement des programmes, conduisent à la nécessité d'augmenter la qualité et la longévité des logiciels et donc d'améliorer l'efficacité de la conception.

Cette réflexion conduit à une conclusion unanime, quel que soit le domaine d'application de l'Informatique : il est nécessaire, avant d'aborder une réalisation effective d'un problème, d'en construire une spécification formelle et abstraite c'est-à-dire indépendante de toute contrainte technique.

La spécification d'un système est une tâche difficile pour laquelle il est nécessaire de mettre en oeuvre deux étapes :

(i) spécifier une expression abstraite de la réalité à représenter ; étape, qualifiée selon les domaines d'abstraite [ABR 80, FIN 79, PAI 79, BROY 78, GOG 79] ou de conceptuelle [ANS 75, TSI 77, ISO 81] ;

(ii) choisir une réalisation concrète à partir de cette expression.

La spécification obtenue doit être de qualité [DER 79, THI 76] c'est-à-dire qu'elle doit être une image fidèle et cohérente de la partie du monde réel modélisée, et également établir un consensus entre les utilisateurs futurs et les concepteurs du SI.

4.2. Les moyens à mettre en oeuvre

Les moyens de spécification sont définis selon quatre axes :

(i) La spécification d'un système est le résultat d'un travail d'analyse pour lequel il faut disposer de "modèle" de pensée qui aide le concepteur à structurer la réalité. Un modèle doit permettre une conceptualisation formelle, claire, abstraite et complète ; il est la base de la représentation de la connaissance, que son domaine d'application soit plus explicitement Bases de Données [CHE 76, BRA 76, FLO 77] ou SI complet [BEN 76, BRO 82, VER 82, FOU 82] ou lié à l'Intelligence Artificielle [CRE 83, KOW 81, ISO 81, BEL 83] ou à la théorie de la programmation [QUE 84, SMI 77, BROY 78] ;

(ii) pour décrire les résultats de la modélisation, il faut disposer de langages clairs, de haut niveau, aussi abstraits que possible. Notons que c'est surtout en théorie de la programmation qu'ont été proposés des langages de spécification non procéduraux dont certains principes ont trouvé leur application en Bases de Données [PIR 76, CRE 75, STO 76, AST 76] ou en conception de Systèmes d'Informations [QUE 84, DUB 84, ABR 77] ;

(iii) une méthode associée au langage et au modèle permet au concepteur de réaliser sa spécification en lui proposant une démarche de raisonnement ; qu'elle soit orientée Informatique de gestion [IFI 82-84] ou construction de programmes [FIN 79, QUE 84, JAC 84] ;

(iv) à bien des étapes de la construction de la spécification, peuvent être proposées des aides automatisées, que ce soient des générateurs [HER 74, LAMY 77, SGI, SLIGOS, SIS 78], des outils de contrôle et de documentation [THI 76, KEB 84, IFI 82-84] ou de véritables environnements de programmation [BRIGRE 80-82, GENIE 85]. Un logiciel d'aide à la spécification est indispensable.

5. NOS OBJECTIFS

Notre but essentiel est de montrer que l'intégration des résultats de l'Informatique de Gestion et de la théorie de la programmation est possible et présente un intérêt pour les deux domaines de recherche.

Le principal intérêt de cette démarche pour la conception des SI est de rendre plus formelle et intégrée la modélisation du monde réel.

Pour répondre à ce souci nous avons :

(i) défini une formalisation des SI à l'aide des Types Abstraits s'appuyant sur quatre schémas de type prédéfinis, permettant la spécification d'un SI : ceci a constitué notre premier objectif de travail,

(ii) développé un langage et une méthode associés à la formalisation : ceci a constitué le deuxième objectif de notre travail,

(iii) réalisé un logiciel d'aide à la construction de la spécification : ceci a constitué le troisième objectif de notre travail,

(iv) appliqué notre propre démarche et notre propre langage, pour autovalider nos propositions, à un sous-ensemble du logiciel de gestion des SI : ceci a constitué la dernière étape de notre travail.

6. PLAN DE LA THESE

6.1. Contexte de notre recherche

Ce travail s'est déroulé, en grande partie, dans le cadre du projet REMORA développé et dirigé par C. ROLLAND depuis 1974.
L'équipe REMORA a fait un ensemble de propositions pour répondre aux problèmes soulevés par la conception de schémas conceptuels de SI de qualité [ROL 82].
Plus précisément, a été développée une démarche structuraliste pour concevoir des SI en Informatique d'organisation. Ces différentes recherches ont abouti à la proposition d'un modèle conceptuel complet et d'outils de gestion du SI présentés dans la thèse d'état de O. FOUCAUT [FOU 82]. Par la suite nous avons affiné le raisonnement de modélisation par l'introduction des Types Abstraits en collaboration avec J.P. FINANCE à Nancy.

6.2. Plan de la thèse

Pour répondre aux objectifs de notre travail, cette thèse comporte trois parties découpées en chapitres et intitulées respectivement :
- LASSIF : un LAngage de Spécification de Système d'InFormation,
- LASSIF : un Logiciel d'Aide à la Spécification des Systèmes d'InFormation,
- Une solution relationnelle pour le processeur de la dynamique.

La première partie (chapitre 1) présente une analyse bibliographique de différents projets actuellement développés dans le cadre des recherches sur les SI en Informatique d'Organisation. Nous y dégageons des critères d'évaluation, notamment en ce qui concerne les thèmes de spécification abstraite et de modélisation à l'aide des Types Abstraits (TA). Nous concluons ce chapitre sur la nécessité de proposer une formalisation à l'aide des TA pour les SI en Informatique de Gestion.

Le chapitre 2 de cette première partie introduit la formalisation des SI, fondée sur les chémas de type permettant d'exprimer les concepts de base du modèle REMORA [FOU 82], i.e. les concepts de classe d'objets, classe d'événements et classe d'opérations. Le raisonnement s'appuie sur une axiomatisation algébrique et se conclut sur l'intérêt d'une telle formalisation pour notre modèle.

Un troisième chapitre est consacré à la présentation du langage de spécification choisi (LASSIF). Le langage structurel permet la description des aspects statiques d'un SI, il est fait fort proche de la formalisation précédente. Le langage d'énoncé permet l'expression des textes de programmes, il est à syntaxe quasi-libre, prédicatif et non procédural.

La deuxième partie est consacrée à l'exposé de nos résultats pratiques en matière de prototype de gestion d'un SI. Nous y présentons le logiciel d'aide à la construction de la spécification d'un SI (LASSIF) et le logiciel d'interprétation de la spécification construite (LISSIF). Nous justifions nos choix de réalisation présents et futurs.

La troisième partie est consacrée à la présentation de travaux qui sont à l'origine de notre réflexion : la justification de l'intérêt d'un outil de gestion d'un SI au cours du temps. Nous spécifions cet outil à l'aide de LASSIF et explicitons une solution relationnelle de réalisation après avoir défini la fonction de représentation Types Abstraits $\Rightarrow$ modèle relationnel.

PARTIE I

L A S S I F :

LAngage de Spécification

de Systèmes d'Information

basé sur les types abstraits

INTRODUCTION

Le but de cette première partie est de proposer un langage de spécification de Système d'Information reposant sur une théorie de SI basée sur les Types Abstraits. Ainsi que nous l'avons explicité dans l'introduction générale, la spécification d'un SI à l'aide des nouveaux concepts développés actuellement dans les différents domaines de recherche repose, en grande partie, d'une part sur le concept d'abstraction, d'autre part sur les outils de modélisation que sont les types abstraits. L'idée de base est la nécessité de structurer la description d'un SI de la façon la plus riche possible c'est-à-dire, à notre point de vue, de la façon la plus abstraite et la plus typée possible.

En effet, ainsi que le montre le développement qui suit, par ses caractéristiques intrinsèques, l'abstraction de Données à l'aide des Types Abstraits est non seulement un outil modulaire qui permet des contrôles sous forme de restrictions et d'invariants, mais c'est également un outil de structuration de Données plus riche que les modèles actuellement proposés tels que le modèle relationnel [COD 70] ou le modèle entité-association [CHE 76]. L'abstraction de Données à l'aide des Types Abstraits permet, dans la mesure où elle repose sur un modèle descriptif du monde réel de qualité, une meilleure représentation de la réalité ; en effet chaque constructeur, chaque schéma de type introduits répondent à la définition d'un problème particulier et donc augmentent la richesse de représentation.

Cette première partie s'ordonne suivant trois chapitres :

1) Le premier est un chapitre bibliographique sur l'état de l'art à ce sujet : il repose sur une analyse de différentes méthodes de conception de systèmes d'information. Nous n'avons retenu que des projets orientés abstraction, à l'aide (peu ou prou) des types abstraits, et langage de spécification ; nous faisons précéder cette analyse comparative par une étude bibliographique de l'évolution des techniques d'abstraction au cours des quinze dernières années. Nous conclurons sur les deux points fondamentaux : abstraction et Types Abstraits et sur leur intérêt.

2) Le deuxième nous permet d'introduire une "formalisation de Système d'Information" ; nous y explicitons en particulier l'intérêt, pour nous, d'utiliser les Types Abstraits en conception de SI et montrons comment un schéma conceptuel de SI peut s'exprimer à l'aide de quatre types paramétrés particuliers CLASSE-OB, COPERATION, CEVENT et SI.

3) Le troisième introduit le langage de spécification choisi : langage de description des aspects statiques d'un SI et langage d'énoncé pour les textes de programmes ; fondé sur la théorie introduite au chapitre précédent, donc sur les Types Abstraits, il est déclaratif et prédicatif, se veut d'un haut niveau avec une syntaxe quasi-libre et abordable par un utilisateur non informaticien.

CHAPITRE 1

Méthodes de conception de S.I.

Personne ne peut être convaincu ou réfuté
si ce n'est par les arguments qu'il admet.

"Théologie chrétienne" ABELARD

"Quelques méthodes de conception de SI utilisant les concepts d'abstraction et de Types Asbtraits de données".

1. INTRODUCTION

1.1. Les différents types de méthodes

Les coûts importants de développement des gros logiciels en entreprise, la qualité trop basse du produit obtenu surtout dans le cas de programmes complexes à longue durée de vie ont amené chercheurs et praticiens à repenser les techniques de conception/ réalisation des systèmes informatiques. De nombreuses méthodes se sont développées au cours des années 70, environ une centaine selon BUBENKO dans [BUB 83], que l'on peut classer en deux catégories :

i) des méthodes orientées "analyse des besoins" qui se situent à la première étape du processus de conception des SI : elles ont pour rôle d'établir, en accord avec les utilisateurs finals du logiciel, un consensus sur le contenu du système informatique à obtenir ; pour employer un terme traditionnel d'analyse de gestion : elles servent à la rédaction du cahier des charges de l'application. Leur vocabulaire est souvent basé sur l'analyse des flux de l'information et des prises de décision. A partir de l'analyse des besoins elles visent à caractériser l'aspect informationnel de l'organisation (représenté ou non dans un SI automatisé) ainsi que son aspect décisionnel à tous les niveaux de décideurs (fonction automatisable ou non).
Etant donné qu'il n'est pas de notre propos ici de développer une étude de ce genre de méthodes, nous pouvons citer l'article, de Dr. SURYA et de B. YADAV, publié dans DATA BASE au printemps 83 [SUR 83], qui est une bonne synthèse sur ce type de méthodes.

ii) des méthodes orientées conception structurelle et dynamique de l'organisation, concrétisée par une Base de Données ou un Système d'Information qui se situent en principe en aval des précédentes car elles partent du résultat de l'analyse de besoins pour construire le système automatisé. Il faut cependant noter que la frontière n'est pas toujours très claire, certaines méthodes situées dans la première catégorie ayant des prolongements dans la deuxième (par exemple ISAC [LUN 82A] et DADES [OLI 82]) et inversement certaines méthodes de la deuxième catégorie faisant des propositions pour l'analyse des flux d'information et des besoins (par exemple CIAM [GUS 82], EDM [RZE 82] ou USE [WAS 82A]). Nous nous intéresserons à cette deuxième catégorie de méthodes dont fait partie REMORA.

Parmi les méthodes appartenant à cette deuxième catégorie certaines ont une vue du SI à contruire que l'on peut qualifier de décisionnelle : le SI est alors un objet artificiel essentiellement voué à supporter le système de décision de l'organisation ; il n'est pas possible, dans cette optique, de choisir les informations à stocker et à traiter sans analyser le processus de décision. L'exemple type de cette approche est le développement d'un système expert où le SI peut être considéré comme un moyen de représenter la connaissance sur le réel de l'organisation. L'objectif principal de cette approche est la prise de décision.

D'autres, la plupart, ont une vue du SI à construire que l'on peut qualifier d'informationnelle : le SI est alors un objet artificiel construit pour représenter l'objet naturel "organisation" tel que le traitement de l'information géré par le SI soit toujours une représentation adéquate des transformations des flux physiques de l'organisation. Le SI est alors un modèle de l'organisation au sens de "prototype conceptuel". L'objectif principal de cette approche est la construction d'un SI représentant fidèlement la réalité.

Dans notre étude nous nous attachons aux méthodes ayant une vue informationnelle du SI.

1.2. Evolution des concepts d'abstraction et de Type Abstrait

i) Concept d'abstraction

Contrôler le développement et la maintenance de logiciel a toujours compris la gestion de la complexité des programmes et des systèmes de programmes. Pour mieux appréhender cette complexité un thème dominant dans l'évolution des méthodes et des langages associés est le développement d'outils basés sur le concept d'abstraction ou de ce que l'on appelle en Base de Données la notion de "conceptualisation".

Une abstraction est une image simplifiée ou une spécification d'un système qui recouvre des détails ou propriétés de "niveau abstrait" ou "conceptuel" en en supprimant d'autres. Elle permet de distinguer dans une description plusieurs niveaux : à un certain niveau d'abstraction certaines propriétés sont explicitées tandis que d'autres, jugées plus concrètes, sont rejetées à un niveau inférieur. Une bonne abstraction est significative pour l'utilisateur, elle permet d'exprimer les propriétés fonctionnelles du logiciel à construire sans en donner les techniques de réalisation et d'implémentation.

Depuis 1970 les recherches sont centrées sur la conception de nouvelles structures de données permettant une meilleure représentation de la réalité, ainsi que sur les techniques de spécification permettant de décrire l'image abstraite. Ces recherches viennent des langages de programmation ; elles sont fondées sur le développement de l'outil de modélisation qu'est le Type Abstrait. Cette démarche permet de mettre en évidence différents concepts avec leurs opérations caractéristiques et donc de raisonner à un niveau de détail "abstrait".

ii) Evolution des techniques d'abstraction dans les langages : vers le développement des Types Abstraits

. Avant 1960 le plus important dans les préoccupations de recherche était essentiellement la proposition de langages à syntaxe "propre", adaptés aux différents nouveaux domaines auxquels s'appliquait l'informatique. Les premiers concepts de Types Abstraits ont été introduits dans les langages de programmation par le biais des types prédéfinis tels que ENTIER, REEL, BOOLEEN ou TABLEAU. Les structures de données sont apparues véritablement complètes (et complexes) vers 1968 et l'idée qu'un programme puisse définir les types de données spécifiques à un problème particulier s'est imposée vers 1967. Les différentes recherches ont trouvé leur mise en évidence lors des congrès sur le "SOFTWARE ENGINEERING" de 1968 et 1969 [SOFT].

. Après 1960 on a vu apparaître un effort pour s'abstraire des notations prédéfinies des langages de programmation de façon à ce que tout programmeur puisse ajouter des notations et des types de données au langage de base. Si des recherches sur des "langages extensibles" se sont vite averées plus ou moins sans issue car trop complexes, elles ont fortement influencé les recherches sur les types de données abstraites et sur les définitions génériques des années 70 [SCH 71].

. Au niveau des méthodes de conception de programmes l'évolution des idées s'est déroulée en parallèle : au début des années 70 est apparue la "programmation structurée" [DIJ 76]. La clé du succès de cette méthode est le degré d'abstraction imposé par la sélection de structures de données et d'opérations de haut niveau, qui sont spécifiées de façon informelle et dont on diffère l'implémentation à un niveau plus bas. Cependant, le programme final n'est pas toujours un bon reflet de la série d'abstractions mises en évidence. En effet, on ne sait pas toujours à quel niveau effectuer les choix d'organisation de données et de traitements, vu l'imprécision des spécifications informelles.

. Enfin on peut dire que ce qui a fortement influencé l'apparition des Types Abstraits de données dérive directement des recherches en vérification de programme. Ces techniques visent à formuler des instructions formelles sur les valeurs que peuvent prendre des variables au cours de l'exécution d'un programme et par là, donc, à rendre précises et manipulables les instructions sur lesquelles travaille un programme. Cependant s'est posé le problème de savoir quelle information est utile dans une spécification afin de bien placer les "preuves". Rendre plus formelles les spécifications associées aux problèmes de vérification a été la base du travail sur les Types Abstraits.

. Ainsi, dans les années 70, on a reconnu l'importance d'organiser les programmes en modules [DER 74] de façon à ce que les détails d'implémentation soient aussi localisés et différés que possible. Ceci a conduit à des langages, supports des types de données, basés à l'origine sur le concept de classe de SIMULA [DAH 70]. Durant, en particulier, ces dix dernières années beaucoup de travaux sur les langages de programmation se sont regroupés sous le thème "Types Abstraits". Ce concept est défini comme [HIB 81] <<étendant effectivement l'ensemble des types disponibles dans un programme ; il exprime les propriétés d'un nouveau groupe de variables en spécifiant les valeurs que chacune de ces variables peut avoir et il explique les opérations permises en donnant leur effet sur les valeurs>>. On trouvera dans [HIB 81] un inventaire de ces différentes recherches : ADA [ADA 80], ALPHARD [WUL 76], CLU [LIS 77], EUCLID [LAM 77], CP [BRI 75], GYPSY [AMB 77]...

iii) Evolution des techniques d'abstraction en tout domaine : vers le développement des TA dans tous les domaines

Si le concept d'abstraction à l'aide des Types Abstraits a vu, ainsi que nous venons de l'exposer, son origine dans les recherches en Langages de Programmation, les Types Abstraits ont été très nettement introduits depuis ces cinq dernières années à la fois en Intelligence Artificielle et en Bases de Données. Les échanges croissants entre ces trois domaines ont vu leur aboutissement lors du congrès "High level abstraction" qui s'est déroulé à Pingree Park en 1980 [BRO 80] et lors du congrès VLDB de MEXICO [BRO 83 B]. Ces congrès ont mis en évidence la nécessité d'adopter pour les trois domaines des modèles orientés-objets, intégrant à la fois la modélisation des aspects statiques et dynamiques de la réalité. On y a introduit la notion de Type Abstrait de données comme <<une primitive utilisée en Intelligence Artificielle, Base de Données et Langage de Programmation destinée à atteindre certains buts de modélisation liés à l'intégrité>> ; ainsi le TA y est vu comme un outil de vérification de la cohérence. Evidemment la vue que l'on peut avoir d'un TA dépend du domaine d'application :

- en Intelligence Artificielle : les TA sont vus comme des descriptions incrémentales, partielles des objets du monde réel et sont utilisés pour contrôler un espace de recherche,

- en Langage de Programmation : les TA produisent des moyens pour modéliser des opérations, lier les opérations aux données et protéger des représentations,

- en Bases de données : les TA définissent des caractéristiques d'objets et sont utilisés pour contrôler des états et la validité des opérations. On trouvera dans [BRO 80] le développement complet de la notion de TA suivant les différents domaines où elle s'applique.

De plus, lors de ces congrès, ont été développées les différentes formes d'abstraction de données retenues, supportées suivant leur domaine d'application par des notations variées : la généralisation et la spécification, l'agrégation, la classification et l'instanciation, la représentation et la partition. Chacune de ces formes d'abstraction représente un principe organisationnel permettant de modéliser à la fois la structure et la dynamique du monde réel. Elles ont été établies à partir des recherches sur l'abstraction de données réalisées par J.M. SMITH et D.C.P. SMITH [SMI 77] qui ont introduit les mécanismes d'agrégation et généralisation définis comme tels :

- l'agrégation se réfère à une abstraction dans laquelle une relation entre objets est vue comme un objet de plus haut niveau, ce qui permet de considérer une donnée (ou un traitement) comme un tout alors qu'il s'agit en fait d'une collection de composants ou de parties.
Exemple : un "employé" est un objet agrégé de n° employé, nom, salaire ...

- la généralisation se réfère à une abstraction dans laquelle un ensemble d'objets similaires est vu comme un objet générique, ce qui permet de caractériser un ensemble de données (ou de traitements) par des propriétés communes à plusieurs classes de données (ou de traitements) tout en ne considérant pas leurs caractéristiques individuelles.
Exemple : un "employé" est générique de "secrétaire".

Les deux notions, ainsi que le démontre [SMI 77], s'appliquent parfaitement et enrichissent un schéma de relations en 3FN car elles permettent une structuration plus complète, moins "à plat".

Elles ont été, au cours du temps, complétées par les notions de classification et d'association et ont vu leur application tout au moins partielle dans de nombreux travaux.

- la classification permet de considérer une collection de traitements (ou de données) de haut niveau. La classe définie précise les propriétés partagées par chacun des éléments tout en ignorant leurs différences.
Exemple : classe d'objets EMPLOYE.

- l'association permet de considérer un ensemble répétitif de types de données ou de traitements comme un tout. Cette forme permet de définir des sous-ensembles dans une classe d'objets.
Exemple : l'ensemble "syndicat" est une association de membres "employés".
Notons, d'ailleurs, que CODD a proposé un modèle étendu (le modèle RMT [COD 79]) avec des opérateurs permettant la gestion des valeurs nulles et avec la possibilité de construire des entités par abstractions successives.
En France, les travaux de l'équipe TIGRE [ADI 85], [VEL 84], de BIG [CHR 85] et ceux de R. DEMOLOMBE [DEM 85] vont dans ce sens.

Ces mécanismes d'abstraction sont à la base de certaines des méthodes que nous allons maintenant développer.

2. ANALYSE COMPARATIVE DE QUELQUES METHODES

2.1. Introduction : de la difficulté d'établir un cadre de comparaison

Comme le fait remarquer M.L. BRODIE dans [BRO 83A] il est en effet difficile d'établir un cadre comparatif rigide pour analyser différentes méthodes de conception de SI et, cela, pour plusieurs raisons :

i) Les recherches menées sur la conception de SI sont encore relativement récentes en ce qui concerne de nombreux points en particulier les Types Abstraits ; il est donc difficile d'intégrer ce point comme niveau de sélection,

ii) Le vocabulaire et les termes employés comportent souvent des synonymes voire des polysèmes d'une méthode à l'autre ; par exemple : A. OLIVE dans [OLI 83] situe un certain nombre de méthodes [ACM/PCM, REMORA, USE] à un niveau "logique" dont il donne une définition correspondant à ce que l'on appelle habituellement un niveau "conceptuel" ; par contre il donne une tout autre définition du niveau "conceptuel"...
Certaines tentatives de comparaison du vocabulaire ont été faites dans CRIS 2 [IFI 83], en particulier par FALKENBERG et al. [FAL 83] pour ACM/PCM, CIAM, ISAC et NIAM et par NISSEN [NIS 83] par rapport au vocabulaire du rapport ISO [ISO 81] pour CIAM, NIAM et ISAC,

iii) Les méthodes ont différentes façons d'aborder le problème de la conception

- certaines sont orientées "traitements" : les approches les plus anciennes en conception de SI sont basées sur la définition d'une collection d'actions interdépendantes. Le SI est vu comme un boîte noire qui est alors un processeur d'information produisant

un flux de résultats à partir d'un flux de données. Dans CRIS 1 [IFI 82] un exemple de ce type de méthode est présenté dans ISAC [LUN 82A] ;

- certaines sont orientées donnée: elles proposent une forme de structuration de problème basée sur les structures de données et utilisent des formes d'abstraction reposant sur le concept de schéma conceptuel de Données ou plus directement sur le concept de Types Abstraits : NIAM [VER 82] et ACM/PCM [BRO 82] sont de ce type dans CRIS 1 ;

- certaines sont orientées dynamique :

  . soit elles reposent sur la notion de condition d'état où le SI est vu comme un système générant des transitions d'un état à un autre : certaines de ces méthodes utilisent en particulier les réseaux de PETRI pour représenter la dynamique [RIC 82, PET 80, PET 77, LEO 81, BOD 79, ANT 81].

  . soit elles sont orientées "événement" qui est vu comme un changement d'état remarquable dans le SI et est lié plus ou moins explicitement à des conditions de déclenchement de traitements associés [ANT 81, GUS 82, BEN 76, RID 78, BRE 79].

- certaines, enfin, essaient d'intégrer à la fois la structuration des données, des traitements et de la dynamique des traitements sur les données : c'est le cas de REMORA [ROL 82] et de USE [WAS 82A] qui ont, de plus, ces dernières années, pris une orientation modélisation à l'aide des Types Abstraits.

Il est cependant possible, à notre avis, de définir un cadre minimal de référence qui serve de base à notre analyse avec les hypothèses suivantes.

i) Définition de la notion de SI automatisé

C'est un objet artificiel construit pour représenter un objet naturel. Un SI possède un cycle de vie : il nait au moment de son implémentation, évolue au cours du temps et meurt quand il est devenu obsolète, c'est-à-dire inutile ou inefficace. En réalité, si un SI a su évoluer au cours du temps en accord avec l'organisation qu'il représente, il ne doit jamais devenir obsolète (sauf évidemment s'il s'agit de l'abandon d'une activité). Le manque d'adéquation d'un SI est le résultat de l'incapacité du SI à évoluer et à tenir compte de nouvelles technologies.
Le but des méthodes proposées est de permettre au SI de se développer au cours de son cycle de vie en restant une image fidèle de la vie de l'organisation et de retarder si ce n'est d'éviter (changement d'activité, de langage, de matériel ...) sa mort.

ii) Nous limitons notre analyse aux cycles d'abstraction et de contrôle du SI au sens de [BOD 83] dans CRIS 2 :

Cycle d'abstraction : un SI est une représentation de l'organisation, c'est un modèle qui est une abstraction de la réalité perçue. Le cycle d'abstraction prend en compte les différents types d'abstraction utiles durant le processus de conception afin de se limiter à des classes spécifiques de problèmes (conceptuelles, logiques, physiques) et à ignorer les autres.

Cycle de contrôle : il résulte de la nécessité de contrôler que le SI, durant sa vie, est conforme aux normes requises et aux standards. Le cycle de contrôle fait l'inventaire des différents types de contrôle qui peuvent être appliqués au SI et détermine tous les points de transition du cycle de vie du SI où il doit y avoir contrôle.

Pour répondre au premier point nous analyserons les caractéristiques des modèles et langages de spécification utilisés, pour répondre au deuxième nous nous limiterons aux propositions concernant le respect des qualités de la spécification abstraite.

iii) Idées directrices pour définir notre cadre.

D'après notre introduction il est évident que notre analyse va reposer sur la façon dont les méthodes manipulent les concepts d'abstraction et de Types Abstraits.

Il nous semble que pour être une "bonne" méthode, une méthode de conception de SI doit avoir un certain nombre de qualités ou d'objectifs :

- elle doit se placer à un certain niveau d'abstraction : elle doit permettre un raisonnement indépendant de toute contrainte technique d'implémentation. Cela va bien sûr dépendre des concepts proposés c'est-à-dire, en fait, dans notre domaine, du modèle conceptuel proposé et, par conséquent, de la possibilité de construire une spécification abstraite, par exemple, un schéma conceptuel ;

- elle doit permettre de construire facilement une spécification : et aussi de construire une spécification "compréhensible".

Il faut noter que la spécification formelle obtenue à l'aide des concepts proposés par la méthode doit servir de base de dialogue entre les concepteurs du SI et ses utilisateurs finals. Elle doit donc être "compréhensible" par tous. Pour parvenir à ce but il faut qu'elle soit également facile à construire. Ceci impose des concepts sous-jacents clairs et bien situés, une démarche de construction s'appuyant, si possible, sur un ensemble d'outils d'aide à la conception / réalisation de la spécification, un langage aussi déclaratif que possible afin de ne pas imposer la notion d'algorithme à un utilisateur non

informaticien, enfin une représentation graphique des structures de données et de la dynamique pour bien "visualiser" le résultat de la conception ;

- elle doit permettre de construire une spécification minimale, cohérente et complète : la méthode doit conduire à une spécification formelle de qualité (complète, cohérente, fidèle, non ambigue ...); elle doit donc inclure des outils de vérification formelle et surtout reposer sur un modèle conceptuel permettant une couverture minimale de la réalité au sens de DELOBEL [DEL 73]. En effet, ainsi que le fait remarquer BUBENKO dans [BUB 83], s'il existe tant de méthodes et si la plupart n'ont pas trouvé de diffusion, la principale cause est leur manque de formalisme et donc de concepts sous-jacents formellement définis. Dans ce domaine les apports du modèle relationnel [COD 70] et plus récemment, des Types Abstraits ont été certainement déterminants, vu leur fondement formel.

2.2. Définition du cadre de comparaison retenu

Il couvre quatre sujets principaux :

i) le niveau d'abstraction où se place la méthode, couvre-t-elle le niveau qualifié de "conceptuel" en Bases de Données, c'est-à-dire un niveau de spécification indépendant de la réalisation du SI ?
Peu importe, dans notre analyse, si la méthode couvre d'autres niveaux.

ii) le modèle de spécification proposé par la méthode. Les concepts sous-jacents sont-ils clairs, explicites ou cachés ; couvrent-ils les aspects statiques et dynamiques de la réalité ? Permettent-ils la représentation historique du SI, si oui sous quelle forme ? Ont-ils un bon pouvoir de représentation, c'est-à-dire conduisent-ils à un schéma conceptuel de qualité ?
Les Types Abstraits sont-ils utilisés comme outils formels de modélisation ? Quel est leur développement et leur utilité dans la méthode ?

iii) le langage de spécification proposé par la méthode : son niveau, ses caractéristiques prédicatives ou procédurales. Il faudrait plutôt dire "les" langages car nous y incluons les langages de représentation graphiques proposés par la plupart des méthodes. Les langages proposés forment-ils un tout intégré ? Conduisent-ils à des spécifications de qualité ? Sont-ils orientés utilisateurs ? Ce dernier critère étant souvent lié aux possibilités représentatives de la spécification et à l'aspect déclaratif du langage de spécification.

iv) les outils de spécification y compris les outils de validation proposés par la méthode, de contrôle et d'aide à la conception. Nature et niveau de ces outils, en particulier lors de l'utilisation des TA, quelles sont les possibilités de valider la spécification formelle obtenue ? Quels sont les logiciels proposés ?

Ces quatre points majeurs seront précédés par une présentation succincte de la méthode, du lieu où elle est développée, de l'équipe qui la développe, de ses objectifs "avoués" et de son environnement ; et suivis d'un paragraphe de conclusion rapide sur chacune donnant notre opinion à son sujet.

L'analyse se conclura par un tableau récapitulatif et comparatif reprenant les critères qui nous semblent fondamentaux, critères essentiellement qualitatifs.

Remarquons, en effet, que notre analyse se veut qualitative et sémantique et non quantitative et technique. Pour disposer de ce deuxième type d'analyse on peut se reporter à un article publié par M. PORCELLA, P. FREEMAN, A. WASSERMAN dans ACM SIGSOFT en janvier 1983 dans [POR 83] ainsi que dans CRIS 2 [IFI 83] ; cet article est un résumé d'analyse de questionnaires remplis par vingt cinq concepteurs de méthodes, destinés à explorer une méthode possible pour l'environnement d'ADA. L'analyse a été réalisée selon sept critères principaux qui recouvrent des domaines techniques divers tels que les points compatibles avec ADA mais aussi les supports automatisés et la portabilité des outils. Il s'agit d'une analyse intéressante mais qui poursuit des objectifs différents des nôtres.

Le chapitre I se terminera par la mise en valeur des points importants du développement : notre opinion sur l'abstraction que l'on peut et doit trouver dans une méthode et sur l'utilisation que l'on peut faire des TA dans la spécification des SI en informatique de gestion. Il nous conduira ainsi à la formalisation de SI développée au chapitre II.

2.3. Les méthodes étudiées

Il n'est pas possible de faire un inventaire exhaustif de la centaine de méthodes de conception de SI existant actuellement dans le monde ; ne serait-ce d'ailleurs que parce que la plupart ne rentreraient pas dans le "référentiel" défini précédemment ; de plus nous ne disposons pas toujours de la documentation suffisante pour analyser et juger les méthodes.
Bien que nous n'ayons retenu pour une analyse détaillée que six méthodes répondant à nos préoccupations, bien d'autres sont à des titres divers très intéressantes. Parmi elles nous pouvons citer les méthodes développées lors du congrès CRIS 1 [IFI 82], analysées et comparées lors de CRIS 2 [IFI 83] ainsi que deux nouvelles méthodes présentées lors de CRIS 3 :

- D2S2 [MAC 82] : méthode qui permet le développement coordonné de séries d'applications à l'aide du modèle entité / association. C'est une méthode "commerciale" qui est donc très orientée utilisateur mais elle est essentiellement "manuelle".
- ISAC [LUN 82A] : méthode qui permet d'identifier les problèmes de communication entre analystes et utilisateurs. Elle est orientée "traitements" et ne propose pas de modélisation des données. Elle est très graphique mais essentiellement "manuelle".
- DADES [OLI 82] : méthode de structuration essentiellement orientée donnée. Elle propose un modèle dérivé du modèle relationnel intégrant les aspects statiques et dynamiques du monde réel. Actuellement elle est essentiellement "manuelle".
- EDM [RZE 82] : méthode d'analyse "traditionnelle" couvrant la totalité du cycle de vie du SI. De nombreux outils y ont été développés.

- IML [RIC 82] : essentiellement un outil de description à utiliser dans un processus de conception donc à intégrer dans une méthode quelconque de conception de SI. Il utilise la représentation graphique des réseaux de PETRI [PET 80] pour modéliser les traitements.
- DAISEE [SOL 82] : méthode qui propose modèles et outils d'analyse, de conception et d'implémentation de projet. Elle préconise l'utilisation d'un modèle proche du modèle entité/association intégrant le temps. C'est une méthode essentiellement manuelle.
- CIAM [GUS 82] : méthode qui couvre les étapes de l'analyse des besoins à la conception des traitements. Elle est orientée données et préconise l'utilisation d'un modèle entité/association intégrant le temps. C'est une méthode assistée par un système d'aide à la conception automatisé.

Il faut noter, par ailleurs, le développement de ce type de recherches en conception de Bases de Données et de SI en Europe du Nord. Citons en particulier, en plus des travaux de BUBENKO et GUSTAFSSON sur CIAM en Suède, ceux de SOLVBERG en Norvège [SOL 82] sur ISSM, ceux de LUNDBERG [LUN 82A] en Suède sur ISAC et ceux de NIJSSEN sur NIAM [VER 82]. L'origine de ces travaux est la recherche entreprise par LANGEFORS dès 1963 [LAN 63]. LANGEFORS a été un des premiers à insister sur la notion de temps et à montrer son importance dans la construction de SI ; ses travaux ont été à l'origine, selon [SUR 83], d'une bonne dizaine de méthodes.

- MERISE [LE 84] : méthode née dans les années 70, qui a pour intérêt d'intégrer bien des idées nouvelles telles que la modélisation conceptuelle des données (basée sur le modèle entité/association [CHE 76]) ; et la modélisation conceptuelle dynamique des traitements (basée sur le modèle événement/opération /synchronisation). Elle contribue, vu son développement dans les administrations et les entreprises, à faire progresser en France, les idées issues de la recherche chez les utilisateurs de l'informatique.

- IDA [BOD 84] : méthode qui a vu son développement à la suite des travaux que F. BODART a effectués en collaboration avec TEICHROEW [TEI 79] [YAM 82] dans le cadre du projet ISDOS. Elle est basée sur un modèle conceptuel proche de celui de MERISE (elle est d'ailleurs compatible avec cette méthode) mais intègre également des notions plus traditionnelles d'analyse de gestion telles que l'étude de l'opportunité ou l'analyse fonctionnelle qui la rendent plus assimilable par les gestionnaires. Elle propose de plus tout l'environnement de logiciels du projet ISDOS.

Nous avons rejeté les méthodes précédentes de notre analyse pour certaines parce qu'elles étaient trop orientées analyse des besoins, trop traditionnelles, donc généralement n'étaient pas d'un niveau assez abstrait [EDM, ISAC, D2S2] ; ou qu'elles étaient incomplètes [IML] ; pour d'autres parce qu'elles n'intégraient aucun concept proche des Types Abstraits et donc ne correspondaient pas à notre cadre d'analyse [DAISEE, DADES, CIAM, MERISE, IDA].

D'autres projets, hors CRIS 1, CRIS 2 et CRIS 3, nous semblent intéressants :

- les recherches de GANGE et al. [GAN 82] qui ont été présentées lors du congrès VERY LARGE DATA BASES de Mexico en 82 semblent être très intéressantes car basées sur une approche orientée-objet avec des notions sous-jacentes proches des nôtres,

- A Nancy citons :
  . Les recherches sur le projet TYP [HEN 80] et sur ses frères les projets VEGA [CHA 83] et ADONIS [NAS 83]. Ces dernières recherches semblent particulièrement intéressantes parce qu'introduisant non seulement la définition de Types Abstraits par les utilisateurs mais aussi la manipulation de concepts

relationnels tels que le nuple labellé. Elles ont su intégrer les deux approches importantes en conception de Bases de Données des dix dernières années,

. Les recherches présentées par Eric DUBOIS dans [DUB 84] qui ont pour but de préciser les idées développées dans le projet SPES [FIN 83] à Nancy pour les rendre davantage applicables à la spécification de SI au sein d'une organisation. Après avoir introduit un certain nombre d'outils de spécification qui sont en fait des schémas de types spécifiques à l'informatique d'organisation, E. DUBOIS définit un méta-algorithme de spécification de problème fondé sur un certain nombre de stratégies de construction et sur la programmation déductive [FIN 79] développée à NANCY. Les recherches sont particulièrement intéressantes car elles ont débouché sur une application "grandeur réelle" dans la société Elf-Aquitaine.

Nous nous sommes, en fait, intéressé à des méthodes se situant à un niveau de "spécification" suffisamment abstrait : ce sont souvent des méthodes orientées données ou intégrant les aspects "données" et "dynamique" car ce sont, de fait, les méthodes qui ont accordé le plus d'importance à la modélisation abstraite du monde réel. Parmi elles nous avons volontairement limité notre cadre d'investigation à celles qui préconisent peu ou prou la modélisation à l'aide des Types Abstraits.

Afin de concilier ces préoccupations et le besoin de documentation suffisante pour une étude digne de ce nom, nous avons limité notre investigation à celles des méthodes présentées dans CRIS 1, puis analysées dans CRIS 2, qui intégraient des concepts d'abstraction et de modélisation par les Types Abstraits. Nous en avons retenu cinq classées par ordre croissant de lien "avec les types abstraits" que nous allons successivement détailler.

- NIAM [VER 82]
- SYSDOC [ASH 82]
- SDLA [KNU 82]
- USE [WAS 82A]
- ACM/PCM [BRO 82].

Nous leur avons adjoint REMORA [ROL 82] présentée également lors de ce congrès, base de notre travail. Cette méthode se situe principalement au niveau conceptuel (et a des prolongements au niveau logique) donc à un niveau d'abstraction "de spécification" entrant dans notre cadre. Cependant dans le développement explicité lors de CRIS 1 elle ne comporte pas de notion liée aux Types Abstraits : c'est en fait un des objectifs de nos travaux actuels que d'intégrer les TA comme outil de spécification de SI en informatique de gestion.

N.B. :
Les exemples présentés dans le développement suivant se réfèrent au "cas concret" soumis aux différents concepteurs des méthodes d'analyse présentées lors de CRIS1 [IFI 82] : il s'agit de la gestion de l'organisation d'une conférence IFIP.

2.4. Analyse détaillée des méthodes

2.4.1. La méthode NIAM

A. Présentation

La méthode NIAM (en clair NIJSSEN's Information Analysis Method) a été développée chez Control Data au Pays Bas à partir des travaux de NIJSSEN, connu, plus particulièrement, pour avoir introduit la notion d'univers du discours [NIJ 77]. Elle est actuellement en cours d'exploitation et d'évolution. Cette méthode est très orientée "analyse de l'information" : toute l'analyse des besoins repose sur la conception de diagrammes de flux d'information. Elle possède, de plus, des concepts propres à la modélisation des données et intègre quelques notions abstraites.

Objectifs : proposer des concepts, des langages et des outils lors de l'étape d'analyse de l'information, première étape du développement d'un SI.

B. Niveau d'abstraction

NIAM est principalement une méthode utilisant un cycle d'abstraction spécifique et un cycle de contrôle de la vie du SI. Elle ne couvre pas toutes les étapes de développement d'un SI mais se limite à l'étape d'analyse de l'information indépendamment de toute contrainte technique de réalisation. Elle est donc située, de par les concepts qu'elle recouvre et ses objectifs, à un haut niveau d'abstraction.

C. Les concepts

Beaucoup de termes employés par la méthode sont à l'origine des définitions normalisées données dans le rapport du groupe ISO TC97/SC5/WG3 [ISO 81] sur les "Concepts and Terminology for the Conceptual Schema", rapport qui propose un cadre normalisé pour la description du Schéma Conceptuel (SC).

Les concepts principaux de NIAM sont le SC et les diagrammes de flux d'information. Un SC consiste en un ensemble de types de phrases (ou types de faits), un ensemble de types d'objets non lexicaux, de types d'objets lexicaux et un ensemble de contraintes d'intégrité très puissant. Un trait intéressant est la notion de sous-typage. La dynamique est décrite en utilisant les concepts de base de fonctions et de flux d'information constituant un modèle de fonction. Ce modèle permet une décomposition fonctionnelle montrant des sous-fonctions décrites comme une paire <transformation, flux d'information>.

Nous explicitons ici plus particulièrement les définitions origines de la terminologie proposée par [ISO 81] et introduisons les concepts propres à la modélisation des aspects statiques, c'est-à-dire de la structure du SI.

i) Système objet : part de la réalité pour laquelle on veut collecter de l'information et retrouver l'information éventuellement dérivée. Le SI est un outil qui délivre l'information nécessaire au fonctionnement du système objet ;

ii) Système d'abstraction : modèle mental du système objet consistant en des classes d'objets et d'activités d'une part, des règles d'autre part.

iii) Base d'information : les flux d'information y prennent leur origine et leur terme, elle agit comme un moyen de stockage des messages.
Cette définition est complétée à partir de la notion de
modèle de phrase : la base d'information consiste en un ensemble de phrases élémentaires qui sont un cas particulier des phrases en langue naturelle telle que :

"Pieters vit à Amersfoot"

Une phrase de ce genre, ou "fait", n'a pas de sens par elle même, elle n'amène de l'information que si elle est citée dans un contexte. Ce contexte est précisé par l'introduction de deux nouveaux concepts.

iv) Objets lexicaux : ce sont des chaînes référant un objet.

ex : "Pieters"

Les objets lexicaux sont regroupés en types d'objets lexicaux ou LOT.

ex : LOT nom de famille, LOT titre, LOT nom de ville, etc...

Ce sont en fait les types propriétés habituels en conception de Bases de Données.

v) Objets non lexicaux : Ce sont des choses réelles ou abstraites dans le système objet qui sont regroupées en types d'objets non lexicaux ou NOLOT. Ce sont en fait les types entités habituels.

ex : NOLOT personne, ville, papier ...

ainsi "Pieters" est une occurrence du LOT nom de famille, "Amersfoot" est une occurrence du LOT nom de ville ; la phrase précédente, explicitée par rapport à son contexte devient le type ou modèle de phrase suivant :

"La personne (NOLOT) dont le nom de famille (LOT) est "Pieters" vit dans la ville (NOLOT) dont le nom de ville (LOT) est "Amersfoot".

Cette phrase traduit en fait une association entre les deux objets non lexicaux "personne" et "ville" ; "vit dans" s'appelle un "rôle" en NIAM.

Notons qu'il s'agit toujours d'associations binaires.

vi) Association : chaque phrase a une structure profonde et peut spécifier deux types d'associations entre :

- des objets non lexicaux : on appelle une telle association une "idée" :

ex : "Une personne vit en ville" traduit un type d'idée.

- un objet non lexical et un objet lexical : on appelle une telle association un "pont".

ex : "Une personne est référencée par son nom de famille" traduit un type de pont.

vii) Sous typage : ce concept, directement dérivé de la notion de "généricité" de [SMI 77], permet d'exprimer le partage ou non partage de types propriétés par des types-objets.

ex : le NOLOT "personne" couvre les sous-types "membres du comité de programme" et "représentants nationaux".

viii) Les contraintes d'intégrité : elles sont classées principalement en contrainte de clé identifiante, d'inégalité, d'égalité, de sous-ensemble, de rôle total, d'exclusion et de cardinalité. Toutes ces contraintes sont spécifiées au moyen d'une représentation graphique, les autres sont décrites dans un langage de manipulation procédural. L'ensemble forme un outil d'expression de contraintes très puissant.

D. Utilisation des types abstraits :

NIAM, en plus du fait qu'elle se place à un haut niveau d'abstraction, comporte des notions proches ou directement liées aux TA.

- Directement liée : notion de sous-typage et de généricité.
- Proches : les différents types de concepts introduits même s'ils ne sont pas exprimés en termes de formalisme des TA, en sont proches par leur conception modulaire et normalisée.

Ce qui manque, dans cette optique, c'est une expression des structures de données intégrant pour chaque NOLOT par exemple, les types d'opérations qui lui seraient rattachés.

E. Le langage

Il y a deux sortes de langage :

1) un langage graphique permettant une spécification statique et dynamique du système ; vu la simplicité du graphisme l'utilisateur peut participer à l'élaboration de la grammaire conceptuelle (ou schéma conceptuel) à partir de l'analyse du système objet ;

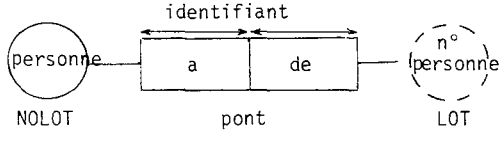
2) un langage formel RIDL (Referential IDea Language) qui est le langage de la grammaire conceptuelle qui permet :

- la spécification des IFD (diagrammes de flux d'information) et des ISD (diagrammes de structure d'information),
- la spécification des contraintes d'intégrité (CI) qui ne peuvent être représentées par une simple notation graphique ; ce sont les contraintes "procédurales",
- la spécification des transformations de flux d'information sous forme de procédures.

RIDL est un langage de description de données et de contraintes d'intégrité utilisant un formalisme à base de mots clés prédéfinis pour les prédicats et procédural pour les textes des fonctions.

EXEMPLES :

a) exemple de CI exprimée graphiquement :



Une personne a un seul n° de personne et vice-versa.

b) exemples de ISD exprimés graphiquement

- la première figure de la page suivante donne le squelette de la structure d'information de la conférence,
- la deuxième, le diagramme de l'information.

c) Utilisation du langage RIDL :

. définition de types entités (NOLOT)

```
ADD NOLOT person, paper, conference
```

. définition de propriétés (LOT)

```
ADD LOT person-nr, paper-nr, surname, title
```

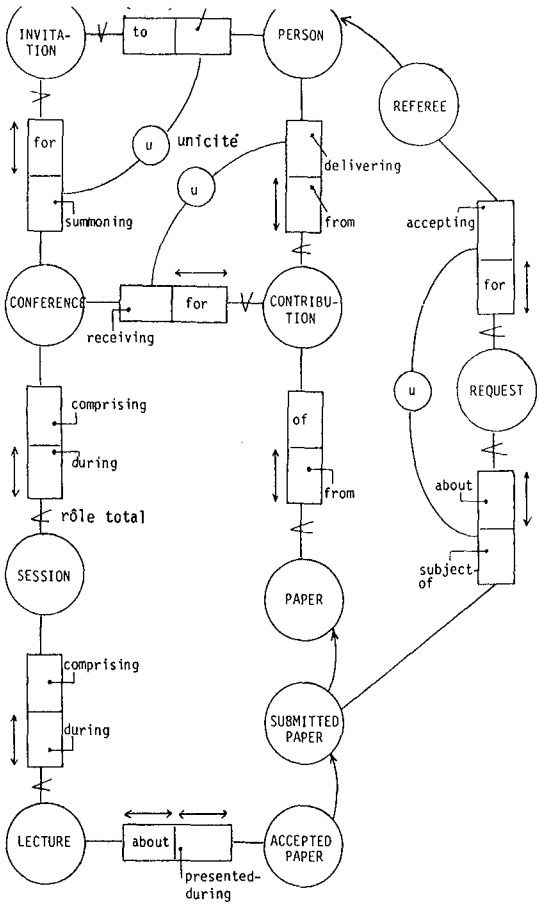
. définition d'une contrainte d'intégrité :

le n° de session identifie la session et tout congrès comprend une session au moins

```
ADD CONSTRAINT session-identification
     CONDITION
          session IDENTIFIED-BY session-nr OF session
          AND conference COMPRISING session
     HOLDS
```

. définition d'une prodécure : listage des conférences invitées

```
     PROCEDURE list-callees IN conf : conference-nr ;
BEGIN
     callees : = person delivering Invited - contribution
FOR conference IDENTIFIED-BY conf ;
SORT callees ON surname OF person ;
FOR callees
     LIST (person-nr OF, initials OF, surname OF)
END ;
```



Structure d'Information de la Conférence IFIP

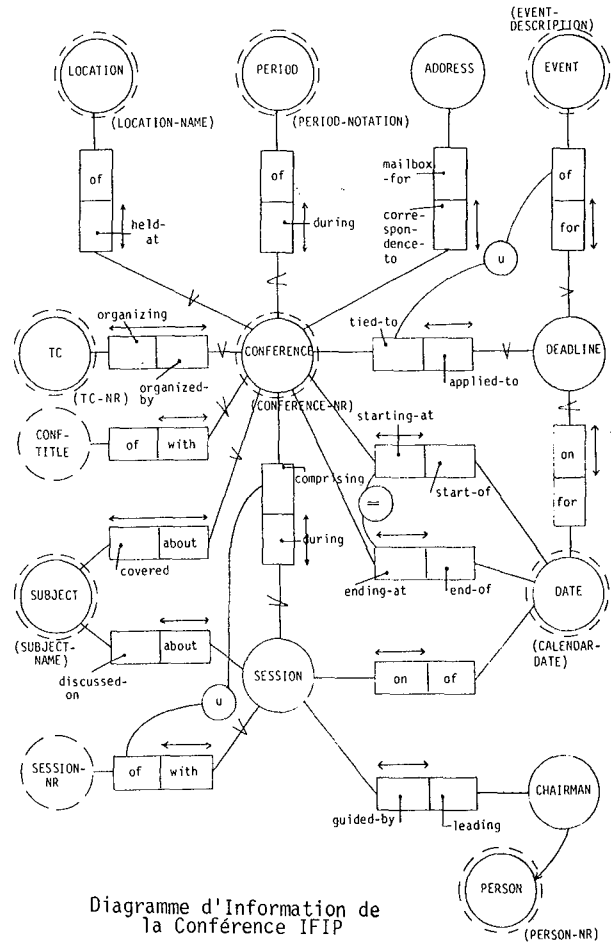


Diagramme d'Information de
la Conférence IFIP

F. Les outils proposés

De nombreux outils de contrôle de la cohérence et de documentation sont proposés tout au long du processus de conception du SI.
Le principal est ISDIS, méta-système constitué d'un dictionnaire d'information et d'un environnement d'outils.

- il stocke et met à jour la grammaire conceptuelle en s'appuyant sur les concepts précédents de NIAM,
- il montre les implications et les conséquences sur la grammaire lors de l'entrée d'une nouvelle définition.
- il compile la grammaire conceptuelle afin de la rendre exploitable par l'ENFORCER.

Ce dernier outil (l'ENFORCER) paraît particulièrement intéressant ; en effet, en s'appuyant sur le schéma conceptuel (ou grammaire conceptuelle) du SI et sur le SI lui même, il met en oeuvre les mises à jour du SI et les productions d'information qui en résultent. Il semble posséder un sous-ensemble des fonctions attribuées par [ISO 81] au processeur d'information. Cet outil ne peut cependant pas faire fonctionner ni même simuler la dynamique du SI à cause du manque de complétude de la description dynamique. La partie graphique n'est pas non plus automatisée, tous ces outils sont associés aux spécifications RIDL.

G. Conclusion

En résumé, ce n'est pas une méthode de conception complète de SI mais une aide à l'analyse.

Ses points forts sont :

- une description statique puissante du SI au niveau conceptuel mais le résultat ne peut, actuellement, pas être généré automatiquement,
- une notation graphique complète qui permet un pont entre l'analyste et l'utilisateur ;

Ses points faibles sont :

- une description dynamique réalisée pour compléter la description statique de façon à concevoir la base d'information mais qui ne permet ni validation, ni simulation de la dynamique,

- la complexité du schéma conceptuel obtenu : dire que la grammaire sera compréhensible par les utilisateurs est optimiste.

2.4.2. La méthode SYSDOC

A. Présentation

SYSDOC est une méthode développée par ASCHIM et BRAATEN dans un institut de recherche à OSLO (Norvège) depuis 1974.
Actuellement MOSTVE dirige le projet au "Central Institute for Industrial Research". Cette méthode a été largement appliquée dans l'industrie et le secteur public. Elle se situe à un niveau de spécification de besoins. Elle s'appuie sur une définition informelle du problème et des objectifs du SI.

Objectifs : proposer des concepts et des outils permettant la conception et la réalisation de SI interactifs.

B. Niveau d'abstration

La spécification obtenue comporte deux aspects principaux : un modèle de données conceptuel qui décrit les informations à stocker dans la base de données, l'interface utilisateur qui décrit surtout les transactions, les règles de gestion et le dialogue, les procédures manuelles.
Les programmes obtenus sont décrits dans un langage de haut niveau indépendant de toute contrainte technique : ce sont, qualifiés par la méthode même, les programmes "abstraits". Cette méthode se situe donc à un haut niveau d'abstraction : elle couvre les cycles d'abstraction et de contrôle.

C. Les concepts

Un modèle de données conceptuel SYSDOC contient trois sortes de concepts : des types entités, des types relations, des types éléments de données. Ces concepts peuvent être élémentaires, dérivés, généralisés ou groupés. Le concept de dérivation permet d'exprimer la dynamique des Données, la combinaison des autres concepts décrit la structure.

i) Concepts élémentaires : ils sont basés sur le modèle entité-association.

α) type entité : objet physique, abstrait ou également événement ; il est décrit par ses propriétés, ses règles générales d'application, et ses contraintes de cardinalité.

Exemples : - un résumé (de communication)
- un "appel" aux communications
- un comité d'organisation (de congrès)

β) type relation : association entre des types entités ; il faut spécifier si la relation est optionnelle ou obligatoire au niveau des occurrences ; elle peut être 1 - 1, 1 - n, m - n et n'a pas de propriété.

Exemples : la personne "dupont" possède la voiture immatriculée "2244 RR 54". Le type-relation correspondant est "possède" entre les types-entités "personne" et "voiture".

γ) type élément de donnée : précisé par son type, sa longueur, ses contraintes d'intégrité, sa spécification par rapport à un écran, il décrit un attribut d'un type-entité mais peut s'appliquer à plusieurs types-entités.

Exemples : . le n° de communication
. l'heure de départ d'un avion

ii) Concepts dérivés

Les trois concepts précédents peuvent être élémentaires ou dérivés.

α) concept élémentaire : une occurrence du concept doit être obligatoirement donnée au SI depuis l'extérieur. C'est une donnée générale du système.

β) concept dérivé : une occurrence du concept peut être dérivée par le SI à partir d'occurrences de concepts élémentaires données au système depuis l'extérieur.

On en déduit que :

. Un type donnée dérivé : est calculé par le SI.
Exemple : le poids moyen d'un groupe de personnes est dérivé (calculé) par le SI si le poids des différentes personnes concernées existe dans le SI ;

. Un type entité dérivé : est une entité fabriquée à partir d'autres types entités.
Exemple : il est possible de définir le type "couple-marié" à partir d'un SI contenant des informations sur les "personnes" et leur naissance, mort, mariage et divorce.

. Un type relation dérivé : est une relation fabriquée à partir d'autres relations ou la combinaison d'une relation et de types données.
Exemple : un client passe une commande qui nécessite un certain travail sur une certaine machine ; on en déduit qu'un client est associé à une certaine machine.

Ces règles de dérivation permettent l'expression partielle de la dynamique du système. Il y manque, principalement, la notion explicite d'événement et d'information dérivée d'un événement, ainsi que l'on peut la trouver dans CIAM [GUS 82] par exemple.

iii) Concepts pour décrire les alternatives et la généralisation : la généralisation, au sens de SMITH et SMITH [SMI 77], est utilisée pour décrire les alternatives.

- un type entité généralisé permet d'exprimer qu'une occurrence d'entité peut être considérée de différents types dans différents contextes.

Exemple : . automobile est une entité généralisée de camion et de voiture.

. entité-légale est une généralisation de compagnie et personne ; une automobile peut être possédée par une entité-légale (compagnie ou personne) mais n'est conduite que par une personne (sous-type de entité-légale) ;

- un type relation généralisé permet de décrire que deux types de relation sont alternatifs pour deux occurrences d'entité participantes.

Exemple : entre projet et personne existent les types relations "participe" (une personne participe à un projet) et "dirige" (une personne dirige un projet). On exprime le fait qu'un directeur de projet n'est pas un participant ordinaire en introduisant un type relation généralisé représentant la réunion des deux types relations précédents et définissant toutes les personnes engagées sur un projet.

- un type donnée alternatif représente un nom donné à deux ou plusieurs types données dont seulement un d'entre eux peut exister pour une occurrence d'entité.

Ces formes permettent de caractériser des sous-ensembles sans les stocker explicitement.

D. Utilisation des types abstraits :

La notion principale de modélisation par les TA est l'utilisation de la généralisation pour décrire les alternatives et permet donc d'en déduire les sous-types et leurs propriétés.

C'est une modélisation très puissante qui conduit à une structuration très riche des données. Cependant il n'y a pas de description type par type, par exemple des types-entités, avec toutes les règles de dérivation qui lui sont applicables mais une description morcelée des différentes règles, certaines graphiques, d'autres spécifiées dans d'autres langages.

E. Les langages

Il y a deux sortes de langages :

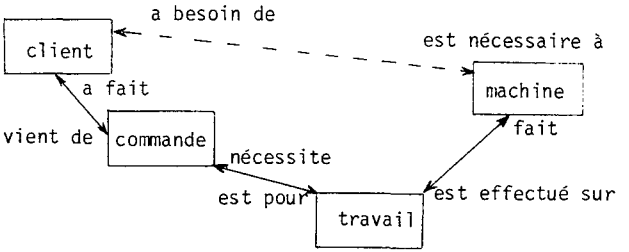
a) un langage graphique qui permet de représenter le modèle de données,

b) le langage SYSDUL : il sert à la spécification des programmes abstraits et à la description de l'interface utilisateur ; il est :

- prédicatif et déclaratif en ce qui concerne les assertions correspondant aux contraintes d'intégrité définies sur la Base de Données,
- de normes proches des propositions du CODASYL [CODA 71] en ce qui concerne l'expression des textes procéduraux. Il s'agit d'un speudo-code COBOL procédural,
- associé aux langages définis pour les écrans en ce qui concerne l'interface utilisateur.

Exemples :

1) exemple de type relation-dérivée
représentée graphiquement (<--->)



2) Définition d'un "invité" en SYSDUL :

personne (qui a soumis un papier) OU (est représentant national d'un comité technique quelconque qui est impliqué dans le congrès) OU (participe - dans(un groupe de travail OU dans un groupe quelconque associé avec le groupe de travail)) OU (qui n'est pas invité en priorité dans le groupe des invités).

3) édition du rapport des référees, pour un papier, en SYSDUL :

```
OBTAIN c-title
FIND AND DISPLAY contr-st OF contrib WITH GIVEN c-title
FIND AND DISPLAY p.nom AND p.adresse OF personne WHO :
          est-auteur-de contrib
FIND AND DISPLAY p.nom AND p.adresse OF ALL personnes :
     WHO est-co-auteur-de contrib
FOR ALL referin WHICH concerne contrib
GET AND DISPLAY d.envoie, d.rec, d.rest, eval.cod :
     AND grade
FIND AND DISPLAY p.nom OF person WHICH referin
     est-fait-par
END FOR
```

Evidemment par ailleurs sont décrits les prédicats correspondant à "est-auteur-de" , "est-co-auteur-de" , "concerne" , "est-fait-par".

F. Les outils

Le logiciel SYSTEMATOR est un ensemble d'outils d'aide à la conception. Ses modules sont les suivants :

1) spécification des besoins : modules pour stocker, modifier, documenter et analyser les besoins du système,

2) mise à jour du modèle de données : module interactif pour stocker et modifier les descriptions détaillées des types entités, relations et données,

3) mise à jour interface utilisateur : module pour stocker les programmes abstraits

4) implémentation : génération du schéma conceptuel de la Base de Données dans le langage de description de données d'un SGBD,

5) génération des programmes : en Cobol ou Fortran à partir des programmes abstraits.

(4) et (5) sont en cours de développement.

G. Conclusion

En résumé c'est la méthode la plus orientée utilisateur parmi celles que nous avons étudiées.

Ses points forts sont :

- une description statique complète du SI au niveau conceptuel,
- une notation graphique du modèle de données qui facilite l'obtention d'un consensus analyste/utilisateur,
- beaucoup d'aides.

Ses points faibles sont :

- une description dynamique uniquement réalisée au moyen du concept de dérivation,
- un langage SYSDUL très procédural et donc qui ne permet pas une compréhension facile par l'utilisateur non informaticien.

2.4.3. La méthode SDLA

A. Présentation

SDLA (System Description and Logical Analyzer) est une méthode développée par KNUTH et son équipe à Budapest en Hongrie. Cette méthode est proche, par certains aspects, du projet ISDOS (développé à l'université de Michigan aux USA). Elle a, en effet, été mise en oeuvre en parallèle et en coopération avec l'équipe de TEICHROEW puis en coopération avec F. BODART [TEI 77, KNU 80, TEI 79]. Elle couvre les activités d'analyse du système réel et permet l'obtention d'une solution qui est suffisamment complète pour être quasiment opérationnelle.

Objectifs : proposer une méthode et des outils de conception assistée indépendants du langage d'application.

B. Niveau d'abstraction

SDLA est une méthode qui ne couvre pas toutes les étapes du développement d'un SI mais se limite au niveau de la spécification indépendamment même de tout langage particulier d'application. Elle recouvre le cycle d'abstraction et partiellement de contrôle. De par ses objectifs, elle est située à un "haut niveau d'abstraction".

C. Les concepts

Le modèle présenté est orienté "donnée" ; tous les composants pouvant être modélisés comme des objets ayant des attributs.

i) Statique

Le concept principal est dérivé de la classe de SIMULA [DAH 70].

Il se nomme

- concept : il représente une "entité" une "association" ou un "attribut" au sens de CHEN [CHE 76]. Il est caractérisé par un ensemble d'attributs.

Dans la base de données sont stockés les :

- objets : ce sont les occurrences des concepts. Un objet est décrit par ses valeurs d'attributs.

Un ensemble d'objets, occurrences d'un concept abstrait donné, peut être vu comme une

- relation : sous-ensemble du produit cartésien des domaines de valeurs des attributs.

Enfin à chaque concept (ou type) abstrait peuvent être associés des

- sous-types : inverses de la notion de généricité au sens de [SMI 77], ils peuvent partager des caractéristiques du type mais également être enrichis par de nouvelles propriétés. Il s'agit ici de la notion de dérivation d'un type.

Exemples : CONCEPT process ; [définition du type processus]

CONCEPT manual-process is process ; } [définition de sous-types]
CONCEPT computerized-process is process ; }

CONCEPT file (opening : procedure, blocked : boolean)

[définition du type fichier avec deux attributs l'un du type procédure, l'autre du type booleen]

ii) Dynamique

La dynamique des données est exprimée à l'aide de deux notions :

- la dérivation des données : qui permet de déduire l'algorithme du calcul des résultats en fonction des données,

Exemples : PROCESS p ;
USES d TO-DERIVE e ;
DATA d ;
USED-BY p TO-DERIVE e ;
DATA e ;
DERIVED-BY p USING d ;

Ces trois définitions disent la même chose : le traitement "p" utilise la donnée "d" pour calcul "e".

- la fonction et l'action : qui permettent d'exprimer les transactions sur le système d'information.

L'action recouvre la spécification de toutes les procédures qu'elles soient manuelles ou automatisées, la fonction explicite les procédures automatisées au niveau logique.

D. Utilisation des types abstraits

Outre la notion de sous-typage et de dérivation proche de celle développée pour la méthode SYSDOC, il existe dans SDLA une modélisation beaucoup plus nette à l'aide des TA au niveau de l'étape de spécification.

Exemple : description d'un module

CONCEPT modul ;
CONCEPT data (part-of : data) ;
CONCEPT condition (associated-data : data) ;
CONCEPT précondition (to-activate : modul, when : condition) ;
CONCEPT post-condition (termination-of : modul, resulting : condition) ;
CONCEPT invariant (condition, associated : modul) ;

On retrouve ici les termes génériques de description d'un TA en pré/post et invariants. Cependant il faut remarquer d'une part qu'il y a dissociation de la spécification des données et de leurs opérations, d'autre part qu'à l'étape logique, on ne voit plus trace de ces définitions, le système étant alors décrit en termes de fonctions paramétrées.

E. Le langage

Au niveau de la spécification des données et de la dynamique une certaine liberté est donnée au concepteur. A chaque concept on peut attacher un nombre fixé de "formes" permettant la description du SI :

Exemple : pour expliciter la dérivation des données

```
CONCEPT data-dérivation (process, used : data, derived: data)
FORM used : USED-BY process TO-DERIVE derived ;
FORM derived : DERIVED-BY process USING used ;
FORM process : USES used TO-DERIVE derived ;
```

Ce qui correspond bien à l'exemple donné dans le paragraphe C sur la dérivation de "e" à partir de "d" dans "p".

Une fois ce cadre précisé, chaque concept est décrit avec un langage formel de haut niveau permettant la définition des modules (pre/post et invariants), des actions au niveau "description de l'environnement" automatisées ou non et des fonctions automatisées.

Exemples :

1) description de l'activité "enregistrement d'une inscription"

```
MANUAL TECHNICAL
MADE BY conference-secretariat
USING conference-mail-received !
UTILIZING conference-supporting-system !
ACTION.
    UPDATE participant-list ;
```

2) description de l'activité "établissement des documents finals"

```
MANUAL TECHNICAL
ACTION ;
PRINT participant-list,
      program-table,
      general-information,
BY conference-supporting-system ;
```

3) description de la fonction "enregistrement des papiers"

```
FUNCTION registrate-papers ;
        RECEIVES submission-head-information
        UPDATES submission-set ;
```

Il faut remarquer que dans l'ensemble développé dans CRIS 1, il s'agit de descriptions statiques de phases d'activité, de structure de données et de traitements de mise à jour, ainsi aucune contrainte d'intégrité, aucun calcul, à proprement parler, ne sont exprimés. Il est donc difficile de savoir si le langage proposé est prédicatif, déclaratif ou procédural.

F. Les outils

Il en existe de deux types. Ce sont :

- des outils traditionnels de documentation qui sont particulièrement intéressants car ils produisent des rapports formatés utilisant le langage défini par l'utilisateur,
- des outils de contrôle de type qui s'appuient sur les concepts abstraits définis.

G. Conclusion

En résumé SDLA est un outil (ou ensemble d'outils) plutôt qu'une véritable méthode complète de conception/réalisation de SI.

Ses points forts sont :

- la modélisation permise à l'aide des types abstraits,
- la possibilité qu'a l'utilisateur d'intervenir dans la forme du langage qu'il utilisera.
- beaucoup d'outils d'aide.

Ses points faibles sont :

- une description dynamique réalisée, au niveau conceptuel, essentiellement au moyen du concept de dérivation, complétée au niveau logique par une description de fonctions réalisant les traitements,
- pas d'outil ni de représentation graphique aidant aux discussions entre analyste et utilisateur,
- il paraît peu réaliste d'assurer que l'on peut implémenter le système à partir de la spécification obtenue.

2.4.4. La méthode USE

A. Présentation

La méthode USE (User Software Engineering) est développée à l'université de San Francisco aux USA par l'équipe de WASSERMAN. Les propositions d'origine [WAS 78, WAS 81] concernaient une méthode, appuyée sur un ensemble d'outils d'aide à la conception, et un langage (PLAIN) orienté manipulation et développé autour d'un noyau PASCAL [WIR 76]. Récemment de nouveaux travaux ont été développés, en particulier par N. LEVESON [LEV 80], très nettement orientés Types Abstraits ; ils ont trouvé leur aboutissement dans l'introduction d'une étape nouvelle de spécification en amont des étapes de conception/réalisation à l'aide de USE et PLAIN. L'approche USE commence par l'analyse des objectifs et se termine lorsque le système est implémenté.
Elle est actuellement en cours d'exploitation et d'évolution.

Objectifs : créer une méthode pour permettre la spécification, la conception et l'implémentation de SI interactifs. Trois objectifs principaux :

- développer un prototype de SI et supporter de cette façon la détermination de la cohérence du système au travers du cycle de développement,
- assister les évolutions possibles des systèmes en proposant une documentation du système et un logiciel bien structuré,
- aider à la compréhension du problème par les développeurs, les utilisateurs et toutes les parties concernées pendant la phase d'analyse et de spécification.

Bien que le papier publié dans les actes de CRIS 1 concerne l'ensemble du processus, nous nous contenterons ici de développer la partie spécification par les Types Abstraits.

B. Niveaux d'abstraction

USE est une méthode qui comprend les trois cycles définis dans [BOD 83] : les cycles d'abstraction, de contrôle et de décision. De par sa complétude et les concepts sous-jacents, la méthode USE peut être située à "un haut niveau d'abstraction".

C. Les concepts

En fait, à la lecture des différents papiers exposant la méthode USE, n'apparaissent pas en clair les concepts sous-jacents que ce soit pour la modélisation de la structure ou pour celle de la dynamique de la réalité. On donne les résultats à obtenir à la fin de chaque étape du développement d'un SI en explicitant le langage et les outils à utiliser, la démarche conseillée mais pas, explicitement du moins, le modèle associé. En schématisant, d'après ce que nous avons étudié, USE préconise le modèle relationnel pour les données et les diagrammes de transition pour la dynamique. Les notions d'abstraction de données par les Types Abstraits sont utilisées pour définir, lors de l'étape de spécification, les objets abstraits et les opérations qui leur sont associées.

i) Objets abstraits : il s'agit d'entités, au sens traditionnel du terme, (ex : une "communication") sur lesquelles on définit, au sens des TA, un ensemble d'opérations. Pour chaque opération définie formellement sur un objet abstrait, la démarche de développement formel BASIS inclut une spécification d'intégrité sémantique consistant en trois parties :

- l'image abstraite
- l'invariant
- les contraintes d'entrée et de sortie sous forme de pré et post-conditions. Tout objet sera associé ensuite à une relation en PLAIN.

2.4.4. La méthode USE

A. Présentation

La méthode USE (User Software Engineering) est développée à l'université de San Francisco aux USA par l'équipe de WASSERMAN. Les propositions d'origine [WAS 78, WAS 81] concernaient une méthode, appuyée sur un ensemble d'outils d'aide à la conception, et un langage (PLAIN) orienté manipulation et développé autour d'un noyau PASCAL [WIR 76]. Récemment de nouveaux travaux ont été développés, en particulier par N. LEVESON [LEV 80], très nettement orientés Types Abstraits ; ils ont trouvé leur aboutissement dans l'introduction d'une étape nouvelle de spécification en amont des étapes de conception/réalisation à l'aide de USE et PLAIN. L'approche USE commence par l'analyse des objectifs et se termine lorsque le système est implémenté.
Elle est actuellement en cours d'exploitation et d'évolution.

Objectifs : créer une méthode pour permettre la spécification, la conception et l'implémentation de SI interactifs. Trois objectifs principaux :

- développer un prototype de SI et supporter de cette façon la détermination de la cohérence du système au travers du cycle de développement,
- assister les évolutions possibles des systèmes en proposant une documentation du système et un logiciel bien structuré,
- aider à la compréhension du problème par les développeurs, les utilisateurs et toutes les parties concernées pendant la phase d'analyse et de spécification.

Bien que le papier publié dans les actes de CRIS 1 concerne l'ensemble du processus, nous nous contenterons ici de développer la partie spécification par les Types Abstraits.

B. Niveaux d'abstraction

USE est une méthode qui comprend les trois cycles définis dans [BOD 83] : les cycles d'abstraction, de contrôle et de décision. De par sa complétude et les concepts sous-jacents, la méthode USE peut être située à "un haut niveau d'abstraction".

C. Les concepts

En fait, à la lecture des différents papiers exposant la méthode USE, n'apparaissent pas en clair les concepts sous-jacents que ce soit pour la modélisation de la structure ou pour celle de la dynamique de la réalité. On donne les résultats à obtenir à la fin de chaque étape du développement d'un SI en explicitant le langage et les outils à utiliser, la démarche conseillée mais pas, explicitement du moins, le modèle associé. En schématisant, d'après ce que nous avons étudié, USE préconise le modèle relationnel pour les données et les diagrammes de transition pour la dynamique. Les notions d'abstraction de données par les Types Abstraits sont utilisées pour définir, lors de l'étape de spécification, les objets abstraits et les opérations qui leur sont associées.

i) Objets abstraits : il s'agit d'entités, au sens traditionnel du terme, (ex : une "communication") sur lesquelles on définit, au sens des TA, un ensemble d'opérations. Pour chaque opération définie formellement sur un objet abstrait, la démarche de développement formel BASIS inclut une spécification d'intégrité sémantique consistant en trois parties :

- l'image abstraite
- l'invariant
- les contraintes d'entrée et de sortie sous forme de pré et post-conditions. Tout objet sera associé ensuite à une relation en PLAIN.

ii) Pré-conditions, post-conditions et invariants : ils permettent d'introduire les contraintes d'intégrité sur la Base de Données, c'est-à-dire des contraintes de domaine ou des contraintes inter-relations. Elles seront implémentées ultérieurement par des assertions en PLAIN.

iii) Transition : chaque opération doit être associée à tout ou partie d'une action dans un diagramme de transition. Un diagramme de transition est un réseau de noeuds et de chemins directs, chaque chemin peut contenir un jeton correspondant à une chaîne de caractères ou à un nom d'un autre diagramme. S'il est à blanc il sera traversé comme une case normale. Ces diagrammes sont utilisés pour modéliser le dialogue utilisateur/machine.
Exemple :

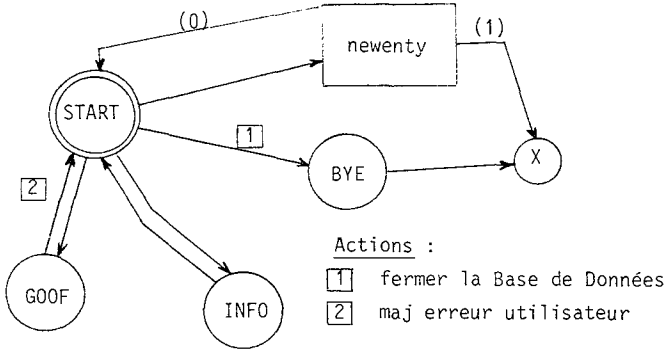


Diagramme de transition de commandes initiales du système

Le diagramme est initialement en START (message : "SVP tapez une commande) ; on lit le message si "Quit" alors on va en BYE (message : "Au revoir") et on effectue l'action 1 ; si "Help" alors on va en INFO (message : "Les commandes valides sont ...") puis retour en START ; si "enter" on va au diagramme "Newenty" qui retourne les codes 0 et 1, déterminant le chemin suivi et l'action à réaliser après ; en cas d'erreur on va en GOOF (message "commande illégale") et on effectue l'action 2.

Les concepteurs de USE ne proposent, en fait, rien de nouveau ici mais renvoient aux travaux de SMITH et SMITH [SMI 77] sur le modèle SHM (étendu par ailleurs dans la méthode ACM/PCM [BRO 82]) ainsi qu'à la méthode SSA (Structured Systems Analysis) [GAN 79, MAR 79] sur les data-stores (repris par SOLVBERG dans la méthode DAISEE [SOL 82])

iv) Structure de la spécification complète : une spécification d'un SI interactif consiste en :

. (1) un ensemble de diagrammes de transition représentant le dialogue homme/machine en entier avec la description des actions ;

. (2) une Base de Données spécifiée comme un ensemble de relations normalisées, déduites des objets abstraits ;

. (3) des opérations avec, pour chaque opération liée au diagramme, son nom d'action, sa description formelle et informelle.

D. Utilisation des Types Abstraits

Les nouveaux travaux sont orientés Types Abstraits. La méthode de spécification formelle est basée sur les travaux de SMITH et SMITH [SMI 77] et sur l'approche axiomatique développée pour ALPHARD [WUL 76] et étendue aux SI par N. LEVESON [LEV 80]. Elle permet le renforcement des contraintes d'intégrité sémantiques sur la Base de Données et la description de la dynamique du système. La méthode conduit à spécifier les conditions pré/post de chaque opération.

E. Le langage

Il exite en fait plusieurs langages

- un langage graphique et les commandes associées pour spécifier le dialogue utilisateur/machine, qui sert d'entrée à l'interprète de diagramme de transition (TDI),
- un langage d'expression des relations normalisées de la Base de Données associée au SGBD TROLL,
- un langage de spécification des objets abstraits en termes de pré et post-conditions, les traitements étant exprimés à l'aide d'un langage déclaratif basé sur le calcul de prédicat du premier ordre,
- le langage PLAIN, au niveau implémentation ; langage proche de PASCAL relationnel, il est le langage effectivement utilisé dans l'environnement logiciel.

Il ne paraît pas y avoir, tout au moins actuellement, de mapping automatique entre ces deux derniers niveaux, donc ces deux langages.

Exemples :

1) Description de la relation "papers"

```
RELATION papers [KEY paperno] of
   paperno : paperrange ;
   title : string ;
   resp-pc-member : person ;
   status : paperstatus
END ;
```

Les domaines des attributs sont définis par ailleurs.

2) Définition de l'objet abstrait "paper"

```
OBJECT paper
ABSTRACT IMAGE
      title : text
      authors : SET OF author
      status : paperstatus
      paperno : 1..900
```

```
ABSTRACT INVARIANT
OPERATIONS
. receive-paper (papertitle : text, submitters : SET OF author)
      RETURNS p : paper
POST title = papertitle
     authors = submitters
     status = received
     paperno = cardinality (paper)
. review-paper (p : paper)
PRE p.status = received
POST p.status = in-review
      .....
(toute autre opération telle que l'acceptation, le rejet ...)
```

3) Définition de l'affectation d'un papier à un referee

```
OPERATION select-refs
NAME     Assign-referee (p : paper)
INFORMAL
(définition informelle de l'opération)
FORMAL
   PRE p.count refs >= 0 & p.count refs < MAXREFS
      & r.number-assigned < MAXPAPERS
   POST is-referee (r,p) & p.count refs' = p.count refs + 1
      & date-sent(r,p) = [today's date] & ~(r.name IN
      p.authors)
```

4) Définition partielle de la procédure PLAIN pour la même opération

```
IF ~exist(referee-list [$0]) THEN
      BEGIN print "unknow referee name" ; END
      ELSE {valid referee nam }
      BEGIN referee.list [$0].number-assignated :=
                     referee.list [$0].number-assignated + 1
            INSERT reviewing [$0, $1, $2] ;
      END ;
```

où $0 est le paramètre formel pour le nom du referee, $1 pour le n° de papier, $2 pour le jour et "reviewing" est la relation où on insère un nuple.

F. Les outils

Au niveau de la spécification à l'aide des TA, aucun environnement logiciel d'aide ou de contrôle ne paraît, actuellement, prévu. En revanche de nombreux outils sont proposés lors du développement effectif du SI :

- TDI (Transition Diagram Interpreter) outil destiné à décrire les diagrammes de transition,
- TROLL, outil qui propose un interface de type algèbre relationnelle pour un petit système de gestion de Bases de Données relationnelles, utilisé lors de l'exécution de programmes PLAIN,
- RAPID (RApid Prototypes of Interactive Dialogues), outil de construction rapide de systèmes partiels,
- USE : environnement logiciel de la méthode.

G. Conclusion

En résumé il s'agit d'une méthode de développement complète pour la conception/réalisation de SI qui couvre toutes les activités.

Ses points forts sont :

- la modélisation permise à l'aide des TA,
- l'orientation utilisateur qui intervient lors de la conception des dialogues,
- les nombreux outils d'aide, en particulier ceux de simulation et de construction de prototypes.

Ses points faibles sont :

- un modèle conceptuel plus implicite qu'explicite,
- un manque d'enchaînement entre la phase de spécification avec BASIS et la phase de réalisation avec USE et PLAIN. Un peu comme si ces deux phases apparaissaient juxtaposées et non-inter-dépendantes,
- un environnement logiciel coûteux pour le développement de gros SI.

2.4.5. La méthode ACM/PCM

A. Présentation

La méthode ACM/PCM (Active and Passive Component Modeling) est développée à l'Université de Maryland aux USA par l'équipe de M. BRODIE.

Elle est basée à la fois sur les travaux effectués en Bases de Données et sur les langages de programmation. Elle reprend en particulier les propositions de SMITH et SMITH [SMI 77] ; on y retrouve les différentes formes d'abstractions introduites dans la première partie de ce chapitre. Cette méthode préconise une modélisation en parallèle des données et des transactions.

Elle est actuellement en cours d'exploitation et de développement dans les entreprises.

Objectifs : proposer des moyens pour la conception et le développement de SI complexes, de grande taille, orientés objets, qui permettent l'usage de transactions sur des Bases de Données interactives ; en particulier gérer leur complexité, leur définition et leur intégrité sémantique.

B. Niveau d'abstraction

De toutes les méthodes présentées ici et parmi les quinze retenues dans CRIS 1, ACM/PCM est celle qui s'appuie le plus sur le principe d'abstraction. Elle couvre le cycle d'abstraction et en partie le cycle de contrôle. De par ses objectifs et ses concepts sous-jacents ACM/PCM peut être située à un "haut niveau" d'abstraction.

C. Concepts

C.1. Modélisation de la structure

ACM/PCM propose une extension du modèle SHM [SMI 77] appelée SHM$^+$ (modèle hiérarchique sémantique étendu) ; il repose sur le concept structurel d'"objet" et sur quatre formes d'abstractions

structurelles : les formes proposées dans [SMI 77] [l'agrégation et la généralisation] ainsi que la classification et l'association.

i) Classification : forme d'abstraction dans laquelle une collection d'objets est considérée comme une classe d'objets de haut niveau, regroupant tous les objets possédant les mêmes caractéristiques.
C'est en fait la notion de classe, type, population habituellement rencontrée en conception de Bases de Données.
Exemple : classe d'objets EMPLOYE
Cette forme d'abtraction induit la notion d'instanciation d'un objet dans une classe, aussi introduit-on la relation "occurrence-de".

ii) agrégation : forme d'abtraction où une relation entre des objets composants est considérée comme un objet agrégé de haut niveau ; cette forme permet d'exprimer les propriétés des objets d'une classe.
Exemple : un "employé" est un objet agrégé de n° employé, nomemployé, salaire ...
Cette forme d'abtraction induit la relation "partie-de".

iii) généralisation : forme d'abstraction où une relation entre des objets d'une catégorie est considérée comme un objet générique de haut niveau ; cette forme permet d'exprimer des relations hiérarchiques entre les objets.
Exemple : un "employé" est générique de "secrétaire".
Cette forme d'abstraction induit la spécialisation, d'où la relation "est-un" : une secrétaire "est-une" employée.

iv) association : forme d'abstraction où une relation entre des objets membres est considérée comme un objet d'ensemble de haut niveau ; cette forme permet de définir des sous-ensembles dans une classe d'objets.
Exemple : l'ensemble "syndicat" est une association de membres "employés".
Elle induit la relation "membre-de".

Pour modéliser la structure on utilise des techniques de composition/ décomposition et généralisation/spécialisation. L'application répétée de ces processus à l'aide des quatre concepts précédents aboutit à la construction du modèle conceptuel de l'application.

C.2. Modélisation de la dynamique

SHM+ propose des concepts pour la modélisation de la dynamique en accord avec la modélisation de la structure. Les concepts primitifs sont en fait dérivés des propositions issues de la recherche sur les langages de programmation. Chaque transaction sur la Base de Données est définie en termes d'actions sur les objets. Pour chacune la structure d'information est décrite dans un schéma d'objet pour obtenir le schéma de la dynamique. Les contraintes dynamiques sont imposées au moyen de pré et de post-conditions sur les actions et les transactions.
SHM+ repose sur la définition d'opérations primitives sur les objets de la Base de Données et de trois formes d'abstractions de contrôle avec lesquelles on compose des opérations orientées application.

i) Opérations primitives : chaque opération sur la Base de Données est une opération de modification ou d'interrogation qui ne porte que sur une seule occurrence d'une classe d'objet. Les opérations permises sont :

- en modification : INSERT, DELETE, UPDATE
- en recherche : FIND, CREATE, REQUEST.

ii) Abstractions de contrôle : elles sont utilisées pour relier des opérations, pour former des opérations de haut niveau. Les trois formes (séquence, choix, répétition) sont en correspondance avec les concepts de la modélisation de la structure.

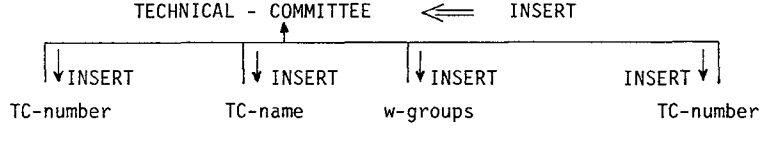
α) agrégation → séquence : une opération sur un agrégat est composée d'une séquence d'opération, une sur chaque composant.
Exemple : créer un employé revient à valoriser son n°, son nom ...

β) généralisation → choix : avec deux types de constructions, if then else et case : une opération sur un générique est composée d'une opération appliquée à la catégorie générique.
Exemple : créer un employé de type secrétaire (opération sélective).

γ) association → répétition : avec deux types de constructions, do while et foreach : une opération sur un ensemble est appliquée sur chaque membre.
Exemple : constituer un "syndicat" et éditer les appels aux cotisations.

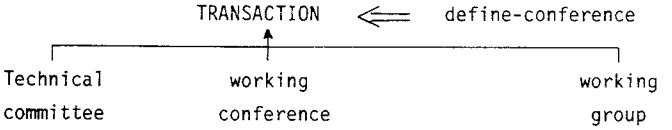
iii) Modélisation des transactions : on adjoint au modèle précédent deux formes d'abstractions procédurales destinées à la modélisation des transactions sur la Base de Données.

α) action : c'est une opération orientée application conçue pour un objet afin de s'assurer que toutes les propriétés de l'objet sont satisfaites. Avant d'effectuer l'action certaines pré-conditions doivent être remplies, après avoir réalisé l'action une post-condition doit être vérifiée et l'opération est exécutée. Cette forme d'abstraction permet de faire la différence entre l'objet considéré altéré par l'opération et les autres atteints uniquement par des actions. La dynamique d'un objet est complètement définie par ses actions.
Exemple : schéma d'action "insérer" le comité-technique : pour faire une telle insertion, il faut insérer les constituants clés et les constituants essentiels tc-members et w-groups qui sont des objets associés. Schématiquement :



β) transaction : c'est le seul moyen pour les utilisateurs finals d'agir sur la Base de Données ; elle provoque une action spécifiée par des pré/post conditions mais ne contient pas d'opération dans sa spécification ; l'action, elle, induit des opérations sur la Base de Données.

Exemple : schéma de transaction de définition du congrès : insérer un objet "conférence" avec les groupes impliqués dans l'organisation du congrès. Schématiquement :



D. Utilisation des types abstraits

Comme le montre le développement précédent, ACM/PCM est la méthode qui utilise le plus largement le vocabulaire "Types Abstraits" dans ses concepts de modélisation structurelle et dynamique. Elle représente ainsi un précurseur dans le domaine des méthodes de conception de SI.

E. Les langages

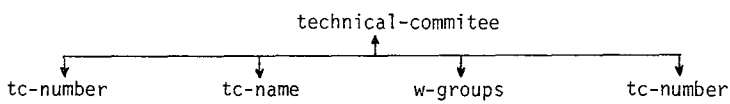
Deux sortes de langages :

- un langage graphique complet qui permet la modélisation de la structure et de la dynamique. Il est très aisé à comprendre et assure un lien de communication entre utilisateurs et concepteurs de SI,
- le langage de spécification BETA qui permet la description des schémas d'objets et des transactions, il est :
  - . de haut niveau (les détails de procédure ne sont pas exprimés)
  - . déclaratif et prédicatif (non procédural)
  - . basé sur une spécification pré/post des opérations sur les TA.

Remarquons que, dans l'exemple de la gestion d'un congrès donné dans [BRO 82], aucun calcul n'est exprimé. Seules sont explicitées des opérations de mise à jour de la Base de Données et la validité de ses états ; on ne peut donc en déduire la non-procéduralité de l'expression des traitements.

Exemples :

1) représentation graphique d'un agrégat



2) description d'un committee-technique en BETA

```
TECHNICAL-COMMITTEE = OBJECT
AGGREGATE OF
     tc-numbers : ESSENTIAL
     tc-number ;
     tc-name ;
     w-groups : ESSENTIAL ;
KEY tc-name ;
PRIMARY KEY tc-number
END OBJECT ;
```

3) définition d'une contrainte d'intégrité : tout comité technique a au moins un tc-number.

```
ALL tc IN technical-committee (SOME tcms IN tc-number  (tcms PART
OF tc))
```

4) définition d'une action : insérer un papier

```
ACTION insert-paper (pn, title)
IN (a : authorship, title : paper-title, d : registrationdate,
pn : paper-number)
OUT (p : paper)
```

```
LOCAL (sp : submitted-papers)
PRE-CONDITION :
     no-paper (pn,title) ?
     submitted - papers - exist (pn) ?
POST-CONDITION : authors-exist (pn) ?
DB-OPERATION : INSERT paper (title, pn, d) ;
```

F. Les outils

Le papier présenté dans CRIS 1 ne parle pas de cet aspect. Dans [POR 83] des outils de vérification formelle partielle de la spécification ainsi que de test du prototype obtenu sont dits "existant" dans la méthode mais M. BRODIE, lui-même, dans CRIS 2, dit qu'aucun logiciel automatisé n'est actuellement opérationnel.

G. Conclusion

En résumé il s'agit d'une méthode concentrée sur l'activité de spécification.

Ses points forts sont :

- une modélisation précise et détaillée à la fois des données et des traitements du SI, basée sur les TA,
- un langage graphique très développé,
- un langage de spécification compréhensible et complet.

Ses points faibles sont :

- l'absence d'expression autre que graphique des concepts d'événement ou de trigger,
- le défaut d'environnement logiciel,
- une spécification obtenue encore bien loin de l'implémentation du SI.

2.4.6. La méthode REMORA

A. Présentation

REMORA est une méthode développée au CRIN à Nancy depuis 1974, à Paris I depuis 1978, par l'équipe de C. ROLLAND. Son origine tient à la notion de "pilotage" du processus de conception/réalisation d'un SI (le "remora" est considéré dans la légende comme le poisson "pilote" du requin) c'est-à-dire que l'idée fondamentale de ce projet est de proposer un environnement logiciel d'aide à la conception de SI. Cette méthode est en cours de développement et d'exploitation dans l'industrie.

Objectifs : proposer des modèles, langages, outils aux niveaux conceptuel et logique du processus de conception d'un SI.

B. Niveau d'abstraction

REMORA ne couvre pas l'intégralité du cycle de vie d'un SI, ses propositions concernent en priorité le niveau "conceptuel" ou "abstrait" de conception du SI et également le niveau "logique". Cette méthode couvre principalement les cycles d'abstraction et de contrôle.

De par ses objectifs et les concepts sous-jacents REMORA peut être située à un "haut niveau" de formalisme d'abstraction. Nous nous limiterons ici aux propositions concernant le niveau le plus abstrait c'est-à-dire le niveau conceptuel.

C. Concepts

Le modèle conceptuel de REMORA repose sur une description causale de la dynamique d'un système.

L'état du système est défini par celui de ses éléments appelés objets, les changements d'état sont la conséquence de l'exécution d'actions, appelées opérations déclenchées par des stimuli externes ou internes, appelés événements.

Les objets, les opérations et les événements sont structurés en classes. Les règles de gestion du sous-système causal d'un SI peuvent s'exprimer par des relations entre ces trois types de classes.

Le modèle REMORA propose, de plus, un mode de structuration particulier des trois types de phénomènes (objet, opération, événement) qui conduit à une structuration en classes élémentaires, reposant sur une intégration du temps qui permet de construire des SI historiques.

Ces trois classes élémentaires sont, en fait, les trois concepts de base du système.

(i) CLASSE-OB

Sémantique : représentation, au cours du temps, d'une classe d'objets ou d'une classe d'associations entre objets ;

composition : ensemble de sous-schémas de classes, appelés c-objets, représentant chacun un aspect temporel de la classe. Il existe trois types de c-objets :

- c-objet permanent,
- c-objet variable,
- c-objet suppression.

(ii) CLASSE-OP

sémantique : représentation, au cours du temps, d'une classe d'opérations élémentaires ; soit telle que chaque opération de la classe modifie l'état d'un unique c-objet.

composition : ensemble de c-opérations.

(iii) CLASSE-EV

sémantique : représentation, au cours du temps d'une classe d'événements élémentaires ; soit telle qu'un événement de la classe constate le changement d'état d'un unique c-objet ; en revanche un événement de la classe peut déclencher (sous condition ou de

manière itérative éventuellement) plusieurs opérations appartenant à différentes classes d'opérations. Le changement d'état est défini par un état initial et un état final du SI exprimés par un prédicat.

composition : ensemble de c-événements

Une première formalisation de ces concepts a été effectuée en utilisant le formalisme du modèle relationnel de CODD. Cette formalisation a nécessité l'introduction dans le modèle relationnel de concepts supplémentaires :

- type de relation,
- type de constituant,
- dépendance fonctionnelle permanente dont la définition est : une dépendance fonctionnelle permanente entre deux attributs A et B est une dépendance élémentaire, directe et canonique où $\forall a \in A$ et $b \in B$ dépendant fonctionnellement de a, a et b ont la même durée de vie,
- relation permanente : relation 3FN [COD 70] où chaque attribut est en dépendance fonctionnelle permanente avec l'identifiant de la relation.

A l'aide de cette formalisation un Schéma Conceptuel de SI se présente comme une collection de schémas de relation appartenant à différents types de relation, correspondant aux différents types possibles de c-objets, c-opérations et c-événements.

Exemples

a) classe d'objets

La classe "proposition de conférence" sera représentée par deux c-objets

- proposition (nprop, nconf, receptpropdate, title, abstract) regroupant ses aspects permanents, et

- prop-result (nprop, nconf, resultdate, genmark, resultstate) représentant l'évolution de la vie d'une proposition de conférence.

b) classe d'événements

Représentation de la classe "arrivée de proposition de conférence"

(1) arrivée-proposition (nev5, datev5, npropl) avec
npropl = nprop, nconf

(2) ev-proposition-perm (nev5, predicatev5)

(3) declenche-refus (nev5, nopl, nonc3)

où (1) décrit les arrivées effectives des propositions de conférence,

(2) décrit les aspects permanents du changement d'état,

(3) décrit le déclenchement de l'opération de refus (identifiée par nopl) par un événement de la classe (nev5) si la condition nonc3 est remplie (la proposition de conférence n'est pas arrivée dans les délais).

c) classe d'opérations

Représentation de la classe "analyse de proposition de conférence"

(1) prop-conf-perm (nopl, typecre)

(2) prop-conf-regle (nopl, datextopl, textopl)

(3) prop-conf-execution (nopl, datexecopl, nrefusl)

car il y a un seul type de modification possible sur l'objet créé (1) par l'opération d'analyse de la proposition de conférence, plusieurs textes d'opération (2) au cours du temps, plusieurs exécutions (3) de l'opération au cours du temps.

Remarque : Dans la formalisation du second chapitre de cette partie, CLASSE-OB, classe d'objets, sera explicité par le type CLASSE-OB, CLASSE-EV, classe d'événements, par le type CEVENT ("C" pour classe), CLASSE-OP, classe d'opérations, par le type COPERATION.

Nous emploierons indifféremment les termes de c-objets ou sous-schémas, et préférons parler de type d'opération et de type d'événement plutôt que de c-opération et c-événement.

D. Utilisation des Types Abstraits

Si l'on peut affirmer que, grâce au modèle conceptuel, REMORA permet une modélisation abstraite de qualité, il faut reconnaître que, dans les développements proposés lors de CRIS 1 [ROL 82], la méthode ne préconise aucune utilisation des Types Abstraits. En fait c'est l'objectif de nos travaux actuels que d'introduire la modélisation par les Types Abstraits en gardant comme base les concepts du modèle, afin de proposer une formalisation de SI intégrant la richesse des Types Abstraits et les qualités du modèle conceptuel.

E. Les langages

Il y a deux sortes de langage :

- un langage graphique permettant la description de la statique et de la dynamique d'un SI,

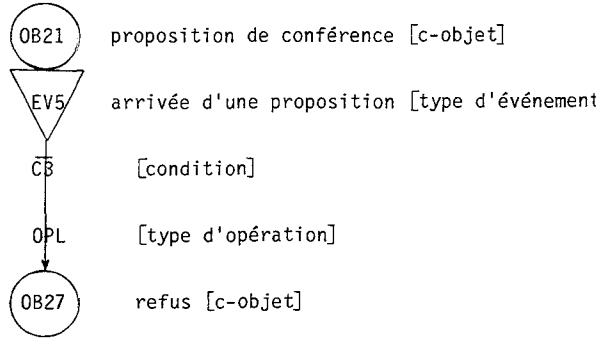
- le langage ISDEL qui permet une spécification complète de la modélisation conceptuelle. C'est un langage à la fois de programmation et de description de Données adapté à la spécification

de SI. Lié au modèle conceptuel de REMORA, il possède des caractéristiques de type relationnel comme en particulier la définition du type relation, les opérations de base du calcul relationnel et la structure FOREACH qui permet de parcourir les occurrences d'une relation. Pour décrire les algorithmes de calcul, ISDEL intègre des caractéristiques procédurales autour d'un noyau PASCAL [WIR 71] : par ce biais il est proche de PLAIN [WAS 82A] ou de Pascal Relationnel [SCH 77].

Enfin pour décrire les contraintes d'intégrité et les accès aux données, il possède un formalisme prédicatif permettant l'énoncé d'assertions.

Exemples :

1) représentation graphique de la dynamique de l'arrivée d'une proposition de conférence.



2) description d'accès aux données : donner tous les referees anglais de la conférence appelée "modèles formels et outils pratiques pour la conception de SI".

(x.nref) : referees <x> AND conference <y>
AND EXIST <y> (y.confdesig = 'modèles formels et outils pratiques pour la conception de SI'
AND EXIST <x> (x.nconf = y.nconf AND x.refcountry = 'ENGLAND'))

3) description d'un texte d'opération : évaluation des communications d'après les réponses des referees.

```
IS-TYPE COPERATION
    Texopn (nopn,datopn, textopn)
    DOMAIN nopn : SAME OF opnper ;
           datopn : DATE (8) ;
           textopn : CHAR (10) ;
    ASSERT astexopn-1 : nopn,  datopn, COMPOSKEY nopndatopn ;
           astexopn-2 : textopn TEXT (VAR factopn : RELATION fnconf :
                        INTEGER (5), fnprop : INTEGER (5) END ;)
           VAR propmark : RELATION pmprop : INTEGER (5) ; pmmark :
                          REAL (4,2) END ;
              statvar : CHAR (16) ;
              datvar : DATE (8) ;
              propaverage : REAL (4,2) ;
           BEGIN
             propmark : <(x.nprop, x.mark) referee-prop-response < x >
                        AND EXIST < x > (x.nconf = factopn.fnconf
                            AND x.nprop = factopn.fnprop) ;
              datvar : = TIME (8) ;
              IF COUNT (x.propmark) = 0 THEN
                 INSERT (factopn.fnconf, factopn.fnprop, datvar, '99.99',
                        'not reviewed') IN prop-result
                                        ELSE
```

```
        BEGIN propaverage := AVERAGE ((x.pmmark) propmark < x > ;
              IF propaverage ⩾ 14 THEN statvar := 'accepted'
                                  ELSE statvar := 'to be discussed' ;
              INSERT (factopn.fnconf, factopn.fnprop, datvar,
                    p-average, statvar) IN prop-result ;
              END ;
END ;
```

F. Les outils

REMORA inclut :

. un système de conception assistée par ordinateur où le processus de conception du SI est contrôlé par un automate "le pilote" qui coordonne les interventions des hommes et des outils. Les outils remplissent quatre fonctions : contrôle, intégration, simulation et documentation. Ils utilisent une méta-description du schéma conceptuel stockée dans une Base de Données.

. un système de gestion de SI (SGSI) comportant un outil particulièrement intéressant : le processeur d'événements qui contrôle automatiquement la dynamique du SI en sélectionnant et exécutant la transaction liée à tout changement d'état reconnu comme un événement. L'architecture de cet outil est proche de celle définie dans [ISO 81] pour le processeur d'informations.

G. Conclusion

En résumé REMORA est une méthode qui s'attache principalement, à partir de l'analyse de besoins, à l'étape de spécification d'un SI.

Ses points forts sont :

- sa puissance de modélisation statique et dynamique au niveau conceptuel,

- la proposition de règles de mapping pour parvenir au niveau logique,
- ses outils, dont en particulier, le processeur d'événement original dans sa conception.

Ses points faibles sont :

- une certaine incomplétude par rapport au cycle de vie du SI, en particulier la gestion et le contrôle du cycle de vie du SI sont limités au processus de conception,
- un formalisme conceptuel, qui de l'avis de toutes les analyses de CRIS 2, est difficile à comprendre,
- la non-utilisation des TA comme outil de modélisation.

Espérons que la résolution du 3ème point faible induira la résolution du 2ème.

3. CONCLUSION

3.1. Comparaison des six méthodes

A. Présentation : origine et objectifs

Il faut noter que la plupart des méthodes sont d'origine universitaire. Seule NIAM est développée dans un laboratoire de recherche privé : en l'occurrence CONTROL DATA, encore faut-il dire que l'origine de ces recherches est universitaire. C'est en fait assez logique si l'on se souvient du cadre d'analyse que nous nous sommes fixé, en particulier, de l'importance que nous avons accordée à la spécification plutôt, par exemple, qu'à l'implantation du système.
Selon les méthodes la couverture du cycle de vie du SI est plus ou moins importante.

B. Niveau d'abstraction

Toutes les méthodes retenues se situent à un haut niveau d'abstraction, soit qu'elles se limitent explicitement à l'étape de spécification du SI (NIAM, ACM/PCM, SDLA), soit qu'elles fassent des propositions complémentaires plus proches du niveau de la réalisation du SI (SYSDOC, REMORA, USE) mais accordent plus d'importance à l'étape conceptuelle.

C. Concepts et Types Abstraits

Dans l'ensemble toutes les méthodes proposent une modélisation statique claire et bien introduite. Par contre, selon les méthodes, la modélisation dynamique est explicite et puissante (dans REMORA), plus ou moins bien introduite (dans USE, ACM/PCM), plutôt mal introduite (comme dans NIAM, SYSDOC, SDLA).
Les concepts apparaissent clairement sauf dans USE.
Enfin USE, dans ses derniers développements et ACM/PCM proposent une modélisation à l'aide des Types Abstraits.

D. Les langages

Toutes sauf SDLA proposent un langage graphique, particulièrement puissant dans le cas de NIAM et ACM/PCM. Toutes également proposent un langage de spécification, parfois trop procédural comme RIDL dans NIAM, SYSDUL dans SYSDOC, PLAIN dans USE et ISDEL dans REMORA, carrément déclaratif comme BETA dans ACM/PCM ou plus ou moins modifiable par le concepteur comme dans SDLA.

E. Les outils

Toutes sauf ACM/PCM, proposent un environnement logiciel d'aide à la conception du SI, permettant des contrôles, de la simulation et de la documentation.

3.2. Tableau récapitulatif

Le tableau de la page suivante est en fait un récapitulatif de toutes les réflexions précédentes, il doit conduire à une visualisation précise et complète des six méthodes présentées. Les critères de classification proposés sont d'une part les cinq points retenus dans le plan de développement de chaque méthode, d'autre part des critères plus qualitatifs (donc plus subjectifs) sur la modélisation ou les qualités de la spécification obtenue.

3.3. Vers de nouvelles propositions

Du développement précédent, nous pouvons tirer deux conclusions :

1) Le niveau d'abstraction où on se place doit être le plus haut possible, ou on ne peut pas concevoir correctement un SI si on ne dispose pas d'un modèle sous-jacent clair, net et précis. Ce modèle doit de plus intégrer les aspects statiques et dynamiques

critères / méthode	Concepts				Pourvoir de représentation du modèle				langages de spécification			Outils			Qualités de la spécification obtenue			
	explicites	statique	dynamique	TA	précision	complexité	complétude	historique	graphique	niveau nom	nature	contrôle	document.	autres	facilité à construire	compréhensibilité	cohérence	évolutivité
NIAM	oui	+++ . LOTS . NOLOTS ...	+ procédure	+ sous-typage	+++	+++	+ pas assez dynamique	-	puissant	RIDL haut niveau	. prédicatif pour les D . procédural pour les T	nombreux ISDIS		ENFORCER	++ (pas beaucoup de méthode)	+++	++ (outils de contrôle)	++ (outils d'intégration)
SYSDOC	oui	++ . entité association	+ . dérivation événement	+ . généralisation . classification	++	++	+ idem	+ concept d'événement	puissant (outils)	SYSDUL haut niveau	. prédicatif pour les CI . normes codasyl pour textes	nombreux SYSTEMATOR		mapping autom. conceptuel ↓ physique	+ (pas de méthode)	++ (modèle pas toujours clair)	++ (outils de contrôle)	++ (outils d'intégration)
SDLA	oui	++ . entité association	+ . dérivation . fonction	+ . sous typage . SIMULA	++	++	+ idem	-	aucun	haut niveau . orienté utilisateur	. descriptif structure . pas de traitement	outils SDLA			+++ (suivant la méthode)	- (résultat peu clair)	+++ (outils contrôle de type)	- pas d'outils
USE	non	++ . objet abstrait . relationnel	++ . transition (diagramme)	++ . SHM . ALPHA RD	+++	+++	++ dynamique plaquée	-	juste pour dialogues	. BASIS spécif. . PLAIN relation.	. déclaratif prédicatif . procédural	. pas au niveau spécification . outils à USE liés		. RAPID . TDI . TROLL	++ (pas beaucoup de méthode)	+++	- (manque au niveau spécification)	- d'outils
ACM/CPM	oui	+++ . SHM complété	++ . abstraction de contrôle . action/ . transaction	+++ . SHM+ . classif . génér. . agreg. . assoc.	+++	+++	++ dynamique plaquée	-	puissant	. BETA haut niveau	. déclaratif prédicatif pour les D . traitement en pré-post	. peu vérification formelle partielle	. aucun		++ (idem)	++ (difficile à saisir)	- (manque)	- d'outils)
REMORA	oui	+++ relationnel c-objet	+++ c-événem. C-opérat.	-	+++	+++	+++	+++ relation permanente	. statique . dynamique	ISDEL	. procédural	. nombreux CAO		SGSI	++ (pas beaucoup de méthode)	++ (modèle difficile à saisir)	++ (outils de contrôle et d'intégration)	++

de la réalité et être associé à un langage formel. IL faut noter que les méthodes retenues répondent en grande partie à cette remarque. REMORA, en particulier, grâce à son modèle conceptuel fondé sur les seuls concepts de classe d'objets, classe d'événements et classe d'opérations et pourtant permettant une représentation intégrée et complète du SI au cours du temps est certainement la méthode qui répond le mieux à cette proposition.

2) Le formalisme à utiliser au niveau de la spécification n'est pas quelconque : pour avoir le degré d'abstraction formelle voulu, la possibilité de contrôler la spécification, la richesse de représentation et de structuration nécessaire, la modélisation doit utiliser le formalisme des Types Abstraits. Le formalisme induit, tout naturellement, par ailleurs un langage de spécification de haut niveau. Les six méthodes retenues répondent à des degrés divers à cette deuxième remarque. REMORA, ainsi que nous allons le montrer dans le chapitre suivant, y répond dans ses nouveaux développements.

CHAPITRE 2

Une formalisation des Systèmes d'Informations fondée sur les Types Abstraits

1. NOTRE POINT DE VUE SUR LES TYPES ABSTRAITS

1.1. Introduction : vers des modèles plus formels

Des analyses ont montré [BUB 83] que, parmi la centaine de méthodes de conception de SI proposées dans le monde, si la plupart ont vu un maigre développement, ceci est dû, surtout, à leur manque de formalisme. Par ailleurs les méthodes les plus formelles et les plus répandues, actuellement, sont souvent fondées sur des modèles de données à base plus ou moins mathématique tels que le modèle entité-association ou relationnel. C'est le cas, en particulier, de IDA [BOD 84], MERISE [LE 84], NIAM [VER 82], CIAM [GUS 82] et REMORA [ROL 82].

Comme nous l'avons fait remarquer précédemment, le modèle relationnel, aussi bien d'ailleurs que le modèle entité-association, ne permettent pas une structuration assez riche de la réalité car ils induisent une modélisation trop "à plat". Ce sont les mécanismes d'abstraction qui permettent de capturer le mieux la sémantique du monde réel. Souvent, de plus, les méthodes proposant une telle modélisation relationnelle, ainsi que le note FURTADO dans [FUR 83], n'ont pas été associées à des modèles présentés de façon assez formelle et mathématique. L'introduction des Types Abstraits a permis de pallier un certain nombre de problèmes en conception de SI. Un panel du congrès VLDB 82 à MEXICO (voir [BRO 83B] pour un résumé) a montré le consensus des participants sur l'importance accordée actuellement aux Types Abstraits (TA) en conception des Bases de Données et maintenant en conception des SI.

1.2. De l'intérêt des TA en conception de SI

Rappelons que, de façon intuitive, un type abstrait permet la définition d'un ensemble d'objets par l'expression (axiomatique ou pré/post) des opérations servant à manipuler ces objets. Trois propriétés du concept de Type Abstrait nous ont paru particulièrement intéressantes :

. validation du modèle ;
. réutilisation des connaissances ;
. méthode de structuration à laquelle les TA conduisent.

1.2.1. Validation du modèle

Lors de la définition de SI complexes, même avec des modèles aussi formels et structurels que possible, c'est-à-dire permettant une description modulaire du SI et garantissant, si on respecte les concepts, un schéma conceptuel cohérent, on est amené à préciser les contrôles à effectuer sur la spécification, puis sur le SI réalisé, afin de s'assurer de sa fidélité à la réalité modélisée. Ce complément sémantique est ce que [FOU 82]appelle le "système d'intégrité" ; il est traditionnellement décrit, en conception des SI, sous la forme d'un ensemble de contraintes d'intégrité. Remarquons que, d'un SI à l'autre, le système d'intégrité est souvent voisin, tout au moins en ce qui concerne les types d'objets.
Or, avec les moyens de modélisation actuels, on est amené, à chaque conception d'un nouveau SI, à redéfinir les contrôles de Types à effectuer sur les données. Notons, de plus, que les langages traditionnels de l'Informatique de gestion tels que COBOL, PL1, GAP .... proposent peu (voire rien) en matière de contrôle de type.

Les Types Abstraits apportent un certain nombre de solutions à ces problèmes :

i) Il est démontré que le cadre formel associé aux Types Abstraits permet la vérification de la cohérence et de la complétude d'une spécification ; des techniques de preuves existent en particulier pour les TA algébriques en utilisant les axiomes comme règles de réécriture.

ii) Il existe actuellement des outils de décision et de contrôle, associés ou non à des langages, qui sont capables de vérifier la cohérence d'un ensemble de spécifications de types ; il s'agit de systèmes intelligents qui permettent de détecter des anomalies telles que l'incomplétude, la récursivité, la mauvaise utilisation des règles de visibilité. En particulier CLU [LIS 77] réalise le contrôle de la complétude de descriptions de types.

iii) Le fait de décrire un SI en termes de types permet d'expliciter des contrôles d'intégrité au plus haut niveau de définition des schémas de type servant à la spécification du SI au lieu de les répéter pour chaque nouveau type introduit. Le mécanisme de généricité des types permet d'expliciter, ainsi, des contrôles automatiques à appliquer lors de l'instanciation des schémas. Des langages tels que CLU ou ADA [ADA 80] apportent, ici aussi, sécurité et cohérence par des contrôles de types proposés sur les types instanciés.

iv) Lors de l'implémentation d'un SI particulier, c'est-à-dire lors de l'entrée des occurrences correspondant aux instanciations des schémas, de nouveaux contrôles peuvent être appliqués pour vérifier que les objets respectent les invariants et autres restrictions définis sur les types. Notons qu'aucun langage actuel ne propose une définition propre et facile, autre que par des appels à des procédures, pour l'expression des invariants et des restrictions ; même CLU est très limité sur cet aspect.

1.2.2. La réutilisation de la connaissance

De façon similaire aux contrôles, chaque conception d'un nouveau SI induit la redéfinition de traitements et de leurs conditions d'application sur les données. Or certaines opérations telles que la création d'un objet, ou sa suppression, voire sa mise à jour, sont toujours de même nature, à quelques paramètres près, pour tous les types d'objets. Il est donc inutile de les redéfinir pour chacun d'entre eux.

Un autre problème est de constater qu'un type de donnée (ou un type de traitement) est presque le "même" qu'un autre type et qu'il serait utile de réutiliser une définition de type déjà exprimée en la reprécisant pour un nouveau contexte.

Là aussi les TA apportent la résolution de ces problèmes.

i) Les opérateurs des schémas de type sont génériques. Ils sont définis une fois pour toutes et toute instanciation d'un constructeur induit la possibilité d'utiliser des opérateurs prédéfinis. On diminue ainsi le travail du concepteur et on accroît son efficacité. Les clusters de CLU et les packages d'ADA permettant une telle définition de schémas paramétrés (en fait pas totalement quelconques) munis de leurs opérations.

ii) Un certain nombre de mécanismes permettent la réutilisation de définition de type, ce que l'on appelle la construction incrémentale de la spécification. Les plus connus sont les mécanismes :

. d'"enrichissement" ou "extension" : il permet la construction d'une spécification à partir d'une autre plus simple par adjonction d'opérateurs,

. de "restriction" : il conduit à la suppression d'opérations,

. de "renommage" : il permet de réutiliser un type déjà décrit dans un nouveau contexte,

. de "paramétrisation" : il induit une généralisation qui permet une utilisation plus étendue des définitions.

Notons que les langages actuels proposent peu de choses à ce niveau mise à partune paramétrisation plus ou moins développée.

1.2.3. La méthode de structuration de la réalité

i) La spécification à l'aide des TA implique une méthode modulaire efficace et évolutive ainsi qu'une conception par étapes. Les TA

offrent un arrière plan formel pour raisonner sur la spécification des SI, en particulier les Types hiérarchiques offrent un outil uniforme pour une conception structurée et minimale des SI.

ii) La spécification du SI peut être réalisée sans préjuger de la représentation choisie pour les types. Une fois déterminée la fonction de transformation Types abstraits $\Rightarrow$ Types représentés, l'implémentation du SI peut être réalisée automatiquement. CLU dissocie la définition d'un cluster de sa représentation, les outils de vérification de type étant opérationnels sur les spécifications abstraites.

iii) Comme pour tout système modulaire, l'évolutivité d'un ensemble de spécifications réalisées à l'aide des TA est grande :

. soit en usant d'un mécanisme de réutilisation de la connaissance pour définir un nouveau type,

. soit en modifiant un type lui-même ; la définition regroupée des opérations d'un type facilite la maintenance de la spécification puisque les conséquences d'un changement sont facilement localisables.

Ces différentes remarques nous ont conduit à penser que les TA peuvent enrichir la conception d'un SI. La suite de ce chapitre présente une généralisation formelle de la définition d'un SI en termes de Types Abstraits. Nous fondons notre formalisation sur le modèle conceptuel de REMORA [FOU 82] ; cette base nous permet de prédéfinir des schémas de type servant à la spécification d'un SI munis de leurs opérateurs explicites ainsi au niveau le plus générique qu'il soit. Notons que dans les méthodes de conception proposées actuellement [BRO 82, CHA 82, VER 82, ASH 82, KNU 82, WAS 82A], les contrôles et les opérations ne sont pas spécifiés à un niveau "méta", (c'est-à-dire "schémas") mais sont à redéfinir pour chaque nouveau type introduit.

2. UNE FORMALISATION DES SI A L'AIDE DES TA

2.1. Introduction

Notre but est de proposer une formalisation de SI, c'est-à-dire une théorie qui permette de déterminer l'ensemble des formules vraies, bien formées, ou encore l'ensemble des termes acceptés dans le SI par analogie à l'axiomatisation des algèbres.

Les termes valides sont, pour nous, les spécifications obtenues à l'aide du modèle dont nous avons rappelé les grandes lignes au chapitre précédent.

Un système d'information est, par rapport à notre modèle, un produit cartésien de classes d'objets, d'opérations et d'événements ; les contraintes d'intégrité sont exprimées par un ensemble de prédicats permettant de vérifier la cohérence de ses différents composants.
Avec la convention : < , , > représente un produit cartésien, on peut donner la définition suivante d'un SI :
< cob : CLAS, cop : COP, cev : CEV >

où
. CLAS est un produit cartésien de classes d'objets de la forme
$< C_{r1} : cob_1, \ldots, C_{r\alpha} : cob_\alpha, \ldots, C_{rkcob} : cob_{kcob} >$
. COP est un produit cartésien de classes d'opérations de la forme
$< O_{r1} : cop_1, \ldots, O_{r\beta} : cop_\beta, \ldots, O_{rkcop} : cop_{kcop} >$
. CEV est un produit cartésien de classes d'événements de la forme
$< E_{r1} : cev_1, \ldots, E_{r\gamma} : cev_\gamma, \ldots, E_{rkcev} : cev_{kcev} >$

Chaque classe d'objets cob_α, d'opérations cop_β, d'événements cev_γ subit, elle-même, une décomposition hiérarchique en fonction de constructeurs plus primitifs ; ces constructeurs sont, à leur tour, définis à partir de schémas paramétrés habituels tels que le type SUITE, FILE, produit cartésien .... jusqu'à l'axiomatisation de base que nous ne redonnons pas ici. Notre formalisation en couches successives peut se schématiser ainsi :

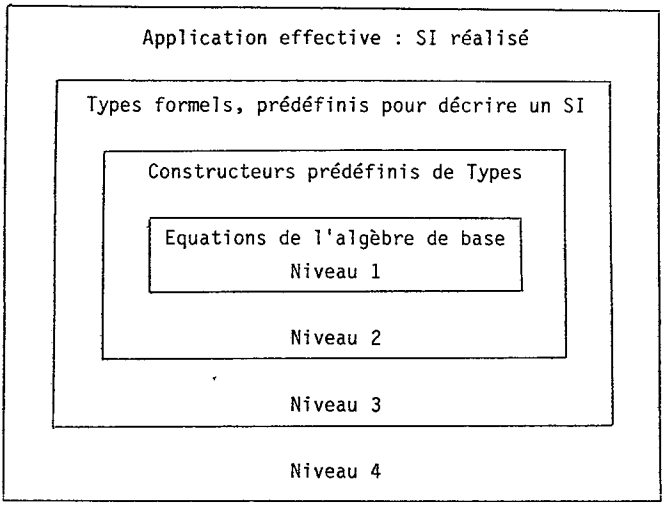


Figure 1 : formalisation en couches

Nous nous plaçons dans ce travail directement au niveau 2, on trouvera en annexe A, les rappels théoriques qui permettent d'expliciter le niveau 1. Dans un premier temps nous donnons les constructeurs de base. Nous définissons ensuite le formalisme d'expression de nos types et explicitons, à l'aide de ce formalisme, les différents types paramétrés permettant la spécification de SI (niveau 3). Nous concluons cette partie par la définition du type SI, des différents invariants, contraintes et opérations qui s'y appliquent.

Le chapitre III de cette partie, les parties II et III de cette thèse permettront d'apporter des éléments de réponse au niveau 4.

2.2. Les constructeurs prédéfinis de Type

2.2.1. Le formalisme d'écriture

La définition des constructeurs de base est fondée sur les Types Abstraits Algébriques, les invariants étant spécifiés en termes de pré/post-conditions. La spécification algébrique de chacun des constructeurs se compose d'une partie syntaxique et d'une partie sémantique.

i) partie syntaxique comprenant :

a) le nom du type et la liste de ses paramètres, s'il y a lieu, introduits par le mot clé type ;

b) la description des opérateurs introduits par le mot-clé opérations ; c'est-à-dire des noms des opérations (opérateurs infixés, notées op) et des fonctions (opérateurs préfixés, notées fonct).
A chacune d'elles est associé un profil qui est une liste de symboles de domaines et codomaines appelés sortes.

ii) partie sémantique comprenant :

a) la liste des équations conditionnelles, récursives caractérisant l'effet des combinaisons autorisées des opérations associées au type à partir d'autres opérations (choisies comme "constructeurs primitifs"). Elle est introduite par le mot clé axiomes.

Acceptant la surcharge d'opérateurs dans la définition des équations nous notons :

. ω la constante indéfinie,

. = le prédicat d'égalité

. si-alors-sinon l'opération définie pour tout type T vérifiant les axiomes suivants : soient t,t' : T

si-alors-sinon (vrai,t,t') = t

si-alors-sinon (faux,t,t') = t'

b) la restriction des objets produits par le type à ceux satisfaisant une propriété appelée invariant ;

c) la définition des opérations partielles au moyen de pré-conditions ;

d) l'expression de propriétés particulières sur les paramètres d'un schéma de type par une "contrainte sur les paramètres".

Afin de pouvoir introduire des commentaires, nous adoptons dans toute la suite une présentation en trois colonnes : lexique, profil et définition formelle.

Les types que nous explicitons, dans ce paragraphe, sont les constructeurs de base élémentaires dont la définition hiérarchique est limitée à un seul niveau de décomposition. Ils s'appuient sur trois types primitifs non paramétrés que nous rappelons ici.

2.2.2. Les types primitifs

Nous avons retenu trois types primitifs qui sont :

a) le type ENTIER,

b) le type BOOL,

chacun d'eux étant muni des opérations habituelles : lois de composition interne, opérateurs de comparaison, opérateurs spécifiques aux entiers tels que MAX, MIN, MOYENNE etc ...

c) le type CHAINE muni des opérateurs traditionnels sur les chaînes tels que

. CONCAT (ch1, ch2) qui réalise la concaténation des deux sous-chaines ch1 et ch2 ;

. EXTRACT (ch1,i,j) qui extrait une sous-chaîne dans ch1 (à partir du caractère i et sur j caractères) ;

. RECH (ch1, ch2) qui recherche la présence de ch2 dans ch1.

On supposera connue la définition algébrique de ces trois types de base ainsi que l'axiomatisation de leurs opérateurs.

2.2.3. Le constructeur "ensemble des parties"

Il permet de définir l'ensemble des parties que l'on peut construire sur un ensemble E ; il est noté $\mathcal{P}(E)$

lexique	profil	définition formelle
		type $\mathcal{P}(E)$ opérations
. création d'un ensemble de parties	vide : fonct ( ) $\mathcal{P}(E)$	. vide
. adjonction d'un élément à un ensemble des parties	ADJPAR : fonct($\mathcal{P}(E)$, E) $\mathcal{P}(E)$	. ADJPAR
. suppression d'un élément dans un ensemble	SUPAR : fonct($\mathcal{P}(E)$, E) $\mathcal{P}(E)$	. SUPAR
. teste si un élément appartient à un ensemble	$\in$: op($\mathcal{P}(E)$,E) BOOL	. $\in$
. teste si l'ensemble des parties est vide	ESTVIDE : fonct ($\mathcal{P}(E)$) BOOL	. ESTVIDE
. cardinal d'un ensemble de parties	CARD : fonct($\mathcal{P}(E)$) ENTIER	. CARD
On choisit "vide" et "ADJPAR" comme constructeurs primitifs.	x,y $\in$ E A : $\mathcal{P}$ (E)	axiomes
. définition de l'appartenance d'un élément à un ensemble de parties		$\begin{cases} x \in \text{vide} = \text{faux} \\ x \in \text{ADJPAR}(A,y) = \text{si } x=y \text{ alors vrai sinon } x \in A \end{cases}$

. définition de la suppression d'un élément dans un ensemble de parties . définition du test de la vacuité d'un ensemble de parties . définition du cardinal d'un ensemble de parties	A : $\mathcal{P}$ (E) x ∈ E	SUP(vide,x) = vide SUP(ADJPAR(A,y),x) = si x = y alors A sinon ADJPAR(SUP (A,x),y) ESTVIDE(vide) = vrai ESTVIDE(ADJPAR(A,x)) = faux CARD(vide) = 0 CARD(ADJPAR(A,x)) = si x ∈ A alors CARD(A) sinon CARD(A)+1 pré-condition pré ADJPAR(A,y) = non(y ∈ A)

Nous définissons une opération complémentaire qui permet de construire un sous-ensemble d'un ensemble des parties ; notée { } elle a pour profil : { } : fonct ($\mathcal{P}$(E)) $\mathcal{P}$(E).
En fait ce n'est pas n'importe lequel des sous-ensembles qui nous intéresse, mais celui dont les éléments sont sélectionnés, parmi tous, suivant un certain prédicat φ, dont le profil est :

φ : fonct (E) BOOL.

On étend alors le type d'opération, défini par la constitution de sous-ensemble notée { } , par une famille de fonctions de ce type permettant de sélectionner des sous-ensembles selon un critère exprimé par le prédicat φ ; cette famille d'opérations est notée, par abus d'écriture, { }$_\varphi$.
Son profil est { }$_\varphi$: fonct ($\mathcal{P}$ (E)) $\mathcal{P}$(E).

Axiomes soient A : $\mathcal{P}(E)$, $x \in E$

$\{\ \}_{\varphi}$ (vide) = vide

$\{\ \}_{\varphi}$ (ADJPAR(A,x)) = si appl (φ,x)
alors ADJPAR ($\{\ \}_{\varphi}$ (A), x)
sinon $\{\ \}_{\varphi}$ (A)

2.2.4. Le constructeur TABLE

Il est le constructeur fondamental pour l'expression de nos types conceptuels. Il représente les fonctions d'un ensemble de valeurs de clés noté CLE, dans un ensemble de valeurs noté A ; il permet de définir le type dont les éléments sont toutes les fonctions qui, à une valeur de clé c ∈ CLE, fait correspondre une valeur v ∈ A.
On le note [CLE → A].

lexique	profil	définition formelle
		type TABLE(CLE, A)
		opérations
. opération de création d'une table	Tvide : fonct ( ) TABLE	. Tvide
. adjonction d'un couple (clé, valeur) dans une table	AJOUT : fonct (TABLE, CLE,A) TABLE	. AJOUT
. changement d'une valeur dans une table, associée à une clé donnée	CHG : fonct (TABLE, CLE,A) TABLE	. CHG
. ensemble des clés d'entrée dans une table	DOM : fonct (TABLE) $\mathcal{P}$(CLE)	. DOM

. ensemble des valeurs dans une table	CODOM : fonct (TABLE) $\mathcal{P}$(A)	. CODOM
. trouver la valeur associée à une clé donnée	APP : fonct (TABLE,CLE)A	. APP
. trouver la clé associée à une valeur (ou les clés)	PROJT : fonct (TABLE,A) $\mathcal{P}$(CLE)	. PROJT
. supprimer un couple (clé,valeur) dans une table	SUP : fonct (TABLE,CLE, A) TABLE	. SUP
. cardinal d'une table (nombre de couples (clé, valeur))	cardt : fonct (TABLE) ENTIER	. cardt
Les constructeurs primitifs sont Tvide et AJOUT		axiomes
. définition de APP ; ω représentant la valeur indéfinie	c,c' : CLE v : A T : TABLE(CLE,A)	APP(Tvide,c) = ω APP(AJOUT(T,c,v),c') = si c = c' alors v sinon APP(T,c')
. définition de DOM ; utilise la fonction adjonction de l'ensemble des parties $\mathcal{P}$	c : CLE v : A ADJPAR : fonct ($\mathcal{P}$(A),A) $\mathcal{P}$(A)	DOM(Tvide) = ω DOM(AJOUT(T,c,v)) = ADJPAR(DOM(T),c)
. définition de codomaine CODOM		CODOM(Tvide) = ω CODOM(AJOUT(T,c,v)) = ADJPAR(CODOM(T),v)

. définition de PROJT ; utilise la même fonction ADJPAR	v,v' : A c : CLE T : TABLE(CLE,A)	PROJT(Tvide,v) = Tvide PROJT(AJOUT(T,c,v),v') = si v = v' alors ADJPAR(PROJT(T,v),c) sinon PROJT(T,v)
. définition de la suppression d'un couple (clé, valeur)	c,c' : CLE v,v' : A	SUP(Tvide,c,v) = Tvide SUP(AJOUT(T,c,v),c',v') = si c = c' alors T sinon AJOUT(SUP(T,c',v'), c,v)
. définition du changement de la valeur qui est associée à l'entrée c : CHG en fonction de AJOUT, SUP (et APP pour connaître la valeur associée à c)		CHG(Tvide,c,v) = Tvide CHG(T,c,v) = AJOUT(SUP(T, APP(T,c)),c,v)
. cardinal d'une table T est le nombre de couples (c,v) associés aux entrées c	c : CLE	cardt(Tvide) = 0 cardt(AJOUT(T,c,v)) = si c ∈ DOM(T) alors cardt(T) sinon cardt(T) + 1

pré-conditions soient c : CLE, v : A
T : TABLE (CLE,A)
pré APP (T,c) = c ∈ DOM(T)
pré AJOUT (T,c,v) = non (c ∈ DOM(T))
pré SUP (T,c,v) = c ∈ DOM(T)
pré CHG (T,c,v) = c ∈ DOM(T)

Symétriquement à ce que nous avons fait pour le constructeur ensemble des parties, nous définissons l'opération d'extraction d'une sous-table, notée extract : fonct (TABLE) TABLE.

En fait nous étendons cette opération par une famille de fonctions de ce type permettant de sélectionner des sous-ensembles selon un critère exprimé par un prédicat Ψ où Ψ : fonct (A) BOOL ; cette famille d'opérations est notée, par abus d'écriture, extract$_{\Psi}$; son profil est extract$_{\Psi}$: fonct (TABLE) TABLE.

axiomes Soient T : TABLE (CLE,A), c : CLE,v : A
extract$_{\Psi}$ (Tvide) = Tvide
extract$_{\Psi}$ (AJOUT(T,c,v)) = si Appl(Ψ,v)
alors AJOUT(extract$_{\Psi}$(T),c,v)
sinon extract$_{\Psi}$(T)

2.2.5. Le constructeur SUITE

Il permet de définir les suites finies que l'on peut construire sur un ensemble E ; notées SUITE(E) ce sont les fonctions α de la forme [ENTIER → E] dont le domaine est un intervalle de borne inférieure notée bi(α) et de borne supérieure notée bs(α).

lexique	profil	définition formelle
		type SUITE(E)
		opérations
. création d'une suite	<> : op ( ) SUITE(E)	. <>
. concaténation d'une suite avec un nouvel élément de E	∿ : op(SUITE(E),E) SUITE(E)	. ∿
. donne la suite, hors dernier élément	DEBUT : fonct(SUITE(E)) SUITE(E)	. DEBUT
. donne le dernier élément d'une suite	DER : fonct(SUITE(E)) E ∪ {ω}	. DER
. donne la longueur d'une suite c.a.d. son nombre d'éléments	LONG : fonct(SUITE(E)) ENTIER	. LONG
. donne le premier élément d'une suite	PREM : fonct(SUITE(E)) E ∪ {ω}	. PREM
. donne la suite, hors premier élément	RESTE : fonct(SUITE(E)) SUITE(E)	. RESTE
. donne la borne inférieure d'une suite	bi : fonct(SUITE(E)) ENTIER	. bi
. donne la borne supérieure d'une suite	bs : fonct(SUITE(E)) ENTIER	. bs
		axiomes
Les constructeurs primitifs choisis sont : <> et ∿		
. définition de DEBUT	α: SUITE(E) x : E	DEBUT(α ∿ x) = α DEBUT(<>) = <>

lexique	profil	définition formelle
. définition de $<>$; le dernier élément d'une suite vide est défini	α : SUITE(E) x : E	$\begin{cases} DER(<>) = \omega \\ DER(\alpha \frown x) = x \end{cases}$
. définition de LONG ; l'opération + est celle des ENTIER		$\begin{cases} LONG(<>) = 0 \\ LONG(\alpha \frown x) = LONG(\alpha) + 1 \end{cases}$
. définition de PREM ; le premier élément d'une suite vide est indéfini		$\begin{cases} PREM(<>) = \omega \\ PREM(\alpha \frown x) = PREM(\alpha) \end{cases}$
. définition de RESTE		$\begin{cases} RESTE(<>) = <> \\ RESTE(\alpha \frown x) = RESTE(\alpha) \frown x \end{cases}$
. définition de borne inférieure, la suite vide a 1 pour borne inférieure		$\begin{cases} bi(<>) = 1 \\ bi(\alpha \frown x) = bi(\alpha) \end{cases}$
. définition de borne supérieure ; la suite vide a 0 pour borne supérieure ; l'opération + est celle des ENTIER		$\begin{cases} bs(<>) = 0 \\ bs(\alpha \frown x) = bs(\alpha) + 1 \end{cases}$

2.2.6. Le constructeur FILE

Il est défini comme une restriction du type SUITE.
En effet seul un sous-ensemble des opérations du type SUITE est nécessaire à la construction de ses objets.
type FILE(E) = SUITE(E)

Opérations
< > : op ( ) FILE(E) (création d'une file)
PREM : fonct (FILE(E)) E ∪ {ω} (prendre le premier élément)
RESTE : fonct (FILE(E)) FILE(E) (suppression du premier élément)
∿ : op (FILE(E),E) FILE(E) (rajouter un élément en fin)

Acceptant la surcharge d'opérateurs dans ce cas particulier nous avons gardé les mêmes symboles ; les axiomes et pré-conditions de définition des opérations partielles sont identiques à ceux du type SUITE.

2.2.7. Le constructeur produit cartésien

Il permet de définir le produit cartésien de n ensembles T1, T2, ..., Tn. Il admet pour ensembles sous-jacents les ensembles sous-jacents à T1, T2, ..., Tn et leur produit cartésien. Un nuple de ce type est noté <x1 : T1, x2 : T2, ..., xn : Tn>

lexique	profil	définition formelle
un objet de type PRODCAR est un nuple de valeurs particulières		type PRODCAR(T1, T2,...,Tn) opérations
. création d'un nuple	CONS : fonct(T1,T2, ...,Tn) PRODCAR	. CONS
. projection sur une valeur ai de type Ti	$PROJ_i$: fonct(PRODCAR) Ti	. $PROJ_i$
	ai : Ti	axiome $PROJ_i$(CONS(a1,...,an)) = ai

En conception de Base de Données, il est souvent nécessaire lorsqu'on manipule des relations au sens de CODD [COD 70] de disposer du nom des différents constituants, champs, de la relation.
C'est ainsi que J.J. CHABRIER dans le système VEGA [CHA 83] a introduit la notion de nuple labellé. De façon similaire nous introduisons un nouveau constructeur "produit cartésien indicé" dérivé du précédent et noté :

type $PRODCAR_{x1, x2,...,xn}(T1, T2,...,Tn)$ = <x1 : T1, x2 : T2, ..., xn : Tn>

En fait grâce à cette notation nous introduisons une infinité de constructeurs paramétrés PRODCAR, indicés donc dépendant tous des noms de leurs constituants, c'est-à-dire des noms des champs projection du produit cartésien.

Nous introduisons alors l'opération de projection sur un champ xi notée :

$PROJ_{xi}$: fonct $(PRODCAR_{x1,...,xn,})$ Ti

qui rend la valeur ai du champ xi. Par la suite cette opération sera notée, par commodité d'écriture, avec l'opérateur de :

Soit A : $PRODCAR_{x1,...,xn,}$, alors $PROJ_{xi}$ (A) sera notée : xi de A.

De façon similaire nous introduisons l'opération de modification d'un produit cartésien, qui change la valeur d'un champ en la remplaçant par une nouvelle ; elle est notée :

$CHGPROD_{xi}$: fonct $(PRODCAR_{x1,...,xn}$, Ti) $PRODCAR_{x1,...,xn}$

et son axiome est

$CHGPROD_{xi}$ $(CONS(a1,...,ai,...,an),v) = CONS(a1,...,a_{i-1},v,a_{i+1},...,an)$

Par la suite pour des raisons de commodité d'écriture et par abus de notation, on remplacera la notation :

$CHGPROD_{xi}$ (A,v) par CHGPROD (A,xi,v)

où : A : PRODCAR, v : Ti, xi : IDENT

Ceci permettra de passer le nom du champ en paramètre de l'opération et exprime, donc, que A, produit cartésien, voit son champ xi valorisé à v.

2.2.8. Le constructeur UNION

Il permet de définir l'union de n ensembles E,F,...,Z.
Il est noté C = UNION (E,F,...Z) et admet pour ensembles sous-jacents E,F,...,Z et pour opérations implicites la réunion des opérations sur E,F,...,Z. Nous l'utilisons pour définir l'union des types.

lexique	profil	définition formelle
On définit le type UNION comme un produit cartésien de Types Ti et de valeurs indéfinies ωi.		type UNION(T1,...,Tn) = <t1 : T1 ∪ {ω1}, t2 : T2 ∪ {ω2}, tn : Tn ∪ {ωn}>
		invariant
Appelons T le type obtenu par la réunion des Ti. Si un objet x est du type T, il ne peut être que du type T1 ou du type T2... ou du type Tn	T = UNION(T1,...,Tn) x : T	x : T ⟺ ∃! i , ti de x ≠ ωi
		opérations
. Opération qui permet de savoir pour un objet x de type T de quel type projection il est	$estype_i$: fonct(T) BOOL j ∈ [1...i-1 i+1...n]	. $estype_i(x)$ ⟺ tj de x = ωj ∀ j ≠ i
. convertit x de type T en un des types projections de l'union sachant que x est du type Ti	$conv_i$: fonct(T) Ti	. $conv_i(x)$ = xi t.q. ti de x = xi et tj de x = ωj ∀ j ≠ i
		pré-condition
		pré $conv_i(x)$ = $estype_i(x)$

Par la suite, par extension linguistique, plutôt que d'indicer les opérations estype et conv on préfèrera construire un nouvel opérateur en concaténant respectivement ces deux préfixes avec le suffixe représentant le type sur lequel opère l'opération. Ainsi on parlera de estypeLETTRE ou de estypeCHIFFRE si un élément est respectivement du type LETTRE ou du type CHIFFRE. Par commodité d'écriture lorsqu'un objet sera du type UNION, on confondra convTi(x) et ti de x.

2.2.9. Le type IDENT

Nous l'avons introduit, lors de la définition du produit cartésien indicé, comme permettant de nommer les différents champs d'un nuple. De fait tous les langages de programmation manipulent des identificateurs qui doivent obéïr à certaines règles. Cependant, d'habitude, l'identificateur d'un objet n'a pas d'intérêt par lui même mais plutôt en tant que fonction d'accès à une valeur. Dans notre développement nous avons besoin de manipuler les identificateurs des objets.

Ce type représente l'ensemble des identificateurs possibles autorisés dans notre langage. Il est défini comme une suite de chiffres et de lettres, appartenant à l'alphabet choisi, commençant par une lettre et dont la longueur ne dépasse pas douze caractères. C'est un type non paramétré dont la définition est limitée.

```
type IDENT = SUITE (UNION(CHIFFRE,LETTRE))
invariants
soit id : IDENT
      . LONG(id) ⩽ 12
      . estype LETTRE (PREM(id))
```

2.3. Les types du langage typé

Ces types sont tous des types paramétrés construits à partir des constructeurs prédéfinis introduits précédemment. Leurs opérations et fonctions sont définies à partir de l'axiomatisation algébrique introduite au paragraphe 2.2. Leur définition s'appuie sur la notion de type de base déduite, elle même, de la notion de type primitif et de type DATE. Ces types permettent la formalisation d'un SI (niveau 3, figure 1, page 92).

2.3.1. Formalisme d'expression des opérations

Nous en donnons ici les grandes lignes car le langage d'expression des textes sera développé plus tard (cf. paragraphe 2.3.6).
En fait un sous-ensemble de ce que nous appelons le "langage d'énoncé" (cf. chapitre III) nous est nécessaire ici. Le principe de base est un raisonnement déductif sur l'opération à formaliser, les structures et les constructions utilisables étant celles de notre langage LASSIF.

Nous donnons ici les notations retenues :

i) le parenthésage ( ) permet d'exprimer l'application d'une opération du type en cours de définition ou d'un type qui lui est hiérarchiquement supérieur ;

ii) les crochets [ ] permettent d'exprimer une fonction d'entrée dans une table, c'est-à-dire l'application d'une clé à une entrée de table donnée ; rendant la valeur correspondant à l'entrée ;

iii) la projection sur un champ d'un produit cartésien s'exprime à l'aide du mot-clé de ; nom-du-champ de nom-du-produit et est éventuellement itérative.

iv) les notations ensemblistes habituelles $\{ / \}$, $\exists$, $\forall$ sont utilisées.

v) les notations proposées au paragraphe 2.2.1. pour exprimer les axiomes des constructeurs de base restent valables ici ; en particulier le prédicat d'égalité et la construction si-alors-sinon.

Pour respecter une démarche déductive et un raisonnement progressif, nous avons introduit beaucoup d'intermédiaires d'écriture, lors de l'explicitation des types, en espérant que cela facilitera la compréhension de la suite.

2.3.2. Les types de base

La notion de type de base recouvre les types primitifs définis précédemment, souvent plus ou moins explicites dans les langages de programmation, plus un type particulier : le type DATE.

TYPEBASE = UNION (TYPEPRIM, DATE)

où

TYPEPRIM = UNION (ENTIER, BOOL, CHAINE)

Le type DATE

Ce type est fondamental dans la conception de Systèmes d'Informations dans la mesure où l'on veut faire une représentation historique de la réalité ; notre modèle conceptuel incluant le temps dans sa définition, il nous a été nécessaire d'introduire ce type comme type de base. Sa définition est analogue à ce que propose [ADI 85] dans le cadre du projet TIGRE.

Ce type représente l'ensemble des dates absolues, aussi précises que possible dans l'utilisation d'un système informatique, donc définies par rapport à un calendrier de temps aussi élémentaire que possible : année, mois, jour, heure, minute et seconde.

Il est défini, comme par exemple en CLU [LIS 77], comme un produit cartésien d'entiers et donc les opérations qui lui sont spécifiques se déduisent des opérations sur les entiers et sur le produit cartésien.

lexique	profil	description formelle
une date est composée de jour, mois, année, heure, minute, seconde	d : DATE	type DATE : <j : ENTIER, m : ENTIER, a : ENTIER, h : ENTIER, mn : ENTIER, s : ENTIER>
. contraintes succinctes et grossières		invariant $1 \leqslant j$ de d $\leqslant 31 \wedge 1 \leqslant m$ de d $\leqslant 12 \wedge$ $0 \leqslant a$ de d $\leqslant 99 \wedge 0 \leqslant h$ de d $< 24 \wedge$ $0 \leqslant mn$ de d $< 60 \wedge 0 \leqslant s$ de d < 60
. comparaison de deux dates	d1 : DATE d2 : DATE $\leqslant$: op (DATE, DATE) BOOL	opérations $\leqslant$: tq d1 $\leqslant$ d2 = (a de d1 < a de d2) $\vee$ (a de d1 = a de d2 $\wedge$ (m de d1 < m de d2 $\vee$ (m de d1 = m de d2 $\wedge$ (j de d1 < j de d2 $\vee$ (j de d1 = j de d2 $\wedge$ (h de d1 < h de d2 $\vee$ (h de d1 = h de d2 $\wedge$ (mn de d1 < mn de d2 $\vee$ (mn de d1 = mn de d2 $\wedge$ s de d1 $\leqslant$ s de d2))))))))
. différence de deux dates, ramenées en secondes grossièrement	- : op (DATE, DATE) ENTIER	- : tq d1 - d2 = [s de d1 + 60 * (mn de d1 + 60 * (h de d1 + 24 * ((j de d1 - 1) + 30 * (m de d1 - 1) + 12 * (a de d1 - 1))))] - [s de d2 + 60 * (mn de d2 + 60 * (h de d2 + 24 * ((j de d2 - 1) + 30 * ((m de d2 - 1) + 12 * (a de d2 - 1))))]
. maximum de deux dates	MAX : fonct(DATE, DATE) DATE	MAX : tq MAX(d1, d2) = si d1 $\geqslant$ d2 alors d1 sinon d2

2.3.3. Le type PREDICAT

Il permet de définir les prédicats associés aux éléments de description de notre langage. Un élément de type PREDICAT paramétré par l'ensemble E est en fait une fonction de cet ensemble E dans BOOL.

type PREDICAT(E) = TABLE(E, BOOL)

Ce type est muni d'une opération particulière qui fait correspondre au prédicat particulier "toujours", la valeur booléenne "vrai".

opération

toujours : fonct( ) PREDICAT(E)

axiome

APPL(toujours, e) = "vrai"

Nous introduisons deux constructeurs supplémentaires permettant l'expression des composantes permanentes et variables d'une classe de c-objets : SCHEMAP, SCHEMAV.

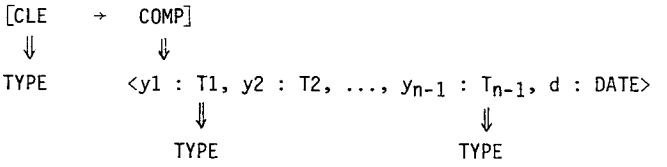
Nous en donnons une représentation graphique où :

- une table (A,B) sera représentée par [A → B]
- une décomposition à un niveau inférieur de la hiérarchie des types sera représentée par ⇒.
- un produit cartésien par <--,--,-->

2.3.4. Type SCHEMAP

Un SCHEMAP représente une table qui, à une valeur de clé donnée, fait correspondre une produit cartésien COMP représentant les champs permanents d'une classe d'objets. Par hypothèse le dernier champ de COMP est une date.

Schématiquement on peut représenter ce type ainsi :



La clé est d'un type quelconque représenté par le mot-clé TYPE ; elle peut être éventuellement un produit cartésien, son type sera redéfini ultérieurement.

De la même façon les Ti sont des types quelconques redéfinis ultérieurement pour permettre une description hiérarchique des composantes.

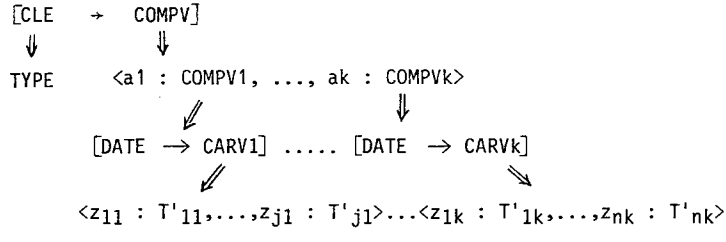
Dans l'utilisation de SCHEMAP pour définir de nouveaux types ou constructeurs, nous ne représentons plus la hiérarchisation CLE $\Longrightarrow$ TYPE, ni Ti $\Longrightarrow$ TYPE, elle sera supposée implicite.

lexique	profil	définition formelle
c'est une table de CLE dans COMP, lui même produit cartésien dont le dernier champ est une DATE	CLE : TYPE COMP = PRODCAR $^{(T1, T2, \ldots, T_{n-1}, DATE)}_{y_1, y_2, \ldots, y_{n-1}, d}$	type SCHEMAP(CLE,COMP) = TABLE(CLE,COMP)
	y_j : IDENT d : IDENT T_j : TYPE	opération
pour une valeur de clé donnée, ajouter les valeurs des champs permanents correspondants, cela revient à une adjonction en table	ADJP : fonct(SHEMAP, CLE,COMP)SCHEMAP SP : SCHEMAP c : CLE x : COMP	ADJP(SP,c,x) = AJOUT(SP,c,x)

Par hypothèse, les feuilles de la décomposition hiérarchique de CLE et des Ti devront être l'un des types de base introduits précédemment : plus précisément le type CLE ne peut être construit qu'à partir d'arguments de type TYPEPRIM (ou encore une clé d'un SCHEMAP ne doit pas contenir de DATE), les Ti peuvent être construits à partir d'arguments de type TYPEBASE quelconques.

2.3.5. Type SCHEMAV

Le type SCHEMAV a été introduit pour permettre la représentation de l'ensemble des champs variables d'une classe d'objets. C'est une TABLE qui, à une valeur de clé donnée, fait correspondre un produit cartésien COMPV dont chaque élément est lui-même une table faisant correspondre à une date un ensemble de champs variables ayant le même comportement dynamique.
Les ai de type COMPVi correspondent, dans notre modèle, aux sous-schémas variables de la classe d'objets, les CARVi représentent la liste des constituants (des champs) de chaque c-objet variable.
Schématiquement on peut représenter ce type ainsi :



Chaque T' ainsi que le type CLE sont des types quelconques redéfinis ultérieurement avec les mêmes contraintes de domaine des types feuilles que pour le type SCHEMAP.

lexique	profil	définition formelle
c'est une table de CLE dans COMPV ; COMPV est un produit cartésien dont chacun des champs est une table de DATE dans un produit cartésien de champs	CLE : TYPE COMPV = PRODCAR$_{a1,\ldots,ak}$ (COMPV$_1$,...,COMPV$_K$) COMPVi = TABLE(DATE, CARVi) CARVi = PRODCAR$_{zi1,..,zin}$ (T'i1,...,T'in) z_{ij}, ai : IDENT T'j : TYPE	type SCHEMAV(CLE,COMPV) = TABLE(CLE,COMPV)
. SV' est le nouveau schemav obtenu après l'adjonction d'un nuple il est obtenu après changement du produit cartésien p' de SV . on exprime la transformation de p' en utilisant l'opérateur sur les produits cartésiens CHPROD . il est obtenu en ajoutant à la sous table concernée (champ ai de p) la valeur v avec une entrée de date d.	ADJV : fonct(SCHEMAV, CLE, IDENT, DATE, CARVi) SV : SCHEMAV clé : CLE ai : IDENT d : DATE v : CARVi SV' : SCHEMAV p,p' : COMPV	opération ADJV(SV,clé,ai,d,v) = SV' SV' = CHG(SV,clé,p') p' = CHPROD(p,ai,AJOUT (ai de p,d,v))
. le produit cartésien p concerné est celui qui correspond à l'entrée de clé c dans la table SV		p = SV[c]

2.3.6. Expression des concepts du modèle conceptuel de SI

Rappelons que les concepts de base de notre modèle sont :

- la classe d'objets (CLASSE-OB)
- la classe d'événements (CEVENT)
- la classe d'opérations (COPERATION).

Nous avons choisi ces concepts comme types de base de notre spécification. A l'intérieur de la classe CEVENT sont représentées les relations avec les deux autres types de classe. Cela permet de limiter les invariants et les contrôles, en particulier de complétude, définis sur le schéma conceptuel.

A - Le type CLASSE-OB

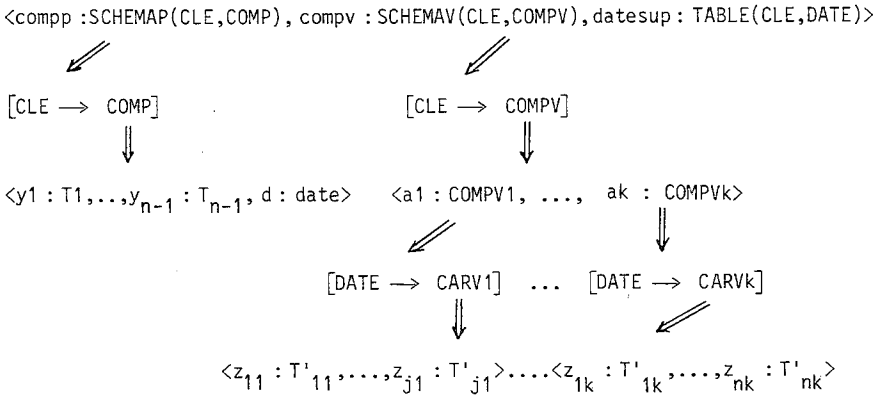
Il permet l'expression du concept de classe d'objets. Sémantiquement un élément de type CLASSE-OB représente une classe d'objets réels ou une classe d'associations d'objets réels.

Plus précisément, il comprend :

- une propriété (ou ensemble de propriétés) identifiante : la clé (CLE) ;
- une liste de constituants permanents c'est-à-dire gardant, pour un objet donné, une valeur constante au cours du temps et connue lors de la création de l'objet ; parmi eux il y a le constituant "date de création" de l'objet, placé en dernier champ (COMP) ;
- une collection de sous-schémas variables représentant les différents aspects variables de la classe, chaque sous-schéma (COMPVi) correspondant à un ensemble de propriétés ayant le même comportement dynamique au cours du temps ;

- une propriété date de suppression permettant de savoir, pour chaque objet de la classe, s'il a disparu du SI considéré, à un instant donné (par ex : démission d'un employé) ;
- un ensemble de prédicats assurant la cohérence de la classe (les contraintes d'intégrité) : predc représente les prédicats sur les aspects permanents, predvi ceux associés à un sous-schéma variable COMPVi, preg les prédicats généraux de la classe.

Schématiquement on peut représenter ce type ainsi :



lexique	profil	définition formelle
une CLASSE-OB est un produit cartésien d'un SCHEMAP, d'un SCHEMAV exprimant respectivement les composantes permanentes et variables de la classe, et d'une table des dates où les objets ont disparu du SI	preg : $\mathcal{P}$(PREDICAT(CLASSE-OB)) predc : $\mathcal{P}$(PREDICAT(COMP) predvi = $\mathcal{P}$(PREDICAT($COMPV_i$)) $COMP = PRODCAR_{y_1,..,y_{n-1},d}^{(T_1,...,T_{n-1},DATE)}$ $COMPV = PRODCAR_{a1,..,ak}^{(COMPV_1,...,COMPV_k)}$ $COMPV_i = TABLE(DATE,CARV_i)$ $CARV_i = PRODCAR_{zi1,...,zin}^{(T'_{i1},...,T'_{in})}$	type CLASSE-OB(CLE,compp,COMP, compv,COMPV,datesup,predc, predv1,...,predvk,preg) = <compp : SCHEMAP(CLE,COMP), compv : SCHEMAV(CLE,COMPV), datesup : TABLE(CLE,DATE)>
. Tout objet supprimé doit exister auparavant dans le système . Tout schéma variable doit correspondre à un schéma permanent existant . Tout prédicat général de la classe (preg) doit être vérifié . Tout prédicat ($pred_c$) associé aux aspects permanents de la classe doit être vérifié . A chaque sous schéma variable est associé un ensemble de prédicats ($predv_i$) qui doivent être vérifiés.		invariants . DOM(datesup de cob) ⊂ DOM(compp de cob) . DOM(compv de cob) = DOM(compp de cob) . cob : CLASSE-OB ⟺ ∀ q ∈ preg, APPL(q,cob) = vrai ; ∀ p ∈ $pred_c$, ∀ nu ∈ compp de cob, APPL(p,nu) = vrai ; ∀ p ∈ $predv_i$, ∀ nu ∈ a_i de cob, APPL(p,nu) = vrai

lexique	profil	définition formelle
. ajouter une occurrence aux aspects permanents d'une classe ⇒ ajouter à compp et ⇒ initialiser à "vide" les sous schémas variables correspondants	ADJCP : fonct(CLASSE-OB,CLE, COMP) CLASSE-OB cob : CLASSE-OB cle : CLE y : comp sp,sp" : COMP sv, sv' : COMPV cob', cob" : CLASSE-OB nup : PRODCAR	opérations . ADJCP(cob,clé,y) = cob"
Soit cob" la nouvelle classe, elle est obtenue par modification du produit cartésien cob', en fait de la partie compp ; soit sp" ce nouveau schéma permanent il est obtenu par adjonction au schéma permanent, à la clé "clé", de la valeur "y" ; sp est en fait le schéma permanent compp de la classe cob considérée		cob" = CHPROD(cob',compp,sp") sp" = ADJP(sp,clé,y) sp = compp de cob
cob' est en fait obtenu par la modification de la partie variable compv ; soit sv' le schéma variable modifié en ajoutant au schéma sv le nuple vide.		cob' = CHPROD(cob,compv,sv') sv' = AJOUT(sv,clé,nup) nup = <Tvide, ..., Tvide>
sv représente en fait le schéma variable compv de la classe cob considérée		sv = compv de cob
. supprimer une occurrence de c-objet d'une classe ; soit cob' la classe obtenue en modifiant le produit cartésien cob ; en fait la table des dates dsup' ; dsup' est obtenue en ajoutant à dsup la date d pour l'entrée de clé "clé" ; dsup est en fait la table datesup de la classe cob considérée	SUPC : fonct(CLASSE-OB,CLE, DATE) CLASSE-OB	. SUPC(cob,clé,d) = cob' cob' = CHPROD(cob,datesup,dsup') dsup' = AJOUT(dsup,clé,d) dsup = datesup de cob

lexique	profil	définition formelle
. ajouter une occurrence d'un sous schéma variable d'une classe cob : obtenu en modifiant le produit cartésien cob par l'adjonction d'une ligne à un sous schéma a_i de compv	ADJCV : fonct(CLASSE-OB,CLE, IDENT,DATE,CARVi) CLASSE-OB	. ADJCV(cob,clé,ai,d,y) = cob' cob' = CHPROD(cob,compv,sv') sv' = ADJV(sv,clé,ai,d,y)
. donne les clés valides à l'instant t pour la classe cob, c'est-à-dire l'ensemble de clés des occurrences de la composante permanente qui existent à l'instant t ; ce sont celles qui n'ont pas été supprimées du SI ou celles qui ont été supprimées après t.	clévalides : fonct(CLASSE-OB, DATE) $\mathcal{P}$(CLE) cleval : $\mathcal{P}$(CLE) clé : CLE cob : CLASSE-OB	clévalides(cob,t) = cleval cleval = {clé $\in$ DOM(compp de cob)/ c $\notin$ DOM(datesup) de cob ou datesup de cob [c] > t}
. état d'une classe à l'instant t : ensemble des occurrences permanentes de clés valides, ensemble des occurrences de sous-schémas variables valides à l'instant t. Soit enscompl l'ensemble des occurrences valides représentant l'état ; il représente l'ensemble des couples . cperm qui est obtenu en appliquant à compp l'ensemble des clés valides . cvar qui est obtenu en extrayant de chaque sous schéma variable la valeur des champs valides de $CARV_i$; chaque nuple rendu est composé de la date de changement de valeur du sous-schéma variable (di) la plus proche de t, et des champs de $CARV_i$.	valides : fonct(CLASSE-OB, DATE) $\mathcal{P}$ (CLASSE-OB) extrait : fonct(COMPV,DATE) NUPLE NUPLE = <c1 : CARV1, d1 : DATE,...,ck : CARVK, dk : DATE> com : COMPV d : DATE	. valides(cob,t) = enscompl enscompl = {<cperm,cvar> / $\exists$ c $\in$ valides(cob,t), cperm = compp de cob [c] et cvar = extrait(compv de cob[c], t)} où extrait(com,t) = nuple où $\forall$i,1 $\leqslant$ i $\leqslant$ k ci de nuple = ai de com [di de nuple] di de nuple = MAX {d $\in$ DOM(ai de com) / d $\leqslant$ t}

B. Les Types TEXTE et FACTEUR

Ils sont nécessaires à la définition des types paramétrés COPERATION et CEVENT. Après avoir rappelé la définition générale de ces deux types dans notre modélisation, nous expliciterons la sémantique opératoire de notre langage d'énoncé. Les opérations permettent l'expression des énoncés de types TEXTE et FACTEUR ainsi que l'expression des contraintes d'intégrité ou prédicats de cohérence et des conditions de déclenchement (définies comme de type PREDICAT(E)).

1. Définition générale

Pour définir ces deux types, nous avons besoin d'un constructeur particulier noté EC-OB qui représente l'ensemble des parties que l'on peut définir sur tous les c-objets des classes d'objets d'un système d'informations : EC-OB = $\mathcal{P}_{SI}$ (CLASSE-OB)

a) Le type TEXTE

C'est un type un peu particulier : il représente l'ensemble des fonctions dont la partie gauche du profil est un ensemble de nuples de c-objets permanents, variables ou suppressions parmi lesquels un c-objet est privilégié, le c-objet paramètre du texte, les autres étant en fait les c-objets utilisés dans le texte ; et dont la partie droite est un nuple de c-objet modifié par le texte. En résumé un texte exprime le fait qu'une opération exploite un ensemble d'objets et rend le nuple de c-objet qui a été modifié.

type TEXTE = fonct(EC-OB, CLASSE-OB) CLASSE-OB

b) Le type FACTEUR

Il représente l'ensemble des fonctions dont la partie gauche du profil est un ensemble d'objets utilisés par le facteur ainsi que le nuple du c-objet paramètre qui a changé d'état ; et dont la partie droite est un ensemble de nuples de c-objets qui seront, ultérieurement, les paramètres d'un texte de opération.

type FACTEUR = fonct(CLASSE-OB, EC-OB) EC-OB

En résumé un facteur de déclenchement définit à partir d'un objet qui a changé d'état, les nuples de c-objet qui seront exploités successivement par un texte de c-opération. Il permet l'expression d'opération sur des ensembles.

2. Sémantique du langage d'énoncé

a) But de ce paragraphe

Notre but est de décrire comment un texte, en langage d'énoncé, exprime une transformation du Système d'Information. Il s'agit, en fait, d'associer un sens c'est-à-dire une valeur sémantique aux différents programmes du langage, soit aux textes d'opération, de facteur, de condition et de prédicat. Certains textes représentent une fonction de transformation du SI d'un état à un autre : c'est le cas des textes d'opération.

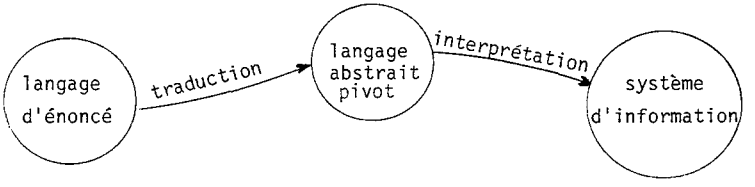
D'autres ne sont que l'examen de l'état du SI : textes de condition et de prédicat. Les textes de type facteur permettent de définir l'ensemble des objets du SI qui est intéressant pour un problème donné : c'est une fonction d'état.

Dans tous les cas, il faut étudier les propriétés des objets manipulés dans les programmes que ce soient des opérateurs de base (0-aires), des opérateurs simples (unaires ou n-aires) faisant intervenir en paramètres des opérateurs de base, ou des constructeurs permettant d'assembler les structures précédentes (schémas de programmes, itératifs, récursifs ...).

Le résultat de l'analyse est, dans un premier temps, de donner la syntaxe abstraite du langage d'énoncé choisi qui permet d'associer aux différents programmes les arbres abstraits construits par l'analyseur syntaxique abstrait. (cf. annexe 4 pour la grammaire abstraite du langage LASSIF).

C'est la définition de ce que certains appellent le langage pivot [LIV 78] (dans lequel les programmes sont des arbres), ou langage noyau, car il comporte des constructions plus simples que celles du langage initial. On exprime (ou traduit) chaque programme du langage d'énoncé dans la syntaxe du langage pivot.

Dans un deuxième temps on détermine l'interprète sémantique du langage pivot qui décrit comment associer à chaque programme sa valeur sémantique, c'est-à-dire comment les différents opérateurs et constructeurs agissent sur le SI et le font passer d'un état à l'autre. La figure suivante schématise ces deux étapes :



Notre analyse couvre deux étapes :

- définition du langage pivot
- définition de l'interprète du langage pivot.

b) Définition du langage pivot (ou abstrait)

Il s'agit de mettre en valeur les termes du langage c'est-à-dire les arbres abstraits que l'on peut construire à partir du langage utilisateur, soit : les opérateurs 0-aires, feuilles des arbres abstraits, les termes simples correspondant aux opérations de base des types, les constructeurs de programme.

Prenons un exemple de texte exprimé dans le langage d'énoncé LASSIF (cf. exemple complet en annexe) ; partie du texte correspondant au calcul d'un nouvel intervalle de disponibilité pour une chambre :

chdisp = ADJCV(chdisp de Cchambre, (numhot de nuple, numch de nuple), date, ddebres de nuple, dfinres de nuple)

ddebres de nuple = si chambgauche alors ndat1 de chambdispl
sinon ddebres de fact1
fsi

où . chdisp est le sous-schéma variable de la classe d'objets Cchambre représentant les chambres disponibles du système de réservation

. nuple représente le nuplet à adjoindre à chdisp,

. chambgauche est un booléen permettant de savoir si une chambre disponible a un intervalle de disponibilité qui jouxte à gauche l'intervalle de la chambre que l'on rend au système, cette recherche est exprimée par le calcul de chambdispl.

Cet extrait de texte met en valeur une opération de base ADJCV (ajonction d'un nuplet à un sous-schéma variable d'une classe d'objets) ; une construction de texte "si alors sinon" ; une définition de résultat " = ".

chambgauche est redéfini comme un booléen valant "vrai" s'il existe une chambre jouxtant à gauche :

chambgauche = cardt(chambdispl) = 1

où cardt est une opération de base sur les tables donnant le nombre d'élément de chambdispl,

et chambdispl est définie par l'expression prédicative d'accès aux données suivante qui est une nouvelle construction du langage :

chambdispl = {(x,n) dans dom (chambre de Cchambre),
(d2) dans dom (chdisp de Cchambre (x,n))
(d2) = dfindis de (chdisp de Cchambre ((x,n),d1)/

Soit y = ((x,n),d1), EXIST y dans dom (chdisp de Cchambre) t.q.
numhot de chambre de Cchambre [(x,n)] = numhot de fact1
et numch de chambre de Cchambre [(x,n)] = numch de fact1
et dfindis de chdisp de Cchambre [y] = ddebres de fact1}

Cette expression met en valeur un nouvel opérateur "dom" qui permet de connaître l'ensemble des clés d'une table.
Etendons cette analyse à l'ensemble des termes du langage.

i) Les opérateurs 0-aires du langage pivot

Ce sont les feuilles des arbres abstraits que le traducteur peut construire. Ils correspondent à l'expression :

- des constantes faisant partie des expressions arithmétiques,
- des identificateurs majuscules et minuscules spécifiés comme des chaînes de caractères particulières avec un champ étiqueté correspondant à l'identificateur lui même,
- des types de base utilisés par le langage : entier, booléen, chaîne, date dont le champ étiqueté valeur contiendra la valeur correspondant à l'objet typé.

Ces termes sont les opérations "constantes" du langage c'est-à-dire sans profil gauche.

ii) les opérateurs de base du langage pivot

Nous avons considéré un certain nombre de types prédéfinis dans le développement précédent (entier, booléen, chaîne, date) ; de façon similaire, ici, nous considérons comme prédéfinis dans le type texte les symboles d'entier, booléen, chaîne et date, ainsi que les symboles de leurs opérations.
Nous explicitons ici la définition, dans le langage pivot, des opérations de base paramétrées introduites précédemment. Nous nous limitons ici à la partie utile pour exprimer des textes d'informatique de gestion.

α) - les opérateurs de base associés au constructeur ensemble de parties $\mathcal{P}(E)$

. cardinal d'un ensemble de parties

CARDT : fonct (TEXTE) TEXTE

. extraction de sous-ensembles dans un ensemble de parties suivant un prédicat

$\{\}_{\varphi}T$: fonct (TEXTE) TEXTE

β) - les opérateurs de base associés au constructeur TABLE

. adjonction d'un couple (clé, valeur) dans une table

AJOUTT : fonct (TEXTE, TEXTE, TEXTE) TEXTE

. changement d'une valeur associée à une clé donnée dans une table

CHGT : fonct (TEXTE, TEXTE, TEXTE) TEXTE

. ensemble des clés d'une table

DOMT : fonct (TEXTE) TEXTE

- trouver la valeur associée à une clé dans une table

APPT : fonct (TEXTE, TEXTE) TEXTE

- supprimer un couple (clé, valeur) dans une table

SUPT : fonct (TEXTE, TEXTE, TEXTE) TEXTE

- cardinal d'une table (nombre de couples (clé, valeur))

CardtT : fonct (TEXTE) TEXTE

- extraction d'une sous table dans une table suivant un prédicat

extract$_{\varphi}^{T}$: fonct (TEXTE) TEXTE

γ) - Les opérateurs de base associés au constructeur CLASSE-OB

- adjonction, à une classe d'objets, d'une occurrence de sous-chéma permanent.

Ajp : fonct (TEXTE, TEXTE, TEXTE) TEXTE

- adjonction, à une classe d'objets, d'une occurrence de sous-schéma variable.

Ajv : fonct (TEXTE, TEXTE, TEXTE, TEXTE, TEXTE) TEXTE

- suppression d'une occurrence de c-objet dans une classe

supt : fonct (TEXTE, TEXTE, TEXTE) TEXTE

- état d'une classe d'objets

etat : fonct (TEXTE, TEXTE) TEXTE

iii) Les constructeurs du langage pivot

Pour exprimer des textes de programme il faut disposer, en plus des opérateurs 0-aires et des opérateurs de base, d'un certain nombre de constructions qui permettent de réaliser, en fait, des compositions de texte.

α - conditionnelle

cond : fonct (TEXTE, TEXTE, TEXTE) TEXTE

β - itération avec condition d'arrêt

tque : fonct (TEXTE, TEXTE) TEXTE

γ - itération sur un ensemble

pche : fonct (TEXTE, TEXTE, TEXTE) TEXTE

δ - séquence d'instructions

Elle permet de définir un ensemble d'instructions, composition des précédentes instructions et constructions.

seq : fonct (TEXTE, TEXTE) TEXTE

ε - déclaration de variables

Elle permet de définir une variable mathématique.

declv : fonct (TEXTE) TEXTE

η - accès aux données

acces : fonct (TEXTE, TEXTE) TEXTE

κ - définition d'un résultat

Elle permet d'introduire la définition formelle déductive d'un résultat ou d'un résultat intermédiaire.

egal : fonct (TEXTE, TEXTE) TEXTE

c) Définition de l'interprète du langage pivot

A chaque symbole du langage abstrait sera associée une fonction sémantique de transformation du SI. Chaque type d'arbre abstrait construit dans le langage pivot sera interprété lors de l'activation du texte correspondant dans le SI.

Nous donnons, comme précédemment, l'interprétation des opérateurs de base puis des constructions.

i) Interprétation des opérateurs de base

La fonction "eval" nous permet d'expliciter l'interprétation des termes ou objets de base du langage pivot avec la définition récursive générale suivante :

eval(φ(x1,...,xp)) = image (φ) (eval(x1),..., eval (xp))

où

. φ est l'opérateur, terme du langage pivot,

. image (φ) est le terme de transformation correspondant dans le SI,

. x1,...,xp sont les paramètres de type TEXTE du terme abstrait,

. eval(x1),..., eval(xp) sont les interprétations dans le SI des paramètres.

Soient t1, t2, t3, t4, t5 : TEXTE.

α) - opérateurs sur $\mathcal{P}(E)$:

. eval(CARDT(t1)) = CARD(eval(t1))

. eval($\{\}_\varphi^T$ (t1)) = $\{\}_\varphi$(eval(t1))

β) - opérateurs de TABLE :

. eval (AJOUTT(t1,t2,t3)) = AJOUT (eval(t1), eval(t2), eval(t3))

. eval (CHGT(t1,t2,t3)) = CHG (eval(t1), eval(t2), eval(t3))

. eval (DOMT(t1)) = DOM (eval(t1))

. eval (APPT(t1,t2)) = APP (eval(t1), eval(t2))

. eval (SUPT(t1,t2,t3)) = SUP (eval(t1), eval(t2), eval(t3))

. eval (cardtT(t1)) = cardt (eval(t1))

. eval (extract$_\varphi^T$(t1)) = extract$_\varphi$ (eval(t1))

γ) - opérateurs de CLASSE-OB

. eval (Ajp(t1,t2,t3)) = ADJP (eval(t1), eval(t2), eval(t3))

. eval (Ajv(t1,t2,t3,t4,t5)) = ADJV (eval(t1), eval(t2), eval(t3), eval(t4), eval(t5))

. eval (Supt(t1,t2,t3)) = SUPC (eval(t1), eval(t2), eval(t3))

. eval (etat(t1,t2)) = valides (eval(t1), eval(t2))

ii) Interprétation des constructeurs

La fonction "interp" nous permet d'expliciter l'interprétation des constructions du langage pivot en termes de transformation du SI. Les définitions sont récursives également ici.

Nous définissons un état de l'interprète [LIV 78] comme un couple <t/état> où état est un état de mémoire et t un morceau de programme. De manière intuitive t représente la suite des instructions qu'il reste encore à exécuter pour terminer l'interprétation du programme t_0 de départ.

La fonction "interp" permet d'exprimer la transition d'un état de mémoire à un nouvel état après avoir exécuté une instruction ins. Dans la suite nous présentons cette définition sur un état de la forme : interp (ins ; t/état).

Soient t1, t2, t3 : TEXTE.

α - interprétation d'une conditionnelle

```
interp (cond(t1,t2,t3) ; t/état) =
      si eval(t1/état) alors interp(t2 ; t/état)
                       sinon interp(t3 ; t/état)
      fsi
```

t1 étant une expression conditionnelle, eval(t1/état) permet de l'évaluer sur l'état de mémoire actuel.

β - interprétation de l'itération avec condition d'arrêt

```
interp (tque (t1,t2) ; t/état) =
      tantque eval (t1/état) faire interp (t2 ; t/état) ftque
```

γ - interprétation de l'itération sur un ensemble

```
interp (pche (t1,t2,t3) ; t/état) =
      pourchaque eval (t1/état) dans eval (t2/état)
      faire interp (t3 ; t/état) fpche
```

δ - interprétation de la séquence d'instruction

```
interp (seq (t1,t2) ; t/état) = interp (t1/état)
                                interp (t2 ; t/état)
```

ε - interprétation de la déclaration de variables x : TEXTE

```
interp (declv (x) ; t/état) = var x : TYPE
                              interp (t/état')
```

où état' est le nouvel état obtenu après l'adjonction d'une variable.

η - accès aux données

```
interp (accès(t1,t2) ; t/état) = {interp (t1/état) /
                                  interp (t2 ; t/état)}
```

κ - définition d'un résultat x, exp : TEXTE

interp (egal(x,exp) ; t/état) = interp (AJOUT(état,x,eval (exp/état)) ; t/état')

où état' est l'état de mémoire obtenu après adjonction d'une nouvelle valeur calculée (résultat de l'évaluation de exp) à l'entrée x de la table des états du système d'informations.

C. le type COPERATION

Sémantiquement ce type permet la représentation d'une classe d'opérations réelles. Plus précisément elle comprend :

- une propriété identifiante (unique) (NCOP) ;

- une propriété date d'exécution donnant pour chaque opération réelle exécutée sa date d'exécution dans le SI (datexec) ;

- une propriété c-objet modifié donnant pour chaque opération l'objet COB2 (permanent ou variable) qui est modifié par elle (cobmod) ;

- le texte de la classe d'opération ; plus exactement la solution choisie permet de conserver l'historique des textes d'opération et donc de savoir quel texte est applicable à une date donnée (top) ; le paramètre d'entrée est COB1, celui de sortie COB2 ; ce sont deux types d'objets définis par ailleurs ;

- un ensemble de prédicats assurant la cohérence de la classe d'opération.

type COPERATION (NCOP,datexec,cobmod,COB1,COB2,top,preg) =
<datexec : TABLE(NCOP,DATE), cobmod : TABLE(NCOP,COB2),
top : TABLE(DATE,TEXTE (COB1,COB2))>

Les types COB1 et COB2 ne sont pas redéfinis ici, ils correspondent en fait, ainsi qu'on l'explicitera dans le type SI, à des sous-schémas de classes d'objets.

lexique	profil	définition formelle
l'attribut de type TEXTE a deux paramètres : un sous-schéma en entrée (COB1) et un en sortie (COB2) qui est le c-objet modifié par le type d' opération	NCOP = <x1 : ENTIER> preg : $\mathcal{P}$ (PREDICAT(COPERATION))	type COPERATION(NCOP,datexec, cobmod,COB1,COB2,top,preg) = <datexec : TABLE(NCOP,DATE), cobmod : TABLE(NCOP,COB2), top : TABLE(DATE,TEXTE(COB1,COB2))> invariants cop : COPERATION $\Leftrightarrow \forall$ p $\in$ preg APPL(p,cop) = vrai
. les prédicats doivent être vérifiés		
. les domaines de valeurs des clés des tables datexec et cobmod sont les mêmes		. dom(cobmod de cop) = dom(datexec de cop)
	cop : COPERATION cob2 : COB2	opérations
. après l'exécution d'une opération, ajouter l'occurrence correspondante dans le SI : mettre sa date d'exécution dans la table "datexec" et mettre l'occurrence de c-objet dans la table "cobmod" ; ceci modifie le produit cartésien cop de plus haut niveau en cop'.	d : DATE ncop : NCOP ADJOP : fonct(COPERATION, NCOP,DATE,COB2) COPERATION	. ADJOP(cop,ncop,d,cob2) = cop' cop' = CHPROD(cop",datexec,dex') dex' = AJOUT(dex,ncop,d) dex = datexec de cop cop" = AJOUT(cobm,ncop,cob2) cobm = cobmod de cop

lexique	profil	définition formelle
. modifier le texte d'une opération revient à ajouter une occurrence de texte dans la table "top" du produit cartésien cop, changé en cop'	MODTOP : fonct(COPERATION, DATE,TEXTE) COPERATION d : DATE t : TEXT(COB1,COB2)	. MODTOP (cop,d,t) = cop' cop' = CHPROD(cop,top,top') top' = AJOUT(textop,d,t) textop = top de cop
. obtenir le texte d'une opération valide à l'instant d, c'est appliquer la fonction table "top" avec comme clé la date t c'est-à-dire la date la plus proche directement inférieure à d parmi les dates de texte de cop.	TEXTOP : fonct(COPERATION, DATE) TEXTE t,d : DATE	. TEXTOP(cop,d) = texte texte = top de cop [t] t = MAX {date ∈ DOM(top de cop, / date ⩽ d}

D. Le type CEVENT

Sémantiquement il permet la représentation d'une classe d'événements réels.
Plus précisément il comprend :

- une propriété identifiante (unique) (NCEV) ;
- une propriété date arrivée donnant pour chaque événement réel sa date de prise en compte dans le SI (datearriv) ;
- une propriété objet constaté COB donnant pour chaque événement l'objet qui a changé d'état et a donc provoqué l'événement, (cobconst) ;
- une propriété "prédicat" qui permet de conserver l'historique des textes de prédicat associé au type d'événement et donc de savoir, à une date donnée, quel est le texte à appliquer,
- une propriété composée "DEC" qui donne en fait pour chaque type d'opération pouvant être déclenchée à la suite d'un type d'événement (xi) (ou plus exactement pour chaque couple "type d'événement" "type d'opération" associés) :
  - . l'historique des déclenchements(h_{i1}) effectifs
  - . l'historique des textes de condition de déclenchement (h_{i2}) ;
  - . l'historique des textes de facteur de déclenchement (h_{i3}) ;
- un ensemble de prédicats destinés à vérifier la cohérence de la classe d'événements (preg).

type CEVENT(NCEV,datearriv,cobconst,COB,DEC,prédicat,copd,preg) =
<datearriv : TABLE(NCEV,DATE), cobconst : TABLE(NCEV,COB), prédicat : TABLE(DATE,PREDICAT(COB)), copd : DEC>

<x1 : DEC1, ... , xl : DECl>

<h_{11} : TABLE(NCEV,COP1), h_{12} : TABLE(DATE,COND1), h_{13} : TABLE (DATE,FAC1)>

lexique	profil	définition formelle
DEC représente le produit cartésien des triplets DECi :{copération, condition, facteur} associés à un type d'événement ; $COND_i$, FAC_i, PREDICAT ont tous trois pour paramètre le c-objet constaté COB.	DEC = PRODCAR(DEC1,...,DECl) x1,...,xl DEC_i = <h_{i1} : TABLE(NCEV,COP_i), h_{i2} : TABLE(DATE,$COND_i$), h_{i3} : TABLE(DATE,FAC_i)>	type CEVENT(NCEV,datearriv,COB, prédicat,cobconst,copd,DEC,preg) = <datearriv : TABLE(NCEV,DATE), cobconst : TABLE(NCEV,COB), prédicat : TABLE(DATE,PREDICAT(COB)), copd : DEC>
. tout prédicat associé à la classe doit être vérifié . les tables datearriv et cobconst ont même domaine . toute opération déclenchée l'est à la suite du changement d'état d'un objet	$COND_i$: PREDICAT(COB) FAC_i : FACTEUR(COB) preg : $\mathfrak{P}$ (PREDICAT(CEVENT)) NCEV = <x1 : ENTIER>	invariants cev : CEVENT ⟺ . ∀ p ∈ preg, APPL(p,cev) = vrai . DOM(cobconst de cev) = DOM(datearriv de cev) . ∪ DOM(h_{11} de xi de copd de cev) ⊂ DOM(cobconst de cev)
. ajouter une occurrence d'événement c'est valoriser une nouvelle date d'arrivée et une nouvelle occurrence d'objet constatée ; donc changer le produit cartésien cev en cev'	ADJEV : fonct(CEVENT, NCEV, DATE,COB) CEVENT cev : CEVENT ncev : NCEV d : DATE cob : COB	opérations . ADJEV(cev,ncev,d,cob) = cev' cev' = CHPROD(cev",datearriv,dar') dar' = AJOUT(dar,ncev,d) dar = datearriv de cev cev" = AJOUT(cobc,ncev,cob) cobc = conbconst de cev
. modifier le texte d'un prédicat c'est ajouter une occurrence à l'historique des textes "prédicat"	MODTPRED: fonct(CEVENT,DATE, PREDICAT)CEVENT p : PREDICAT(COB) d : DATE	. MODTPRED(cev,d,p) = cev' cev' = CHPROD(cev,prédicat,pred') pred' = AJOUT(pred,d,p) pred = prédicat de cev

lexique	profil	définition formelle
. obtenir le texte d'un prédicat valide à l'instant t directement inférieur à une date d donnée	PRED : fonct(CEVENT,DATE) PREDICAT d,t,date : DATE	. PRED(cev,d) = predic predic = prédicat de cev [t] t = MAX {date ∈ DOM(prédicat de cev) / date ⩽ d}
. déclenchement d'une opération par un événement c'est valoriser une ligne de h_{i1} dans un xi donné (opération AJOUT) cette opération implique le changement successif des trois produits cartésiens de la hiérarchie du type	DECLEX : fonct(CEVENT,NCEV, IDENT,COP_i) CEVENT ncev : NCEV cop : COP_i xi : IDENT	. DECLEX(cev,ncev,xi,cop) = cev' cev' = CHPROD(cev,copd,copd') copd' = CHPROD(p,xi,p') p' = CHPROD(t,h_{i1},t') t = AJOUT(t1,ncev,cop) t1 = h_{i1} de xi de copd de cev t = xi de copd de cev p = copd de cev
. modifier le texte d'une condition de déclenchement c'est ajouter une occurrence à l'historique h_{i2} du xi donné	MODTCOND : fonct(CEVENT, IDENT,DATE,$COND_i$) CEVENT c : $COND_i$ d : DATE	. MODTCOND(cev,xi,d,c) = cev' cev' = CHPROD(cev,copd,copd') copd' = CHPROD(p,xi,p') p' = CHPROD(t,h_{i2},t') t' = AJOUT(t2,d,c) t2 = h_{i2} de xi de copd de cev t = xi de copd de cev p = copd de cev

lexique	profil	définition formelle
. obtenir le texte d'une condition de déclenchement valide à l'instant t le plus proche de d donnée	COND : fonct(CEVENT,IDENT, DATE) PREDICAT t, date, d : DATE	. COND(cev,xi,d) = condition condition = h_{i2} de xi de copd de cev [t] t = MAX { date ∈ DOM / h_{i2} de xi de copd de cev) / date ⩽ d }
. modifier le texte d'un facteur de déclenchement c'est ajouter une occurrence à l'historique h_{i3} du xi donné	MODTFAC : fonct(CEVENT,IDENT, DATE,FAC_i) CEVENT f : FAC_i	. MODTFAC(cev,xi,d,f) = cev' cev' = CHPROD(cev,copd,copd') copd' = CHPROD(p,xi,p') p' = CHPROD(t,h_{i3},t') t' = AJOUT(t3,d,f) t3 = h_{i3} de xi de copd de cev p = copd de cev t = xi de copd de cev
. obtenir le texte d'un facteur valide à une date d donnée	FAC : fonct(CEVENT, IDENT,DATE) FACTEUR t,date,d : DATE	. FAC(cev,xi,d) = facteur facteur = h_{i3} de xi de copd de cev [t] t = MAX {date ∈ DOM(h_{i3} de xi de copd de cev / date ⩽ d}

E. Le type SI

Ce qui nous intéresse, en fait, pour définir un schéma conceptuel de SI, et construire le SI associé particulier, ce n'est pas l'ensemble des classes d'objets, d'opérations et d'événements que l'on peut trouver dans la réalité de l'organisation mais la partie d'entre elles qui constitue le SI utile. Un SI particulier est alors un ensemble cohérent et complet d'éléments de types classes d'objets, d'opérations et d'événements vérifiant un certain nombre de contraintes d'intégrité ; ou encore un ensemble consistant d'objets, occurrences d'instanciations des schémas types paramétrés CLASSE-OB, COPERATION et CEVENT introduits précédemment.

Sur cet ensemble on peut définir, sous forme d'invariants sur les occurrences des types instanciés, et de contraintes sur les paramètres, c'est-à-dire sur les instanciations de schémas, un certain nombre de contrôles de cohérence.

Nous définissons, de plus, outre l'opération qui permet de connaître l'état valide de l'organisation à une date "d", l'opération DECLE de gestion de la dynamique d'un Système d'Informations au cours du temps.

Cette opération exprime le fonctionnement dynamique causal d'un SI. Le changement d'état d'un aspect d'une occurrence de classe d'objets (notons la "cob") peut correspondre à une occurrence de classe d'événements ("cev_i" par exemple). Pour trouver cette classe on recherche, dans l'ensemble des classe de type CEVENT, celles qui admettent "cob" comme paramètre et on évalue leur prédicat (obtenu par l'opération PRED).

Ce "cev_i" peut avoir pour conséquence l'exécution d'occurrences d'opérations appartenant aux classes d'opérations op_{i1} ... op_{in}.

Ces op_{ij} sont présentes dans la définition de "cev_i" ; pour savoir lesquelles ont des occurrences à exécuter effectivement, on évalue, pour chacune, leur condition de déclenchement (obtenue par COND). L'évaluation des facteurs (obtenus par l'opération FAC) puis des textes des classes d'opération "déclenchables" conduit à la définition de nouvelles occurrences de c-objets et donc à un nouvel état du SI.

Les contraintes sur les paramètres ainsi que l'opération de gestion de la dynamique nous conduisent à utiliser, non seulement le SI lui même, mais aussi son schéma conceptuel, ce que nous avons appelé [FOU 82] la méta-base du SI. O. FOUCAUT avait choisi, dans sa thèse, une solution relationnelle lui faisant correspondre une implémentation SOCRATE dont les occurrences d'entité étaient le schéma conceptuel du SI considéré. Nous même, dans [THI 82], avions choisi une solution relationnelle organisée autour du SGBD SYNTEX [DEM 75].

Sans préconiser de la nature de l'implémentation donnée à la métabase de SI, ici nous présentons une solution où un SI est alors un produit cartésien constitué d'une part d'un ensemble d'éléments, occurrences de schémas de types CLASSEOB, COPERATION et CEVENT instanciés, liées par des contraintes d'intégrités (CI) sous forme de prédicats, d'autre part de son schéma conceptuel, à proprement parler, sous forme de tables liant les différents schémas instanciés entre eux.

Ainsi l'opération DECLE de gestion de la dynamique sera décrite au moyen des opérations PRED, COND, FAC précédemment définies et des fonctions CONST, OPERA et MOD que nous allons introduire sur ces tables.

Supposons qu'un SI soit constitué d'une part, de kcob classes d'objets, kcop classes d'opérations, kcev classes d'événements, complétées par un ensemble de prédicats définissant les CI entre éléments définis traditionnellement sur un SI, d'autre part par un ensemble de tables.

a) classes d'objets

Notons predicC_α l'ensemble des prédicats associés à une c-classe α : {predcC_α, predv$_{1_\alpha}C_\alpha$, ..., predv$_{k_\alpha}C_\alpha$, predcC_α} par simplification d'écriture ; chaque classe cob est de la forme :

cob1 = CLASSE-OB(CLEC$_1$, comppC$_1$, COMPC$_1$, compvC$_1$, COMPVC$_1$, datesupC$_1$, predicC$_1$)

cob2 = CLASSE-OB(CLEC$_2$, comppC$_2$, COMPC$_2$, compvC$_2$, COMPVC$_2$, datesupC$_2$, predicC$_2$)

le terme général cob$_\alpha$ s'écrivant :

cob$_\alpha$ = CLASSE-OB(CLEC$_\alpha$, comppC$_\alpha$, COMPC$_\alpha$, compvC$_\alpha$, COMPVC$_\alpha$, datesupC$_\alpha$, predicC$_\alpha$)

et le terme final :

cob$_{kcob}$ = CLASSE-OB(CLEC$_{kcob}$, comppC$_{kcob}$, COMPC$_{kcob}$, compvC$_{kcob}$, COMPVC$_{kcob}$, datesupC$_{kcob}$, predicC$_{kcob}$)

On peut alors définir le type CLAS, qui représente l'ensemble des classes d'objets d'un SI particulier, sous forme d'un produit cartésien des classes précédentes.

CLAS = <Cr$_1$: cob$_1$, Cr$_2$: cob$_2$, ..., Cr$_\alpha$: cob$_\alpha$, ..., Cr$_{kcob}$: cob$_{kcob}$>

que l'on peut, par convention d'écriture, noter ainsi :

$$\text{CLAS} = \text{PRODUITCAR}\ (Cr_\alpha : cob_\alpha)\quad \alpha = 1, kcob$$

En résumé, CLAS représente un produit cartésien d'éléments de type classe d'objets, chacun étant lui même défini comme un produit cartésien d'un SCHEMAP, d'un SCHEMAV et d'une TABLE de dates de suppression.

b) classes d'opérations :

Chaque cop appartenant au SI est de la forme :

cop1 =

COPERATION(NCOPO1,datexecO1,cobmodO1,COB1O1,COB2O1,topO1,pregO1)

cop2 =

COPERATION(NCOPO2,datexecO2,cobmodO2,COB1O2,COB2O2,topO2,pregO2)

.....

le terme général s'écrivant :

cop_β = COPERATION($NCOPO_\beta$, $datexecO_\beta$, $cobmodO_\beta$, $COB1O_\beta$, $COB2O_\beta$, $topO_\beta$, $pregO_\beta$)

et le terme final :

cop_{kcop} = COPERATION($NCOPO_{kcop}$,$datexecO_{kcop}$,$cobmodO_{kcop}$,$COB1O_{kcop}$, $COB2O_{kcop}$,$topO_{kcop}$,$pregO_{kcop}$)

On en déduit COP l'ensemble des classes d'opérations du SI :

$COP = \langle Or_1 : cop_1, Or_2 : cop_2, \ldots, Or_\beta : cop_\beta, \ldots, Or_{kcop} : cop_{kcop} \rangle$

ou encore selon la convention d'écriture introduite précédemment :

$$COP = PRODUITCAR(Or_\beta : cop_\beta)_{\beta = 1,\ kcop}$$

c) classes d'événements :

Chaque cev du SI est de la forme :

cev1 = CEVENT(NCEV1,datearrivE1,COBE1,prédicatE1,cobconstE1,copdE1, DECE1,pregE1)

cev2 = CEVENT(NCEV2,datearrivE2,COBE2,prédicatE2,cobconstE2,copdE2, DEC2,pregE2)

.....

le terme général s'écrivant :

cev_γ = CEVENT($NCEVE_\gamma$,$datearrivE_\gamma$,$COBE_\gamma$,$prédicatE_\gamma$,$cobconstE_\gamma$,$copdE_\gamma$, $DECE_\gamma$,$pregE_\gamma$)

et le terme final :

cev_{kcev} = CEVENT($NCEVE_{kcev}$, $datearrivE_{kcev}$, $COBE_{kcev}$, $prédicatE_{kcev}$, $cobconstE_{kcev}$, $copdE_{kcev}$, $DECE_{kcev}$, $pregE_{kcev}$)

On en déduit CEV l'ensemble des classes d'événements du SI :

CEV = $<Er_1 : cev_1, Er_2 : cev_2, \ldots, er_\gamma : cev_\gamma, \ldots, Er_{kcev} : cev_{kcev}>$

ou encore :

$$CEV = PRODUITCAR(Er_\gamma : cev_\gamma) \quad \gamma = 1, kcev$$

d) les tables

Le but de l'introduction de ces tables est de définir des sous-ensembles de types appartenant à un SI particulier liés entre eux, c'est-à-dire ne plus considérer le SI comme un ensemble de "données" au sens large du terme mais passer à un niveau "méta".

Nous définissons trois tables principales, une par classe de types que l'on peut trouver dans un SI.

(1) TNOMTYPE-OB : TABLE [UNION (cob_α), IDENT]
$cob_\alpha \in$ CLAS de si

où "si" représente un SI particulier, cob_α une classe d'objet lui appartenant.

Cette table représente la table de tous les identificateurs associés aux types instanciés de classes d'objets d'un SI, avec la définition suivante :

$\forall$ cob $\in$ CLAS de si, $\forall$ $cob_{\alpha 1}$, $cob_{\alpha 2}$: cob_α

$\Rightarrow$ APP(TNOMTYPE-OB, $cob_{\alpha 1}$) = APP(TNOMTYPE-OB, $cob_{\alpha 2}$)

c'est-à-dire deux occurrences $cob_{\alpha 1}$, $cob_{\alpha 2}$ d'une même classe d'objets cob_α d'un "si" considéré ont même identificateur de type.

En fait pour simplifier on donnera comme identificateur celui de la classe elle même "cob_α".

exemple : deux occurrences "client DURAND" et "client DUPONT" de la classe "CLIENT" ont même identificateur de type "CLIENT".

Ceci a pour principal intérêt de pouvoir déterminer en particulier lorsqu'une occurrence de c-objet a changé d'état dans un SI, à quel type de schéma de classe elle appartient, le relai étant assuré par le nom du type lui même ainsi qu'il est noté dans le schéma conceptuel du système d'information.

En fait cette table ne suffit pas à l'expression de la partie du schéma conceptuel correspondant aux classes d'objet. Il nous faut l'affiner par l'introduction d'une nouvelle table qui détermine pour une classe d'objets donnée les identificateurs des schémas permanents et variables lui appartenant, ceci pour respecter la définition hiérarchique du schéma de type CLASSE-OB.

TNOMTYPE-SCHEMA : TABLE $\quad [\ \underset{cob_\alpha \in CLAS\ de\ si}{UNION}\ (cob_\alpha)\ ,\ \mathcal{P}(IDENT)\ ]$

définie par $\forall\ cob_\alpha \in$ CLAS de si, $\forall\ cob_{\alpha 1}, cob_{\alpha 2} : cob_\alpha$

$\Rightarrow$ APP(TNOMTYPE-SCHEMA,$cob_{\alpha 1}$) = APP(TNOMTYPE-SCHEMA,$cob_{\alpha 2}$)

c'est-à-dire deux occurrences $cob_{\alpha 1}$, $cob_{\alpha 2}$ d'une même classe d'objets cob_α d'un "si" considéré ont mêmes noms de sous-schémas permanent et variables. Pour simplifier on donnera comme identificateurs ceux introduits lors de la définition de la classe.

exemple : deux occurrences "client DURAND" et "client DUPONT" de la classe "CLIENT" ont pour même identificateur de schéma permanent "CLIENTPERM" et pour mêmes identificateurs de sous-schémas variables "CLIENTCA" (chiffres d'affaire successifs) et "CLIENTADR" (adresses et numéros de téléphone successifs).

Ceci nous est indispensable pour définir, en particulier, le processeur de gestion de la dynamique car ce n'est pas une classe d'objets dans son entier qui change d'état mais un de ses sous-schémas, dont il faut connaître le type.

(2) TNOMTYPE-OP : TABLE [UNION (cop), IDENT]
$cop_\beta \in$ COP de si

Elle représente la table de tous les identificateurs associés aux types instanciés de classes d'opérations d'un SI, avec la définition suivante :

$\forall cop_\beta \in$ COP de si, $\forall cop_{\beta 1}, cop_{\beta 2} : COP_\beta$

$\Rightarrow$ APP(TNOMTYPE-OP, $cop_{\beta 1}$) = APP(TNOMTYPE-OP, $cop_{\beta 2}$)

c'est-à-dire deux occurrences $cop_{\beta 1}$, $cop_{\beta 2}$ d'une même classe d'opérations cop_β d'un "si" considéré ont même identificateur. En fait pour simplifier on donnera comme identificateur celui de la classe elle même "cop_β".

exemple : deux occurrences successives du "calcul de la paie" en janvier et février 1984 ont même identificateur de type "PAIE".
Ceci a pour principal intérêt de reconnaître, pour une occurrence d'opération donnée, à quel type de schéma de classe elle appartient ; le relai étant assuré par le nom du type lui même ainsi qu'il est noté dans le schéma conceptuel du SI.

(3) TNOMTYPE-EV : TABLE [UNION (cev_γ), IDENT]
$cev_\gamma \in$ CEV de si

Elle représente la table de tous les identificateurs associés aux types instanciés de classes d'événements d'un SI, avec la définition suivante :

$\forall cev_\gamma \in$ CEV de si, $\forall cev_{\gamma 1}, cev_{\gamma 2} : cev_\gamma$

$\Rightarrow$ APP(TNOMTYPE-EV, $cev_{\gamma 1}$) = APP(TNOMTYPE-EV, $cev_{\gamma 2}$)

c'est-à-dire deux occurrences $cev_{\gamma 1}$, $cev_{\gamma 2}$ d'une même classe d'événements cev_γ d'un "si" considéré ont même identificateur. En fait pour simplifier on donnera comme identificateur celui de la classe "cev_γ" elle même.

exemple : deux occurrences successives d'événements "arrivée d'une commande" ont même identificateur de type "ARRCOM".

Ceci a pour principal intérêt de reconnaître, pour une occur.ence d'événement donnée, à quel type de schéma de classe elle appartient ; le relai étant assuré par le nom du type lui même ainsi qu'il est noté dans le schéma conceptuel du SI.

L'étape suivante, nécessaire à la description des contraintes sur les paramètres du SI et à la spécification du fonctionnement du processeur de la dynamique, est de définir des liaisons entre tous ces identificateurs de types. Nous le réalisons par la construction de nouvelles sous-tables associées aux tables précédentes, toujours dans le cadre d'un SI particulier, l'ensemble de ces différentes tables constituant la description du schéma conceptuel du SI.

(4) T-CONST : TABLE $[\cup\ CEV_\gamma\ ,\ IDENT]$
$cev_\gamma \in CEV$ de si

Elle représente la table des identificateurs de c-objets dont le changement d'état est constaté par les événements du "si" considéré, avec la définition suivante :

$\forall$ cev $\in$ CEV de si $\exists\ cob_\alpha \in$ CLAS de si t.q.

$\forall\ cev_{\gamma 1}, cev_{\gamma 2} : cev_\gamma \implies$ APP(T-CONST, $cev_{\gamma 1}$) =

APP(T-CONST, $cev_{\gamma 2}$) = "$cobjet_\alpha$"

où $cobjet_\alpha \in$ APP(TNOMTYPE-SCHEMA, cob_α)

soit : deux occurrences $cev_{\gamma 1}$, $cev_{\gamma 2}$ d'une même classe d'événement cev_γ d'un "si" considéré constatent le changement d'état d'objets appartenant à un même type de sous-schéma noté "$cobjet_\alpha$", d'une même classe d'objets "cob_α".

On en déduit une fonction particulière :

$\forall$ $cev_{\gamma i}$: cev_{γ} $\Rightarrow$ CONST($cev_{\gamma i}$) = APP(T-CONST, $cev_{\gamma i}$)

CONST rend pour un événement donné le type de l'objet dont il constate le changement d'état.
Ceci est indispensable, en particulier, pour la première étape du processeur de la dynamique qui est de trouver, lors du changement d'état d'un objet, si un type d'événement peut lui être associé.

(5) T-MOD : TABLE [UNION (cop_{β}), IDENT]
COP de si

De façon symétrique à la précédente, elle représente la table des identificateurs, ou types de c-objets, qui sont modifiés par des opérations d'un "si" donné, avec la définition suivante :

$\forall$ cop_{β} $\in$ COP de si, $\exists$ cob_{α} $\in$ CLAS de si t.q.

$\forall$ $cop_{\beta 1}$, $cop_{\beta 2}$: cop $\Rightarrow$ APP(T-MOD, $cop_{\beta 1}$) = APP(T-MOD, $cop_{\beta 2}$)
= "$cobjet_{\alpha}$"

où $cobjet_{\alpha}$ $\in$ APP(TNOMTYPE-SCHEMA, cob_{α})

Soit : deux occurrences $cop_{\beta 1}$, $cop_{\beta 2}$ d'une même classe d'opérations cop_{β} d'un "si" considéré modifient des objets appartenant à un même type de sous-schéma noté "$cobjet_{\alpha}$", d'une même classe d'objets "cob_{α}".

On en déduit une nouvelle fonction :

$\forall$ $cop_{\beta j}$: cop_{β} $\Rightarrow$ MOD($cop_{\beta j}$) = APP(T-MOD, $cop_{\beta j}$)

MOD rend, pour une c-opération donnée, le type de l'objet qu'elle modifie.
Ceci est indispensable, en particulier, pour l'expression de l'étape de mise à jour du SI pour le processeur de la dynamique.

(6) T-OPERA : TABLE [UNION (cev_γ), $\mathfrak{P}$ (IDENT)]
CEV de si

Elle représente la table des types d'opérations susceptibles d'être déclenchées par des classes d'événements du SI, avec la définition suivante :

$\forall\ cev_\gamma \in$ CEV de si $\exists cop_1, \ldots, cop_\beta, \ldots, cop_n \in$ COP de si t.q.

$\forall\ cev_{\gamma 1}, cev_{\gamma 2} : cev_\gamma \Rightarrow$ APP(T-OPERA, $cev_{\gamma 1}$) = APP(T-OPERA, $cev_{\gamma 2}$)

Soit : deux occurrences $cev_{\gamma 1}$, $cev_{\gamma 2}$ d'une même classe d'événements cev_γ d'un "si" considéré sont susceptibles de déclencher des opérations appartenant aux mêmes classes d'opérations $cop_1, \ldots, cop_\beta, \ldots, cop_n$.

On en déduit une dernière fonction :

$\forall\ cev_{\gamma i} : cev_\gamma \Rightarrow$ OPERA($cev_{\gamma i}$) = APP(T-OPERA, $cev_{\gamma i}$)

OPERA rend pour un type d'événement donné, les types d' opérations qu'il peut déclencher.

Ceci est indispensable, en particulier, pour l'expression de la deuxième étape du processeur de la dynamique qui est de rechercher pour un événement donné quelles sont les opérations à déclencher.

e) les prédicats :

Ces prédicats permettent d'exprimer les contraintes d'intégrité définies sur le SI entre deux objets de types quelconques.
Appelons UT la réunion de tous les types du SI ; soit si : SI un système d'information donné, soit cob_α classe de c-objets de CLAS de si, cop_β copération de COP de si, cev_γ c-événement de CEV de si,
alors

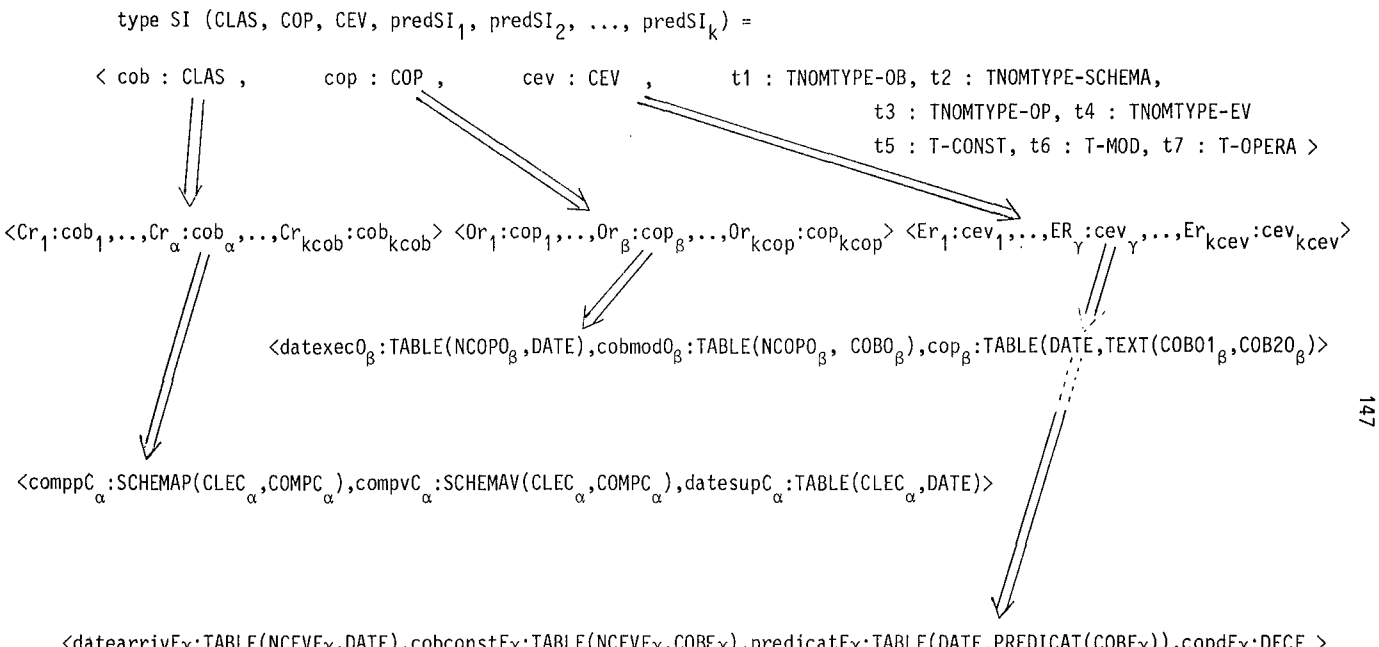
$$UT = UNION(UNION(cob_\alpha), UNION(cop_\beta), UNION(cev_\gamma))$$
$$\alpha=1,kcob \quad \beta=1,kcop \quad \gamma=1,kcev$$

Alors un type si, instanciation du type SI, est caractérisé par un ensemble de prédicats de la forme :

pred SI_1 = PREDICAT (<r_{11} : UT, r_{12} : UT>)
pred SI_2 = PREDICAT (<r_{21} : UT, r_{22} : UT>)
.....
pred SI_k = PREDICAT (<r_{k1} : UT, r_{k2} : UT>)

Ces prédicats seront des paramètres du constructeur de type SI.

La figure suivante donne la schématisation hiérarchique du type SI.



Représentation graphique du type SI (en se limitant à deux niveaux de hiérarchisation, la décomposition des types CLASSE-OB, COPERATION, CEVENT étant donnée précédemment)

lexique	profil	définition formelle
le type SI est formé d'un ensemble de classes complété par un ensemble de CI et de tables explicitant le schéma conceptuel		type SI (CLAS, COP, CEV, $predSI_1$, ..., $predSI_k$) = <COB : CLAS, cop : COP, cev : CEV, t_1 : TNOMTYPE-OB, t_2 : TNOMTYPE-SCHEMA t_3 : TNOMTYPE-OP, t_4 : TNOMTYPE-EV, t_5 : T-CONST, t6 : T-MOD, t7 : T-OPERA>
. estype T_i est une opération du type UNION qui donne le type d'un constituant . de est l'opérateur projection du produit cartésien . tout prédicat du si doit être vérifié ; entre les relations rj_1 de type T_1 et rj_2 de type T_2	si : SI $estypeT_i$: fonct(T) BOOL T = UNION (T_1,T_2) de : fonct(PRODCAR) T avec xi de T ≡ $PROJ_{xi}$(T)	invariant $\bigwedge_{j=1}^{k}$ [estype T_1(rj_1 de pred SI_j) et estype T_2 (rj_2 de pred SI_j) ⇒ APPL($predSI_j$ (T_1 de si), (T_2 de si)) = vrai]
. tout type d'événement constate le changement d'état d'un c-objet	CONST : fonct(T-CONST,CEV) IDENT	contraintes sur les paramètres si : SI . ∀ γ ∈ [1..kcev], ∃ α ∈ [1..kcob] t.q. CONST(cev_γ de cev de si) ∈ APP(T-NOM-SCHEMA, cob_α)
. tout type d'opération modifie un c-objet	MOD : fonct(T-MOD,COP) IDENT	. ∀ β ∈ [1..kcop], ∃ α ∈ [1..kcob] t.q. MOD(cop_β de cop de si) ∈ APP(TNOM-SCHEMA, cob_α)

texte	profil	définition formelle
. tout type d'opération est déclenché par au moins un type d'événement	OPERA : fonct(T-OPERA,COP) $\mathcal{P}$ (IDENT)	. $\forall \beta \in [1..kcop]$, $\exists \gamma \in [1..kcev]$ t.q. cop_β de cop de si $\in$ OPERA(cev_γ de cev de si)
. opération qui, à un SI donné, à un instant donné, associe son état défini comme la réunion des états valides des classes d'objets qui le composent	si : SI d : DATE ETATSI : fonct(SI,DATE) $\mathcal{P}$(CLAS) valides : fonct(CLASSE-OB, DATE) $\mathcal{P}$ (CLASSE-OB)	opérations . ETATSI(si,d) = $\bigcup_{\alpha=1,kcob}$ valides(cob_α de cob de si, d)
. nuple est la valeur du c-objet (de la classe cob) qui a changé d'état, il est le paramètre du facteur de déclenchement ; DECLE exprime le déclenchement des opérations à la suite d'un événement dans un SI donné et rend un nouvel SI ; on applique FAC du couple $\{cev_i, xj\}$ considéré au nuple paramètre afin d'obtenir les nuples paramètres du texte obtenu par la fonction TEXTOP. L'ensemble des indices I est donné par l'événement lié au changement d'état de cob dont le prédicat est vérifié ; J est, pour l'ensemble I précédent, l'ensemble des opérations (données par le champ xj de cev_i) à déclencher à la suite de l'événement ;	DECLE : fonct (SI, IDENT, DATE, CLAS) SI	. DECLE(si,cob,d,nuple) = $\bigcup_{i,j,k}$ APPL(TEXTOP(cop_{ij},d), APPL(FAC_k(cev_i,xj,d),nuple)) I : {$cev_i \in$ cev de si t.q. APPL(CONST(cev_i), APP(TNOM TYPE-SCHEMA,cob)) = "vrai" et APPL(PRED(cev_i,d),nuple) = "vrai"} $\forall i \in I, J$: {$cop_{ij} \in$ OPERA(cev_i) t.q. APPL(COND(cev_i,xj,d),nuple) = "vrai"}

lexique	profil	définition formelle
K, pour I et J donné, donne l'ensemble des facteurs à appliquer à nuple pour avoir les paramètres successifs des textes des opérations ; xj est le champ dans le produit cartésien cev_i qui correspond à la cop_{ij} concernée		$\forall i \in I, j \in J$ K : {FAC(cev_i,xj,d)}

3. CONCLUSION

La formalisation proposée repose sur une algèbre de types paramétrés permettant de construire les schémas du niveau 4 (cf. figure 1). Ces schémas de type permettent la spécification d'un SI : ce sont les constructeurs
CLASSE-OB (expression d'une classe d'objets du monde réel),
COPERATION (expression d'une classe d'opérations),
CEVENT (expression d'une classe d'événements et de ses conséquences),
et SI (expression de la composition d'un SI particulier).

Par rapport à la modélisation proposée jusqu'à présent dans le projet REMORA [FOU 82], nous pensons que cette formalisation apporte trois types d'avantages :

- au point de vue de la simplification de la présentation du modèle,
- au point de vue de la complétude d'expression,
- au point de vue de la spécification intégrée modèle/outil.

(i) simplification de la présentation du modèle :

La présentation précédente a mis en évidence les seuls trois concepts de base du modèle descriptif de REMORA : les classes d'objets, les classes d'opérations et les classes d'événements.
Dans la formulation relationnelle, explicitée dans [FOU 82], ces trois concepts n'apparaissent, en fait, plus clairement ; en effet, la normalisation, fondée sur le concept de dépendance fonctionnelle permanente, oblige à introduire, par décomposition des relations, de multiples types de relation (neuf en tout). En conséquence il y a confusion, dans la présentation du modèle conceptuel, entre les trois concepts "noyaux" de la formalisation et les concepts introduits pour satisfaire les contraintes du modèle relationnel. Il en résulte une apparente complexité, critiquée d'ailleurs dans CRIS2 [IFI 83].

(ii) complétude de l'expression de la richesse du modèle

Le modèle REMORA est un modèle riche puisqu'il permet l'expression complète, dans ses aspects statiques et dynamiques, du fonctionnement du système ; il a une puissance de modélisation égale à celle des langages de programmation les plus typés. La formalisation relationnelle ne permet pas d'exprimer la modélisation de la cohérence de la spécification : la collection des relations exprimée à l'aide des concepts du modèle est complétée par un ensemble de contraintes d'intégrité ; ce que O. FOUCAUT appelle le "système d'intégrité" est un ensemble de douze règles de cohérence du schéma conceptuel (cf. [FOU 82] chapitre 4).

La spécification, à l'aide des quatre schémas CLASSE-OB, COPERATION, CEVENT et SI, permet l'expression de la cohérence intégrée à la structure même de la modélisation :

- soit parce que la composition du type hiérarchique implique la description intégrée des concepts : par exemple dans le schéma CEVENT sont décrites les conséquences en termes de classes d'opérations à déclencher ;
- soit par la définition même du type SI : en effet ce constructeur a été introduit pour exprimer la composition d'un SI particulier. Les instanciations de ce schéma doivent obéir à un certain nombre de règles introduites comme des "contraintes sur les paramètres" qui induisent la cohérence de la spécification.

Ce type permet de plus une description intégrée du SI lui même et de sa structure. L'ensemble des classes d'associations introduites dans [FOU 82] pour définir les relations existant entre les trois concepts de base sont exprimées ici sous forme d'un ensemble de tables faisant partie intégrante du type SI ; les applications complétant la description de la structure du SI étant représentées ici par un ensemble de fonctions sur ces tables.

(iii) spécification intégrée du modèle et de l'outil

L'outil noyau de l'interprétation d'une spécification, réalisée au moyen des concepts précédents, correspond à l'implémentation de la gestion de l'évolution du SI au cours du temps.
Ce que O. FOUCAUT, définit dans la classe d'association DECLENCHE comme une application particulière, est introduit directement par l'opération DECLE du schéma de type SI.

En résumé, la formalisation à l'aide des Types Abstraits nous semble à la fois plus riche et plus claire que la formulation relationnelle. Nous pensons avoir fortement clarifié les idées par nos propositions.

La spécification formelle obtenue est, par définition, cohérente et minimale. Il reste à donner les moyens de sa mise en oeuvre afin d'expliciter le niveau 4 "Application SI effectif".

La chapitre suivant, consacré au langage LASSIF, apporte une première pierre à cet édifice.

ANNEXE A

Rappels théoriques sur les Types Abstraits Algébriques

Nous avons choisi une définition axiomatique, algébrique des types abstraits, gardant une spécification pré/post pour les invariants. C'est, en effet, l'approche algébrique qui a connu les développements les plus importants ces dernières années.
Les TAA sont la base des travaux actuels sur la représentation des TA, les systèmes de réécriture, les problèmes de preuves, de complétude et de consistance. On les qualifie d'algébriques [GUT 78, GOG 77, GTW 78, GUT 80] car les objets et les opérations apparaissant dans un type peuvent être considérés comme définissant une algèbre abstraite.

Nous allons rappeler ici quelques notions fondamentales sur les TAA nécessaires à notre développement. Cette présentation est le résultat d'un consensus sur les TAA largement inspiré de [LEV 84], lui même obtenu à partir des travaux de [REM 82] et [BID 81].

1. Définitions de base

Signature

Une signature $\langle S,\Sigma \rangle$ est la donnée d'un ensemble S de domaines (ou types) appelés "sortes" et d'une famille de symboles d'opérations $\Sigma = (\Sigma_{w,s})$, $w \in S^*$, $s \in S$, ou S^* est l'ensemble des suites finies sur S.

Le profil d'un symbole précise le type des arguments (domaines) et sa sorte résultat (co-domaine) :

$\forall f \in \Sigma_{w,s}$ profil(f) = (w,s) noté $f : w \to s$
domaine(f) = w
sorte(f) = s

Exemple de signature : sorte des BOOLEEN

$S = \{BOOLEEN\}$

$\Sigma = \{vrai, faux, non, et, ou, eq, impl\}$

avec les profils

vrai : $\rightarrow$ BOOLEEN

faux : $\rightarrow$ BOOLEEN

non : BOOLEEN $\rightarrow$ BOOLEEN

et : BOOLEEN x BOOLEEN $\rightarrow$ BOOLEEN

ou : BOOLEEN x BOOLEEN $\rightarrow$ BOOLEEN

eq : BOOLEEN x BOOLEEN $\rightarrow$ BOOLEEN

impl : BOOLEEN x BOOLEEN $\rightarrow$ BOOLEEN

Σ - algèbre

Soit $<S,\Sigma>$ une signature. Une $<S,\Sigma>$-algèbre ou Σ-algèbre est la donnée :

- d'une famille d'ensembles, indicée par S : $A = (A_s)_{s \in S}$ appelée support de l'algèbre et
- d'une famille d'opérations dans A indicée par Σ :

$\Sigma_A = (f^A)_{f \in \Sigma}$ telle que si $f : s_1 \ldots s_n \rightarrow s$ alors

$f^A : A_{s1} \ldots A_{sn} \rightarrow A_s$

Morphisme d'algèbres ; algèbre initiale

Un morphisme d'algèbres h est une famille d'applications $(h_s)_{s \in S}$ entre les supports $A = (A_s)_{s \in S}$ et $B = (B_s)_{s \in S}$ de deux (S,Σ)-algèbres (A,Σ_A) et (B,Σ_B), vérifiant la propriété suivante :

$\forall f : s_1,\ldots,s_n \rightarrow s$ de Σ, $\forall x_1 \in A_{s_1},\ldots, x_n \in A_{sn}$

$h_s(f^A(x_1 \ldots x_n)) = f^B(h_{s_1}(x_1),\ldots h_{sn}(x_n))$

Si h est une famille de bijections alors h est un isomorphisme de (A,Σ_A) dans (B,Σ_B).

La relation "il existe un unique morphisme h de (A,Σ_A) vers (B,Σ_B)" définit un préordre sur la classe des (S, Σ)-algèbres. L'algèbre minimale pour cette relation, unique à un isomorphisme près, est appelée l'algèbre initiale.

Algèbre des termes

L'algèbre des termes sur la signature $<S,\Sigma>$ est notée $T_{<S,\Sigma>}$ ou T_Σ. Son support noté support (T_Σ) (abrégé en T_Σ) est défini par :

- $\forall s \in S \quad \Sigma_{\wedge,s} \subset \text{support}(T_\Sigma)$
- $\forall f \in \Sigma_{s_1..s_n,s} \quad \forall t_1,...,t_n \text{ support}(T_\Sigma)$ tels que sorte$(t_i) = s_i$ pour $i = 1...n$ (où sorte est prolongée aux termes) $f(t_1,...,t_n) \in \text{support}(T_\Sigma)$.

Les opérations de l'algèbre des termes sont telles que le terme associé aux termes $t_1,...,t_n \in \text{support}(T_\Sigma)$, de sortes $s_1...s_n$ respectivement, par l'opération f de $\Sigma_{s_1...s_n,s}$ est $f(t_1...t_n)$.

Propriété :

L'algèbre des termes est initiale dans la classe des $<S,\Sigma>$-algèbres.

Algèbre finement engendrée

Une algèbre (A,Σ_A) est une algèbre finement engendrée si les éléments de son support sont obtenus par composition des opérations. C'est-à-dire si le morphisme h de l'algèbre des termes dans (A, Σ_A) est surjectif.

Remarque :

Si Σ ne contient pas de constantes ($\Sigma_{\wedge,s} = \emptyset$) alors les supports des algèbres finiment engendrées sont vides.

A chaque sorte $s \in S$ on associe un ensemble X_s de variables de sorte s. On note X la famille des ensembles disjoints $(X_s)_{s \in S}$. On définit profil(x) pour tout $x \in X$, comme étant l'unique (s,s) tel que $x \in X_s$.

Algèbre libre des termes

L'algèbre libre des termes sur la signature $\langle S,\Sigma \rangle$ est notée $T_{\langle S,\Sigma \rangle}(X)$ ou $T_\Sigma(X)$. Son support, noté support $(T_\Sigma(X))$ (abrégé en $T_\Sigma(X)$) est défini par :

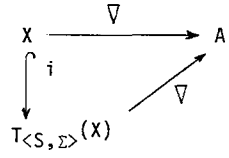
- $\forall s \in S \quad \Sigma_{\Lambda,s} \cup X_s \subset \text{support}(T_\Sigma(X))$
- $\forall f \in \Sigma_{s_1 \ldots s_n,s} \quad \forall t_1 \ldots t_n \in \text{support}(T_\Sigma(X))$
  tels que $\text{sorte}(t_i) = s_i$ pour $i = 1,\ldots,n$
  $f(t_1,\ldots,t_n) \in \text{support}(T_\Sigma(X))$

Les opérations de l'algèbre libre des termes sont telles que le terme avec variables, associé aux termes avec variables $t_1 \ldots t_n \in \text{support}(T_\Sigma(X))$, de sortes $s_1 \ldots s_n$, par l'opération $f \in \Sigma_{s_1 \ldots s_n,s}$, est $f(t_1 \ldots t_n)$.

Le profil d'un terme est le couple formé du domaine du terme et de sa sorte :

$$\forall t \in T_\Sigma(X) \quad \text{profil}(t) = (\text{domaine}(t), \text{sorte}(t))$$
$$\text{domaine}(f(t_1 \ldots t_n)) = \text{domaine}(t_1) \ldots \text{domaine}(t_n)$$
$$\text{sorte}(f(t_1 \ldots t_n)) = \text{sorte}(f)$$

Pour toute $\langle S,\Sigma \rangle$-algèbre (A,Σ_A), une application $\nabla : X \to A$ appelée affectation peut être prolongée de façon unique en un morphisme encore noté ∇ de $T_{\langle S,\Sigma \rangle}(X)$ vers A tel que le diagramme suivant commute : $i : X \to T_{\langle S,\Sigma \rangle}(X)$ est l'injection canonique.



Morphisme de signature

Soit $<S,\Sigma>$ et $<S',\Sigma'>$ deux signatures munies des ensembles de variables typées $X = (X_s)_{s\in S}$ et $X' = (X'_{s'})_{s'\in S'}$. Un morphisme de signature h est un couple d'applications (h_S, h_Σ) où $h_S : S \to S'$ et $h_\Sigma : \Sigma \to T_\Sigma'(X')$ tel que

- pour tout $f \in \Sigma$ tel que profil(f) = $s_1 \ldots s_n, s, h$ est étendu au terme de $T_\Sigma(X)$ par profil($h_\Sigma(f)$) = $h_S(s_1) \ldots h_S(s_n), h_S(s)$.
- pour tout $x \in X$ h(x) est un élément de X' vérifiant :
  sorte(h(x)) = h_S(sorte(x))
- pour tout $f(t_1 \ldots t_n) \in T_\Sigma(X)$
  $h(f(t_1 \ldots t_n)) = h_\Sigma(f)\ (h(t_1) \ldots h(t_n))$
  donc pour tout terme $t \in T_\Sigma(X)$
  profil(h(t)) = (h_S(domaine(t)), h_S(sorte(t))).

Intuitivement un tel morphisme est une application qui associe aux termes de $T_\Sigma(X)$, des termes de $T_{\Sigma'}(X')$ de profil correspondant.

Σ-equations

Une Σ-équation e est un couple de termes g,d de $T_\Sigma(X)$ de même sorte. Elle est notée e = [g = d] ou g = d.

Modèle

Une Σ-algèbre (A,Σ_A) valide une Σ-équation $e = [g=d]$ si et seulement si pour toute application $\nabla : X \to A$, $\nabla(g) = \nabla(d)$. A valide un ensemble E d'équations si et seulement si A valide toutes les équations de E. On dit alors que A est un modèle de E.

Théorie inductive

Soient E un ensemble de Σ-équations et $g = d$ une Σ-équation. On dit que E vérifie $g = d$ et on note $E \vdash\!\!\!- g = d$ si et seulement si $g = d$ est valide dans tout modèle de E.

E vérifie un ensemble E' d'équations et on note $E \vdash\!\!\!- E'$ si et seulement si E vérifie toutes les équations de E'.

L'ensemble des équations vérifiées par E est appelé la théorie inductive définie par E.

Congruence

Soit E un ensemble de Σ-équations. Une relation $\sim_E$ est une congruence compatible avec E si et seulement si

- $\sim_E$ est une relation d'équivalence sur les termes de T_Σ validant les équations de E et
- $\sim_E$ est compatible avec les opérations de Σ. C'est-à-dire
  $\forall f \in \Sigma s_1 \ldots s_n, s$, $\forall t_1 \ldots t_n, t'_1 \ldots t'_n \in T_\Sigma$ tels que
  $\text{sorte}(t_i) = \text{sorte}(t'_i) = s_i$ pour $i = 1 \ldots n$,
  $t_1 \sim_E t'_1, \ldots, t_n \sim_E t'_n \Rightarrow f(t_1,\ldots,t_n) \sim_E f(t'_1,\ldots,t'_n)$.

Algèbre quotient

Soit E un ensemble de Σ-équations.
On note $T_{\Sigma,E}$ l'algèbre quotient de l'algèbre des termes par la plus petite congruence compatible avec E, notée $\equiv_E$: $T_{\Sigma,E} = T_\Sigma/\equiv_E$.
Cette algèbre valide E. Elle est initiale dans la classe des Σ-algèbres validant les équations de E.

Définition d'un TAA

Un TAA est une théorie algébrique typée dont la "présentation", ou la spécification, est composée de trois parties notée $<S,\Sigma,E>$ où

- $<S,\Sigma>$ est une signature, et
- E est un ensemble de Σ-équations.

C'est la classe des Σ-algèbres finement engendrées validant les équations de E et isomorphes à l'algèbre initiale $T_{\Sigma,E}$.

Morphisme de spécifications

Soient $<S,\Sigma,E>$ et $<S',\Sigma',E'>$ deux spécifications. Un morphisme de spécifications h est un morphisme de signatures de $<S,\Sigma>$ vers $<S',\Sigma'>$ munies des ensembles de variables typées X et X' tel que E' vérifie les images des équations de E :

$$\forall\, g = d \in E \quad E' \vdash\!\!\!\!\!\!- \; h(g) = h(d)$$

Propriété

La composition de deux morphismes de spécification est un morphisme de spécification.

Exemple : Complétons l'exemple précédent.

Type BOOLEEN

Opérations

vrai : → BOOLEEN
faux : → BOOLEEN
non : BOOLEEN → BOOLEEN
et : BOOLEEN x BOOLEEN → BOOLEEN
ou : BOOLEEN x BOOLEEN → BOOLEEN
eq : BOOLEEN x BOOLEEN → BOOLEEN
impl : BOOLEEN x BOOLEEN → BOOLEEN

Axiomes

soit b : BOOLEEN
non(vrai) = faux
non(faux) = vrai
vrai et b = b
faux et b = faux
vrai ou b = vrai
faux ou b = b
eq(vrai,b) = b
eq(faux,b) = non(b)
vrai impl b = b
faux impl b = vrai

Remarquons que parmi les opérations de Σ, permettant de "construire" les termes du type, certaines ont un rôle privilégié : ce sont sur elles que s'appliqueront les autres opérations dans les axiomes. On les appelle les "constructeurs primitifs" : ici il s'agit de "vrai" et "faux". On peut alors considérer les axiomes comme des procédures portant sur des objets formés à partir des constructeurs et donner une représentation arborescente de ces objets.

2. Propriétés des spécifications

La construction d'une spécification se fait par étapes successives : partant d'une spécification primitive, on peut rajouter, soit des opérations, on parle alors "d'enrichissements", soit également une nouvelle sorte, on parle alors "d'extensions". A chaque étape on doit s'assurer que le type abstrait, défini à l'étape précédente, n'a pas été modifié soit par adjonction de nouveaux objets non équivalents à ceux définis, soit par modification des relations d'équivalences.

Extension d'une spécification

Une spécification $\langle S, \Sigma, E\rangle$ est une extension d'une spécification $\langle S_0, \Sigma_0, E_0\rangle$ si et seulement si

$$S_0 \subset S,\ \Sigma_0 \subset \Sigma \quad \text{et} \quad E \vdash\!\!\!- E_0$$

Enrichissement d'une spécification

Un enrichissement est une extension laissant l'ensemble des sortes inchangé.

Restriction d'une spécification

Une spécification SPEC est une restriction d'une spécification SPEC' si et seulement si SPEC' est un enrichissement de SPEC.

Spécification structurée

Une spécification structurée (ou hiérarchique [WIR 81, BROY 82]) est définie comme extension d'une spécification primitive SPEC = $\langle S, \Sigma, E\rangle$, appelée son environnement, par le triplet $\langle \{T\}, \Sigma_T, E_T\rangle$ appelé présentation du type T.

T est le type d'intérêt de la spécification $\langle S \cup \{T\},\ \Sigma \cup \Sigma_T,\ E \cup E_T\rangle$ définie.

Une spécification structurée est consistante vis-à-vis de son environnement si la relation d'équivalence sur les termes des types de l'environnement n'a pas été modifiée.

Consistance d'une spécification lors d'une extension

Une spécification SPEC = $\langle S, \Sigma, E\rangle$, extension de la spécification $SPEC_0 = \langle S_0, \Sigma_0, E_0\rangle$ est consistante vis-à-vis de $SPEC_0$ si et seulement si la restriction de $\equiv_E$ aux Σ_0-termes coïncide avec $\equiv_{E_0}$ c'est-à-dire $\forall t_0, t'_0 \in T_{\Sigma 0}$,

$$t_0 \equiv_E t'_0 \Rightarrow t_0 \equiv_{E_0} t'_0$$

La spécification SPEC est consistante sur les variables vis-à-vis de $SPEC_0$ si et seulement si

$$\forall t_0, t'_0 \in T_{\Sigma 0}(X) \quad t_0 \equiv_E t'_0 \Rightarrow t_0 \equiv_{E_0} t'_0$$

Une spécification structurée est suffisamment complète vis-à-vis de son environnement si tout terme de sorte appartenant à l'environnement est équivalent à un terme de l'environnement.

Complétude suffisante lors d'une extension

Une spécification SPEC = $\langle S, \Sigma, E\rangle$, extension de la spécification $SPEC_0 = \langle S_0, \Sigma_0, E_0\rangle$, est suffisamment complète vis-à-vis de $SPEC_0$ si et seulement si

$$\forall t \in T_\Sigma \text{ tq sorte}(t) \in S_0 \quad \exists t_0 \in T_{\Sigma 0}, \ t \equiv_E t_0$$

SPEC est suffisamment complète sur les variables vis-à-vis de $SPEC_0$ si et seulement si

$$\forall t \in T_\Sigma(X_0) \quad \text{tq} \quad \text{sorte}(t) \in S_0$$

(les variables de t sont d'un type de S_0)

$$\exists t_0 \in T_{\Sigma_0}(X_0), \ t \equiv_E t_0.$$

3. Propriétés des TAA

A. Consistance et complétude suffisante d'un TAA

Se poser la question de la consistance d'un TAA c'est se demander si les axiomes ne sont pas contradictoires ou s'ils ne conduisent pas à des contradictions vis-à-vis des spécifications déjà définies. Se poser la question de la complétude d'un TAA c'est se demander si on a donné un nombre suffisant d'axiomes pour décrire toutes les propriétés du TA que l'on voulait spécifier au départ et dont on n'a en général, qu'une idée intuitive.

Si on connaît, avant d'écrire les axiomes, un modèle mathématique standard de ce que l'on veut spécifier, une méthode de vérification est de considérer un modèle (l'algèbre initiale) du TA et de montrer qu'il est isomorphe au modèle standard. Si on ne dispose pas de tel modèle, il faut vérifier la consistance et la complétude du système d'axiomes. Ces deux problèmes sont, en général indécidables [GTW 78]. Cependant il existe des méthodes syntaxiques de spécification assurant la complétude suffisante de la spécification obtenue.

KNUTH-BENDIX [KNU 70] ont proposé un algorithme permettant de vérifier la propriété de CHURCH-ROSSER pour un ensemble d'axiomes.

M. BIDOIT [BID 81] préconise la méthode de présentation gracieuse d'un TAA.

B. Problème des opérations partielles

Pour spécifier un type abstrait consistant et suffisamment complet, il faut définir de façon axiomatique toutes les opérations par rapport aux opérations de base nommées "constructeurs" du type. Certaines opérations ne sont pas définies sur tout le domaine, on dit qu'elles sont partielles.

Exemple : définition du type ENTIER

Type ENTIER

Opérations

zéro : → ENTIER
suc : ENTIER → ENTIER
plus : ENTIER x ENTIER → ENTIER
moins : ENTIER x ENTIER → ENTIER
mult : ENTIER x ENTIER → ENTIER

Axiomes

Soient e,e' : ENTIER

plus(suc(e),e') = suc(plus(e,e'))
plus(zero,zero) = zero
moins(zero,zero) = zero
moins(suc(e),zero) = suc(e)
moins(suc(e),suc(e')) = moins(e,e')
mult(zero,e) = zero
mult(suc(e),e') = plus(mult(e,e'),e')

Remarquons qu'on ne sait pas définir : moins(zero,suc(e)).

Il y a plusieurs types de méthodes pour résoudre ce genre de problèmes : algèbres partielles [BROY 82, WIR 81], algèbres à opérateurs multi-cibles [BOI 83], spécification des erreurs par des E-algèbres [ADJ 78] [GOG 79] [REM 82].

Nous résolvons ce problème, lors de la spécification de nos types, par l'introduction de restrictions c'est-à-dire des pré-conditions limitant l'applicabilité des axiomes et donc l'usage du TAA.

C. Les invariants

Un invariant est une propriété que doivent vérifier tous les objets du type. Un invariant détermine une restriction sur l'ensemble des objets générés par les constructeurs du type.

L'adjonction d'un invariant est réalisée par l'introduction d'une précondition aux équations définissant la construction des objets du type. De cette manière seuls les objets vérifiant l'invariant seront effectivement construits.

Exemple : définir le jour d'un mois à partir des entiers.

invariant soit j : ENTIER

$j \leqslant 31$

ou encore en termes d'opérations sur les entiers

$j \leqslant (suc^{31}(zero))$

Nous définissons ainsi les invariants lors de la spécification de nos types.

D. Type abstrait paramétré

Pour spécifier une structure de suite d'éléments, il n'est pas nécessaire de connaître avec précision de quels éléments il s'agit. En effet les caractéristiques d'une suite sont les mêmes, qu'il s'agisse d'une suite d'entiers ou d'une suite de caractères. Nous avons envie de pouvoir décrire cette structure indépendamment des éléments qui la composent et de substituer lors d'une utilisation au type des "éléments quelconques" le type des éléments effectifs.

Une spécification paramétrée est une spécification structurée comportant, dans son environnement, un type, appelé paramètre formel, dont la spécification ne comprend que les opérations utilisées par le type d'intérêt et pas forcément de constructeurs.

Pour utiliser une spécification paramétrée il faut instancier le type paramètre formel par un type défini : le type SUITE instancié par le type des caractères donnera le type des suites de caractères.

La propriété demandée à une spécification paramétrée est la protection de paramètre effectif, c'est-à-dire la consistance et la suffisante complétude de la spécification obtenue par

instanciation d'une spécification paramétrée, vis-à-vis de son paramètre effectif. Cette propriété appelée la persistance est stable par composition.

Les types paramétrés sont définis par [ADJ 78] comme des facteurs associant à un type d'objets de base, l'extension de ce type par le type des objets structurés, composés sur ces objets de base.

Définitions

Soient SPEC = $<S, \Sigma, E>$ une spécification appelée environnement, et SPECP = $<Sp, \Sigma p, Ep>$ une présentation appelée paramètre formel, sur SPEC.

Spécification paramétrée

On appelle spécification paramétrée par Sp, la spécification définie par : SPECTP = SPEC + SPECP + $<\{TP(P)\}, \Sigma_{TP}, E_{TP}>$
telle qu'il existe un constructeur c des objets du type TP qui comporte parmi les types de son domaine les sortes Sp : Sp $\in$ domaine(c).

Exemple : le type SUITE(P)

Soit la spécification d'environnement SPEC = $<S,\Sigma,E>$ avec
S = {BOOLEEN, ENTIER}

P type paramètre formel dont la présentation sur SPEC est
SPECP = $<\{P\}, \Sigma p, Ep>$
avec comme environnement : BOOLEEN, ENTIER

opérations
indef : $\rightarrow$ P (constante indéfinie)
eq : P x P $\rightarrow$ BOOLEEN (prédicat d'égalité)

axiomes soient p, p', p" : P
eq(p,p) = vrai
eq(p,p') = eq(p,p)
(eq(p,p') et eq(p',p")) impl eq(p,p") = vrai

Type SUITE (P)

présenté par $< \{SUITE\ (P)\}, \Sigma_{SUITE}, E_{SUITE} >$ sur SPEC + SPECP où P est le type paramètre formel, avec comme environnement : BOOLEEN, ENTIER, P.

opérations

$< >$	:	$\rightarrow$ SUITE	(suite vide)
$\sim$	:	SUITE(P) x P $\rightarrow$ SUITE(P)	(adjonction en tête)
indef	:	$\rightarrow$ SUITE (P)	(constante indéfinie)
DEBUT	:	SUITE (P) $\rightarrow$ SUITE(P)	(suite moins dernier élément)
DER	:	SUITE (P) $\rightarrow$ P	(dernier élément d'une suite)
LONG	:	SUITE (P) $\rightarrow$ ENTIER	(longueur de la suite)
PREM	:	SUITE (P) $\rightarrow$ P	(premier élément d'une suite)
RESTE	:	SUITE (P) $\rightarrow$ SUITE (P)	(suite moins premier élément)
bi	:	SUITE (P) $\rightarrow$ ENTIER	(borne inférieure d'une suite)
bs	:	SUITE (P) $\rightarrow$ ENTIER	(borne supérieure d'une suite)

axiomes α : SUITE (P), x : P

DEBUT $(\alpha \sim x) = \alpha$

DEBUT $(< >) = < >$

DER $(< >)$ = indef

DER $(\alpha \sim x) = x$

LONG $(< >)$ = zéro

LONG $(\alpha \sim x)$ = suc(LONG(α))

PREM $(< >)$ = indef

PREM $(\alpha \sim x)$ = PREM(α)

RESTE $(< >) = < >$

RESTE $(\alpha \sim x)$ = RESTE $(\alpha) \sim x$

bi $(< >)$ = 1

bi $(\alpha \sim x)$ = bi(α)

bs $(< >)$ = zéro

bs $(\alpha \sim x)$ = suc (bs(α))

Soit SPECTP = SPEC + SPECP + $\langle \{TP(S_P)\}, \Sigma_{TP}, E_{TP} \rangle$ une spécification paramétrée par la présentation SPECP = $\langle S_P, \Sigma_P, E_P \rangle$ du paramètre formen sur l'environnement SPEC = $\langle S, \Sigma, E \rangle$.

Définition : Une présentation SPECE = $\langle S_E, \Sigma_E, E_E \rangle$ sur le même environnement, est un paramètre effectif admissible de SPECTP s'il existe un morphisme de spécification, appelé morphisme de passage de paramètres, noté

$$i : SPEC + SPECP \rightarrow SPEC + SPECE$$

qui soit le morphisme identité sur SPEC.

Intuitivement, une spécification est un paramètre effectif admissible d'une spécification paramétrée, si elle comprend des opérations vérifiant les propriétés du paramètre formel.
Ainsi le type ENTIER est un paramètre effectif admissible de la spécification paramétrée SUITE(P).

Présentation obtenue par instanciation

La présentation obtenue par instanciation de SPECP par le paramètre effectif admissible SPECE, selon le morphisme de passage de paramètres i, est obtenue en substituant dans

- les profils des opérations de Σ_{TP}, les occurrences des sortes de S_P par leur sorte associée par i dans S_E,
- les équations de E_{TP}, les occurrences des opérations de Σ_P par leurs opérations associées par i de Σ_E et les occurrences des variables d'un type de S_P par une variable du type de S_E associé par i.

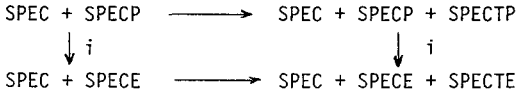
Spécification obtenue par instanciation

$$SPECTE = SPEC + SPECE + \langle \{TP(S_E)\}, \Sigma_{TE}, E_{TE} \rangle$$

où $\Sigma_{TE} = \Sigma_{TP}[i(P) \rightarrow P, \ \forall P \in S_P]$

$E_{TE} = E_{TP}[X_{i(P)} \rightarrow X_P][i(f) \rightarrow f, \ \forall f \in \Sigma_P]$

Diagramme :



Propriétés de complétude suffisante et de consistance

Une spécification paramétrée est une extension de la spécification paramètre ; elle doit être consistante et suffisamment complète vis-à-vis de sa spécification paramètre. Cela se démontre [LEV 84] à partir de la définition de la complétude et de la consistance sur les variables de la spécification paramétrée.

E. Problèmes de représentation

Les représentations sont généralement étudiées comme moyen de transformer par étapes successives une spécification en un programme ([ADJ 78], [FIN 79]). Chaque représentation précisant certains détails d'implantation, rend la spécification moins abstraite tout en en conservant les propriétés. Le problème de la représentation d'un type abstrait consiste à définir, d'une part les objets, d'autre part les opérations, en termes de types "plus simples", de telle façon que les axiomes du type soient des théorèmes de la représentation.

De manière intuitive, représenter un type source par un type cible, c'est définir les objets et les opérations du type source à l'aide d'objets et d'opérations du type cible tout en conservant les propriétés de la source et de la cible.

De façon générale dans la littérature deux approches de ce problème sont proposées pour donner une représentation d'un TA,

- préciser le support de la représentation, puis définir une fonction allant des objets de la représentation (objets "concrets") vers les objets du Type Abstrait : c'est ce que l'on appelle la "fonction d'abstraction". [GAU 80, HOA 72, GHM 78] ;

- préciser le suppport de la représentation, puis définir une fonction allant des objets du type abstrait vers les objets de la représentation et associant à chaque terme du type à représenter un terme du type représentant : c'est ce qu'on appelle la "fonction de représentation". [GTW 78, GAU 79].

Dans la troisième partie de cette thèse nous nous attacherons à l'implémentation des TA par le modèle relationnel. Nous étudierons la fonction de passage entre TA et modèle relationnel, sachant que la seule structure du type cible est alors la relation que l'on peut voir comme une partie du produit cartésien d'ensembles ou, dans notre cas ce sera plus intéressant, comme une table. Nous nous limiterons à une représentation faible au sens de [LEV 84] c'est-à-dire ne conservant que les propriétés observables, soient les valeurs des termes de sorte dans l'environnement.

CHAPITRE 3

LASSIF :

Un LAngage de Specification de Système d'InFormation

1. INTRODUCTION

Un Schéma Conceptuel de Système d'Informations est un schéma formel, unique et intégré dans lequel la structure statique du réel (les composants) et sa structure dynamique (les transformations et leurs interrelations) sont représentées.

La spécification d'un SI est la description de son schéma conceptuel au moyen d'un modèle formel.
Ainsi que le fait remarquer BRODIE dans [BRO 83B] pour spécifier un SI, dans un domaine particulier, tel qu'ici l'informatique de gestion, il faut disposer de langages adaptés ; et proposer un seul formalisme pour exprimer tous les problèmes et tous les points de vue. Le chapitre II de cette partie a introduit un tel formalisme. Il faut cependant remarquer que, si une formalisation à fondements mathématiques est nécessaire, imposer un tel vocabulaire à des utilisateurs gestionnaires n'est pas réaliste.

Le langage de spécification doit être plus "convivial", plus "tourné" vers l'informatique de gestion, d'autant qu'un sous ensemble de la théorie précédente suffit pour exprimer la plupart des problèmes en Informatique de Gestion (I.G.).
En effet il faut noter que les applications dans ce domaine manipulent un petit nombre de concepts en matière de traitements et reposent, par contre, sur une structuration des données volumineuse et importante. Le principal problème à résoudre lors de la programmation de ce genre d'applications est le parcours de fichiers plus ou moins complexes selon des critères plus ou moins compatibles entre eux, ce que l'on appelle le problème de "la cinématique des fichiers". Les calculs, eux, n'ont rien de commun, dans leur difficulté, avec les calculs scientifiques mais se résument généralement à quelques expressions arithmétiques et manipulations simples de tables.

Les buts d'un tel langage dépendent de ce que l'on attend comme bénéfices potentiels de la spécification d'un SI.

2. OBJECTIFS DE LA SPECIFICATION : QUALITES DU LANGAGE

2.1. Objectifs d'une spécification de SI en I.G.

Outre la qualité d'abstraction que nous reconnaissons à toute spécification, comme idée de base de nos travaux, une spécification de SI en Informatique de Gestion doit être :

. Formelle : disposer d'un formalisme c'est connaître de façon stricte l'ensemble des règles de description d'un énoncé de problème. Le manque de formalisme peut aboutir à des erreurs d'interprétation de la spécification. De plus on peut alors définir des techniques de preuves sous-jacentes pour vérifier les propriétés statiques et dynamiques du SI obtenu.

. Structurée : structurer c'est organiser la modélisation avec un nombre limité d'éléments quelles que soient la complexité et la diversité du problème. La formalisation doit reposer sur un petit nombre de concepts.

. Représentative : un SI doit être un modèle complet, cohérent, fidèle de la réalité de l'organisation. Pour être représentatif le SI doit être modélisé au moyen de concepts et de termes qui sont familiers à l'environnement du problème. Dans notre domaine, cela signifie que la modélisation du SI doit manipuler des notions accessibles à l'ensemble des utilisateurs de l'Informatique de Gestion.

. Lisible et communicable : la communication des concepts et leur connaissance parmi les personnes impliquées dans la conception et le développement de logiciel sont un point critique. La spécification doit être faite de façon à ce qu'elle puisse effectivement être communiquée : elle doit être une sorte de "contrat" entre concepteurs, implémenteurs, utilisateurs.

. Automatisable : une spécification est automatisable s'il est possible d'en déduire une solution technique viable. Il faut alors que la spécification, même si elle est exprimée dans un langage de haut niveau, possède toutes les informations nécessaires à la construction non ambiguë d'un SI. Elle doit être la base de définition de l'intégrité du SI.

2.2. Les qualités d'un langage de spécification

Pour satisfaire ces objectifs nous avons choisi de considérer la spécification comme un complément de la modélisation conceptuelle, soit de faire reposer le langage largement sur le modèle choisi. Certaines qualités de la spécification peuvent être atteintes par les concepts de modèle sous-jacents (structuration, abstraction, représentation) ; d'autres doivent être atteintes par les propriétés du langage lui même (formalisme, lisibilité, automatisabilité).

Un langage de spécification en Informatique de Gestion doit, à notre sens :

a) permettre une bonne représentation abstraite, structurée et formelle de la réalité ; il doit donc être complet, de haut niveau et reposer sur un petit nombre de concepts à base mathématique ;

b) être un moyen de communication lisible et compréhensible ; il doit donc :

. être prédicatif mais reposer sur des expressions simples, pas trop ésotériques (car les gestionnaires sont rarement des spécialistes en mathématiques),

. être non procédural, donc déclaratif ; ainsi la notion d'algorithme est transparente aux utilisateurs non informaticiens,

. reposer sur un petit nombre de constructions, traditionnelles en informatique, telles que les conditionnelles et les itérations,

. disposer d'une syntaxe quasi libre fortement aidée par un logiciel, ceci afin de faciliter la description cohérente d'une spécification ; la syntaxe doit permettre de définir quelles sortes d'assertions on accepte sur l'univers du discours.

c) permettre une spécification "automatisable" ; il doit donc être le support d'une traduction dans un langage cible aussi aisée que possible toute en restant de haut niveau.

L'objectif de ce chapitre est de décrire le langage LASSIF. Nous présentons le langage structurel permettant la spécification du Schéma Conceptuel de SI et le langage d'énoncé permettant l'expression des textes de programmes.

Nous replaçons enfin nos propositions parmi le cadre plus général des recherches sur les langages de spécification.

3. NOTRE PROPOSITION : LE LANGAGE LASSIF

Le langage que nous proposons, ici, est conçu en osmose avec un Logiciel d'Aide à la Spécification des Systèmes d'Information, développé en Partie II de cette thèse. Nous donnons dans ce chapitre sa syntaxe formelle, on pourra par ailleurs trouver en Annexe la description de la grammaire abstraite associée à notre logiciel.

Pensant proposer une entrée conversationnelle avec cadres prédéfinis et "trous" à remplir par l'utilisateur, nous comptons offrir une solution de langage à syntaxe quasi libre et simplifiée qui amènera le maximum de convivialité à notre système.

3.1. Les composantes de LASSIF

Spécifier un SI à l'aide de LASSIF c'est décrire de façon formelle, lisible et automatisable tous les détails du résultat de la phase de conception, basé sur le formalisme de SI proposé. Ceci demande que LASSIF permette l'expression :

(1) de la structure de SI, c'est-à-dire selon le modèle développé au chapitre précédent, des différentes classes d'objets, d'événements, d'opérations, du type SI lui même, ainsi que des contraintes d'intégrité associées aux données ;

(2) des accès aux informations dans le SI lui-même sous forme d'expressions prédicatives ;

(3) de la transformation des données sous forme des textes de conditions, facteurs et de règles de gestion.

(1) permet l'implémentation des données ;

(2) et (3) permettent l'expression des propriétés dynamiques des composants du SI dont les valeurs sont des séquences d'instructions ; c'est la base de l'implémentation des textes.

LASSIF contient donc à la fois des constructions structurelles et des constructions dynamiques : il est à la fois langage structurel et langage d'énoncé. Le langage structurel permet la description des classes du SI ; le langage d'énoncé permet la définition des règles, c'est-à-dire recouvre l'expression des prédicats (accès aux données et contraintes d'intégrité) et des textes à proprement parler (conditions, facteurs et opérations).

Pour des raisons de logique, de commodité d'écriture et de présentation, nous avons séparé ces deux aspects à la fois dans le langage et dans l'outil.

3.2. Le langage structurel

3.2.1. Introduction

LASSIF repose sur un formalisme de SI qui doit permettre l'expression complète de la structure d'un SI par un ensemble de schémas de type prédéfinis CLASSE-OB, CEVENT, COPERATION, SI. Ces schémas reposent, eux mêmes, sur un ensemble de types de base (ENTIER, BOOLEEN, CHAINE, DATE ...) et un ensemble de constructeurs paramétrés prédéfinis (TABLE, SUITE, PRODCAR ...).

Ainsi LASSIF propose un environnement complet de types prédéfinis, plus développé que dans les langages de programmation habituels. Ces types suffisent, à notre avis, à l'expression des problèmes en Informatique de Gestion où, en fait, le nombre de types manipulés est relativement limité. On peut dire que LASSIF est, de cette façon, un langage typé. Cependant, LASSIF nous semblant suffisamment riche ainsi, il ne nous a pas paru utile dans un premier temps, de permettre la définition de nouveaux types tels que le font les langages ADA [ADA 80], CLU [LIS 77] ou SPES [FIN 83]. On ne peut donc pas dire que LASSIF soit fortement typé mais, étant donnés le public et la catégorie de problèmes touchés par nos travaux, ainsi que les exemples concrets le montrent, ceci ne paraît pas être un obstacle au bon fonctionnement de notre système.

3.2.2. Le formalisme utilisé

Une spécification structurelle en LASSIF consiste en une collection de descriptions, chacune correspondant à la définition d'une classe du SI, et en une collection de dépendances fonctionnelles introduites explicitement par des fonctions. L'ordre des descriptions des instanciations des schémas de types, n'a pas d'importance ; l'outil guide cependant la description, ceci afin de faciliter l'entrée des définitions par l'utilisateur et la construction des arbres syntaxiques résultants.

A. Les classes

Remarque :

Tous les exemples du développement suivant se réfèrent à un problème de réservation de chambres d'hôtel dans des stations de sport d'hiver.
La description statique et dynamique de l'énoncé ainsi que son expression en LASSIF sont données en annexe.

Quelle que soit la classe, le format de la description est identique ; seuls le nombre et la nature des paramètres diffèrent d'une classe à l'autre selon le formalisme introduit au chapitre II.

La syntaxe est la suivante :
Type nomtype = nomconstructeur (liste de paramètres)
où
. "nomconstructeur" est, à ce niveau de description, un des schémas de types prédéfinis autorisés : CLASSE-OB, CEVENT, COPERATION, SI.
. "liste de paramètres" est la liste liée à la nature de la classe, où tout paramètre inexistant est représenté par la valeur particulière "nil" et tout paramètre, dont on ne veut pas donner la définition complète à ce niveau, doit être introduit en majuscules et défini ultérieurement.

a) Les classes d'objets (chapitre II paragraphe 2.3.6-A)

Elles sont introduites par le schéma CLASSE-OB :

Type nom d'une classe d'objets = CLASSE-OB(CLE,compp,COMP,compv, COMPV,datesup,predc,predv1,...,predvk,predg)

Rappelons :

. CLE : clé de la classe, identificateur de type quelconque ;

. compp : sélecteur du champ "schéma permanent" ;

. COMP : schéma permanent, généralement type en majuscules redéfini ensuite ;

. compv : sélecteur du champ "schéma variable" ;

. COMPV : identificateur de schéma variable redéfini ensuite comme un produit cartésien de sous-schémas variables ;

. datesup : sélecteur du champ "table de suppression" ;

. predc,predv2,...,predg : noms des contraintes d'intégrité de la classe définies ultérieurement dans le langage d'énoncé.

Exemples* :

. classe Cdemande :

Type Cdemande = CLASSE-OB (<ndem:ent>, demande, DEMANDE, nil, NIL, datedemsup)

Cette classe est composée d'un seul schéma permanent DEMANDE qui sera redéfini ultérieurement comme un produit cartésien :

Type DEMANDE = <dnom:chaîne, dadr:chaîne, dcat:chaîne, ddebdem:date, dfindem:date, nbch:ent, datedemcre:date>

La clé de la classe Cdemande est simple ; aussi a-t-on choisi de la décrire directement comme un produit cartésien.

---

* Par souci de clarté les paramètres de type prédicat seront omis dans les exemples

. classe Créservation :

Type Créservation = CLASSE-OB (<nres:ent>, réservation, RESERVATION, resvar, RESVAR, dateressup)

Cette classe est composée d'un schéma permanent RESERVATION et d'un schéma variable RESVAR qui seront redéfinis par la suite dans des produits cartésiens :

Type RESERVATION = <ddebres:date, dfinres:date, datecreres:date>
Type RESVAR = <demvara:ETATRES, demvarb:ANNRES>

ETATRES et ANNRES sont les deux sous-schémas variables (ou c-objets variables) de la classe Créservation ; ils sont redéfinis ainsi :

Type ETATRES = <datechgtetat:date, etat:ent>
Type ANNRES = <dateannul:date, motif:chaîne>

b) Les classes d'opérations (chapitre II paragraphe 2.3.6.-C)

Elles sont introduites par le schéma COPERATION :

Type nom d'une classe d'opérations = COPERATION (NCOP,datexec,cobmod, COB1,COB2,top,preg)

Rappelons :
. NCOP : clé de la classe
. datexec : sélecteur du champ "table des exécutions" des opérations
. cobmod : sélecteur du champ "table des modifications d'objet"
. COB1 : identificateur du c-objet paramètre en entrée du texte d'opération
. COB2 : identificateur du c-objet modifié par l'opération
. top : sélecteur du champ "table des textes d'opération"
. preg : nom des contraintes d'intégrité.

Exemple :

déclaration de la classe OP5 "mise à jour des disponibilités" majdisp

Type majdisp = COPERATION (<nmajd:ent>, datexecmajd, cobmodmajd, DEMANDE, CHDISP, topmajd)

où DEMANDE est le c-objet en entrée de la règle de gestion, CHDISP le c-objet modifié par le texte.

c) les classes d'événements (chapitre II paragraphe 2.3.6-D)

Elles sont introduites par le schéma CEVENT :

Type nom d'une classe d'événements = CEVENT (NCEV, datearriv, cobconst, predicat, copd, DEC, preg)

Rappelons :

. NCEV : clé de la classe
. datearriv : sélecteur du champ "table des arrivées" d'événements
. cobconst : sélecteur du champ "table des objets constatés"
. COB : identificateur du c-objet constaté
. prédicat : sélecteur du champ "table des textes" de prédicats
. copd : sélecteur du champ de déclenchement des opérations par les événements
. DEC : identificateur de type redéfini comme un produit cartésien à trois champs représentant la "table des exécutions" des opérations, la "table des textes" de condition, la "table des textes" de facteur.
. preg : ensemble des contraintes d'intégrité.

Exemple :

Déclaration de la classe EV4 "annulation de réservation" : evannul

Type evannul = CEVENT (<nevannul : ent>, tabdatarrivannul, tabconsannul, ANNRES, predicannul, copdannul, DECANNUL, pregannul)

où ANNRES est le c-objet dont le changement d'état est constaté par evannul et DECANNUL sera redéfini ultérieurement par un produit cartésien.

DECANNUL = <x1:DECANNULMAJDISP, x2:DECANNULETAT>

Les deux types DECANNULMAJDISP et DECANNULETAT permettent d'introduire les tables de déclenchement de OP5 par EV4 et OP6 par EV4 (cf. schéma dynamique en annexe), leur condition de déclenchement et leur facteur de déclenchement.

Type DECANNULMAJDISP = < h_{51} : TABLE (nevannul, majdisp), h_{52} : TABLE (DATE, nil), h_{53} : TABLE (DATE, FACTEV4OP5)>

Dans cet exemple le déclenchement de OP5 par EV4 est inconditionnel, aussi la table des textes est vide, FACTEV4OP5 est, lui même, un type à redéfinir sous la forme :

Type nomdefacteur = FACTEUR (nom du schéma en entrée)

Soit ici :

Type FACTEV4OP5 = FACTEUR (ANNRES)

DECANNULETAT sera redéfini de façon similaire.

d) La classe SI (chapitre II paragraphe 2.3.6-B)

Il n'y en a qu'une par système d'information. Elle est introduite par le schéma de type SI et permet d'exprimer le regroupement des classes de type CLASSE-OB, CEVENT et COPERATION appartenant au même SI.

type nom de classe SI = SI (CLAS, COP, CEV, $predSI_1$,..., $predSI_k$)

Rappelons :

. CLAS : produit cartésien de classes de type CLASSE-OB
. CEV : produit cartésien de classes de type CEVENT
. COP : produit cartésien de classes de type COPERATION
. $predSI_1$, ..., $predSI_k$: contraintes d'intégrité de la classe

Exemple :

Type SIRESERV = SI (CLOBRES, CLOPRES, CLEVRES)

où les types paramètres auront été définis par ailleurs suivant la hiérarchie schématisée paragraphe 2.3.6-E, chapitre II.

B - Les dépendances fonctionnelles

Exprimées par des fonctions, elles sont de plusieurs sortes :

a) Construites entre classes d'objets

Les classes d'objets (instanciations de CLASSE-OB) représentent les classes d'objets du monde réel et les associations m-n qui les lient. Pour représenter les associations 1-n nous introduisons les dépendances fonctionnelles entre les classes d'objets.

Elles sont définies à l'aide du constructeur TABLE selon le format :

nom de la fonction = TABLE (nomsouschema1 de nomclasse1 [cle1],
nomsouschema2 de nomclasse2 [cle2])

où

. nomsouschema1 représente le schéma permanent (variable) de la classe 1, origine de la dépendance fonctionnelle

. nomsouschema2 représente le schéma permanent (variable) de la classe 2, but de la dépendance fonctionnelle

. cle1 et cle2 sont respectivement les clés des souschema1 et souschema2.

Exemples : (cf. schéma statique en annexe)

. exprimer qu'un hôtel est attaché à une station

hotstat = TABLE (hôtel de Chotel [numhot], station de Cstation [numstat])

. une chambre existe pour un hôtel

chhot = TABLE (chambre de Cchambre [numhot, numch], hôtel de Chôtel [numhot])

Remarque

Les dépendances fonctionnelles structurelles seront utilisées ultérieurement pour l'expression des accès aux données. De la même manière il en existe, et si on utilisait le modèle relationnel il faudrait les définir explicitement, à l'intérieur même d'une classe d'objets. Elles représentent les relations entre un c-objet permanent (ou schéma permanent) et les c-objets variables (ou sous-schémas variables) de la classe. Du fait même du formalisme introduit précédemment, nous n'avons pas à les expliciter car elles sont exprimées par la structure de représentation du type CLASSE-OB.

b) construites entre classes d'objets, d'événements, d'opérations

Les dépendances fonctionnelles permettent d'expliciter le type SI. Elles servent à l'expression des contrôles et de la dynamique. Elles sont, en fait, définies à l'aide des opérations CONST, MOD, OPERA introduites précédemment (chapitre II paragraphe 2.3.6-D).

. entre classe d'opérations et classe d'objets : la fonction MOD rend le type du sous-schéma permanent (variable) modifié par les opérations de la classe.

Exemple

Type majdisp = COPERATION (<nmajd:ent>, datexecmajd, cobmodmajd, DEMANDE, CHDISP, topmajd)

MOD (majdisp) rend CHDISP sous-schéma variable de la classe Cchambre modifié par le texte de l'opération.

. entre classe d'objets et classes d'événements : la fonction CONST rend le type du sous-schéma permament (variable) dont le changement d'état est constaté par les événements de la classe.

Exemple :

Type evannul = CEVENT (<nevannul : ent>, tabdatarrivannul, tabconsannul, ANNRES, predicannul, copdannul, DECANNUL, pregannul)

CONST(evannul) rend ANNRES sous-schéma variable de la classe Créservation paramètre en entrée du texte du prédicat.

. entre classe d'événements et classes d'opérations : l'opération OPERA rend les types des opérations déclenchées, éventuellement, par les événements de la classe.

Exemple :

OPERA (evannul) rend majdisp et annreserv, opérations, composantes de DECANNUL, déclenchées si les conditions sont respectées à la suite de l'annulation de réservation.

Notons qu'il s'agit là d'une fonction multivaluée.

Les fonctions dérivées d'opérations ne sont pas, en fait, à définir explicitement par le concepteur ; le système des déduira automatiquement des descriptions.

3.3. Le langage d'énoncé

3.3.1. Introduction

LASSIF repose sur un formalisme de SI qui doit permettre d'une part l'expression complète des attributs dynamiques des classes précédentes (soit des textes des règles de fonctionnement), d'autre part l'expression des contraintes d'intégrité relatives aux classes. Le langage d'énoncé recouvre deux aspects principaux :

(i) La description des expressions prédicatives relatives à l'accès aux données. Ce qui dans d'autres systèmes [COD 70, DEM 75, AST 75] s'exprime par des constructions de l'algèbre ou du calcul relationnels, s'exprime ici, par des prédicats. Les assertions prédicatives d'accès s'appuient sur les opérations associées aux constructeurs TABLE et produit cartésien qui sont, en fait, les constructeurs de base des classes.

(ii) La description des textes des règles à proprement parler, c'est-à-dire des facteurs, des conditions et des opérations ainsi que des prédicats d'intégrité. Le langage proposé, là, s'appuie sur le formalisme introduit au chapitre II et sur un raisonnement déductif où l'on pourra être amené à définir des intermédiaires de calcul.

Les intermédiaires de calcul sont des variables mathématiques qui feront l'objet d'une déclaration intermédiaire afin de déterminer leur type et leur structure.

3.3.2. Les expressions prédicatives d'accès aux données

LASSIF est un langage d'assertions qui permet de décrire le cheminement d'accès aux données de façon déductive. Pour accroître la lisibilité des textes, les opérateurs logiques habituels sont exprimés en clair par des mots-clés. La forme générale

d'une expression prédicative d'accès aux données est la suivante :

variable de type TABLE = { liste des données cherchées/expression de la recherche }

(i) liste des données cherchées : il s'agit du produit cartésien de champs constitué à partir des fonctions définies entre classes d'objets et des mots-clés :

- dans dom exprimant la recherche d'une valeur de clé dans un domaine, i.e. l'entrée dans une table du SI,
- de permettant la qualification des sous-schémas par rapport à la classe à laquelle ils appartiennent,
- [ ] permettant la définition de la clé à appliquer à une entrée de table.

(ii) expression de la recherche : il s'agit d'une expression prédicative obtenue à partir des :

- mots clés : dans dom, de, [ ] et des opérateurs habituels de logique EXIST, TOUT, ET, OU, NON, t.q.,
- des expressions conditionnelles traditionnelles avec opérateurs de comparaison,
- des fonctions définies entre classes ou intraclasses représentant des dépendances fonctionnelles ou l'expression d'opérations structurelles.

Exemples :

a) Donner tous les numéros de réservations faites dans la station "d'Alpe d'Huez" pour l'hôtel "Bel Alpe".

```
numres = {x dans dom (réservation de Créservation) / EXIST y dans
     dom (hôtel de Chôtel) t.q. (y = reshot [x]
% Il s'agit de l'hôtel réservé pour cette demande %
et nomhôtel de hôtel de Chôtel [y] = "Bel Alpe"
% le nom de l'hôtel trouvé est Bel Alpe %
et EXIST z dans dom (station de Cstation) t.q. (z = hostat [y]
% Trouver la station à laquelle appartient l'hôtel %
et nomstat de station de Cstation [z] = "Alpe d'Huez"))}
```

b) Pour la réservation 2002 donner le nom et l'adresse du client et le numéro de la station demandée.

resev = { numstat = statdem [ndem],
nomcli = nomcli de client de Cclient [ncli],
adrcli = adrcli de client de Cclient [ncli] /

EXIST ndem dans dom (demande de Cdemande) t.q. ndem = resdem [2002]
% reservation concernée %
et EXIST ncli dans dom (client de Cclient) t.q. ncli = rescli [2002] }
% client concerné par cette réservation %

3.3.3. Les déclarations

Elles sont la définition des intermédiaires de calcul, ou variables au sens mathématique du terme, ne prenant qu'une valeur au cours d'un module. Ce sont, en fait, des instanciations mathématiques auxquelles nous donnons un nom. Ainsi l'opérateur "=" permet d'introduire la définition d'un intermédiaire. La définition formelle d'une telle variable est donnée à l'aide d'un des types primitifs introduits ou par instanciation d'un constructeur de type. Cette sorte de variable locale ne reçoit qu'une définition c'est-à-dire ne doit apparaître qu'une fois en partie gauche d'un signe d'affectation.

Sa déclaration est la suivante :

VAR nomvariable : nomtype [(liste de paramètres)]

Les types disponibles sont tous les types de base et les constructeurs définis au chapitre II. Soient :

. ENTIER, BOOLEEN, CHAINE, DATE.

. Les constructeurs : TABLE, $\mathcal{P}$(E), SUITE(E), PRODCAR labellé qui permettent, en fait, d'exprimer toutes les structures de données de base en Informatique de gestion.

. Le type IDENT utile si on veut définir un symbole de données en tant que tel.
Chaque déclaration de variable est suivie de sa définition informelle et formelle.

Les textes manipulent également des données du Système d'Information. Elles doivent avoir été introduites au préalable et on ne rappelle ici que leur symbole et leur définition informelle. En fait nous simplifions, ici, le formalisme de leur déclaration : ainsi, citer dans un texte le sous-schéma permanent réservation de la classe Créservation, équivaut à dire que l'on manipule un objet du type paramétré instancié Créservation, lui même instanciation du schéma CLASSE-OB.

Exemple :

Complétons l'exemple précédent en déclarant numres et en définissant les données introduites. Nous utilisons le formalisme SPES [QUE 84].

```
VAR numres : TABLE (<x:ent>, nil)
    numres ? reçoit les numéros de réservation
    réservation ? aspect permanent de la classe des réservations
    Créservation ? classe des réservations
    hôtel ? aspect permanent de la classe des hôtels
    Chôtel ? classe des hôtels
    station ? aspect permanent de la classe des stations
    Cstation ? classe des stations de sport d'hiver
```

de plus réservation, Créservation, hôtel et Chôtel, station et Cstation seront tous déclarés comme étant du type SI :

```
réservation : SI
```

signifiant dans ce cas que réservation est une table existant déjà dans le Système d'Informations.

3.3.4. Le langage de spécification des textes

A. Introduction : rappels sur le langage SPES

Le langage de spécification des textes que nous proposons s'appuie sur des constructions, une syntaxe et un raisonnement déductif proposés dans le langage du projet SPES [QUE 84] développé au CRIN à Nancy, sous la direction de J.P. Finance.

L'objectif du projet SPES est le développement d'un système interactif de spécification et de résolution de problèmes [FIN 83]. Dans ce système l'activité de programmation recouvre deux phases :

1) la spécification du problème,

2) la construction d'un algorithme puis du programme résolvant ce problème.

Pour aider le concepteur à formaliser son énoncé, on lui propose un langage de spécification [QUE 84], une méthode [FIN 79] et un ensemble d'outils logiciels. La construction de l'algorithme puis du programme est vue alors comme une succession de transformations formelles.

Le langage SPES est caractérisé par les points suivants :

(i) Une spécification est formée d'un ensemble d'énoncés et d'un ensemble de descriptions de sortes (types).

(ii) Un énoncé permet de définir les objets de façon descendante et déductive à partir des résultats ; les objets sont définis informellement puis formellement avec leur sorte. Chaque définition d'objet peut introduire de nouveaux résultats intermédiaires et précise la relation qui lie l'objet aux objets utilisés. Cette hypothèse induit une approche que certains qualifient, selon les domaines, d'analytique ou par les sorties ou top-down ou de conception descendante.

(iii) Les définitions algorithmiques classiques (expression, conditionnelle, itération) sont permises lors de la définition d'un résultat.

(iv) Une description de sorte se fait soit par une expression utilisant des constructeurs de sorte (comme les définitions de types en Pascal), soit en décrivant la sorte à l'aide d'opérateurs et d'équations selon le formalisme des Types Abstraits Algébriques.

Le langage, par sa nature, veut s'adapter à toutes sortes de problèmes et de domaines en particulier au domaine de l'informatique de gestion, tout au moins dans son noyau. Cependant il nous semble encore relativement ésotérique pour des gestionnaires même informaticiens. C'est pourquoi nous avons conservé un certain nombre d'idées de base et avons fait de nouvelles propositions.

De façon similaire à la méthode déductive de programmation [FIN 79] base de SPES, nous préconisons un raisonnement déductif pour définir les résultats conduisant le concepteur à agir par raffinements successifs. C'est en effet une des idées d'origine du projet REMORA [LAMY 77, PER 76]. Nous avons limité l'utilisation des propositions de SPES au domaine de l'algorithme.

En effet, proposant une structuration des données par le langage structurel et considérant, ainsi que nous l'avons dit, que LASSIF était "abstractionnement" complet c'est-à-dire qu'il contenait assez de types prédéfinis pour autoriser toute spécification d'Informatique de gestion, nous n'avons pas jugé utile de permettre la définition de nouvelles sortes.

Le langage de spécification des textes est un langage de définition de problèmes de gestion et non un langage de formulation de la méthode algorithmique de leur résolution.
L'énoncé est purement statique et déclaratif, l'ordre dans lequel il est fait ne reflète aucunement l'ordre de l'algorithme réordonné.

B. La spécification à l'aide du langage

Nous avons conservé du langage SPES, outre le raisonnement déductif, certains éléments de syntaxe :

- tout résultat, intermédiaire, donnée introduits lors de l'explicitation de l'énoncé doivent être clarifiés par une définition informelle ;
- les définitions implicites sont introduites par le mot-clé tq ; en fait nous utilisons ce type de définition lors des assertions d'accès aux données ;
- les définitions explicites sont introduites par le mot-clé = ;
- nous autorisons les constructions traditionnelles telles que les conditionnelles et les itérations que nous avons complétées par de nouvelles formes.

(i) Le formalisme

Rappelons que le langage d'énoncé doit permettre la spécification des contraintes d'intégrité associées aux données ainsi que la définition des attributs dynamiques du SI. Nous donnons ici la forme, les instructions et les constructions de base. On trouvera la grammaire abstraite du langage d'énoncé en annexe.

Le format général de la spécification d'un texte est le suivant :

tc_i
res_i X_i ($liste_i$)
I_i

où

α) . tc_i est un mot-clé à cinq valeurs possibles selon le type de l'énoncé :

- ASSERT identifie une spécification de contrainte d'intégrité,
- CONDITION identifie le texte d'une condition de déclenchement d'une opération par un événement,
- FACTEUR identifie un facteur de déclenchement d'une opération par un événement,

- PREDICAT identifie un prédicat associé à un type d'événement, exprimant un changement d'état de c-objet,

- OPERATION identifie l'ensemble des actions associées à un type d'opération.

β) . $\underline{res}_i$ identifie le résultat principal de la spécification. Suivant la nature du texte, il peut être de différents types.

Plus précisément :

- les contraintes d'intégrité, conditions de déclenchement et les prédicats associés à des types d'événements sont tous des assertions prédicatives. Ils sont définis, dans le formalisme introduit au chapitre II, comme étant du type : PREDICAT(E) défini comme TABLE(E,BOOL).
Dans ce cas res_i est du type BOOLEEN ;

- le type TEXTE a été défini ainsi :
type TEXTE = fonct (EC-OB, CLASSE-OB) CLASSE-OB
Il rend un nuple de c-objet appartenant à une classe d'objets ; dans ce cas le résultat est du type sous-schéma d'une classe du SI. On le déclarera de type SI ;

- le type FACTEUR a été défini par :
type FACTEUR = fonct (CLASSE-OB, EC-OB) EC-OB
Il rend un ensemble de nuples de c-objet ; qui seront paramètres d'entrée du texte de l'opération déclenchée ; le type de res_i est alors une table de nuplets.

γ) . $\underline{X}_i$ est l'identificateur du module.

λ) . $\underline{liste}_i$ identifie la liste des paramètres en entrée ;

ε) . $\underline{I}_i$ est l'ensemble des assertions LASSIF constituant la spécification du résultat principal. Le corps fonctionnel est une liste de déclarations et d'instructions éventuellement regroupées par des constructions. Lors de la définition d'un résultat on peut être amené à introduire un intermédiaire dont on donne la spécification, toujours selon les étapes :

- nom et type du résultat,
- définition informelle,
- définition formelle.

Le raisonnement se poursuit par raffinements successifs jusqu'à parvenir aux données.

(ii) les instructions de LASSIF

Notons que l'écriture de textes à l'aide de LASSIF, étant donné le domaine où nous nous plaçons et le type de modélisation choisi, repose sur deux structures de données principales :

- la structuration en TABLE qui est en fait la base de notre formalisme car elle représente un ensemble de nuples répondant à un critère donné,

- la classe d'objets CLASSE-OB, produit cartésien de tables, qui est le noyau de notre modèle de données.

Au niveau du langage d'énoncé nous avons donc défini et réalisé l'implémentation, dans un premier temps, de toutes opérations associées aux types TABLE et CLASSE-OB, définies au chapitre II.

De plus notre langage possède toutes les opérations associées aux types de base ENTIER, BOOLEEN, CHAINE et DATE définies traditionnellement et permet des déclarations de variables mathématiques.

(iii) Les constructions disponibles

Ainsi que nous l'avons dit lors de la définition de la sémantique du type TEXTE (cf. chapitre II paragraphe 2.3.6-B) les constructions sont des compositions de texte en termes d'instructions de base.

Rappelons les constructions définies précédemment :

. la conditionnelle : soient t1, t2, t3 : TEXTE
cond(t1,t2,t3) = si t1 alors t2 sinon t3 fsi

. l'itération avec condition d'arrêt : soient t1, t2 : TEXTE
tque (t1,t2) = tantque t1 faire t2 ftque

. l'itération sur un ensemble de valeurs : elle est très utile en informatique de gestion, elle représente en particulier le parcours d'un ensemble de nuplets d'une TABLE.

soient x, y, t1 : TEXTE
pche (x,y,t1) = pourchaque x dans y faire t1 fpche

Le noyau d'instructions et de constructions que nous avons défini et implémenté nous semble suffisant pour spécifier le type de problème auquel nous sommes confrontés en informatique de gestion.

4. UN ESSAI DE PROPOSITION DE METHODE

4.1. Problème de la modélisation de l'organisation

Le problème à résoudre, lors de la conception d'un SI, est avant tout un problème de représentation. Il s'agit de construire des représentations aussi complètes et aussi fidèles que possible de l'ensemble des aspects des phénomènes réels de l'organisation qui présentent un intérêt pour la gestion de celle-ci. Le SI doit prendre en compte la réalité statique, dynamique et évolutive de l'organisation.

Nous ne chercherons pas à répondre ici à la question : "quel découpage en applications proposer selon les sous-systèmes de l'organisation considérée" ce n'est pas notre propos. Certains, dans l'équipe REMORA se sont penchés sur ce problème ; ainsi :

- Walter PEREA [PER 76] a proposé une approche modulaire de l'organisation qui induit son découpage en sous-systèmes,
- Odile FOUCAUT [FOU 82] a repris la définition de l'organisation de MESAROVIC [FOR 67] qui préconise deux sous-systèmes :

  - le sous-système causal qui est composé des unités causales, qui sont, par exemple, les divers processus technologiques, dont les réponses aux perturbations qu'elles subissent sont parfaitement définies par la transformation qui lie leurs entrées et leurs sorties ;

  - le sous-système téléologique composé d'unités poursuivant des objectifs, telles que des services, des comités ou des hommes que l'entreprise définit par les objectifs qu'elles sont chargées de poursuivre et leurs moyens d'action sur le sous-système causal.

O. FOUCAUT a conclu, à partir de cette analyse, que nous nous intéressions au sous-système causal. Nous ne cherchons pas à donner de méthode pour obtenir l'explicitation de ce sous-système.

4.2. Nos hypothèses de travail

(i) Les propositions méthodologiques que nous pouvons faire sont liées à l'outil d'aide à la conception que nous présentons dans la partie suivante de cette thèse. Nous avons fixé un canevas de description du Schéma Conceptuel de SI.

En effet bien que nous considérions que l'ordre d'entrée des définitions ait peu d'importance, il nous semble préférable de définir un ordre de saisie des classes d'objets, classes d'événements et classes d'opérations et un mode de prise en compte des modifications. Ceci simplifie le travail de l'outil, en particulier en ce qui concerne la création des arbres syntaxiques, et permet moins d'interrogations et de descriptions pour l'utilisateur.

(ii) L'outil d'aide à la conception n'évite pas une réflexion préliminaire sur la modélisation de l'organisation concernée. Il faut, dans un premier temps, cerner le problème par des discussions et séries d'entretiens auprès des futurs utilisateurs du SI ; dans un deuxième temps, déterminer les classes structurelles du futur SI et leurs transformations dynamiques. Ceci demande un travail "papier" avant toute saisie informatique. En effet, comme le fait remarquer J.P. JACQUOT [JAC 84], la spécification d'énoncé directe, à la console, ne peut concerner que de petits algorithmes où l'étape de réflexion, vu la nature du problème, peut quasiment être sautée et où l'outil peut jouer un rôle prédominant.

4.3. La méthode proposée

On trouvera en partie II une analyse des différentes démarches possibles.

Bien que les données fassent partie d'un tout intégré, nous préconisons un processus en deux étapes :

(i) description et saisie des classes structurelles d'objets, d'événements, d'opérations,

(ii) description et saisie des textes exprimés à l'aide de LASSIF.

4.3.1. Définition des classes structurelles

Il s'agit d'effectuer, ici, un raisonnement sur les objets et leurs changements d'état. Le point de départ de l'analyse est ce que J.P. FINANCE [FIN 79] appelle <<une stratégie d'induction sur les données>> c'est-à-dire où le guide essentiel est la structure des données.

Il s'agit, au terme du processus d'analyse du réel, d'apporter la réponse aux deux questions [BEN 79] :

. Quels sont les objets et les associations sur lesquels l'organisation souhaite être informée ?

. Quelles sont les propriétés qui permettent de les caractériser de manière satisfaisante ?

Dans l'équipe REMORA, M. KRIEGUER [KRI 78] a proposé un processus méthodique de construction du SC qui permet de définir l'ensemble des c-objets et de les structurer en classes de c-objets.
(On trouvera le résultat de ce raisonnement appliqué à l'exemple des réservations en annexe).

C'est un processus de structuration par composition qui, à partir d'une liste d'attributs et un ensemble de dépendances fonctionnelles fournis par le concepteur, procède par agrégation des attributs en c-objets et des c-objets en classes.

Nous lui adjoignons un raisonnement sur la dynamique des c-objets qui permet au concepteur :

- d'analyser les changements d'état que peuvent subir les c-objets des classes et d'en déduire les classes d'événements correspondantes,

- de déterminer les conséquences de ces changements d'état et d'en déduire les classes d'opérations résultantes.

L'ensemble de ces classes sont décrites ensuite à l'aide du langage LASSIF et introduites dans le système suivant les étapes conversationnelles du logiciel d'aide à la conception LASSIF.

4.3.2. Définition des textes

Nous préconisons un raisonnement déductif se basant sur une définition du résultat principal voisine de ce que nous propose MEDEE comme "stratégie d'induction sur les résultats" [FIN 79] [JAC 84]. Dans le projet REMORA, nous avons [LAM 77] proposé de schématiser ce raisonnement de définition de résultat par un arbre qui fait apparaître la nature (résultat, donnée, résultat intermédiaire) et le type (inconditionnel, conditionnel, liste de résultats) des différents éléments de calcul introduits jusqu'à expliciter les données.
A chacun des noeuds de cet arbre correspondent une définition informelle, la définition de son type et son explicitation formelle exprimées en LASSIF.

4.4. Role de l'outil

Lors de l'entrée des spécifications abstraites, l'outil a pour rôle principal d'aider à la conception du schéma conceptuel du SI ; il doit alors :

(i) proposer un canevas méthodologique et des logiciels les plus "conviviaux" possibles (tels que des éditeurs vidéo et l'utilisation de touches fonctions). Cependant nous pensons qu'il ne s'agit là que d'outils d'aide qui "pilotent" la conception, en guidant le concepteur dans sa réflexion, en l'interrogeant en cas de conflit et en effectuant à sa place certains choix possibles. A notre avis le concepteur reste maître de sa conception et le processus ne peut lui être transparent ;

(ii) contrôler les spécification entrées, permettre les retours en arrière en cas d'erreur, intégrer les nouvelles spécifications parmi le système existant. En particulier l'outil doit déclencher les procédures de vérification introduites lors de la définition des schémas de types CLASSE-OB, CEVENT, COPERATION et SI (cf. chapitre II) ; et se livrer aux contrôles syntaxiques de type classique sur les textes d'énoncés ;

(iii) documenter les concepteurs sur les spécifications déjà entrées ; soit par des éditions systématiques sous un format fixé ; soit par des éditions paramétrables permettant une interrogation plus souple.

5. LE CADRE GENERAL DE NOS PROPOSITIONS : LES LANGAGES ACTUELS

De même qu'il n'y a pas de consensus général sur ce qu'est la modélisation d'un problème, où, quand, comment elle doit être utilisée, il n'y a pas non plus identité de vue sur les langages que l'on peut proposer selon les domaines.

Les langages de programmation ont été les premiers langages d'expression formelle des problèmes ou plus exactement des algorithmes de résolution des problèmes. Mais comme le fait remarquer BACKUS [BAC 78] : <<les langages de programmation sont devenus de plus en plus énormes mais pas meilleurs ... impossibilité de construire de nouveaux programmes à partir d'anciens et manque de propriétés mathématiques pour raisonner sur les programmes>>. Différentes analyses sur les langages [BUB 83] [BAC 78] incriminent les principes de VON NEUMANN : proposer des langages avec "affectations", où toutes les instructions et constructions sont réalisées autour de l'instruction d'"affectation" conduit à un raisonnement algorithmique. [BUB 83] montre comment la modélisation des données et la conception des Bases de Données ont particulièrement souffert de ce principe.

En analyse de gestion, l'apparition de nouveaux langages se justifie par deux objectifs :

(i) trouver une interface entre l'énoncé du problème et sa solution programmatoire,

(ii) rechercher un passage automatique de l'interface à l'algorithme.

R. REIX [REI 70], en particulier, a fait des propositions pour atténuer la procéduralité de l'expression des traitements. Le CODASYL [CODA 71] et le projet ISDOS [TEI 77] ont développé de nouvelles structures de données moins proches du niveau physique d'implémentation.

De nombreuses méthodes d'analyse (PROTEE [SIS 78], ARIANE [GAM 77], CORIG [SGI] ...) ont proposé des générateurs réalisant, à partir du schéma logique d'un programme, son algorithme dans un langage donné.

Actuellement, ainsi que le remarque J.P. FINANCE dans [FIN 79] de très <<nombreux travaux de recherche ont consisté à définir des langages permettant d'exprimer des énoncés de façon plus ou moins formelle, plus ou moins procédurale ... certains de ces langages sont (ou se disent) universels, d'autres sont spécifiques d'un domaine particulier>> (cf. [FIN 79] pour une analyse des différents types de langage).

Ce sont :

(i) des langages de spécification généraux, parfois développés pour répondre à des domaines particuliers mais extensibles : SETL [KEN 75], ALICE [LAU 76], CIP [BAU 78], PROLOG [WAR 77], MEDEE [BEL 78, PAI 79, FIN 79] sont de ce type ;

(ii) des langages orientés "types" "abstraits" : CLU [LIS 77], ATM [MIN 79], VEGA [CHA 83] sont de cette catégorie ; ainsi que le langage Z développé par ABRIAL [ABR 78, ABR 80] qui a été expérimenté dans le domaine particulier de l'informatique de gestion [ABR 77].

En conception de Bases de Données, actuellement, les langages que l'on peut qualifier de relationnels semblent prédominer. Ces langages utilisent, soit l'algèbre relationnelle et le calcul des prédicats comme le langage ALPHA défini par CODD [COD 70], soit le calcul relationnel comme le langage SQUARE [BOY 74], soit des aspects de ces trois théories comme le langage SEQUEL [CHA 74].

Ce type de langage, s'il permet la manipulation des relations et l'expression des prédicats, ne permet pas la définition de types de schémas de relation, de types de relations et de types de constituants. Seule l'introduction des Types Abstraits propose cette expression, réalisée généralement par le biais du type RELATION. Ainsi J.J. CHABRIER a introduit le type nuple labellé dans VEGA [CHA 83]. H.J. SCHMITT [SCH 77] et A. WASSERMAN [WAS 82A] ont spécifié le type RELATION, respectivement dans les langages PASCAL-R et PLAIN, tous deux construits autour du noyau PASCAL [WIR 76].

Les nouveaux langages en conception des SI, ainsi que nous l'avons explicité au chapitre précédent, en particulier ceux présentés aux congrès CRIS1 et CRIS2 sont fortement fondés sur les concepts de Type Abstrait. SYSDUL [ASH 82], ACM/PCM [BRO 82], CIAM [GUS 82], SDLA [KNU 82], RIDL [VER 82], BASIS [WAS 82A] appartiennent à cette nouvelle tendance. Il en est de même pour les langages de Bases de Données multimédia tels que le proposent TIGRE [ADI 85] et BIG [CHR 85].

6. CONCLUSION

Dans ce chapitre nous avons explicité le type de langage que l'on peut proposer pour la conception des SI en Informatique de Gestion. Il doit être aussi proche que possible de la langue naturelle des gestionnaires tout en reposant sur des concepts abstraits bien structurés. Nous voulons lui donner une syntaxe quasi-libre, ne gardant comme impérative que la substantifique moelle de la grammaire abstraite.

Cette possibilité est apportée par le logiciel d'aide à la conception dont la fonctionnalité est décrite partie II de cette thèse.

PARTIE II

L A S S I F

Logiciel d'Aide à la construction

de la Spécification

d'un Système d'Informations

1. INTRODUCTION

La partie I nous a conduit à faire un certain nombre de propositions en termes de modélisation des SI. Nous y avons développé une formalisation de SI associée à un langage permettant sa spécification, puis esquissé quelques traits de la méthode choisie. Cet ensemble de propositions est insuffisant, dans l'évolution actuelle des idées, pour en faire une méthode autonome de conception des SI.

En effet concevoir un Schéma Conceptuel, puis réaliser le SI correspondant, est une tâche complexe où de nombreux acteurs interviennent et où il y a un grand nombre d'informations disparates et de concepts à manipuler simultanément. Il faut aider le concepteur en le déchargeant au maximum de ce qui ne relève pas de son activité.

Un logiciel d'aide à la construction de la spécification d'un SI est nécessaire à ce niveau ; il doit assurer les rôles traditionnels de contrôle, documentation et construction des spécifications entrées, sous la direction d'un outil privilégié que nous appelons le "pilote".

Après avoir défini précisément les fonctions d'un tel système, réfléchi sur les différentes démarches d'analyse que l'on peut proposer lors de la spécification d'un SI, nous présentons notre réalisation actuelle (et ses développements futurs) dans l'environnement LISP/CEYX. Nous conclurons par une présentation rapide de quelques systèmes existants.

2. LES FONCTIONS D'UN LOGICIEL D'AIDE A LA SPECIFICATION DES SI

2.1. Les caractéristiques d'un tel logiciel

Nous donnons, ici, quelques idées sur ce que peut être un logiciel de conception assistée des SI intégrant les nouveaux développements des environnements de programmation, dont on peut trouver trace dans les actes de la conférence "7th Conference on Software Engineering" (Orlando USA, Mars 1984) [GUY 84, JOH 84, FIS 84, SHA 84, STU 84].

A notre sens, un logiciel de CAO des SI, doit être :

- un environnement intégré de spécification,
- un environnement de vérification,
- un environnement de documentation,
- un environnement de pilotage.

A. Un environnement intégré de spécification

La première qualité que doit posséder un logiciel de CAO des SI est, par analogie aux environnements de programmation et aux ateliers logiciels, de proposer un ensemble d'outils non seulement fortement connectés mais conçus de manière intégrée, manipulant les mêmes structures de données, les mêmes commandes et les mêmes éditeurs. Le concepteur, pour décrire ses spécifications, doit pouvoir accéder à un système conversationnel :

- soit traditionnel (ainsi que nous l'avons réalisé, cf. paragraphe 4.4) par questions/réponses effectuant la saisie intégrée de la spécification,

- soit, disposant d'écran et d'éditeur vidéo, par affichage de la syntaxe concrète du langage en laissant des "trous" à remplir par le concepteur pour compléter la phrase de la spécification. Cette deuxième solution permet une syntaxe concrète du langage quasi-libre et paramétrable puisque seule la syntaxe abstraite, base de construction de l'arbre abstrait associé à la spécification, n'est retenue et n'a, de fait, de l'importance.

B. Un environnement de vérification

Il est impératif que les différents contrôles des spécifications entrées aient lieu de façon interactive et la plus conviviale possible ; l'utilisateur prend connaissance de l'erreur, si cela est possible il la corrige, l'outil effectuant les retours en arrière nécessaires. Il est souhaitable que ces contrôles puissent être lancés de l'outil sans quitter l'environnement, ou comme dans EDME [GUY 84, JAC 84] qu'ils soient directement intégrés aux commandes de l'éditeur.

C. Un environnement de documentation

Il doit permettre une documentation associée au système lui même et sur les spécifications entrées.

(i) associé au système : cette "documentation en ligne" permet d'interroger le système sur le format précis d'une commande, par exemple, ou d'un appel à un processeur. Cette aide permet au concepteur de rejeter de ses préoccupations tout problème de syntaxe pour ne se concentrer que sur sa spécification. Pour être efficace, une telle documentation doit être associée à des commandes simples, voire des touches fonctions, et la réponse doit être courte (tenir sur un écran, voire sur une fenêtre).

(ii) sur les spécifications entrées : à un point donné de la spécification, le concepteur doit pouvoir disposer, à sa demande, d'un affichage des descriptions qu'il a déjà faites. Cette documentation doit être associée à un langage simple (voire des commandes ou des touches fonctions) ; le principal rôle du système étant d'affiner la demande en collaboration avec le concepteur, si la réponse s'avère trop longue.

D. Un environnement de pilotage

C'est la caractéristique la plus importante et la moins classique d'un logiciel de CAO. Le logiciel doit être associé à une démarche, c'est-à-dire à un canevas ou fil conducteur précis, où, à chaque étape, le concepteur sait les actions qu'il peut entreprendre, les choix qu'il a à faire. On peut compléter ce rôle en proposant au concepteur une "sorte de documentation en ligne" sur la méthode elle même ; lui donnant, à chaque instant, la liste des actions qu'il peut entreprendre, à partir de ce qui a déjà été spécifié.

Remarquons que cette démarche d'analyse pourrait ne pas être figée mais qu'alors l'outil "pilote", dirigeant la conception, pourrait être paramétré par le type de démarche choisie. C'est la solution qu'a choisie J.L. CAVARERO [CAV 79] pour le projet LAPAGE développé à Nice.

Nous même [ROL 80A] avons réalisé un pilote, autour du SGBD SOCRATE [SOC 77] qui, s'appuyant sur le schéma conceptuel du logiciel de CAO, (c'est-à-dire sur un ensemble de règles méthodiques) dirigeait la saisie des spécifications selon ces règles.

2.2. Architecture du logiciel

Le logiciel a pour but d'aider le concepteur à construire un schéma conceptuel de "qualité" sous la direction d'un "pilote".
Le schéma conceptuel est stocké progressivement dans une méta-base (nommée ainsi car elle décrit la structure du SI et non les occurrences) qui contient l'ensemble des instanciations des schémas de type CLASSE-OB, CEVENT, COPERATION et SI constituant la structure du SI (la partie III de cette thèse en présente une solution relationnelle).

Le logiciel réalise les fonctions suivantes :

- saisie et contrôle des spécifications statiques,
- construction progressive et intégrée du schéma conceptuel,
- documentation du schéma conceptuel ;

ces fonctions étant agencées par l'outil "pilote".

(i) saisie et contrôle des spécifications statiques

Le schéma conceptuel peut être vu comme la base de connaissance du SI ; les règles sont alors un ensemble de définitions formelles concernant la structure et la dynamique de l'univers du discours. Le résultat de la modélisation est un ensemble de formules correctes exprimées à l'aide du langage de spécification, fondé lui même sur un modèle formel, et pouvant donc être contrôlées et prouvées. Les solutions que l'on peut proposer pour contrôler une spécification donnée sont basées sur des contrôles syntaxiques et sémantiques, la saisie étant interactive et aussi conviviale que possible.

(ii) construction progressive et intégrée du schéma conceptuel

Il s'agit là de vérifier que la spécification, décrite et contrôlée précédemment intrinsèquement, s'intègre bien dans les spécifications déjà entrées. C'est ce que nous appelons [THI 76] la fonction "intégration" pour laquelle nous avons proposé des contrôles de détection de synonymes, polysèmes et incomplétudes.

(iii) documentation du schéma conceptuel

La connaissance apportée par le schéma conceptuel permet de répondre aux questions des concepteurs non seulement sur les faits contenus dans le SI mais aussi sur les concepts, les hypothèses et les règles elles mêmes. Cette base de connaissance est support de documentation et d'aide à la décision ; en effet le schéma conceptuel constitue un stock documentaire, concrétisé dans la méta-base. Elle correspond à ce que MERISE [LE 84] nomme la meta-base d'information (MBI). Un ensemble d'outils lui est associé visant à sa gestion au cours du temps et à son utilisation par des interrogations (le GBI de MERISE).

L'architecture du logiciel d'aide à la spécification d'un SI peut être schématisée ainsi :

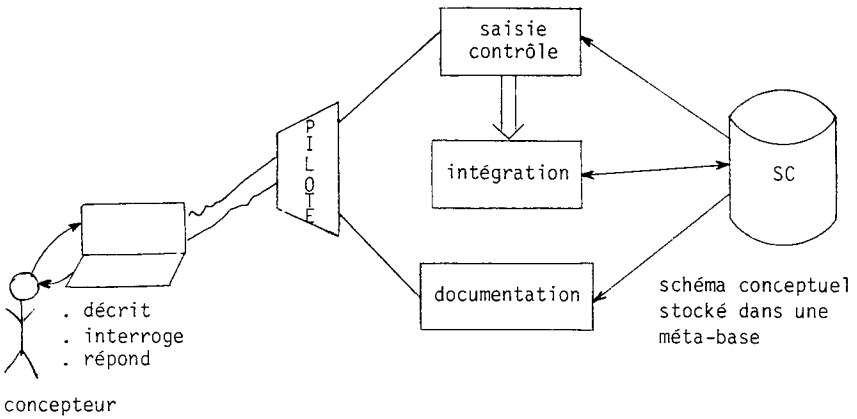


Figure 1. Architecture d'un logiciel d'aide à la spécification des SI

3. VERS UNE METHODE D'ANALYSE DES SPECIFICATIONS ENTREES

3.1. Hypothèses

Au chapitre III de la partie I nous avons fixé quelques hypothèses de travail dont il faut retenir que :

- nous nous intéressons à un SI global ou, dans le cas d'organisations trop vastes, à des sous-systèmes du SI indépendants les uns des autres et nous ne nous posons pas le problème de la détermination de ces sous-systèmes ;

- nous pensons que, si le logiciel d'aide à la conception, s'appuyant sur la démarche d'analyse choisie, peut guider le concepteur, voire effectuer à sa place certaines déductions, il ne peut le remplacer totalement.
En effet un système de CAO des SI est incapable de modéliser la compréhension du monde réel dans sa totalité ; le concepteur reste maître de la spécification ;

- nous en concluons que l'utilisateur d'un logiciel n'exclut pas une réflexion préliminaire sur "papier", après, par exemple, entretiens avec les futurs utilisateurs du SI, afin d'établir un consensus sur le contenu futur du SI.

3.2. Inventaire des démarches possibles

On retrouve ici trois grands types de démarches d'analyse possibles correspondant aux trois modes de raisonnement de modélisation :

. une démarche orientée traitements,
. une démarche orientée données,
. une démarche orientée événements.

(i) orientée traitements : le point de départ de cette démarche est l'analyse des fonctions de la procédure existante (éventuellement productrices de documents) ; pour respecter nos concepts, il faut modéliser les fonctions en termes de types d'opérations puis analyser les objets manipulés par le système.
Nous avons déjà noté que cette démarche d'analyse, la première historiquement parlant, a pour principal inconvénient de produire des SI trop calqués sur le système existant et composés de procédures juxtaposées.
Remarquons que, si la démarche permet d'identifier les fonctions principales et si le logiciel propose, comme le réalise SPES [QUE 84] [LEV 84], des fonctions de transformation permettant de restructurer la description, une telle solution est réaliste.
En outre, dans notre modèle conceptuel, le concept de classes d'opérations n'est pas central ; il est fortement lié, dans sa définition, au concept de classe d'objets et de classe d'événements. La modélisation ne peut donc reposer uniquement sur la recherche des classes d'opérations de la réalité.
Notons, cependant, qu'avec un autre modèle sous-jacent, tel celui de SADT [ROS 77] par exemple, cette démarche d'analyse pourrait se justifier.

(ii) orientée données : le point de départ de cette démarche est l'étude des composantes statiques de l'organisation, c'est-à-dire des états d'un SI, sachant que l'état d'un SI est défini par l'ensemble de ses objets.
C'est une démarche Bases de Données traditionnelle qui consiste à faire l'inventaire des objets de l'organisation avec deux questions principales à l'esprit [BEN 79] :

. quels sont les objets et les associations sur lesquels l'organisation souhaite être informée ?

. quelles sont les propriétés qui permettent de les caractériser de manière satisfaisante ?

Les traitements, que peuvent subir les objets, sont analysés ultérieurement et traduits sous forme de contraintes d'intégrité.
Cette démarche constitue, en partie, le point de départ de l'analyse que nous avons choisie, le concept de classe d'objets étant central dans notre modélisation. Il n'est, cependant, pas possible de se limiter à l'analyse des classes d'objets de l'organisation indépendamment de tout autre concept, ne serait ce qu'à cause de la difficulté qu'il y a de faire un tel inventaire. De plus, la démarche orientée données a pour inconvénient, nous l'avons dit, d'ignorer les propriétés dynamiques de l'organisation c'est-à-dire la façon dont un SI passe d'un état à un autre. Elle nous semble donc incomplète.
Notons que, dans une optique de modélisation orientée objet, telle que la préconisent SMALLTALK [GOL 83], BRACCHI [BRA 76], ISAC [LUN 82A], ... une démarche de ce type se justifie.

(iii) orientée événements : le point de départ de cette démarche est l'étude des événements pouvant se produire dans l'organisation, le plus simple étant de respecter la typologie des événements :

. externes c'est-à-dire d'origine extérieure au SI (par exemple un appel téléphonique, l'arrivée d'un bon de commande),

. internes c'est-à-dire produits par le SI lui même (par exemple la production d'un bon de livraison à transférer au magasin, une rupture de stock).

Le concept principal qui permet de refléter la dynamique d'une organisation est le concept d'événement. C'est en effet lui qui exprime le changement d'état du SI. Cependant, à notre sens, il est fortement lié au concept d'objet puisqu'un événement se produisant dans une organisation peut être vu comme le changement d'état d'un objet (éventuellement concrétisé par un document, un téléphone ou tout autre support) ; et, afin d'exprimer la transformation du SI, il est lié au concept d'opération puisque les actions à déclencher à la suite d'un événement peuvent être vues comme des opérations dans le SI.

Nous préférons, à une approche purement orientée objet ou purement orientée événement, une approche intégrant les deux aspects. C'est, ce que nous appelons, une approche orientée dynamique.

3.3. La démarche choisie

Pour aider le concepteur à faire l'inventaire des composantes statiques et dynamiques, commencer par la représentation graphique du sous-schéma des données (les classes d'objets et d'associations) et du sous-schéma de la dynamique (les classes d'événements et d'opérations), telle que nous la présentons en annexe, nous semble la meilleure solution.
Dans un deuxième temps, le logiciel permet la saisie des différentes classes selon le canevas choisi. Plusieurs solutions sont possibles pour définir la "démarche d'analyse" :

(i) soit privilégier la description d'un cycle dynamique, c'est-à-dire la spécification d'un c-objet, du type d'événement associé à sa modification et de l'ensemble des types d'opérations susceptibles d'être déclenchés à la suite de l'événement ; c'est bien entendu un processus récursif puisque les opérations modifient des objets qui, à leur tour, peuvent être associés à des événements etc... Cette solution, choisie par D. BANON dans [BAN 79] est très conviviale : en effet elle est "parlante" et évite des erreurs car toute description est liée à la précédente. L'outil a moins de vérifications à faire en termes de complétude mais cette saisie impose une conception du logiciel en modules fortement liés puisque manipulant les mêmes structures de données.

(ii) soit privilégier la description des classes par rapport à celle des cycles. Cette solution a pour avantage de permettre la saisie différée des composantes dynamiques par rapport à celle des composantes statiques. Elle est par contre moins conviviale car le concepteur est amené à décrire les liens des composantes dynamiques avec les composantes statiques. Cette solution, choisie par O. FOUCAUT dans [FOU 82], est également la nôtre dans LASSIF.

Elle permet une conception du logiciel d'aide à la spécification en modules indépendants.

La démarche d'analyse choisie repose sur une étude intégrée des classes d'objets, classes d'événements et classes d'opérations de l'organisation. Plus précisément, il s'agit de faire le bilan :

- des différentes classes d'objets associées à des classes d'événements d'origine externe,
- des différentes classes d'opérations conséquentes de ces classes d'événements, en précisant leurs conditions de déclenchement,
- des conséquences, en termes de classes d'objets et de classes d'événements d'origine interne, de ces classes d'opérations.

Le résultat de cette analyse est saisi sous la direction du pilote ; successivement (cf. chapitre III, partie I, paragraphe 3.2.2) :

a) saisie des composantes statiques :

. des classes d'objets munies de leurs paramètres fixes,

. du sous-schéma (c-objet) permanent de chaque classe d'objets muni de ses paramètres fixes et variables,

. des sous-schémas variables de chaque classe d'objets munis de leurs paramètres fixes et variables,

. des fonctions structurelles (dépendances fonctionnelles) existant entre les classes d'objets ;

b) saisie des composantes dynamiques :

. des classes d'événements munies de leurs paramètres fixes,

. des classes d'opérations munies de leurs paramètres fixes en association avec les classes d'événements et d'objets, ce qui induit la cohérence de la spécification.

4. NOTRE REALISATION

Notre réalisation, bien que concernant essentiellement le logiciel d'aide à la spécification (LASSIF) des composantes statiques et dynamiques, est en fait partie intégrante d'un logiciel de gestion des systèmes d'information, atelier général que nous nommons LGSI.

4.1. Le LGSI

Ce type de logiciel ayant déjà été étudié dans l'équipe REMORA ([BAN 79], [LEI 80], [FOU 82]), nous en rappelons rapidement les rôles et explicitons ses composantes avant d'entrer dans le détail de notre réalisation.

Le LGSI résulte de l'extension aux Systèmes d'Information des fonctions qui sont assignées habituellement aux Systèmes de Gestion de Bases de Données : c'est-à-dire la manipulation des données - en offrant des aides à l'interrogation / mises à jour - et l'exécution automatique des opérations de contrôle sur les Données afin de maintenir à la fois l'intégrité et la cohérence des Données.

Le SI, tel que nous le définissons, est une représentation plus complète de la réalité qu'une Base de Données car il inclut simultanément une description statique (structurelle) de l'organisation et sa description dynamique (fonctionnelle). Les fonctions d'un LGSI sont donc plus complexes que celles d'un SGBD car elles comprennent, non seulement la manipulation cohérente des données, mais aussi le contrôle des opérations sur les données et des événements qui déclenchent ces opérations.
Ses rôles sont donc :

(i) de réduire le travail du concepteur à celui de l'analyse et de la représentation des faits réels et d'aider le concepteur dans la formalisation de cette analyse,

(ii) de faire en sorte que le SI soit en permanence une image fidèle de la réalité à représenter, en gérant son évolution au cours du temps,

(iii) de permettre une utilisation variée, souple et simple des informations contenues dans le SI.

Pour remplir ces différents rôles, nous avons conçu le LGSI selon deux composantes principales :

- le logiciel d'aide à la construction de la spécification d'un SI (LASSIF) dont nous venons d'exposer les principes,
- le logiciel d'interprétation de la spécification du SI (LISSIF) qui a pour but, à partir de la spécification du SI obtenue à l'aide de LASSIF, de construire le SI résultant, de le faire vivre au cours du temps en accord avec la vie de l'organisation et de permettre son utilisation. Le paragraphe 4.5. l'explicite plus précisément.

4.2. Choix d'un logiciel de développement

A. Analyse des différentes solutions

Lorsque l'on réalise un logiciel, plusieurs choix sont possibles quant à l'environnement de développement :

(i) on peut choisir un système "clos", c'est-à-dire se placer dans un environnement qui en principe permet de gérer les données et les traitements sur les données avec un même langage et un même système d'exploitation. C'est le choix que nous avions pris [ROL 80A] pour réaliser le pilotage de la CAO autour d'un SGBD SOCRATE : en fait le langage de requête est en principe autonome mais peu adapté au logiciel que nous avions construit ;

(ii) on peut choisir de faire "cohabiter" des langages et des systèmes de gestion de données évolués et, alors, de résoudre les problemes d'interface se posant obligatoirement. C'est le choix

que nous avions fait [THI 82] pour réaliser le logiciel de gestion de la dynamique d'un SI (cf. partie III) autour du SGBD SYNTEX : nous nous sommes heurté à de grosses difficultés analogues à celles que rencontrent les concepteurs des systèmes d'exploitation ;

(iii) une troisième voie consiste à choisir un des environnements de programmation existant actuellement, où tous les composants sont conçus de façon intégrée et extensible et où, donc, il n'y a pas, en principe, de problème d'interface ni de compatibilité. En effet, la production de logiciel est une tâche complexe où on a intérêt à utiliser au maximum des logiciels existants, en rajoutant, éventuellement, des couches supplémentaires pour effectuer des fonctions complémentaires.

Notons, de plus, que le schéma conceptuel d'un SI, décrit avec nos schémas de type CLASSE-OB, CEVENT, COPERATION et SI peut, par définition des concepts, être vu comme une arborescence (cf. schématisation du type SI au chapitre II partie I) ; et qu'il est assez naturel de voir les textes de programmes comme des arbres syntaxiques abstraits lorsqu'on dispose de la grammaire du langage.

Ces différentes remarques nous ont conduit à choisir le constructeur d'Arbres Abstraits CEYX [HUL 83-84], sursystème de LISP [WIN 81], comme logiciel de développement ; et donc à choisir LISP comme langage de développement. Nous avons pu ainsi profiter de la puissance de LISP quant à la génération dynamique d'instructions, l'exécution dynamique de fichiers programmes, l'utilisation du lambda calcul et des fonctions prédéfinies.

B. Avantages de cette solution

(i) Utiliser un éditeur syntaxique tel que CEYX permet de disposer d'un ensemble de fonctions de manipulation des arbres abstraits ; ce qui facilite largement la programmation de la construction des arbres.

(ii) Définir la grammaire abstraite d'un langage oblige à un effort de formalisation qui permet de détecter assez vite les erreurs de conception. De plus CEYX générant automatiquement les fonctions de construction de l'arbre associées aux opérateurs noeuds, il n'y a pas à les programmer explicitement.

(iii) On peut attacher aux opérateurs noeuds des fonctions sémantiques, par exemple d'analyse syntaxique, mais aussi d'affichage ce qui permet de vérifier tout de suite, éventuellement par morceaux, si un arbre est correctement construit et facilite, donc, la mise au point.

(iv) On dispose, bien sur, de toutes les possibilités qu'offre l'environnement LISP lui même.

En résumé, la puissance de ce genre d'environnement en général et de LISP en particulier est évidente. La possibilité de générer des instructions, d'exécuter des textes de programmes de façon dynamique conduit à une solution en termes de fonctions LISP modulaires très courtes, beaucoup moins lourde que les logiciels réalisés précédemment dans l'équipe [BAN 79, ROL 80A, FOU 82].

C. Inconvénients de cette solution

Le principal inconvénient concerne les structures de Données, ainsi que nous le montrerons dans les développements suivants :

(i) Nous ne disposons pas de Système de Gestion de Bases de Données associé à LISP/CEYX alors que le volume des informations l'emporte sur le volume des traitements en Informatique de Gestion.

(ii) Les accès aux fichiers sont lourds, complexes et (surtout) sont modifiés d'une version à l'autre de LISP.

D. Présentation du logiciel CEYX

CEYX a été développé initialement à l'INRIA comme une aide au développement de circuits VLSI et a été étendu pour permettre la manipulation de hiérarchies quelconques telles que des Bases de Données, des documents, des programmes ... CEYX est actuellement un ensemble d'outils permettant de faciliter la tâche des programmeurs LISP pour définir et manipuler des structures abstraites. Aux structures définies en CEYX sont associés des espaces sémantiques, organisés hiérarchiquement, dans lesquels sont rangées les fonctions de manipulation propres à ces structures. Un style de programmation orientée objets à la SMALLTALK [GOL 83] est ainsi possible. On peut donc voir CEYX comme une extension de LISP qui propose à l'utilisateur un ensemble de fonctions de création et de manipulation d'objets. Ces objets sont arrangés en familles de manière hiérarchique de telle façon qu'ils soient munis des propriétés sémantiques de leurs ancêtres. CEYX permet de créer ces structures, de définir automatiquement des fonctions LISP de création d'instances de ces structures ainsi que de définir automatiquement des fonctions d'accès en lecture et en écriture aux composantes de ces instances.

Les deux concepts principaux de CEYX sont le modèle et le type.

(i) le modèle

Ce concept permet de définir une structure de données, soit en spécifiant ses champs, soit par une assertion prédicative, et de lui associer un "espace sémantique" c'est-à-dire un ensemble de fonctions permettant de construire des instanciations du modèle puis de les manipuler.

CEYX permet de définir des objets de type modèle et propose des fonctions prédéfinies pour générer les fonctions sémantiques associées au modèle.

On trouvera en annexe 5 un exemple de définitions de modèles pour spécifier des types de LASSIF.

(ii) le type

Sa différence avec le concept de modèle est que les instances d'un type sont étiquetées par le nom du type qui les a engendrées. Les objets du type ont ainsi accès à l'espace sémantique (les fonctions) associé à ce nom.

Pour faciliter la manipulation des arbres, CEYX propose deux constructions supplémentaires : deftree et defcons.

(iii) l'arbre

La construction "deftree" permet de définir des arbres, éventuellement hiérarchisés, éventuellement avec des attributs sémantiques.

(iv) le constructeur

La construction "defcons" permet de spécifier comment sont structurés les fils de l'arbre auquel est rattaché le constructeur.

En quelque sorte il définit les opérateurs noeuds des arbres abstraits que l'on peut construire, par exemple à partir de la grammaire abstraite d'un langage. La construction "defcons" permet de définir des types composés :

. le premier constituant du type est un ensemble de fils repérés par leurs noms, si le noeud est d'arité fixe, ou par une liste, si le noeud est d'arité variable,

. le deuxième constituant est un ensemble d'attributs permettant d'étiqueter l'arbre.

Utiliser ces deux concepts (cf. annexe 5) comme constructeurs de types permet de disposer de l'espace sémantique (ensemble des fonctions de construction et manipulation) des arbres, généré automatiquement par CEYX.

Dans notre cas particulier décrire l'agencement possible des objets que nous avons à manipuler revient à donner la grammaire abstraite de LASSIF. On trouvera, en annexe 5, la grammaire CEYX du langage structurel et, en annexe 9, celle du langage d'énoncé.

4.3. Implémentation de la démarche d'analyse

A. Un choix de description : des transactions

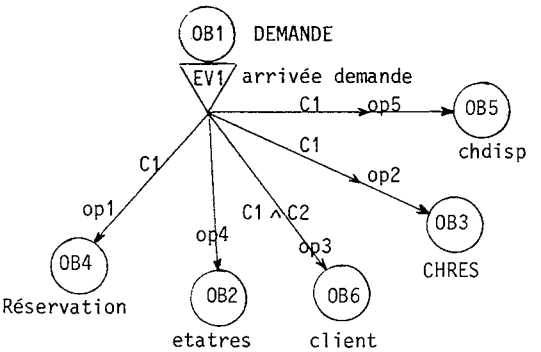
Nous avons choisi d'implémenter la démarche d'analyse décrite au paragraphe 3.3. en l'étendant à la saisie de transactions.
Le concepteur spécifie le schéma conceptuel d'un Système d'Information non pas de façon élémentaire c'est-à-dire type d'opération par type d'opération munis de leur condition et facteur de déclenchement mais sous forme de modules complets correspondant au moins à un cycle dynamique.

Définition du cycle dynamique

Un type d'événement, les types d'opérations déclenchées, les c-objets modifiés correspondent à un module élémentaire ou cycle dynamique où le paramètre d'entrée est le c-objet associé au type d'événement et les paramètres de sortie sont les c-objets modifiés. Les raisons qui nous ont poussés à ce choix sont, en premier lieu, d'ordre "spécification du SC".

(i) spécification du sc

Quel que soit le langage de spécification proposé, il est difficile de demander, au niveau conceptuel, une description de chaque instanciation de type de façon élémentaire surtout au niveau des textes de facteur, prédicat, condition et opération. En effet on peut être amené à redécrire plusieurs fois la même séquence de spécification.
Prenons, par exemple, le cycle dynamique associé à la demande de réservation de chambres d'hôtel dans une station de sports d'hiver. L'hypothèse du système est que l'on n'effectue la réservation (création de OB4, modification de OB5 et OB3) et que l'on ne crée le client (OB6) que si la demande de réservation peut être satisfaite. Le schéma dynamique en est le suivant :



C1 : la demande peut être satisfaite
C2 : le client n'existe pas

Pour évaluer C1, il faut explorer l'ensemble des disponibilités du système (chdisp), vérifier qu'il existe suffisamment de chambres disponibles, de catégorie demandée, dans la station demandée pour satisfaire la demande de réservation. Or le texte de l'opération op5 reprend cette même définition pour effectuer réellement la réservation ; d'où une certaine lourdeur fastidieuse de spécification.

Cet inconvénient serait relativement léger si le système était capable de déduire une structuration logique des traitements et des données directement à partir de la spécification conceptuelle ; mais tel n'a pas été notre propos. Nous en déduisons une deuxième raison d'ordre "fonctionnement du SI".

(ii) fonctionnement du SI

C'est surtout lors de la gestion de l'évolution du SI au cours du temps par le sous-ensemble du LGSI appelé "processeur de la dynamique" (cf. paragraphe 4.5 et partie III) que le découpage élémentaire des cycles dynamiques paraît lourd voire incohérent à faire fonctionner de façon synchronisée. En effet :

- si, au cours d'un même cycle dynamique, plusieurs opérations sont déclenchées avec la même condition (ou des conditions qui s'excluent) la condition est réévaluée pour chaque déclenchement : ainsi, pour notre exemple, C1 sera évaluée quatre fois,

- si des textes de condition et d'opération possèdent, en partie les mêmes séquences d'instructions (par exemple C1 et op5), l'évaluation de la séquence sera effectuée deux fois ; cela pose en outre des problèmes de synchronisation car il ne faut pas qu'il y ait de changement dans le SI entre l'évaluation de la condition et le déclenchement de l'opération,

- la communication par fichiers est coûteuse en temps, il vaut mieux évaluer les facteurs de déclenchement dans le même module que les opérations correspondantes et conserver les nuples dans des tableaux en mémoire centrale.

Ces constatations nous ont conduits à penser qu'il vaudrait mieux faire exécuter un module complet comprenant un ensemble de conditions, facteurs et opérations à déclencher en une seule fois. C. RICHARD et C. ROLLAND [ROL 82] ont introduit des règles de modélisation logique des traitements que nous avons reprises à notre compte au niveau conceptuel. Ces règles sont du même type que les propositions de F. BODART dans IDA [BOD 84] ou celles de la méthode MERISE [LE 84].

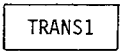
B. Règles de regroupement en transactions

Nous n'en donnons ici que quelques grandes lignes, on se reportera à [ROL 82] pour plus de détails. Les règles de regroupement reposent sur deux nouveaux concepts : la transaction et le trigger.

(i) Transaction

C'est une séquence d'instructions exécutée dans cet ordre, reconnue par son nom et ayant un et un seul point d'entrée. Elle est décrite par un bloc ou une procédure avec un paramètre d'entrée et plusieurs

paramètres de sortie. On la représente ainsi :



(ii) Trigger

C'est une condition (ou prédicat) portant sur l'état du SI et permettant de déclencher l'exécution d'une et une seule transaction. Elle peut être nommée et il lui correspond un texte ou une fonction booléenne. Elle peut être vérifiée, soit après l'exécution de une ou plusieurs transactions (déclenchement interne), soit après l'arrivée d'un stimulus externe. On la représente ainsi :

Une arête orientée d'un trigger vers une transaction exprime que le trigger déclenche l'exécution de la transaction. Inversement, une arête orientée d'une transaction vers un trigger exprime que l'exécution de la transaction contribue à la validation du déclenchement.

Un schéma de synchronisation de transactions est un graphe bi-alterné sur l'ensemble des transactions et des triggers c'est-à-dire un graphe où un noeud de type transaction suit toujours un noeud de type trigger et inversement.

C. ROLLAND et C. RICHARD ont introduit cinq règles, basées essentiellement sur la notion de dépendance chronologique entre classes d'événements, qui permettent de définir le schéma de synchronisation à partir des transactions élémentaires c'est-à-dire des cycles dynamiques.

L'application de ces règles conduit au regroupement de transactions élémentaires en une transaction plus importante (avec des gains de temps d'exécution considérables) et permet de trouver des conditions d'indépendance entre traitements ; ce qui, dans un environnement permettant le parallélisme, induira un déroulement possible en parallèle des transactions concernées.

4.4. Entrée des spécifications statiques à l'aide du logiciel

Le logiciel d'aide à la construction de la spécification d'un SI (LASSIF) a été conçu en couches successives à partir du noyau LISP/CEYX selon le schéma suivant :

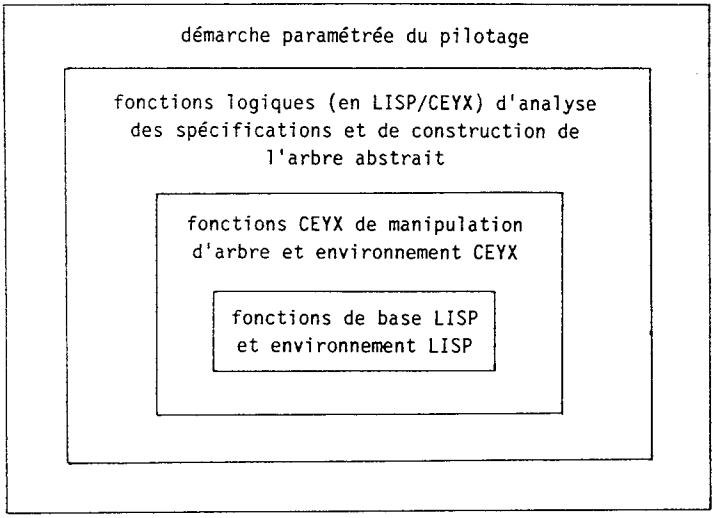


Figure 2. Réalisation en couche

Remarquons, en particulier, que la démarche choisie relève du niveau le plus haut. Elle est implémentée, dans LASSIF, sous forme d'un module directeur gérant les appels aux différents autres modules, et pourrait donc être aisément modifiée.

A. Fonctions du logiciel

La spécification du SI (son schéma conceptuel) est construite progressivement par LASSIF sous forme d'un arbre abstrait qui représente donc la structure des classes d'objets, d'événements et d'opérations du Système d'Information.

Le système est composé actuellement d'un ensemble de modules réalisés en LISP/CEYX permettant :

(i) la saisie conversationnelle par un système de questions/réponses, sous contrôle d'un module directeur,

- des classes d'objets,
- des classes d'événements (les triggers),
- des classes de transactions,
- des dépendances fonctionnelles entre classes d'objets,
- des déclarations de type : le module permet ici, de définir les sous-schémas variables des classes d'objets.

(ii) la construction progressive et intégrée du schéma conceptuel d'un SI sous forme d'un arbre abstrait dont les noeuds sont des opérateurs (constructeurs) de la grammaire et dont les attributs des feuilles sont les éléments du schéma conceptuel (un exemple de S.C., sous forme d'arbre abstrait, est donné en annexe 8).

(iii) la documentation systématique des spécifications entrées : elle est produite par le lancement de fonctions sémantiques d'affichage permettant une visualisation progressive de l'état de la description.

Notons que, dans la version précédente de CEYX (version 13), on disposait, de plus, du contrôle automatique des spécifications entrées ; en effet le type des fils d'un constructeur d'un arbre déterminait la vérification des descriptions. La version actuellement disponible (version 15) ne contrôle plus le type des fils reconnus, d'ailleurs, par leurs noms.

B. Détails du système

L'architecture du système, actuellement opérationnel, est visualisée par la figure suivante :

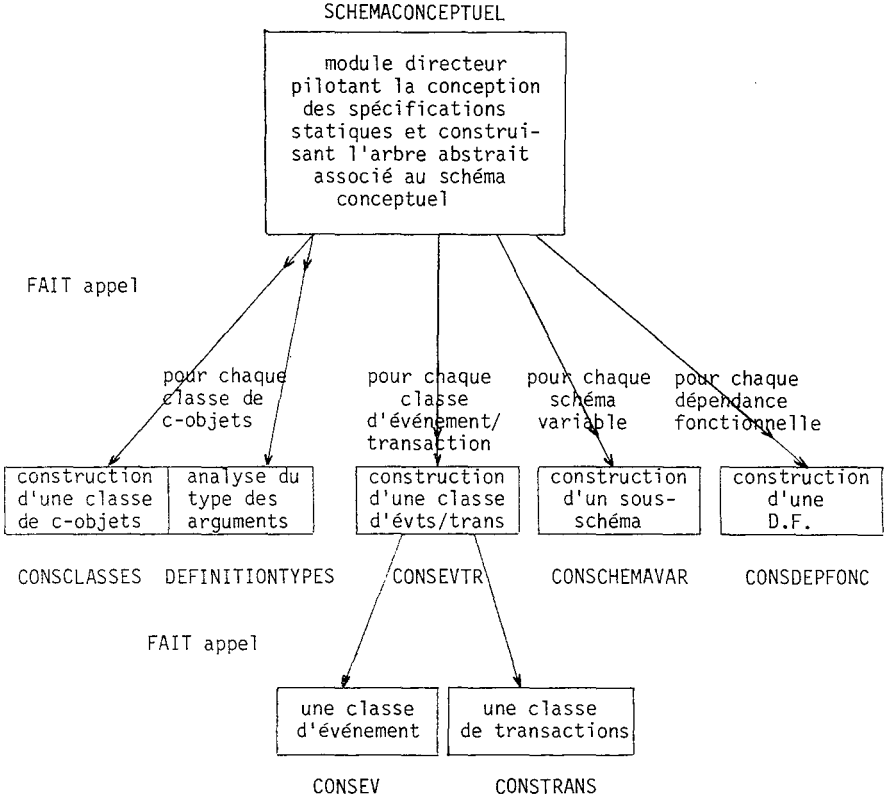


Figure 3. Architecture de LASSIF

Les modules des deuxième et troisième niveaux font, eux mêmes, appel à des modules d'analyse et de décodage syntaxique afin de reconnaître le type des arguments entrés et d'en déduire le type de construction à réaliser dans l'arbre.

On trouvera en Annexe :

- 5 . la grammaire CEYX du langage structurel,
- 6 . le listing des différents modules LISP de la figure 3,
- 7 . un extrait du déroulement de l'exécution et le compte rendu de l'affichage du SC après le déroulement complet de l'outil,
- 8 . un arbre abstrait correspondant à un sous-ensemble de l'exemple des réservations.

Nous donnons, ici, les algorithmes correspondant aux principaux modules réalisés en LISP donnés en annexe 6.

(i) module SCHEMA CONCEPTUEL

explication

Module LISP sans paramètre dirigeant la conception, il est lancé par le concepteur.

algorithme

α) . initialisation des arbres abstraits "schéma" et "base" correspondant respectivement à la construction du schéma conceptuel (opérateur "scop") et du sous-ensemble de description des classes (opérateur "baseop") ;

β) . tantque rep = 'n'
- construire une classe d'objets (CONSCLASSES)
- saisir les composantes permanentes et variables (DEFINITIONTYPES)
- est-ce la dernière classe d'objets (o/n) ?

ftque

. construire le sous-arbre associé à la liste des classes d'objets et l'associer à "base" (deuxième fils) ;

γ) . tantque rep = 'n'
- construire une classe d'événements/transactions (CONSEVTR)
- est-ce la dernière classe (o/n) ?

ftque

. construire le sous-arbre associé à la liste des classes d'événements/transactions et l'associer à "base" (troisième fils) ;

. associer "base" à "schéma" (premier fils) ;

δ) . tantque rep = 'n"
- saisir la définition d'un sous-schéma variable (CONSCHEMAVAR)
- est-ce le dernier sous-schéma (o/n) ?
ftque

. construire le sous-arbre associé à la liste des sous-schémas variables et l'associer à "schéma" (deuxième fils) ;

ε) . tantque rep = 'n'
- saisir la définition d'une dépendance fonctionnelle (CONSDEPFONC)
- est-ce la dernière dépendance fonctionnelle (o/n) ?
ftque

. construire le sous-arbre associé à la liste des dépendances fonctionnelles et l'associer à "schéma" (troisième fils) ;

(ii) module CONSCLASSES

explication

Module LISP sans paramètre construisant une classe d'objets.

algorithme

α) . saisir les différents paramètres d'une instanciation de type CLASSE-OB ;

β) . analyser la forme du paramètre CLE (ANALYSE1 CLE) ;
Ce paramètre peut être de deux formes :
- soit un identificateur de type en majuscules indiquant que son type sera défini ultérieurement (DEFINITIONTYPES),
- soit directement entré sous forme d'un produit cartésien, il est alors reconnu et décodé par le module (CONSTRUCTPROD CLE) ;

γ) . analyser la forme du paramètre schéma permanent SCHEMAP (ANALYSE2 SCHEMAP) ;
comme précédemment ce paramètre peut être soit un identificateur de type, défini ultérieurement (DEFINITIONTYPES), soit un produit cartésien (CONSTRUCTPROD SCHEMAP) ;

δ) . construire le sous-arbre associé à la classe "classeob" (à l'aide du constructeur "classeop").

(iii) module DEFINITIONTYPES

explication

Module LISP sans paramètre saisissant :
- s'ils n'ont pas été donnés, directement sous forme de produits cartésiens, la clé et le schéma permanent d'une classe d'objets,
- les identificateurs de types des sous-schémas variables.

algorithme

α) . si type (CLE) = "chaine"
alors - saisir la clé (TYPECLE) sous forme d'un produit cartésien
- (CONSTRUCTPROD TYPECLE)
fsi

β) . si type (SCHEMAP) = "chaine"
alors - saisir le schéma permanent (TYPECOMPP) sous forme d'un produit cartésien
- (CONSTRUCTPROD TYPECOMPP)
fsi

γ) . saisir le schéma variable sous forme d'un produit cartésien d'identificateurs de type de sous-schémas variables ; analyse et décodage (CONSCOUPLEV TYPECOMPV)

δ) . associer les résultats des étapes précédentes à la classe d'objets "classeob" et rajouter la classe construite à l'arbre des classes d'objets "listeclasseob".

(iv) module CONSEVTR

explication

Module LISP sans paramètre construisant une classe d'événements/transactions.

algorithme

α) . saisir les caractéristiques de l'événement (CONSEV) ;

β) . saisir les caractéristiques de la transaction (CONSTRANS) ;

γ) . construire l'arbre abstrait associé à la classe "classeevtr" (à l'aide du constructeur "evoperop") ; et le rajouter à l'arbre des classes d'événements/transactions "listeclasseevtr".

(v) module CONSEV

explication

Module LISP sans paramètre, saisissant les différents paramètres d'une instanciation de type d'événement et construisant une classe d'événement "événement" (à l'aide du constructeur "evop").

(vi) module CONSTRANS

explication

Module LISP sans paramètre, saisissant les différents paramètres d'une classe de transaction.

algorithme

α) . saisir les caractéristiques de la transaction ;

β) . décoder la liste des objets modifiés (CONSTLISTIDENT LISTIDENT) ;

γ) . construire le sous arbre associé à la classe de transaction "transaction" (à l'aide du constructeur "transop").

(vii) module CONSCHEMAVAR

explication

Module LISP sans paramètre permettant la saisie d'un sous-schéma variable d'une classe d'objets.

algorithme

α) . saisir le schéma variable sous forme d'un produit cartésien, décodage et analyse (CONSCPV SCHEMAVAR) ;

β) . construire l'arbre abstrait "schemavarob" correspondant au sous-schéma (à l'aide du constructeur) et le rajouter à la liste des sous-schémas "listeschemavar".

(viii) module CONSDEPFONC

explication

Module LISP sans paramètre permettant la saisie d'une dépendance fonctionnelle entre deux classes d'objets.

algorithme

α) . saisir la source de la dépendance fonctionnelle : nom sous-schéma, nom de la classe et liste d'identificateurs constituant la clé ;

β) . analyser la clé (CONSTLISTIDENT LISTECLE1) ;

γ) . saisir le but de la dépendance fonctionnelle ;

δ) . analyser la clé (CONSTLISTIDENT LISTECLE2) ;

ε) . construire l'arbre abstrait "depfonc" correspondant (à l'aide du constructeur "depop") et le rajouter à la liste des dépendances "listedepfonc" ;

C. Evaluation et améliorations possibles du logiciel

Ce logiciel souffre d'un certain nombre de manques qui le rendent relativement peu convivial, par exemple il serait intéressant de disposer d'une documentation en ligne sur la forme de description des paramètres.

En cours de réalisation (dans le cadre d'un stage de DEA)

la fonction utilisation de la spécification :

(i) un ensemble de fonctions LISP de manipulation du schéma conceptuel sont en cours de mise au point ; elles permettront de répondre aux demandes de documentation du concepteur et complèteront ainsi les affichages systématiques.

(ii) les arbres syntaxiques étant un peu ésotériques, nous rajoutons un ensemble de libellés explicatifs pouvant être afichés sous forme de lexiques à la demande du concepteur.

Améliorations nécessaires

(i) la fonction contrôle : cette fonction n'est plus du tout assurée avec la nouvelle version de CEYX ; la réaliser est notre premier objectif à court terme (dans les quelques mois à venir).

(ii) une optique vidéo : proposer une saisie des spécifications non plus par questions/réponses mais avec une optique "vidéo", analogue à ce que propose CORNELL/SYNTHESIZER [TEI 81], est notre objectif à moyen terme. Cela permettrait d'éviter une longue série d'interrogation pour saisir les différents paramètres des instanciations de type.

4.5. LISSIF : logiciel d'interprétation du SI

LISSIF a pour but de réaliser, à partir du schéma conceptuel, une implémentation du système d'information, puis de le gérer au cours du temps afin que le SI reste une image fidèle de l'organisation. Il recouvre les fonctions de :

- génération de la structure interne du SI,
- création de la version initiale du SI et documentation du SI,
- gestion de la dynamique du SI au cours du temps,

selon l'architecture suivante :

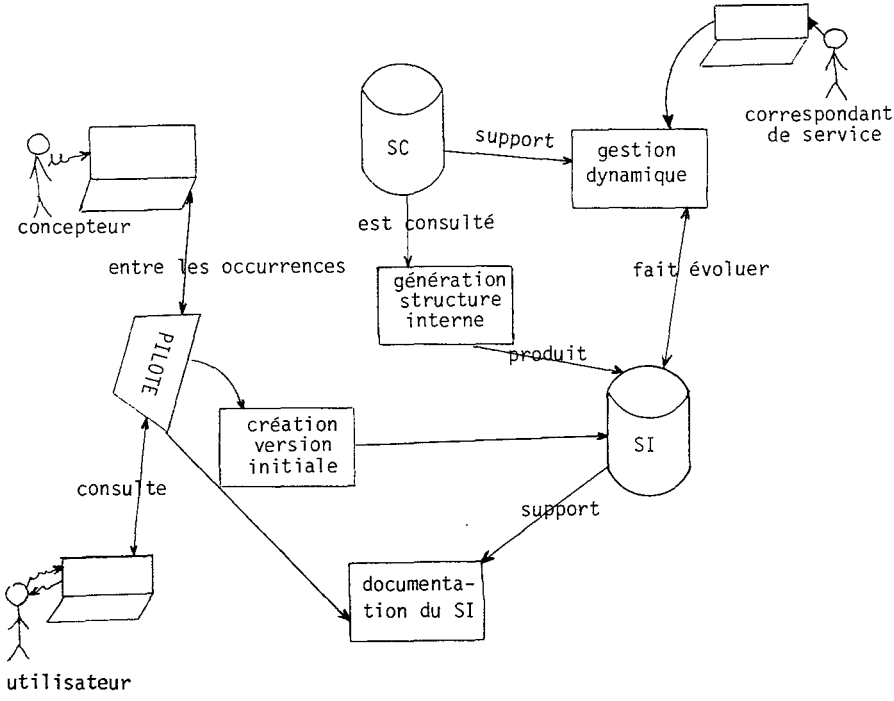


Figure 4. Architecture de LISSIF

A. Génération de la structure interne

L'outil génère, à partir de la description du schéma conceptuel stockée dans la méta-base, la description de la structure interne du SI. Ne disposant pas de structures de données riches en LISP (telles que le permet un SGBD), les instanciations des schémas de type étant des produits cartésiens de table, nous avons choisi d'implémenter le SI sous forme de tables stockées dans des fichiers.

Les règles de transformation structure conceptuelle → structure interne du SI sont en fait analogues à ce que nous proposons dans la partie III, pour la transformation Types Abstraits → modèle relationnel. On pourra y trouver le détail de l'analyse et de la justification.

Règle 1

A toute classe de c-objets correspondent n+2 tables :

. 1 table pour le schéma permanent,

. n tables pour la composante variable : 1 par sous-schéma variable,

. 1 table pour le c-objet suppression.

Règle 2

A toute classe d'événement/transaction correspondent m+6 tables :

. 1 table pour les arrivées d'événements,

. 1 table pour les changements d'état du SI,

. 1 table pour les éxécutions de transaction,

. m tables pour les modifications d'objet, 1 par objet modifié par la transaction,

. 1 table pour les déclenchements effectifs de la transaction par l'événement,

. 1 table pour les textes de prédicats associés, au cours du temps, à l'événement,

. 1 table pour les textes de transaction au cours du temps.

Actuellement le SI est construit par appel direct, par le concepteur, des procédures LISP de manipulation de tables (cf. annexe 10).

B. Création de la version initiale du SI

La création de la version initiale du SI consiste à mettre en place dans le SI :

(i) d'une part les occurrences des classes d'objets qui représentent l'état initial de l'organisation à l'instant où le système devient opérationnel, avec vérification du respect des contraintes d'intégrité. Actuellement cette saisie est réalisée par appel direct, par le concepteur, de la procédure LISP "AJOUTELEM" (cf. annexe 10).

(ii) d'autre part les versions initiales de tous les textes de prédicats, opérations, conditions et facteurs de déclenchement. Le système est actuellement en cours de réalisation : il est composé d'un ensemble de modules LISP/CEYX pilotant l'entrée des textes par un système de questions/réponses ; les fonctions en sont les suivantes :

. construction de l'arbre abstrait associé au texte et réaffichage en fin de saisie,

. pour chaque résultat principal

- appel du module de décodage des accès aux données s'il y a lieu (réalisé par des fonctions LISP de recherche en table) ;
- saisie des résultats intermédiaires et de leurs définitions formelles ; construction progressive de l'arbre des résultats ;
- après la saisie de tous les intermédiaires, réordonnancement des instructions d'après l'arbre des résultats ;
- traduction effective en LISP des instructions et création des tables intermédiaires (variables locales) ;

. strockage du résultat de l'interprétation dans un fichier.

Evaluation et améliorations

Ce module est en cours de réalisation ; l'interprète réalisé dans le cadre d'un stage MIAGE sera disponible d'ici la fin juin 1985. Il sera complété, à court terme, par un ensemble de commandes de documentation sur le SI, le but étant de proposer un système d'utilisation voisin de [FOU 82] et [KEB 84].

Une optique "vidéo" est également prévue à moyen terme. Notons que l'interprète est déjà conçu sous cette formule, puisqu'à chaque étape de description, un menu est proposé au concepteur. Celui-ci peut choisir entre plusieurs constructions, le système affichant la syntaxe concrète de la définition, le concepteur entrant les définitions formelles.

C. Le processeur de la dynamique

Le Système d'Information doit renseigner les gestionnaires sur l'état actuel, sur l'état passé et, dans certains cas, sur l'état futur de l'organisation. Pour que le SI remplisse ce rôle, il faut que l'ensemble des représentations qu'il contient soit une image toujours permanente de la réalité.

Nous proposons un outil dont le but principal est de faire évoluer le SI au cours du temps, en accord avec les événements se produisant dans l'organisation : le "processeur de la dynamique" correspond, en fait, à l'implémentation de l'opération DECLE définie dans le type SI (chapitre II, Partie I paragraphe 2.3.6-E).

Son rôle est :

- de reconnaître les changements d'état survenus dans l'organisation, en termes de type d'événement,
- d'en évaluer les conséquences, en termes de types d'opérations,
- de déclencher les actions corresponsantes,
- d'en évaluer les conséquences, en termes de changements d'état.

Ce type d'outil correspond à ce que le rapport ISO [ISO 81] appelle le "processeur d'information" qu'il définit ainsi :
<<le processeur d'informations agit pour produire des changements dans la base d'informations ou le schéma conceptuel à la réception d'un message>>.

Les recherches en cours sur les outils de ce genre sont encore à leurs balbutiements. La plupart des solutions proposées concernent des outils de documentation et de contrôle sur une méta-base ou un meta-SI qui permettent de manipuler la structure de SI [ASH 82, KNU 82, MAC 82, FLO 77, LE 84].
SYSTEM-R [AST 76, DAT 77] propose une solution de description "point d'exception/action" en introduisant le concept de "TRIGGER" qui représente, en fait, les actions à déclencher en cas d'anomalie ; mais dans les versions commercialisées, il ne semble plus y avoir trace de cette possibilité.

Récemment la méthode NIAM [VER 82] a proposé un outil d'"ENFORCER", défini comme "devant imposer toutes les règles du schéma conceptuel sur le contenu de la Base d'Information".
L'ENFORCER parait correspondre au sous-ensemble du processeur d'information que le rapport ISO nomme "le moniteur de transition" mais ne parait pas être un outil complet de gestion de la dynamique.

Plus intéressantes semblent les propositions faites dans OBE [ZLO 82] par M.M. ZLOOF. OBE, extension de QBE [ZLO 77], est un langage (et un système) orienté objet permettant la manipulation aussi bien de relations que de rapports, de modules objets ou de documents. M.M. ZLOOF reprend le concept de "TRIGGER" : il permet de définir des expressions prédicatives de modification, évaluées après un changement d'état dans le SI, et de surveillance du SI dans le temps. Le prédicat associé au TRIGGER peut être complexe.

De façon générale il s'agit de sortes de points de synchronisation organisés autour de la paire "condition (de déclenchement) / actions (résultantes)".

Les propositions d'OBE sont intéressantes, bien entendu si elles sont effectives dans le système commercialisé.

Nous présentons ici les grandes lignes de la réalisation faite par M. LE CRAS [LEC 84] en LISP/CEYX ; on se reportera à son rapport de DEA pour plus de détails. L'annexe 11 donne le programme LISP correspondant au module principal du processeur.

Nous proposons une solution relationnelle dans la partie III après avoir précisé la spécification abstraite de l'outil.

(i) architecture du processeur de la dynamique

Un module principal PROCESSEUR-DYN fait appel à une série de fonctions LISP, utilisant elles-mêmes des fonctions de travail, selon l'architecture suivante :

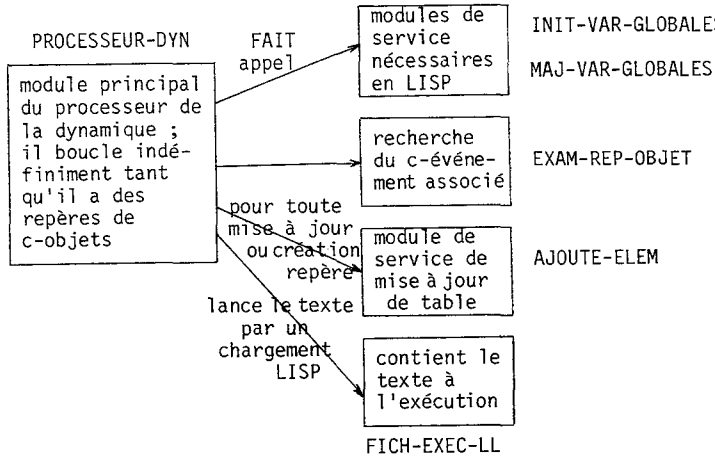


Figure 5. Architecture du processeur

(ii) algorithme (cf. arbre abstrait en annexe 8 et programme LISP en annexe 11)

α) . recherche de l'existence d'un événement ; schéma contient le schéma conceptuel du SI sous forme de l'arbre abstrait construit par LASSIF ; c-objet est le c-objet qui a changé d'état.

```
liste-even ← (liste-de-fils(troisième-fils(premier-fils(schema))))
existe ← faux

Pour chaque elem dans liste-even jusqu'à existe
faire  elem-courant ← elem
       even-opération ← deuxième-fils (elem-courant)
       si quatrième-fils (even-opération) = nom-c-objet
       alors
            si  APPL  (cinquième-fils(even-opération),  nuple-c-
                      objet)
            alors existe = vrai
            fsi
         fsi
    fpour
```

. si existe alors

β) . obtention de la table des arrivées de l'événement ;
nom-table ← deuxième-fils (even-opération)

γ) . obtention du nom de l'événement (trigger) ;
nom-even ← premier-fils (elem-courant)
. création du repère d'événement (appel de AJOUTELEM avec, en paramètre, le nuple de repère d'événement) ;

δ) . obtention du nom de la transaction à déclencher et de son texte ;
nom-transac ← sixième-fils (elem-courant)
nom-texte ← quatrième-fils (quatrième-fils (elem-courant))
. obtention de la table des exécutions de la transaction ;
nom-table ← deuxième-fils (quatrième-fils (elem-courant))

. création du repère de transaction à déclencher (appel de AJOUTELEM avec en paramètre le nuple repère de transaction) ;

ε) . exécution du texte de la transaction : changement du fichier texte et lancement de la fonction LISP correspondant au texte ; le texte rend "liste-nuple", liste des nuples de c-objets modifiés par la transaction ;

η) . création du repère de transaction exécutée (appel de AJOUTELEM avec en paramètre le nuple repère de transaction) ;

κ) . création des repères de c-objets :

```
pour chaque nuple dans liste-nuple
(AJOUTELEM 'rep-objet nuple)
fpour
fsi
```

4.6. En guise de conclusion de notre réalisation

Bien qu'un bon noyau de logiciel soit déjà opérationnel, il reste encore bien du travail surtout pour l'améliorer et le rendre plus convivial. De plus, la solution actuelle n'est pas entièrement satisfaisante en ce qui concerne les données. En effet, proposer, en Informatique de Gestion, une implémentation du SI sous forme de tables amenées en mémoire centrale lors de l'exécution du système n'est pas réaliste (même dans une optique de gestion des données par des mécanismes de mémoire virtuelle).

Il faudrait disposer d'un SGBD tel que INGRES [STO 76] (sur VAX), par exemple, pour associer la puissance de LISP à celle d'un SGBD mais alors il y aurait bien des problèmes à résoudre, en particulier pour réaliser l'interface LISP/INGRES ... le résultat serait peut être, en outre, inefficace.

Nous nous orientons, actuellement, vers une solution d'implémentation directe des Types Abstraits à l'aide de primitives du système UNIX (sur VAX) gérés sur mémoire secondaire ; ceci, afin d'être le plus près possible du noyau "système" et de ne pas accumuler les couches successives de réalisation.

5. ANALYSE DE QUELQUES SYSTEMES EXISTANTS

Nous avons déjà au chapitre I de la partie I analysé et comparé un certain nombre de méthodes de conception des systèmes d'information en Informatique de Gestion. Ici, nous nous plaçons dans une autre optique : étudier l'évolution des idées sur l'ergonomie des logiciels proposés aussi bien dans notre domaine que dans le cadre plus général du génie logiciel.

Une hypothèse est commune à tous les systèmes : on ne peut considérer la conception comme une boite noire où les concepteurs entreraient leurs définitions et l'outil en déduirait un SI construit après avoir pris toutes les décisions s'imposant. Toutes les méthodes ont évolué peu ou prou dans le sens d'une conception interactive organisée autour de la paire (automate, homme) gérée par un outil "pilote".

Les systèmes d'aide à la conception peuvent être classés en deux catégories :

(i) les systèmes traditionnels, plus ou moins dérivés des propositions de méthodes d'analyse telles que CORIG, PROTEE, MINOS ... développées durant les années 60/70.

(ii) les nouveaux systèmes", où l'on s'est attaché à la définition d'environnements de spécification particulièrement aisés à manipuler.

A. Les systèmes traditionnels

Dans ces systèmes la démarche méthodique de conception prédomine ; le but de ce genre de système est essentiellement de définir un canevas, s'appuyant sur une analyse (généralement descendante), pour découper une application en procédures puis en sous-fonctions. Notons que les méthodes associées à ce type de définition ont généralement leur origine en Informatique de Gestion, où les travaux à automatiser sont particulièrement volumineux.

Parmi ce type de méthodes, on peut distinguer deux tendances :

- les méthodes basées sur des logiciels travaillant sur des lots de données,
- les méthodes basées sur des logiciels interactifs.

Cette dichotomie résulte bien sûr de l'évolution des idées mais aussi de l'évolution du matériel ; en effet maintenant la possibilité de disposer de terminaux dans tous les services d'une organisation entraîne le désir des utilisateurs d'obtenir en temps réel des réponses à leurs besoins.

(i) les méthodes orientées "traitement par lots"

La caractéristique principale de ces méthodes est de proposer des rapports systématiques, à chaque niveau de conception, et fort peu d'interaction sauf pour l'interrogation des spécifications entrées ; la méthode restant essentiellement manuelle. Jusqu'à un niveau de décomposition quasiment de programmation, l'outil n'est là souvent que pour stocker les différentes informations, rapports et dossiers afin de simplifier leur accès et leur mise à jour. Ensuite, généralement, des outils de génération automatique, dans un langage évolué tel que COBOL ou PL1, sont proposés. CORIG [SGI], PROTEE [SIS 78], MINOS [SLIGOS], HIPO [STA 76] et SADT [ROS 77] sont de ce type.

Dans les méthodes d'analyse intégrant les idées issues des recherches sur les étapes de conception et les schémas conceptuels de données et traitements, les logiciels ont été nettement développés. Parmi les méthodes présentées dans CRIS1, 2, 3 [IFI 82-84] MERISE, par exemple, propose de stocker le schéma conceptuel dans une méta-base d'information gérée automatiquement [LE 84] ; IDA [BOD 84] propose un environnement logiciel complet de contrôle et de documentation fondé sur les outils développés dans le projet ISDOS [TEI 77]. EDM [RZE 82] propose de nombreux outils. NIAM [VER 82], SDLA [KNU 82], ISAC [LUN 82A], IML [RIC 82] commencent à développer des outils comme supports d'analyse.

(ii) des méthodes orientées "interaction"

La caractéristique principale de ces méthodes est une conception progressive et interactive des spécifications où l'outil intervient beaucoup en particulier pour guider le concepteur et le solliciter si nécessaire.

Parmi les méthodes répondant à ce critère, on peut citer CIAM [GUS 82], DAISEE [SOL 82], VEGA [CHA 83] à Nancy, REMORA où tous les systèmes d'aide à la conception étudiés ces dernières années [BAN 79, ROL 80A, FOU 82] sont interactifs, USE [WAS 82A] et SYSDOC [ASH 82] particulièrement concernés puisqu'ils visent, par ailleurs, la construction de SI interactifs.

Il faut cependant constater que ces systèmes sont généralement basés sur un processus de questions/réponses où aucun effort n'a été fait pour combiner les fonctionnalités intéressantes des outils avec les capacités et les techniques actuelles des machines ; en particulier en ce qui concerne par exemple les éditeurs vidéo ou même les gestionnaires d'écran.

B. Les nouveaux systèmes

Ils sont orientés "environnement de programmation", c'est-à-dire qu'ils proposent d'intégrer, dans un seul environnement complet, tous les outils ergonomiques spécifiques à la programmation des applications.

Démarches méthodiques et logiciels ne sont pas alors simple juxtaposition d'outils et canevas (tels que beaucoup de méthodes d'analyse en proposent) mais concourent de façon uniforme à l'aide au concepteur.

Il faut noter, cependant, que ces systèmes, issus du génie logiciel, ne s'attachent généralement pas aux mêmes problèmes que les méthodes de conception en Analyse de Gestion. Leur cadre se limite souvent à de petits énoncés dont l'algorithme de résolution est simple et dont le résultat de la spécification n'excède pas quelques écrans.

Parmi les environnements les plus connus actuellement, citons MENTOR développé à l'INRIA, CORNELL/SYNTHESIZER développé aux USA, et MAIDAY développé à Nancy sous la direction de J. Guyard [GUY 84], autour de MENTOR, pour la méthode déductive MEDEE [FIN 79].

(i) MENTOR [DON 75]

MENTOR est un système dont le but initial est de permettre la manipulation d'informations structurées (documents, arbres, programmes...).

Il est conçu pour s'appliquer spécialement à la construction d'environnement de programmation interactifs. Le noyau de MENTOR est un éditeur dirigé de syntaxe dans lequel chaque objet est représenté par un arbre. Il est associé à un langage de manipulation des arbres MENTOL. Le langage METAL permet d'écrire le meta-formalisme des définitions. Autour de ce noyau ont été développés des environnements plus complets mais généralement limités à une classe de problèmes. Il en résulte une multiplicité des commandes qui rend l'ensemble un peu complexe.

(ii) CORNELL/SYNTHESIZER [TEI 81]

CORNELL/SYNTHESIZER est un environnement interactif pour créer, éditer, exécuter et mettre au point des programmes. L'édition et l'exécution sont guidées par la structure syntaxique du langage de programmation. La grammaire du langage est une collection de "schémas" prédéfinis pour tous les types d'instructions les plus simples. Les programmes sont créés progressivement en insérant de nouvelles instructions et expressions à une position du curseur dans le squelette des schémas précédemment entrés. Le curseur peut se déplacer dans les schémas qui sont générés par des commandes. Des aides au diagnostic sont proposées lors de l'entrée des spécifications et le code est généré, donc exécutable, à chaque entrée d'un schéma.

Un tableau de clés, un écran vidéo et un gestionnaire d'écran sont les composants de base de CORNELL/SYNTHESIZER ; ils en permettent une utilisation aisée et un minimum de risques d'erreurs.

(iii) EDME [JAC 84] (projet MAIDAY [GUY 84])

Depuis 1974 s'est développée, à Nancy, une recherche active autour d'une méthodologie déductive de conception de programme : MEDEE [FIN 79]. Différentes propositions ont été faites sur différents thèmes liés au développement de MEDEE [HUC 78, BEL 78, SOU 82].

L'objectif principal du projet MAIDAY, sous la direction de J. GUYARD [GUY 84], est de construire un environnement de programmation fondé sur la méthode déductive. J.P. JACQUOT [JAC 84] propose un éditeur d'algorithme EDME (EDiteur pour MEdee), coeur de la maquette de l'environnement qui guide, aide, conseille le programmeur ; le dialogue réalisant la mise en oeuvre de la méthodologie. Pour faciliter l'utilisation d'EDME, la gestion d'écran est particulièrement aisée et ergonomique.

La plupart des environnements de conception proposés sont issus du génie logiciel ([MED 82], [EST 83], [AND 84], [WER 83], [MIN 84]). Notons que les propositions dans ce domaine, en matière d'outils, sont très nettement plus avancées qu'en Informatique de Gestion où, hormis MAPPER [HER 84], l'intérêt des ateliers intégrés de logiciels n'est pas encore vraiment perçu.

Les propositions, que nous avons faites dans ce qui précède dans LASSIF et LISSIF montrent, nous l'espérons, que la réalisation de logiciels intégrés est une de nos préoccupations prédominantes.

6. CONCLUSION

Le logiciel d'aide à la construction de la spécification est un complément indispensable à la modélisation d'un SI à l'aide du langage LASSIF. En effet il n'est plus possible, dans un environnement qui se veut efficace, de proposer une méthode essentiellement manuelle.

Les points forts du logiciel

Le logiciel LASSIF constitue un ensemble intégré de modules développés dans l'environnement complet LISP/CEYX, permettant actuellement :

- la saisie des aspects structurels de la réalité, c'est-à-dire des instanciations des schémas de types formels de SI. Cette saisie est réalisée avec peu d'effort de syntaxe de la part du concepteur puisqu'elle est conversationnelle. Le système affiche en fin de session les spécifications entrées suivant le formalisme du langage LASSIF. Il construit et stocke dans un fichier l'arbre abstrait correspondant au SC du SI ; arbre qui sera réutilisé par le processeur de la dynamique et par le logiciel de documentation ;

- la saisie des textes de contraintes d'intégrité, de prédicats et de transactions. La description fournie par le concepteur est minimale puisqu'il dispose de menu de choix de constructions et également, sans effort de syntaxe, puisque le système affiche automatiquement la syntaxe concrète du langage d'énoncé ; le système réordonne et interprète les instructions ;

- la gestion de l'évolution du SI par le processeur de la dynamique.

Ses manques

Nous les avons signalés dans notre développement :

- des contrôles très incomplets,

- un défaut de documentation en ligne sur la syntaxe et le type des descriptions,

- une documentation à la demande incomplète sur le SC et le SI (partiellement en cours de réalisation),
- des dialogues trop pauvres et une saisie contraignante sans éditeur vidéo.

Ces quatre points constituent des manques, bien sûr, mais il est, en fait, relativement aisé d'y remédier à court ou moyen terme.
Même complètement achevé, cet outil souffrirait de grandes limites comme les logiciels, proposés par ailleurs par d'autres systèmes [WAS 82A], [BOD 84], [LE 84], ...

En effet, même si le système pilote la conception, il a pour défaut principal de ne proposer que des aides statiques au concepteur c'est-à-dire qu'il suit un canevas prédéfini qu'il n'est pas capable de modifier ni même de rendre paramétrable.

L'orientation actuelle, en génie logiciel, qui commence à être effective en programmation, est de concevoir des outils de CAO qui, possédant une certaine connaissance sur la démarche d'analyse (les règles de raisonnement) d'une part, sur les spécifications déjà entrées d'autre part, sont capables d'aider le concepteur à prendre des décisions de façon dynamique, le logiciel tenant compte de ce qu'il sait déjà et améliorant sa connaissance par des déductions. J.P. JACQUOT [JAC 84] a fait un certain nombre de propositions pour une aide effective à la construction de programmes basée sur MEDEE. Lors du congrès "7th Conference Software Engineering" (Orlando Mars 1984), d'autres propositions ont été faites dans ce sens, en particulier le système PROUST [JOH 84] qui aide les programmeurs novices à construire leur algorithme.

Cette nouvelle orientation, issue du génie logiciel et de l'intelligence artificielle, nous parait réaliste également pour la conception des SI en Informatique de Gestion.

PARTIE III

Une solution relationnelle

pour le processeur de la dynamique

1. INTRODUCTION

Le but de cette partie est de présenter, en détail, une réalisation d'un sous-ensemble du logiciel d'interprétation LISSIF : le processeur de la dynamique, auquel nous avons appliqué notre propre démarche de conception de logiciel.

Dans la partie II, nous avons explicité les grandes lignes de cet outil et développé la solution réalisée en LISP/CEYX par M. LE CRAS dans le cadre de son stage de DEA.

Cette partie :

(i) . précise la nature de l'outil en définissant sa spécification abstraite. Cette spécification est réalisée à l'aide du langage LASSIF, ce qui nous a permis d'autovalider la puissance d'expression du langage,

(ii) . présente une variante de réalisation construite autour du SGBD relationnel SYNTEX [DEM 75]. Après avoir défini la fonction de représentation Types Abstraits → modèle relationnel, nous expliciterons notre solution relationnelle.

2. LA SPECIFICATION ABSTRAITE DU PROCESSEUR DE LA DYNAMIQUE

2.1. Son rôle

Il a pour rôle d'assurer automatiquement la gestion de l'évolution du SI. Il doit prendre en compte tous les événements qui se produisent dans la vie de l'organisation et réaliser et contrôler l'exécution dans le SI des transformations résultantes.
Le résultat de l'application du processeur de la dynamique à un SI est en fait la création d'un nouvel état du Système d'Information. Ce nouvel état, à l'instant t, est obtenu par l'exécution d'un texte d'opération sur l'ancien état du SI, à l'instant t-1.

$$\text{état}_{si_{t-1}} \times \text{texte} \xrightarrow{\text{produit}} \text{état}_{si_t}$$

Un état de SI est défini par l'état des objets qui le composent à un instant donné.

2.2. Sa spécification abstraite

Le déclenchement des actions à entreprendre à la suite d'un événement est exprimé par l'opération DECLE, définie dans le type SI (cf. Chapitre II partie I) de la façon suivante :

DECLE : fonct (SI, IDENT, DATE, CLAS) SI

Soient si : SI, cob : IDENT, d : DATE, nuple $\in$ CLAS

$$\text{DECLE}(si, cob, d, nuple) = \bigcup_{i,j,k} \text{APPL}(\text{TEXTOP}(cop_{ij}, d), \text{APPL}(FAC_k (cev_i, x_j, d), nuple))$$

où

. I : $cev_i \in$ cev de si t.q. $\text{CONST}(cev_i) = \text{APP}(\text{TNOMTYPE-SCHEMA}, cob)$ et $\text{APPL}(\text{PRED}(cev_i, d), nuple)$

. $\forall i \in I, J$: $\{cop_{ij} \in OPERA(cev_i)$ t.q. $APPL(COND(cev_i, x_j, d), nuple)\}$

. $\forall i \in I, J \in J, K$: $\{FAC(cev_i, x_j, d)\}$

Cette opération est à lancer à chaque nouveau changement d'état de c-objet dans le SI.

Le processeur de la dynamique est alors un outil récursif définissant, à chaque appel, un nouvel état de SI à une date donnée. La définition récursive de l'outil est la suivante :

Soient

. si_{in} : SI un Système d'Information initial valide à une date d : DATE donnée.

. obext : $\mathcal{P}$(CLAS) ensemble des objets munis de leur date de création, liés à des événements d'origine externe (par exemple l'arrivée d'une commande), la définition récursive de l'outil est la suivante :

proc* (si_{in}, d, obext) = proc* (PROCESSEUR(si_{in}, d, obext))

où . PROCESSEUR (si_{in}, d, obext) est un module qui rend un nouvel état de SI et une nouvelle date.

A. Module principal

(Nouvelétatsi, datenouv) PROCESSEUR (si_{in}, d, obext)

Nouvelétatsi ? représente un élément de la suite des états d'un SI au cours du temps

si_{in} ? représente l'état initial du SI

Nouvelétatsi, si_{in} : SI

d ? date où est lancé le processeur

d : DATE

datenouv = Choix-d'un-changement-d'état(si_{in}, d, obext)

Nouvelétatsi = Traitement-événement-en-attente(si_{in}, d)

datenouv ? nouvelle date d'appel du processeur

datenouv : DATE

B. Choix-d'un-changement-d'état

Ce module explore l'ensemble des objets ayant changé d'état depuis la date de lancement du processeur de la dynamique (d), choisit le plus ancien, et évalue si ce changement d'état est un événement pour le SI. Il est composé de trois appels de procédure :

. PRENDRE-OB (si_{in}, d, obext) qui rend, s'il existe, ob, l'objet choisi et la nouvelle date d'échéancier,

. RECH-EV (si_{in}, ob) qui rend ev l'événement associé, s'il existe,

. CRE-EV (ev, si_{in}, ob) qui crée l'événement dans le SI.

Sa spécification est :

datenouv Choix-d'un-changement-d'état (si_{in}, d, obext)

datenouv = PRENDRE-OB (si_{in}, d, obext)
si ob $\neq \omega$ et RECH-EV (si_{in}, ob)
alors CRE-EV (ev, si_{in}, ob)
sinon choix -d'un-changement-d'état (si_{in}, datenouv, obext)

(i) Module prendre-objet

Ce module recherche dans le SI le premier objet, s'il existe, qui a changé d'état depuis le lancement du processeur.
Il s'agit d'un objet qui, soit appartient au SI initial (si_{in}) ou a été produit par une action externe à l'organisation, il appartient alors à obext.
Rappelons la forme type d'un objet dans le SI :
Soient . nomob l'identificateur du c-objet,
. nupleob son nuple de valeurs, composé d'un tuple de valeurs "val" et d'une date d'entrée dans le SI "datecre".

(ii) La spécification du module est :

(datenouv, ob) PRENDRE-OB (si_{in}, d, obext)

ob ? représente l'objet choisi parmi les objets en attente

ob : (CLAS de si_{in}) U obext U ω

ob = (nomob de cob_i, nupleob de cob_i) /

$\exists$ i : ENTIER t.q.
$cob_i \in$ (CLAS de si_{in}) U obext
et
datecre de nupleob de $cob_i \geqslant d$
et $\forall j \neq i$
datecre de nupleob de $cob_j \geqslant$ datecre de nupleob de cob_i

datenouv = si ob $\neq \omega$ alors datecre de nupleob de cob_i
sinon DATESYST

CLAS : produit cartésien des classes d'objets du SI

(iii) Module recherche de l'événement associé

Ce module recherche si le changement d'état de l'objet ob choisi à l'étape précédente est bien un événement pour le SI.
Son résultat est le type d'événement s'il existe. Soit nomevt l'identificateur de l'événement trouvé, il appartient à la table des identificateurs d'événements du SI : TNOMTYPE-EV.

Sa spécification est :

ev RECH-EV (si_{in}, ob)
ev ? représente le type d'événement trouvé
ev : IDENT U ω
ev = (nomevt de cev_i) / $\exists$ $cev_i \in$ CODOM (TNOMTYPE-EV) t.q.
CONST (cev_i) = nomob de ob et
APPL (PRED(cev_i, datecre de nupleob de ob), nupleob de ob)
TNOMTYPE-EV ? table des identificateurs d'événements du SI.

(iv) Module création d'un événement

Ce module ajoute une occurrence à la classe des événements donnée.

Sa spécification est :

nouvelensevt CRE-EV (ev, sin_{in}, ob)
nouvelensevt ? nouvel ensemble d'événements
nouvelensevt : CEV de si_{in}
nouvelensevt = ADJEV (ev, idev, datev, nupleob de ob)

idev ? identifiant de l'événement
idev : ENTIER
idev = num+1
num ? numéro du dernier événement de type ev entré dans le SI
num : ENTIER
num = MAX (DOM (datearriv de ev))
datearriv ? champ du produit cartésien ev correspondant à la table des arrivées des événements
datev ? date d'arrivée de l'occurrence d'événement
datev : DATE
datev : DATESYST

C. Traitement d'un événement en attente

Ce module explore les événements du SI en attente d'être activés ; il choisit le plus ancien, en évalue les conséquences en termes d'opérations qu'il déclenche effectivement.

Il est composé de trois appels de procédure :

. PRENDRE-EV (si_{in}, d) qui rend l'événement choisi cev s'il existe, l'occurrence d'objet ob associé ; et la nouvelle date d'échéancier ;

. RECH-OP (cev, si_{in} ob) qui rend un ensemble d'opérations à déclencher "opération" ;

. DEC-OP (opération, d, ob) qui rend un nouvel état de SI.

Sa spécification est :

nouvelétatsi Traitement-événement-en-attente (si_{in}, d)
nouvelétatsi = PRENDRE-EV (si_{in}, d)
si cev $\neq \omega$ et RECH-OP (cev, si_{in}, ob)
alors DEC-OP (opération, d, ob)
sinon Traitement-événement-en-attente (si_{in}, datenouv)

(i) Module prendre-événement

Ce module recherche, dans l'ensemble des événements en attente dans le SI, le premier s'étant produit après le lancement du processeur.

Sa spécification est :

(datenouv, cev, ob) PRENDRE-EV (si_{in}, d)

cev ? événement choisi

cev : (CEV de si_{in}) U ω

ob ? objet associé

ob : (CLAS de sin_{in}) U ω

(cev, ob) = (nomev de cev_i, datearriv de cev_i, ob de cev_i) /

$\exists$ i : ENTIER t.q.

$cev_i \in$ CEV de si

et datearriv de $cev_i \geqslant$ d

et $\forall$ j $\neq$ i

datearriv de $cev_j \geqslant$ datearriv de cev_i

datenouv = si cev $\neq \omega$ alors datearriv de cev_i

sinon DATESYST

(ii) Module recherche-opération

Ce module cherche, lorsqu'un événement a été pris en compte, quels sont les types d'opérations à déclencher (ceux dont la condition de déclenchement est vérifiée). Son résultat est une suite de types d'opérations, ses paramètres sont l'événement choisi à l'étape précédente et l'occurrence d'objet associée.

Sa spécification est :

opération RECH-OP (cev, sin_{in}, ob)

opération ? représente l'ensemble des types d'opérations à déclencher à la suite de l'événement cev

opération : SUITE (op_{ij})

op_{ij} ? opération à déclencher

op_{ij} : IDENT U ω

opération = {(nomop de copération$_{jj}$) /
∃ copération$_{jj}$ ∈ CODOM (TNOMTYPE-OP) t.q.
copération$_{jj}$ ∈ OPERA (nomev de cev)
et
APPL {COND (nomev de cev, xj de cev, datearriv de cev), ob)}
TNOMTYPE-OP ? table des identificateurs d'opérations
xj ? champ du produit cartésien déclenche l'événement

(iii) Module de déclenchement effectif

Ce module réalise, de façon récursive, l'application du texte d'une opération à un état de SI. Il a en paramètre la liste des types d'opérations à déclencher, le nuple ob qui a changé d'état (associé à l'événement choisi) et la date de déclenchement et, pour résultat, un nouvel état du SI.

Sa spécification est :

nouveletatsi DEC-OP (opération, d, ob)
nouveletatsi = APPL (TEXTOP (nomop de PREM (operation), d), ob)
DEC-OP (RESTE (opération), d, ob)

3. MECANISME DE MISE EN OEUVRE DU PROCESSEUR DE LA DYNAMQIUE

3.1. Architecture de l'outil

Pour mettre en oeuvre les actions précédentes, le processeur de la dynamique s'appuie :

- sur le schéma conceptuel du SI qui fournit la description de la logique de fonctionnement du SI. Cette structure conceptuelle est stockée dans une méta-base, support également de la fonction documentation du logiciel d'aide à la spécification,

- sur le SI lui-même qu'il met à jour et dont il analyse l'état, pour fonctionner,

- sur des informations qui lui sont propres "les repères" qui permettent de conserver

  . le sous-ensemble des objets du SI qui ont changé d'état depuis la date d,

  . le sous-ensemble des événements du SI en attente,

  . le sous-ensemble des opérations du SI en attente d'être exécuté.

Ces repères sont des objets techniques, des files d'attente qui caractérisent l'état de traitement des éléments du SI dont le processeur de la dynamique assure la gestion.
L'architecture de l'outil peut être schématisée ainsi :

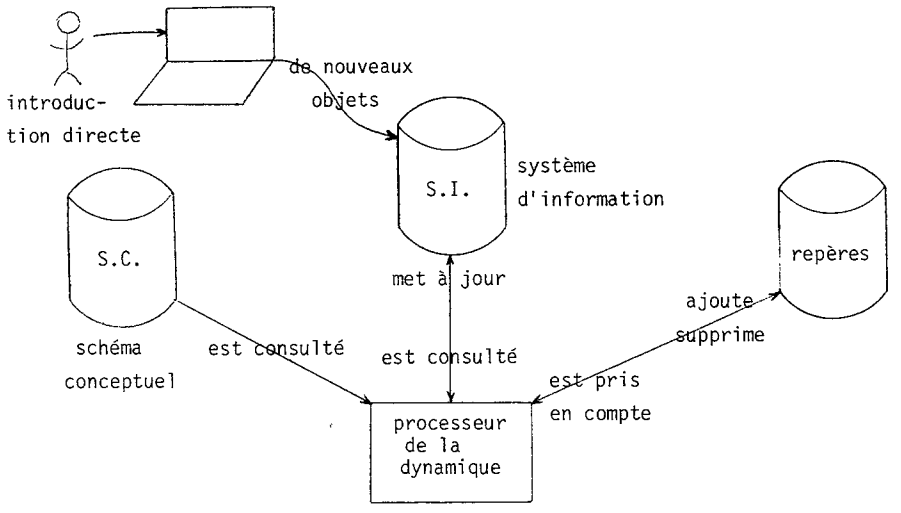


Figure 1. Architecture du processeur
de la dynamique

3.2. Définition abstraite des informations manipulées

A. Structure abstraite du schéma conceptuel et du système d'informations

Rappelons qu'au chapitre II de la partie I nous avons introduit le schéma de type abstrait SI dont une instanciation nous permet de définir un système d'informations particulier sous forme d'un produit cartésien :

. de classes d'objets

. de classes d'événements

. de classes d'opérations

. d'un ensemble de tables permettant d'exprimer le schéma conceptuel du Système d'Information.

Type SI (CLAS, COP, CEV, predSI$_1$, predSI$_k$) =
<cob:CLAS, cop:COP, cev:CEV, t1:TNOMTYPE-OB, t2:TNOMTYPE-SCHEMA, t3:TNOMTYPE-OP, t4:TNOMTYPE-EV, t5:T-CONST, t6:T-MOD, t7:T-OPERA>
où predSI$_1$, ..., predSI$_k$ sont des prédicats exprimant des contraintes d'intégrité entre les éléments du SI.

B. Structure abstraite des repères

Nous les définissons comme des instanciations du schéma paramétré de type FILE (cf. chapitre II partie I) dont le paramètre effectif est un produit cartésien.

(i) repère de c-objet
type REP-OBJET (nssche, nclasseob, nupleob) =
FILE (<nssche : IDENT, nclasseob : IDENT, nupleob : CLAS de SI)
avec la restriction suivante :
nssche ∈ APP (TNOMTYPE-SCHEMA, nclasseob)
où . nssche est le nom du cobjet qui a changé d'état, c'est un sous-schéma de la classe nclasseob,
. nupleob est le nuple de c-objet ayant changé d'état :
nupleob : <clé : CLE, val : PRODCAR(T1,...,Tn), date : DATE>
où clé est la clé du nuple, date, sa date de parution dans le SI, val, les valeurs de ses constituants.

(ii) repère d'événement
type REP-EVEN (evi, idev, datevi, rob) =
FILE (<evi : IDENT, idev : NCEV, datevi : DATE, rob : REP-OBJET>)
où . evi est le nom de l'événement
. idev la valeur de son identifiant
. datevi sa date d'arrivée dans le système
. rob le repère de c-objet associé (représente le nuple d'objet ayant changé d'état).

(iii) repère d'opération à déclencher

type REP-OP (nomop, idop, datedecl, nuplev) =

FILE (<nomop:IDENT, idop:NCOP, datedecl:DATE, nuplev:REP-EVEN>)

où

. nomop est le nom du type d'opération à déclencher

. idop est la valeur de son identifiant

. datedecl sa date de déclenchement

. nuplev le nuple d'événement correspondant dans la file des événements.

(iv) repère d'opération exécutée

type REP-OPEX (nupleop, datexec, nupleobjet) =

FILE (<nupleop:REP-OP, datexec:DATE, nupleobjet:REP-OBJET>)

où

. nupleop est le nuple correspondant dans la file des opérations à déclencher

. datexec est la date d'exécution de l'opération

. nupleobjet est l'image d'un repère de c-objet correspondant à l'objet modifié par l'opération.

4. UNE SOLUTION RELATIONNELLE

4.1. Justification du choix d'un SGBD relationnel

Trois raisons majeures nous ont poussé à ce choix :

(i) Dans le domaine de l'Informatique de Gestion, le poids accordé aux structures de données l'emporte très nettement sur l'importance des traitements. Il faut disposer de moyens d'accès aux informations très efficaces. Un Système de Gestion de Bases de Données s'impose donc.

(ii) La formalisation initiale des concepts du modèle REMORA [FOU 82], préconisée au moment où nous avons réalisé cet outil, était fondée sur le modèle relationnel. Il était donc intéressant, pour minimiser la représentation de la structure conceptuelle en une structure physique, de choisir un SGBD relationnel.

(iii) Les SGBD relationnels proposent, le plus souvent, un langage de requête puissant et aisé à utiliser. La manipulation des données s'effectue sans connaître leur structure physique, ni même logique.

Nous avons donc choisi de représenter la structure conceptuelle du SI comme une collection de relations, exprimées dans le langage de description de données du SGBD relationnel choisi.

Bien que cette analyse ait été faite, en réalité, a postériori, nous allons étudier comment représenter les schémas de Types Abstraits permettant la spécification de SI par une collection de relations.

4.2. Fonction de représentation TA → relations

Le but de ce paragraphe est d'établir la "fonction de représentation", associant aux objets construits à l'aide de types paramétrés (chapitre II partie I) des objets "relations" [COD 70].

Définition mathématique d'une relation

Si on désigne par D1, D2, ..., Dn une suite ordonnée de n domaines non forcément distincts, une relation R n-aire, sur cette suite, est une partie du produit cartésien D1 x D2 x ... x Dn.

Une relation s'exprime à partir de deux constructeurs introduits au chapitre II de la partie I (paragraphe 2.2) :

- ensemble des parties noté $\mathcal{P}$,
- produit cartésien PRODCAR(D1,...,Dn) schématisé par <x1 : D1, ..., xn : Dn>.

Une relation est une instanciation du schéma de type $\mathcal{P}$(E) où E est de la forme <x1 : D1, ..., xn : Dn>.

Pour faire une analyse systématique de la fonction de représentation TA → relations, il faut analyser chaque terme de l'univers à représenter, c'est-à-dire dans notre cas :

- $\mathcal{P}$(E)
- TABLE(CLE,A)
- SUITE(E)
- PRODCAR(T1,...,Tn)

où les paramètres effectifs des schémas sont de type quelconque.
En fait, considérant que dans la réalité seul un sous-ensemble de ces cas se produit, nous limitons notre analyse aux termes les plus évidents.

Nous définissons les représentations TA → relation cas par cas, puis nous en déduirons la fonction globale de représentation, que nous appliquerons enfin sur les schémas CLASSE-OB, CEVENT, COPERATION et SI.

4.2.1. Etude de la représentation des termes simples

Les termes simples sont des instanciations de schémas $\mathcal{P}$(E), TABLE (CLE,A), SUITE(E), PRODCAR(T1,...,Tn), où les paramètres effectifs sont l'un des types de base introduits : ENTIER, BOOLEEN, CHAINE ou DATE.

(i) étude de $\mathcal{P}(E)$ où E est un type élémentaire

Représentation 1 : $\mathcal{C}1 : \mathcal{P}(E) \rightarrow \mathcal{R}(x1 : E)$

Si E, paramètre du type $\mathcal{P}(E)$, est un type élémentaire alors on définit la représentation $\mathcal{C}1$ par :

$$\forall p : \mathcal{P}(E),\ \mathcal{C}1(p) = \{e \ / \ e \in p\}$$

Le terme de représentation est une relation unaire, notée $\mathcal{R}(x1 : E)$; x1 est le nom du constituant de domaine E.

(ii) étude de TABLE(CLE,A) où A est élémentaire

Par définition le paramètre CLE est simple : il s'agit soit de valeurs, soit de nuples de valeurs, constituant les clés d'entrée dans la table. Si A est élémentaire, un objet de type TABLE(CLE,A) est un ensemble de couples (clé, valeur) où clé $\in$ CLE et valeur $\in$ A.

Représentation 2 : $\mathcal{C}2$: TABLE(CLE,A) $\rightarrow$ $\mathcal{R}$(x1 : CLE, x2 : A)

Si A, paramètre du type TABLE(CLE,A), est un type élémentaire alors on définit $\mathcal{C}2$ par :

$$\forall t : \text{TABLE(CLE,A)},\ \mathcal{C}2(t) = \{(c,v) \ / \ c : \text{CLE},\ v : A,\ \underline{\text{et}}\ v = \text{APP}(t,c)\}$$

Le terme de représentation est une relation, notée $\mathcal{R}$(x1 : CLE, x2 : A), binaire, où x1 et x2 sont les constituants, respectivement de domaines CLE et A. Cette relation est en première forme normale puisque x2 dépend fonctionnellement de x1.

Généralisons $\mathcal{C}2$:

Soit CLE un nuple de n constituants :

CLE = <x1 : T1, ..., xn : Tn>

Soit A un nuple de m constituants :

A = <y1 : T'1, ..., ym : T'm>

alors $\forall t$: TABLE(CLE,A)

$\mathcal{C}2(t) = \{((c1,\ldots,cn), (v1,\ldots,vm)) \ / \ c1 : T1,\ldots, cn : Tn$ et v1 : T'1,..., vm : T'm et (v1,...,vm) = APP (t,(c1,...,cn))$\}$.

Le terme de représentation est une relation en première forme normale notée $\mathcal{R}$(x1 : T1,..., xn : Tn, y1 : T'1,..., ym : T'm) de degré m + n.

(iii) étude de SUITE(E) où E est élémentaire

Si E est un type élémentaire, un objet du type SUITE(E) est un ensemble de couples du type <i : ENTIER, x : E>
où i représente l'indice de l'élément x de la suite E.

Représentation 3 : $\mathcal{C}3$: SUITE(E) → $\mathcal{R}$(x1 : I, x2 : E)

Si E, paramètre du type SUITE(E), est un type élémentaire, on définit $\mathcal{C}3$ par :

$\forall$ s : SUITE (E), $\mathcal{C}3$ (s) = {(i,x) / i : I, x : E }

Le terme de représentation est une relation binaire notée $\mathcal{R}$(x1 : I, x2 : E)

Cette relation est en première forme normale puisqu'à chaque valeur de I correspond une valeur de E, le constituant x1 de domaine I joue le rôle d'une clé.

(iv) étude de PRODCAR(T1,...,Tn) où Ti est élémentaire $\forall i \in [1...N]$

Un objet de ce type s'exprime sous la forme <x1 : T1,..., xn : Tn>, les Ti étant des types de base.

Représentation 4 : $\mathcal{C}4$: PRODCAR(T1,...,Tn) → <x1 : T1,..., xn : Tn>

Si T1,...,Tn, paramètres du type PRODCAR(T1,...,Tn), sont des types élémentaires, on définit $\mathcal{C}4$ par :

$\forall$ prod : PRODCAR(T1,...,Tn), $\mathcal{C}4$ (prod) = (t1,t2,...,tn) / t1 : T1, t2 : T2, tn : Tn

Le terme de représentation est un nuple de relation de degré n noté <x1 : T1,..., xn : Tn>. Notons que nous avons introduit la relation comme : $\mathcal{P}$(<x1 : T1,..., xn : Tn>).

4.2.2. Etude de la représentation des termes complexes

Les termes complexes sont des instanciations des schémas précédents dont un des paramètres effectifs est une relation. L'étude porte sur :

- TABLE(CLE,R)
- SUITE(R)
- PRODCAR(T1,...,R,...,Tn)

où R est une relation obtenue à l'étape 1, de degré n et en première forme normale.

(i) étude de TABLE(CLE,R)

Pour tout objet de ce type, à chaque entrée de la clé correspond un ensemble de nuplets qui constitue, par hypothèse, une relation en 1FN.

Représentation 5 : $\mathcal{C}5$: TABLE(CLE,R) → R_1(x1 : CLE, x2 : R)

Si R est paramètre effectif de type relation de degré n, du type TABLE (CLE,R) alors

$\forall$ t : TABLE(CLE,R), $\mathcal{C}5(t)$ = {(,r1,...,rn) / c : CLE, (r1,...,rn) : R et (r1,...,rn) = APP(t,c)}

Le terme de représentation est une solution R_1, notée R_1(x1 : CLE, x2 : R) de degré n + 1. En généralisant, si CLE est composée de m constituants, R_1 est de degré m + n.

La relation obtenue est en 1FN si on adjoint le constituant CLE à la clé de la relation.

(ii) étude de SUITE(R)

Pour tout objet de ce type, à chaque indice i du couple <i : ENTIER, r : R> correspond un ensemble de nuplets qui constituent par hypothèse, une relation en 1FN.

Représentation 6 : $\mathcal{C}6$: SUITE(R) → R_1 (x1 : I, x2 : R)

Si R est paramètre de type relation de degré n du type SUITE(R) alors

$\forall$ s : SUITE(R), $\mathcal{C}6(s)$ = {(i,r1,...,rn) / i : I, (r1,...,rn) : R}

Le terme de représentation est une relation R_1, notée R_1(x1 : I, x2 : R) de degré n + 1. La relation obtenue est en 1FN si on adjoint le constituant I à la clé de R.

(iii) étude de PRODCAR(T1,...,R,...,Tn)

où R est une relation de degré m en 1FN ; <t1 : T1,...t_{i-1}, : T_{i-1}, t_{i+1} : T_{i+1}, ..., tn : Tn> un tuple de n-1 valeurs.

Représentation 7 : $\mathcal{C}7$: PRODCAR(T1,...,R,...,Tn) → R_1(x1 : T1,..., xi : R,..., xn : Tn)

Si R, de relation de degré m, est paramètre du type PRODCAR(T1,...,R, ...,Tn) alors

prod : PRODCAR(T1,...,R,...,Tn)

$\mathcal{C}7$(prod) = {t1,...,t_{i-1},r1,...,rm,t_{i+1}...,tn) / t1 : T1,..., t_{i-1} : T_{i-1}, (r1,...,rm) : R, t_{i+1} : T_{i+1},..., tn : Tn}

Le terme de représentation est une relation de degré m + n - 1 notée R_1(x1 : T1,...,x_{i-1} : T_{i-1}, xi : R,..., xn : Tn)

Pour rendre cette relation en 1FN il faut adjoindre l'ensemble des constituants de domaine T_1,..., T_{i-1}, T_{i+1}, T_n à la clé de R.

Cette structure est en fait hiérarchique ; l'applatir sous forme d'une seule relation induit une grande redondance avec tous les inconvénients que cela entraîne. La normalisation conduit, en fait, à une décomposition car un des principes de base du modèle relationnel est d'interdire des constituants de relation à valeurs multiples (en l'occurrence en un constituant R de type relation). Notons que les nouveaux développements du modèle relationnel permettent la représentation de structures hiérarchiques [SCHE 84] : en effet la non première forme normale NF^2 permet l'expression d'attributs multivalués, se redécomposant eux mêmes en attributs multivalués ..., sous forme d'arborescences. Cette expresssion est associée à une algèbre relationnelle particulière.

4.2.3 Définition de la fonction globale de représentation

type = UNION($\mathcal{P}$(E), TABLE(CLE,A), SUITE(E), PRODCAR(T1,...,Tn),
TABLE(CLE,R), SUITE(R), PRODCAR(T1,...,R,...,Tn))

où les paramètres effectifs des schémas sont du type de ceux étudiés ci-dessus : $\forall$ x : type

$\mathcal{C}$(x) =

si estype$\mathcal{P}$(E) (x) alors $\mathcal{C}$1 (x)
sinon si estypeTABLE(CLE,A) (x) alors $\mathcal{C}$2 (x)
sinon si estypeSUITE(E) (x) alors $\mathcal{C}$3 (x)
sinon si estypePRODCAR(T1,...,Tn) (x) alors $\mathcal{C}$4 (x)
sinon si estypeTABLE(CLE,R) (x) alors $\mathcal{C}$5 (x)
sinon si estypeSUITE(R) (x) alors $\mathcal{C}$6 (x)
sinon si estypePRODCAR(T1,...,R,...,Tn) (x) alors $\mathcal{C}$7 (x)
fsi

4.2.4. Application de la fonction de représentation globale aux schémas de specification du SI

L'expression de la fonction de représentation est immédiate puisque les schémas permettant l'expression du schéma conceptuel de Système d'Information sont des combinaisons plus ou moins complexes des constructeurs précédents.

Nous introduisons directement la décomposition en relations des termes de représentation. Cette solution permet une expression plus simple. Nous nous limiterons à l'étude en détail du type CLASSE-OB, en particulier en ce qui concerne la représentation des opérations et l'expression des dépendances fonctionnelles entre classes d'objets. Nous donnons ensuite directement les résultats pour les autres types, l'analyse étant similaire. Rappelons qu'un objet de type CLASSE-OB est un produit cartésien d'un SCHEMAP, d'un SCHEMAV et d'une TABLE.

(i) étude du type SCHEMAP

type SCHEMAP(CLE,COMP) = TABLE(CLE,COMP) où COMP est un PRODCAR c'est-à-dire un nuple de relation. C'est un cas particulier de l'étude de TABLE(CLE,R).

Représentation 8 : $\mathcal{C}$: SCHEMAP(CLE,COMP) → R(x1 : CLE, x2 : COMP)

∀ schp : SCHEMAP(CLE,COMP)

$\mathcal{C}$(schp) = {((c1,...,cm), (r1,r2,...,rn)) / (c1,...,cm) : CLE
(r1,r2,...,rn) : COMP et
(r1,r2,...,rn) = APP(schp, (c1,...,cm))}

Le terme de représentation est une relation de degré m + n, notée R(x1 : CLE, x2 : COMP),
m étant le nombre de constituants de CLE, n le nombre de constituants de COMP.

(ii) étude du type SCHEMAV

type SCHEMAV(CLE,COMPV) = TABLE(CLE,COMP) où COMPV est un PRODCAR de tables de la forme TABLE(DATE,$CARV_i$), elles mêmes paramètrées par k $CARV_i$ de type PRODCAR. Il s'agit d'un type hiérarchique que nous décomposons directement.

Représentation 9 :

$\mathcal{C}$: SCHEMAV(CLE,COMPV) → { . R_1 (x_{11}:CLE, x_{12}:DATE, x_{13}:$CARV_1$)
. R_2 (x_{21}:CLE, x_{22}:DATE, x_{23}:$CARV_2$)
....
. R_k (x_{k1}:CLE, x_{k2}:DATE, x_{k3}:$CARV_k$)

Le terme de représentation est en fait k relations (autant que de $CARV_i$), de clé composée des constituants de CLE et d'une date.

(iii) étude du type classe-OB

α) fonction de représentation du type

type CLASSE-OB(CLE,compp,COMP,compv,COMPV,datesup,predc,...,preg) =
<compp:SCHEMAP(CLE,COMP), compv:SCHEMAV(CLE,COMPV), datesup:TABLE(CLE, DATE)>

Représentation 10 :

$\mathcal{C}$: CLASSE-OB(CLE,...,preg) → { . R_p (x_1:CLE, x_2:COMP) ; . R_1 (x_{11}:CLE, x_{12}:DATE, x_{13}:$CARV_1$) ; ..... ; . R_k (x_{k1}:CLE, x_{k2}:DATE, x_{k3}:$CARV_k$) ; . R_s (x'_1:CLE, x'_2:DATE) }

Le terme de représentation est en fait k + 2 relations :

. 1 relation R_p correspondant au SCHEMAP,

. k relations correspondant au SCHEMAV,

. 1 relation R_s correspondant à la table des suppressions.

β) fonction de représentation des dépendances fonctionnelles

Rappelons que les dépendances fonctionnelles entre classes d'objets permettent l'expression des associations 1 → n. Elles ont été définies (cf. partie I, chapitre III) comme des tables.

La transformation est alors immédiate.

Soit dep = TABLE(CLE1,CLE2)

CLE1 est la clé de la première classe, CLE2 celle de la deuxième classe.

Représentation 11 :

$\mathcal{C}$: TABLE(CLE1,CLE2) → R(x_1:CLE1, x_2:CLE2)

Remarquons que cette méthode de représentation des dépendances fonctionnelles conduit à la construction de relations de mêmes clés qui pourront être, ensuite, réunies en une seule relation.

γ) représentation des opérations

Rappelons les opérations du type CLASSE-OB :

(1) ADJCP : fonct(CLASSE-OB,CLE,COMP) CLASSE-OB

opération qui ajoute une occurrence aux aspects permanents d'une classe d'objets ;

(2) ADJCV : fonct(CLASSE-OB,CLE,IDENT,DATE,$CARV_i$) CLASSE-OB
opération qui ajoute une occurrence aux aspects variables d'une classe d'objets ;

(3) SUPC : fonct(CLASSE-OB,CLE,DATE) CLASSE-OB
opération qui détermine la suppression d'un objet dans le SI (par adjonction d'une occurrence à la table des suppressions) ;

(4) Clevalides : fonct(CLASSE-OB,DATE) $\mathcal{P}$(CLE)
opération qui retourne l'ensemble des clés des objets existant à l'instant t ;

(5) valides : fonct(CLASSE-OB,DATE) $\mathcal{P}$ (CLASSE-OB)
état d'une classe à l'instant t : ensemble des objets qui existent à cet instant.

Le constructeur de base du type CLASSE-OB (ainsi que des autres schémas) est le type TABLE, facilement représentable par une relation 1FN. Les opérations du type CLASSE-OB sont facilement représentables par les opérateurs de base de l'algèbre relationnelle. Appelons :
- INSERT l'opérateur qui permet d'insérer un nuplet dans une relation,
- PROJECT l'opérateur qui permet de projeter une relation sur un sous-ensemble de ses constituants,
- SELECT l'opérateur qui permet de sélectionner des nuples d'une relation selon un prédicat.

cob : CLASSE-OB clé : CLE y : comp sp = compp de cob
(1) TOP1 : ADJCP(cob,clé,y) → INSERT(sp,clé,y)
d : date ai : ident y : $CARV_i$ sv = compv de cob
(2) TOP2 : ADJCV(cob,clé,ai,d,y) → INSERT(sv,clé,ai,d,y)
dsup = datesup de cob
(3) TOP3 : SUPC(cob,clé,d) → INSERT(dsup,clé,d)
(4) TOP4 : Clevalides(cob,d) → $PROJECT_c(R_1)$
où
$R_1 = SELECT_{p1}(CLASSE\text{-}OB)$
c : CLE

où

p_1 = {c ∉ $PROJECT_{clé}$(datesup) ou datesup de cob[c] > d}

(5) TOP5 : valides(cob,d) → $SELECT_{p1}$(CLASSE-OB)

où

p_1 = {<cperm, cvar> / ∃ c ∈ Clévalides(cob,d),
cperm = compp de cob[c]
et cvar = compv de cob[c]}

(iv) type COPERATION

Type COPERATION(NCOP,datexec,cobmod,COB1,COB2,top,preg) =
<datexec:TABLE(NCOP,DATE), cobmod:TABLE(NCOP,COB2), top:TABLE(DATE, TEXT(COB1,COB2)>

Représentation 11

$\mathcal{C}$: COPERATION(NCOP,...,preg) → { . R_1 (x_{11}:NCOP, x_{12}:DATE)
. R_2 (x_{21}:NCOP, x_{22}:COB2)
. R_3 (x_{31}:DATE, x_{32}:TEXT)

Le terme de représentation est constitué de 3 relations :

. 1 relation R_1 de degré 2 (datexec), de clé NCOP,

. 1 relation R_2 de degré n+1 (cobmod), n degré de la relation c-objet COB2, de clé NCOP,

. 1 relation R_3 de degré 2 (top), de clé (date).

Remarquons que les deux premières relations peuvent être regroupées.

(v) type CEVENT

type CEVENT(NCEV,datearriv,COB,prédicat,cobconst,copd,DEC,preg) =
<datearriv:TABLE(NCEV,DATE), cobconst:TABLE(NCEV,COB), prédicat:TABLE (DATE,PREDICAT(COB)), copd:DEC>

DEC est un produit cartésien de l triplets de tables de la forme :
<h_{i1}:TABLE(NCEV,COP_i), h_{i2}:TABLE(DATE,$COND_i$), h_{i3}:TABLE(DATE,FAC_i)>

Représentation 12 :

$\mathcal{C}$: CEVENT(NCEV,...,preg) → {
- . Rel_1 (x_{11}:NCEV, x_{12}:DATE)
- . Rel_2 (x_{21}:NCEV, x_{22}:COB)
- . Rel_3 (x_{31}:DATE, x_{32}:PREDICAT)
- . R_{11} (x'_{111}:NCEV, x'_{112}:NCOP)
- . R_{12} (x'_{121}:DATE, x'_{122}:COND)
- . R_{13} (x'_{131}:DATE, x'_{132}:FACTEUR)
- .....
- . R_{k1} (x'_{k11}:NCEV, x'_{k12}:NCOP)
- . R_{k2} (x'_{k21}:DATE,..., x'_{k24}:COND)
- . R_{k3} (x'_{k31}:DATE,..., x'_{k34}:FACTEUR)

Le terme de représentation est constitué de

. 1 relation Rel_1 de degré 2, de clé NCEV (datearriv) ,

. 1 relation Rel_2 de degré n+1, de clé NCEV (cobconst) ,

. 1 relation Rel_3 de degré 2, de clé (date) (prédicat) ,

. 1 * 3 relations où d1,d2 : DATE

- . h_{i1} correspond à une relation de degré 2, de clé (NCEV,NCOP),
- . h_{i2} correspond à une relation de degré 2, de clé (d1),
- . h_{i3} correspond à une relation de degré 2, de clé (d2).

(vi) type SI

type SI (CLAS, COP, CEV, pred S1,..., pred SI_k) =

<cob:CLAS, cop:COP, cev:CEV, t1:TNOMTYPE-OB, t2:TNOMTYPE-SCHEMA, t3:TNOMPTYPEOP, t4:TNOMTYPE-EV, t5:T-CONST, t6:T-MOD, t7:T-OPERA>

Nous donnons la fonction $\mathcal{C}$ de représentation globalement, le détail venant d'être explicité.

Représentation 13 :

$\mathcal{C}$: SI(CLAS,...,pred SI_k) → { . Relclas
. Relop
. Relev
. RT1 (x_{11}:CLAS, x_{12}:IDENT)
. RT2 (x_{21}:CLAS, x_{22}:IDENT)
. RT3 (x_{31}:COP, x_{32}:IDENT)
. RT4 (x_{41}:CEV, x_{42}:IDENT)
. RT5 (x_{51}:CEV, x_{52}:IDENT)
. RT6 (x_{61}:COP, x_{62}:IDENT)
. RT7 (x_{71}:CEV, x_{72}:IDENT)

Le terme de représentation est constitué de :

. Relclas : produit cartésien de k_{cob} * (k+2) relations (CLAS)

. Relop : produit cartésien de k_{cop} * 3 relations (COP)

. Relev : produit cartésien de k_{cev} * (3+1*3) relations (CEV)

. RT1, RT2,..., RT7, 7 relations de degré 2.

4.3. Le SGBD SYNTEX

4.3.1. Architecture

Le SGBD SYNTEX est un SGBD relationnel développé à Toulouse, sur IRIS 80, par le CERT [DEM 75].

Dans sa version initiale il s'agit d'un outil conversationnel, écrit en LP 70, permettant de définir des Bases de Données relationnelles, de créer des occurrences, puis de les interroger.

Cette première version a été complétée ultérieurement par un nouvel outil, écrit en COBOL. L'ensemble permet des questions sous forme d'arbres plus complexes en conversationnel aussi bien qu'en différé par programme. La réponse à la requête est stockée dans un fichier. Cette "couche" supplémentaire, nommée SURSYNTEX, lance l'exécution

du module LM-SYNTEX par l'intermédiaire du module d'interface LM-SYNTEX-INTERF. Le résultat est un ensemble fort complexe ainsi que le montre la figure suivante. En particulier, la multiplicité des communications par fichiers et les mécanismes d'empilement nécessaires à la bonne marche du logiciel, rendent l'exécution coûteuse en place et en temps.

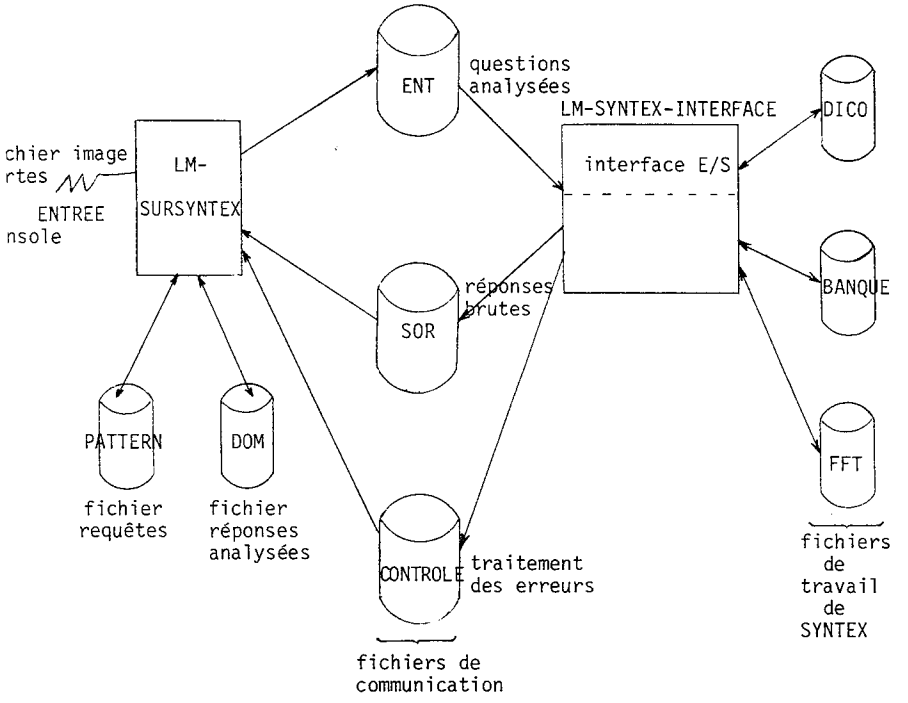


Figure 2. Architecture générale du système SURSYNTEX

4.3.2. Le langage de requêtes

A. Présentation

SYNTEX est un SGBD relationnel, de ce fait son langage est issu directement de la notion mathématique de relation.
Rappelons [DEL 82] qu'il existe deux familles de langages relationnels :

- les langages algébriques dont le principe fondamental consiste à considérer que l'information cherchée peut s'exprimer par applications successives d'opérateurs algébriques sur les relations de la Base de Données,

- les langages prédicatifs qui sont fondés sur le calcul des prédicats du premier ordre, donc la logique mathématique.

Parmi les langages prédicatifs, on distingue :

- les langages à variables n-uplets où, dans les formules du calcul des prédicats, la variable libre et les variables quantifiées désignent des nuplets de relations,

- les langages à variables domaines où, dans les expressions prédicatives, les variables prennent leurs valeurs dans des domaines de constituants de relations.

Le langage de requêtes du SGBD SYNTEX est du type prédicatif principalement à variables domaines, avec une teinte de variables nuplets pour le traitement des "molécules". (cf. B.ii).

Il permet deux grands types d'actions :

- la création et la suppression de relations : le schéma d'une base de données est évolutif ; on peut rajouter de nouvelles relations mais modifier un schéma de relation oblige à sa suppression puis à sa recréation et signifie donc une perte de données ;

- le traitement des données : entrée des informations en mode mise à jour ; interrogation de la base sous forme d'expressions prédicatives parenthésées connectées par des opérateurs logiques.

B. Les types de relations

Une Base SYNTEX est définie comme des nuples de valeurs (atomes ou molécules) interconnectés par des RDA (relations définies sur atomes) ou des RDM (relations définies sur molécules).

(i) atomes

Un atome est une valeur individuelle dont l'existence est indépendante de toute autre valeur. Il peut être numérique (entier ou réel) ou non numérique (type objet ou texte).

exemples : N.B. : tous les exemples présentés ici se réfèrent au déroulement de processeur présenté en annexe 12.

. entier : 0010Ó

. objet : DDE AN RES
TRIG3

(ii) molécules

Une molécule est un nuple ordonné de valeurs.

Les valeurs qui composent une molécule ne sont manipulables qu'en tant que composants de la molécule. Chacune des valeurs peut être numérique ou de type chaine.

exemple : (100, 19810115, 19810201) représente un nuplet de valeurs de la molécule définie pour la relation CHAMDISP (numcli, ddebdis, dfindis).

(iii) relations définies sur atomes (RDA)

Soit A l'ensemble des atomes de même type d'une base SYNTEX.

Une RDA est une relation sur la collection d'ensembles (A1,A2,...,An).

C'est l'ensemble des n-uples ordonnés <a1,a4,...,an> ou ai $\in$ Ai.

(iv) relations définies sur molécules (RDM)

Soit A l'ensemble des atomes de même type d'une base SYNTEX.

Soit M l'ensemble des molécules m1, m2,..., ml d'une base SYNTEX tel

que : $\forall$ i,j : ENTIER
. degré(mi) = degré(mj)
. $\forall$ k : ENTIER, $\forall$ $v_k \in$ mi, $\mu_k \in$ mj
type(v_k) = type(μ_k)
et (estypeENTIER(v_k) ou estypeREEL(v_k)
ou estypeCHAINE(v_k))
et (estypeENTIER(μ_k) ou estypeREEL(μ_k)
ou estypeCHAINE(μ_k))

Une RDM est une relation R sur la collection d'ensembles (A1,A2,..., An,M). C'est un sous-ensemble de l'ensemble des n-uples ordonnés <a1,a2,...,an,mj> où $\forall$ i, ai $\in$ A_i et mj $\in$ M.

C. Manipulation des schémas de relation

(i) création d'un schéma de relation RDA

Un schéma RDA décrit une relation dont les arguments sont uniquement du type atome. On le crée selon la syntaxe :

CRE : modèle, type1, type2,..., typen !

ou type (typei) $\in$ {O, E, R, C}

. O : objet
. E : entier
. R : réel
. C : texte

exemples : . création de la relation trigger TRIG3 (cf. annexe 12) dans le SI,

CRE : TRIG3 <X> <Y> <Z> <W> , 0, 0, 0, 0 !

où . X représente l'atome TRIG3
. Y l'identifiant de la relation
. Z la date de prise en compte de l'événement
. W l'identifiant de l'objet qui a changé d'état

. création, dans la méta-structure SYNTEX, permettant de stocker le schéma conceptuel, de la relation donnant la liste des types triggers actuellement valorisés.

CRE : RELATION <X> DU TYPE TRIGGER, 0 !

où . X représente l'atome "type de trigger".

(ii) création d'un schéma de relation RDM

Un schéma RDM décrit une relation dont un argument et un seul est de type molécule. On le crée selon la syntaxe :

CRE : modèle, type2,..., typen, RDM : nom fichier, (nom champ, type) !

où : . modèle comprend n-1 atomes et une molécule,

. un seul $type_i$ est "molécule",

. nom fichier permet au SGBD SYNTEX de construire le nom du fichier associé à la RDM,

. (nom champ, type) est le couple d'un constituant et de son type ;

exemple : création de la RDM CHAMDISP (chambres disponibles)

CRE : CHAMDISP <X> <Y>, 0, M, RDM : FIC4, (numch,E) (DDEBDIS, C(08))
(DFINDIS, C(08)) !

où : . X représente l'atome "CHAMDISP",

. Y représente la molécule composée de :

. numch, le numéro de la chambre,

. ddebdis, la date de début de disponibilité,

. dfindis, la date de fin.

(iii) suppression de schéma de relation

Sa syntaxe est :

SUP : schéma de relation !

D. Le traitement des données

(i) interrogation des données

Les expressions prédicatives d'accès aux données sont élaborées à partir :

. d'arguments exprimés à l'aide de variables ou ayant des valeurs imposées,

. du qualificateur existentiel EX, ou EX.var indique que l'argument de la relation repéré par la variable "var" ne doit pas être listé,

. d'opérateurs logiques : ET, OU, MAIS PAS, NON.

Nous donnons ici des exemples d'interrogation, on se reportera à [DEM 75] pour plus de détail.

exemples :

α) donner la liste des relations du SC de type TRIGGER :

question :

RELATION <X> DU TYPE TRIGGER ?

réponse :

% trig3

Il y a un trigger dont l'atome "nom" est "trig3".

β) chercher le trigger associé à l'objet DDE AN RES ainsi que l'identificateur de son texte de prédicat :

question :

(COB % DE AN RES % DU TRIGGER <X>) ET (TEXT <Y> DU TRIGGER <X>) ?

réponse :

% trig3 % predic %

Le trigger associé s'appelle "trig3" et son prédicat "predic".

γ) chercher les valeurs d'identifiant de TRIG3 (désigné par X) :

question :

(EX.Y, EX.7, (TRIG3 % TRIG3 % <X> <Y> <Z>)) ?

On ne veut éditer ni l'identifiant de l'objet, ni la date de prise en compte.

réponse :

% 000103 % 000102 % 000101 % 000100 % 000099 %

Il y a actuellement cinq occurrences de TRIG3 dans le SI, leur identifiant varie de 99 à 103.

δ) chercher si la chambre 103 a une période de disponibilité qui se termine le 15 janvier 1981 :
CHAMDISP % CHAMDISP % <X> ET (X.DFINDIS = % 19810115 %) ET (X.NUMCH.103) ?
où . % CHAMDISP % est l'atome dont la valeur est imposée,
. X représente la molécule dont certains champs sont fixés par la qualification qui suit.

(ii) mise à jour de données
Les occurrences de relations sont créées par l'attribution de valeurs aux arguments de types atome et molécule.
exemples :

α) création d'une occurrence de la transaction TR3 dans le SI :
TR3 % TR3 % 000103 % 820622090016 % !
Une occurrence de TR3 est créée avec l'identifiant 103 et la date d'exécution valorisée au 22 juin 1982 à 9 heures 16 secondes.

β) création d'une chambre disponible :
CHAMDISP % CHAMDIPS % <X> : 101, % 19810101 %, % 19810201 % !
La chambre 101 est disponible du 1er janvier au 1er février 1981.
<X> représente la molécule dont les valeurs suivent.

(iii) suppression de données
Les nuples à supprimer sont qualifiés par les valeurs imposées de leurs arguments. Si plus d'un nuple est à supprimer, on utilise la clause TOUS.
exemples :

α) supprimer la chambre disponible 101 de dates [15 janvier 1981, 1er février 1981] :
(CHAMDISP % CHAMDISP % <X>) ET (X.NUMCH = 101) ET
(X.DDEBDIS = % 19810115 %) ET (X.DFINDIS = % 19810201 %) //

β) supprimer toutes les disponibilités de la chambre 100 :
(CHAMDISP % CHAMDISP % <X>) ET (X.NUMCH = 100), TOUS //

4.4. Présentation de la solution

Nous avons, en fait, réalisé deux outils :

- le premier assure le fonctionnement élémentaire d'un SI, évalue chaque condition, chaque facteur, lance l'exécution de chaque texte d'opération. Nous avons montré les inconvénients d'une telle solution dans la partie II. En particulier, en raison de la complexité globale du système, le fonctionnement de l'outil s'est révélé très coûteux en place et en temps. Cet outil nous semble utile à des fins de simulation au niveau conceptuel, pour par exemple réaliser un prototype du système à construire ; ce qui permet de valider la spécification même sous forme de maquette,

- le deuxième assure le fonctionnement d'une transaction c'est-à-dire d'un regroupement d'opérations tel qu'il a été défini dans la partie II et dans [ROL 82]. Cette solution correspond aux choix de réalisation que nous avons faits dans LASSIF et LISSIF.

Après avoir présenté la représentation physique des informations manipulées par le processeur de la dynamique, nous donnerons l'architecture du logiciel et son algorithme programmatoire.

4.4.1. Implémentation des informations

A. La méta-structure

Elle permet de stocker le schéma conceptuel du SI dans une collection de relations exprimées dans le langage de description de données du SGBD SYNTEX.

Nous avons défini un certain nombre de règles d'implémentation qui permettent de représenter une collection de relations, obtenues en appliquant la fonction de représentation définie en 4.2. sur un ensemble d'instanciations de schémas de type, par un ensemble d'occurrences de schémas SYNTEX.

Règle 1

A chaque attribut d'une relation de la description conceptuelle correspond un atome, (cf. paragraphe 4.3.2.)

Règle 2

A chaque nom de relation de type c-objet, trigger, transaction, texte ou déclenche correspond un atome,

Règle 3

La description des relations est exprimée par des relations définies sur atomes (RDA),

Règle 4

L'enchaînement dynamique est exprimé par des RDA.

Compte tenu des règles précédentes, la méta-structure SYNTEX est la suivante :

1. RELATION <X> DU TYPE TRIGGER
   donne les différents triggers du SC d'un SI, c'est-à-dire les types d'événements surgissant dans la vie de l'organisation ;
2. RELATION <X> DU TYPE COB
   donne les différents c-objets composants du SC ;
3. RELATION <X> DU TYPE TRANS
   donne les différents types de transactions répertoriés ;
4. RELATION <X> DU TYPE DECL
   donne les différentes relations "déclenche" répertoriées ;
5. COB <X> DU TRIGGER <Y>
   donne, pour chaque trigger Y, le c-objet X dont il constate le changement d'état ;
6. TEXT <X> DU TRIGGER <Y>
   donne, pour chaque trigger Y, le texte X de son prédicat ;
7. TEXT <X> DE LA TRANS <Y>
   donne, pour chaque transaction Y, le texte X de son programme ;

8. TRIGGER <X> DECL <Y> TRANS <Z>
   indique, pour chaque trigger X, la relation déclenche Y associée et le type de la transaction Z qu'il déclenche ;
9. COMPOSITION RELATION <X> <Y> <W> <Z>
   donne la composition des relations en attributs

où . X est le nom de la relation,
   . Y le nom de l'attribut,
   . W l'ordre (1 pour le premier, 2 pour le deuxième, etc...)
   . Z le type et le format (entier, chaîne de caractères ...) ;

10. COMPOSITION CLE <X> <Y> <Z>
   donne la composition des clés de relations en attributs

où . X est le nom de la relation,
   . Y est le nom de la clé,
   . Z est le nom du constituant faisant partie de la clé.

On pourra trouver, en annexe 12, le compte rendu de l'interrogation de la méta-structure SYNTEX sur le contenu de ces relations.

B. Le système d'information

Nous avons défini de nouvelles règles d'implémentation entre les relations du SI et leur expression en SYNTEX.

Règle 1

A chaque nom de relation du SC d'un SI correspond un atome, (cf. paragraphe 4.3.2.) ;

Règle 2

A chaque attribut d'une relation du SC correspond un champ attribut d'un argument molécule ;

Règle 3

A chaque relation du SC correspond une relation définie sur molécules (RDM).

La forme générale d'une telle RDM est la suivante :
nom-de-relation <X> <Y>, O, M, RDM : nomf, molécule
où :

. X est une variable atome représentant le nom de la relation,

. Y est une variable molécule représentant l'ensemble des champs constituants de la relation,

. O, M, RDM, nomf sont des caractéristiques propres à SYNTEX (cf. 4.3.2-C),

. molécule est la liste des champs de la relation, sous la forme : (nomchamp, typechamp).

On trouvera en Annexe 12 la création des relations dans le SI correspondant à notre exemple de réservations.

C. Réalisation des repères

Notre outil a été réalisé dans le langage COBOL, langage hôte proposé par le SGBD SYNTEX. La seule structure de données aisément manipulables en COBOL étant le fichier, nous avons représenté, de façon immédiate les repères, définis comme des files d'attente au paragraphe 3.2, par des fichiers COBOL.

Les opérations associées au type FILE ne sont pas facilement implémentables par des manipulations de fichiers séquentiels COBOL : ainsi par exemple l'opération RESTE, qui donne les éléments d'une file hors premier élément, peut difficilement être traduite, à moins d'une recopie de tout le fichier hors premier enregistrement.

La complexité des manipulations de fichiers dans certains contextes nous a amenés à choisir des fichiers à accès direct où un enregistrement est accessible par son numéro relatif au début du fichier.
Pour gérer plus facilement les files d'attente, nous avons associé, à chacun des fichiers repères, un premier bloc, différent des suivants, qui contient :

INCLUS	TRAITE

- le numéro du dernier bloc inclus dans la file d'attente du repère, ce qui donne à chaque étape, la longueur de la file d'attente.
- le numéro du dernier bloc traité qui donne, à chaque étape, l'endroit où on en est dans la file d'attente

Les autres blocs contiennent des informations toutes du même type sur les blocs inclus dans la file d'attente. Ils sont différents d'un repère à l'autre.

a) repère de c-objet : représentation du type REP-OBJET (3.2).

NOM-REP	ATTRIBUTS-OB-ACTUAL

où :

. nom-rep est le nom de la relation c-objet,
. attributs-ob-actual ses valeurs de nuple.

b) repère de trigger : représentation du type REP-EVEN

NOM-EV-REP	ID-EV-REPEV	DATE-EV	CAR-OB-REPEV	REP-OB-EV

où

. nom-ev-rep est le nom de la relation trigger
. id-ev-repev est la valeur de son identifiant
. date-ev, la date à laquelle il a été constaté
. car-ob-repev, les attributs du c-objet associé
. rep-ob-ev, le numéro du bloc repère de c-objet associé.

c) repère de transaction à déclencher : représentation du type REP-OP

NOM-REP-OPADEC	VAL-ID-COPPER	DESC-REP-EVCONST-OB	DESC-REP-EVCONST-EV
NOM-DECL-OPADEC	DATE-DEC	REP-EV-OPADEC	

où

- nom-rep-opadec est le nom de la relation transaction à déclencher
- val-id-copper, la valeur de son identifiant
- desc-rep-evconst-ev, les attributs du trigger déclencheur
- desc-rep-evconst-ob, les attributs du c-objet constaté
- nom-decl-opadec, le nom de la relation déclenche
- date-dec, la date où la transaction a été déclenchée
- rep-ev-opadec, le numéro du bloc repère de trigger associé.

d) repère de transaction exécutée : représentation du type REP-OPEX

NOM-OPEX	ID-OPEX	DATE-OPEX	DES-EV-OPEX	DES-OB-OPEX	DATE-DEC-OPEX
NOM-DECL-OPEX	REPOPADEC				

où

- nom-opex est le nom de la transaction exécutée
- id-opex est la valeur de son identifiant
- date-opex, la date de sa fin d'exécution
- des-ev-opex, les attributs du trigger déclencheur
- des-ob-opex, les attributs du c-objet modifié
- date-dec-opex, la date de son déclenchement
- nom-decl-opex, le nom de la relation déclenche
- repopadec, le numéro du bloc repère de transaction à déclencher associé.

4.4.2. Architecture du logiciel réalisé

Le processeur de la dynamique fait appel à SURSYNTEX pour toute question ou mise à jour portant sur le schéma conceptuel ou le système d'information lui-même. Pour réaliser cet enchaînement nous avons conçu le processeur comme un "sur" système de SURSYNTEX ; l'ensemble constituant un seul module de chargement à l'exécution. La figure suivante schématise la complexité de cette réalisation en couches ; d'autant, qu'à chaque nouvelle couche de réalisation, plus on s'éloigne du niveau "système" plus se multiplient des difficultés d'interface et de compatibilité.

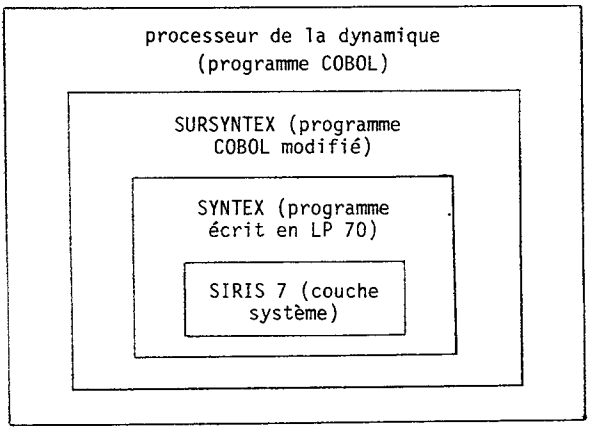


Figure 3. Réalisation en couches du processeur de la dynamique

Ce processeur de la dynamique lance l'exécution de fichiers programmes de façon aléatoire ; il a fallu donc mettre en oeuvre des mécanismes d'assignation dynamique de fichiers ; qui pouvant eux-mêmes faire appel à SURSYNTEX sont réalisés également suivant la schématisation de la figure 3.

On imagine aisément le volume et la complexité de l'ensemble, surtout dans la mesure où le langage COBOL n'est pas vraiment adapté à la réalisation de logiciels manipulant des concepts proches du système d'exploitation.

Remarquons, sur ce plan, à quel point la solution LISP/CEYX présentée dans la partie II, est plus puissante car LISP, langage interprété, permet toutes les manipulations dynamiques de fonctions et de fichiers. Les temps d'exécution sont alors négligeables.

4.4.3. Algorithme programmatoire

Nous donnons l'algorithme programmatoire du processeur de la dynamique. Il suit le déroulement de l'exécution que nous présentons en Annexe 12. On se reportera à cette annexe pour plus de détails ; nous explicitons ici les grandes lignes des actions.

Notons que, COBOL ne permettant pas le parallélisme ni la récursivité, nous avons conçu cet outil comme un module qui boucle tant qu'il reste des événements à traiter.

A chaque étape le système explore complètement la file d'attente créée à l'étape précédente.

PROCESSEUR

résultat : nouveletatSI

nouveletatSI : debp ; tantque boo rep SUPERVISEUR ; finp

SUPERVISEUR

boo : Init ; pchque rob dans REP-OB rep PRENDRE-OBJET (rob) ; fpche
si boo alors
pchque rtrig dans REP-EV rep PRENDRE-TRANS (rtrig) ;
fpche
ens-nuples : pchque rtrans dans REP-OP rep DECLENCHER-TRANS (rtrans) ;
fpche
nouveletatSI : pchque rtransex dans REP-OPEX rep
ACTUALISATION-SI (rtransex) ; fpche
fsi
ens-nuple : SUITE(nuple)
nuple = <nomob : IDENT, val : PRODCAR $<T_1,..,T_n>$, datecre : DATE>

PRENDRE-OBJET

Le module correspond à la PREMIERE ETAPE de l'Annexe 12
{<ordre : ENTIER, type : TYPEBASE>} : RECH-CAR-OBJET (rob) ;
boo : RECH-TRI-OBJET (rob) ;
: MAJ-REP -OBJET (rob) ;

RECH-CAR-OBJET

<ordre : entier, type : TYPEBASE> : question SYNTEX (Annexe 12, Etape 1, A) donnant pour la relation c-objet, sa composition en constituants.

RECH-TRIG-OBJET

(trig, predic) : question SYNTEX (Annexe 12, Etape 1, B) évaluant pour le type d'objet, si un type d'événement lui est associé ainsi que le nom du texte du prédicat ;
COND : si trig $\neq \omega$ alors
évaluation du texte du prédicat (Annexe 12, Etape 1, B)
fsi

boo : si trig ≠ ω et COND
alors . boo ← 1 ;
. création d'un repère de trigger (Annexe 12, Etape 1, C) ;
fsi

MAJ-REP-OBJET

Mise à jour du premier bloc de repère de c-objet (+ 1 à TRAITE) ; (Annexe 12, Etape 1, D)

PRENDRE-TRANS

Ce module correspond à la DEUXIEME ETAPE de l'Annexe 12
. CRE-TRIG (rtrig) ;
(decl, trans) : RECH-TRANS (rtrig) ;
. CRE-REP-TRANS (rtrig, decl, trans) ;

CRE-TRIG

Requête SYNTEX qui crée une nouvelle occurrence de trigger dans le SI ; (Annexe 12, Etape 2, A)

RECH-TRANS

(decl, trans) : question SYNTEX qui donne le type de la relation déclanche et de la relation transaction associées au trigger (Annexe 12, Etape 2, B) ;

CRE-REP-TRANS

. recherche de la valeur de l'identifiant de la dernière occurrence de la transaction donnée, exécutée dans le SI, génération de la nouvelle valeur ;

. création d'une nouvelle occurrence de repère de transaction à déclencher (Annexe 12, Etape 2, C) ;

. mise à jour du repère de trigger étudié (on ajoute 1 à TRAITE) (Annxe 12, Etape 2, D) ;

DECLENCHER-TRANS

Ce module correspond à la TROISIEME ETAPE de l'Annexe 12.

ens-nuples : EXEC-TEXT (rtrans) ;

. pchque nuple dans ens-nuples rep
CRE-REP-TRANSEX (rtrans, nuple)
fpche ;

. MAJ-REP-TRANSEX (rtrans) ;

EXEC-TEXT

. question SYNTEX qui recherche le texte associé à la transaction à déclencher (Annexe 12, Etape 3, A) ;

ens-nuples : évaluation du texte de la transaction (Annexe 12, Etape 3, B) ;

CRE-REP-TRANSEX

(Annexe 12, Etape 3, C)

. création d'un repère de transaction exécutée ;

. mise à jour du premier bloc (+ 1 à INCLUS) ;

MAJ-REP

. mise à jour du premier bloc de repère de transaction exécutée (+ 1 à TRAITE) ;

ACTUALISATION-SI

Ce module correspond à la QUATRIEME ETAPE de l'annexe 12

. CREATRANS (rtransex) ;

. CREADECL (rtransex) ;

. CREAREP (rtransex) ;

CREATRANS

. création d'une occurrence de transaction exécutée dans le SI (Annexe 12, Etape 4, A) ;

CREADECL

. création d'une occurrence de déclenchement effectif dans le SI (Annexe 12, Etape 4, B) ;

CREAREP

(Annexe 12, Etape 4, C)

. création d'une occurrence de repère de c-objet ;

. mise à jour du premier bloc de repère de c-objet (+ 1 à INCLUS) ;

. mise à jour du premier bloc de repère de transaction exécutée (+ 1 à TRAITE) ;

debp

ouverture fichiers ; boo ← 1

Init

boo ← 0

finp

imprimer "fin processeur" ;

fermeture fichiers ;

5. CONCLUSION

Le but de cette partie était de détailler la conception/réalisation d'un logiciel, spécifié à l'aide du langage LASSIF lui même.
Le sous-ensemble choisi du logiciel d'interprétation LASSIF, le "processeur de la dynamique", est la partie que nous connaissons le mieux pour en avoir réalisé une version relationnelle (puis fait réaliser une version en LISP/CEYX).

Rappelons que le choix d'un SGBD relationnel, comme noyau du processeur, a été essentiellement lié à la formalisation du modèle REMORA. La solution obtenue, réalisée autour d'un SGBD prototype, n'est pas très satisfaisante, en particulier, en ce qui concerne les temps d'exécution mais elle a permis de valider des idées qui ont, depuis, fait largement leurs preuves [ISO 81]. D'autres SGBD relationnels sont maintenant opérationnels (tel que MRDS par exemple sur MULTICS) qui rendraient la solution sans doute plus efficace dans la mesure où ils permettent le lancement de commandes systèmes.

Ces propositions nous ont permis également d'élaborer la fonction de représentation TA → modèle relationnel. Mais cette représentation ne peut être qu'incomplète, en effet :

(i) les SGBD actuels ne permettent pas la prise en compte des Contraintes d'Intégrité sauf dans certains cas très limités ; une tentative, pour pallier cette insuffisance, a été faite par R. NASSIF [NAS 83].
L'ensemble des invariants et autres contraintes sur les paramètres doivent être définis par un "système d'intégrité" [FOU 82] ;

(ii) les SGBD actuels n'admettent pas la notion de type de schéma de relation. Le type SI n'est donc pas entièrement descriptible dans un Langage de Description de Données ; c'est pour cette raison que nous avons introduit un ensemble de sept relations permettant d'implémenter la méta-base.

En fait, ainsi que nous l'avons montré au chapitre II, partie I, la modélisation TA est plus riche que la modélisation relationnelle.

Notons, enfin, que nous avons réalisé deux versions du processeur :

(i) l'une en LISP/CEYX très efficace mais irréaliste quant aux données,

(ii) l'autre, autour de SYNTEX, efficace au point de vue données mais inefficace au point de vue système.

La solution que nous préconisons, c'est une réalisation LISP/CEYX gérant les instanciations des TA implémentés, par des primitives du système.

En fait nous dépassons là le domaine de l'Informatique d'organisation pour aborder des problèmes proches des systèmes d'exploitation.

C O N C L U S I O N

BILAN ET PERSPECTIVES

Tout finit bien, puisque tout finit,
J. CHARDONNE "Demi-jour" Grasset

1. BILAN VIS-A-VIS DE NOS OBJECTIFS

Rappelons que l'objectif central de notre travail était de montrer que l'intégration des résultats de l'Informatique de Gestion et de la théorie de la programmation était possible et présentait un intérêt pour les deux domaines de recherche.

1.1. Intérêt du travail vis-à-vis de la conception des SI en IG

Notre développement a contribué à démontrer cet intérêt selon deux axes :

A. Vis-à-vis des propositions du projet REMORA

C'est une sorte d'autovalidation critique.

(i) Au cours du chapitre II partie I, nous avons montré combien la formalisation à l'aide des TA des trois concepts de base du modèle REMORA apportait clarté, en minimisant le nombre de types nécessaire pour décrire le SI, cohérence et complétude en termes de contrôles et d'opérations associées aux types. Nous avons ainsi répondu aux critiques développées contre le modèle lors de CRIS2.

(ii) La réalisation LISP/CEYX du logiciel LASSIF a montré l'intérêt d'utiliser un manipulateur d'arbres abstraits pour construire le SC, vu alors comme une arborescence, CEYX construisant automatiquement des fonctions de manipulation des noeuds que nous avions explicitement programmées dans les logiciels construits précédemment [FOU 82], [KEB 84], [BAN 79].

(iii) La réalisation LISP/CEYX du processeur de la dynamique a démontré la puissance des idées développées par la conception de cet outil ; que la version relationnelle n'avait pu prouver de façon indiscutable.

B. Vis-à-vis de la conception des SI

(i) Lors de l'étude des méthodes de conception de SI (chapitre I partie I) nous avons mis en évidence la confusion des concepts proposés et mis en valeur l'intérêt des concepts d'abstraction et de TA dans ce domaine ; nous avons complété cette justification au début du chapitre II partie I ; notre démarche basée sur le modèle de REMORA pourrait ainsi s'appliquer à d'autres modèles conceptuels de SI.

(ii) Le langage de spécification proposé (chapitre III partie I) allie le souci de simplicité du formalisme, indispensable aux Informaticiens de Gestion, à la rigueur, nécessaire à tout langage fondé sur la formalisation proposée.

(iii) La partie II a démontré combien des environnements de spécification, proches des ateliers logiciels, sont devenus nécessaires alors que rares sont les méthodes de conception des SI qui proposent des ateliers intégrés.

1.2. Intérêt du travail vis-à-vis de la théorie de programmation

(i) Nous avons noté, au cours de notre développement, combien les problèmes spécifiés en théorie de la programmation sont de taille réduite. En appliquant les résultats de ce domaine à l'Informatique de Gestion, nous avons ouvert de nouveaux champs d'application, le volume des informations manipulées est très important et le type des données particulier.

(ii) Nous avons été ainsi amené à introduire de nouveaux types inhabituels en théorie de la programmation, i.e. par exemple le type IDENT (partie I chapitre II paragraphe 2.2.9) et le type SI (partie I chapitre II paragraphe 2.3.6-E). Ces deux types sont liés à la manipulation de la structure conceptuelle du SI (le niveau "meta"), ce qui est assez inusité en programmation.

(iii) Les structures de données manipulées dans le domaine de la théorie de la programmation sont souvent simples et réduites. Le problème du stockage des informations ne se pose pas lorsqu'on manipule des espaces de petite dimension exploitables en Mémoire Centrale.
Les parties II et III, consacrées au développement des logiciels, ont introduit la nécessité d'ouvrir les outils de gestion de données vers des interfaces avec des SGBD puissants.

(iv) La partie II a montré l'intérêt qu'il y a d'intégrer, aux environnements de programmation, un ensemble de règles méthodiques afin de réaliser de véritables ateliers de spécification.

2. PERSPECTIVES D'AVENIR

Nous les développons suivant plusieurs axes.

2.1. L'axe logiciel

Outre améliorer et développer notre logiciel LASSIF, dans ses propositions actuelles, nous devons :

- le compléter car il ne prend pas en compte la maintenance du SI et l'évaluation des conséquences d'une mise à jour, ni l'historique des structures tel que A. KEBAILI l'a réalisé pour le projet REMORA [KEB 84] ;

- l'expérimenter en entreprise, en parallèle avec la validation du langage LASSIF et la démarche de spécification préconisée ; ainsi que nous l'avons fait, nous mêmes, pour un sous-ensemble du logiciel LASSIF (cf. partie III).

2.2. L'axe atelier logiciel

Si la plupart des ateliers logiciels proposés actuellement (cf. [BOU 85] et [GENIE 85] pour une analyse et un inventaire complets) sont très

puissants quant aux outils proposés, peu préconisent une méthode de développement de logiciel [WIL 81] [SNO 80] [LE 84].
Nous pensons que notre axiome :
méthode de conception des SI + atelier de spécification = tout intégré peut faire avancer les idées aussi bien dans le domaine de la conception des SI que dans le domaine du génie logiciel.

2.3. L'axe intelligence artificielle

Avoir défini une démarche d'analyse statique (et non paramétrable) et demander une spécification initiale globale de la réalité à représenter nous semblent les reproches fondamentaux que l'on peut faire à notre logiciel LASSIF.

En fait nous imposons, là, une représentation de la connaissance, sur l'organisation à modéliser, qui doit être complète. Il nous semble possible de s'orienter vers l'acquisition d'une connaissance moins complète, plus approximative, le système faisant, dynamiquement, des déductions et des propositions aux concepteurs.

Nous rejoignons, par cette réflexion, les idées des Bases de Données déductives [BOUJ 84], [MAD 85], [GIR 85], [KOU 85], [BOUZ 85] qui nous semblent applicables à la conception des SI [FLO 85], [NGU 85].

ANNEXES

ANNEXE 1

Exemple des réservations : énoncé

On désire mettre en place un système de réservation de chambres d'hôtel dans des stations de sport d'hiver. Le système :

- prend en compte les demandes faites par les clients,
- fait les réservations correspondantes quand la demande peut être satisfaite (on réserve alors les chambres),
- ou met les demandes "en attente".

Les clients peuvent résilier leur réservation : il faut alors rendre au système les chambres réservées et les transformer en disponibilités ; on essaie ensuite d'étudier les demandes en attente afin de voir si on peut les satisfaire.

De la même façon si une nouvelle chambre est créée dans le système (un hôtel en rend une disponible par exemple), il faut étudier les demandes en attente.

Structure de données de l'exemple

Elle est composée des six classes d'objets suivantes :

1) classe "Cdemande" qui comprend un c-objet (ou schéma) permanent la "demande".
2) classe "Cstation" qui comprend un c-objet permanent la "station".
3) classe "Chôtel" qui comprend un c-objet permanent l'"hôtel".
4) classe "Cclient" qui comprend un c -objet permanent le "client".
5) classe "Créservation" qui comprend un c-objet permanent la "réservation", et deux c-objets (ou sous-schémas) variables : l'état de la réservation ("etatres") et l'annulation de la réservation "annres".
6) classe "Cchambre" qui comprend un c-objet permanent la "chambre" et deux c-objets variables : la chambre disponible ("chdisp") et la chambre réservée ("chres").

La figure 1 représente la collection de classes d'objets.

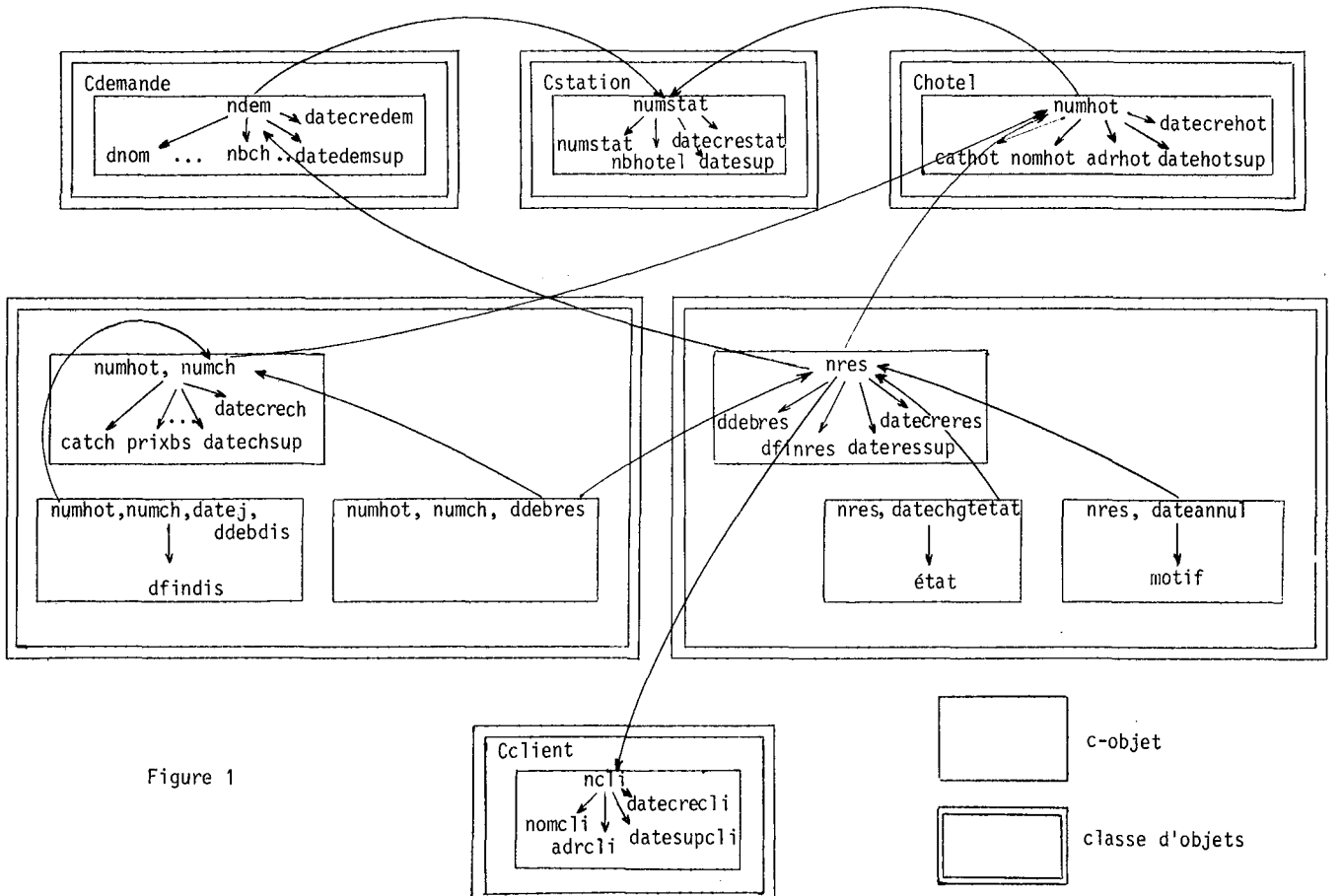

Cdemande
ndem
datecredem
dnom ... nbch ..datedemsup
Cstation
numstat
numstat datecrestat
nbhotel datesup
Chotel
numhot
datecrehot
cathot nomhot adrhot datehotsup
numhot, numch
datecrech
catch prixbs datechsup
numhot,numch,datej,
ddebdis
dfindis
numhot, numch, ddebres
nres
ddebres datecreres
dfinres dateressup
nres, datechgtetat
état
nres, dateannul
motif
Cclient
ncli
datecrecli
nomcli datesupcli
adrcli
c-objet
classe d'objets
dépendance fonctionnelle

Figure 1

Dynamique de l'exemple

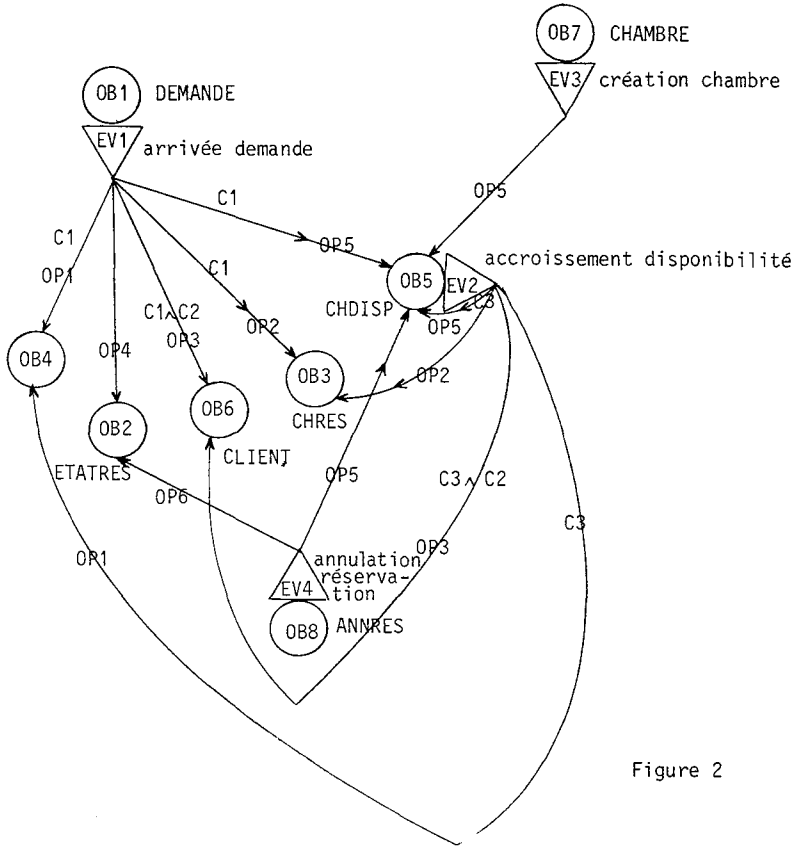
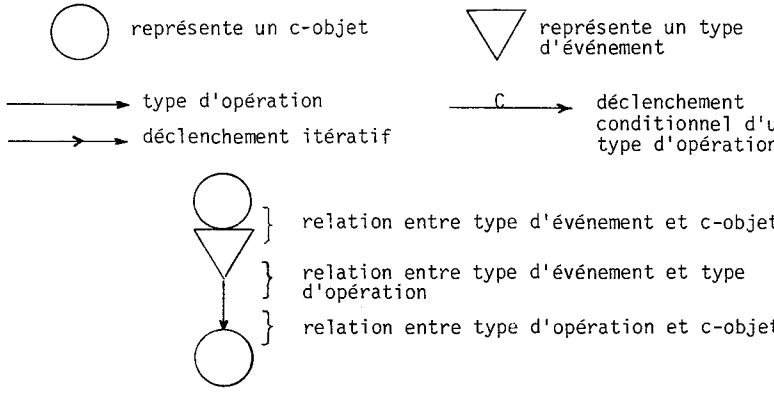


Figure 2



Conditions

C1 : la demande peut être satisfaite

C2 : le client n'est pas connu par le système

C3 : au moins une demande "en attente" peut être satisfaite

types d'opérations

OP1 : création d'une réservation

OP2 : réservation de chambres

OP3 : création client

OP4 : mise à jour état réservation

OP5 : mise à jour disponibilités

OP6 : annulation d'une réservation

types d'événements

EV1 : arrivée d'une demande (EVDEM)

EV2 : accroissement des disponibilités (EVDISP)

EV3 : création d'une chambre (CREACH)

EV4 : annulation d'une réservation (EVANNUL)

c-objets

OB1 : DEMANDE

OB2 : ETATRES (état réservation)

OB3 : CHRES (chambres réservées)

OB4 : RESERVATION

OB5 : CHDISP (chambres disponibles)

OB6 : CLIENT

OB7 : CHAMBRE

OB8 : ANNRES (annulation réservation)

Commentaires

EV1 est la représentation d'une arrivée de demande de réservation correspondant à une nouvelle occurrence du c-objet "demande".

Il induit un nouvel état dans la base de données correspondant :

- à une acceptation si la condition C1 est vraie (c'est-à-dire s'il y a assez de chambres dans un hôtel de catégorie donnée, d'une catégorie donnée, durant une période donnée, dans une station donnée) ; ceci implique l'exécution
  - . de OP1 : création d'une occurrence de réservation (OB4)
  - . de OP2 : création de chambres réservées (OB3)
  - . de OP5 : mise à jour des disponibilités du système (OB5)
  - . de OP4 : mise à jour de l'état de la réservation ("acceptée")(OB2)
- à un refus de la demande (OP4) si la condition C1 est fausse (état de la réservation à "en attente") (OB2)
- à la création d'une occurrence de client (OP3) si les conditions C1 et C2 sont vraies (la demande est satisfaite et le client n'existe pas dans le système) (OB6).

EV2 est la représentation d'un accroissement de disponibilités.

Il déclenche les mêmes opérations que EV1 pour toutes les demandes en attente qui peuvent être satisfaites par cette nouvelle disponibilité. Constatons que EV2 suit systématiquement EV4 ou EV3.

EV3 se produit lorsqu'un hôtel met une nouvelle chambre à la disposition du système. Il déclenche l'exécution de OP6 pour la mise à jour des disponibilités (OB5)

EV4 est la représentation d'une annulation de réservation. Il déclenche la mise à jour (OP5) des disponibilités (OB5) et la mise à jour de l'état de la réservation (OP6, OB2)

ANNEXE 2

Langage structurel appliqué à l'exemple des réservations

A. LES CLASSES D'OBJETS

(1) Classe Cdemande

```
Type Cdemande = CLASSE-OB (<ndem:ent>, demande, DEMANDE, nil, NIL,
                                        datedemsup)
```

Définition du c-objet permanent

```
Type DEMANDE = <dnom:chaîne,   dadresse:chaîne,   dcathot:chaîne,
     ddebdem:date, dfindem:date, nbch:ent, datecredem:date>
```

(2) Classe Cclient

```
Type Cclient = CLASSE-OB (<ncli:ent>, client, CLIENT, nil, NIL,
                                        datesupcli)
```

Définition du c-objet permanent

```
Type CLIENT = <adrcli:chaîne, nomcli:chaîne, datecrecli:date>
```

(3) Classe Chôtel

```
Type Chôtel = CLASSE-OB (<numhot:ent>, hôtel, HOTEL, nil, NIL,
                                        datehotsup)
```

Définition du c-objet permanent

```
Type HOTEL = <cathot:chaîne,    nomhot:chaîne,   adrhot:chaîne,
                                        datecrehot:date>
```

(4) Classe Créservation

```
Type Créservation = CLASSE-OB (<nres:ent>, réservation, RESERVATION,
                resvar, RESVAR, dateressup)
```

Définition du c-objet permanent

```
Type RESERVATION = <ddebres:date, dfinres:date, datecreres:date>
```

Définition du schéma variable

```
Type RESVAR : <demvara:ETATRES, demvarb:ANNRES>
```

Définition des sous-schémas variables

Type ETATRES = <datechgtetat:date, etat:ent>

Type ANNRES = <dateannul:date, motif:chaîne>

(5) Classe Cchambre

Type Cchambre = CLASSE-OB (<numhot:ent, numch:ent>, chambre, CHAMBRE, chvar, CHVAR, datechsup)

Type CHAMBRE = <catch:chaîne, prixbs:ent, prixhs:ent, datecrech:date>

Type CHVAR = <chvara:CHRES, chvarb:CHDISP>

Type CHRES = <ddebres:date, nres:ent>

Type CHDISP = <ddebdis:date, datej:date, dfindis:date>

(6) Classe Cstation

Type Cstation = CLASSE-OB (<numstat:ent>, station, STATION, nil, NIL, datestatsup)

Type STATION = <nomstat:chaîne, nbhotel:ent, datecrestat:date>

B. LES DEPENDANCES FONCTIONNELLES ENTRE CLASSES D'OBJETS

(1) une demande est faite pour une station :

statdem = TABLE (demande de Cdemande [ndem], station de Cstation [numstat])

(2) un hôtel appartient à une station :

hotstat = TABLE (hôtel de Chotel [numhot], station de Cstation [numstat])

(3) une réservation est faite :

- pour une demande

resdem = TABLE (reservation de Creservation [nres], demande de Cdemande [ndem])

- pour un client

```
rescli = TABLE (reservation de Creservation [nres], client de Cclient
                                                    [ncli])
```

- pour un hôtel

```
reshot = TABLE (reservation de Creservation [nres], hôtel de Chotel
                                                    [numhot])
```

(4) une chambre est, à une date donnée, associée à une réservation :

```
chambres = TABLE (chres de Cchambre [numhot, numch, ddebres],
                  reservation de Creservation [nres])
```

(5) une chambre existe pour un hôtel :

```
chhot = TABLE (chambre de Cchambre [numhot, numch], hôtel de Chotel
                                                    [numhot])
```

C - LES CLASSES D'OPERATIONS

(1) création d'une réservation (OP1)

```
Type creares = COPERATION (<ncreares:ent>, datexecreares,
               cobmodcreares, DEMANDE, RESERVATION, topcreares)
avec RESERVATION = MOD (creares)
```

(2) réservation de chambres (OP2)

```
Type reservch = COPERATION (<nreservch:ent>, datexecreservch,
                cobmodreservch, DEMANDE, CHRES, topreservch)
avec CHRES = MOD (reservch)
```

(3) création d'un client (OP3)

```
Type creacli = COPERATION (<ncreacli:ent>, datexeccreacli,
               cobmodcreacli, DEMANDE, CLIENT, topcreacli)
avec CLIENT = MOD (creacli)
```

④ maj état réservation (OP4)

Type majétat = COPERATION (<nmajetat:ent>, datexecmajetat, cobmodmajetat, DEMANDE, ETATRES, topmajetat)

avec ETATRES = MOD (majétat)

⑤ maj disponibilités (OP5)

Type majdisp = COPERATION (<nmajd:ent>, datexecmajd, cobmodmajd, DEMANDE, CHDISP, topmajd)

avec CHDISP = MOD (majdisp)

⑥ annulation d'une réservation (OP6)

Type annreserv = COPERATION (<nannr:ent>, datexecannr, cobmodannr, ANNRES, ETATRES, topannreserv)

avec ETATRES = MOD (annreserv)

D - LES CLASSES D'EVENEMENTS

① Arrivée d'une demande (EV1)

Type evdem = CEVENT (<nevdem:ent>, datearrivevdem, cobconstevdem, DEMANDE, predicevdem, copdevdem, DECEVDEM)

avec DEMANDE = CONST (evdem)

Type DECEVDEM = <x1:DECDEMCREAR, x2:DECDEMRES, x3:DECDEMCREAC, x4:DECDEMMAJE, x5:DEDEMMAJD>

. Type DECDEMCREAR = <h_{11}:TABLE(nevdem,creares), h_{12}:TABLE(DATE,C1), h_{13}:TABLE(DATE,FACTEV1OP1)>

Type C1 = PREDICAT(DEMANDE) Type FACTEV1OP1 = FACTEUR(DEMANDE)

. Type DECDEMRES = <h_{21}:TABLE(nevdem,reservch), h_{22}:TABLE(DATE,C1), h_{23}:TABLE(DATE,FACTEV1OP2)>

Type FACTEV1OP2 = FACTEUR(DEMANDE)

. Type DECDEMCREAC = <h_{31}:TABLE(nevdem,creacli), h_{32}:TABLE(DATE,C2),
h_{33}:TABLE(DATE,FACTEV1OP3)>

Type C2 = PREDICAT(DEMANDE) Type FACTEV1OP3 = FACTEUR(DEMANDE)

. Type DECDEMMAJE = <h_{41}:TABLE(nevdem,majétat), h_{42}:TABLE(DATE,nil),
h_{43}:TABLE(DATE,FACTEV1OP4)>

Type FACTEV1OP4 = FACTEUR(DEMANDE)

. Type DECDEMMAJD = <h_{51}:TABLE(nevdem,majdisp), h_{52}:TABLE(DATE,nil),
h_{53}:TABLE(DATE,FACTEV1OP5)>

Type FACTEV1OP5 = FACTEUR(DEMANDE)

et avec OPERA(evdem) = {creares, reservch, creacli, majétat, majdisp}

(2) Accroissement de disponibilité(EV2)

Type evdisp = CEVENT (<nevdisp:ent>, datearrivevdisp, cobconstevdisp,
CHDISP, predicevdisp, copdevdisp, DECEVDISP)

avec CHDISP = CONST(evdisp)

Type DECEVDISP = <x1:DECDISPRES, x2:DECDISPCREAC, x3:DECDISPCREAR,
x4:DECDISPMAJD>

. Type DECDISPRES = <h_{21}:TABLE(nevdisp,reservch), h_{22}:TABLE(DATE,C3),
h_{23}:TABLE(DATE,FACTEV2OP2)>

Type C3 = PREDICAT(CHDISP) Type FACTEV2OP2 = FACTEUR(CHDISP)

. Type DECDISPCREAC = <h_{31}:TABLE(nevdisp,creacli), h_{32}:TABLE(DATE,
C4), h_{33}:TABLE(DATE,FACTEV2OP3)>

Type C4 = PREDICAT(CHDISP) Type FACTEV2OP3 = FACTEUR(CHDISP)

. Type DECDISPCREAR = <h_{11}:TABLE(nevdisp,creares), h_{12}:TABLE(DATE,
C3), h_{13}:TABLE(DATE,FACTEV2OP1)>

Type FACTEV2OP1 = FACTEUR(CHDISP)

. Type DECDISPMAJD = <h_{51}:TABLE(nevdisp,majdisp), h_{52}:TABLE(DATE, C3), h_{53}:TABLE(DATE,FACTEV2OP5)>

Type FACTEV2OP5 = FACTEUR(CHDISP)

et avec OPERA(evdisp) = {creares, reservch, creacli, majdisp}

(3) Création chambre (EV3)

Type creach = CEVENT (<ncreach:ent>, datearrivcreach, cobconstcreach, CHAMBRE, prediccreach, copdcreach, DECCREACH)

avec CHAMBRE = CONST(creach)

Type DECCREACH = <x1:DECCREACHMAJD>

Type DECCREACHMAJD = <h_{51}:TABLE(ncreach, majdisp), h_{52}:TABLE(DATE, nil), h_{53}:TABLE(DATE,FACTEV3OP5)>

Type FACTEV3OP5 = FACTEUR(CHAMBRE)

et avec OPERA(creach) = {majdisp}

(4) Annulation de réservation(EV4)

Type evannul = CEVENT (<ncevannul:ent>, tabdatarrivannul, tabconsannul, ANNRES, predicannul, copdannul, DECANNUL)

avec ANNRES = CONST(evannul)

Type DECANNUL = <x1:DECANNULMAJDISP, X2:DECANNULETAT>

. Type DECANNULMAJDISP = <h_{51}:TABLE(nevannul, majdisp), h_{52}:TABLE (DATE,nil), h_{53}:TABLE(DATE,FACTEV4OP5)>

Type FACTEV4OP5 = FACTEUR(ANNRES)

. Type DECANNULETAT = <h_{61}:TABLE(nevannul,annreserv), h_{62}:TABLE(DATE, nil), h_{63}:TABLE(DATE,FACTEV4OP6)>

Type FACTEV4OP6 = FACTEUR(ANNRES)

et avec OPERA(evannul) = {majdisp,annreserv}

E. LA CLASSE SI

Type SIRESERV = SI(CLOBRES, CLOPRES, CLEVRES)
où

. CLOBRES est redéfini comme un produit cartésien des classes d'objets :

CLOBRES = <Cr1:Cdemande, Cr2:Cclient, Cr3:Chotel, Cr4:Créservation, Cr5:Cchambre, Cr6:Cstation>

. CLOPRES est redéfini comme un produit cartésien des classes d'opérations :

CLOPRES = <Or1:creares, Or2:reservch, Or3:creacli, Or4:majetat, Or5:majdisp, Or6:annreserv>

. CLEVRES est redéfini comme un produit cartésien des classes d'événements :

CLEVRES = <Er1:evdem, Er2:evdisp, Er3:creach, Er4:evannul>

ANNEXE_3

Application du langage d'énoncé à l'exemple

Nous avons retenu un élément de chaque type d'énoncé tci.

A. SPECIFICATION D'UNE CONTRAINTE D'INTEGRITE

Elle permet de définir un prédicat associé à une classe quelconque. Alors :

resi est du type BOOLEEN,
tci est égal à ASSERT,
Xi est le nom de la contrainte d'intégrité,
Ii définit le corps de la contrainte,
listei donne les champs concernés par la contrainte.

exemple : la période de réservation demandée doit satisfaire la contrainte : la date de fin de réservation demandée est supérieure à la date de début, soit **dfindem > ddebdem.**

ASSERT
CI1 PRED-DEM-1(ddebdem de demande de Cdemande, dfindem de demande de Cdemande)
CI1? contrainte d'intégrité sur les dates de demande de réservation
CI1: BOOLEEN
CI1 = dfindem de demande de Cdemande > ddebdem de demande de Cdemande
dfindem, ddebdem?dates de fin et début de réservation
ddebdem, dfindem: DATE

B. SPECIFICATION D'UNE CONDITION DE DECLENCHEMENT

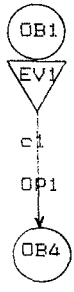
Tout déclenchement d'un type d'opération par un type d'événement doit être associé à un texte de condition s'il est conditionnel.
Dans ce cas :

resi est du type BOOLEEN,
tci est égal à CONDITION,
Xi est le nom de la condition,
listei donne le nom du c-objet en entrée (associé au type d'événement),
Ii définit le texte de la condition.

exemple : déclenchement conditionnel de OP1 par EV1, définition de **c1.** Rappelons :

```
DECDEMCREAR = (h11:TABLE(nevdem,creares),h12:TABLE(DATE,c1),
               h13:TABLE(DATE,FACTEV1OP1))
```

Ceci correspond au sous ensemble du schéma dynamique :



demande

arrivée-demande

c1 : la demande peut être satisfaite

OP1 : création d'une réservation

réservation

On cherche si la demande de réservation peut être satisfaite c'est-à-dire s'il existe assez de chambres, dans la station demandée, dans un hotel de catégorie demandée, pour la période demandée.
Les définitions sont introduites toujours de la même façon :

définition informelle
type du résultat (ou de la donnée)
définition du résultat (explicite en général)

```
CONDITION
c1 COND-DEM-1 (demande de Cdemande)
c1?peut-on satisfaire la demande de réservation?
c1: BOOLEEN
c1 = cardt(chambdisp) )= nbch de demande de Cdemande
                          (ndem)
chambdisp?reçoit  les chambres disponibles satisfaisant  à
la demande
VAR chambdisp : TABLE((numhot:ent,numcli:ent),ddebdis:date)
chambdisp = ∫(x,n)  dans dom(chambre de Cchambre),
            \(d) dans dom(chdisp de Cchambre (x,n)) /
EXIST y dans dom(hotel de Chotel) tq
§même numéro de station§
   hotstat (y) = statdem (ndem de demande de Cdemande)
§même catégorie d'hotel§
  et cathot de hotel de Chotel (y) = dcathot de demande
                        de Cdemande (ndem)}
```

```
§même numéro d'hotel§
  et (x = y
§date début convient§
  et d (= ddebdem de demande de Cdemande (ndem)
§date fin convient§
  et  dfindis  de  chdisp  de Cchambre( (x,n), d)  )=
          dfindem de  demande de Cdemande (ndem)

Cdemande?classe des demandes de réservation
demande?aspect permanent de la classe des demandes
ndem?numéro de la demande concernée
Chotel?classe des hotels
hotel?aspect permanent de la classe des hotels
chambre?aspect permanent de la classe des chambres
chdisp?sous-schéma  variable  de la  classe  des  chambres
représentant les périodes de disponibilites des chambres
```

C. SPECIFICATION D'UN FACTEUR DE DECLENCHEMENT

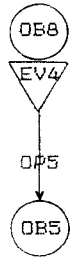
Tout déclenchement d'un type d'opération par un type d'événement doit être associé à un texte de facteur s'il est itératif.
Dans ce cas :

resi est du type TABLE,
tci est égal à FACTEUR,
Xi est le nom du facteur,
listei donne le nom du c-objet en entrée (associé au type d'événement),
les assertions du bloc Ii définissent l'ensemble des tuples qui sont les paramètres du déclenchement itératif de l'opération.

exemple : déclenchement itératif de OP5 par EV4; définition de FACTEV4OP5. Rappelons :

```
DECANNULMAJDISP = (h51:TABLE(nevannul,majdisp),h52:TABLE(DATE,
                  nil),h53:TABLE(DATE,FACTEV4OP5))
```

avec FACTEV4OP5 = FACTEUR(ANNRES)
Ceci correspond au sous ensemble du schéma dynamique :



annres

annulation-réservation

OP5 : mise à jour des disponibilités

chambre-disponible

On cherche toutes les chambres réservées qui doivent être rendues disponibles après une annulation de réservation par un client.

FACTEUR
fact1 FACTEV4OP5(annres de Créservation)
fact1?nuples de chambres réservées rendues au système
fact1:TABLE((numhot:ent,numch:ent,ddebres:date),dfinres:date)
fact1 = { (x,n) dans dom(chambre de Cchambre),
(d1) dans dom(chres de Cchambre (x,n)) ,
(d2) = dfinres de réservation de Créservation
(nres de chres de Cchambre ((x,n),d)) /

EXIST y = ((x,n),d1) dans dom (chres de Cchambre)
tq chambres(y) = annres de Créservation (nres,datannul)}

annres?occurrence de réservation dont le client demande l'annulation;sous-schéma variable de la classe des réservations
chres?sous-schéma variable de la classe des chambres déjà affectées par le système
Créservation?classe des réservations
chambre?aspect permanent de la classe des chambres
Cchambre?classe des chambres
réservation?aspect permanent de la classe des réservations
annres, chres, chambre, Cchambre, Créservation, réservation : SI

D. SPECIFICATION D'UNE OPERATION

Chaque type d'opération du schéma conceptuel du SI est associé à un texte. Dans ce cas :

resi est d'un type c-objet du SI,
tci est égal à OPERATION,
Xi est le nom du texte,
listei donne le nom du facteur en paramètre,
les assertions du bloc Ii précisent les actions associées au type d'opération.

exemple : texte de l'opération OP5 introduit dans la table identifiée par topmajd. Ceci correspond au sous ensemble du schéma dynamique précédent, défini pour le facteur FACTEV4OP5.
Le texte s'appuie sur le facteur précédent. Il prend chaque chambre réservée libérée et la rend disponible en recalculant l'intervalle de disponibilité : en effet, pour que le système reste cohérent, si une chambre est disponible du premier au quinze janvier et qu'on la libère du quinze au trente et un janvier, il faut remplacer la première disponibilité par une chambre disponible du premier au trente et un janvier; idem à droite de l'intervalle de réservation.

```
OPERATION
chdisp TOPMAJD(fact1)
chdisp?représente les disponibilités du système
chdisp: SI
fact1?nuples passés par le facteur précédent FACTEV4OP5
fact1: TABLE((numhot:ent,numch:ent,ddebres:date),dfinres:date)
chdisp  = ADJCV(chdisp de Cchambre,(numhot de  nuple,numch
    de nuple),date,ddebres de nuple,dfinres de nuple)
nuple?nuple à rajouter à la chambre disponible correspondant
à la chambre réservée libérée par une annulation
VAR nuple : TABLE((numhot:ent,numch:ent,ddebres:date),
                          dfinres:date)
ddebres   de  nuple  =  si  chambgauche  alors  ndat1   de
                        chambdisp1
                  sinon ddebres de fact1 fsi
```

```
chambgauche?booléen  indiquant  si une chambre  disponible
dont   l'intervalle  de  disponibilité  jouxte  à  gauche
l'intervalle de la chambre libérée, existe dans le système

VAR chambgauche : BOOLEEN
chambgauche = cardt(chambdisp1) = 1
chambdisp1?nuple correspondant à cette chambre
VAR chambdisp1,chambdisp2:TABLE((nhotd:ent,nchd:ent,ndati:date),
                                ndat2:date)
chambdisp1 ={(x,n) dans dom(chambre de Cchambre),
            {(d1) dans dom(chdisp de Cchambre (x,n) )
            {(d2)   =  dfindis  de  chdisp  de  Cchambre
                       ((x,n),d1) /
     EXIST Y = ((x,n),d1) dans dom(chdisp de Cchambre) tq
numhot de chambre de Cchambre (x,n) = numhot de fact1
et numch de chambre de Cchambre (x,n) = numch de fact1
et dfindis de chdisp de chambre (y) = ddebres de fact1)

dfinres   de  nuple  =  si  chambdroite  alors  ndat2  de
                        chambdisp2
                    sinon dfinres de fact1 fsi

chambdroite?booléen  indiquant si une chambre  disponible,
dont   l'intervalle  de  disponibilité  jouxte  à  droite
l'intervalle de la chambre libérée, existe dans le système

VAR chambdroite: BOOLEEN
chambdroite = cardt(chambdisp2) = 1
chambdisp2?nuple correspondant à cette chambre
chambdisp2 ={(x,n) dans dom(chambre de Cchambre),
            {(d1) dans dom(chdisp de Cchambre(x,n) ),
            {(d2) = dfinres de chdisp de Cchambre ((x,n),d1) /
     EXIST y = ((x,n),d1) dans dom (chdisp de Cchambre tq
numhot de chambre de Cchambre (x,n) = numhot de fact1
et numch de chambre de Cchambre (x,n) = numch de fact1
et ddebdis de chdisp de Cchambre(y) = dfinres de fact1)
numhot de nuple = numhot de fact1
numch de nuple = numch de fact1
```

E. SPECIFICATION D'UN PREDICAT ASSOCIE A UN EVENEMENT

Chaque type d'événement du schema conceptuel de SI est associé à un prédicat. Il spécifie le changement d'état pertinent définissant le type d'évenement par le bloc i. Dans ce cas :

resi est du type BOOLEEN,
tci est égal à PREDICAT,
Xi est le nom du prédicat,
listei donne le nom du c-objet associé au type d'événement.

exemple : texte du prédicat PREDICEVDISP introduit dans la table des textes de prédicats associée à EV2.
Les chambres disponibles peuvent subir plusieurs types de modification : - soit une chambre devient "réservée", la disponibilité correspondante est mise à jour,
- soit une nouvelle disponibilité se produit dans le système (libération d'une réservation ou allocation d'une nouvelle chambre par un hotel).
Le prédicat exprime que l'on peut étudier les demandes en attente seulement si les disponibilités du système viennent de croitre.

```
PREDICAT
predicat2 PREDICEVDISP(chdisp de Cchambre)
predicat2?permet de savoir si on a un accroissement des
disponibilités
predicat2: BOOLEEN
predicat2 = restitution ou création
restitution?booléen indiquant une libération de chambre
réservée
VAR restitution: BOOLEEN
restitution = cardt(restch) = 1
restch?nuple correspondant à une chambre libérée
VAR resch, creatch:TABLE(<numhot:ent, numch:ent, ddebres:
                          date), nil)
restch = {(x,n) dans dom(chambre de Cchambre),
          (d1) dans dom(chres deCchambre(x,n)) /
  EXIST y = ((x,n),d1) dans dom(chres de Cchambre)
  EXIST z = ((x,n),d2) dans dom(chdisp de Cchambre)
tq
ddebres de chres de Cchambre(y) = ddebdis de chdisp de
                          Cchambre(z)}
```

```
Cchambre?classe des chambres
chdisp?chambres disponibles
chres?chambres réservées
Cchambre,chdisp,chres: SI


creation?booléen indiquant si on a crée une nouvelle
chambre
VAR creation: BOOLEEN
creation = cardt(creatch) = 1
creatch?nuple correspondant à une chambre crée
creatch = {(x,n) dans dom(chambre de Cchambre),
          |(d1) dans dom(chdisp de Cchambre(x,n)) /
  EXIST y = (x,n) dans dom(chambre de Cchambre)
  EXIST z = ((x,n),d1)) dans dom(chdisp de Cchambre)
tq
datecre  de chambre de Cchambre(y) = ddebdis de chdisp  de
                        Cchambre(z)}
```

ANNEXE 4

Grammaire abstraite du langage

Nous avons choisi une syntaxe MENTOR qui nous parait plus explicite que la syntaxe CEYX choisie pour l'implémentation de notre logiciel.

Rappelons les principes de MENTOR : [DON 75]

C'est un système général de manipulation d'information structurée s'appliquant spécialement à la construction d'environnements de programmation interactifs. Le noyau de MENTOR est un éditeur dirigé de syntaxe dans lequel chaque objet est représenté comme un arbre opérateur-opérande, appelé habituellement un arbre de syntaxe abstraite. Pour un langage donné, une grammaire en arbre, i.e. la syntaxe abstraite du langage, définit les arbres abstraits corrects dans ce langage.

La syntaxe abstraite d'un langage est obtenue à partir d'opérateurs et de phyla.

(i) les opérateurs étiquettent les noeuds des arbres abstraits. Ils sont de deux sortes : à arité fixe et liste d'opérateurs. Les opérateurs à arité zéro sont les feuilles des arbres abstraits et représentent les atomes du langage. Les opérateurs à arité fixe, non nulle, peuvent avoir des fils de différentes sortes ; les fils d'opérateurs listes doivent être tous de la même sorte. La sorte d'un fils est formalisée par le concept de phylum.

(ii) un phylum est une suite non vide d'opérateurs. A chaque position fils d'un opérateur est associé un phylum. Le phylum indique les opérateurs permis à cet endroit comme opérateurs racines des sous-arbres associés à chaque position fils.
Ces deux concepts permettent de construire des arbres abstraits corrects d'après la phrase du langage entrée. Nous présentons ici la grammaire associée à la solution transaction choisie pour la réalisation et explicitée partie II.
Les phyla sont exprimés en majuscules, les opérateurs en minuscules ; un opérateur liste est représenté par : nom opérateur + ...

Grammaire abstraite du langage structurel

```
sc  →  BASE LISTE-TYPE  LISTE-DEP
BASE ::= baseop
baseop → ID LISTE-CLASSES  LISTE-EVOPER
(1) LISTE-CLASSES ::= lclasse
lclasse → CLASSE +...
CLASSE ::= classeop
classeop → ID PRODCAR ID SCHEMAP ID SCHEMAV ID
ID ::= idop    IDT ::= idtop
idop  →  ;    idtop  →  ;
PRODCAR ::= prodcarop idtop
prodcarop → LISTE-COUPLE
LISTE-COUPLE ::= lcouple
lcouple → COUPLE +...
COUPLE ::= coupleop
coupleop  → ID TYPEPRIM
TYPEPRIM ::= ent bool chaine
ent  →  ;  bool  →  ;  chaine  →  ;
SCHEMAP ::= schemapop idtop
schemapop → LISTE-COUPLE ID DATE
DATE ::= date     date  →  ;
SCHEMAV ::= schemavop idtop
schemavop  →  LISTE-COUPLEV
LISTE-COUPLEV ::= lcouplev
lcouplev → COUPLEV +...
COUPLEV ::= couplevop
couplevop → ID IDT
```

```
(2) LISTE-EVOPER ::= levoper
    levoper → EVOPER +...
    EVOPER ::= evoperop
    evoperop → IDT EV IDT TRANS
    EV ::= evop
    evop → ID ID ID ID ID ID
    TRANS ::= transop
    transop → ID ID LISTE-IDENT ID
    LISTE-IDENT ::= lid
    lid → ID +...
(3) LISTE-TYPE ::= ltype
    ltype → TYPE +...
    TYPE ::= typeop
    typeop → ID TID
    TID ::= produitv prodcar schemap schemav
    produitv → LISTE-CPV
    LISTE-CPV ::= lcpv
    lcpv → CPV +...
    CPV ::= cpvop
    cpvop → ID TYPEBASE
    TYPEBASE ::= TYPEPRIM date
(4) LISTE-DEP ::= ldep
    ldep → DEP +...
    DEP ::= depop
    depop → ID ID ID LISTE-IDENT ID ID LISTE-IDENT
```

Grammaire abstraite du langage d'énoncé

```
    enonceop → IDT IDT LISTE-PARAM  LISTE-RES  LISTE-DONNEES
(1) LISTE-PARAM ::= lparam
    lparam → PARAM +...
    PARAM ::= paramop
    paramop → ID ID ID
(2) LISTE-RES ::= lres
    lres → RES +...
    RES ::= resop
    resop → IDT UTYPE CHAINERES ARBREFORM
    UTYPE ::= TYPEBASE tableop
    tableop → ID ENTIER  LISTE-CPV
    CHAINERES ::= chaineop
    chaineop →  ;
(3) LISTE-DONNEES ::= ldonnées
    ldonnées → DONNEES +...
    DONNEES ::= donneesop
    donneesop → ID CHAINERES
    ARBREFORM ::= arbreformop
    arbreformop → INSTRS
(4) INSTRS ::= affect sialors sialorsinon tantque pourchaque accesdonnées
    affect → TOUTID EXP
    sialorsinon →  EXP  LISTE-INSTRS  LISTE-INSTRS
    sialors → EXP  LISTE-INSTRS
    tantque → EXP  LISTE-INSTRS
    pourchaque → ID ID  Liste-INSTRS
    accèsdonnées → ID  LISTE-VARIABLE  EXPRECH
    LISTE-INSTRS ::= linstrs
    linstrs → INSTRS +...
(5) TOUTID ::= qualifcomplop  qualifop  idop
    qualifcomplop → ID QUALIF
```

```
QUALIF ::= qualifop
qualifop  → ID ID  LISTE-CLE
LISTE-CLE ::= lcle
lcle → LISTE-IDENT +...
(6) EXP  ::= plus moins fois div egal différent supeg infeg infdate
     diffdate cardt ajout adjcp adjcv dom codom app
plus → EXPU EXPU
moins → EXPU EXPU
fois → EXPU EXPU
div → EXPU EXPU
egal → EXPU EXPU
différent → EXPU EXPU
supeg → EXPU EXPU
infeg → EXPU EXPU
infdate → NDATE NDATE
diffdate → NDATE NDATE
cardt → ID
ajout → ID NUPLE
adjcp → ID ID NUPLE
adjcv → ID ID NUPLE
dom → ID ID
codom → ID ID
app → ID ID ENTIER
EXPU ::= EXP TOUTID entierop booleenop AUX
NDATE ::= ndateop
ndateop → ;
ENTIER ::= entierop
entierop → ;
booleenop → ;
(7) NUPLE ::= nupleop
nupleop → LISTE-CONST  LISTE-CONST
LISTE-CONST ::= lconst
lconst → CONSTU +...
```

```
CONSTU ::= entierop  booleenop  chaineop  indateop
AUX ::= etop  ouop  deref
etop  → EXPU EXPU
ouop  → EXPU EXPU
deref  → EXPU
(8) LISTE-VARIABLE ::= lvariable
lvariable  → VARCHERCH +...
VARCHERCH ::= expclé qualifdirect
expclé  → LISTE-IDENT QUALIF
qualifdirect  → ID ID QUALIF
EXPRECH ::= exprechop
exprechop  → LISTE-PREDICAT
LISTE-PREDICAT ::= lprédicat
lprédicat  → PREDICAT +...
PREDICAT ::= prédicatop
prédicatop  → LISTE-CLE QUALIF SUITE-PREDIC
SUITE-PREDIC ::= lprédic
lprédic  → PREDIC +...
(9) PREDIC ::= egalq differq supeq infeq infq suppq
egalq  → ID QUALIFONCU ID QUALIFONCU
differq  → ID QUALIFONCU ID QUALIFONCU
supeq  → ID QUALIFONCU ID QUALIFONCU
infeq  → ID QUALIFONCU ID QUALIFONCU
infq  → ID QUALIFONCU ID QUALIFONCU
suppq  → ID QUALIFONCU ID QUALIFONCU
QUALIFONCU ::= qualifoncop qualifop idop
qualifoncop  →  LISTE-CLE
```

ANNEXE 5

Grammaire CEYX du langage de description des spécifications statiques

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;definition d un schema conceptuel;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(deftree sc)


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;une base et un ensemble de types;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defcons {sc}:scop(base liste-type liste-dep))
(deftree {sc}:base)
(defcons {base}:meta-base)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;un nom,une liste de classes et une liste d evenements operations;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defcons {base}:baseop(id liste-classes liste-evoper))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;univers des classes;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(deftree liste-classes)
(defcons {liste-classes}:lclasse sons~(List classe))
(deftree classe)
(defcons {liste-classes}:meta-liste-classes)
(defcons {classe}: meta-classe)


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;un nom,une cle,un schemap,un schemav et une date de suppression;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defcons {classe}:classeop(id prodcaru id1 schemapu id2 schemavu id3))
(deftree prodcar)
(defcons {prodcar}:meta-prodcar)
(deftree id (val~string))
(deftree idt (val~string))
(deftree schemap)
(defcons {schemap}:meta-schemap)
(deftree schemav)
(defcons {schemav}:meta-schemav)
(defcons {id}:idop)
(defcons {id}:id1)
(defcons {id}:id2)
(defcons {id}:id3)
(defcons {id}:id4)
(defcons {id}:meta-id)
(defcons {idt}:idtop)
(defcons {idt}:meta-idt)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;definition d un produit cartesien,d un schemap,d un schemav;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defcons {prodcar}:prodcarop(liste-couple))
(defcons {schemap}:schemapop(liste-couple id datet))
(defcons {schemav}:schemavop(liste-couplev))
(deftree liste-couple)
(defcons {liste-couple}:meta-liste-couple)
(defcons {liste-couple}:lcouple sons~(List couple))
(deftree couple)
(defcons {couple}:meta-couple)
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;couple d un produit cartesien;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(deftree typebase)
(defcons {couple}:coupleop(id typeprim))
(deftree {typebase}:typeprim)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;types primitifs et types de base;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defcons {typeprim}:ent)
(defcons {typeprim}:bool)
(defcons {typeprim}:chaine)
(deftree datet)
(defcons {datet}:date)
(deftree liste-couplev)
(defcons {liste-couplev}:meta-liste-couplev)
(defcons {liste-couplev}:lcouplev sons~(List couplev))
(deftree couplev)
(defcons {couplev}:meta-couplev)


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;couple d un schemav;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defcons {couplev}:couplevop(id idt))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;univers des evenements operations;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(deftree liste-evoper)
(defcons {liste-evoper}:meta-liste-evoper)
(defcons {liste-evoper}:levoper sons~(List evoper))
(deftree evoper)
(defcons {evoper}:evoperop(id ev idt trans))
(defcons {evoper}:meta-evoper)
(deftree ev)
(defcons {ev}:meta-ev)
(deftree trans)
(defcons {trans}:meta-trans)


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;definition d un evenement et d une transaction;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defcons {ev}:evop(id idt id1 id2 id3 id4))
(defcons {trans}:transop(id idt liste-ident id1))
(deftree liste-ident)
(defcons {liste-ident}:meta-liste-ident)
(defcons {liste-ident}:lid sons~(List id))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;univers des listes de types;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(deftree {sc}:liste-type)
(defcons {liste-type}:meta-liste-type)
(defcons {liste-type}:ltype sons~(List type))
(deftree type)
(defcons {type}:meta-type)
(defcons {type}:typeop(id tidu))
(deftree tid)
(defcons {tid}:meta-tid)
(defcons {tid}:produitv(liste-cpv))
(deftree liste-cpv)
(defcons {liste-cpv}:meta-liste-cpv)
(defcons {liste-cpv}:lcpv sons~(List cpv))
(deftree cpv)
(defcons {cpv}:meta-cpv)
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;couple d un produitv, resolution des types phyla union;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defcons {cpv}:cpvop (id typebase))
;(union-de-type prodcaru '(prodcar idt))
;(union-de-type schemapu '(schemap idt))
;(union-de-type schemavu '(schemav idt))
;(union-de-type typebase '(typeprim datet))
;(union-de-type tidu '(prodcar schemap schemav tid))
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;definition des dependances fonct.;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(deftree {sc}:liste-dep)
(defcons {liste-dep}:meta-liste-dep)
(defcons {liste-dep}:ldep sons"(List dep))
(deftree dep)
(defcons {dep}:meta-dep)
(defcons {dep}:depop(id idt id1 liste-ident id2 id3 liste-ident1))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;definition de types necessaires;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defmodel {string}:booleen(predicate
     (lambda(objet)
       (cond((equal objet "vrai")t)
            ((equal objet "faux")t)
            (t nil)))))
(defmodel {fix}:integer<31(predicate(lambda(n)(<= n 31))))
(defmodel {fix}:integer<12(predicate(lambda(n)(<= n 12))))
(defmodel {fix}:integer<24(predicate(lambda(n)(< n 24))))
(defmodel {fix}:integer<60(predicate(lambda(n)(< n 60))))
(defabbrev {fix}:integer<31 integer<31)
(defabbrev {fix}:integer<12 integer<12)
(defabbrev {fix}:integer<24 integer<24)
(defabbrev {fix}:integer<60 integer<60)
(defmodel mydate(vector(field jj integer<31)
                       (field mm integer<12)
                       (field aa fix)
                       (field ss integer<60)))
```

ANNEXE 6

Listing LISP/CEYX des principaux modules de LASSIF

(Ils correspondent aux algorithmes donnés partie II)

(1) SCHEMA CONCEPTUEL (création d'un schéma conceptuel)

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;; entree du schema conceptuel ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(de schemaconceptuel ( )
(print "entree d un nouveau schema conceptuel ")
(setq schema (scop (meta-base) (meta-liste-type)(meta-liste-dep)))
(send 'debut schema)
(print "nom de la base")
(readline)
(setq nombase (read))
(setq base (baseop(idop) (meta-liste-classes) (meta-liste-evoper)))
(sendq id base nombase)
;;(output "sgsische.sa")                          ;ajout********
;;(print nombase)                                 ;ajout********
;;(output nil)                                    ;ajout********
(send 'debut base)


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;entree des classes d objets ;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(print "entree des classes d objets")
(setq listeclasseob nil)
(setq rep 'n)
(while (equal rep 'n)
      (consclasses) ;
      (definitiontypes) ;
       (print "derniere classe d objet pour ce schema conceptuel (o/n)")
       (setq rep (read))
       (cond ((equal rep 'o) (setq listeclasseobis (lclasse))
                             (sendq sons listeclasseobis listeclasseob)
                             (sendq liste-classes base listeclasseobis)
                            (send 'debut base)
                        ;;;;    (ecritschema '(finclasseop))
 ) ) )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;entree des transactions ;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(print "entree des classes de transaction")
(setq listeclasseevtr nil)
(setq rep 'n)
(while (equal rep 'n)
       (consevtr)
       (print "derniere classe de evenement transaction pour ce schema")
       (print " conceptuel (o/n)")
       (setq rep (read))
       (cond ((equal rep 'o)
              (setq listeclasseevtrbis (levoper))
              (sendq sons listeclasseevtrbis listeclasseevtr)
              (sendq liste-evoper base listeclasseevtrbis)
              (send 'debut base)
;;;;              (ecritschema '(finevoperop))
 ) ) )
(sendq base schema base)
(send'debut schema)
```

SCHEMA CONCEPTUEL (suite)

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;entree des schemas variables;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(print "entree de la definition des schemas variables")
(setq listeschemavar nil)
(setq rep 'n)
(while (equal rep 'n)
       (conschemavar)
       (print "dernier schema variable pour ce schema (o/n)")
       (setq rep (read))
       (cond ((equal rep 'o)
              (setq listeschemavarbis (ltype))
              (sendq sons listeschemavarbis listeschemavar)
              (sendq liste-type schema listeschemavarbis)
              (send 'debut schema)
;;              (ecritschema '(fintypeop))
 ) ) )
;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;entree des dependances fonctionnelles;;
;;;;;;;;;;;;;;;;;;;;;;;;;;
(print "entree des dependances fonctionnelles")
(setq listedepfonc nil)
(setq rep 'n)
(while (equal rep 'n)
        (consdepfonc)
        (print "derniere dependance fonctionnelle pour ce schema(o/n)")
        (setq rep (read))
        (cond ((equal rep 'o)
              (setq listedepfoncbis (ldep))
              (sendq sons listedepfoncbis listedepfonc)
              (sendq liste-dep schema listedepfoncbis)
              (send 'debut schema)
;;              (ecritschema '(findepop))
 ) ) )
 ) ;fin schema conceptuel
```

(2) CONSCLASSES (construction d'une classe d'objets)

```
(de consclasses ()
(print "nom de la classe")
(readline)
(setq nomclasse (read))
(print "cle")
(setq nomcle (read))
(print "nom de schema permanent")
(setq nomcompp (read))
(print "schemap")
(setq schemap (read))
(print "nom de schema variable")
(setq nomcompv (read))
(print "schemav")
(setq schemav (read))
(print "nom table suppression")
(setq tablesup (read))
(analyse1 nomcle)
(analyse2 schemap)
(setq compv (idtop))
(sendq val compv schemav)
(terpri)
(setq classeob (classeop (idop) cle (idop) compp (idop) compv (idop)))
(sendq id classeob nomclasse)
(sendq id1 classeob nomcompp)
(sendq id2 classeob nomcompv)
(sendq id3 classeob tablesup)
(send 'debut classeob)  )
(print "fin")
```

③ DEFINITIONTYPES (saisie des produits cartésiens correspondant à la clé, au schéma permanent, au schéma variable)

```
(de definitiontypes ()
(setq typem "ident")
;;;;;;;;;;;cle en majuscules;;;;;;;;;;;
(cond ((eq(type typem)(type nomcle))
      (prin "entree de la definition de la cle:")
      (prin (sendq val cle))
      (terpri)
      (setq typecle (read))
      (constructprod typecle)
;rend coupleprod concatenation des couples du produit cartesien;
      (setq prv (lcouple))
      (sendq sons prv coupleprod)
      (setq cle (prodcarop prv))
;remplacement dans classeob;;
      (sendq prodcaru classeob cle)
    (send 'debut classeob) ) )
;;schemap est en majuscules;;
(cond ((eq(type typem)(type schemap))
      (prin "entree de la definition du schema permanent sans date:")
      (prin(sendq val compp))
      (terpri)
      (setq typecompp (read))
      (constructprod typecompp)
;rend coupleprod concatenation des couples du produit cartesien;;
       (setq prv (lcouple))
       (sendq sons prv coupleprod)
      (setq compp (schemapop prv (idop) (date)))
      (print "entree du champ date")
      (setq champdate (read))
      (sendq id compp champdate)
      (sendq datet compp "date")
;;remplacement dans classeob;;
      (sendq schemapu classeob compp)
      (send 'debut classeob) ) )

;;schemav est par definition en majuscules;;
(prin  "entree de la definition du compv:")
(prin(sendq val compv))
(terpri)
(setq typecompv (read))
(conscouplev typecompv)
;;rend couplevprod concatenation des couples du produit cartesien;;
;creation du schemav;
(setq prv (lcouplev))
(sendq sons prv couplevprod)
(setq compv (schemavop prv))
;;remplacement dans classeob;;
(sendq schemavu classeob compv)
(send 'debut classeob)
;;rajouter ces classes aux precedentes;;
;;(ecritschema classeob)
(newr listeclasseob classeob) )
```

(4) CONSEV (saisie d'une classe d'événements)

```
;;;;;;;;;;;;;
; construction d evenements
;;;;;;;;
(print "debcons")
(de consev ()
  (print "nom de la cle  de l evenement")
  (setq nomcleev (read))
  (print "nom de la table date d arrivee")
  (setq nomdatearriv (read))
  (print "nom de la table objet constate")
  (setq nomtableconst (read))
  (print "nom de l objet constate")
  (setq nomcobconst (read))
  (print "nom de la table des predicats")
  (setq nompredicat (read))
  (print "nom de la transaction")
  (setq notrans (read))
  (setq evenement (evop (idop) (idop) (idop) (idop) (idop) (idop)))
  (sendq id evenement nomcleev)
  (sendq idt evenement nomdatearriv)
  (sendq id1 evenement nomtableconst)
  (sendq id2 evenement nomcobconst)
  (sendq id3 evenement nompredicat)
  (sendq id4 evenement notrans)
 ) ;fin consev
(print "fincons")
```

(5) CONSTRANS (saisie de la transaction)

```
;;;;;;;;;;;;;;;;
; construction de la transaction
;;;;;;;;;;;;;;;;;;
;
(de constrans ()
  (print "nom de la cle de la transaction")
  (readline)
  (setq nomcletrans (read))
  (print "nom de la table date execution")
  (setq nomdatexec (read))
  (print "nom de la table des textes")
  (setq nomtexte (read))
  (print "entree de la liste des objets modifies")
  (setq listident (read))
  (constlistident listident)
  (setq prvident (lid))
  (sendq sons prvident coupleident)
  (setq transaction (transop (idop) (idop) prvident (idop)))
  (sendq id transaction nomcletrans)
  (sendq idt transaction nomdatexec)
  (sendq id1 transaction nomtexte)
(terpri) ) ;fin constrans
;
(de constlistident (listident)
  (setq coupleident nil)
  (mapcar '(lambda (x)                                  décodage d'une liste
            (setq filsident (idop))                     d'identificateurs
            (sendq val filsident(string(car x)))
            (terpri)
            (newr coupleident filsident) ) listident)
)
```

(7) CONSCHEMAVAR (saisie d'un sous-schéma variable)

```
(print "debut")
;;;;;;;;;;;;;;;;;;;;
;construction d un schema variable;;
;;;;;;;;;;;;;;;;;;;;
(de conschemavar ()
    (print "nom du schema variable")
    (setq nomschemavar (read))
    (print "entree de sa description")
    (setq schemavar (read))
    (conscpv schemavar)
;;;;rend couplecpv concatenation des couples du produit cartesien;;
;creation du type schema variable;;;
(setq prv (lcpv))
(sendq sons prv couplecpv)
(setq schemavob (produitv prv))
(terpri)
(setq schemavarob (typeop (idop) schemavob))
(sendq id schemavarob nomschemavar)
(send 'debut schemavarob)
(terpri)
;;(ecritschema listeschemavar)
(newr listeschemavar schemavarob) )
(print "fin")
```

(8) CONSDEPFONC (saisie d'une dépendance fonctionnelle)

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;construction dependance fonctionnelle;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(de consdepfonc()
(print "nom de la fonction")
(setq nomdepfonc (read))
(print "nom du premier sous-schema")
(setq nomsouschema1 (read))
(print "nom de la premiere classe")
(setq nomclassel (read))
(print "composition de la premiere cle")
(setq listeclel (read))
(constlistident listeclel)
(setq prvident1 (lid))
(sendq sons prvident1 coupleident)
(print "nom du deuxieme sousschema")
(setq nomsouschema2 (read))
(print "nom de la deuxieme classe")
(setq nomclasse2(read))
(print "composition de la deuxieme cle")
(setq listecle2 (read))
(constlistident listecle2)
(setq prvident2 (lid))
(sendq sons prvident2  coupleident)
(setq depfonc (depop (idop)(idop)(idop) prvident1 (idop)(idop)prvident2))
(sendq id depfonc nomdepfonc)
(sendq idt depfonc nomsouschema1)
(sendq id1 depfonc nomclassel)
(sendq id2 depfonc nomsouschema2)
(sendq id3 depfonc nomclasse2)
(send 'debut depfonc)
(terpri)
(newr listedepfonc depfonc) )
```

ANNEXE 7

Déroulement (partiel) de l'exécution de LASSIF et affichage en fin d'exécution

Le module LISP qui permet de lancer la création du schéma conceptuel se trouve (sur VAX Nancy) en
usr/users/sgdb/thiery/mlcinit.ll

Il se charge en tapant, après appel de ceyx (version 15) :
(load mlcinit.ll)

Si tout se passe bien, un compte rendu du chargement des différents modules de LASSIF est affiché, qui se termine par :
"vous pouvez y aller"

Il suffit alors de taper :
(schemaconceptuel)

Après saisie du nom de la base de données à construire, le système demande la description des définitions dans l'ordre explicité précédemment. A chaque étape il affiche l'état intermédiaire des spécifications déjà entrées.

(1) Entrée d'une classe d'objet : classe Creservation

```
entree des classes d objets
nom de la classe
? "Creservation"
"Creservation"
cle
? (nres : ent)
(nres : ent)
nom de schema permanent
? "reservation"
"reservation"
schemap
? "RESERVATION"
"RESERVATION"
nom de schema variable
? "resvar"
"resvar"
schemav
? "RESVAR"
"RESVAR"
nom table suppression
? "dateressup"
"dateressup"
```

```
{type Creservation=classe-ob  (<nres:ent>                    } affichage du
{,reservation,RESERVATION,resvar,RESVAR,dateressup)          } compte rendu
```

```
entree de la definition du schema permanent sans date:RESERVATION
? (debres : date , dfinres : date)
(debres : date , dfinres : date)
```

```
entree du champ date
? "datecreres"
"datecreres"
{type Creservation=classe-ob  (<nres:ent>                                  } affichage après prise en compte
{,reservation,<debres:date,dfinres:date,datecreres:date>                   } du produit cartésien correspondant
{,resvar,RESVAR,dateressup)                                                } au schéma permanent
```

```
entree de la definition du compv:RESVAR
? (demvara : "ETATRES" , demvarb : "ANNRES")
(demvara : "ETATRES" , demvarb : "ANNRES")
```

```
{type Creservation=classe-ob  (<nres:ent>                                  } affichage du compte rendu de la
{,reservation,<debres:date,dfinres:date,datecreres:date>                   } description de la classe
{,resvar,<demvara:ETATRES,demvarb:ANNRES>                                  } d'objets
{,dateressup)                                                              }
```

```
derniere classe d objet pour ce schema conceptuel (o/n)
? o
```

(On répond "oui" s'il n'y a pas d'autres classes à entrer)

Nous avons souligné ce qui est entré par le concepteur

(2) Description des schémas variables : classe Creservation

```
entree de la definition des schemas variables
nom du schema variable
? "ETATRES"
"ETATRES"
entree de sa description
? (datechgtetat : date , etat : ent)
(datechgtetat : date , etat : ent)
```

{type ETATRES = <datechgtetat:date,etat:ent> } affichage du sous-schéma entré

```
dernier schema variable pour ce schema (o/n)
? n
n
nom du schema variable
? "ANNRES"
"ANNRES"
entree de sa description
? (dateannul : date , motif : chaine)
(dateannul : date , motif : chaine)
```

{type ANNRES = <dateannul:date,motif:chaine> } affichage du sous-schéma

```
dernier schema variable pour ce schema (o/n)
? o
```

(3) Description d'une classe d'événement/transaction : EVANNUL

```
entree des classes de transaction
nom de l evenement
? "EVANNUL"
"EVANNUL"
nom de la transaction
? "TRANSANNUL"
"TRANSANNUL"
nom de la cle de l evenement
? "nevannul"
"nevannul"
nom de la table date d arrivee
? "tabdatarrivannul"
"tabdatarrivannul"
nom de la table objet constate
? "tabconsannul"
"tabconsannul"
nom de l objet constate
? "annres"
"annres"
nom de la table des predicats
? "predicannul"
"predicannul"
nom de la transaction
? "transannul"
"transannul"
nom de la cle de la transaction
? "ntransannul"
"ntransannul"
nom de la table date execution
? "tabdatexecannul"
"tabdatexecannul"
nom de la table des textes
? "texteannul"
"texteannul"
entree de la liste des objets modifies
? ((etatres)(chres)(chdisp))
((etatres)(chres)(chdisp))
```

affichage par le logiciel classe EVANNUL / TRANSANNUL

```
{evenement : EVANNUL(nevannul,tabdatarrivannul,tabconsannul,annres,predicannul,
 transannul)

{transaction :TRANSANNUL(ntransannul,tabdatexecannul,etatres,chres,chdisp,
 texteannul)

 derniere classe de evenement transaction pour ce schema
  conceptuel (o/n)
 ? o
```

(4) Saisie des dépendances fonctionnelles : entre la classe d'objets Creservation et les autres classes d'objets du SI.

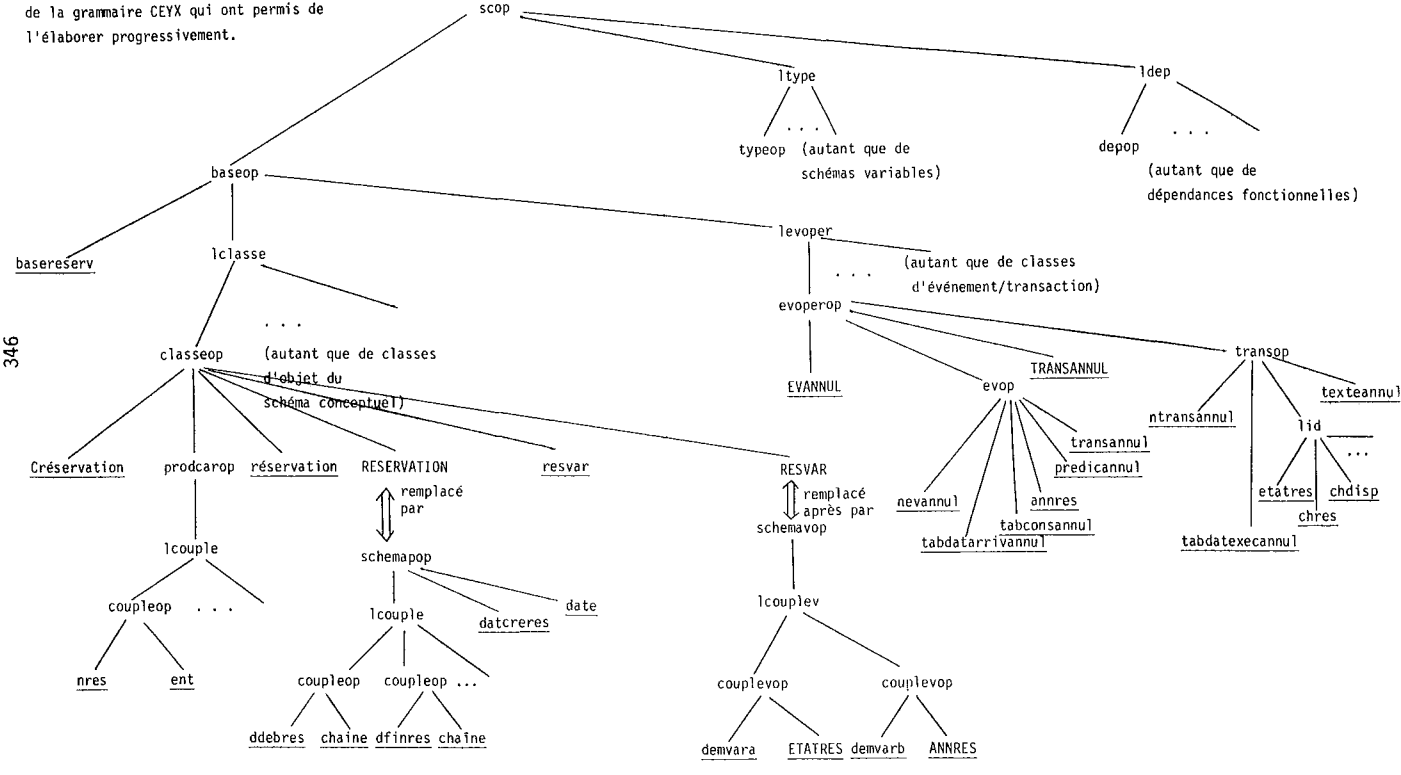
```
entree des dependances fonctionnelles
nom de la fonction
? "resdem"
"resdem"
nom du premier sous-schema
? "reservation"
"reservation"
nom de la premiere classe
? "Creservation"
"Creservation"
composition de la premiere cle
? ((nres))
((nres))

nom du deuxieme sousschema
? "demande"
"demande"
nom de la deuxieme classe
? "Cdemande"
"Cdemande"
composition de la deuxieme cle
? ((ndem))
((ndem))

{resdem=TABLE(reservation de Creservation[nres],
 demande de Cdemande[ndem,])}  affichage de la dépendance fonctionnelle saisie

derniere dependance fonctionnelle pour ce schema(o/n)
? n
n
nom de la fonction
? "rescli"
"rescli"
nom du premier sous-schema
? "reservation"
"reservation"
nom de la premiere classe
? "Creseravation"
"Creseravation"
composition de la premiere cle
? ((nres))
((nres))

nom du deuxieme sousschema
? "client"
"client"
nom de la deuxieme classe
? "Cclient"
"Cclient"
composition de la deuxieme cle
? ((ncli))
((ncli))

{rescli=TABLE(reservation de Creseravation[nres],
 client de Cclient[ncli])}  affichage

derniere dependance fonctionnelle pour ce schema(o/n)
? o
```

(5) Arbre résultat associé à cet exemple

```
(pprint schema)
(#:Tree:sc:scop
      #[#(#:Tree:sc:base:baseop #[basereserv #(#:Tree:liste-classes:lclasse
      (#(#:Tree:classe:classeop #[Creservation #(#:Tree:prodcar:prodcarop
      #[#(#:Tree:liste-couple:lcouple (#(#:Tree:couple:coupleop #[nres ent]
      . #[])) . #[])] . #[]) reservation #(#:Tree:schemap:schemapop #[#(#:
      Tree:liste-couple:lcouple (#(#:Tree:couple:coupleop #[debres date] .
      #[]) #(#:Tree:couple:coupleop #[dfinres date] . #[])) . #[])
      datecreres date] . #[]) resvar #(#:Tree:schemav:schemavop #[#(#:
      Tree:liste-couplev:lcouplev (#(#:Tree:couplev:couplevop #[demvara
      ETATRES] . #[]) #(#:Tree:couplev:couplevop #[demvarb ANNRES] . #[]))
      . #[])] . #[]) dateressup] . #[])) . #[]) #(#:Tree:liste-evoper:
      levoper (#(#:Tree:evoper:evoperop #[EVANNUL #(#:Tree:ev:evop #[
      nevannul tabdatarrivannul tabconsannul annres predicannul transannul]
      . #[]) TRANSANNUL #(#:Tree:trans:transop #[ntransannul
      tabdatexecannul #(#:Tree:liste-ident:lid (#(#:Tree:id:idop #[] . #[
      etatres]) #(#:Tree:id:idop #[] . #[chres]) #(#:Tree:id:idop #[] . #[
      chdisp])) . #[]) texteannul] . #[])] . #[])) . #[])] . #[]) #(#:
      Tree:sc:liste-type:ltype (#(#:Tree:type:typeop #[ETATRES #(#:Tree:
      tid:produitv #[#(#:Tree:liste-cpv:lcpv (#(#:Tree:cpv:cpvop #[
      datechgtetat date] . #[]) #(#:Tree:cpv:cpvop #[etat ent] . #[])) .
      #[])] . #[])] . #[]) #(#:Tree:type:typeop #[ANNRES #(#:Tree:tid:
      produitv #[#(#:Tree:liste-cpv:lcpv (#(#:Tree:cpv:cpvop #[dateannul
      date] . #[]) #(#:Tree:cpv:cpvop #[motif chaine] . #[])) . #[])] . #[]
      )] . #[])) . #[]) #(#:Tree:sc:liste-dep:ldep (#(#:Tree:dep:depop #[
      resdem reservation Creservation #(#:Tree:liste-ident:lid (#(#:Tree:
      id:idop #[] . #[nres])) . #[]) demande Cdemande #(#:Tree:liste-ident:
      lid (#(#:Tree:id:idop #[] . #[ndem])) . #[])] . #[]) #(#:Tree:dep:
      depop #[rescli reservation Creseravation #(#:Tree:liste-ident:lid (
      #(#:Tree:id:idop #[] . #[nres])) . #[]) client Cclient #(#:Tree:
      liste-ident:lid (#(#:Tree:id:idop #[] . #[ncli])) . #[])] . #[])) .
      #[])] . #[])
```

Arbre abstrait correspondant à un sous-ensemble de l'exemple des réservations

Les feuilles de l'arbre sont soulignées.
Les noeuds de l'arbre sont les constructeurs de la grammaire CEYX qui ont permis de l'élaborer progressivement.



Compte rendu de l'exécution

Affichage de la spécification entrée (en fin d'entrée de la spécification)

```
schema
 base
base : basereserv
liste des classes
type Creservation=classe-ob  (<nres:ent>
,reservation,<ddebres:date,dfinres:date,datecreres:date>
,resvar,<demvara:ETATRES,demvarb:ANNRES>
,dateressup)

type Cchambre=classe-ob  (<numhot:ent,numch:ent>
,chambre,<catch:chaine,prixbs:ent,prixhs:ent,datecrech:date>
,chvar,<chvara:CHRES,chvarb:CHDISP>
,datechsup)

type Cstation=classe-ob  (<numstat:ent>
,station,<nomstat:chaine,nbhotel:ent,datecrestat:date>
,nil,<nil:NIL>
,datestatsup)

type Cdemande=classe-ob  (<ndem:ent>
,demande,<dnom:chaine,dadr:chaine,dcat:chaine,ddebdem:date,dfindem:date,nbch:
ent,datedemcre:date>
,nil,<nil:NIL>
,datedemsup)

type Cclient=classe-ob  (<ncli:ent>
,client,<adrcli:chaine,nomcli:chaine,datecrecli:date>
,nil,<nil:NIL>
,dateclisup)

type Chotel=classe-ob  (<numhot:ent>
,hotel,<cathot:chaine,nomhotel:chaine,adrhot:chaine,datecrehot:date>
,nil,<nil:NIL>
,datehotsup)
```

affichage des six classes d'objets

```
liste des transactions
evenement : EVDEM(nevdem,tabdatarrivdem,tabconstdem,demande,predicdem,transdem
)

transaction :TRANSDEM(ntransdem,tabdatexecdem,reservation,etatres,client,chres
,chdisp,texttransdem)

evenement : CREACH(ncreach,tabdatarrivch,tabconstch,chambre,predicch,
transcreach)

transaction :TRANSCREACH(ncreach,tabdatexecch,chdisp,textch)

evenement : EVDISP(nevdisp,tabdatarrivdisp,tabconstdisp,chdisp,predicdisp,
transdisp)

transaction :TRANSDISP(ntransdisp,tabdatexecdisp,reservation,client,chres,
chdisp,textedisp)

evenement : EVANNUL(nevannul,tabdatarrivannul,tabconstannul,annres,predicannul
,transannul)

transaction :TRANSANNUL(ntransannul,tabdatexecannul,etatres,chdisp,texteannul)
```

affichage des quatre classes d'événements/ transactions

Affichage (suite)

types

```
type ETATRES = <datechgtetat:date,etat:ent>

type ANNRES = <dateannul:date,motif:chaine>

type CHRES = <ddebres:date,nres:ent>

type CHDISP = <debdisp:date,dfindisp:date>
```

dependances fonctionnelles entre classes d objets

```
statdem=TABLE(demande de Cdemande[ndem],
station de Cstation[numstat])

hotstat=TABLE(hotel de Chotel[numhot],
station de Cstation[numstat])

resdem=TABLE(reservation de Creservation[nres],
demande de Cdemande[ndem])

rescli=TABLE(reservation de Creservation[nres],
client de Cclient[ncli])

reshot=TABLE(reservation de Creservation[nres],
hotel de Chotel[numhot])

chambres=TABLE(chres de Cchambre[numhot,numch,ddebres],
reservation de Creservation[nres])

chhot=TABLE(chambre de Cchambre[numhot,numch],
hotel de Chotel[numhot])
```

ANNEXE 9

Grammaire du langage d'énoncé

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;grammaire enonce;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(deftree enonce)
(defcons {enonce}:meta-enonce)
(defcons {enonce}:enonceop(id idt liste-param liste-res liste-donnees))
(deftree liste-res)
(deftree liste-param)
(deftree liste-donnees)
(defcons {liste-res}:meta-liste-res)
(defcons {liste-param}:meta liste-param)
(defcons {liste-donnees}:meta-liste-donnees)
(defcons {liste-res}:lres sons~(List res))
(deftree res)
(defcons {res}:meta-res)
(defcons {liste-param}:lparam sons~(List param))
(deftree param)
(defcons {param}:meta-param)
(defcons {liste-donnees}:ldonnees sons~(List donnees))
(deftree donnees)
(defcons {donnees}:meta-donnees)
(defcons {res}:resop(idt utype chaineres arbreform))
(deftree chaineres)
(defcons {chaineres}:chaineop (val~string))
(deftree arbreform)
(defcons {arbreform}:meta-arbreform)
(defcons {param}:paramop(id idt idl))
(defcons {donnees}:donneesop(id chaineres))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;description du noyau;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defcons {arbreform}:arbreformop(instrs))
(deftree instrs)
(defcons {instrs}:meta-instrs)
(defcons {instrs}:affect(toutid exp))
(defcons {instrs}:sialors(exp liste-instrs))
(defcons {instrs}:sialorsinon(exp liste-instrs liste-instrs1))
(defcons {instrs}:tantque(exp liste-instrs))
(defcons {instrs}:pourchaque(id idl liste-instrs))
(defcons {instrs}:accesdonnees(id liste-variable exprech))
(deftree liste-instrs)
(deftree {liste-instrs}:liste-instrs1)
(defcons {liste-instrs}:meta-liste-instrs)
(defcons {liste-instrs}:linstrs sons~(List instrs))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;les expressions;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(deftree exp)
(deftree {exp}:exp1)
(defcons {exp}:plusop(exp exp1))
(defcons {exp}:moins(exp exp1))
(defcons {exp}:fois(exp exp1))
(defcons {exp}:divop(exp exp1))
(defcons {exp}:egal(exp exp1))
(defcons {exp}:different(exp exp1))
(defcons {exp}:supeg(exp exp1))
(defcons {exp}:infeg(exp exp1))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;expressions de dates;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defcons {exp}:infdate(ndate ndat1))
(defcons {exp}:diffdate(ndate ndat1))
(deftree ndate)
(deftree {ndate}:ndat1)
(defcons {ndate}:ndateop (val~mydate))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;les nombres habituels;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(deftree entier)
(deftree boolean)
(deftree aux)
(defcons {entier}:entierop (val~fix))
(defcons {boolean}:booleenop (val~booleen))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;traitement des booleens;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defcons {aux}:etop(exp expl))
(defcons {aux}:ouop(exp expl))
(defcons {aux}:deref(exp))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;traitement des tables;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(deftree table)
(defcons {table}:tableop(id entier liste-cpv))
(defcons {exp}:cardt(id))
(defcons {exp}:ajout(id nuple))
(deftree nuple)
(defcons {nuple}:nupleop(liste-const liste-constl))
(deftree liste-const)
(deftree {liste-const}:liste-constl)
(defcons {liste-const}:lconst sons~(List constu))
(defcons {exp}:adjcp(id idt nuple))
(defcons {exp}:adjcv(id idt nuple))
(defcons {exp}:dom(id idt))
(defcons {exp}:codom(id idt))
(defcons {exp}:app(id idt entier))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;expressions predicatives;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(deftree liste-variable)
(defcons {liste-variale}:lvariable sons~(List varcherch))
(deftree varcherch)
(defcons {varcherch}:expcle(liste-ident qualif))
(defcons {varcherch}:qualifdirect(id idt qualif))
(deftree qualif)
(defcons {qualif}:qualifop(id idt liste-cle))
(deftree liste-cle)
(defcons {liste-cle}:meta-liste-cle)
(defcons {liste-cle}:lcle sons~(List liste-ident))
(deftree exprech)
(defcons {exprech}:exprechop(liste-predicat))
(deftree liste-predicat)
(defcons {liste-predicat}:lpredicat sons~(List predicat))
(deftree predicat)
(defcons {predicat}:meta-predicat)
(defcons {predicat}:predicatop(liste-cle qualif suite-predic))
(deftree suite-predic)
(defcons {suite-predic}:meta-suite-predic)
(defcons {suite-predic}:suitepredicop(liste-predic))
(deftree liste-predic)
(defcons {liste-predic}:meta-liste-predic)
(defcons {liste-predic}:lpredic sons~(List predic))
(deftree predic)
(defcons {predic}:meta-predic)
(defcons {predic}:egalq(id qualifonc idt qualifoncl))
(deftree qualifonc)
(deftree {qualifonc}:qualifoncl)
(defcons {qualifonc}:qualifoncop(liste-cle))
(defcons {predic}:differq(qualifonc qualifoncl))
(defcons {predic}:supeq(qualifonc qualifoncl))
(defcons {predic}:infeq(qualifonc qualifoncl))
(defcons {predic}:infq(qualifonc qualifoncl))
(defcons {predic}:suppq(qualifonc qualifoncl))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;les unions;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(deftree qualifcompl)
(defcons {qualifcompl}:qualifcomplop(id qualif))
;(union-de-type toutid '(qualifcompl qualif id))
;(union-de-type qualifoncu '(qualifonc qualif id))
;(union-de-type constu '(entier boolean chaineres ndate))
;(union-de-type utype '(typebase table))
;(union-de-type expu '(exp toutid entier boolean aux))
```

ANNEXE 10

Quelques modules de recherche en table (réalisés par M. LECRAS)

```
;*****************************************************************
;*                                                               *
;*  procedures de recherche dans une liste                       *
;*                                                               *
;*****************************************************************



;RECH-ELEM-POS [-elem -lis p-rech]
;
;                recherche dans la liste <-lis> la position <p-rech> de l'element <p-rech>
;                  sert essentiellement a reperer la position d'un champ dans la liste des
;                  champs d'une table.
;                   si le champ est dans la cle (premiere composante de la liste), la
;                   position est augmentee de 100.
;
(de rech-elem-pos (-elem -lis p-rech)
  (cond ((listp (car -lis))
         (setq p-rech (rechelem -elem (car -lis) p-rech))
         (ifn (equal p-rech '0) (setq p-rech (+ p-rech '100))
           (setq p-rech (1+ (rechelem -elem (cdr -lis) p-rech)))
         )
        )
        (t
         (setq p-rech (rechelem -elem -lis p-rech))
        )
  )
);fin rech-elem-pos

;RECHER [nom-de-table l-champ-cle valeurs-cle l-champ resultat]
;
;         recherche dans la table <nom-de-table> de la valeur <resultat>
;          des champs donnes dans la liste <l-champ> suivant la cle d'acces
;          composee des champs de la liste <l-champ-cle> dont les valeurs
;          sont donnees dans la liste <valeurs-cle>.
;         s'il s'agit de la cle d'acces normale (premier element de la table),
;          <l-champ-cle> doit etre une liste vide.
;
(de recher (nom-de-table l-champ-cle valeurs-cle l-champ resultat)
  (setq pos '0)
  (setq pos (rechelcar nom-de-table systemtab pos))
  (if (equal pos '0) (print "table non existante")
    (setq table (eval nom-de-table))
    (setq champ-table (cadr (nth (1- pos) systemtab)))
    (setq liste nil)

    (setq liste (rech-res (car table) l-champ champ-table liste))
;
; on recherche dans le premier element de la table les elements demandes (l-champ)
; si la liste retournee n'est pas de meme longueur que la liste des champs demandes
; c'est qu'il y a incompatibilite dans la liste des champs demandes.
;

   (ifn (equal (length liste) (length l-champ))
        (print "la liste des champs demandes est incorrecte")
     (cond ((equal l-champ-cle nil)   ;recherche sur cle normale entiere

            (setq pos-table '0)
            (setq pos-table (rechelcar valeurs-cle table pos-table))
            (if (equal pos-table '0) (print "element non existant pour cette cle")
              (setq ligne (nth (1- pos-table) table))
              (setq resultat ())

              (setq resultat (rech-res ligne l-champ champ-table resultat))
;                    recherche des valeurs dans la ligne de la table
            )
           )
           ( t                        ;recherche sur cle normale partielle ou cle de champs

            (setq liste nil)
            (setq liste (rech-res (car table) l-champ-cle champ-table liste))
```

```
;
; recherche dans le premier element de la table de la valeurs des elements des champs
; de la cle (l-champ-cle). Si la longueur de la liste retournee est differente de celle
; des champs cles, c'est qu'il y a incompatibilite.
;
            (ifn (equal (length liste) (length l-champ-cle))
                 (print "la liste des champs cles est incorrecte.")
              (setq resultat nil)
              (setq trouve '0)
              (setq liste table)
              (until (or (equal trouve '1) (equal (car liste) nil))
                (setq l-val-cle nil)
                (setq l-val-cle (rech-res (car liste) l-champ-cle champ-table
                                                        l-val-cle))
                (cond ((equal l-val-cle valeurs-cle)
                       (setq resultat (rech-res (car liste) l-champ
                                                champ-table resultat))
                       (setq trouve '1)
                      )
                )
                (setq liste (cdr liste))
              )


              (ifn (equal trouve '1) (print "element non existant pour cette cle."))
             )
            )
      )
    )
  )
(setq resultat resultat)
);fin-recher
```

Modules de gestion des tables

```
;*************************************************************
;*                                                           *
;* Ensemble de procedures permettant de GERER DES TABLES     *
;*                                                           *
;*************************************************************


;(df while-list (--l --car . --corps)
;                  (while-list <syml> <sym> <s1>...<sn>)
;                     syml: symbole dont la valeur est une liste
;                     sym : recevra tous les car de la liste
;                     s1,...,sn: expressions evaluees
;
;                     equivalent a : (while <syml>
;                                      (setq <sym> (car <syml>))
;                                      (setq <syml> (cdr <syml>))
;                                      <s1> ... <sn>)
;
;                      la liste donnee n est pas modifiee
;
(df while-list (--l --car . --corps)
(cond ((not (variablep --l)) (syserror "while-list" "l'argument n'est pas une variable"
                                                                --l))
      ((not (variablep --car)) (syserror "while-list"
                                         "l'argument n'est pas une variable" --car))
     ((listp (eval --l)) (eval ['mapc (kwote (mcons 'lambda [--car] --corps)) --l]))
      (t (syserror "while-list" "la valeur de l'argument n'est pas une liste" --l))
);fin cond
);fin while-list
```

```
;(de creation-table (nom l-champ)
;                          ********************* creation d'une table
;                          nom: nom de la table
;                          l-champ: liste des champs de la table
;                                   la cle est constituee par le 1er champ qui
;                                    peut etre une liste ou un atome
;            ATTENTION:
;                      si l'on cree une table portant un nom deja existant,
;                        l'ancienne est perdue...
;
(de creation-table (nom l-champ)
  (setq pos '0)
  (setq pos (rechelcar nom systemtab pos))
  (ifn (equal pos '0) (supprim-table nom))
           ;suppression de la table
  (setq systemtab (cons (list nom l-champ) systemtab))
  (set nom '((fin)))
); fin creation-table


;(de rechelcar (el lis pos-rechelcar)
;                       ***********recherche sur premier element d une liste
;                       el : valeur de l element
;                       lis : liste donnee
;                       pos-rechelcar : position de l element dans la liste
(de rechelcar (el lis pos-rechelcar)
  (if (equal nil (car lis)) (setq pos-rechelcar '0)
    (if (equal el (caar lis)) (setq pos-rechelcar (1+ pos-rechelcar))
      (setq pos-rechelcar (1+ pos-rechelcar))
      (rechelcar el (cdr lis) pos-rechelcar)
)));fin rechelcar


;(de ajoutelem (nom-de-table liste-des-elts)
;                      *******ajout dune ligne dans une table
;                               ordre : LIFO
;                               cle ~ 1er elt de liste-des-elts
(de ajoutelem (nom-de-table liste-des-elts)
  (setq pos '0)
  (setq pos (rechelcar nom-de-table systemtab pos))
  (if (equal pos '0) (print "table non existante")
     set nom-de-table (cons liste-des-elts (eval nom-de-table)))
))
```

ANNEXE 11

Module principal du processeur de la dynamique

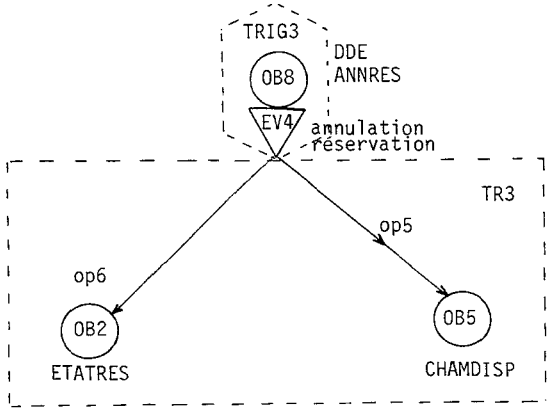
```
;*************************************************************************
;*                                                                       *
;*                    PROCESSEUR DE LA DYNAMIQUE                         *
;*                                                                       *
;*************************************************************************


(de processeur-dyn ()
  (init-var-globales)  ;initialisation a nil de toutes les  variables globales
;                        devant servir comme parametres.
;
  (while t                    ;le processeur est destine a "tourner" sans arret
    (exam-rep-objet)
;               examen de la table des reperes de cobjet
;               redonne la main lorsqu'il y a un cevenement associe
    (maj-var-globales champ-nuple-a nuple-a)  ;mise a jour des variables globales
;                                               qui servent en parametres.
    (setq base (sendq base schema ))
    (setq liste-even (sendq liste-evoper base))
;               examen de l'arbre du schema du systeme d'information afin de retrouver
;                  l'evenement associe parmi une liste d'evenements existants.
    (setq var-aux nil)
    (until (or (equal (nth n(sendq sons liste-even)) nil) (equal var-aux nom-sous-schema))
      (setq evenement (nth n(sendq sons liste-even)))
      (setq even-operation (sendq ev evenement))
      (setq var-aux (sendq id2 even-operation))
      (setq i n 1))
    )
    (ifn (equal var-aux nom-sous-schema) (print "pas d'evenement associe cette fois.")
;                                         on retourne  a l'examen des reperes de cobjet
;                                        car il n'y a pas d'evenement associe.
;    sinon : on enchaine sur la creation dun repere de cevenement.
      (setq nom-table (sendq idt even-operation))
      (setq table (car (eval nom-table)))   ;on prend le dernier element rentre
      (setq numero (1+ (car table)))
      (setq date (cadr table))
      (setq nom-even (sendq id evenement))
      (setq date-lue (lecture-date))
      (setq nuple-b (list nom-even numero date-lue nuple-a))
      (ajoutelem 'rep-even nuple-b)
      (ajoutelem nom-table (list numero date-lue))
;    creation d'un repere de transaction a declencher
      (setq nom-transac (sendq id4 even-operation))
;    recherche du nom du texte
      (setq nom-texte (sendq id1(sendq trans evenement)))
;    recherche du nom de la table des executions
      (setq nom-table (sendq idt(sendq trans evenement)))
      (setq table (car (eval nom-table)))  ;on prend le dernier element
      (setq numero (1+ (car table)))
      (setq date-lue (lecture-date))
      (ajoutelem 'rep-transac (list nom-transac numero nuple-b date-lue))
;    lancement du texte
      (setq liste-nuple nil)
      (print "execution de :") (print nom-texte)
      (setq nom-fich (catenate nom-texte ".ll"))
      (comline (catenate "cp " (string nom-fich) " fich-exec.ll"))
      (load fich-exec.ll)
      (apply nom-texte ())
```

```
;    le texte s'execute...
;                          rappelons que les parametres sont en variables globales.
;
;    creation d'un repere de transaction executee par element de liste-nuple.
      (print "reprise du fonctionnement du processeur")
      (setq date-finexec (lecture-date))
      (while-list liste-nuple nuple
        (ajoutelem 'rep-transac-executee (list nom-transac numero date-lue date-finexec
                                               nuple-b nuple))
        (ajoutelem 'rep-objet nuple)  ;creation d'un repere de cobjet
       )
;    valorisation de la table des executions
      (ajoutelem nom-table (list numero date-finexec))
;    valorisation de la table des declanchements
      (setq pos '0)
      (setq nom-table (concat "dec-" nom-even "-" nom-transac))
      (setq pos (rechelcar nom-table systemtab pos))
      (if (equal pos '0) (creation-table nom-table '(even transac date)))
      (ajoutelem nom-table (list nom-even nom-transac date-lue))
    );fin du if
        (print "pour laisser tourner le processeur, taper <go>")
        (setq --lu (read))  ;a enlever plus tard, ne sert qu'a la mise au point
  );fin du while
);fin processeur-dyn
```

ANNEXE 12

Déroulement du processeur de la dynamique sur un exemple

Nous présentons, ici, le déroulement de l'exécution sur le sous-ensemble de l'exemple des réservations présenté en Annexe 1, qui correspond à l'annulation d'une réservation.



Le texte de la transaction TR3, déclenché par le trigger TRIG3, est le regroupement des opérations op6 (annulation de la réservation) et op5 (restitution des chambres diponibles).

op5 est itérative ; le déclenchement de op5 et op6 est inconditionnel. Le texte de TR3 s'appelle "ANNULDEM", le texte du prédicat associé à TRIG3 se nomme "PREDIC".

occurrences de la méta-base

Les requètes SYNTEX suivantes et leurs réponses indiquent le contenu de la méta-base correspondant à l'exemple.

question:

RELATION(X)DU TYPE TRIGGER?

réponse:

nbre d'information:1
(il y a un trigger)

%trig3

question:

RELATION(X)DU TYPE COB?

réponse:

nbre d'information:3
(il y a trois c-objets)

%chamdisp
%etatres
%dde an res

question:

RELATION(X)DU TYPE TRANS?

réponse:

nbre d'information:1
(il y a une transaction)

%tr3

question:

RELATION(X)DU TYPE DECL?

réponse:

nbre d'information:1
(il y a un déclenche effectif)

%decl33

question:

COMPOSITION RELATION⟨X⟩⟨Y⟩⟨Z⟩⟨W⟩?

réponse:

nbre d'information:17
(la méta-base comporte 6 relations dont on donne la composition en constituants)

```
%decl33          (nom de la relation)
%datdecl33       (nom du constituant)
%3               (numéro du constituant)
%c0800           (type du constituant)

%decl33          (la relation decl33 a trois
%notr3           constituants :
%2               notrig3, notr3,datdecl33)
%e0400

%decl33
%notrig3
%1
%e0400

%tr3             %tr3
%datr3           %notr3
%2               %1
%c0800           %c0400

%trig3           %trig3           %trig3
%numres          %datrig3         %notrig3
%3               %2               %1
%c0400           %c0800           %e0400

%chamdisp        %chamdisp        %chamdisp
%dfindis         %ddebdis         %numch
%3               %2               %1
%c0800           %c0800           %e0400

%etatres         %etatres         %etatres
%etat            %datdem          %ndem
%3               %2               %1
%c0100           %c0800           %e0400

%dde an res      %dde an res      %dde an res
%raison          %datanul         %nres
%3               %2               %1
%c0800           %c0800           %e0400
```

question:

COB(X)DU TRIGGER(Y)?

réponse:

nbre d'information:1
(il y a un trigger, donc un c-objet associé)

%dde an res (c-objet)
%trig3 (trigger)

question:

TEXT(X)DU TRIGGER(Y)?

réponse:

nbre d'information:1
(il y a un texte de prédicat dans le SI)

%predic (nom du prédicat)
%trig3 (nom du trigger)

question:

TEXT(X)DE LA TRANS(Y)?

réponse:

nbre d'information:1
(il y a un texte de transaction dans le SI)

%annuldem (nom du texte)
%tr3 (nom de la transaction)

question:

TRIGGER(X)DECL(Y)TRANS(Z)?

réponse:

nbre d'information:1
(il y a un déclenche effectif dans le SI)

%trig3 (nom du trigger)
%decl33 (nom du déclenche)
%tr3 (nom de la transaction)

question:

```
COMPOSITION CLE%DDE AN RES%(Y)(Z)?

nbre d'information:2
(la cle de dde an res est composée de deux constituants)

%dde an res          (nom de la relation)
%numres              (nom de la clé)
%datanul             (nom du constituant)

%dde an res
%numres
%nres
```

création des relations dans le SI

relations sur des atomes:

Nous donnons ici les requètes SYNTEX.

```
CRE:TRIG3(X)(Y)(Z)(W),0,0,0,0!
(création de la relation trigger)

CRE:TR3(X)(Y)(Z),0,0,0!
(création de la relation transaction)

CRE:DECL33(X)(Y)(Z)(W),0,0,0,0!
(création de la relation déclenche effectif)
```

relations molécules:

Nous donnons également les requétes Syntex.

```
CRE:DDE AN RES(X)(Y),0,M,RDM:FIC1,(NRES,E)(DATANUL,C(08))_
                         (RAISON,C(08))!
(demande d'annulation de réservation)

CRE:CHAMBRES(X)(Y),0,M,RDM:FIC2,(NUMCH,E)(NRES,E)!
(chambres réservées)
```

```
CRE:ETATRES(X)(Y),O,M,RDM:FIC3,(NDEM,E)(DATDEM,C(08))_
                              (ETAT,E)!
(état de la réservation)

CRE:CHAMDISP(X)(Y),O,M,RDM:FIC4,(NUMCH,E)(DDEBDIS,C(08))_
                          (DFINDIS,C(08))!
(chambres disponibles)

CRE:RESERVATION(X)(Y),O,M,RDM:FIC5,(NRES,E)(DDEBRES,C(08)_
            )(DFINRES,C(08))(NUMHOT,E)(NCLI,E)!
(réservation de chambres)
```

les résultats

Nous commentons ici les différentes étapes de l'outil pas à pas en faisant apparaitre les questions transmises à SYNTEX sous forme d'arbres, les réponses obtenues, le lancement effectif des textes de trigger et de transaction enfin les mises à jour effectuées sur le SI par l'intermédiaire de SURSYNTEX.

PREMIERE ETAPE : recherche du trigger

A. recherche des caractèristiques du c-objet

question:

```
1 (EX.X,(COMPOSITION RELATION%DDE AN RES%(X)(Y)(Z)))?
2 %BID%
3 FIN*!
```

(la question est posée sous la forme d'un arbre "bidon" afin de récupérer la réponse dans un fichier; dans la suite nous ne préciserons plus tous les niveaux)

réponse:

3%C0800%2%C0800%1%E0400%
(numéro et type de chaque constituant)

B. Recherche du(des) trigger(s) associé(s) et de leur texte

question:

(COB%DDE AN RES%DU TRIGGER(X)) ET (TEXT(Y)DU TRIGGER(X))?

réponse:

%trig3%predic%
(nom du trigger et de son texte de prédicat)

lancement effectif du texte et récupération du résultat dans COND

lancement de OBPREDIC
on est dans obpredic
COND 0001

C. création du repère de trigger

En fait, la création d'une annulation de réservation déclenche toujours TR3 :

- mise à jour du premier bloc de repère de trigger (on ajoute un à INCLUS)

REP-TRIG 00010002

- recherche de la valeur de l'identifiant de l'occurrence précédente du trigger (en effet le système génère la nouvelle valeur en ajoutant un à l'ancienne).

question:

(EX.Y,EX.Z,(TRIG3%TRIG3%(X)(Y)(Z)))?

réponse:

%000103%000102%000101%000100%000099%
(il y a actuellement cinq occurrences de TRIG3 dans le SI)

après décodage du résultat, obtention du dernier identifiant

IDENT 000103

- création du repère de trigger, selon la forme précisée au paragraphe 4-4 partie III, avec 104 comme valeur d'identifiant.

nom du trigger/identifiant/date arrivée/nuple de c-objet
TRIG3 000104 220682085018 AN RES 0001008101

nuple(suite)/bloc repère c-objet
01MALADIE 0002

D. mise à jour du repère de c-objet

On ajoute un à TRAITE, on arrête d'analyser les repères de c-objet quand il y a égalité entre INCLUS et TRAITE.

REP-OB 00020002

DEUXIEME ETAPE : recherche de la transaction

A. création d'une occurrence de trigger dans le SI

question:

TRIG3%TRIG3%_
%000104%%220682085018%%0001000810101%!

(création d'une occurrence de la relation sur les atomes trig3)

B. recherche de la transaction et de la relation déclenche

question:

(TRIGGER%TRIG3%DECL(Y)TRANS(Z))?

réponse:

decl33%tr3%
(nom de la relation déclenche correspondant à trig3 et du texte de la transaction tr3)

C. création du repère de transaction à déclencher

- mise à jour du premier bloc de repère de transaction à déclencher : on ajoute un à INCLUS

REPTRANS 00010002

- recherche de la valeur de l'identifiant de la précédente occurrence de transaction

question:

(EX.Y,(TR3%TR3%(X)(Y)))?

réponse:

%000102%000101%000100%000099%
(il y actuellement quatre occurrences de tr3 dans le SI)

après décodage, obtention du dernier identifiant

IDENT 000102

- création du repère de transaction à déclencher

nom de la transaction/identifiant/recopie repère trigger
TR3 000103 TRIG3 0001042206820850
repère trigger(suite) /relation déclenche/bloc
18 DDE AN RES 000100810101MALADIE DECL33 220682085359 0002

D. mise à jour de repère de trigger traité

On ajoute un au premier bloc de repère de trigger : à TRAITE.

REPTRIG 00020002
(on arrête quand il y a égalité entre INCLUS et TRAITE)

TROISIEME ETAPE : déclenchement effectif de la transaction

A. recherche du texte de la transaction

question:

(TEXT(X)DE LA TRANS%TR3%)?

réponse:

%annuldem%
(nom du texte)

B. lancement effectif

Le texte de la transaction est la concatènation de plusieurs étapes :

1- recherche des dates de réservation et des numéros de chambres à rendre disponibles;
2- recopie du nuple de c-objet ayant changé d'état;
3- restitution des chambres devenues disponibles avec calcul de l'intervalle de disponibilité;
4- passage de l'état de la réservation à "annulé".

1- Supposons que l'on annule la réservation numéro 100 :

- lancement du texte de la transaction "annuldem":

lancement de OBANNULDEM
on est dans demannu

- recherche de la réservation 100 et stockage des dates de réservation :

question:

(RESERVATION%RESERVATION%(X)) ET (X.NRES=100)?

réponse:

%%100%19810101%19810115%1020%10%

- décodage des dates :

DAT1 19810101
DAT2 19810115

- recherche des chambres associées à la réservation :

question:

(CHAMBRES%CHAMBRES%(X)) ET (X.NRES=100)?

réponse:

%%100%100%%%101%100%%%102%100%%%103%100%%%
(couples numéro de réservation, numéro de chambre)

- après décodage, constitution du fichier des chambres à libérer :

NUM 100 ELEMENT 1001981010119810115
NUM 101 ELEMENT 1011981010119810115
NUM 102 ELEMENT 1021981010119810115
NUM 103 ELEMENT 1031981010119810115

2- recopie demande en entrée :

FDEM 000100810101MALADIE

3- on cherche, pour chaque chambre libérée, si un intervalle de disponibilité jouxte à gauche puis à droite:

question:

(CHAMDISP%CHAMDISP%(X)) ET (X.DFINDIS=%19810101%) ET_
(X.NUMCH=100)?

réponse:

DOMAINE VIDE
(il n'y en a pas à gauche)

question:

(CHAMDISP%CHAMDISP%(X)) ET (X.DDEBDIS=%19810115%) ET_
(X.NUMCH=100)?

réponse:

%%100%19810115%19810201%
(il y en a une à droite)

- on supprime l'ancienne disponibilité :

question:

```
(CHAMDISP%CHAMDISP%(X))      ET      (X.NUMCH=100)     ET_
(X.DDEBDIS=%19810115%) ET (X.DFINDIS=%19810201%)//
```

- on crée la nouvelle :

question:

```
CHAMDISP%CHAMDISP%(X):101,%19810101%,%19810201%!
```

- on recommence avec la chambre suivante libérée :

question:

```
(CHAMDISP%CHAMDISP%(X))   ET   (X.DFINDIS=%19810101%)   ET_
(X.NUMCH=101)?
```

réponse:

DOMAINE VIDE
(il n'y en a pas à gauche)

question:

```
(CHAMDISP%CHAMDISP%(X))   ET   (DDEBDIS=%19810115%)    ET_
(X.NUMCH=101)?
```

réponse:

DOMAINE VIDE
(il n'y en a pas non plus à droite)

- on recommence avec la nouvelle chambre libérée :

question:

```
(CHAMDISP%CHAMDISP%(X))   ET   (X.DFINDIS=19810101%)   ET_
(X.NUMCH=102)?
```

réponse:

%%102%19801215%19810101%
(il y en a une qui jouxte à gauche)

- on supprime l'ancienne disponibilité

question:

```
(CHAMDISP%CHAMDISP%(X))      ET      (X.NUMCH=102)      ET_
(X.DDEBDIS=%19801215%) ET (X.DFINDIS=%19810101%)//
```

- on crée la nouvelle

question:

```
CHAMDISP%CHAMDISP%(X):102,%19801215%,19810115%!
```

- on recommence avec la dernière :

question:

```
(CHAMDISP%CHAMDISP%(X))  ET   (X.DFINDIS=%19810115%)   ET_
(X.NUMCH=103)?
```

réponse:

DOMAINE VIDE

question:

```
(CHAMDISP%CHAMDISP%(X))   ET   (X.DDEBDIS=%19810101%)   ET_
(X.NUMCH=103)?
```

réponse:

DOMAINE VIDE

4- on fait passer la réservation à l'état "annulée" :

- on la cherche :

question:

```
(ETATRES%ETATRES%(Y)) ET (Y.NDEM=100)?
```

réponse:

%%100%19800115%1%
(numéro, date, état de la réservation)

- on la supprime puis on la recrée :

questions:

```
(ETATRES%ETATRES%(Y)) ET (Y.NDEM=100) ET (Y.DATDEM=_
%19800115%) ET (Y.ETAT=1)//

ETATRES%ETATRES%(X):100,%19800115%,2!
```

C. création des repères de transaction exécutée :

Pour chaque transaction on crèe autant de repères qu'il y a de nuples modifiés par la transaction. Les résultats se présentent ainsi :

- le nuple de c-objet retourné par la transaction,
- le premier bloc de repère de transaction exécutée mis à jour (on ajoute un à INCLUS),
- le repère de transaction exécutée créé.

premier nuple:

```
            identificateur/ nuple
ENRNUPLE    CHAMBDISP    1001981010119810201
```

premier bloc:

```
REPTRANSEX 00010002
```

repère créé:

```
nom transaction/ident./date exec./trigger déclencheur
   TR3        000103 820622090016 TRIG3 00104 820622
déclenche(suite)/bloc
085359 DECL33     0002
```

deuxième nuple:

```
ENRNUPLE CHAMDISP 1011981010119810115

REPTRANSEX 00010003

   TR3        000103 820622090016 TRIG3 00104 820622
085359 DECL33      0002
```

troisieme nuple:

ENRNUPLE CHAMDISP 1021980121519810115

REPTRANSEX 00010004

TR3 000103 820122090016 TRIG3 00104 820622 085359 DECL33 0002

quatrième nuple:

ENRNUPLE CHAMDISP 1031981010119810115

REPTRANSEX 00010005

TR3 000103 820122090016 TRIG3 00104 820622 085359 DECL33 0002

cinquième nuple:

ENRNUPLE ETATRES 100198001152

REPTRANSEX 00010006

TR3 000103 820122090016 TRIG3 00104 820622 085359 DECL33 0002

D. mise à jour du repère de transaction à déclencher

On ajoute un à TRAITE.

DESTRANSDEC 00020002

QUATRIEME ETAPE : actualisation du SI

Pour chaque transaction exécutée on réalise :

A. création d'une occurrence de transaction dans le SI question:

TR3%TR3%%000103%%820622090016%!

B. création d'une occurrence de déclenche dans le SI question:

DECL33%DECL33%%000104%%000103%%085359%!

C. mise à jour des repères de c-objets

On effectue la mise à jour du premier bloc de repère de c-objet (on ajoute un à INCLUS) et la création d'un repère de c-objet pour chaque nuple communiqué par la transaction. On met à jour le repère de transaction exécutée (on ajoute un à TRAITE).

REPOB 00020003

identificateur/attributs
CHAMDISP 1001981010119810201

DESTRANSEX 00020006

REPOB 00020004

CHAMDISP 1011981010119810115

DESTRANSEX 00030006

REPOB 00020005

CHAMDISP 1021980121519810115

DESTRANSEX 00040006

REPOB 00020006

CHAMDISP 1031981010119810115

DESTRANSEX 00050006

REPOB 00020007

ETATRES 100198001152

DESTRANSEX 00060006

on s'arrête car il y a égalité entre TRAITE et INCLUS.

CINQUIEME ETAPE : première étape

Pour chaque nuple de c-objet associé à un repère :

A. on cherche les caractèristiques du c-objet

question:

(EX. X(COMPOSITION RELATION%CHAMDISP%(X) (Y) (Z)))?

réponse:

%3%C0800%2%C0800%1%E0400%

B. on cherche le trigger éventuel associé

Ici il n'y en a pas.

question:

(COB%CHAMDISP%DU TRIGGER(X) ET (TEXT(Y) DU TRIGGER(X)))?

réponse:

DOMAINE VIDE

C. mise à jour du premier bloc de repère de c-objet

On ajoute un à TRAITE.

On recommence ensuite pour chaque occurrence de c-objet du repère, à chaque étape il y a mise à jour du premier bloc de repère de c-objet associé au c-objet CHAMDISP.

REPOB 00040007
REPOB 00050007
REPOB 00060007

D. mèmes étapes pour ETATRES

question:

(EX. X(COMPOSITION RELATION%ETATRES%(X) (Y) (Z)))?

réponse:

%3%E0100%2%C0800%1%E0400%

question:

(COB%ETATRES%DU TRIGGER(X)) ET (TEXT(Y)DU TRIGGER(X))?

réponse:

DOMAINE VIDE

Le dernier repère de c-objet inclus a été traité et aucun repère de trigger n'a été créé; le traitement s'arrête.

REPOB 00070007

FIN PROCESSEUR

BIBLIOGRAPHIE

BIBLIOGRAPHIE LIEE AU PROJET REMORA

BAN79 BANON D. "Projet REMORA : un outil pour la gestion des Systemes d' Informations" These 3eme cycle Nancy Mai 1979.

FOU82 FOUCAUT O. "Modele et outil pour la conception des Systemes d' Informations dans les Organisations, projet REMORA" These d' etat Nancy 1 CRIN 1982.

KEB84 KEBAILI A. "Une contribution a l'etude du probleme de l'evolution de structure dans les Systemes d'Information historiques" These 3eme cycle Nancy 1 CRIN Juin 1984.

KRI78 KRIEGUER M. "Modele de representation et Methode de construction des composantes d' un S.I. : projet REMORA" These 3eme cycle Nancy 1978.

LAMY77 LAMY D. "Les traitements, leurs structures et leur generation dans le projet REMORA" These 3eme cycle Nancy 1977.

LEI80 LEIFERT S. "Un langage de Specification des Systemes d'Information un outil pour leur Gestion" These Docteur Ingenieur Nancy 1980.

LEC84 LE CRAS M. "Conception du processeur de la dynamique d' un S.I. : une realisation en LISP-CEYX" Rapport de DEA CRIN Nancy 1 1984.

PER76 PEREA-VILLACORTA W. "Methode d' analyse dans le projet REMORA" These 3eme cycle Nancy 1976.

RIC79 RICHARD C. "Une approche conceptuelle des problemes de synchronisation" These 3eme cycle Nancy 1979.

ROL78 ROLLAND C., FOUCAUT O. "Concepts for design of an Information System Conceptual Schema and its utilisation in the REMORA project" Proc. of VLDB Berlin 1978.

ROL79a ROLLAND C., FOUCAUT O. "Dynamic dimension in Data Models" Colloque INRIA/ACM sur les Bases de Donnees : modeles, mise en oeuvre, evaluation Paris 197

ROL79b ROLLAND C., LEIFERT S., RICHARD C. "Tools for Information System Dynamics Management" Proc. of VLDB Rio de Janeiro 1979.

ROL79c ROLLAND C., FOUCAUT O., RICHARD C., THIERY O. "Information System Design and Information System Computer-aided Design" Proc. of EURO-IFIP Londres 1979.

ROL80a ROLLAND C., THIERY O. "Proposition for an Information System Computer-aided Design Software" Proc. of ADM Congress Sorento 1980.

ROL80b ROLLAND C., LEIFERT S., RICHARD C. "A proposal for Information System Design and Management" ACM SIGDA Aout 1980.

ROL81a ROLLAND C., THIERY O. "Constructions de Base d' un langage de Specification de Systemes d' Information" Proc. Congres Bases de Donnees Tunis 1981.

ROL81b ROLLAND C., THIERY O. "Une machine-systeme d' information" Convention Informatique Latine Barcelone 1981.

ROL82 ROLLAND C., RICHARD C. "The REMORA Methodology for I.S. Design and Management" Conference CRIS1 IFI82.

THI76 THIERY O. "L' aide a la conception dans le projet REMORA" These 3eme cycle Nancy 1976.

THI82 THIERY O. "Conception et realisation d' un processeur de la dynamique" Rapport CRIN 1982.

BIBLIOGRAPHIE GENERALE

ABR77 ABRIAL J.R. "Utilisation du langage Z pour l'analyse d'une petite application de gestion" Rapport 1977.

ABR78 ABRIAL J.R. "A specification language Z" Seminaire de HAUTE-NENDAZ Mars 1978 .

ABR80 ABRIAL J.R.,SCHUMANN S.A.,MEYER B."Specification language" Proc. Summer School on the construction of programs BELFORT, Cambridge University Press 1980.

ADA80 Department of Defense "Reference Manual for the Ada Programming Language" Nov. 1980.

ADI85 ADIBA M. "Notion de temps dans les Bases de Donnees generalisees" Journees Bases de Donnees Avancees Saint Pierre de Chartreuse Mars 1985.

ADJ78 GOGUEN J.A. et al. "An initial Algebra approach to the specification, correctness and implementation of Abstract Data Types" Current trends in Programming Methodology vol.3 1978.

AMB77 AMBLER A.L. et al. "GYPSY: A Language for Specification and Implementation of verifiable Programs" ACM SIGPLAN 12(3),Mars 1977.

AND84 ANDRE E. et al. "Vers un atelier flexible et integre de logiciel: le projet CONCERTO" L'echo de recherches num. 115, 1984.

ANS75 ANSI/X3/SPARC "Interim Report from the Study Group on Database Management Systems" Bulletin of ACM 7(2) 1975.

ANT81 DE ANTONELLIS V., ZONTA B. "Modelling Events in Database Application Design" Proc. congres VLDB CANNES 1981.

ASH82 ASCHIM F., MOSTUE B.M. "IFIP WG 8.1 Case solved using Sysdoc and Systemator" Conference CRIS 1 IFI82.

AST75 ASTRAHAN M.M., CHAMBERLIN D.D. "Implementation of a structured E English Query Language" CACM 18(10) 1975.

AST76 ASTRAHAN M.M. et al. "System R : a Relational Approach to Databases Management" ACM TODS 1(2) 1976.

BAC78 BACKUS J. "Can programming be liberated from the Von Neumann styles? A fonctional style and its algebra of programs" CACM Aout 1978.

BAU78 BAUER F.L., PEPPER P., WOSSNER H."Notes on the project CIP: outline of a transformation System" Program Construction BAUER, BROY eds. Lecture Notes in Computer Science 69, Springer Verlag 1978.

BEL78 BELLEGARDE F. et al. "MEDEE : a type of Language for the deductive Programming Method" Conference on Reliable Software German ACM chapter BONN 1978.

BELO3 BELAID A. "SAFIN : Systeme d' Aide a la Programmation du traitement d' Images Numerisees" Rapport CRIN 83-R-032.

BEN76 BENCI G. ,BODART F. ,BOUAERT H., CABANES A. "Concepts for the Design of a Concertual Schema" Proc. of IFIP TC2 WC Freudenstadt 1976.

BEN79 BENCI G., ROLLAND C. "Bases de donnees, conception canonique pour une realisation extensible" SCM Paris 1979.

BID81 BIDOIT M. "Une methode de presentation des types abstraits, applications" These troisieme cycle Paris Sud 1981.

BIGRE80 Actes "Environements, Supports et Systemes de programmation" Rennes (IRISA) Dec. 1980.

BIGRE82 Actes "Systemes integres de production de logiciels" Grenoble (IRISA) Jan. 1982.

BOD79 BODART F., PIGNEUR Y. "A Model and a Language for fonctional Specifications and evaluation of Information System Dynamics" IFIP TC8 Conference on Formal Models and practical Tools for Information System Design, North Holland 1979.

BOD83 BODART F. et al. "Evaluation of CRIS1 I.S. Developments Methods using a three cycles Framework" Conference CRIS2 IFI83.

BOD84 BODART F., PIGNEUR Y. "IDA, une methode de conception assistee des S.I." Conference CRIS3 IFI84.

BOI83 BOISSON F. et al. "Algebres a operateurs multicibles" Rapport LRI Universite de Paris Sud Juin 1983.

BOU85 BOURGEOIS J. "Ateliers de genie logiciel : etat de l'art et perspectives." GENIE85

BOUJ84 BOUJLIDA N. "SINDBAD un systeme d'aide a la specification et a l'utilisation des Bases de Donnees deductives" These de docteur ingenieur INPL CRIN 1984.

BOUZ85 BOUZELHOUB M., GARDARIN G. "SECSI : un outil interactif de modelisation conceptuelle des Bases de Donnees" Journees Bases de Donnees Avancees Saint Pierre de Chartreuse Mars 1985.

BOY74 BOYCE R.F. et al. "Specifying queries as relational expressions:SQUARE" Proc. IFIP TC2 WC North Holland 1974.

BRA76 BRACCHI G. et al. "Binary logical associations in data modelling" Proc. IFIP TC2 WC Freudenstadt 1976.

BRE79 BREUTMANN C., FALKENBERG E., MAUER R. "CSL:a language for defining Conceptual Schemas" IFIP Working Conference on Database Architecture Venise 1979.

BRI75 BRINCH HANSEN P. "The Programming Language Concurrent Pascal" IEEE Transaction Software Engineering , 1975.

BRO80 BRODIE M.L. "Data Abstraction, Databases and Conceptual Modelling" Proc. VLDB 1980.

BRO81 BRODIE M.L., ZILLES A.N.(Eds.):Proc. Workshop on Data Abstraction Databases and Conceptual Modelling SIGPLAN Notices 16(1) 1981.

BRO82 BRODIE M.L., SILVA E. "Active and Passive Component Modelling:ACM/PCM" Conference CRIS1 IFI82.

BRO83A BRODIE M.L. et al. "On a framework for Information Systems Design Methodologies" Conference CRIS2 IFI83.

BRO83B BRODIE M.L. "Research Issues in Data Base Specification SIGMOD Record Avril 1983.

BROY78 BROY M., GNATZ R., WIRSING M. "Semantics of Nondeterministic Noncontinuous Constructs,Program Construction" Lecture Notes in Computer Science 69, Springer Verlag 1978.

BROY82 BROY M., WIRSING M. "Partial Abstract Types" Acta Informatica 18(1) Nov. 1982.

BUB77 BUBENKO J.A. "The Temporal Dimension in Information Modelling" Proc. IFIP TC2 WC Nice 1977.

BUB83 BUBENKO J.A. "Information and Data Modelling : State of the Art and Research Directions" SYSLAB Report 20 Avril 1983.

CAV79 CAVARERO J.L. "LAPAGE : un modele et un outil d'aide a la conception des SI" These d'etat Nice 1979.

CHA74 CHAMBERLIN D.D., BOYCE R.F. "SEQUEL: a structured English Query Language" Proc. ACM SIGMOD Workshop on Data Description, Access and Control 1974.

CHA82 CHABRIER J.J. "Specification des systemes orientes Bases de Donnees : techniques et langages bases sur le concept de Types Abstraits. These d'etat Universite de Nancy 1 CRIN 1982.

CHA83 CHABRIER J.J., CHABRIER J. "VEGA : Un systeme interactif de specification algebrique avec erreurs" Rapport CRIN 83-R-036.

CHE76 CHEN P. "The Entity-Relationship Model towards a Unified View of Data" ACM TODS vol.1 Mars 1976.

CHR85 CHRISMENT C., ZURFLUH C. "Bases d'Informations generalisees : projet BIG modele agregatif, langage de manipulation et interface multi-media" Journees Bases de Donnees Avancees Saint Pierre de Chartreuse Mars 1985.

COD70 CODD E.F. "A Relational Model of Data for Large Shared Data Banks" CACM 13(6) 1970.

COD79 CODD E.F. "Extending the Data Base Relational Model to capture more meaning" ACM TODS 4(4) Dec. 1979.

CODA71 CODASYL DBTG "Report on the CODASYL Data Base Task Group" ACM New York 1971.

CRE75 CREHANGE M. "Description formelle, Representation, Interrogation des informations complexes : Systeme PIVOINES" these d'etat Nancy 1 Dec. 1975.

CRE83 CREHANGE M. et al. "EXPRIM : An expert system to aid in progressive retrieval from a pictorial and descriptive Database" Special Workshop on new Application of Databases ICOD-2 Cambridge 1983.

DAH70 DAHL O.et al. "SIMULA 67 Common Base Language" NCC 1970.

DAT77 DATE C.J. "An introduction to Data Base Systems" Addison Wesley 1977.

DEL73 DELOBEL C. "Contributions theoriques a la conception et a l' evaluation d' un systeme d' information appliqué a la gestion" These d' etat Grenoble 1973.

DEL82 DELOBEL C., ADIBA M. "Bases de donnees et systemes Relationnels" Dunod Informatique 1982.

DEM75 DEMOLOMBE R. et al. "SYNTEX 2" Rapport DRME 74222 CERT Toulouse 1975.

DEM85 DEMOLOMBE R. "STREL : une extension du modele relationnel pour representer et manipuler les objets structures" Journees Bases de Donnees Avancees Saint Pierre de Chartreuse Mars 1985.

DER74 DERNIAME J.C. "Le projet CIVA, un systeme de gestion modulaire". These d'etat Universite Nancy 1 1974.

DER79 DERNIAME J.C., FINANCE J.P. "Types Abstraits de Donnees : Specification Utilisation et Realisation" Cours de l'ecole d'ete de l'AFCET Monastir Rapport CRIN 79-E-57 1979.

DIJ76 DIJKSTRA E.W. "A discipline of programming" Prentice Hall 1976.

DON75 DONZEAU-GOUGE et al. "A structure oriented Program Editor : a first step towards computer-assisted programming" International Computing Symposium North Holland Publication 1975.

DOS82 DOSH W., MASCARI G., WIRSING M. "On Algebraic Specification of Databases" Proc. of VLDB Mexico 1982.

DUB84 DUBOIS E. "Cadre et methode de specification de S.I. fondes sur les types de donnees" These Docteur Ingenieur Nancy 1 CRIN Avril 1984.

DUF80 DUFOURD J.F. "Maquettes pour evaluer les systemes d'information des organisations" These d'etat Nancy 1 CRIN 1980.

EST83 ESTUBIER J. et al. "Design principles of ADELE Programming Environment" Proc. of International Computing Symposium on Application Systems Development Nuremberg 1983.

FAL83 FALKENBERG E. et al. "Feature Analysis of ACM/PCM, CIAM, ISAC and NIAM" Conference CRIS2 IFIP83.

FIN79 FINANCE J.P."Etude de la construction de programmes : methodes et langages de specification et de resolution de problemes" These d'etat Nancy 1 1979.

FIN83 FINANCE J.P. et al. "SPES : un systeme pour specifier et transformer" Rapport CRIN 83-R-079 1983.

FIS84 FISHER G., SCHNEIDER M. "Knowledge-based Communication Processes in Software Engineering" 7th Conference on Software Engineering Orlando Mars 1984.

FL077 FLORY A. "Un modele et une methode pour la conception logique d' une Base de Donnees" These d' etat Lyon 1977.

FL085 FLORES B., ROLLAND C. "OICSI un outil intelligent pour aider la conception des SI" Journees Bases de Donnees Avancees Saint Pierre de Chartreuse Mars 1985.

FOR67 FORTET R., LE BOULANGER H. "Elements pour une synthese sur les systemes a auto-organisation" Revue Metra num. 12 1967.

FUR83 FURTADO A.L. "An informal approach to formal Specifications" SIGMOD Record Avril 1983.

GAL78 GALLAIRE H., MINKER J. "Logic and Databases" Plenum 1978.

GAM77 Groupe GAMMA "L' ensemble ARIANE d' aide a l' analyse-programmation" Paris 1977.

GAN79 GANE C.,SARSON T. "Structured Systems Analysis" Englewood Cliffs NJ : Prentice Hall 1979.

GAN82 GANGE D., DAYAL U., BROWNE J.C. "Semantics of network data manipulation languages : an object-oriented Approach" VLDB Mexico 1982.

GAU79 GAUDEL M.C. "Specifications incompletes mais suffisantes de la representation des Types Abstraits" Groplan AFCET num. 7 Mai 1979.

GAU80 GAUDEL M.C. "Generation et preuve de compilateurs basees sur une semantique formelle des langages de programmation" These d'etat Paris VI 1980.

GENIE85 Revue Genie Logiciel numero 1 Publication ADI.

GHM78 GUTTAG J.A. et al. "Abstract Data Types and Sofware Validation" CACM vol. 21 Dec. 1978.

GIR85 GIRAUDIN J.P., DELOBEL C., DARDAILLER P. "Elements de construction d'un Systeme Expert pour la modelisation progressive d'une Base de Donnees" Journees Bases de Donnees Avancees Saint Pierre de Chartreuse Mars 1985.

GOG77 GOGUEN J.A. et al. "Initial Algebras Semantics and Continuous Algebras" Jour. ACM Mars 1977.

GOG79 GOGUEN J.A., TARDO J.J. "An Introduction to OBJ : a Language for writing and testing Formal Algebraic Programm Specifications" Proc. of Specification of Reliable Software IEEE Cat. 79 1979.

GOL83 GOLDBERG A., ROBSON D. "Smalltalk 80 : the Language and its implementation" Addison Wesley 1983.

GTW78 GOGUEN J.A. et al. "An initial Approach to the Specification, Correctness and Implementation of Abstract Data Types" In Current Trends in Programming Methodology vol.3 Data Structuring Prentice Hall.

GUS82 GUSTAFSSON M.R., KARLSSON T., BUBENKO J.R. "A declarative approach to Conceptual Information Modeling" Conference CRIS1 IFI82.

GUT78 GUTTAG J.V., HORNING J.J. "The Algebraic Specification of Abstract Data Types" Acta Informatica 10 1978.

GUT80 GUTTAG J.V. "Notes on Types Abstraction" IEEE Trans. on Software Engineering vol.6 1980.

GUY84 GUYARD J., JACQUOT J.P. "MAIDAY : an environment for guided Programming with a definitional Language" 7th. Conference on software Engineering Orlando Floride USA Mars 1984.

HEN80 HENRI P."La connexion dans le projet TYP" These 3eme cycle Nancy 1 CRIN 1980.

HER74 HERTSCHUH N. "L' analyse dans le projet CIVA" These 3eme cycle Nancy 1 1974.

HER84 HERTSCHUH N., BERGER P. "MAPPER, un exemple de "langage" nouvelle generation" TSI 3(2) MARS 1984.

HIB81 HIBBARD P. et al. "Studies in ADA style" Springer Verlag 1981.

HOA72 HOARE C. "Proof of Correctness of Data Representations" Acta Infomatica vol.1 1972.

HSU79 HSU J., ROUSSOPOULOS N. "Database Conceptual Modelling" Proc. of International Conference on the E-R Approach to Systems Analysis and Design Chen(ed.) North Holland 1979.

HUC78 HUC B. "Mise en oeuvre de la methode de la programmation deductive" These 3eme cycle Nancy 1 1978.

HUL83 HULLOT J.M. "CEYX a multiformalism programming Environment" Proc. IFIP Paris 1983.

HUL84 HULLOT J.M. "Programmer en CEYX. CEYX Version 15" Publication INRIA Ete 1984.

IFI82 IFIP WG 8.1 Working Conference on "Comparative review of Infomation Systems Design Methodologies" Noordwijkerhout Pays Bas North Holland Publication 1982.

IFI83 IFIP WG 8.1 Working Conference on "Information Systems Design Methodologies : a feature Analysis" York North Holland Publication 1983.

IFI84 "Concevoir vos S.I., une selection internationnale de methodes de conception : le choix de l' IFIP" Paris Octobre 1984.

INF75-85
Colloques et Seminaires: publications INRIA :
- Pont-a-Mousson "Langages d'analyse et de conception des S.I. automatises" 1975.
- St Pierre de Chartreuse "Representation des S.I." 1975.
- Caen "Recherche en Informatique d' organisation" 1976.
- Cergy Pontoise "Dynamique des S.I." 1978.
publication CETE :
- Aix en Provence "Implementation de la dynamique dans les S.I." 1980.
- Ajjaccio 1983.
- Bandol 1984.
- Luchon "Bases de Donnees Relationnelles" 1985.

ISO81 ISO TC 97/SC5/WG3 "Preliminary Report Concepts and Terminology for the Conceptual Schema" JJ Van Griethuysen (ed.) fevrier 1981.

JAC84 JACQUOT J.P. "Etude d' un outil d' assistance a la conception methodique de programmes: modelisation et evaluation d' une maquette" These Docteur Ingenieur CRIN Nancy 1984.

JOH84 JOHSON L., SOLOWAY E. "PROUST : knowledge-based Program Understanding" 7th Conference on Software Engineering Orlando Mars 1984.

KEN75 KENNEDY K., SCHWARTZ J. "An introduction to the set Theory Language SETL" Computers and Mathematics with Applications 1(97) 1975.

KEN77 KENT W. "Entities and Relationships in Information" Proc. of IFIP TC2 Nice 1977.

KNU70 KNUTH D.E., BENDIX P.B. "Simple word problems in universal Algebras" Computational problems in abstract Algebras J. Leech (ed.) Pergammon Press 1970.

KNU80 KNUTH E., RADO P. "Principles of computer-aided System description" Studies 117/1981 Computer and Automation Institute Hungarian Academy of Sciences 1980.

KNU82 KNUTH E. et al."SDLA, System Descriptor and Logical Analyser" Conference CRIS1 IFI82.

KOU85 KOULOUMDJIAN J., LEBAUBE P., LEPAPE B. "Systeme de traitement avance de l'information base sur la programmation en logique." Journees Bases de Donnees Avancees Saint Pierre de Chartreuse Mars 1985.

KOW81 KOWALSKI R. "Logic as a Database Language" Report Dept. of Computing Imperial College London 1981.

KRA82 KRAKOWIAK S. "Systemes integres de production de logiciel : concepts et realisations" TSI 1(3) 1982.

LAM77 LAMPSON B.N. et al. "Report on the Programming Language EUCLID" ACM SIGPLAN 12(2) Fevrier 1982.

LAN63 LANGEFORS B. "Some Approaches to the theory of Information Systems" BIT,3,1963 pp 229-254.

LAU76 LAURIERE J.L. "Un langage et un programme pour resoudre des problemes combinatoires" These d' etat Paris VI 1976.

LE 84 LE F., TARDIEU H., TABOURIER Y. "La methode MERISE" Conference CRIS3 IFI84.

LEO81 LEONARD M., LUONG B.T. "Information System Approach integrating Data and Transactions" Proc. VLDB Cannes 1981.

LES79 LESCANNE P. "Etude algebrique et relationnelle des Types Abstraits et de leurs representations" INPL CRIN 1979.

LEV80 LEVESON N.G. "Applying Behavior Abstraction to Information System Design and Integrity" Thesis University of Los Angeles 1980.

LEV84 LEVY N. "Methode et outils d' aide a la transformation de Types Abstraits Algebriques" These 3eme cycle CRIN Nancy 1984.

LIN79 LINDGREEN P. "Event directed program execution from declarative Specifications" Proc. of IFIP Oxford 1979.

LIS77 LISKOV B. et al. "Abstraction Mechanisms in CLU" CACM 20(8) Aout 1977.

LIV78 LIVERCY C. "Theorie des programmes" DUNOD 1978.

LON82 LONCHAMP J.,DUFOURD J.F., CREHANGE M. "Rapport annuel d'activites de l' equipe ARA" Rapport CRIN Janv. 1982.

LUN82A LUNDBERG M. "The ISAC Approach to Specification of Information Systems and its Application to the organization of an IFIP Working Conference" Conference CRIS1 IFI82.

LUN82B LUNDBERG M. "An Analysis of the Concept of Event" Syslab Working Paper num. 58 Nov. 1982.

LUN83 LUNDBERG M. "Some comments on the ISAC Approach in connection with the CRIS2 papers" Conference CRIS2 IFI83.

MAC82 MACDONALD I.G., PALMER I.R. "System Development in a shared Data Environment, the D2S2 Methodology" Conference CRIS1 IFI82.

MAD85 MADELAINE J. "Deduction dans la machine Base de Donnees SABRE" Journees Bases de Donnees Avancees Saint Pierre de Chartreuse Mars 1985.

MAR79 DE MARCO T. "Structured Analysis and System Specification" Englewood Cliffs, NJ : Prentice-Hall 1979.

MED82 MEDINA-MORA R. "Syntax-directed Editing : towards Integrating Program Environments" Ph. D. Thesis Carnegie Mellon University Dept. of Computer Sciences 1982.

MIN79 MINOT R. "ATM : un systeme de fabrication de programmes base sur les concepts de modularite et de Types Abstraits" These 3eme cycle Nancy 1979.

MIN84 "Premieres specifications du systeme EMERAUDE" Document BULL 1984.

NAS83 NASSIF R. "Projet ADONIS : conception et realisation d un systeme de gestion de multibases de donnees" These de Docteur Ingenieur Nancy CRIN 1983.

NGU85 NGUYEN G.T., OLIVARES J. "L'intégrite semantique dans une Base de Donnees Generalisees" Journees Bases de donnees Avancees Saint Pierre de Chartreuse Mars 1985.

NIJ77 NIJSSEN G.M. "Current issues in Conceptual Schema Concepts" Proc. of IFIP TC2 WC Nice 1977.

NIS83 NISSEN H.E. "Subject Matter Separability in Information Systems Design Methodes" Conference CRIS2 IFI83.

OLI82 OLIVE A. "DADES : a Methodology for Specification and Design of I.S." Conference CRIS1 IFI82.

OLI83 OLIVE A. "Analysis of conceptual and logical Models in I.S. Design Methodologies" Conference CRIS2 IFI83.

PAI79 PAIR C. "La construction de programmes" RAIRO Informatique 15(2) 1979.

PET77 PETERSON J.L. "Petri Nets" ACM Computing Surveys 9(3) 1977.

PET80 PETRI C.A. "Introduction to general Net Theory" Lecture Notes in Computer Sciences num.84 Springer Verlag 1980.

PIR76 PIROTTE A. "Explicit description of entities and their manipulation in languages for the relational Databases Model" These universite libre de Bruxelles 1976.

POR83 PORCELLA M. et al. "ADA Methodology Questionnaire Summary" ACM SIGSOFT 8(1) Janv. 1983.

QUE84 QUERE A. et al. "SPES : un systeme pour specifier et transformer" CGL2 Nice Juin 1984.

REI70 REIX R. "L'Analyse en informatique de gestion" DUNOD 1970.

REM82 REMY J.L. "Etude des Systemes de reecriture conditionnels et applications aux Types Abstraits Algebriques" These d' etat INPL CRIN 1982.

RIC82 RICHTER G., DURCHHOLTZ R. "IML-Inscribed High-Level Petri-Nets" Conference CRIS1 IFI82.

RID78 RIDDLE W. et al. "DREAM : a Software Design Aid System" Proc. of 3rd. Jerusalem Conference on Information Technology North Holland publication 1978.

ROS77 ROSS D.T., SCHOMAN K.G. "Structured Analysis for Requirements Definition" IEEE Transaction on Software Engeneering 3(1) 1977.

RZE82 RZEVSKI G. et al. "The Evolutionary Design Methodology Applied to I.S. IFIP case" Conference CRIS1 IFI82.

SCH71 SCHUMAN S.A. (ED.) "Proceedings of the International Symposium on extensible Languages" ACM SIGPLAN Notices 6 Dec. 1971.

SCH77 SCHMIDT J.W. "Some High Level Language Constructs for Data of Type Relation" ACM TODS 2(3) 1977.

SCHE84 SCHECK H.J., SCHOLL M.H. "An Algebra for the Relational Model with Relation-valued Attributes" Ecole d'ete IBM Davos Suisse 1984.

SGI Manuel de Presentation de la Mehode CORIG Paris.

SHA84 SHAPIRO D.G. et al. "A knowledge Base for supporting an intelligent Program Editor" Proc. of 7th. Conference on Software Engineering Orlando Mars 1984.

SIS78 Societe d' Informatique et de Systemes "Le Systeme d' Analyse PROTEE" Paris 1978.

SLIGOS Manuel de Presentation de la Methode MINOS Paris.

SMI77 SMITH J.M., SMITH D.C.P. "Data Base Abstraction : Aggregation and Generalization" ACM TODS 2(2) Juin 1977.

SNO80 SNOWDON R.A. "An experience based assesment of development Systems" Software Development tools Springer Verlag Publ. 1980.

SOC77 SOCRATE II "Manuel d' utilisation" Ref. F24742 CII Honeywell Bull 1977.

SOFT Software Engineering
- NAUR P., RANDELL B. (Ed.) NATO 1969
"Report on a Conference Sponsored by the NATO Science Commitee, Garnisch Germany Oct. 1968"
- BUXTON J.N., RANDELL B. (Ed.) NATO 1970
"Report on a Conference Sponsored by the NATO Science Commitee, Rome Italy Oct.1969".

SOL82 SOLVBERG A. "A draft proposal for Integrating System Specification Models" Conference CRIS1 IFI82.

SOU82 SOUQIUERES J. "Construction et Transformation de Programmes Iteratifs" These de Docteur Ingenieur Nancy 1 CRIN 1982.

STA76 STAY J.F. "HIPO and Integrated Program Design" IBM Systems Journal 15(2) 1976.

STO76 STONEBRAKER M.E. et al. "The Design and Implementation of INGRES" ACM TODS 1(3) 1976.

STU84 STUDER R. "Abstract Models of Dialogue Concepts" Proc. of 7th. Conference on Software Engineering Orlando Mars 1984.

SUR83 SURYA D., YADAV B. "Determining an organization's Information Requirements : a state of the art survey" DATA BASE Spring 1983.

TEI77 TEICHROEW D., HERSHEY E.A. "PSL/PSA a computer aided technique for structured documentation and analysis of Information Processing Systems" IEEE Transaction on Software Engineering 1977.

TEI79 TEICHROEW D. et al. "Application of the Entity-Relationship Approach to Information Processing Systems Modelling" Proc. of Conference on E/R Approach Los Angeles Dec.1979.

TEI81 TEITELBAUM T., REPS T. "The CORNELL Program Synthesiser : A Syntax Directed Program Environment" CACM 24(9) 1981.

TSI77 TSICHRITZIS D.C., KUG A. "The ANSI/X3/SPARC DBMS Framework" AFIPS Press 1977.

VEL84 VELEZ F. "Un modele et un langage pour les bases de donnees generalisees : projet TIGRE" These de Docteur Ingenieur INP Grenoble 1984.

VER82 VERHEIJEN G.M.A.,VAN BEKKUM J. "NIAM : an Information Analysis Method" Conference CRIS1 IFI82.

WAR75 WARNIER D. "Les procedures de traitements et leurs donnees (LCP)" Editions d' organisation Paris 1975.

WAR77 WARREN D.H.D., PERREIRA L.M. "PROLOG : the Language and its Implementation compared with LISP" Proc. of the ACM Symposium on A.I. and Programming Languages Rochester USA 1977.

WAS78 WASSERMAN A.I. "A Software Engineering view of Data Base Management" Proc. of VLDB Berlin 1978.

WAS81 WASSERMAN A.I. et al. "Revised Report on PLAIN" ACM SIGPLAN 16(5) 1981.

WAS82A WASSERMAN A.I. "The User Software Engineering Methodology : An Overview" Conference CRIS1 IFI82.

WAS82B WASSERMAN A.I., GUTZ S. "The future Programming" CACM 25(3) 1982.

WER83 WERTZ H. "Etude, realisation et evaluation d' un environnement de programmation utilisant des representations multiples pour le developpement continu de logiciels tres evolues" These d' etat Paris VIII Vincennes 1983.

WIL81 WILLIS R.R. "AIDES : a computer-aided Design of software Systems" Proc. of software Engineering Environments North Holland Publi. 1981.

WIN81 WINSTON P.H., HORN B.K.P. "LISP" Addison Wesley Publication 1981.

WIR71 WIRTH N. "The Programming Language Pascal" Acta Informatica 1 1971.

WIR81 WIRSING M., BROY M. "An analysis of semantic models for algebraic specifications" Proc. Int. Summer School on Theorical Foundations of programming methodology 1981.

WUL76 WULF N. et al. "An introduction to the Construction and Verification of ALPHARD Programs" IEEE Trans. on Software Engineering Dec.1976.

YAM82 YAMAMOTO Y. "An Approach to the generation of software life cycle support Systems" Ph. D. Dissertation University of Michigan 1982.

ZL077 ZLOOF M.M. "QBE : A data base language" IBM System Journal 16(4) 1977.

ZL082 ZLOOF M.M. "QBE : A business language that unifies data and word processing and electronic mail" IBM System Journal 31(3) 1982.

)M DE L'ETUDIANT : Madame THIERY Odile

ATURE DE LA THESE : Doctorat d'Etat ès sciences

VU, APPROUVE ET PERMIS D'IMPRIMER

NANCY, le 20 MAI 1985 n° 825

LE PRESIDENT DE L'UNIVERSITE DE NANCY I

UNIVERSITE NANCY I
LE PRESIDENT
MAINARD

RESUME

Cette thèse propose des solutions pour la spécification abstraite des systèmes d'information de "qualité" en informatique de gestion. Elle se situe au confluent de l'Informatique de Gestion et de la théorie de programmation. Ses fondements sont le modèle conceptuel de REMORA et les Types Abstraits Algébriques.

La première partie (chapitre 1) développe, tout d'abord, une analyse bibliographique de différents projets de recherche sur les SI.
Nous y dégageons des critères d'évaluation, notamment en ce qui concerne les thèmes de spécification abstraite et de modélisation à l'aide des Types abstraits. Nous concluons ce chapitre sur la nécessité de proposer une formalisation à l'aide des Types Abstraits pour les SI.

Nous exposons ensuite cette formalisation (chapitre 2). Nous introduisons les schémas de type permettant d'exprimer les concepts du modèle REMORA complété par l'introduction du type SI permettant l'expression complète et le contrôle d'un SI particulier. La formalisation s'appuie sur l'axiomatisation algébrique de constructeurs paramétrés de base.

Un troisième chapitre de cette partie 1 est consacré à la présentation du langage de spécification choisi (LASSIF). Le langage structurel permet la description des aspects statiques d'un SI, il est en fait fort proche de la formalisation précédente.
Le langage d'énoncé permet l'expression des textes de programmes, il est à syntaxe quasi-libre, déclaratif et non procédural.

La deuxième partie est consacrée à l'exposé de nos résultats pratiques en matière de prototype d'aide à la conception d'un SI. Nous y justifions nos choix de réalisation actuels et futurs.

La troisième partie est consacrée à la présentation de travaux qui sont à l'origine de notre réflexion : la justification de l'intérêt d'un processeur de la dynamique, outil permettant le lancement automatique des programmes à l'arrivée de nouvelles informations.
Nous présentons une solution relationnelle pour cet outil et proposons des schémas de transformation Types Abstraits modèle relationnel.

Mots clés :
Informatique de gestion - Systèmes d'informations - Types Abstraits - Logiciel d'aide à la conception - Gestion de la dynamique - Langage de spécification.