SHIVA - un modèle de données relationnel étendu pour la mise en oeuvre de base de connaissances centrée objets

Ali Bensaid

THESE

présentée à

l'Institut National Polytechnique de Grenoble

pour obtenir le grade de
DOCTEUR INGENIEUR
«Informatique»

par

Ali BENSAID

— SHIVA —
UN MODELE DE DONNEES RELATIONNEL ETENDU POUR LA MISE EN OEUVRE DE BASES DE CONNAISSANCES CENTREES OBJETS

Thèse soutenue le 10 mai 1985 devant la commission d'examen.

C. DELOBEL	Président
J. MOSSIERE	
M. SCHOLL	Examinateurs
J. MERMET	
D. PECCOUD	

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Année universitaire 1982-1983

Président de l'Université : D. BLOCH

Vice-Président : René CARRE
Hervé CHERADAME
Marcel IVANES

PROFESSEURS DES UNIVERSITES :

ANCEAU François	E.N.S.I.M.A.G.
BARRAUD Alain	E.N.S.I.E.G.
BAUDELET Bernard	E.N.S.I.E.G.
BESSON Jean	E.N.S.E.E.G.
BLIMAN Samuel	E.N.S.E.R.G.
BLOCH Daniel	E.N.S.I.E.G.
BOIS Philippe	E.N.S.H.G.
BONNETAIN Lucien	E.N.S.E.E.G.
BONNIER Etienne	E.N.S.E.E.G.
BOUVARD Maurice	E.N.S.H.G.
BRISSONNEAU Pierre	E.N.S.I.E.G.
BUYLE BODIN Maurice	E.N.S.E.R.G.
CAVAIGNAC Jean-François	E.N.S.I.E.G.
CHARTIER Germain	E.N.S.I.E.G.
CHENEVIER Pierre	E.N.S.E.R.G.
CHERADAME Hervé	U.E.R.M.C.P.P.
CHERUY Arlette	E.N.S.I.E.G.
CHIAVERINA Jean	U.E.R.M.C.P.P.
COHEN Joseph	E.N.S.E.R.G.
COUMES André	E.N.S.E.R.G.
DURAND Francis	E.N.S.E.E.G.
DURAND Jean-Louis	E.N.S.I.E.G.
FELICI Noël	E.N.S.I.E.G.
FOULARD Claude	E.N.S.I.E.G.
GENTIL Pierre	E.N.S.E.R.G.
GUERIN Bernard	E.N.S.E.R.G.
GUYOT Pierre	E.N.S.E.E.G.
IVANES Marcel	E.N.S.I.E.G.
JAUSSAUD Pierre	E.N.S.I.E.G.
JOUBERT Jean-Claude	E.N.S.I.E.G.
JOURDAIN Geneviève	E.N.S.I.E.G.
LACOUME Jean-Louis	E.N.S.I.E.G.
LATOMBE Jean-Claude	E.N.S.I.M.A.G.

.../...

LESSIEUR Marcel	E.N.S.H.G.
LESPINARD Georges	E.N.S.H.G.
LONGEQUEUE Jean-Pierre	E.N.S.I.E.G.
MAZARE Guy	E.N.S.I.M.A.G.
MOREAU René	E.N.S.H.G.
MORET Roger	E.N.S.I.E.G.
MOSSIERE Jacques	E.N.S.I.M.A.G.
PARIAUD Jean-Charles	E.N.S.E.E.G.
PAUTHENET René	E.N.S.I.E.G.
PERRET René	E.N.S.I.E.G.
PERRET Robert	E.N.S.I.E.G.
PIAU Jean-Michel	E.N.S.H.G.
POLOUJADOFF Michel	E.N.S.I.E.G.
POUPOT Christian	E.N.S.E.R.G.
RAMEAU Jean-Jacques	E.N.S.E.E.G.
RENAUD Maurice	U.E.R.M.C.P.P.
ROBERT André	U.E.R.M.C.P.P.
ROBERT François	E.N.S.I.M.A.G.
SABONNADIERE Jean-Claude	E.N.S.I.E.G.
SAUCIER Gabrielle	E.N.S.I.M.A.G.
SCHLENKER Claire	E.N.S.I.E.G.
SCHLENKER Michel	E.N.S.I.E.G.
SERMET Pierre	E.N.S.E.R.G.
SILVY Jacques	U.E.R.M.C.P.P.
SOHM Jean-Claude	E.N.S.E.E.G.
SOUQUET Jean-Louis	E.N.S.E.E.G.
VEILLON Gérard	E.N.S.I.M.A.G.
ZADWORNY François	E.N.S.E.R.G.

PROFESSEURS ASSOCIES

BASTIN Georges	E.N.S.H.G.
BERRIL John	E.N.S.H.G.
CARREAU Pierre	E.N.S.H.G.
GANDINI Alessandro	U.E.R.M.C.P.P.
HAYASHI Hirashi	E.N.S.I.E.G.

PROFESSEURS UNIVERSITE DES SCIENCES SOCIALES (Grenoble II)

BOLLIET Louis
Chatelin Françoise

PROFESSEURS E.N.S. Mines de Saint-Etienne

RIEU Jean
SOUSTELLE Michel

CHERCHEURS DU C.N.R.S.

FRUCHART Robert	Directeur de Recherche
VACHAUD Georges	Directeur de Recherche

.../...

ALLIBERT Michel	Maître de Recherche
ANSARA Ibrahim	Maître de Recherche
ARMAND Michel	Maître de Recherche
BINDER Gilbert	
CARRE René	Maître de Recherche
DAVID René	Maître de Recherche
DEPORTES Jacques	
DRIOLE Jean	Maître de Recherche
GIGNOUX Damien	
GIVORD Dominique	
GUELIN Pierre	
HOPFINGER Emil	Maître de Recherche
JOUD Jean-Charles	Maître de Recherche
KAMARINOS Georges	Maître de Recherche
KLEITZ Michel	Maître de Recherche
LANDAU Ioan-Dore	Maître de Recherche
LASJAUNIAS J.C.	
MERMET Jean	Maître de Recherche
MUNIER Jacques	Maître de Recherche
PIAU Monique	
PORTESEIL Jean-Louis	
THOLENCE Jean-Louis	
VERDILLON André	

CHERCHEURS du MINISTERE de la RECHERCHE et de la TECHNOLOGIRE (Directeurs et Maîtres de Recherches, ENS Mines de St. Etienne)

LESBATS Pierre	Directeur de Recherche
BISCONDI Michel	Maître de Recherche
KOBYLANSKI André	Maître de Recherche
LE COZE Jean	Maître de Recherche
LALAUZE René	Maître de Recherche
LANCELOT Francis	Maître de Recherche
THEVENOT François	Maître de Recherche
TRAN MINH Canh	Maître de Recherche

PERSONNALITES HABILITEES à DIRIGER des TRAVAUX de RECHERCHE (Décision du Conseil Scientifique)

ALLIBERT Colette	E.N.S.E.E.G.
BERNARD Claude	E.N.S.E.E.G.
BONNET Rolland	E.N.S.E.E.G.
CAILLET Marcel	E.N.S.E.E.G.
CHATILLON Catherine	E.N.S.E.E.G.
CHATILLON Christian	E.N.S.E.E.G.
COULON Michel	E.N.S.E.E.G.
DIARD Jean-Paul	E.N.S.E.E.G.
EUSTAPOPOULOS Nicolas	E.N.S.E.E.G.
FOSTER Panayotis	E.N.S.E.E.G.

.../...

GALERIE Alain	E.N.S.E.E.G.
HAMMOU Abdelkader	E.N.S.E.E.G.
MALMEJAC Yves	E.N.S.E.E.G. (CENG)
MARTIN GARIN Régina	E.N.S.E.E.G.
NGUYEN TRUONG Bernadette	E.N.S.E.E.G.
RAVAINE Denis	E.N.S.E.E.G.
SAINFORT	E.N.S.E.E.G. (CENG)
SARRAZIN Pierre	E.N.S.E.E.G.
SIMON Jean-Paul	E.N.S.E.E.G.
TOUZAIN Philippe	E.N.S.E.E.G.
URBAIN Georges	E.N.S.E.E.G. (Laboratoire des ultra-réfractaires ODEILLON)
GUILHOT Bernard	E.N.S. Mines Saint Etienne
THOMAS Gérard	E.N.S. Mines Saint Etienne
DRIVER Julien	E.N.S. Mines Saint Etienne
BARIBAUD Michel	E.N.S.E.R.G.
BOREL Joseph	E.N.S.E.R.G.
CHOVET Alain	E.N.S.E.R.G.
CHEHIKIAN Alain	E.N.S.E.R.G.
DOLMAZON Jean-Marc	E.N.S.E.R.G.
HERAULT Jeanny	E.N.S.E.R.G.
MONLLOR Christian	E.N.S.E.R.G.
BORNARD Guy	E.N.S.I.E.G.
DESCHIZEAU Pierre	E.N.S.I.E.G.
GLANGEAUD François	E.N.S.I.E.G.
KOFMAN Walter	E.N.S.I.E.G.
LEJEUNE Gérard	E.N.S.I.E.G.
MAZUER Jean	E.N.S.I.E.G.
PERARD Jacques	E.N.S.I.E.G.
REINISCH Raymond	E.N.S.I.E.G.
ALEMANY Antoine	E.N.S.H.G.
BOIS Daniel	E.N.S.H.G.
DARVE Félix	E.N.S.H.G.
MICHEL Jean-Marie	E.N.S.H.G.
OBLED Charles	E.N.S.H.G.
ROWE Alain	E.N.S.H.G.
VAUCLIN Michel	E.N.S.H.G.
WACK Bernard	E.N.S.H.G.
BERT Didier	E.N.S.I.M.A.G.
CALMET Jacques	E.N.S.I.M.A.G.
COURTIN Jacques	E.N.S.I.M.A.G.
COURTOIS Bernard	E.N.S.I.M.A.G.
DELLA DORA Jean	E.N.S.I.M.A.G.
FONLUPT Jean	E.N.S.I.M.A.G.
SIFAKIS Joseph	E.N.S.I.M.A.G.
CHARUEL Robert	U.E.R.M.C.P.P.
CADET Jean	C.E.N.G.
COEURE Philippe	C.E.N.G. (LETI)

.../...

DELHAYE Jean-Marc	C.E.N.G. (STT)
DUPUY Michel	C.E.N.G. (LETI)
JOUVE Hubert	C.E.N.G. (LETI)
NICOLAU Yvan	C.E.N.G. (LETI)
NIFENECKER Hervé	C.E.N.G.
PERROUD Paul	C.E.N.G.
PEUZIN Jean-Claude	C.E.N.G. (LETI)
TAIEB Maurice	C.E.N.G.
VINCENDON Marc	C.E.N.G.

LABORATOIRES EXTERIEURS

DEMOULIN Eric	C.N.E.T.
DEVINE	C.N.E.T. (R.A.B.)
GERBER Roland	C.N.E.T.
MERCKEL Gérard	C.N.E.T.
PAULEAU Yves	C.N.E.T.
GAUBERT C.	I.N.S.A. Lyon

ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT-ETIENNE

Directeur	: Monsieur M. MERMET
Directeur des Etudes et de la formation	: Monsieur J. LEVASSEUR
Directeur des recherches	: Monsieur J. LEVY
Secrétaire Général	: Mademoiselle M. CLERGUE

Professeurs de 1ère Catégorie

COINDE	Alexandre	Gestion
GOUX	Claude	Métallurgie
LEVY	Jacques	Métallurgie
LOWYS	Jean-Pierre	Physique
MATHON	Albert	Gestion
RIEU	Jean	Mécanique - Résistance des matériaux
SOUSTELLE	Michel	Chimie
FORMERY	Philippe	Mathématiques Appliquées

Professeurs de 2ème catégorie

HABIB	Michel	Informatique
PERRIN	Michel	Géologie
VERCHERY	Georges	Matériaux
TOUCHARD	Bernard	Physique Industrielle

Directeur de recherche

LESBATS	Pierre	Métallurgie

Maîtres de recherche

BISCONDI	Michel	Métallurgie
DAVOINE	Philippe	Géologie
FOURDEUX	Angeline	Métallurgie
KOBYLANSKI	André	Métallurgie
LALAUZE	René	Chimie
LANCELOT	Francis	Chimie
LE COZE	Jean	Métallurgie
THEVENOT	François	Chimie
TRAN MINH	Canh	Chimie

Personnalités habilitées à diriger des travaux de recherche

DRIVER	Julian	Métallurgie
GUILHOT	Bernard	Chimie
THOMAS	Gérard	Chimie

Professeur à l'UER de Sciences de Saint-Etienne

VERGNAUD	Jean-Maurice	Chimie des Matériaux & chimie industrielle

****

Je tiens à remercier :

Monsieur Claude DELOBEL, Professeur à l'Université Scientifique et Médicale de Grenoble, pour l'intérêt qu'il a porté à ce travail, sa dispostion à mon égard et ses précieux conseils. Je le remercie aussi d'avoir bien voulu me faire l'honneur de présider le jury de cette thèse.

Monsieur Jacques MOSSIERE, Professeur à l'institut National Polytechnique de Grenoble et directeur du Laboratoire de Génie Informatique de l'IMAG, pour avoir pris la responsabilité de cette thèse, pour la confiance qu'il m'a témoigné et d'avoir bien voulu faire partie de ce jury.

Monsieur Michel SCHOLL, Ingénieur de Recherche à l'INRIA, qui a accepté de juger cette thèse et dont les critiques m'ont été précieuses.

Monsieur Jean MERMET, Directeur du laboratoire ARTEMIS de l'IMAG, pour avoir bien voulu faire partie de ce jury.

Monsieur Dominique PECCOUD, Directeur de l'ESAP à Toulouse, pour m'avoir fait l'honneur de participer à ce jury.

Je tiens également à remercier :

Monsieur Francois RECHENMANN, responsable du projet SHIRKA, pour nos fructeuses discusions et son amabilité.

Monsieur Jacques BRIAT, pour sa confiance et ses petits, mais trés intéressants conseils.

Toute l'équipe SHIRKA, pour la chaleureuse ambiance de travail, ainsi que les membres du projet EDORA de Sophia-Antipolis.

Enfin, je remercie l'INRIA, sans qui, ce travail ne serait.

A mon père
A ma mère
A ma famille
Et a mon pote

Résumé:

Cette thèse présente un modèle de base de données relationnel étendu : SHIVA, conçu pour permettre la mise en œuvre de bases de connaissances centrées objets. Le modèle SHIVA a été développé dans le cadre du projet SHIRKA : des bases de connaissances centrées objets et, un interpréteur de ce modèle sera utilisé pour écrire le système de gestion de bases de connaissances du système SHIRKA.

Le premier chapitre est une description du système SHIRKA et de la représentation des connaissances centrée objets. Dans le deuxième chapitre, les principales extensions du mode relationnel sont exposées. Enfin, le troisième chapitre est consacré au modèle SHIVA.

Mots-clefs:

Bases de données relationnelles, extensions du modèle relationnel, relations non normalisées, bases de connaissances centrées objets, langage Lisp.

TABLES DES MATIERES

- INTRODUCTION -

Nous assistons depuis quelques années à une certaine banalisation de l'intelligence artificielle. Longtemps considérée comme une branche à part de l'informatique, l'intelligence artificielle semble maintenant intéresser l'ensemble des grands thèmes de l'informatique.

Dans certains domaines, l'intelligence artificielle est perçue comme un ensemble de techniques utilisées pour les recherches ou les réalisations dans ces domaines. Citons, par exemple, le nombre croissant de systèmes experts réalisés ou à l'étude en CAO de VLSI, ou encore ceux utilisés pour les tests ou les configurations de machines.

Une autre voie suivie est l'apport d'un ou plusieurs thèmes complets de l'informatique classique pour aider à la résolution de certains problèmes de l'intelligence artificielle. C'est le cas notamment de recherches d'architectures logicielles et matérielles pour l'exécution parallèle de logiciels pour l'intelligence artificielle. Citons aussi les efforts faits pour le développement ou l'amélioration de langages pour l'intelligence artificielle, de même les études entreprises pour la réalisation d'interfaces Homme-Machine pour permettre une utilisation plus aisée des ordinateurs par les non-spécialistes de l'informatique.

Dans le cas des grands projets nationaux ou industriels, tels le projet japonais d'ordinateurs dit de cinquième génération ou les recherches financées aux Etats-Unis par de grands constructeurs, l'intelligence artificielle est traitée sur de nouvelles bases et l'ensemble des techniques de l'informatique participent à la réalisation de ces projets.

Il existe cependant une autre approche, moins ambitieuse, mais efficace semble-t'il, qui consiste d'une part, à utiliser les techniques de l'intelligence artificielle pour faire progresser des domaines précis de l'informatique et d'autre part, à apporter les résultats des recherches menées dans ces domaines, pour aider à la résolution de problèmes techniques de l'intelligence artificielle.

Cette dernière démarche est en fait une judicieuse collaboration entre l'informatique classique et l'intelligence artificielle. Sans remettre en cause les progrés accomplis, l'intelligence artificielle apporte une vision nouvelle et permet un enrichissement des techniques classiques par l'apport de nouveaux outils, par exemple les langages de l'intelligence artificielle ou les modèles de représentation de la connaissance. A l'inverse, de part sa nature pluridisciplinaire, l'intelligence artificielle profite des recherches et des travaux,

dont l'efficacité n'est plus à démontrer, effectués dans les disciplines classiques de l'informatique.

Ce type de collaboration se manifeste notamment dans les recherches effectuées pour la réalisation de systèmes de gestion de bases de connaissances en utilisant les modèles de bases de données.

Si l'intelligence artificielle a su développer des modèles de représentation de la connaissaince ainsi que des techniques de mise en oeuvre et de contrôle du raisonnement, la réalisation d'un système de gestion de base de connaissances, assurant un stockage efficace et fiable ainsi que des méthodes d'accès respectant le modèle de représentation de la connaissance choisi, pose encore à l'heure actuelle quelques problèmes. Dans ce domaine, l'approche consistant à utiliser les bases de données en tant que bases de connaissances semble prometteuse. Cette collaboration, bases de données et bases de connaissances, a donné lieu déjà à quelques réalisations telles que les bases de données déductives ou l'utilisation de systèmes de gestion de bases de données dans la conception de systèmes experts. A l'inverse, les modèles de représentation de la connaissance sont étudiés afin d'augmenter la puissance des modèles de bases de données. C'est le cas, notamment de l'utilisation de certains concepts des réseaux sémantiques pour accroître le pouvoir sémantique des modèles de données ou l'utilisation des règles de production pour la vérification des contraintes d'intégrité.

Le modèle SHIVA :

Le modèle de bases de données : SHIVA, a été conçu pour permettre la spécification et la mise en oeuvre du système de gestion des connaissances des applications utilisant la représentation des connaissances centrées objet.

SHIVA est un modèle de données relationnel non normalisé en ce sens où il s'inspire pleinement du modèle relationnel défini par CODD sans toutefois respecter la "première forme normale" de ce modèle. S'il est possible dans le modèle SHIVA de structurer les données sous forme de relations et d'utiliser les principaux opérateurs de l'algèbre relationnelle, la contrainte de "valeur atomique" pour les valeurs des constituants des relations n'est pas imposée.

Dans le modèle SHIVA, une valeur d'un constituant d'une relation peut être une valeur atomique : entier, réel, ou chaîne de caractères mais aussi un ensemble ou une liste de valeurs atomiques. De plus, dans ce modèle, il est possible de définir des "constituants structurés" au sens "enregistrements" du langage Pascal.

Le non respect de la "première forme normale" du modèle relationnel est rendu nécessaire du fait que les objets d'une base de connaissance sont généralement complexes. Le problème de la gestion de données structurées se pose d'une manière plus générale dans la conception de nouvelles applications de bases de données telles que les bases de données textuelles, de gestion de documents ou d'images; ou leur utilisation dans des domaines tels que la CAO, l'économétrie ou l'intelligence artificielle.

Une des première caractéristiques du modèle SHIVA est donc sa capacité à gérer des structures de données complexes en donnant au concepteur de la base de données la possibilité de définir ses propres domaines.

La possibilité de pouvoir définir des objets structurés n'est pas suffisante pour utiliser une base de données dans la conception d'une base de connaissances. En effet, la richesse et le pouvoir sémantique des modèles de représentation de la connaissance seraient perdus, car la transcription de ces modèles dans un modèle de base de données se traduirait par l'ajout et la dispersion de traitements à effectuer lors de l'accès aux données de la base.

Dans le cas du modèle de représentation de la connaissance centrée objet, afin de garder au maximun la possibilité de rassembler les divers aspects de la connaissance attaché à un objet donné dans un même schéma, les domaines du modèle SHIVA ont été enrichis du concept de "facettes" qui permettent notamment d'associer des prédicats complexes aux composantes du domaine, d'assurer l'intégrité et la cohérence de la base et de déclencher des procédures lors de la réussite ou de l'échec de certaines actions.

Objectifs de SHIVA

L'idée principale qui a motivé la spécification du modèle SHIVA, est la définition d'un système de stockage de données externes pour la mise en oeuvre des systèmes de gestion de base de connaissances des applications de l'intelligence artificielle utilisant la représentation des connaissances centrée objets.

Un tel système de stockage se doit d'assurer une indépendance maximum des données physiques et des données logiques. Comme pour bon nombre d'applications informatiques, cette indépendance est primordiale pour une base de connaissances.

Ce système de stockage se doit aussi de proposer un modèle de données d'utilisation et de compréhension simple, mais riche et le mieux adapté possible à l'application utilisant la base de connaissance. Il serait inconcevable de proposer à l'application, tout au moins à la partie assurant le contrôle du raisonnement et l'exploitation des connaissances, une multitudes de structures, de règles et de contraintes pour la représentation des objets et de la

connaissance qui leur est associée.

Le système de stockage à proposer se devra d'offrir, de plus, des méthodes garantissant l'intégrité et la confidentialité des données et des connaissances traitées par l'application utilisant la base de connaissance. Les règles et les méthodes permettant la prise en compte de ces fonctionnalités devront être fournies par le système de stockage, sans quoi l'application serait alourdie par d'innombrables tests et algorithmes de contrôle lors de l'accès aux données.

Ces critères, dans le cas de la représentation des connaissances centrée objets, nous a amené a proposer un système de gestion de bases de données pour la mise en œuvre du système de stockage. Par la qualité des services offerts et son grand pouvoir d'expression, le modèle de données relationnel fut retenu. Toutefois, des extensions de ce modèle doivent être envisagée pour une meilleure correspondance entre les modèle de données des applications et le modèle relationnel.

Offrir un système de stockage et de manipulation de données externes, tenant compte des besoins des systèmes de gestion de connaissances énoncées précédemment et, enrichir le modèle relationnel dans le but de rendre plus efficace son emploi pour la mise en oeuvre de bases de connaissances centrées objets, tels sont les objectifs que se propose d'atteindre le modèle relationnel SHIVA.

Le système SHIRKA :

Le modèle SHIVA sera utilisé pour la mise en oeuvre du système de gestion de base de connaissances du système SHIRKA (Des Bases de Connaissances Centrées Objets) [Gra84,Rec84]. Ce système, en cours de développement au laboratoire ARTEMIS/IMAG, a pour but d'une part, la définition d'un modèle de représentation de la connaissance centrée objets et d'autre part, la réalisation d'un système de gestion de base de connaissances reposant sur ce modèle. Le système SHIRKA est écrit en langage Lisp [Hor81] et utilise actuellement un système de bases de données relationnel : SHIMER [Ben84], pour la gestion de ses données externes.

SHIMER est un système relationnel entièrement écrit en Lisp, à l'aide du système Le_LISP [Cha84] offrant aux utilisateurs de ce langage une vision relationnelle de leurs objets. Combinant à la fois les fonctionnalités des systèmes relationnels et les immenses potentialités du langage Lisp, SHIMER offre ainsi, sous forme de langage de fonctions Lisp, la gestion des structures de données du modèle relationnel et l'ensemble des opérateurs de l'algèbre relationnelle.

SHIRKA accède à la base de données relationnelle en utilisant un mécanisme de traduction qui effectue la correspondance des données entre le modèle de données de SHIRKA et le modèle relationnel de SHIMER. L'objectif, à terme, est de remplacer ce mécanisme de traduction et l'actuelle base de données relationnelle (SHIMER) par un système relationnel évolué reposant sur le modèle SHIVA.

Ce système sera aussi écrit en Lisp, assurant ainsi une meilleure interface avec SHIRKA. La gestion de données complexes, la possibilité d'associer des facettes aux domaines ainsi que l'attachement procédural permettront à SHIRKA d'utiliser le système SHIVA pour la mise en oeuvre de sa base de connaissances.

CHAPITRE 1

- LE MODELE SHIVA ET LE SYSTEME SHIRKA -

1. Le projet SHIRKA

Le projet SHIRKA, en cours de développement au laboratoire ARTEMIS/IMAG, a pour objectif la réalisation d'un système de gestion de bases de connaissances centrées objets.

La réalisation du système SHIRKA s'effectue dans le cadre du projet EDORA (Equations Différentielles Ordinaires et Récurrentes Appliquées) [Gou84,Vig84].' EDORA est un projet de recherche du centre INRIA de Sophia Antipolis et a pour objectif de rendre l'approche de la modélisation mathématique accessible dans divers domaines, en particulier en biologie. C'est un système informatique intelligent dont le but est de soutenir la démarche heuristique du modélisateur, en lui apportant les méthodes mathématiques nécessaires et en lui montrant les possibilités et limites de ces outils.

Le système SHIRKA est tout particulièrement développé pour EDORA, qui sera la première application à utiliser le système de gestion de base de connaissances proposé. Néanmoins, un des axes qui dirigent la conception de SHIRKA est la volonté de ne point réaliser un système expert propre au projet EDORA, en proposant une représentation de la connaissance adaptée au problème et en réalisant des mécanismes d'exploitation «ad-hoc». L'objectif est de définir un modèle de représentation de la connaissance suffisamment général, indépendant des applications et de fournir un système de gestion de bases de connaissances reposant sur ce modèle.

Dans ce but, SHIRKA offre un formalisme de représentation des connaissances centrée objet et des mécanismes d'exploitation dont le contrôle est décrit dans les objets eux-mêmes. Par rapport à la représentation des connaissances par des règles de production [Kin77] classiquement utilisée par les système experts actuels, une représentation centrée objet offre de nombreux avantages [Aik83] :

- Une même description, ou schéma, rassemble les divers aspects de la connaissance attachée à un objet donné.

- La possibilité d'instancier un schéma en donnant des valeurs à ses composantes, ou attributs, permet de réunir base de connaissances et de faits, ces derniers étant décrits par les instances des schémas.

• Les mécanismes d'inférences sont multiples : héritage , instanciation, classification et attachement procédural.

L'attachement procédural permet :

- d'associer des prédicats complexes à un attribut,

- de faire intervenir la connaissance procédurale lors de la détermination des valeurs des attributs,

- d'assurer l'intégrité et la cohérence de la base de faits,

- de diriger le raisonnement en agissant, à la suite de la réussite ou de l'échec du processus d'identification, sur un agenda des actions à effectuer,

- de décrire les méthodes de représentation externes des schémas.

Shirka apparaît particuliement bien adapté lorsque :

- le domaine des connaissances est bien structurable,

- les objets peuvent être définis sous des points de vue multiples,

- une partie des connaissances ne peut être décrite que sous forme procédurale,

- une gestion fine et explicite du raisonnement en fonction du contexte est souhaitable.

2. Représentation des connaissances centrée objets

NOTIONS D'OBJETS

Les représentations des connaissances en objets structurés ont diverses origines. Elles s'inspirent :

- des «frames» de Minsky [Min75],
- des «scripts» de Shan [Pin81], Schank
- des classes du langage SIMULA [Bri83],
- des «objets» utilisés dans les langages orientés comme SMALLTALK [Rob83,Coi83],

et ont donné lieu à plusieurs études et réalisations : les langages KRL [Win77] et FRL [Rob80], les FLAVORS [Wei80] PLASMA [Gra82], CEYX [Hul84], FORMES [Ser82] et le langage LOOPS [Ste82].

Parmi les principales caractéristiques de la représentation à l'aide d'objets, on peut citer :

- Un représentation déclarative reposant sur la notion d'objet. Le monde est vu comme un ensemble d'objets autonomes, chacun ayant une existence réelle dans la base de connaissances.

- Chaque objet comporte à la fois une composante statique et une composante dynamique. On utilise un ainsi une puissante combinaison de déclaratif et de procédural.

- Les notions de hiérarchie d'objets et d'héritage jouent un rôle essentiel dans l'écriture et la manipulation de la base de connaissances.

Cette représentation est utilisée dans deux catégories de logiciels : des représentation centrées objets, dans lesquelles la dynamique est réduite, ou des langages orientés objets.

LES FRAMES

L'idée des schémas («frames») est issue des travaux de Minsky [Min75]. Le frame (schéma) [Kui75] est une structure de données composée d'un groupe d'attributs («slots») chacun décrit par un ensemble facettes («facets») pouvant désigner un schéma ou être un simple identificateur.

La structure d'un schéma est la suivante :

```
(nom-schéma
        (attribut-1 (facette-11 val-11)
                    (facette-12 val-12)
                        ...
                    (facette-1n val-1n))
                ...
        (attribut-p (facette-p1 val-p1)
                    (facette-p2 val-p2)
                        ...
                    (facette-pq val-pq)))
```

Exemples de schémas :

1)

```
(point   (x ($un reel))
         (y ($un reel)))
```

Définition du schéma de nom POINT, comportant deux attributs : X et Y. La facette $UN indique le domaine de définition des attributs et a pour valeur : REEL.

2)

```
(segment
        (ori ($un point))
        (ext ($un point))
        (lg  ($un reel)
             ($si-besoin (distance ori ext))))
```

Les attributs ORI et EXT prennent leurs valeurs dans l'ensemble des POINTS. La facette $SI-BESOIN permet d'enoncer une règle de calcul.

3)

```
(courbe
        (points ($liste-de point)))
```

•

Les valeurs de l'attribit POINTS est une liste d'instances du schéma POINTS.

Un schéma modélise un concept. Un représentant du concept, ou instance du schéma, est obtenu par instanciation, en remplissant les attributs correspondants au cas particulier traité.

Les entités ainsi décrites sont reliées entre elles et forment le plus souvent une arborescence, parfois un réseau. Les liens hiérarchiques dans l'arborescence sont de type généralisation-spécialisation entre concepts et concept-instance, entre un schéma et tous ses représentants. Ces liens permettent d'utiliser un mécanisme d'héritage par lequel un objet hérite des propriétés des entités qui lui sont hiérarchiquement supérieures.

Par exemple :

```
(seg-colores
        ($sorte-de ($valeur segment))
        (couleur   ($un chaine)
                   ($defaut "vert")
                   ($domaine ("bleu" "rouge" "vert"))))
```

L'attribut $SORTE-DE spécifie que le schéma SEG-COLORES est une spécialisation du schéma SEGMENT. Toutes les instances de SEG-COLORES héritent des attributs de SEGMENT et elles ont un attribut supplémentaire : COULEUR.

Les seules valeurs admises pour l'attribut COULEUR sont les chaînes de caractères citées dans la liste de la facette $DOMAINE. A l'aide d'une facette spéciale : $DEFAUT, et en l'absence de valeurs pour une instance du schéma, l'attribut prendra par défaut la valeur "vert".

Des liens horizontaux entre entités ($LISTE) sont utilisés à l'aide de références imbriquées. Un attribut peut avoir pour valeur une indirection vers un autre schéma.

Des facettes rendent les entités actives, à l'aide de réflexes. Ce sont des procédures attachées aux attributs, automatiquement activées lors d'un certain type de manipulation sur le attribut concerné. Le réflexe $SI-BESOIN, désigne la procédure à utiliser pour obtenir la valeur d'un attribut si elle manque. Les réflexes $SI-AJOUT et $SI-SUP indiquent ce qu'il faut faire en cas d'ajout ou de suppression de la valeur d'un attribut.

3. Le modèle de SHIRKA

3.1. Les schémas et les insatnces

Un schéma SHIRKA est une liste à trois niveaux d'imbrication. Au premier niveau se trouve le nom du schéma et aux deuxième niveau la liste des attributs. Chaque attribut est lui même décrit au troisième niveau par une liste de facettes, chacune d'elles possédant une ou plusieurs valeurs.

Les facettes font partie d'un ensemble prédéfini mais leurs valeurs ainsi que le nom du schéma et les attributs sont définis par le concepteur de la base. L'ensemble des facettes est fixé et définit la sémantique de la représentation des connaissances. La valeurs sont elles-mêmes des schémas ou des références à des schémas, mais possèdent des interprétations différentes selon les facettes auxquelles elles sont associées. La puissance d'expression de la connaissance à l'aide des schémas et donc essentiellement lié à la signification des facettes disponibles.

Un schéma définit une famille d'objets. Un objet particulier de cette famille est décrit lui aussi par un schéma, dit schéma d'instance, qui ne diffère du schéma de classe correspondant que par la donnée supplémentaire de valeurs d'attributs. Il est donc composé du nom de l'objet et d'une liste d'attributs possédant tous la facette $valeur et la valeur correspondante.

Une instance peut être complète ou partielle et hérite directement du schéma de classe auquel elle est liée par l'attribut $est_un.

Exemple 1:

Dans le cadre d'un système d'aide à la modélisation, un schéma peut décrire un traitement tel que l'intégration numérique d'un modèle ou l'identification de ses paramètres, un objet de modélisation tel qu'une équation, un modèle ou un paramètre.

Par exemple, le schéma :

```
(EQUATION
        (VARIABLES     ($liste-de VARIABLE))
        (MEMBRE-G      ($un  VAR-ETAT))
        (MEMBRE-D      ($un  EXPR-ARITH)))
```

qui exprime qu'une équation est définie par trois attributs : la liste des variables qui apparaissent, le membre gauche qui fait intervenir une variable d'état et un membre droit composé d'une expression arithmique.

VARIABLE et VAR-ETAT sont les noms de deux schémas définis par le concepteur alors que EXPR-ARITH fait partie des schémas prédéfinis au même titre que BOOLEEN, ENTIER, REEL et CHAINE.

La liaison entre schémas est ainsi introduite à travers les facettes $liste_de et $un, la première décrivant des relations 1:n, la seconde des relations 1:1.

Une équation particulière telle que :

$$x' = a.x - b.y$$

sera appelée instance de ce schéma. MEMBRE-G aura dans ce cas, pour valeur une référence vers l'instance du schéma VAR-ETAT associée à X', MEMBRE-DROIT l'expression arithmétique AX-YB et VARIABLES aura pour valeur la liste des références vers les deux instances du schéma VARIABLE associées à X ey Y.

Exemple 2 :

```
(PERSONNE
(NOM     ($un chaine))
(AGE     ($un entier)
         ($intervalle  (0 120)))
(A_POUR_PERE       ($un PERSONNE))
(GRAND_PERE_DE     ($liste_de PERSONNE))
(DATE_NAISS        ($un DATE)))

(PERSONNE#1
(est_un       ($valeur PERSONNE))
(NOM          ($valeur "jean"))
 (A_POUR_PERE ($valeur PERSONNE#33))
(DATE_NAISS   ($valeur DATE#2)))
```

```
(DATE#2
(est_un ($valeur DATE))
(JOUR   ($valeur 29))
(MOIS   ($valeur "mai"))
(ANNEE  ($valeur 1951)))
```

Le schéma PERSONNE décrit les connaissances que l'on a sur une personne : son nom, son age, le nom de son père, la liste de ses éventuels petits enfants et sa date de naissance. L'objet PERSONNE#1 est une instance de PERSONNE. Dans cette instance, l'attribut est_un donne le nom du schéma correspondant. Cet attribut est géré par SHIRKA et n'apparaît donc pas dans les attributs du schéma PERSONNE.

Seules les valeurs des attributs NOM, A_POUR_PERE et DATE_NAISS sont connues pour la personne d'itenfiant PERSONNE#1. La valeur de l'attribut est_un est le nom du schéma, celle de NOM est une chaîne de caractères et celles des attributs A_POUR_PERE et DATE_NAIS sont des identifiants d'instances, respectivement l'identifiant de l'instance de PERSONNE décrivant le père et l'identifiant du schéma DATE décrivant la date de naissance dont on donne ici les composantes.

3.2. Les facettes

Les facettes sont réparties en plusieurs catégories :

- les facettes de typage : $un et $liste_de, permettent de de définir le type des valeurs des attributs d'un schéma. Ce type est soit simple : entier, réel, booléen ou chaîne de caractères, soit défini par un schéma. Dans ce dernier cas, les valeurs de l'attribut sont des références à des instances ce schéma ou de schémas plus spécifiques.

- les facettes décrivant un moyen d'obtenir la valeur d'un attribut: $valeur, $si_besoin et $defaut.

  - La facette $valeur, au niveau d'un schéma de classe, précise une valeur d'attribut commune à tous les représentants de cette classe. Cette valeur peut être simple, par exemple un entier, un réel ou une référence à une instance, ou bien une liste, mais aussi une description de la ou les valeurs de l'attribut. Au niveau d'une instance, cette description peut être utilisée pour déterminer la valeur par filtrage.

  - La facette $si_besoin permet d'associer des méthodes de calcul, au sens large, des valeurs et réalise l'attachement procédural. Les méthodes sont elles-mêmes décrites par des schémas, mais sont programmées dans un langage algorithmique tel que Lisp ou Pascal.

  - La facette $defaut permet d'associer uen valeur par défaut à un attribut. Cette facette ne sera effectivement retenue qu'en

l'absence d'autres informations et lorsque les différents moyen d'obtention de la valeur auront échoué. Ces à travers cette facette que se traitent les exceptions.

- les facettes «réflexes» : si_ajout, si_modif et si_supprim, sont aussi suivies de la description d'une proédure qui sera appelée, respectivement en cas d'ajout (dans une liste), de modification et de suppression d'une valeur d'un attribut dans une instance. Ces facettes permettent fondamentalement de maintenir la cohérence de la base d'instances en propageant convenablement toute modification sur l'une d'entre elles.

- les facettes de restriction de type : $a_verifier, $domaine et $intervalle.

  - La facette $domaine permet de spécifier une liste de valeurs admissibles d'un attribut.

  - La facette $intervalle permet pour les types simples, de préciser l'intervalle des valeurs admissibles.

  - Grâce à le facette $a_verifier, un prédicat quelconque peut être attaché à un attribut. Ce prédicat doit être vérifié pour toute valeur de l'attribut.

- Les facettes de contrôle : $si_echec et $si_succes, permettent la description des actions à entreprendre en cas respectivement d'échec et de succès d'obtention d'une valeur de l'attribut auquel elles sont attachées.

- Les facettes décrivant les modes de saisie et de représentation externes des instances. Elles permettent de passer de leurs représentation externes à la représentation interne et inversement.

3.3. Représentations internes et externes des instances

Un schéma inclut également la connaissance qui permet de définir la représentation sous forme externe des ses instances, par exemple l'impression. La méthode d'impression est en fait répartie entre les attributs, chaque attribut possédant sa propre méthode décrite à l'aide de facettes particulières. A chacune des facettes doit être associée une fonction, décrivant totalement ou partiellement l'impression de l'attribut pour une instance considérée. Le schéma EQUATION, enrichi de ces facettes, pouvant s'écrire :

```
(EQUATION
        ($EQUATION ($pour_impr (MEMBRE-G MEMBRE-DROIT)))
        (VARIABLES ($liste_de VARIABLE))
        (MEMBRE-G  ($un VAR-ETAT))
        (MEMBRE-D  ($un EXPR-ARITH)
                   ($avant_impr PR-EGAL)
                   ($apres_impr PR-PTVIRG)))
```

où PR-EGAL et PR-PTVIRG sont des fonctions qui impriment respectivement un signe égal et un point virgule.

On a ici un exemple d'attachement procédural. La fonction PR-EGAL est appelée lorsque l'attribut MEMBRE-D doit être imprimé. De même, la fonction PR-PTVIRG sera appelée pour terminer cette impression. MEMBRE-G et MEMBRE-D faisant appel à d'autres schémas, leur impression fait également appel à des méthodes d'impression. D'autres facettes sont définies pour permettre à une instance de se représenter graphiquement ou d'être lue sur un support externe.

L'attribut $EQUATION, quant à lui, rassemble la connaissance sur le schéma en tant que tel, par opposition à la connaissance attachée aux attributs.

3.4. L'héritage

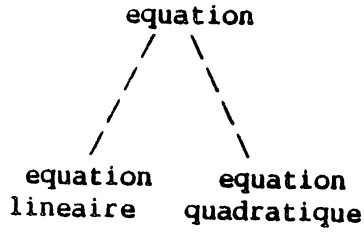
Tout schéma s'inscrit dans une hiérarchie taxinomique par une relation de spécialisation. Il est «l'ancêtre» d'un ou plusieurs schémas plus spécifiques auquels il transmet toute sa connaissance, sauf si ses «héritiers» la redéfinissent en tout ou en partie. Les liens de spécialisation sont décrits à l'aide d'un attribut spécial : $SORTE-DE, qui décrit la connaissance commune à toute les instances d'un schéma.

Par exemple pour le schéma EQUATION-LINEAIRE :

```
(equation-lineaire
        ($sorte-de (valeur (equation)))
        (membre-d
                ($a-verifier (lineaire ?membre-d
                                       ?membre-g))))
```

Il hérite du schéma EQUATION les attributs MEMBRE-G, MEMBRE-D et VARIABLES dans leur intégrité et rajoute à l'attribut MEMBRE-D une restriction de linéarisation à travers la facette $A-VERIFIER, dont la valeur est un prédicat Lisp, nouvel exemple d'attachement procédural.

De même il est possible d'introduire un schéma EQUATION-QUADRATIQUE comme une spécialisation du schéma EQUATION. Ces trois schémas forment une hérarchie partielle (incomplète) comme le montre la figure suivante :



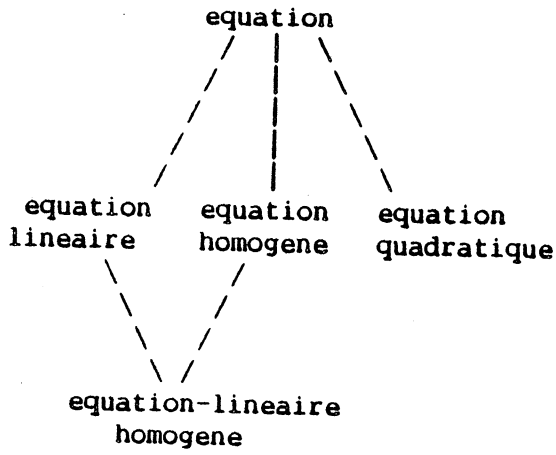
- Exemple de hiérarchie de spécialisation -

Les schémas EQUATION-LINEAIRE et EQUATION-QUADRATIQUE peuvent à leur tour posséder des schémas plus spécifiques, auxquels il transmettent leurs connaissances par héritage.

Un même schéma peut en fait hériter simultanément de plusieurs schémas plus génériques . Par exemple le schéma EQUATION-LIN-HOMOGENE hérite à la fois de la connaissance sur EQUATION-LINEAIRE et sur EQUATION-HOMOGENE. La structure de ce schéma pourrait être :

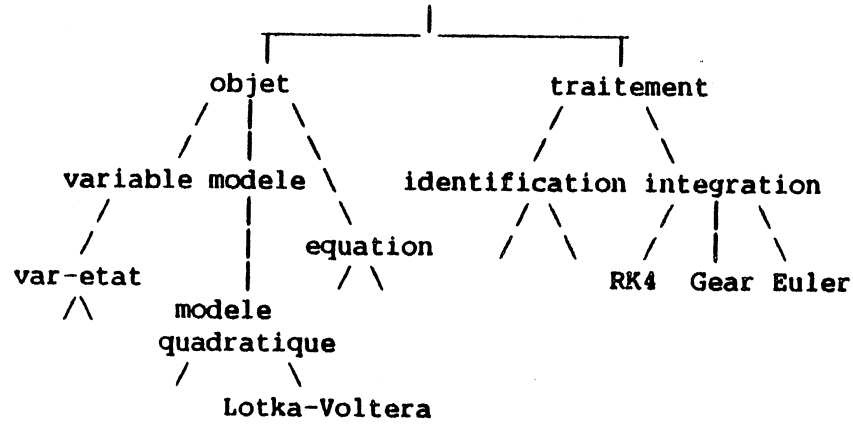
```
(equation-lin-homogene
        ($sorte-de (valeur (equation-lineaire
                            equation-homogene)))
        (   ...
            ))
```

Ce mécanisme, dit d'héritage-multiple, suppose la définition d'une propriété d'héritage, par exemple du "bas" vers le "haut", puis de gauche à droite dans la hérarchie, afin de résoudre d'éventuels conflits. La figure suivante montre la hiérarchie des différentes équations :



- Exemple de hiérarchie multiple -

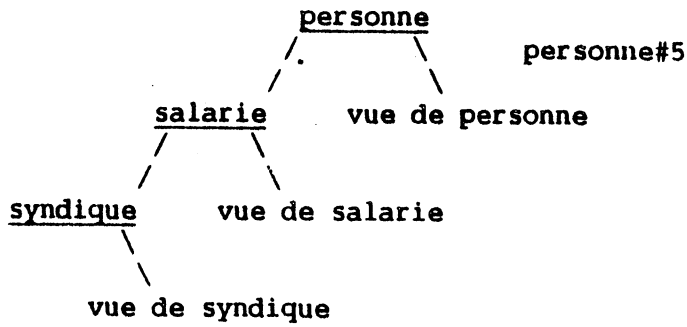
Les schémas permettent de décrire les objets d'un univers et leurs connaissances associées. Ils permettent aussi de prendre en compte les traitements et méthodes qui les manipulent. A titre d'exemple, la figure suivante représente un début de hiérarchie dans un système d'aide à la modélisation en dynamique des populations utilisant un formalisme continu à base d'équations différentielles.



3.5. L'instanciation et les vues

Le mécanime fondamental d'inférence dans une représentation centrée objets est l'instanciation. Il s'agit de construire ou de compléter une instance, c'est à dire d'obtenir des valeurs des attributs du schéma de classe correspondant. Lorsqu'un schéma est instancié, non seulement ses propres attributs peuvent recevoir une valeur, mais également tous les attributs dont il hérite. Une vue est ainsi l'ensemble des valeurs des attributs qu'un schéma possède en propre. L'instance est l'union de toutes ces vues.

Par exemple :



Si la valeur d'un attribut est définie au niveau d'une classe, toutes les instances de cette classe et des classes plus spécifiques héritent, sans exception de cette valeur.

3.6. Le filtrage et la classification

La facette $valeur peut être suivie, dans un schéma de classe, d'une suite de
filtres, c'est à dire de schémas de classe, spécialisations de schémas existants. Ces filtres sont des descriptions des instances susceptibles de constituer la valeur de l'attribut. Ils sont appliqués séquentiellement, chaque filtre portant sur l'ensemble des instances du schéma qu'il spécialise. Dans le cas où la facette de typage est $un, la première instance qui satisfait un filtre et les contraintes de l'attribut est retenue. Dans le cas de la facette $liste_de, toutes les instances trouvées admissibles sont retenues et leur liste constitue la valeur de l'attribut.

Entre le schéma que le filtre spécialise et le schéma typant l'attribu doit exister une relation de spécialisation. La situation où le filtre est dominé par le type ne pose pas de proble particulier car il est alors assuré que les instances satisfaisant le filtre satisfont également le type. Dans le cas contraire, il faut savoir si les instances satisfaisant le filtre peuvent être vues comme des instances du schéma définissant le type. L'instance est donc «descendue» d'un schéma vers l'autre, créant en cas de succ autant de vues nouvelles qu'il existe de niveaux entre les deux schémas. Ce mécanisme de
classification accroît la structuration de la base et son contenu sémantiqu

L'énoncé d'un filtre fait référence aux valeurs de l'attribut du schéma où il apparait par l'intermédiaire de variables. Les variables sont introduite par la facette $variable attachée un attribut et suivie d'un nom. Lors de l'instanciation, dés qu'un attribut recoit une valeur, celle-ci est affectée à la variable correspondante si elle existe. L'apparition d'une variable dans un filtre a deux interprétations différentes : soit la valeur doit être connue au moment du filtrage et constitue un élément du filtre, soit la valeur est déterminée par le résultat du filtrage, créant ainsi un effet de bord. Le choix de l'interprétation retenue est indiquée par le caractère préfixant le nom de la variable, respectivement ! et ?.

Par exemple :

```
   (PERSONNE
      ($PERSONE       ($variable ?LUI))
  ...
      (GRAND_PERE_DE
($liste_de PERSONNE)
($valeur
      (PERSONNE
      (A_POUR_PERE ($valeur (PERSONNE
      (A_POUR_PERE
        ($variable !LUI) ]
```

La valeur associée à la facette $valeur est un exemple de filtre imbriqué; la variable LUI a pour valeur l'instance de PERSONNE dont cherche la valeur de l'attribut GRAND_PERE_DE.

En cas d'échec de mise en correspondance d'une instance et d'un filtre, le dernier choix effectué est remis en cause (par exemple, la dernière valeur retenue pour un attribut est écartée) et un nouveau choix est effectué (une autre valeur d'attribut est choisie). L'obtention d'une valeur pouvant conduire à un processus de filtrage et les filtres pouvant être imbriqués, l'instanciation d'un schéma peut devenir complexe et coûteuse. Cependant, les nombreuses contraintes introduites dans les schémas permettent de limiter sévèrement les solutions possibes et les retours en arrière.

3.7. L'inférence procédurale

Les valeurs des attributs peuvent être calculées, c'est à dire résulter de l'exécution d'un traitement algorithmique externe. Les traitements sont eux mes décrits par des schémas. Leurs attributs sont les paramètres d'entrée et de sortie accompagnés de leurs contraintes respectives. Une instance d'un tel schéma est associée à une exécution effective du traitement. Syntaxiquement, le recours à un traitement ne diffère pas d'un filtre placé derrie la facette $si_besoin, mais l'interprétation en est différente.

La pemière étape consiste à filtrer l'ensemble des instances du schéma traitement. Si aucune instance adéquate n'est trouvée, le traitement est exécuté, calculant les paramètres de sortie. Si leurs valeurs sont admissibles, une instance est crée. Par effet de bord, l'attribut auquel est attaché le traitement par la facette $si_besoin peut recevoir sa valeur.

Par exemple, les schémas de l'exemple précédent, contiennent des connaissances sur les méthodes, les objets mathématiques et sur le domaine de modélisation. Les schémas associés aux traitements, tels que RK4 ou GEAR, décrivent leurs entrées (objets sur lesquels ils s'appliquent, paramètres des méthodes) et leurs sorties (résultats produits) ainsi que leurs conditions d'application.

Une instance d'un schéma traitement par exemple RK4, correspomd à la réalisation de ce traitement, en l'occurence l'application de la méthode d'intégration à une certaine instance d'un modèle, par exemple de type LOTKA-VOLTERA, avec un pas, un intervalle de temps et une erreur maximum de données.

3.8. exploitation de la base de connaissances

Il existe deux stratégies fondamentales d'exploitation de la base. La première laisse l'initiative à l'utilisateur qui formule des interrogations sous la forme de filtres, c'est à dire des schémas incomplets. L'algorithme de filtrage ramène l'ensemble des schémas ou des instances qui satisfont un filtre donné. L'utilisateur peut ainsi rechercher un algorithme possédant un certain nombre de caractéristiques. Le système lui renvoie la liste des algorithmes correspondants. Cette liste peut apparaître comme un menu si l'utilisateur désire exécuter l'un d'entre eux.

A l'aide de filtres plus ou moins sélectifs, l'utilisateur à ainsi la possibilité de naviguer dans la base et donc d'aborder le système sans connaissance a priori. Le même genre de recherche peut être mené sur les instances, s'assimilant alors à une recherche dans une base de données classique. Il est ainsi possible de rechercher une instance particulière en spécifiant un filtre qui décrit les relations qu'elle entretient avec d'autres instances ou toute autre particularité.

La deuxième stratégie laisse au système l'autonomie de résolution du problème en déterminant pour un objet donné la forme appropriée pour une certaine tâche. Ce problème se résoud en deux phases : identification de l'objet dans la hiérarchie des schémas, puis détermination de la méthode associée à laquelle le schéma doit être relié par une facette $un d'un attribut associé à la tâche. L'identification de l'objet s'effectue en parcourant la hiérarchie des schémas en partant d'un schéma connu. Chaque schéma est confronté à l'objet à l'aide d'un algorithme de filtrage («pattern-matching») qui ne se contente pas d'une réponse binaire mais renvoie une note évaluant la distance entre l'objet et le prototype et utilise également les contraintes introduites par les facettes $A-VERIFIER.

Un problème fondamental est celui du contrôle du processus d'identification. Il est en effet inutile de confronter l'objet aux types les plus fins de la hiérarchie si, pour ces types, aucune méthode particulière pour la tâche prévue n'est disponible. Dans ce cas, une méthode moins spécifique doit être employée. L'emploi de cette méthode est spécifiée à un niveau supérieur de la hiérarchie et c'est á ce niveau que le processus de reconnaissance doit s'arrêter. Cela signifie qu'il faut pouvoir contrôler ce processus aussi bien en cas de succés qu'en d'échec.

Ce contrôle de l'identification et du filtrage se réalise à l'aide des facettes $SI-SUCCES et $SI-ECHEC dont les valeurs sont des fonctions Lisp contenant les actions à exécuter respectivement en cas de succés et d'échec du filtrage sur le schéma.

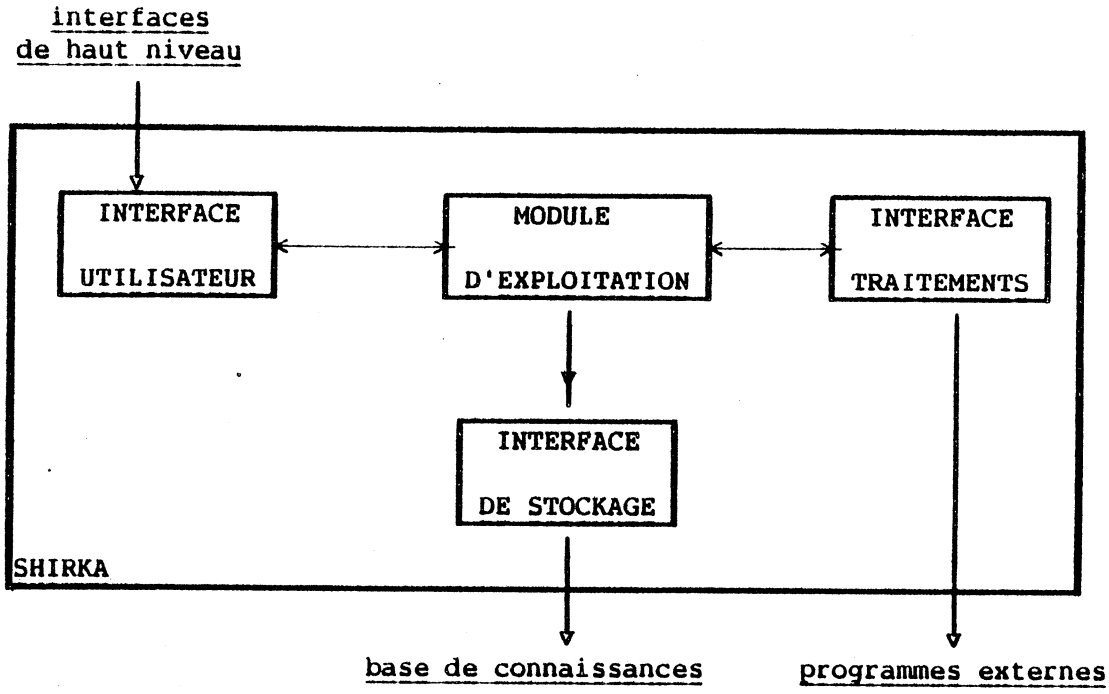
Si au cours de la recherche par filtrage, des valeurs d'attributs sont inconnues, elle peuvent être calculées si à cet attribut a été associée une facette $SI-BESOIN. La valeur de cette facette est une fonction qui renvoie, aprés exécution, la valeur demandée. Parallèlement, les valeurs des facettes $SI-AJOUT et $SI-MODIF sont également des fonctions qui sont exécutées lorsqu'une valeur est respectivement ajoutée et modifiée dans une instance du schéma où elles apparaissent. Elles assurent ainsi la propagation des modifications dans la base.

4. Architecture logicielle de SHIRKA

L'essentiel des fonctionnalités de SHIRKA, c'est à dire l'exploitation de la base de connaissances et le contrôle du raisonnement, sont réalisées par un module de traitement : le module d'exploitation qui constitue le cœur même de SHIRKA. Pour assurer les tâches de communication avec l'extérieur, trois interfaces, appelées modules de communication, sont à la dispostion du module d'exploitation. Il s'agit :

- de l'interface utilisateur,
- de l'interface des traitements
- et de l'interface de stockage.

L'architecture complète de SHIRKA est décrite par la figure suivante :



- ARCHITECTURE LOGICIELLE DE SHIRKA -

Le rôle des modules de communication étant de décharger au maximum le module d'exploitation des tâches de gestion des données externes. Par exemples :

- la recherche d'un ensemble d'instances d'un certain schéma, qui répondent à un filtre précis, est du domaine de l'interface de stockage;

- lorsqu'une méthode de calcul est trouvée, l'interface des traitements aura à sa charge la transmition des valeurs des paramètres, le lancement du programme externe correspondant à la méthode et le contrôle du bon déroulement de cette méthode.

- L'impression des résultat d'une intégration numérique sous forme de courbe, s'effectuera sous le contrôle de l'interface utilisateur, qui se chargera de la traduction des schémas en termes d'objets graphiques.

4.1. Le module d'exploitation

L'exploitation de la base de connaissances et le contrôle du raisonnement sont les tâches essentielles du module d'exploitation. Schématiquemnt, le travail de ce module peut être décompoé de la manière suivante :

1) Prise en compte d'une requête de l'utilisateur, exprimée sous forme de filtres (délivrée par l'interface utilisateur).

2) Pour un filtre donné, accés à la base de connaissances (par l'intermédiaire de l'interface de stockage) pour récupérer l'ensemble des instances dont les valeurs des attributs coincident avec les valeur données explicitement dans le filtre.

3) Déclenchement des mécanismes d'héritage pour déterminer les valeurs manquantes des attributs.

4) Déclenchement des mécanismes d'inférence procédurale pour l'obtention des valeurs devant être fournies par calcul (exécution des programmes sous le contrôle de l'interface des traitements).

5) Répétition des actions 2 à 4 jusqu'à obtenir les instances qui répondent entièrement à la question posée.

6) Sortie des instances trouvées.

4.2. L'interface utilisateur

La communication entre l'utilisateur et le module d'exploitation peut s'établir de différentes façons :

- A l'aide d'un éditeur de schémas et d'instances pour permettre la construction de schémas, la navigation dans la base, l'obtention de valeurs d'attributs et la soumission de requêtes sous forme de filtres.

- A travers une interface en langue naturelle.

- Par utilisation d'un système graphique pour la sortie de résultats sous formes de courbes ou de diagrammes.

- Sous le contrôle d'un gestionnaire de tableaux pour la saisie ou l'édition de données numériques.

- ...

Le rôle de l'interface utilisateur n'est pas de fournir tout ces services (sauf l'éditeur de schémas qui sera intégré au système SHIRKA), mais de proposer un ensemble de primitives permettant l'écriture d'interfaces de haut niveau. Les primitives offertes seront disponibles soit sous forme de fonctions Lisp, soit sous forme de procédures écrites en langage C. Elles permettront, en autres, la construction de schémas, l'accés aux valeurs des attributs des instances et la soumission de filtres sous formes de schémas incomplets.

L'interface utilisateur assure la transformation de la représentation interne des schémas (dans le module d'exploitation) en structures de données conformes à la vision externes des objets de l'utilisateur. Cette transformation s'effectuant à l'aide des renseignements fournis, d'une part dans les facettes d'entrée et de sortie des schémas et d'autre part, dans les arguments des primitives l'interface utilisateur.

Lors d'une session SHIRKA, il peut exister plusieurs systèmes de communication externes connectés au module utilisateur. Au lancement de la session, l'utilisateur se trouve sous l'éditeur de schémas. Il peut effectuer tout ses travaux sous celui-ci ou, par un mécanimes de redirection des entrées et des sorties, diriger les résultats vers un système graphique, par exemple, ou saisir des données à l'aide d'un tableur.

4.3. L'interface traitement

Cette interface assure la mise en œuvre el le contrôle de l'exécution des programmes utilisateurs, qui correspondent à des méthodes de calcul décrites dans des schémas traitements. Ce module gère un ensemble de fonctions Lisp ou de programmes exécutables écrits dans un autre langage et développés à l'extérieur de SHIRKA.

Lors de la construction d'un schéma traitement, sous l'éditeur de schéma, l'utilisateur donnera les informations necéssaires pour réaliser l'appel et la passation des paramètres au programme externe. Ces informations, seront stockées dans une banque de programmes et seront exploitées par l'interface des traitements, lors de l'obtention de la valeur d'un attribut par attachement procédurale.

Dans le cas d'une méthode de calcul écrite en Lisp, le corps de la fonction associée sera directement évaluée dans l'environnement Lisp de l'interface des traitements externes. Elle recevra en paramètre la valeur de la liste qui correspond à l'instance du schéma traitement en cours d'exploitation et pourra, éventuellement, la modifier. L'utilisateur aura aussi la possibilité, à l'intérieur de la fonction Lisp, d'utiliser les primitives du module utilisateur pour émettre des requêtes à destination du module d'exploitation. Aprés évaluation de la fonction, l'instance du schéma traitement sera retournée au module d'exploitation.

Dans Le cas où le traitement externe est un programme utilisateur écrit dans un autre langage, l'évaluation de la méthode s'effectuera de la maniére suivante :

1) Traduction des valeurs des attributs de l'instance-traitement en structures de données conformes au langage utilisée.

2) Exécution du programme externe.

3) Réception des résultats du programme et mise à jour de l'instance du schéma traitement en fonction des résultats transmis.

Dans la version actuelle du système SHIRKA, développée sous le système d'exploitation UNIX (sur un ordinateur DEC VAX 11/780), le fonctionnement de l'interface des traitement est le suivant :

* Lors de la création d'un schéma traitement, à partir des informations sur le langage dans lequel est écrit le programme et sur ses paramètres, une procédure en langage C est automatiquement géneée et associée au programme externe par l'éditeur de liens.

* Cette procédure a pour rôle la récupération des paramètres, l'appel procédural du programme utilisateur et de renvoyer les résultats à l'interface des traitements.

* Lors de chaque appel à l'interface des traitements, pour obtenir la valeur d'un attribut dépendant d'un calcul, un processus est crée sur le programme exécutable correspondant à la méthode de calcul choisie.

* Pour assurer la transmission des paramètres et la réception des résultats, une communication par tube sera établie entre l'interface des traitements et la procédure C générée.

* Ce tube servira aussi, pour transmettre les données intermédiaires dont pourraient avoir besoin la méthode de calcul lors de son exécution.

Ce principe d'exécution et de communication permet ainsi l'évaluation en parallèle de plusieurs méthodes de calcul et de les faire dialoguer pour échanger des données. De plus, ce principe n'impose aucune contrainte à l'utilisateur, dans la programmation de ses méthodes externes ou pour utiliser des bibliothèques de calculs existantes.

4.4. L'interface de stockage

Ce module est en fait le système de gestion de de la base de connaissances de SHIRKA. Il est constitué de l'interpréteur du modèle SHIVA qui prend à sa charge la gestion complète des schémas et des instances sur les support externes. Tous les accès à la base de connaissances s'effectuent depuis le module d'exploitation par utilisation des fonctions de manipulation de données du modèle SHIVA.

L'étude du système de gestion de la base de connaissance du système SHIRKA a montré l'inutilité de sa réalisation directement à l'aide des fonctions d'entrée-sortie du langage Lisp. Les caractéristiques de la gestion mémoire des objets Lisp et la pauvreté des services offerts par ce langage pour la gestion des données sur support externe, ont entrainé la recherche ou la définition d'un système de gestion de données externes non intégré à l'application. La possible utilisation d'un système de gestion de bases de données fut ainsi examinée.

Par la qualité des services offerts et la richesse du modèle relationnel, les systèmes de gestion de bases de données relationnelles furent retenus. Malgré certains aspects trés intéressants du modèle entités-relation, notamment sa capacité à prendre en compte des données structurées, la possibilité de gérer la hiérarchie des schémas et les liens entre les instances, mais surtout la correspondance intuitive de ce modèle avec le modèle de données de SHIRKA, ce modèle ne fut toutefois pas retenu. Ce rejet tient surtout à la lourdeur et à la complexité des services offerts par les langages de manipulation de données des modèles entités-relations. Au contraire, l'accès sélectif aux instances par utilisation des opérateurs de l'algèbre relationnelle a été jugé beaucoup plus intéressant.

Un autre critère pour le choix du modèle de base de données a été l'étude de l'interface entre le système de gestion de bases de données et le langage Lisp dans lequel est programmé SHIRKA. A ce titre, la possibilité d'exprimer et de gérer directement en Lisp des arborescences d'opérateurs relationnels ainsi que les concepts mathématiques sur lesquels repose le modèle relationnel ont été déterminants pour le choix de ce modèle.

Cependant, le couplage d'un système relationnel existant avec le langage Lisp présente plus ou moins de difficultés selon les services d'interface avec d'autres langages offerts par l'interpréteur Lisp choisi. Dans un souci de portabilité et dans le but de disposer d'un système homogène, nous avons envisagé de spécifier et d'écrire les principales fonctionnalités du système relationnel en langage Lisp. Le système ainsi défini : SHIMER, malgré l'introduction de nouvelles facilités dues au langage Lisp (filtrage sur les index, recherche associative et gestion de listes comme valeurs possibles d'un attribut), ne pouvait pas être utilisé directement pour la gestion des objets du système SHIRKA. L'introduction d'un mécanisme de traduction shémas-relations était nécessaire.

L'ajout d'un niveau supplémentaire entraine des pertes d'effacité mais aussi une sous exploitation du modèle relationnel. D'autre part, certaines propriétés du modèle de données SHIRKA, en particulier la gestion des facettes et l'attachement procédural, posent quelques problèmes pour leur prise en compte par le système relationnel et nécessitent la programmation de macro-fonctions Lisp à évaluer lors de la récupération des données de la base. Dans le but d'obtenir un couplage des plus efficaces entre le modèle de représentation de la connaissance et le modèle relationnel des données de SHIMER, une intégration de certaines caractéristiques du modèle SHIRKA, dans le sytème relationnel, doit être envisagée.

C H A P I T R E 2

- EVOLUTION DES BASES DE DONNEES -

1. Introduction

Le concept de bases de données est apparu vers les années 62/63. Les difficultés rencontrées dans la gestion d'un univers, à l'aide de plusieurs fichiers, ont favorisé l'émergence de ce concept. L'objectif principal d'un modèle de base de données est d'assurer une indépendance maximale entre la description logique des données et leur représentation physique.

Depuis le milieu des années 60, plusieurs modèles de bases de données sont apparus. On différenciera principalement trois grandes classes de modèles de données : les modèles hiérarchiques, les modèles réseaux et les modèles relationnels.

En 1970, CODD [Cod70], définit le modèle relationnel de données, avec pour objectifs principaux, une plus grande indépendance des données par rapport aux structures physiques de représentation. Le modèle relationnel offre une vision des informations sous forme de tableaux de valeurs sans préjuger de la façon dont les données sont stockées dans la machine. En outre le modèle relationnel repose sur des bases théoriques solides notamment la théorie des ensembles et le calcul des prédicats.

Les facilités offertes par un système relationnel de gestion de base de données [Adi82,Gar84], sont les suivantes :

- proposer des schémas de données faciles à utiliser;
- améliorer l'indépendance logique et physique;
- mettre à la disposition des utilisateurs des langages de haut niveau;
- optimiser les accès à la base;
- améliorer l'intégrité et la confidentialité;
- prendre en compte une variété d'applications;
- et fournir une approche méthologique dans la construction des schémas.

2. Problèmes et nouvelles applications

Le développement de nouvelles applications dans des dommaines tels que la bureautique, la CAO, l'économie ou encore dans les systèmes d'aide à la décision, a montré une certaine insuffisance des modèles de bases de données à prendre en compte des structures de données complexes et à intégrer la sémantique associée aux objets manipulés par ces applications.

Les bases de données ont été largement utilisées dans des domaines où l'information est simple à utiliser, statique et peu structurée. Les informations pouvaient se comprendre simplement en terme de nombres et de chaînes de caractères. La gestion est un domaine où les bases de données ont beaucoup apporté et à l'heure actuelle, nombreux sont encore les systèmes de gestion de bases de données conçus et commercialisés pour ce domaine.

En effet, les systèmes actuels sont capables de gérer de grandes quantités de données et en cela ils remplissent parfaitement leur rôle. La gestion classique se contente de modèles simples et de systèmes de stockage rapides et assurant une certaine cohérence des données manipulées.

Depuis quelques années on envisage l'extension des bases de données à des domaines où l'information n'a pas tout à fait ces caractéristiques. En CAO, où l'information tout en étant volumineuse revêt divers aspects : textes, dessins et formules mathématiques. En bureautique, on l'on manipule des documents structurés, des formulaires. En génie logiciel, où l'on manipule des programmes. En économie, on s'intéressera à la représentation du temps (séries chronologiques) et à ses relations avec les objets manipulés (équations macroéconomiques). On peut citer aussi la cartographie, la biologie, le traitement d'images, etc... Tous ces domaines demandent des systèmes de stockage gérant de grands volumes de données. L'intégrité, la confidentialité et la cohérence des données sont aussi leurs préoccupations. Et naturellement ces domaines se sont intéressés aux bases de données.

Initialement, ces domaines ont développé leurs propres logiciels, prenant en compte les problèmes spécifiques aux applications et proposant des outils, des langages et des méthodes pour résoudre leurs problèmes. Paradoxalement, faute de pouvoir bénéficier pleinement des fonctionnalités des systèmes de gestion de bases de données, les logiciels développés dans ces domaines prennent en compte entièrement la gestion, le stockage, la manipulation et la représentation des informations sur les supports externes. Ainsi, ces logiciels ont été amenés à utiliser les systèmes de gestion de fichiers classiques. Les données sont alors dispersées dans plusieurs fichiers «ad hoc» et le problème de l'indépendance données logiques et données physiques se

pose de nouveau.

En conclusion, ces nouvelles applications ont besoin de modèles de données plus riches [Oli84], prenant en compte d'avantage d'aspects sémantiques de l'application et elles demandent des outils de manipulation plus puissants en raison de la complexité des données. Le groupe BD3 [BD383], suggère des travaux pour le développement de nouvelles possibilités. Les différents niveaux indiqués sont :

1) Expérimentation dans de nouveaux domaines tels que la conception assistée par ordinateur, la bureautique, etc,...

2) Prise en compte des résultats de recherche acquis, en particulier, ceux des modèles et langages de l'intelligence artificielle.

3) Recherche de bases pour améliorer les résultats théoriques disponibles, par exemple, sur la dérivation d'informations implicites, sur les contraintes d'intégrité, de manière à prendre en compte les modèles réels.

4) Développement de solutions efficaces sur le plan informatique à ces problèmes.

5) Etude d'architectures de matériels intégrant les interfaces d'entrée et de sortie relatives aux données multi-média et les modes de stockage des données.

3. Extensions du modèle relationnel

Le modèle relationnel introduit par CODD [Cod70], a largement démontré sa capacité à résoudre le problème de l'indépendance données logiques et données physiques et à offrir des langages de requêtes de haut niveau. Toutefois, les problèmes liés à l'intégration des données textuelles, de documents et d'images dans les systèmes de bases de données, le développement de nouvelles applications désirant utiliser les modèles et les techniques des bases de données, ainsi que les problèmes liés à l'insuffisance du modèle relationnel au niveau sémantique a donné lieu à de nombreuses études. Citons :

- les bases de données et les nouvelles applications [Ria83,Pal85,Leo85,Mai85],

- les modèles de bases de données généralisés et la gestion de documents dans les systèmes de bases de données relationnels [Mac79,Gut83,Oli84],

- l'extension des modèles de bases de données pour une meilleure prise en compte de la sémantique [Che76,Cod79,Rou85,Oli85,Val85],

- les modèles de bases de données relationnels non normalisés et le

traitement des informations incomplètes [Bid84,Yaz85,Gra85,Lip85],

- la gestion des objets complexes dans les bases de données [Dem85,Vel85,Rie85,Chr85],

- les bases de données déductives et les systèmes experts [Goe83,Mad85,Sim85,Met85,Sto85,Ban85].

Pour notre propos, deux extensions du modèle relationnel nous semblent intéressantes et sont présentées ici :

1) La prise en compte des valeurs nulles ou non-applicables dans le modèle relationnel. En effet, la prise en compte des données incomplètes d'un problème [Bon84], est une caractéristique fondamentale des programmes de l'intelligence artificielle. Etant donné que l'on cherchera à fournir des solutions à un problème même si toutes les données ne sont pas disponibles au moment de la résolution du problème.

2) La remise en cause de la structure de table pour la vision externe des données utilisée par le modèle relationnel. En effet, cette structure, trés efficace pour de nombreuses applications, où les données sont généralement atomiques, nombres ou chaînes de caractères, n'est plus du tout satisfaisante pour la gestion des données d'une base de connaissances, où les informations sont fortement structurées, interdépendantes entre elles et hiérarchisées.

3.1. La prise en compte des valeurs nulles

Siklossy et Lauriere [Lau82], proposent de lever certaines restrictions du modèle relationnel et de l'étendre afin d'inclure des valeurs nulles (des valeurs inconnues ou non-applicables). Cette étude se base sur la constatation que souvent certaines valeurs peuvent ne pas être connues dans la base de données.

Par exemple, la valeur «nom de l'épouse» pour une personne non mariée n'est pas applicable. Dans un système de gestion de base de données classique, pour prendre en compte cette possibilité, il faudrait définir deux relations : une pour les personnes mariées, une autre pour les non mariées.

Le modèle relationnel ainsi étudié est dénommé modèle L2 et un système pour la résolution de problèmes, appelé ALICE, basé sur ce modèle est brièvement présenté ci-dessous.

LE MODELE L2:

Les valeurs des attributs dans un n-uplet, admises dans le modèle L2, sont les suivantes :

1) Les valeurs scalaires : "bleu", 69, "tomate",...

2) Des ensembles de valeurs ou des intervalles de valeurs : {bleu, vert, rouge}, {22..35}

3) Des valeurs non significatives : notées "NR". Un attribut peut avoir plusieurs valeurs non significatives. Par exemple, «nom de l'épouse» peut avoir la valeur "NR1" pour exprimer «jamais marié» ou "NR2" pour exprimer «épouse décédée».

4) Les valeurs inconnues sont représentées par "?". Ces valeurs peuvent être indéxées, par exemple : "?5".

5) Des dépendances fonctionnelles (DF) d'un ensemble d'attributs, vers un autre ensemble d'attributs d'une relation,

6) Les clés, qui sont un cas spécial de réalisation de dépendances fonctionnelles.

EXEMPLES

1) Soit la relation :

Objet	Age	N# Dept	Date Emb.	Salaire
	0..+INF	{1,2,3,4,5}	70..80	0..+INF
x1	60..70	{1,2,3,4,5}	73..75	10000
x2	52..56	{2}	72..76	0..20000
x3	55	{3}	70..71	0..10000
x4	0..+INF	{2,3}	72..74	12000
x5	32	{4}	75	0..+INF

A la question :

(N# Dept dans {2,3})
et
(((Salaire < 20000) et (Date Emb. >= 72))
ou
((Age > 50) et (Salaire < 15000)))

Le système ALICE déterminera que les n-uplets x3 et x4 satisfont à la question, que x2 peut satisfaire à la question et que x1 et x5 ne satisfont pas à la question.

2) Soit la relation :

Nom	Salaire	Etat	Nb. Enf.	N# Dept
"?"	15000..18000	marie	{3,4,5}	{2,3,4}
brown	18000	marie	{2}	{2}

A la question :

((N# Dept = 3) et (Nom = jones))
et
((Salaire < 15000) ou (Nb. Enf. > 2))

ALICE déterminera que "?" peut répondre à la question et que "brown" n'y répond pas.

CAPACITE DE REDUCTION

Pour une relation donnée, ALICE «réduira» d'abord la relation en éliminant les n-uplets redondants et cherchera ensuite des valeurs ou des intervalles pour les valeurs inconnues ou non-applicables.

Considérons la relation :

n-uplet	A	B	C	D
1	(0 ou 4)	?1	c3	?2
2	(0 ou 4)	?3	c4	?4
3	(0 ou 6)	?5 # ?1	c3	?6
4	?7	?8	c5	5
5	2	?9	c4	?10
6	?11	b1	c1	?12
7	1	b1	c1	?12
8	?19	?14	?15	?20
9	3	b1	?16	(1,2,3)
10	?17	b1	c2	?18

Avec les contraintes suivantes :

(I) Clés : A et (B,C)
(II) D.F. : C --> D
(III) Pas plus de 2 C pour 1 B
(IV) Au moins 2 n-uplets différents avec C = c1
(V) 0 <= A <= 10 (A entier)
(VI) 0 <= D <= 5 (D entier)
(VII) A >= 2D

Remarques :

- La valeur : (0 ou 4) de l'attribut A, signifie que pour le n-uplet 1, cet attribut peut avoir la valeur 0 ou la valeur 4 uniquement.

- La valeur : ?5 # ?1 de l'attribut B, signifie que pour le n-uplet 3, cet attribut a une valeur inconnue : ?5, mais que celle-ci est doit être différente de la valeur inconnue ?1.

- Il existe des valeurs inconnues dans la clé A (dans les n-uplets 4, 6, 8 et 10) et de même dans le composant B de la clé (B,C) (dans les n-uplets 1 à 5 et le n-uplet 8).

- Le n-uplet 8 est entièrement inconnu.

- L'indexation des valeurs inconnues est utilisée pour indiquer l'égalité de valeurs inconnues (comme dans 6.D et 7.D), ou pour indiquer l'inégalité. La contrainte de la ligne 3 : "?5#?1" signifie que : 3.B est égal à ?5 mais doit être différent de ?1.

En utilisant les contraintes énoncées, le système ALICE réduira la relation de la manière suivante :

n-uplet	A	B	C	D
n1	(0 ou 4)	?1 # b1	c3	?2=(0,1,2)
n2	(0 ou 4)	?3 # b1	c4	?4=(0,1)
n3	6	?5 # ?1	c3	?2=(0,1,2)
n4	10	?8 # b1	c5	5
n5	2	?9 # b1	c4	?4=(0,1)
n7	1	b1	c1	0
n8	?19	?14 # b1	c1	?12
n9	3	b1	c2	1

Aux Questions suivantes, ALICE répondra :

QUESTIONS	REPONSES	
n-uplet avec :	surement :	peut être :
D = 5 ?	n4	n8
B = b1 ?	n7 , n9	aucun
B = b2 ?	aucun	n1,n2,n3,n4,n5,n8
B = b2 et D = 3 ?	aucun	n8
D = 0 ?	n7	n1,n2,n3,n5,n8
A = 0 ?	aucun	n1,n2

CONCLUSION

Le modèle de base de données L2 est une extension naturelle du modèle relationnel de base de données. Il permet des valeurs nulles dans tous les champs, y compris la clé, la répetition de n-uplets et une grande variété de contraintes.

Le système ALICE effectue des réductions dans les relations de la base de données, recherche les contradictions et tente de répondre au mieux au questions posées. Il est ainsi possible de modéliser des situations qui ne pouvaient l'être avec le modèle relationnel normalisé.

3.2. Le concept de relation non normalisée

Les problèmes liés à la prise en compte de nouvelles structures de données afin de permettre l'intégration de données textuelles, de documents, d'images, ainsi que les problèmes liés à l'insuffisance du modèle relationel au niveau sémantique, sont à l'origine du concept de relation non normalisée.

3.2.1. Les relations «enchassées» : le modèle NF

Makinouchi est un des premiers à proposer une définition décrivant une relation non sous première forme normale [Mak77]. Cette définition est obtenue de façon classique en autorisant comme valeurs de n-uplets :

- soit des valeurs atomiques : entiers, réels ou chaînes de caractères;
- soit des ensembles de n-uplets : des relations dites enchassées "."

Exemple de relation enchassée :

Num. Emp.	Noms Enfants	Historique	
		Annee	Salaire
54002	Arsene	1975	100000
	Mathias	1976	120000
	Gertrude		
69001	Arsene	1975	100000
	Aristide	1976	110000

- La relation EMP-NONR, non normalisée -

Num. Emp.	Noms Enf.	Sal. Hist. Annee	Sal. Hist. Salaire
54002	Arsene	1975	100000
54002	Arsene	1976	120000
54002	Gertrude	1975	100000
54002	Gertrude	1976	120000
54002	Mathias	1975	100000
54002	Mathias	1976	120000
69001	Arsene	1975	100000
69001	Arsene	1976	110000
69001	Arsistide	1975	100000
69001	Arsistide	1976	110000

- La relation EMP-NORM normalisée -

L'intersection d'une ligne et d'une colonne d'une relation non normalisée contient un ensemble de n-uplets lorsque la colonne est une relation colonne. Par exemple, un n-uplet de la relation EMP-NONR :

(69001, {Arsene, Aristide},
. {(1975, 10000), (1976, 12000)})

signifie que l'employé dont le numéro est 69001 a deux enfants dont les noms sont : Arsene et Aristide et, a gagné 100000F en 1975 et 110000F en 1976.

Le formalisme utilisé par Makinouchi ne permet pas d'assurer qu'une relation enchassée peut elle même être une relation enchassée. Il introduit aussi certaines propriétés «raisonnables» que doivent satisfaire les relations enchassées, particulièrement :

Propriété 1 :

• Si X est une relation non normalisée et Y une relation dépendante dans X (c'est à dire enchassée), alors tout n-uplet de Y ne peut exister ni en dehors de X ni indépendament de X.

Par exemple, la relation enchassée HIST-SALAIRE de la relation EMP-NONR est dénué de sens à l'extérieur de EMPLOYE; c'est à dire, on ne peut comprendre chacun des n-uplets de HIST-SALAIRE si un numéro d'employé n'est pas donné.

Propriété 2 :

• Un ensemble vide (dénoté Ø) peut être introduit dans le domaine des valeurs d'une relation enchassée.

Par exemple, si l'on insère le n-uplet :

(77001, Ø , Ø)

dans la relation EMP-NONR; cet n-uplet pourra être compris comme un nouvel employé n'ayant pas d'enfant et ne possédant pas d'historique du salaire.

Dans ce modèle (dénoté : NF), Makinouchi étend aussi la notion de dépendances fonctionnelles sur des relations non normalisées et introduit le concept de normalisation qui ne sous-entend pas un processus de décomposition mais à l'opposé un processus de «regroupement» de valeurs en utilisant des relations enchassées pour minimiser les anomalies de mises à jour de façon plus élégante.

Par exemple, si pour une certaine raison il faut supprimer tous les noms des enfants d'un employé; dans la relation EMP-NORM (la relation normalisée) il faudra supprimer tous les n-uplets contenant le nom des enfants de l'employé et par la même supprimer l'historique de son salaire; alors que dans la relation EMP-NONR (la relation non normalisée) il suffirait de remplacer la valeur de la colonne NOMS-ENF, pour l'employé considéré, par un n-uplet vide (denoté Ø).

3.2.2. L'algèbre des relations non normalisées

Jaeschke et Schek proposent dans [Sch82] une extension du modèle relationel en remettant en cause la première forme normale et définissent une relation non sous première forme normale (modèle denoté NF2) comme un ensemble de n-uplets pour lesquels les valeurs sur un attribut donné sont :

- soit des valeurs atomiques,

- soit des ensembles de valeurs atomiques,

- soit enfin des relations sous première forme normale.

Exemple de relation NF2 :

Auteurs	titre	prix	mots-clés
A1	T1	P1	D1
A2			D2
A2	T2	P2	D1
			D2
A1	T3	P1	D1
			D2
			D3

- La relation LIVRES, non normalisée -

Dans cette relation, les valeurs pour les attributs AUTEURS et MOTS-CLES sont des ensembles.

Pour modéliser les informations contenues dans la relation LIVRES tout en respectant la première forme normale, il faudrait définir les relations suivantes :

N#	titre	prix
1	T1	P1
2	T2	P1
3	T3	P1

- LIVRE -

N#	auteur
1	A1
1	A2
2	A2
3	A1

- AUTEUR -

N#	Mots.
1	D1
1	D2
2	D1
2	D2
3	D1
3	D2
3	D3

- MOTS -

Les auteurs de ce modèle définissent de plus, un ensemble d'opérations sur des relations NF2, notamment des opérations ensemblistes. Par exemple, pour répondre à la question :

«editer le titre et le prix des livres
avec A2 comme AUTEUR et D1 et D2 comme mots-clés»

En utilisant les trois relations LIVRE, AUTEUR et MOTS, la question en SQL/DS sera :

```
select TITRE, PRIX
  from LIVRE, AUTEUR, MOTS X, MOTS Y
       where AUTEUR     = 'A2'
         and AUTEUR.N#  = LIVRE.N#
         and LIVRE.N#   = X.N#
         and X.MOTS     = 'D1'
         and LIVRE.N#   = Y.N#
         and Y.MOTS     = 'D2'
```

Alors que dans le formalisme des auteurs, en utilisant la relation LIVRES non normalisée, la question sera :

```
select TITRE, PRIX
  from LIVRES
       where (D1,D2) in MOTS-CLES
       and   (A2)    in AUTEURS
```

Cette dernière requête étant beaucoup moins complexe que la précédente.

A cette extension naturelle de l'opération de sélection (par introduction de comparateurs d'inclusion et par extension de la notion de constante), deux nouvelles opérations sont définies : NEST et UNNEST.

L'opérateur NEST

Cette opération permet de regrouper des valeurs d'attributs d'une relation 1NF (sous première forme normale) pour obtenir une relation NF2 (non sous première forme normale).

Exemple d'opération NEST

Soit la relation R1 :

Livre	Classe	Eleve	Prof.
b1	math	e1	p1
b1	math	e2	p1
b2	math	e1	p1
b2	math	e2	p1
b3	hist.	e3	p2
b3	hist.	e4	p2

L'opérateur NEST sur l'attribut LIVRE de la relation R1, dénotée:

R2 <-- nest LIVRE (CLASSE,ELEVE,PROF) sur R1

donnera la relation R2 suivante :

Livres	Classe	Eleve	Prof.
b1	math	e1	p1
b2			
b1	math	e2	p1
b2			
b3	hist.	e3	p2
b3	hist.	e4	p2

Cette opération permet donc de regrouper tous les livres d'un même élève pour un même cours avec le même professeur.

L'opérateur UNEST

Intuitivement, UNNEST est l'opération qui assure la fonction inverse de NEST.

3.2.3. Généralisation du modèle NF2

Schek et Scholl dans [Sch84] donnent une définition plus générale pour décrire une relation NF2. Ainsi le domaine associé à une relation NF2 est alors :

- soit un ensemble de valeurs atomiques,
- soit un ensemble de relations NF2.

EXEMPLE

Considérons la relation P (relation 1NF) suivante :

NS#	Resp.	Projet	Equip	Snom
1	chef1	projet1	eq1	ifo
2	chef2	projet2	eq2	syst

- Relation P -

La relation P décrit les projets en cours dans différents services. Chaque ligne de la relation contient pour un service les informations suivantes : un numéro de service (NS#), le nom du responsable du projet (RESP.), le nom du projet (PROJET), l'équipement nécessaire (EQUIP) et le nom du service (SNOM).

Si l'on désire détailler l'attribut PROJET, on remplacera les valeurs "projet1" et "projet2" par des relations 1NF avec les attributs : (PNOM, PEMP, PDESC), où PNOM est le nom du projet, PEMP est une information sur les employés et PDESC est la description du projet.

De même, pour détailler l'attribut EQUIP on remplacera les valeurs "eq1" et "eq2" par des relations 1NF avec les attributs : (EN#, EDESC), où EN# est un numéro d'équipement et EDESC sa description.

On obtient ainsi une relation P' qui n'est plus une relation 1NF mais une relation NF2 :

NS#	RESP	PROJET			EQUIP		SNOM
		PNOM	PEMP	PDESC	EN#	EDESC	
1	chef1	x1	e1	pms	10	type	ifo
		x2	e2	sql	77	pc	
		x3	e3	qmf			
2	chef2	x1	e4	pms	12	cpu	sys
		x4	e5	iso	11	off	

- Relation P' -

Chaque employé peut être caractérisé par un numéro et un nom. En remplacant les valeurs e1, e2,... e5 par des relations 1NF avec les attributs : (EN#, ENOM) ont obtient la relation P'' suivante:

NS#	RESP	PROJET				EQUIP		SMOM
		PNOM	EMP.		PDESC	EN#	EDESC	
			EN#	ENOM				
1	chef1	x1	23	emp1	pms	10	type	ifo
			25	emp2		77	pc	
		x2	88	emp3	sql			
			97	emp4				
		x3	62	emp5	qmf			
			55	emp6				
2	chef2	x1	21	emp7	pms	12	cpu	syst
			26	emp8		11	off	
		x4	36	emp9	iso			

- Relation P'' -

Dans ce modèle, Schek et Scholl définissent des opérations qui se distinguent de celles proposées sur les relations NF2 (non généralisées) par l'introduction de d'opérations de sélection et de projection plus «sophistiquées». Ces opérations sont construites à partir des primitives de projection et de sélection et à partir des opérations NEST et UNNEST.

4. Intelligence artificielle et bases de données

Les bases de données jouent un rôle carrefour en informatique et il n'est pas surprenant de les retrouver sous leur forme la plus évoluée au coeur des systèmes de gestion de connaissance tels qu'ils sont envisagés.

La coopération entre l'intelligence artificielle et les bases de données peut être fructueuse et ce dans de nombreux domaines :

- la conception de nouveaux modèles,
- l'utilisation de bases de données dans les systèmes experts,
- la conception d'interfaces utilisateurs de haut niveau dans les SGBD,
- l'amélioration de l'efficacité des bases de données.

En particulier, la similarité entre les bases de données relationnelles et la déduction basée sur la logique est exploitée afin d'intégrer ces deux éléments de multiples façons. Ainsi de nouvelles fonctionnalités sont étudiées notamment les bases de données logiques ou déductives offrant des capacités :

- de dérivation d'informations non stockées,
- de déduction à partir du schéma de la base,
- de vérification de contraintes d'intégrité,
- et de gestion d'informations incomplètement spécifiées.

Pour illustrer cette coopération, nous retiendrons deux exemples :

1) Une étude de couplage entre un système expert et une base de données relationnelle;

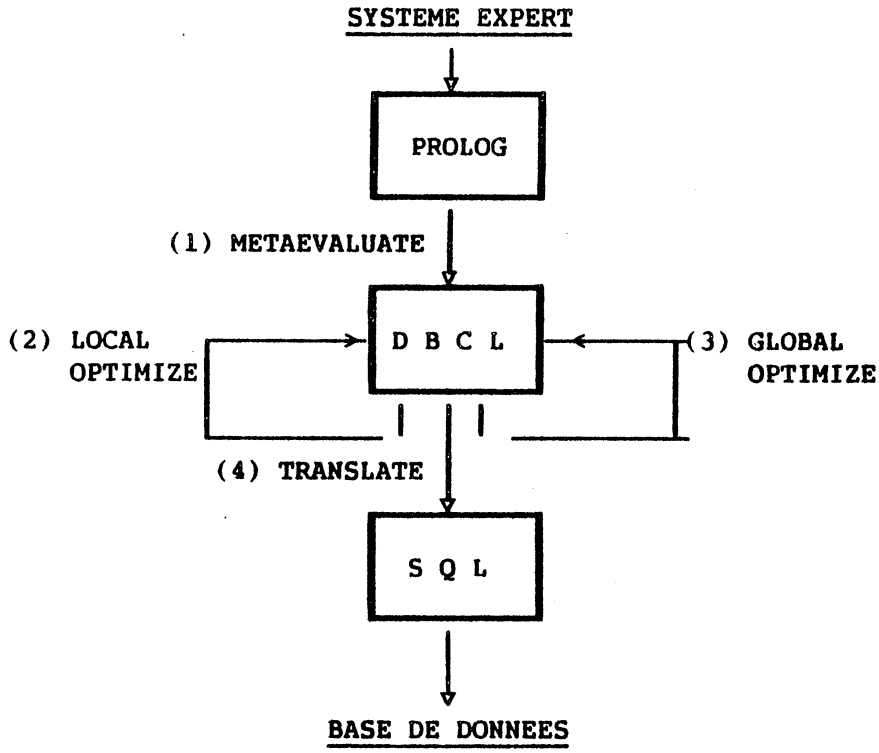
2) et un système expert relationnel pour la prise de décision en gestion.

4.1. Couplage sytème expert et BD. relationnelle

Dans [Vas84], la notion de couplage entre un système expert et une base de donnée relationnelle est étudiée et un mécanisme de traduction optimisé est décrit pour permettre à un système expert, basé sur la logique et écrit en PROLOG, d'accéder à une base de données relationnelle à travers le langage SQL.

Ce mécanisme peut être utilisé pour examiner des informations contenues dans une base de données depuis un système expert ou pour stocker dans la base de données des informations générées par le système expert. L'idée ici, est la définition d'un langage intermédiaire d'appel à la base de données.

Ce langage (DBCL), exprimé en PROLOG, utilise et comprend les relations de la base. L'architecture du mécanisme de traduction est décrite par la figure suivante :



Les fonctions principales du mécanisme de traduction sont les suivantes :

(1) METAEVALUATE : rassemble les requêtes sur la base de données. Ce module traduit les instructions PROLOG en instructions DBCL.

(2) LOCAL OPTIMIZE : simplification syntaxique et sémantique. Ce module supprime la redondance des prédicats DBCL et empêche l'exécution d'opérations inutiles.

(3) GLOBAL OPTIMIZE : optimisation des accès multiples à la base. Ce module a deux fonctions. D'une part il détermine quelles sont les instructions DBCL qui peuvent être évaluées avec la base interne de PROLOG et lesquelles doivent donner lieu à la génération de requêtes à la base de données; et d'autre part, il décide du moment où les résultats d'une requête doivent être stockés dans la base ceci en cas d'appel récursifs.

(4) TRANSLATE : génération de requêtes. Ce module génère les requêtes SQL d'accès à la base à partir des prédicats DBCL.

En conclusion, cette approche qui consiste à définir un mécanisme de translation pour coupler un système expert et une base de données est intéressante. En effet, elle fournit un mécanisme de connexion uniquement attaché au langage du système expert et devient donc indépendante des applications.

4.2. Un système expert pour la gestion

Jusqu'à présent trés peu de systèmes experts ont été développés pour la gestion. Les besoins des entreprises en système expert peuvent se résumer en deux mots : coordination et contrôle. Dans le cadre de l'aide à la décision en gestion les systèmes experts peuvent aider à la coordination des tâches, faciliter des prises de décision plus rapides et jusqu'à accroître les compétences des décideurs. Dans cette optique, C.J. ERSNT [Ers84] propose un système expert relationnel pour le contrôle et la gestion du service infirmier d'un hopital.

La complexité des structures de décision dans les entreprises implique le besoin de grandes bases de données qui nécessitent de plus en plus de systèmes de gestion de bases de données sophistiqués. L'interfacage des grandes bases de données avec les systèmes de déduction rencontre le problème connu sous le nom de «l'explosion des connaissances». Une approche, pour résoudre ce problème consiste à coder des informations sémantiques dans une base de données en terme de réseaux sémantiques.

Le système MANAGER, proposé par ERNST, essai de résoudre ces problèmes en créant un métalangage de programmation appelé METABOL (sigle pour Buisness Oriented Métalangage) en utilisant PROLOG comme langage objet. METABOL est utilisé dans MANAGER pour exprimer les propriétés sémantiques des entités enregistrées dans la base de données. Dans ce but le concept de métarelation est défini.

La stratégie de contrôle dans MANAGER, dans le but de réduire l'espace de recherche lors de la résolution d'un problème, est centrée sur le contrôle des informations qui dépendent des domaines. Les dépendances sont exprimées en METABOL par des métarègles qui choisissent les sous-problèmes à résoudre ou des règles à appliquer afin de résoudre ces dépendances par des recherches dirigées par le contenu.

Dans ce système, une approche représentation des connaissance est utilisée afin d'élaborer un système d'aide à la décision basé sur une technique d'inférence. Un métalangage de programmation logique est proposé intégrant la sémantique et l'espace de décision des processus de prise de décision et assure une interface avec les bases de données relationnelles.

C H A P I T R E 3

- LE MODELE SHIVA -

1. Introduction

1.1. Présentation du modèle SHIVA

- Considérons le tableau EMPLOYES suivant :

Num Emp	Etat Civil		Historique		
	Nom	Prenom	Fonctions	Info. Salaire	
				Date Sal. JJ MM AA	Salaire
4	Dupont	Octave	ouvrier	11 01 78	3500.50
				01 01 79	4000.00
6	Durand	Leo	ouvrier	21 05 77	3300.00
				01 01 78	3500.45
			magasinier	10 05 78	4500.00
8	Lesage	Alfred	magasinier	10 04 77	5000.67
				01 01 78	5500.50
			comptable	10 05 78	7800.50
9	Al-Dor	Zoe	secretaire	11 03 77	5200.00
				01 01 78	7500.00
			secr. dir.	10 06 78	7800.70
				01 01 79	8000.00

Ce tableau donne des informations sur le personnel d'une entreprise. La signification des colonnes du tableau EMPLOYES est la suivante :

- NUM-EMP : numéro d'un employé,
- ETAT-CIVIL : état civil de l'employé,
- HISTORIQUE : historique de l'employé dans l'entreprise.

Chaque ligne principale du tableau EMPLOYES décrit un employé pariculier. Une ligne de ce tableau se comprend ainsi :

- A chaque employé est associé un numéro unique d'identification, une information sur son état civil et une information sur son historique.

- L'information sur l'état civil est constituée de deux données :

  - le nom de l'employé
  - et son prénom.

- Une information sur l'historique d'un employé est aussi constituée de deux parties :

  - l'ensemble des différentes fonctions que l'employé a occupées dans l'entreprise,
  - pour chacune de ses fonctions, une information sur ses différents salaires.

- Une information sur les salaires d'un employé, pour une fonction donnée, est un ensemble dont chaque élément comprend :

  - la date à laquelle une modification de salaire a eu lieu (pour la première fonction occupée, la première date correspond à la date d'embauche),
  - le montant du salaire correspondant.

Remarques :

- Le tableau EMPLOYES est un ensemble dont les éléments sont des informations sur un employé.

- Chaque élément «employé» est une donnée structurée composée de trois parties, appelées composantes.

- La composante NUM-EMP est une donnée simple : un nombre entier.

- Les composantes ETAT-CIVIL et HISTORIQUE sont toutes deux des données structurées.

- ETAT-CIVIL est composé de deux parties : la composante NOM et la composante PRENOM, toutes deux sont des données simples : des chaînes de caractères.

- HISTORIQUE est composée de deux parties : la composante FONCTIONS et la composante INFO-SALAIRES.

- Pour un employé donné, les valeurs de la composante FONCTIONS sont des ensembles dont les éléments sont des données simples : des

chaînes de caractères.

- Pour chaque valeur de la composante FONCTIONS, (d'un employé donnée), correspond une valeur de la composante INFO-SALAIRE, laquelle valeur est aussi un ensemble. Les éléments de cet ensemble sont des données structurées, composées des composantes DATE-SALAIRE et SALAIRE.

- La composante SALAIRE est une donnée simple : un nombre réel qui exprime le montant mensuel en francs du salaire, d'un employé à une date donnée.

- Et la composante DATE-SALAIRE est elle même structurée, JJ, MM et AA en sont ses composantes, toutes les trois sont des données simples : des nombres entiers.

De ce tableau on pourra lire aisément que :

> «Monsieur Léo Durand, dont le numéro d'employé est 6, a été embauché comme ouvrier le 21/05/77 avec un salaire de 3300F. Le 01/01/78, son salaire était de 3500.45F. Monsieur Durand est devenu magasinier le 10/05/78 avec un salaire de 4500F.»

Et on pourra, par lecture, déduire que :

> «Le poste de magasinier attribué à monsieur Durand appartenait auparavant à Monsieur Lesage, lequel est devenu comptable»

1.2. Concepts de base du modèle SHIVA

Ensembles et éléments

Les notions de relations et de n-uplets telles qu'elles apparaissent dans les systèmes relationnels classiques, ne sont pas apparentes dans le modèle SHIVA. En fait, elles sont entièrement comprises dans les notions d'ensembles et d'éléments, respectivement.

Ainsi, on peut considérer que le tableau EMPLOYES est une relation (non sous première forme normale), que NUM-EMP, ETAT-CIVIL et HISTORIQUE sont les attributs de cette relation et que les éléments «employé» sont les n-uplets de cette relation. Les valeurs de l'attribut HISTORIQUE sont des relations dites «enchassées», les attributs de ces relations étant FONCTIONS et INFO-SALAIRES. Les valeurs de la composante INFOS-SALAIRES étant elles-mêmes des relations «enchassées» ayant pour attributs DATE-SALAIRE et SALAIRES.

Les domaines

La notion de domaine du modèle SHIVA est la même que celle du modèle relationnel classique (au sens "type" d'une donnée), avec la particularité que l'utilisateur a la possibilité de définir ses propres domaines.

Trois domaines (appelés domaines de base) sont prédéfinis dans SHIVA. Il s'agit de :

- l'ensemble des nombres entiers,
- l'ensemble des nombres réels,
- l'ensemble des chaînes de caractères.

Les constructeurs

La définition d'un nouveau domaine s'effectue par composition d'un domaine de base et d'un contructeur de domaine.

Les constructeurs de domaines permettent de décrire la structure des éléments du domaine (valeurs simples, valeurs composées ou valeurs d'ensembles). Un domaine, ainsi défini, pourra par la suite être utilisé pour définir de nouveaux domaines utilisateurs.

Les variables

La désignation et l'accès à une donnée s'effectuant grâce au concept de variable. La variable est le moyen unique pour nommer une donnée et l'accéder quelleque soit sa structure, simple ou composée, ensemble ou élément d'un ensemble.

Une base

Par définition, une base de données SHIVA est un ensemble nommé d'informations, chacune d'elles étant identifiée et accédée séparément par une variable. A toute base SHIVA est associée une variable prédéfinie, de nom $BASE, qui est en quelque sorte le catalogue de la base.

les fonctions SHIVA

Le modèle SHIVA propose un emsemble de primitives pour la définition de nouveaux domaines, la construction de variables, la manipulation des données contenues dans une variable et l'accès séparé aux composantes des données structurées ou aux éléments d'emsembles de valeurs.

A cet ensemble de primitives, appelées primitives de base, est adjoint un ensemble d'opérateurs de l'algèbre relationnelle et ensemblistes pour la manipulation des données de la base dont les domaines sont définis à l'aide des constructeurs d'ensembles.

Fonctions utilisateur

A partir des fonctions de base SHIVA, le concepteur d'une base a la possibilité de définir de nouvelles fonctions, appelées fonctions utilisateur, et de les rassembler pour constituer une vue de la base.

A une base SHIVA peuvent être associées plusieurs vues et un utilisateur ne peut accéder aux données d'une base que par l'intermédiaire d'une vue sur cette base. Lors du regroupement d'un ensemble de fonctions utilisateur en une vue, le concepteur spécifiera une liste de noms d'utilisateurs pour lesquels les fonctions de la vue sont accessibles.

Le concepteur d'une base est le seul à pouvoir utiliser les fonctions de base de SHIVA (définition de domaines, de variables, de fonctions utilisateur et de vues). Lors d'une session SHIVA, un utilisateur spécifiera le nom de la base sur laquelle il veut travailler et la vue contenant les fonctions que le concepteur aura mis à sa disposition. Aucune fonction SHIVA de base n'est accessible à un utilisateur en dehors des fonctions de sa vue.

Notations

- La syntaxe employée pour décrire les fonctions du modèle SHIVA et leurs arguments sera celle du langage Lisp : une liste (suite de symboles, précédée d'une parenthèse ouvrante et terminée par une parenthèse fermante). Le nom de la fonction est le premier élément de la liste, les autres éléments sont ses arguments.

- Un mot précédé du signe < et terminé par le signe > (<mot>) indique une information qui doit être fournie par l'utilisateur.

- Les paires de crochets ([]) indiquent des éléments optionnels.

- Les paires d'accolades ({}) indiquent des éléments optionnels qui peuvent être répétés.

- dans les exemples, les lignes débutant par une flèche (-->) sont les réponses que donnera l'interpréteur du modèle SHIVA.

1.3. Définition de domaines

La définition d'un nouveau domaine s'effectue à l'aide de la fonction de base defdom, dont la syntaxe complète est :

```
(defdom <nom-domaine> <structure> [ <facettes> ] )
```

Où <nom-domaine> est le nom du domaine à définir, <structure> est une forme permettant de décrire les structures de données des éléments du domaine (s'il s'agit d'éléments simples, composés ou d'ensembles) et <facettes> est un ensemble de formes facultatives, apportant des informations complémentaires sur les domaines (le rôle et la syntaxe des facettes sont présentés dans le chapitre suivant).

La structure des éléments d'un domaine est spécifiée dans la fonction 'defdom' par l'argument <structure> dont la syntaxe revêt une des deux formes suivantes :

(1) (<constr> <domaine>)

ou

(2) (<constr> (<struc-champ> { <struc-champ> }))

Où <struc-champ> a la forme suivante :

(<champI> (<constrI> <domaineI>) [<facettes>])

La première forme (1), est utilisée pour décrire la structure d'un domaine dont les éléments sont des données indivisibles. Dans cette forme, l'argument <constr> est un mot-clé de SHIVA spécifiant la nature des éléments du domaine : simples ou ensembles. L'argument <domaine> précisant l'ensemble de définition de tous les éléments du domaine en cours de définition.

La deuxième forme (2), est utilisée pour décrire la structure d'un domaine dont les éléments sont des données composées. Ici, pour chaque composante d'un élément, appelée champ, on spécifiera le nom du champ avec l'argument <champI>, le constructeur <constrI>, le domaine des valeurs possibles <domaineI> pour ce champ et éventuellement un ensemble de facettes complémentaires <facettes> associées à ce champ.

Deux contructeurs sont proposés pour définir la structure de données d'un domaine :

- un : toutes les valeurs du domaine sont des données simples,
- ens : toutes les valeurs du domaine sont des ensembles dont les éléments sont tous définis sur le domaine associé.

Un ensemble SHIVA est une collection d'éléments où il peut exister plusieurs occurrences de la même valeur. L'accès aux éléments d'un ensemble ne peut s'effectuer que de manière séquentielle (balayage des éléments un par un) ou sélective (accès à un élément par sa valeur).

Une donnée composée peut être accédée globalement en lecture ou en écriture. Mais il est aussi possible d'accéder séparément aux différents champs d'une donnée composée. Cet accès se faisant par le nom du champ.

Exemple 1 :

```
(defdom UN-NUMERO (un entier))
```

Cette expression permet de définir le domaine de nom UN-NUMERO, dont les éléments sont des données simples et indivisibles. Le domaine de définition étant l'ensemble des nombres entiers. Les valeurs suivantes appartiennent au domaine UN-NUMERO :

```
12 -4 5  -32767 0 +32768
```

Exemple 2 :

```
(defdom UN-SALAIRE (un reel))
```

Définition du domaine UN-SALAIRE; l'ensemble des nombres réels est l'ensemble des valeurs possibles de ce domaine. Les valeurs suivantes appartiennent au domaine UN-SALAIRE :

```
4000.00  -34.5 1.0 -1.7e+7 -32769.0
```

Exemple 3 :

```
(defdom UNE-FONCTION (un chaine))
```

Définition du domaine UNE-FONCTION; l'ensemble de toutes les chaînes de caractères (de longueur variable) est l'ensemble des valeurs possibles de UNE-FONCTION. Les valeurs suivantes appartiennent au domaine UNE-FONCTION :

```
"ouvrier"
"secretaire de direction "
"111Barnabe"
```

```
"-----A-"
"<<A B C>>"
""
```

La chaîne vide (de longueur zero) est notée "".

Exemple 4 :

```
(defdom ENS-NUMEROS (ens entier))
```

Définition du domaine ENS-NUMEROS dont les valeurs sont toutes des ensembles de nombres entiers. Les ensembles suivants appartiennent au domaine ENS-NUMEROS :

```
(12 23 34)
(12 23 34 -12 -34)
(-1 -2 -4 0 4 2 1)
()
```

Un ensemble vide sera notée par ().

Exemple 5 :

```
(defdom UNE-FONCTION (un  chaine))
(defdom ENS-FONCTION (ens UNE-FONCTION))
```

Le domaine UNE-FONCTION est défini sur l'ensemble des chaînes de caractères. Ce domaine est utilisé pour définir le domaine ENS-FONCTION dont les éléments sont des ensembles de chaînes de caractères. Les ensembles suivants appartiennent au domaine ENS-FONCTION :

```
("ouvrier" "secretaire" "comptable" "")
("directeur" "" "l" "lx;" "()" "...")
("")
```

Exemple 6 :

```
(defdom UNE-DATE (un (JJ (un entier))
                     (MM (un entier))
                     (AA (un entier))))
```

Le domaine UNE-DATE est défini sur l'ensemble des valeurs composées de trois nombres entiers. Chacune de ces valeurs est une donnée structurée, dont les champs sont nommés JJ, MM et AA. Les valeurs suivantes appartiennent au domaine UNE-DATE :

```
(12 01 1978)
(31 02 1900
(1   1    1)
(-12 -6 19999)
```

Exemple 7 :

```
(defdom UN-INFO-SALAIRE
        (un (DATE-SALAIRE (un UNE-DATE))
            (SALAIRE      (un reel))))

(defdom DES-HISTORIQUES
        (ens (FONCTION      (un chaine))
             (INFO-SALAIRES (ens UN-INFO-SALAIRE))))
```

Ces deux définitions permettent de construire le domaine du composant HISTORIQUE de la relation EMPLOYES. Les valeurs du domaine DES-HISTORIQUES sont des ensembles dont chaque élément est une donnée structurée. La composante FONCTION prend ses valeurs dans le domaine des chaînes de caractères et les valeurs de la composante INFO-SALAIRES sont des ensembles dont les éléments sont définis sur le domaine UN-INFO-SALAIRE.

1.4. Définition de variables

La définition d'une variable s'effectue à l'aide de la fonction defvar, en spécifiant le nom de la variable, le nom du domaine des valeurs de cette variable et, éventuellement, une valeur initiale. La syntaxe générale de la fonction 'defvar' est la suivante :

```
(defvar <nom-var> <nom-dom> [ <val-init> ] )
```

Où <nom-var> est le nom de la variable à définir, <nom-dom> est le nom du domaine des valeurs de la variable et <val-init> est une expression SHIVA qui, aprés évaluation, donnera la valeur initiale à affecter à la variable.

Une expression SHIVA peut être : une constante, le nom d'une variable ou un appel de fonction. Sauf indication contraire, la plupart des arguments des fonctions SHIVA sont des expressions. Les fonctions SHIVA retournent toujours une valeur, ce qui leur permet d'être utilisées comme arguments d'autres fonctions.

Dans la définition d'une variable, l'argument <val-init> est optionnel, en son absence, la valeur initiale par défaut est la valeur nulle (notée NIL). Cette valeur a la signification de «valeur manquante».

Sauf indication contraire, lors de la définition du domaine, toute variable peut prendre la valeur nulle comme valeur. Et l'accès séparé en lecture à un champ d'une variable définie sur un domaine composé, retourne toujours la valeur NIL, quelque soit le champ, si la variable a pour valeur la valeur nulle.

Exemple 1 :

```
(defvar SALAIRE reel 4000.50)
--> 4000.50
```

Création de la variable de nom SALAIRE, sur le domaine des nombres réels, avec la valeur initiale 4000.50. La fonction 'defvar' retourne la valeur initiale affectée à SALAIRE.

```
(affecter SALAIRE (plus SALAIRE 500.0))
--> 4500.50
```

La valeur de la variable SALAIRE est additionnée à la valeur 500.0 (fonction plus) et le résultat de cette opération est affectée à SALAIRE (fonction plus). La fonction 'plus' retourne la valeur de l'addition, cette valeur est utilisée comme argument de la fonction 'affecter', laquelle retourne la valeur affectée à SALAIRE.

Exemple 2 :

```
(defdom UN-SALAIRE (un reel))
--> UN-SALAIRE

(defdom ENS-SALAIRES (ens UN-SALAIRE))
--> LISTE-SALAIRE

(defvar E-S ENS-SALAIRES () )
--> ()
```

Ces expressions permettent de définir un domaine de nom UN-SALAIRE, dont l'ensemble de définition est l'ensemble des nombres réels. Puis le domaine UN-SALAIRE est utilisé avec le constructeur 'ens' pour définir un nouveau domaine de nom ENS-SALAIRES. L'ensemble des valeurs de ce domaine étant l'ensemble de tous les ensembles de «UN-SALAIRE» possibles. Et à partir du domaine ENS-SALAIRES, la variable E-S est définie avec pour valeur initiale l'ensemble vide (aucun élément)

```
(affecter E-S (500.0 300.0  100.0))
--> (500.0 300.0 100.0)
```

Affectation de l'ensemble (500.0 300.0 100.0) à la variable E-S.

```
(ins_e E-S 800.0)
--> (800.0 500.0 300.0 100.0)
```

La valeur 800.0 est ajoutée en tête de l'ensemble E-S (fonction ins e). La valeur de la variable, aprés ajout, est retournée par la fonction.

```
(defvar LONG entier (long-e E-S) )
--> 4
```

Création de la variable LONG sur le domaine des nombres entiers. La longueur de la liste E-S (fonction long-e) est la valeur initiale de la variable LONG.

```
(defvar MOYENNE reel
        (div-e (som-e E-S) (long-e E-S)))
-->  400.0
```

Création de la variable MOYENNE sur l'ensemble des nombres réels. La moyenne arithmétique des éléments de l'ensemble E-S est calculée et affectée comme valeur initiale à la MOYENNE. Cette moyenne est obtenue par addition de tous les éléments de E-S (fonction som-e) et division de cette somme (fonction div-e) par le nombre d'éléments de la liste E-S.

Exemple 3 :

Les expressions suivantes :

```
(defdom UNE-DATE
        (un (JJ (un entier))
            (MM (un entier))
            (AA (un entier))))
--> UNE-DATE

(defdom UN-INFO-SALAIRE
        (un (DATE    (un UNE-DATE))
            (SALAIRE (un reel))))

(defvar INFO  UN-INFO-SALAIRE
        ( (01 01 78) 5000.50)

-->( (01 01 78) 5000.50)
```

permettent de créer la variable INFO sur le domaine UN-INFO-SALAIRE avec pour valeur initiale la constante structurée :

```
( (01 01 78) 5000.50)
```

- Pour obtenir la valeur du champ DATE :

```
(de INFO DATE)
--> (01 01 78)
```

- Pour modifier le champ SALAIRE :

```
(dans INFO SALAIRE
        (plus 500.0 (de INFO SALAIRE)))
--> 5500.50
```

Exemple 4 :

L'expression suivante :

```
(defdom ENS-FONCTIONS (ens chaine))
--> ENS-FONCTIONS
```

permet de définir le domaine ENS-FONCTIONS dont les éléments sont des ensembles de chaînes de caractères.

```
(defvar S1 ENS-FONCTIONS
        ("ouvrier" "secretaire" "directeur"))
--> ("ouvrier" "secretaire" "directeur"))

(defvar S2 ENS-FONCTIONS ("ouvrier" "secretaire")
--> ("ouvrier" "secretaire")
```

Création des variables S1 et S2 toutes deux sur le domaine ENS-FONCTIONS.

La séquence suivante :

```
(si  (inclus "ouvrier" (int-e S1 S2))
             (ins-e S2 "magasinier")
             (ins-e S1 "comptable"))
--> ("magasinier" "ouvrier" "secretaire")
```

permet d'ajouter la chaîne "magasinier" à l'ensemble S2 si la chaîne "ouvrier" appartient à l'intersection des ensembles S1 et S2, sinon la chaîne "comptable" est ajoutée à l'ensemble S1.

La fonction si admet trois arguments, le premier est une condition à tester, le deuxième et le troisième sont des fonctions qui seront évaluées selon le résultat du test. Si la condition est vérifiée le deuxième argument sera évalué, sinon c'est le troisième qui le sera. La fonction 'si' retourne la valeur de la fonction qui a été évaluée.

Dans cet exemple la fonction int-e effectue l'intersection des ensembles S1 et S2, l'ensemble résultant est utilisé par la fonction inclus qui retourne la valeur NIL si la chaîne "ouvrier" n'appartient pas à l'intersection.

Exemple 5 :

Considérons les expressions suivantes :

```
(defdom ENS-INFO-SALAIRES
        (ens UN-INFO-SALAIRE))

(defdom UN-HISTORIQUE
        (un (FONCTION     (un chaine))
            (INFO-SALAIRE (un ENS-INFO-SALAIRE))))
```

```
(defdom ENS-HISTORIQUES
        (ens UN-HISTORIQUE))

(defvar HISTO ENS-HISTORIQUE
 ( ("ouvrier"    ( ( (10 05 78) 4000.00)
                   ( (01 01 79) 4500.00)) )
   ("magasinier" ( ( (20 05 79) 5000.00)
                   ( (01 01 80) 5500.00)
                   ( (01 01 81) 6000.00)) ) )
```

Ces expressions permettent de définir la variable HIST dont les valeurs possibles sont des ensembles dont le domaine est ENS-HISTORIQUE. La valeur initiale de cette variable correspond au tableau suivant :

FONCTION	HISTORIQUE	
	DATE	SALAIRE
	JJ MM AA	
ouvrier	10 05 78	4000.00
	01 01 79	4500.00
magasinier	20 05 79	5000.00
	01 01 80	5500.00
	01 01 81	6000.00

Exemple 6 :

```
(defdom UNE-LIGNE
        (un (FONCTION      (un chaine))
            (INFO-SALAIRE (un UN-INFO-SALAIRE))))

(defdom UNE-TABLE (ens UNE-LIGNE))

(defvar TABLE UNE-TABLE (eclater HISTO))
--> ( ("ouvrier"    ( (10 05 78) 4000.00 ))
      ("ouvrier"    ( (01 01 79) 4500.00 ))
      ("magasinier" ( (20 05 79) 5000.00 ))
      ("magasinier" ( (01 01 80) 5500.00 ))
      ("magasinier" ( (01 01 81) 6000.00 )) )
```

Ces expressions permettent d'obtenir la table suivante :

FONCTION	HISTORIQUE	
	DATE	SALAIRE
	JJ MM AA	
ouvrier	10 05 78	4000.00
ouvrier	01 01 79	4500.00
magasinier	20 05 79	5000.00
magasinier	01 01 80	5500.00
magasinier	01 01 81	6000.00

Cette table est construite à partir de la valeur de la variable HISTO Les éléments de l'ensemble HISTO sont des données structurées composées d'une valeur simple (FONCTION) et d'un ensemble (INFO-SALAIRE). La fonction eclater permet de d'effectuer le «produit» de chacune des valeurs du composant FONCTION avec les valeurs correspondantes du composant INFO-SALAIRE.

1.5. Définition de fonctions

La définition d'une nouvelle fonction utilisateur s'effectue à l'aide de la primitive defonc. La syntaxe de cette fonction est la suivante :

```
(defonc    <nom-fonc>     <domaine>
           <liste-param> <liste-locaux>
                    <corps> )
```

Où <nom-fonc> est le nom de la fonction utilisateur à définir, <domaine> est le domaine de la valeur que doit retourner la fonction, <liste-param> est une forme décrivant les paramètres de la fonction, <liste-locaux> est une forme décrivant les variables utilisées localement dans la fonction et <corps> est une expression SHIVA dont le résultat, aprés évaluation, sera la valeur de la fonction.

La fonction 'defonc' interdit la définition d'une fonction utilisateur avec le même nom qu'une fonction SHIVA. Si l'argument <nom-fonc> est déjà utilisé pour identifier une autre fonction, la définition en cours remplacera l'ancienne.

La fonction 'defonc' retourne pour valeur le nom de la fonction utilisateur si aucune erreur n'est apparue durant le traitement sinon la valeur NIL est retournée.

L'argument <domaine> doit correspondre au nom d'un domaine de base ou d'un domaine utilisateur qui doit être connu au moment de l'évaluation de la fonction 'defonc'.

La forme de l'argument <liste-param> est une liste dont la syntaxe est la suivante :

```
(  (<param-1> <dom-1>)
   (<param-2> <dom-2>)
            ...
   (<param-N> <dom-N>) )
```

Où chaque liste :

```
(<param-I> <dom-I>)
```

introduit la définition d'un paramètre de la fonction. L'élément <param-I> étant le nom du paramètre et l'élément <dom-I> son domaine de définition. Si la fonction utilisateur n'utilise aucun paramètre, la liste vide sera la valeur de l'argument <liste-param> de la fonction 'defonc'.

La forme de l'argument <liste-locaux> est une liste dont la syntaxe est la suivante :

```
(  (<var-1> <dom-1> [ <val-init-1> ] )
   (<var-2> <dom-2> [ <val-init-2> ] )
                ...
   (<var-N> <dom-N> [ <val-init-N> ] ) )
```

Où chaque liste :

```
(<var-I> <dom-I> [ <val-init-I> ] )
```

introduit la définition d'une variable locale à la fonction utilisateur. L'élément <var-I> étant le nom de la variable locale, <dom-I> son domaine de définition et la valeur initiale éventuelle de la variable. Si la fonction utilisateur n'utilise aucune variable locale, la liste vide sera la valeur de l'argument <liste-locaux> de la fonction 'defonc'.

Lors de l'appel d'une fonction utilisateur, s'il existe une liste de paramètres, alors pour chacun, des variables temporaires seront créées et les valeurs des différents arguments de l'appel seront affectées à ces variables. L'affectation des valeurs s'effectue par position : le paramètre de rang I dans la liste <liste-param> reçoit la valeur de l'argument de même rang dans la liste d'appel de la fonction.

Aprés affectation des valeurs aux paramètres de la fonction, les variables locales sont créées et initialisées. Puis le corps de la fonction est évalué et le résultat de cette évaluation est retourné comme valeur de la fonction.

Toutes les variables créées lors de l'évaluation d'une fonction utilisateur (paramètres et variables locales) seront détruites avant le retour de la valeur de la fonction.

Exemple 1 :

```
(defonc INCREMETER entier
        ( (I entier) ) ()
        (plus I 1))
--> INCREMENTER
```

La fonction 'defonc' est utilisée ici pour définir une fonction utilisateur de nom INCREMENTER. Le domaine de la valeur de la fonction est l'ensemble des entiers. Cette fonction utilise un paramètre, de nom I, défini sur le domaine de base 'entier' et ne déclare aucune variable locale. Le corps de la fonction INCREMENTER est constitué de l'expression :

```
(plus I 1)
```

qui effectue l'addition de la valeur du paramètre I avec la constante entière 1. Le résultat de l'evaluation de cette expression sera la valeur retournée par la fonction INCREMENTER. La fonction INCREMENTER peut s'utiliser ainsi :

```
(INCREMENTER
        (INCREMENTER 3)))
--> 5
```

o la fonction INCREMENTER est appelée récursivement. Une première fois, avec la valeur 3 et la valeur 4 est retournée. Puis une seconde fois avec la valeur 4 ce qui donne le résultat final 5.

Exemple 2 :

```
(defonc INIT entier ((NOM chaine)) ()
      (si (defbase NOM)
                  1
                  (ecrire "BASE EXISTE DEJA")))
 --> INIT
```

La fonction utilisateur INIT effectue la création d'une base dont le nom est fourni en paramètre. Si la base à créer n'existe pas, la fonction INIT retourne la valeur 1. Si la base existait déja, un message est imprimé et la valeur NIL est retournée par la fonction INIT.

La fonction defbase permet d'initialiser une nouvelle base SHIVA. Elle retourne la valeur NIL s'il existe déjà une base de même nom, sinon la base est initialisée et le nom de la base est retournée comme valeur de la fonction 'defbase'.

La fonction ecrire permet d'imprimer un message (la valeur de son argument) et elle retourne toujours la valeur NIL.

Utilisation de la fonction INIT :

```
(init "ENTREPRISE")
--> 1

(init "ENTREPRISE")
"BASE EXISTE DEJA"
--> nil
```

Lors du deuxième appel à la fonction INIT, le message est imprimé et la valeur NIL est retournée, car la base ENTREPRISE existe déja, elle a été créée lors du premier appel à la fonction INIT.

Exemple 3 :

Définition des domaines du tableau EMPLOYES :

```
(defonc INIT-BASE chaine () ()
   (si (init "ENTREPRISE")
       (faire

       (defdom UNE-DATE
               (un (JJ (en entier))
                   (MM (en entier))
                   (AA (en entier))))

       (defdom UN-INFO-SALAIRE
               (un (DATE    (un UNE-DATE))
                   (SALAIRE (un reel))))

       (defdom ENS-INFO-SALAIRE
               (ens UN-INFO-SALAIRE))

       (defdom UN-HISTORIQUE
               (un (FONCTION     (un chaine))
                   (INFO-SALAIRE (un ENS-INFO-SALAIRE))))

       (defdom ENS_HISTORIQUE
               (en UN-HISTORIQUE))

       (defdom UN-ETAT-CIVIL
               (un (NOM    (un chaine))
                   (PRENOM (un chaine))))
```

```
(defdom UN-EMPLOYE
        (un (NUMERO      (un entier))
            (ETAT-CIVIL  (un ETAT-CIVIL))
            (HISTORIQUE  (un ENS-HISTORIQUE))))

(defdom ENS-EMPLOYES
        (ens UN-EMPLOYE))

(defvar EMPLOYES ENS-EMPLOYES ())

"base creee")
"erreur creation"))
```

La fonction INIT-BASE a pour but de créer la base "ENTREPRISE" (si celle-ci n'existait pas déjà), de définir tous les domaines nécessaires pour décrire le tableau EMPLOYES ainsi qu'une variable de nom EMPLOYES (initialisée avec la valeur de l'ensemble vide) qui recevra les informations contenues dans le tableau EMPLOYES.

Une fois ces domaines et la variable EMPLOYES définis, aprés appel à la fonction INIT-BASE, le concepteur pourra utiliser ces domaines et cette variable pour définir de nouvelles structures et écrire des fonctions utilisateurs qui les accèderont.

Il existe plusieurs manières de définir les domaines du tableau EMPLOYES. Celle-ci est une des plus détaillées pour permettre, par la suite, d'éventuelles définitions de variables qui recevront des extraits de cette table.

2. les facettes

La syntaxe complète de la primitive de définition d'un domaine est la suivante :

```
(defdom <nom> <structure> [ <facettes> ] )
```

Où <nom> est le nom du domaine à définir, <structure> est la forme introduisant la structure des éléments du domaine et <facettes> est une suite d'informations complémentaires associées au domaine.

Les facettes permettent :

- d'associer une condition au domaine afin de restreindre l'ensemble des valeurs possibles du domaine,
- de spécifier la valeur initiale que prendra par défaut une variable définie sur le domaine,
- d'interdire éventuellement la valeur nulle (notée NIL) comme valeur du domaine,
- d'interdire la présence d'éléments dupliqués dans un ensemble,
- d'associer des expressions qui seront évaluées sous certaines conditions, pour contrôler l'accès aux données de la base, lors de la modification de la valeur d'une variable, de l'ajout ou de la suppression d'un élément d'un ensemble.

La forme générale d'une facette est la suivante :

```
(<nom-facette> <expr-1> <expr-2> ... <expr-N> )
```

Où <nom-facette> est le nom de la facette et <expr-1>, <expr-2>,... <expr-N> sont des expressions SHIVA dont la signification et le nombre dépendent de la facette.

Dans le cas où une facette est spécifiée dans la définition d'un champ d'un domaine composé, celle-ci ne s'applique qu'au champ lui même, mais les expressions de cette facette peuvent référencer les autres champs de la donnée composée.

2.1. Contraintes d'intégrité et valeurs par défaut

2.1.1. La facette VERIFIER

La facette verifier introduit une expression conditionnelle qui permet de restreindre les valeurs possibles d'un domaine. Cette facette sera généralement utilisée pour énoncer des contraintes d'intégrité.

```
( verifier <expression> )
```

Où <expression> est une expression quelconque, qui sera évaluée lors de toute mise à jour d'une donnée définie sur le domaine auquel la facette 'verifier' est associée.

Une donnée sera acceptée comme valeur possible d'un domaine en cours de définition, si les conditions suivantes sont remplies:

(1) la valeur appartient au domaine spécifié;

(2) l'évaluation de l'expression conditionnelle retourne un résultat différent de la valeur nulle.

Exemple 1 :

- Dans la définition suivante :

```
(defdom NUMERO (un entier)
                (verifier (positif+ $VAL)))
```

La forme : (verifier (positif+ $VAL)) est une facette qui introduit une restriction sur les valeurs possibles du domaine NUMERO : seuls les nombres entiers positifs non nuls sont admis.

Dans l'expression conditionnelle : (positif+ $VAL), le symbole $VAL est une speudo-variable qui contient la valeur de la donnée dont on veut vérifier l'appartenance au domaine NUMERO.

- Dans l'expression suivante :

```
(defvar N NUMERO 4)
--> 4
```

L'expression (positif+ $VAL) est évaluée avec la valeur 4 pour la speudo-variable $VAL. Cette expression retourne la valeur 4 puisque celle-ci est strictement positive (la fonction positif+ retourne la valeur de son argument si celle-ci est strictement positive, sinon la valeur NIL est retournée).

- Pour l'affectation suivante :

```
(affecter T  -6)
--> 4
```

la constante -6 sera refusée comme valeur initiale de T puisque la condition : sera évaluée avec la valeur -6 pour valeur de la speudo-variable $VAL et le résultat de l'évaluation est la valeur NIL. Dans ce cas, la fonction 'affecter' retourne l'ancienne valeur de la variable.

Exemple 2 :

- Dans la définition suivante :

```
(defdom UN-FONCTION
        (un chaine)
        (verifier (inclus $VAL
                          ("ouvrier"
                           "magasinier"
                           "comptable"
                           "secretaire"))))
```

La facette 'verifier' permet de restreindre les valeurs possibles du domaine UNE-FONCTION au seules valeurs "ouvrier", "magasinier", "comptable" et "secretaire".

La fonction inclus retournera la valeur NIL si la valeur du premier argument n'est pas inclus dans la valeur du deuxième.

Exemple 3 :

- La définition suivante :

```
(defdom UNE-DATE
        (un (JJ (un entier)
                (verifier (et (sup=  $VAL 1)
                              (inf=  $VAL 31))))
            (MM (un entier)
                (verifier (et (sup=  $VAL 1)
                              (inf=  $VAL 12))))
            (AA (un entier)
                (verifier (et (sup=  $VAL 1900)
                              (inf=  $VAL 2000))))))
```

```
(verifier
   (ou (et (inclus (de $VAL MM) (4 6 9 11))
           (inf=   (de $VAL JJ) 30))
       (et (egal   (de $VAL MM) 2)
           (zero   (mod (de $VAL AA) 4))
           (inf=   (de $VAL JJ) 29))
       (et (egal   (de $VAL MM) 2)
           (non (zero (mod (de $VAL AA) 4)))
           (inf=  JJ 28)))))
```

permet de construire le domaine UNE-DATE, dont les éléments sont des données composées de trois champs : DD, JJ et MM; tous trois définis sur l'ensemble des nombres entiers. Pour chacun de ces champs, une facette 'verifier' permet de spécifier une contrainte d'intégrité individuelle.

Les ensembles de valeurs des domaines des champs DD, MM et AA sont restreints au sous-ensembles suivants :

- l'ensemble des nombres entiers inclus dans l'intervalle [1,31] pour le champ JJ,
- l'ensemble des nombres entiers inclus dans l'intervalle [1,12] pour le champ MM,
- l'ensemble des nombres entiers inclus dans l'intervalle [1900,2000] pour le champ AA.

Pour l'ensemble des valeurs du domaine DATE, une contrainte d'intégrité «inter-composants» est énoncée, cette contrainte permet de contrôler les configurations de champs JJ, MM et AA corrects.

2.1.2. La facette INIT

Généralement, la valeur initiale d'une variable sera donnée lors de sa définition (fonction defvar), par évaluation de l'argument <val-init> (troisième argument de la fonction). La facette init permet de spécifier la valeur initiale à affecter à une variable en l'absence de l'argument <val-init> de la fonction 'defvar'. La forme de cette facette est la suivante :

```
(init <expression>)
```

Où <expression> est une expression SHIVA qui, si aucune valeur initiale n'est fournie à la fonction 'defvar', sera évaluée lors de toutes définition de variables sur le domaine correspondant. Le résultat de cette évaluation, si le résultat appartient au domaine, sera affecté à la variable.

Exemple 1 :

- Dans la définition suivante :

```
(defdom ENS-NUMERO (ens entier) (init () ))
```

la facette 'init' spécifie que l'ensemble vide est la valeur initiale à effecter à toutes les variables qui seront définies sur le domaine ENS-NUMERO, dans le cas où l'argument <val-init> n'est pas fourni.

- Dans les déclarations suivantes :

```
(defvar E1 ENS-NUMERO (23 45 67))
--> (23 45 67)

(defvar E2 ENS-NUMERO )
--> ()
```

La valeur initiale de la variable E1 est la liste (23 45 67) et celle de la variable E2 est l'ensemble vide.

Exemple 2 :

Dans la définition suivante :

```
(defdom UNE-FONCTION
        (un chaine)
        (verifier (inclus $VAL
                          ("ouvrier"  "magasinier"
                           "secretaire" comptable")))
        (init "ouvrier"))
```

la facette 'init' permet de spécifier que la valeur par défaut, pour une variable déclarée sur le domaine UNE-FONCTION, sera la valeur "ouvrier".

Exemple 3 :

Considérons la table FONC-SAL suivante :

Fonction	Salaire de base
ouvrier	4000.00
livreur	4500.00
magasinier	5000.00
secretaire	5500.00
representant	7000.00
comptable	8000.00

Cette table peut être établie ainsi :

```
(defdom UN-FONC-SAL
        (FONCTION (un chaine))
        (SALAIRE  (un reel)))

(defdom ENS-FONC-SAL (un UN-FONC-SAL))
```

```
(defvar FONC-SAL ENS-FONC-SAL
        ( ("ouvrier"      4000.00)
          ("livreur"      4500.00)
          ("magasinier"   5000.00)
          ("secretaire"   5500.00)
          ("representant" 7000.00)
          ("comptable"    8000.00) ) )
```

et définissons le domaine suivant :

```
(defdom UN-NOUVEAU
        (FONCTION
           (un chaine)
           (verifier
                    (inclus $VAL
                        (projeter (FONCTION) FONC-SAL)))
           (init "ouvrier"))

        (SALAIRE
           (un reel)
           (verifier (et (sup=  $VAL 3000.00)
                         (inf=  $VAL 9000.00)))
           (init (chercher (SALAIRE)
                           FONCSAL (L)
                      (egal (de L FONCTION)
                            (de $STRUCT FONCTION))))))
```

Explications

- Pour le champ FONCTION :

  La facette 'init' donne la valeur "ouvrier" comme valeur par défaut à ce champ.

  La facette 'verifier' permet de restreindre les valeurs qui seront affectées à ce champ, aux seules valeurs de la colonne FONCTION de la table FONC-SAL. Cette vérification s'effectue par l'expression :

  ```
  (inclus $VAL (projeter (FONCTION) FONC-SAL))
  ```

  dans laquelle la fonction projeter, qui réalise l'opérateur «projection» de l'algèbre relationnelle, retourne pour valeur l'ensemble constitué de toutes les valeurs du composant FONCTION de la table FONC-SAL et la fonction 'inclus' vérifiera que la valeur $VAL appartient à l'ensemble résultant.

- Pour le champ SALAIRE :

La facette 'verifier' permet de fixer les valeurs minimales et maximales pour ce champ.

La facette 'init' recherche la valeur par défaut pour ce champ dans la table FONC-SAL en fonction de la valeur courante du champ FONCTION du domaine UN-NOUVEAU. Cette recherche est réalisée par l'expression :

```
(chercher (SALAIRE)
          FONCSAL
          (L)
          (egal (de L FONCTION)
                (de $STRUCT FONCTION)))
```

dans laquelle la fonction chercher est utilisée pour retrouver l'élément du tableau FONC-TABLE, dont la valeur de la colonne FONCTION est égale à la valeur courante du champ FONCTION du domaine UN-NOUVEAU. Puis, la valeur du composant SALAIRE de cet élément est extraite pour être affectée comme valeur initiale par défaut de ce champ.

Remarque

Dans une expression, à l'intérieur d'une facette, la speudo-variable $STRUCT, si le domaine en cours de définition est une structure, contient la valeur complète de la donnée structurée.

2.1.3. La facette SI-BESOIN

La facette si-besoin permet de spécifier la valeur à retourner, lors de l'accès une donnée, si la valeur de cette donnée est la valeur NIL (valeur nulle).

La forme de cette facette est la suivante :

```
(si-besoin <expression>)
```

Où <expression> est une expression SHIVA qui sera évaluée si, lors de l'accès à une donnée définie sur le domaine correspondant, la valeur retournée est la valeur NIL. Le résultat de l'évaluation, s'il appartient au domaine de définition, sera la valeur effectivement retournée pour cette donnée.

Exemple 1 :

- considérons les déclarations suivantes :

```
(defdom ENS-NUMEROS (ens entier))
```

```
(defvar E ENS-NUMEROS)
--> nil

(defvar N entier (long E))
--> nil
```

La valeur initiale de N est NIL, puisque la valeur de l'ensemble E est NIL. Si N, par la suite est utilisée dans un calcul arithmétique (une moyenne par exemple), une erreur apparaitra, car la valeur nulle n'a aucun sens algébrique. Pour éviter une telle erreur, la valeur () doit être spécifiée soit avec la facette 'init'(dans ce cas la valeur NIL ne sera jamais admise à l'initialisation) soit avec la facette 'si-besoin' :

```
(defdom ENS-NUMEROS (ens entier)
                    (si-besoin () )

(defvar E ENS-NUMEROS)
-->nil

(defvar N entier (long E))
--> 0
```

Ici, la valeur initiale de l'ensemble E est NIL (en l'absence de la facette 'init' et d'une valeur initiale pour la fonction 'defvar'), mais la valeur initiale de N sera 0, car lors de l'accès à la valeur de E pour la fonction 'long' la valeur de l'ensemble vide est retournée et celui-ci est de longueur nulle.

Exemple 2 :

- Considérons les déclarations suivantes :

```
(defdom INFO-ENSEMBLE
   (un (VALEUR (ens entier)
               (si-besoin () ))
       (SOMME  (un entier)
               (si-besoin
                  (si (vide (de $STRUCT VALEUR))
                      0
                      (som (de $STRUCT VALEUR)))))))

(defvar INFO1 INFO-ENSEMBLE)
--> nil
```

La variable INFO1, définie sur le domaine INFO-ENSEMBLE est une donnée composée. Le champ VALEUR est un ensemble de nombres entiers et le champ SOMME est un entier qui, s'il est accédé et que sa valeur est NIL, aura pour valeur la valeur de la somme des éléments de l'ensemble VALEUR.

- affectation d'une valeur au champ VALEUR :

```
(dans INFO1 VALEUR (3 4 5 6))
--> (3 4 5 6)
```

- utilisation du champ SOMME :

```
(defvar S entier (de INFO1 SOMME))
--> 18
```

Remarque

Si une facette 'si-besoin' est spécifiée dans la définition d'un domaine, elle n'interdit pas l'accés en écriture aux données définies sur ce domaine. Ainsi, pour les affectations suivantes :

```
(dans INFO1 VALEUR (1 2))
--> (1 2)

(dans INFO1 (de INFO1 SOMME))
--> 3

(dans INFO1 VALEUR (1 2 3))
--> (1 2 3)

(de INFO1 SOMME)
--> 3
```

La valeur du champ SOMME, retournée par la dernière expression, est incohérente par rapport à la valeur du champ VALEUR qui a été modifiée.

2.2. L'attachement procédural

Sous le terme «d'attachement procédural» sont regroupés les concepts du modèle SHIVA qui permettent d'effectuer des opérations lors de l'apparition de certains évènements, c'est à dire :

- l'accès (en lecture ou en écriture) à une donnée,
- la mise à jour d'une donnée (accès en écriture),
- l'insertion d'un élément dans un ensemble,
- la suppression d'un élément d'un ensemble.

A chacun de ces évènements correspond une facette dont la forme générale est :

```
(<nom-facette> <condition> [ <action1> [ <action2> ]])
```

Où <nom-facette> est le nom d'une facette-évènement, <condition> est une expression conditionnelle et <action1>, <action2> sont des expressions SHIVA facultatives.

Lorsque l'évènement correspondant à <nom-facette> se réalise, l'expression <condition> est évaluée et selon le résultat (vrai ou faux) une des deux expressions (<action1> ou <action2>) sera évaluée.

Si l'évaluation de la condition retourne une valeur différente de la valeur NIL alors l'expression <action1> sera évaluée, sinon (valeur NIL) c'est l'expression <action2> qui le sera.

Dans le cas où la condition n'est pas vérifiée, aprés évaluation de l'expression <action2> (si elle existe), l'opération demandée sera refusée et la valeur NIL sera la valeur retournée par la fonction qui a déclenchée l'évènement.

Les facettes d'attachement procédural sont :

si-acc : cette facette sera prise en compte lors de tout accès en lecture ou en écriture à une donnée quelconque,

si-maj : cette facette sera prise en compte lors de tout accès en écriture (modification d'une valeur) à une donnée quelconque,

si-ins : cette facette sera prise en compte lors de toute insertion d'un élément dans un ensemble,

si-sup : cette facette sera prise en compte lors de toute suppression d'un élément d'un ensemble.

2.2.1. La facette SI-ACC

La facette si-acc sera en général utilisée pour contrôler l'accès à une variable et en interdire éventuellement la consultation.

Lors de tout accès à une donnée, pour laquelle la facette 'si-acc' est spécifiée, la valeur de cette donnée sera retournée si la condition associée à la facette est vérifiée, sinon la valeur NIL sera retournée pour cette donnée.

Dans le où la donnée accédée est structure ou un ensemble, si une facette 'si-acc' est spécifiée dans le domaine de la donnée, cette facette ne sera exploitée que si la donnée est globalement accédée.

Exemple 1 :

```
(defdom UN-EMPLOYE
   (un (NOM      (un chaine))
       (SALAIRE  (un reel))
       (SECRET   (un chaine)
                 (si-acc (egal $UTIL "Grand chef")))
   (si-acc (ou (egal $UTIL (de $STRUCT NOM)))
               (egal $UTIL "Grand chef"))))
```

Les données qui seront définies sur le domaine UN-EMPLOYE, ne seront accessibles en lecture que si le nom de l'utilisateur courant ($UTIL) est "Grand-chef" ou est égal à la valeur du champ NOM.

La valeur effective du champ SECRET ne sera retournée que pour l'utilisateur "Grand-chef", la valeur NIL sera retournée pour tous les autres.

Exemple 2 :

- Considérons le tableau EMPLOYES suivant :

Nom	Indice	Fonction	Salaire
Adam	10	ouvrier	100000
Barnabe	10	ouvrier	120000
Charles	11	ouvrier	130000
Simone	12	secretaire	135000
Berthe	13	secretaire	140000
Josette	16	comptable	190000
Georges	18	directeur	330000

- Soit la règle de confidentialité :

"Un employé dont l'indice est I
ne peut pas consulter le salaire d'un employé
dont l'indice est supérieur à I"

- Pour prendre en compte cette règle dans la définition du tableau EMPLOYES :

```
(defdom INFO-EMP
    (un (NOM      (un chaine))
        (INDICE   (un entier))
        (FONCTION (un chaine))
        (SALAIRE  (un entier)
                  (si-acc (inf=  (de $STRUCT INDICE)
                                 (F-INDICE $UTIL))))))
```

```
(defdom DES-EMP (ens UN-EMP))

(defvar EMPLOYES DES-EMP)
--> nil
```

Dans la définition d'un employé (domaine UN-EMP) la fonction utilisateur F-INDICE est utilisée dans la condition de la facette 'si-acc' associée au champ SALAIRE. La définition de la fonction utilisateur F-INDICE est la suivante :

```
(de F-INDICE entier
        ((NOM chaine)) ()
        (chercher (INDICE) EMPLOYES (P)
                (egal NOM (de P NOM))))
```

La fonction INDICE a pour but de rechercher l'indice d'un employé dont le nom est passé en paramètre. La fonction 'chercher' parcours l'ensemble EMPLOYES et recherche le premier élément pour lequel la condition est vérifiée. Si un tel élément existe, la valeur du champ INDICE est retournée. Pour chaque élément parcouru la variable P reçoit la valeur de l'élément et est utilisée lors de l'évaluation de la condition pour accéder aux différents champs de l'élément.

Lors de l'appel de la fonction F-INDICE, dans la facette 'si-acc', l'argument $UTIL est une speudo variable qui contient le nom de l'utilisateur d'une base SHIVA.

Si le concepteur de la base, met à la dispostion des utilisateurs la fonction suivante :

```
(defonc F-CATEGORIE DES-EMPS
        (FONCTION chaine)  ()
        (selecter () EMPLOYES () )
```

qui permet d'obtenir l'ensemble des informations sur une catégorie d'employés, selon l'indice de l'utilisateur, l'information sur l'indice sera fournie ou aura la valeur NIL.

- Par exemple, si l'employé de nom "Barnabe"
  désire la liste des ouvriers :

```
(LISTE-CATEG "ouvrier")
```

Le résultat sera, sous forme de tableau :

Nom	Indice	Fonction	Salaire
Adam	10	ouvrier	100000
Barnabe	10	ouvrier	120000
Charles	11	ouvrier	nil

2.2.2. La facette SI-MAJ

La facette si-maj permet d'effectuer des contrôles de cohérence lors de la mise à jour des donnée et d'interdire l'accès en écriture sous certaines conditions.

Dans le cas où la donnée accédée est une structure ou un ensemble, si une facette 'si-maj' est spécifiée dans le domaine de la donnée, cette facette ne sera exploitée que si la donnée est globalement mise à jour.

Exemple 1 :

```
(defdom INFO-NOMBRES
        (un (VALEUR (ens entier)
                    (si-besoin () ))
            (SOMME  (un entier)
                    (si-besoin
                        (som (de $STRUCT VALEUR)))
                    (si-maj nil ))))
```

La facette 'si-maj' associée au champ SOMME permet d'interdire tout accès en écriture à ce champ. Ceci permet de toujours calculer la somme des éléments du champ VALEUR quelque soit la valeur de ce champ.

Dans la facette 'si-maj' l'argument <condition> est la valeur NIL, ceci afin de signaler le refus de mise à jour du champ SOMME (la condition sera toujours fausse).

Exemple 2 :

A la règle de confidentialité, énoncée prédemment sur le tableau EMPLOYES, ajoutons les règles suivantes :

(1) Seuls le directeur et le comptable peuvent modifier les informations du tableau,

(2) le comptable ne peut pas modifier le salaire d'un employé,

(3) il ne peut exister de diminution d'indice ou de salaire,

(4) un salaire ne peut être augmenté de plus de 10%.

Pour prendre en compte ces règles, le domaine UN-EMP sera défini comme suit :

```
(defdom INFO-EMP
        (un (NOM       (un chaine))

            (INDICE    (un entier)
                       (verifier (positif+ $VAL))
                       (si-maj (sup $MAJ $VAL)))

            (FONCTION (un chaine))

            (SALAIRE   (un entier)
                       (si-maj (et (egal "directeur"
                                         (F-FONCTION $UTIL))
                                   (sup $MAJ $VAL)
                                   (inf=  (moins $MAJ $VAL)
                                          (div   $VAL 10))))
                       (si-acc (inf=  (de $STRUCT INDICE)
                                      (F-INDICE $UTIL)))))

        (si-maj (inclus (F-FONCTION $UTIL)
                        ("comptable" "directeur"))))
```

Dans cette définition, la fonction F-FONCTION est une fonction utilisateur qui retourne pour valeur la «fonction» de l'utilisateur courant. Cette fonction est définie comme suit :

```
(defonc F-FONCTION chaine
        ((NOM chaine)) ()
        (chercher (FONCTION) EMPLOYES (P)
                  (egal NOM (de P NOM))))
```

2.2.3. La facette SI-INS

La facette si-ins lorsqu'elle est spécifiée, pour un domaine défini à l'aide du constructeur 'ens', permet :

- de contrôler la valeur des éléments qui seront ajoutés à une variable définie sur ce domaine,

- de refuser l'ajout d'un nouvel élément si une certaine condition n'est pas remplie,

- d'exprimer des règles d'intégrité entre les éléments d'un ensemble,

- d'effectuer une ou plusieurs opérations lors de toute insertion d'un nouvel élément,

- et d'effectuer une ou plusieurs opérations si l'ajout d'un nouvel élément est refusé.

Exemple 1 :

Dans la définition suivante :

```
(defdom UN-ENSEMBLE
        (ens entier)
        (si-ins
          (si (inf= (long $VAL) 3)
              (ecrire "valeur" $ADD "acceptee")
              (ecrire "valeur" $ADD "refusee"))))
```

La facette 'si-ins' permet de spécifier que toute variable, définie sur ce domaine, devra contenir au plus 3 éléments. Ici, la pseudo-variable $ADD contient la valeur de l'élément à ajouter. Ainsi, pour la variable E définie sur ce domaine :

```
(defvar E UN-ENSEMBLE ())
--> ()

(ins-e E 1)
"valeur 1 acceptee"
--> (1)

(ins-e E 2)
"valeur 2 acceptee"
--> (2 1)

(ins-e E 3)
"valeur 3 acceptee"
--> (3 2 1)

(ins-e E 4)
"valeur 4 refusee"
--> (3 2 1)
```

Les trois premières insertions sont acceptées, mais la dernière est refusée et dans ce cas la valeur retournée est l'ancienne valeur de l'ensemble.

Dans le cas d'une affection globale à un ensemble, la facette 'si-ins' sera exploitée pour toutes les valeurs de l'affectation. Par exemple :

```
(affecter E (3 4 5 6))
"valeur 3 acceptee"
"valeur 4 acceptee"
"valeur 5 acceptee"
"valeur 6 refusee"
--> (3 4 5)
```

Exemple 2 :

Dans le domaine suivant :

```
(defdom ENS-CHAINES
        (ens chaine)
        (si-ins
          (si (non (inclus $ADD $VAL)))))
```

la facette 'si-ins' spécifie que les valeurs en doubles sont interdites dans tous les ensembles qui seront définis sur ce domaine.

Exemple 3 :

Dans les définitions suivantes, la valeur du champ NUMERO (lors de tout ajout d'un nouvel employé), est entièrement géré par le système de gestion de données.

```
(defdom UN-EMPLOYE
        (un (NUMERO (un entier))
            (NOM    (un chaine)))

(defdom ENS-EMPLOYES
        (ens UN-EMPLOYE)
        (init nil)
        (si-besoin ())
        (si-maj nil)

        (si-ins
            (si (et (non (de $ADD NUMERO))
                    (non (inclus $ADD $VAL)))
                (dans $ADD NUMERO
                    (si COMPTEUR
                        (affecter COMPTEUR
                                  (plus 1 COMPTEUR))
                        (defvar COMPTEUR entier 1))))))

(defvar TABLEAU ENS-EMPLOYES)
--> nil
```

La valeur initiale de la variable TABLEAU sera toujours la valeur NIL (facette 'init') et de plus toute mise à jour globale est interdite (facette 'si-maj').

Lors de l'ajout d'un élément, la facette 'si-ins' oblige la valeur du champ NUMERO, de ce nouvel élément, à être la valeur NIL (première partie de la condition) et n'autorise pas de valeurs dupliquées pour le champ NOM (deuxième partie de la condition).

Si ces conditions sont vérifiées, l'élément sera accepté et la valeur du champ NUMERO calculée.

Le calcul de la valeur du numéro est réalisé par les actions spécifiées dans la fonction 'si' de la manière suivante :

- s'il existe une variable de nom COMPTEUR, celle-ci est incrémentée de 1 est le résultat donne la valeur du numéro;
- sinon, la variable COMPTEUR est créée, avec la valeur initiale 1 qui sera la valeur de départ du numéro.

2.2.4. La facette SI-SUP

La facette si-sup lorsqu'elle est spécifiée, pour un domaine défini à l'aide du constructeur 'ens', permet :

- de contrôler l'élimination d'un élément d'un ensemble;

- de refuser la suppression d'un ou plusieurs éléments sous certaines conditions;

- de déclencher des opérations lors de la suppression d'un élément.

- d'effectuer des opérations si la suppression d'un élément est refusée.

Exemple 1 :

Dans la définition suivante :

```
(defdom ENSEMBLE
        (ens entier)
        (si-ins (inf=  5 (long $VAL)))
        (si-sup (sup=  3 (long $VAL)))
        (si-maj (et (inf=  5 (long $VAL))
                    (sup=  3 (long $VAL)))))
```

les facettes spécifiées ici, feront que les toutes variables déclarées sur le domaine ENSEMBLE, devront avoir au moins trois éléments et au plus cinq.

Exemple 2 :

Considérons la table SERVICES suivante :

Services	Num. Employes
comptabilite	1 5 7 8
magasin	2 6 9 12 10
fabrication	4 3 11 17 15
direction	18 13 14 16 21

définie à l'aide des domaines :

```
(defdom LISTE-EMPLOYES (ens entier))

(defdom UN-SERVICE
        (NOM-SERVICE (un chaine))
        (NUM-EMPLOYES LISTE-EMPLOYES))

(defdom ENS-SERVICES
        (ens UN-SERVICE))
```

Pour suprimer le numéro d'un employé dans la table SERVICES lors de la suppression d'un employé du tableau EMPLOYES, la facette suivante :

```
(si-sup (SUP-EMP-SERVICE (de $SUP NUMERO)))
```

doit être ajoutée dans le domaine ENS-EMPLOYES sur lequel est défini le tableau EMPLOYES.

Dans cette facette la speudo-variable $SUP contient la valeur de l'élément devant être supprimé et la fonction SUP-EMP-SERVICE dont la définition est la suivante :

```
(defonc SUP-EMP-SERVICE entier
   ((NUM-EMP entier))
   ((LISTE LISTE-EMPLOYES))
   (pour SERVICES (S)
        (faire
        (affecter LISTE (de S NUM-EMPLOYES))
        (si (inclus NUM-EMP LISTE)
            (dans S NUM-EMPLOYES
                    (sup-e NUM-EMP LISTE)))
        1)))
```

permet de supprimer un numéro d'employé de la liste des numéros d'employés d'un service.

Lors de l'exploitation de la facette 'si-ins', l'action de la suppression du numéro de l'employé de la table SERVICES sera effectuée avant la suppression de l'employé du tableau EMPLOYES.

3. Les opérateurs relationnels

Tous les opérateurs qui suivent, ne peuvent être utilisés que pour des variables dont le domaine est construit avec le constructeur 'ens' et dont les éléments sont des structures (constructeur 'un' suivi d'au moins un nom de champ). De telles variables, seront appellées par la suite : ensembles structurés.

3.1. La jointure

```
(joindre (<nom-var1> [ ( <liste-champ1> ) ] )
         (<nom-var2> [ ( <liste-champ2> ) ] )

         (<nom-fonction> <champ-1> <champ-2>))
```

La fonction joindre réalise l'opérateur de jointure de l'algèbre relationnelle. Cette jointure s'effectue entre les éléments de l'ensemble structuré de nom <nom-var1> et les éléments de l'ensemble structuré de mom <nom-var2>.

Les arguments <champ-1> et <champ-2> sont des noms de composantes des ensembles structurés sur lesquels s'effectue la jointure. L'argument <champ-1> étant le nom d'une composante de <nom-var1> et l'argument <champ-2> le nom d'une composante de <nom-var2>.

L'argument <nom-fonction> est le nom d'une fonction SHIVA ou d'une fonction utilisateur devant posséder deux parmètres d'appel.

Les arguments <liste-champ1> et <liste-champ2> sont des noms de composantes des ensembles <nom-var1> et <nom-var2> respectivement.

La valeur retournée par la fonction de jointure est un ensemble structuré, les éléments de cet ensemble sont constitués :

- de toutes les composantes de l'ensemble <nom-var1> si la liste <liste-champ1> est la liste vide, ou des composantes de <nom-var1> citées dans cette liste,

- et d'une composante de nom <nom-var2>, elle même structurée et composée de toutes les composantes de l'ensemble <nom-var2> si la liste <liste-champ2> est vide, ou des composantes de <nom-var2> dont le nom est cité dans cette liste.

Les éléments de l'ensemble résultat sont constitués de la manière suivante :

- A chaque élément de l'ensemble <nom-var1> correspond un élément de l'ensemble résultat,

- les valeurs des composantes <liste-champ1> de l'élément résultat sont la copie des valeurs de ces composantes dans l'élément de l'ensemble <nom-var1>,

- la valeur de la composante de nom <nom-var2> de l'élément résultat est un ensemble dont les éléments sont tous les éléments de l'ensemble <nom-var2> tels que l'évaluation de la fonction <nom-fonction> avec pour premier argument la valeur de la composante <champ-1> de l'ensemble <nom-var1> et pour deuxième argument la valeur de la composante <champ-2> de l'ensemble <nom-var2>, retourne une valeur différente de NIL.

- les valeurs des composantes <liste-champ2> de la composante <nom-var2> de l'ensemble résultat, sont la copie des valeurs de ces composantes pour les éléments de l'ensemble <nom-var2> qui constituent la valeur de la composante <nom-var2>.

Exemple 1 :

- Considérons les tableaux suivants :

Relation NOMS

NUM-EMP	NOM	PRENOM
4	Dupont	Octave
6	Durand	Leo
8	Lesage	Alfred
9	Al-Dor	Zoe

Relation FONCTIONS

NUM-EMP	FONCTIONS
4	ouvrier
6	ouvrier
6	magasinier
8	magasinier
8	comptable
9	secretaire
9	secr. dir.

Relation SALAIRES

NUM-EMP	JJ	MM	AA	SALAIRES
4	11	01	78	3500.50
4	01	01	79	4000.00
6	21	05	77	3300.00
6	10	05	78	4500.00
6	10	04	77	5000.67
8	01	01	78	3500.45
8	01	01	78	5500.50
8	10	05	78	7800.50
9	11	03	77	5200.00
9	01	01	78	7500.00
9	10	06	78	7800.70
9	01	01	79	8000.00

- L'expression :

(joindre (NOMS) (FONCTIONS) (egal NUM-EMP NUM-EMP))

permet d'obtenir la table suivante :

NUM-EMP	NOM	PRENOM	FONCTIONS	
			NUM-EMP	FONCTIONS
4	Dupont	Octave	4	ouvrier
6	Durand	Leo	6	ouvrier
			6	magasinier
8	Lesage	Alfred	8	magasinier
			8	comptable
9	Al-Dor	Zoe	9	secretaire
			9	secr. dir.

Exemple 2 :

- L'expression :

```
(joindre (NOMS      (NUM-EMP NOM))
         (SALAIRES (JJ MM AA SALAIRE))
         (egal NUM-EMP NUM-EMP))
```

permet d'obtenir la table suivante :

NUM-EMP	NOM	SALAIRES	
		JJ MM AA	SALAIRES
4	Dupont	11 01 78	3500.50
		01 01 79	4000.00
6	Durand	21 05 77	3300.00
		10 05 78	4500.00
		10 04 77	5000.67
8	Lesage	01 01 78	3500.45
		01 01 78	5500.50
		10 05 78	7800.50
9	Al-Dor	11 03 77	5200.00
		01 01 78	7500.00
		10 06 78	7800.70
		01 01 79	8000.00

Dans cette jointure, la composante NUM-EMP de l'ensemble SALAIRE a été supprimée en citant explicitement les noms des composantes de l'ensemble à recopier.

Exemple 3 :

- L'expression

```
(joindre (SALAIRES (AA))
         (SALAIRES (SALAIRES NUM-EMP))
         (egal AA AA))
```

permet d'obtenir la table suivante :

AA	SALAIRES	
	SALAIRES	NUM-EMP
77	5200.00	9
	3300.00	6
	5000.67	6
78	3500.50	4
	4500.00	6
	3500.45	8
	5500.50	8
	7800.50	8
	7500.00	9
	7800.70	9
79	8000.00	9
	4000.00	4

Dans le cas où les noms des ensembles de la fonction 'joindre' correspondent au même ensemble structuré, la fonction 'joindre' permet de réaliser l'opération de regroupement : nest de l'algèbre relationnelle étendue.

3.2. L'éclatement

(eclater <nom-var> [(<liste-champs>)])

La fonction eclater réalise l'opérateur unest de l'algèbre relationnelle étendue, en effectuant pour chaque élément d'un ensemble structuré, le produit cartésien des valeurs de certaines composantes de l'élément.

Pour cette fonction, l'argument <nom-var> est le nom d'un ensemble structuré et l'argument <liste-champs> est une liste optionnelle de noms de composantes de l'ensemble <nom-var> qui, si elle est omise aura pour valeur la liste de toutes les composantes de l'ensemble <nom-var>.

La valeur de la fonction 'eclater' est un ensemble structuré, ayant même nombre d'éléments que l'ensemble <nom-var> et, tel que tout élément de l'ensemble <nom-var> produit un élément dans l'ensemble résultat, obtenu de la manière suivante :

- si l'argument <liste-champ> est omis, l'élément résultat a pour valeur un ensemble dont les éléments sont le résultat du produit cartésien de toutes les valeurs des composantes de l'élément correspondant dans l'ensemble <nom-var>;

- si l'argument <liste-champ> est présent, l'élément résultat a pour valeur, la valeur de toutes les composantes non citées dans la liste et une composante, elle même structurée à partir des composantes citées et dont la valeur est l'ensemble des éléments obtenus en effectuant le produit cartésien des seuls valeurs des composantes citées.

Exemple 1 :

Considérons la table EMPLOYES suivante :

NOMS	ENFANTS	HISTORIQUE	
		DATE	SALAIRE
Dupont	Arsene	1975	100000
	Leopold		
	Gertrude	1976	120000
	Mathias		
Durand	Arsene	1975	100000
	Aristide	1976	110000
Dupre		1975	220000
		1976	230000
		1977	250000

- L'expression :

(eclater EMPLOYES (NOMS ENFANTS))

permet d'obtenir la table :

NOMS	ENFANTS	HISTORIQUE	
		DATE	SALAIRE
Dupont	Arsene	1975	100000
Dupont	Leopold		
Dupont	Gertrude	1976	120000
Dupont	Mathias		
Durand	Arsene	1975	100000
Durand	Aristide	1976	110000
Dupre	nil	1975	220000
		1976	230000
		1977	250000

- Alors que l'expression :

(eclater EMPLOYES)

donne la table :

NOMS	ENFANTS	HISTORIQUE	
		DATE	SALAIRE
Dupont	Arsene	1975	100000
Dupont	Arsene	1976	120000
Dupont	Leopold	1975	100000
Dupont	Leopold	1976	120000
Dupont	Gertrude	1975	100000
Dupont	Gertrude	1976	120000
Dupont	Mathias	1975	100000
Dupont	Mathias	1976	120000
Durand	Arsene	1975	100000
Durand	Arsene	1976	110000
Durand	Aristide	1975	100000
Durand	Aristide	1976	110000
Dupre	nil	1975	220000
Dupre	nil	1976	230000
Dupre	nil	1977	250000

3.3. La recherche

```
(chercher  ( [ <champ-i> ] ) <nom-var>
                [ <nom-elem>    <expression> ] )
```

La fonction chercher a pour but de parcourir un ensemble structuré et de retourner pour valeur, le premier élément de cet ensemble qui vérifie une condition.

Pour cette fonction, l'argument <nom-var> est l'ensemble structuré à parcourir, <nom-elem> est le nom d'une variable temporaire qui servira de variable de parcours et qui pourra être utilisée dans la condition de recherche. L'argument <expression> est une expression SHIVA quelconque, qui sera évaluée pour chaque élément de l'ensemble et l'argument <champ-i> est une liste optionnelle de noms de composantes de l'ensemble structuré.

La valeur de la fonction 'chercher' est la valeur des composantes <champ-i> du premier élément de l'ensemble pour lequel l'évaluation de <expression> retourne une valeur non NIL. Si la liste <champ-i> est vide, la valeur retournée est l'élément complet qui vérifie la condition.

Si aucun élément de l'ensemble structuré ne vérifie la condition, la fonction 'chercher' retourne la valeur NIL.

Exemple 1 :

Considérons le tableau EMPLOYES suivant :

NOMS	ENFANTS	HISTORIQUE	
		DATE	SALAIRE
Dupont	Arsene	1975	100000
	Leopold		
	Gertrude	1976	120000
	Mathias		
Durand	Arsene	1975	100000
	Aristide	1976	110000
Dupre		1975	220000
		1976	230000
		1977	250000

- L'expression :

```
(chercher (NOM) EMPLOYES (UN-EMPLOYE)
          (vide (de UN-EMPLOYE ENFANT)))
```

retourne la valeur : "Dupre"

Cette expression permet d'obtenir le nom de la première personne de tableau EMPLOYES qui n'a pas d'enfant.

Exemple 2 :

- L'expression :

```
(chercher (NOMS HISTORIQUE) EMPLOYES
          (UN-EMPLOYE)
          (chercher () (de UN-EMPLOYE HISTORIQUE)
                    (UN-HISTO)
                    (et (egal (de UN-HISTO DATE) 1976)
                        (sup= (de UN-HISTO SALAIRE) 200000))))
```

retourne la valeur :

```
("Dupre" ( (1975 220000)
           (1976 230000)
           (1977 250000) ) )
```

Cete expression permet d'obtenir le nom et l'historique de la première personne dont le salaire en 1976 est supérieur ou égal à 200000.

Ici, la composante HISTORIQUE de la variable de parcours UN-EMPLOYE est utilisée dans une fonction 'chercher' pour obtenir un élément HISTORIQUE vérifiant la condition sur la date et le salaire. Si un tel élément existe, la deuxième fonction 'chercher' retourne la valeur de cet élément et dans ce cas la condition de la première fonction 'chercher' est vérifiée. Si pour un employé du tableau il n'existe pas d'élément HISTORIQUE vérifiant la condition sur la date et le salaire, la deuxiéme fonction 'chercher' retourne la valeur NIL et dans ce cas,

pour l'employé considéré, la condition de la première fonction 'chercher' a la valeur NIL et l'employé n'est pas pris en compte.

3.4. La projection-sélection

```
(selecter  ( [ <champ-i> ] ) <nom-var>
                [ <nom-elem>    <expression> ] )
```

La fonction selecter combine les l'opérations de projection et de sélection de l'algèbre relationnelle.

L'argument <nom-var> est le nom de l'ensemble structuré sur lequel on veut appliquer une opération de sélection.

L'argument <nom-elem> est le nom d'une variable temporaire qui recevra la valeur de tous les éléments de l'ensemble durant son parcours et qui pourra être utilisée par l'expression de sélection.

L'argument <expression> est une expression SHIVA quelconque, qui exprime la condition de sélection.

Les arguments <champ-i> sont des noms de champs de la variable structurée <nom-var>.

La valeur retournée par la fonction 'selecter' est un ensemble structuré, constistué de tous les éléments de la variable <nom-var> pour lesquels l'argument <expression>, aprés évaluation pour chaque élément, retourne une valeur autre que NIL.

Seuls les champs, dont les noms sont donnés par l'argument <champ-i>, sont recopiés, pour les éléments qui vérifient la condition.

Si l'argument <champ-i> est une liste vide, alors tous les champs de la variable structurée seront recopiés.

Si les arguments <nom-elem> et <expression> sont omis, alors l'ensemble résultat est constitué de tous les éléments de l'ensemble <nom-var>

Dans le cas où seuls les arguments <champ-i> et <nom-var> sont présents, alors la fonction 'selecter' réalise une opérat de projection classique.

Dans le cas où l'argument <champ-i> est une liste vide, alors la fonction 'selecter' réalise une opération de sélection classique.

Exemple 1 :

Considérons le tableau HISTORIQUE suivant

NUM-EMP	NOM	SALAIRES	
		JJ MM AA	SALAIRES
4	Dupont	11 01 78	3500.50
		01 01 79	4000.00
6	Durand	21 05 77	3300.00
		10 05 78	4500.00
		10 04 77	5000.67
8	Lesage	01 01 78	3500.45
		01 01 78	5500.50
		10 05 78	7800.50
9	Al-Dor	11 03 77	5200.00
		01 01 78	7500.00
		10 06 78	7800.70
		01 01 79	8000.00

- L'expression :

(selecter (NUM-EMP NOM) HISTORIQUE)

permet d'obtenir la table suivante :

NUM-EMP	NOM
4	Dupont
6	Durand
8	Lesage
9	Al-Dor

- L'expression :

(selecter (SALAIRES) HISTORIQUES)

donne la table :

SALAIRES	
JJ MM AA	SALAIRES
11 01 78	3500.50
01 01 79	4000.00
21 05 77	3300.00
10 05 78	4500.00
10 04 77	5000.67
01 01 78	3500.45
01 01 78	5500.50
10 05 78	7800.50
11 03 77	5200.00
01 01 78	7500.00
10 06 78	7800.70
01 01 79	8000.00

Remarque

L'ensemble résultant ne possède que quatre éléments. Chaque élément étant lui même un ensemble.

Exemple 2 :

Considérons la table EMPLOYES suivantes :

NOMS	ENFANTS	HISTORIQUE	
		DATE	SALAIRE
Dupont	Arsene	1975	100000
	Leopold		
	Gertrude	1976	120000
	Mathias		
Durand	Arsene	1975	100000
	Aristide	1976	110000
Dupre		1975	220000
		1976	230000
		1977	250000

- L'expression :

```
(selecter (NOMS) EMPLOYES (UN-EMPLOYE)
                  (inclus "Arsene" (de UN-EMPLOYE ENFANTS)))
```

donne la table :

NOMS
Dupont
Durand

- L'expression :

```
(selecter () EMPLOYES (UN-EMPLOYE)
          (chercher () (de UN-EMPLOYE HISTORIQUE)
                    (UN-HISTO)
                    (sup= (de UN-HISTO DATE) 1976)))
```

Retourne pour valeur l'enemble des «employés» travaillant depuis 1976.

3.5. Le produit cartésien

```
(cartesien <nom-var1> <nom-var2>)
```

La fonction cartesien permet d'effectuer le produit cartésien de deux ensembles structurés. Les argument <nom-var1> et <nom-var2> sont les noms des ensembles sur lequel le produit cartésien doit s'effectuer.

La valeur de la fonction 'cartesien' est un ensemble structuré ayant même nombre d'éléments que l'ensemble <nom-var1>. A chaque élément de l'ensemble <nom-var1> correspond un élément de l'ensemble résultat.

Un élément de l'ensemble résultat est composé de toutes les valeurs des composantes de l'élément de l'ensemble <nom-var1> et d'une composante dont la valeur est un ensemble. Cet ensemble étant composé de tous les éléments de l'ensemble <nom-var2>.

3.6. L'intersection

```
(inter <nom-var1> <nom-var2>)
```

La fonction inter permet d'effectuer l'intersection de deux ensembles structurés. Les arguments <nom-var1> et <nom-var2> étant les ensembles structurés dont on veut obtenir l'intersection. Ces deux ensembles devant être définis sur les mêmes domaines.

La valeur de la fonction 'inter' est un ensemble structuré qui contient une copie des éléments communs aux deux ensembles structurés.

3.7. L'union

```
(union <nom-var1> <nom-var2>)
```

La fonction union permet d'effectuer l'union de deux ensembles structurés. Les arguments <nom-var1> et <nom-var2> étant les ensembles structurés dont on veut obtenir l'union. Ces deux ensembles devant être définis sur les mêmes domaines.

La valeur de la fonction 'inter' est un ensemble structuré qui contient une copie des éléments de tous les éléments de l'ensemble <nom-var1> et de tous les éléments de l'ensemble <nom-var2>

3.8. La différence

```
(differ <nom-var1> <nom-var2>)
```

La fonction differ permet d'effectuer la différence de deux ensembles structurés. Les arguments <nom-var1> et <nom-var2> étant les ensembles structurés dont on veut obtenir la différence. Ces deux ensembles devant être définis sur les mêmes domaines.

La valeur de la fonction 'differ' est un ensemble structuré qui contient une copie de tous éléments de l'ensemble <nom-var1> qui n'existent pas dans l'ensemble <nom-var2>.

- CONCLUSION -

Nous avons presenté ici un modèle de données relationnel étendu conçu initialement pour permettre la définition et la réalisation de systèmes de gestion de bases de connaissances centrées objets. Comme nous l'avons montré en introduction et au chapitre 1, l'utilisation des bases de données pour ce type d'applications, nécessitent quelques extensions au modèle de données choisi. Ici, le modèle relationnel.

Les extensions demandées, en particulier le non respect de la première forme normale, ne sont pas caractéristiques de l'application SHIRKA. En effet, nous avons montré au chapitre 2, certaines recherches et réalisations d'enrichissement du modèle relationnel pour permettre son utilisation dans des applications telles que la bureautique, les bases de données en CAO et le traitement d'objets complexes.

Dans SHIVA, nous nous sommes naturellement intéressés à ces nouvelles fonctionnalités. Le modèle offre ainsi des possibilités de description et de gestion d'objets complexes, données structurées ou ensemble de valeurs et fourni de nouveaux opérateurs pour leur traitement et la prise en compte des valeurs nulles.

L'objectif du modèle SHIVA est de proposer un modèle de données permettant une meilleure correspondance entre le modèle de la représentation des connaissances centrées objet et le modèle relationnel. Pour atteindre cet objectif, nous nous sommes attachés à inclure certaines des caractéristiques de la représentation centrée objets, en particulier la notion de facette. Cette notion, qui permet notamment de spécifier des valeurs initiales ou par défaut, de calculer les valeurs manquantes et de déclencher des traitements lorsque se produisent certains évènements tels que la mise à jour ou la destruction de données, se retrouve entièrement dans le modèle SHIVA.

Par association de facettes aux domaines des attributs d'une relation ou au schéma complet d'une relation, nous pouvons exprimer des règles d'intégrité entre attributs ou relations. Nous pouvons aussi traiter les valeurs nulles, les accepter, les refuser ou les tolérer comme valeurs d'attributs ainsi que calculer les valeurs manquantes. Les facettes d'attachement procédural permettent, quant à elles, d'exprimer des règles pour maintenir la cohérence, la sécurité et la protection des données de la base. On pourra ainsi contrôler et réagir aux opérations d'accès et de mise à jour de la base.

Défini et réalisé dans le cadre du projet SHIRKA, le modèle SHIVA, grâce aux fonctionnalités apportées par les facettes, pourra être utilisé dans des applications de gestion de données plus classiques. Son emploi peut être envisagé dans des domaines tels que la bureautique et les bases de données numériques, la biométrique et l'économétrie par exemple. Dans les applications à caractère scientifique, on pourra utiliser les bases de données, non plus uniquement comme moyen de stockage de grandes quantité de données mais aussi pour intégrer les méthodes de dérivation de nouvelles valeurs qui, jusqu'à présent, s'effectuaient à l'extérieur et entrainaients de longs échanges de données.

Il reste, cependant un point qui n'a pu être complètement résolu pour permettre une totale correspondance des deux modèles. Il s'agit des relations de généralisation et de spécialisation entre classes d'objets. Ces deux notions ne sont pas apparentes dans SHIVA. Elles peuvent, néanmoins, être obtenues par la redéfinition de domaines et l'adjonction de nouveau attributs et par la définition et le maintien de relations inverses pour gérer l'ensemble des instances des objets appartenant aux mêmes classes d'une hiérarchie d'objets.

Ce dernier point, la hiérarchisation d'objets, est un des futurs axes de réflexion que nous comptons mener dans une prochaine extension du modèle SHIVA.

En ce qui concerne, la réalisation d'un interpréteur du modèle SHIVA, un point important pose encore des problèmes, il s'agit de l'entière gestion des données Lisp en mémoire. Pour SHIMER, utilisé actuellement par SHIRKA, ce fait est préjuciable car il conditionne d'une part le volume des bases de données et, d'autre part le temps d'évaluation des requêtes. En effet, aussi grand que puisse être l'espace mémoire virtuel, la gestion des cellules de liste est entièrement sous le contrôle de l'interpréteur Lisp. Il est par exemple impossible de rassembler en un même «lieu» mémoire toutes les valeurs d'un n-uplet donné ou de rendre «mitoyens» les n-uplets de deux relations intervenant dans une requête. Cette dispersion des données influence fortement les temps d'accès aux cellules dans un environnement de segmentation ou de pagination mémoire.

Pour obtenir une mise en oeuvre efficace et optimale du système de gestion de bases de connaissances de SHIRKA, la réalisation d'un système de gestion de données reposant sur le modèle SHIVA devra tenir compte au mieux des expériences acquises lors du développement de SHIMER. Les objectifs ne sont plus les mêmes, SHIMER a été conçu comme un système relationnel autonome, indépendament des applications possibles, alors que le modèle SHIVA sera developpé spécialement pour décrire et manipuler les données d'un système de gestion de bases de connaissances centrées objets. Toutefois, les possibilités de définition de fonctions utilisateurs et de vues feront que SHIVA pourra être utilisé comme système de gestion de bases de données.

SHIMER est un système ouvert, dont les services peuvent être partiellement adaptés aux besoins des utilisateurs, sans remise en cause des structures de données. Il n'en est pas de même dans SHIVA, où les structures de données influencent fortement l'exécution des opérateurs. Les mécanismes d'exploitation des facettes interviennent dans toutes manipulations de données, les opérateurs devront tous en tenir compte. Toutes les fonctionnalités du langage Lisp devront être utilisées pour rendre optimum les accès aux données et l'exécution des requêtes.

Aussi certains points devront tout particuliement être étudiés dans la réalisation d'un interpréteur du modèle SHIVA, il s'agit : de la gestion des facettes et de leurs prise en compte dans les fonctions d'accès, de l'écriture de ces fonctions, des méthodes d'accès aux données et du choix des structures pour la représentation des données tant en mémoire que sur les supports externes. Cependant, des tests, sur une maquette de l'interpréteur SHIVA travaillant en mémoire centrale, ont montré que les temps de réponses étaient trés satisfaisants lorsque que le volume de données était faible, mais augmentaient rapidement au fur et mesure de l'accroissement de ce volume.

Pour résoudre ce problème, une interface en langage C a été écrite et certaines fonctions de l'interpréteur Le_Lisp ont été redéfinies. Par un système de pagination de la base de données, cette interface permet de ne maintenir en mémoire (dans l'espace de listes) que les variables utiles à un moment donné. Cette interface est appelée par le «garbage collector» de l'interpréteur Le_Lisp et décharge dans la base les pages inutiles afin de libérer le maximum d'espace mémoire. L'interface est de même appelée, lors de l'accès à une variable dont les valeurs ne sont pas présentent en mémoire et la page contenant ces valeurs sera alors chargée.

Toutefois, l'introduction de ce niveau supplémentaire, entraîne la gestion de nouvelles structures de données et des contraintes sur la profondeur d'imbrication des ensembles. De plus, ce système de pagination interdit l'utilisation de certaines fonctions Lisp qui n'ont pas pu être redéfinies du fait de leur dépendance trés forte avec les mécanismes de gestion mémoire interne à l'interpréteur Le_Lisp. Une perte d'efficacité apparaît donc, car l'utilisation de ses fonctions pouvaient rendre plus optimale la gestion des structures de données de SHIVA.

Une première utilisation d'un interpréteur SHIVA pour la mise en œuvre du système de gestion de la base de connaissances de SHIRKA devrait s'effectuer dans trés prochainement. Dés à présent, on peut constater que si, par rapport à l'utilisation de SHIMER, la correspondance des données entre le modèle relationnel et le modèle SHIRKA, est meilleure, un certain niveau de traduction sera encore nécessaire. Aussi, à terme, une réflexion sur la nature et les spécificités d'un «vrai» système de gestion de base de connaissances centrées objets devra êtree menée.

- ANNEXE 1 - Ordre d'évaluation des facettes -

Les diagrammes des pages suivantes, montrent dans quel ordre les facettes d'intégrité, d'initialisation, de calcul et d'attachement procédural sont traitées lors de tout accès à une donnée.

Le premier diagramme : ACCES A UNE DONNEE, montre à quel moment la facette 'si-acc' est prise en compte. Puis, selon le type d'accès (lecture ou écriture), les autres diagrammes sont enchainés.

Le diagramme ACCES EN LECTURE, montre l'utilisation de la facette 'si-besoin' lors de l'accès en lecture à une variable ayant la valeur NIL.

Le diagramme ACCES EN MISE A JOUR, permet de différencier l'initialisation d'une variable de sa mise à jour.

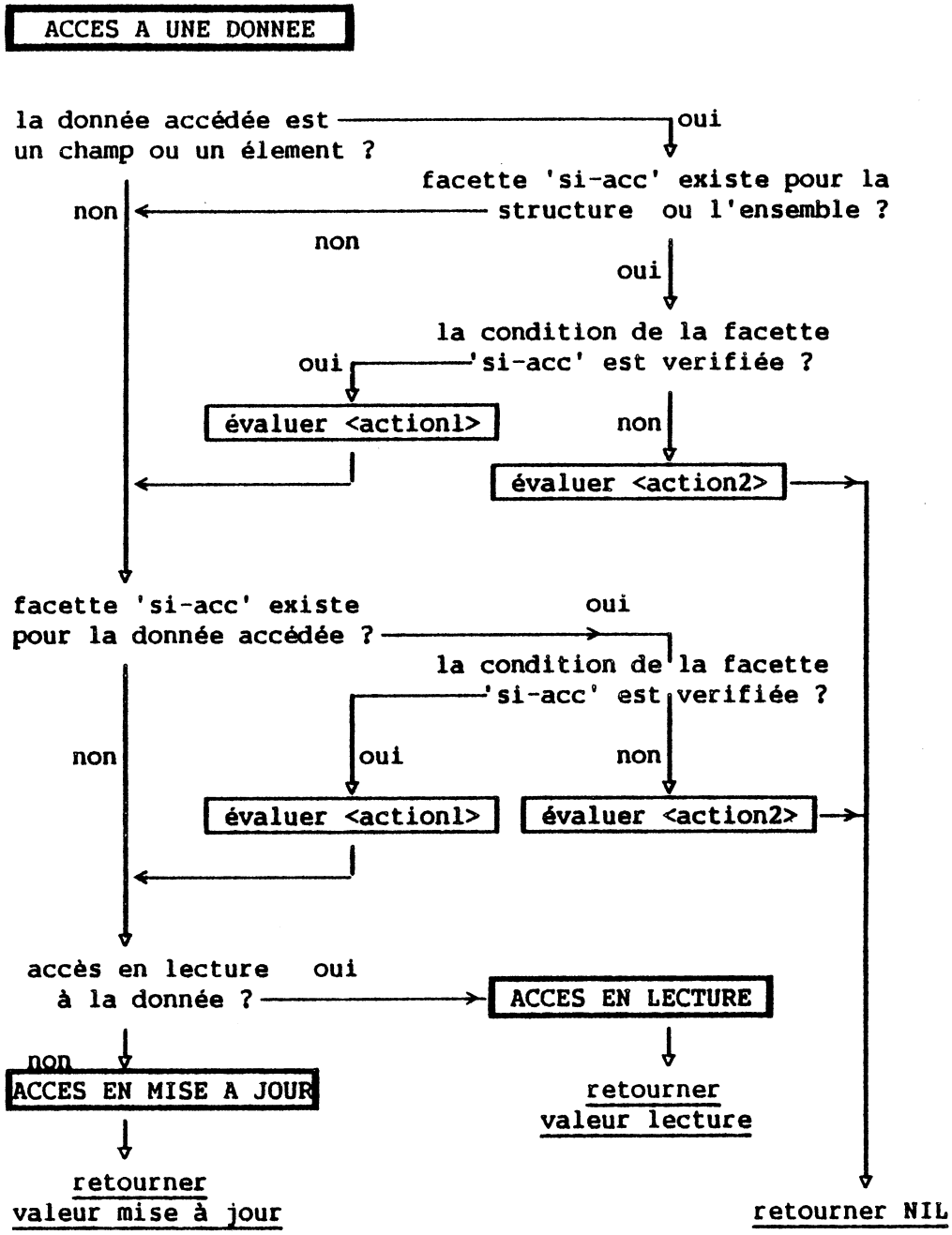
Le diagramme INITIALISATION, montre l'utilisation de la facette 'init' enchaîne sur les traitements de mise à jour selon le type de la donnée accédée.

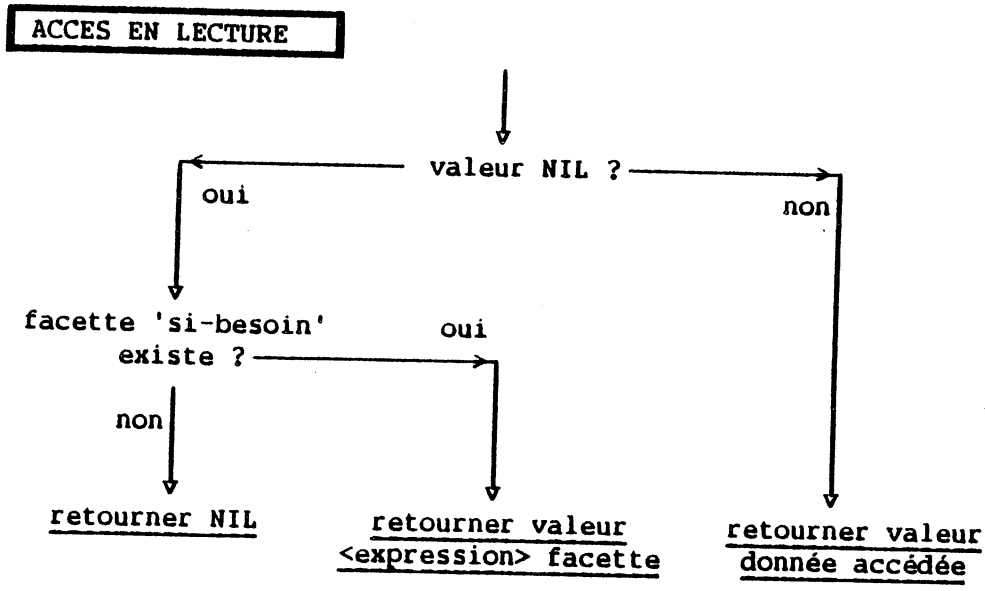
Le diagramme ACCES DONNEE SIMPLE, traite des mise à jour de données qui ne sont ni des structures ni des ensembles. Les facettes 'si-maj' et 'verifier' sont prises en compte dans ce traitement.

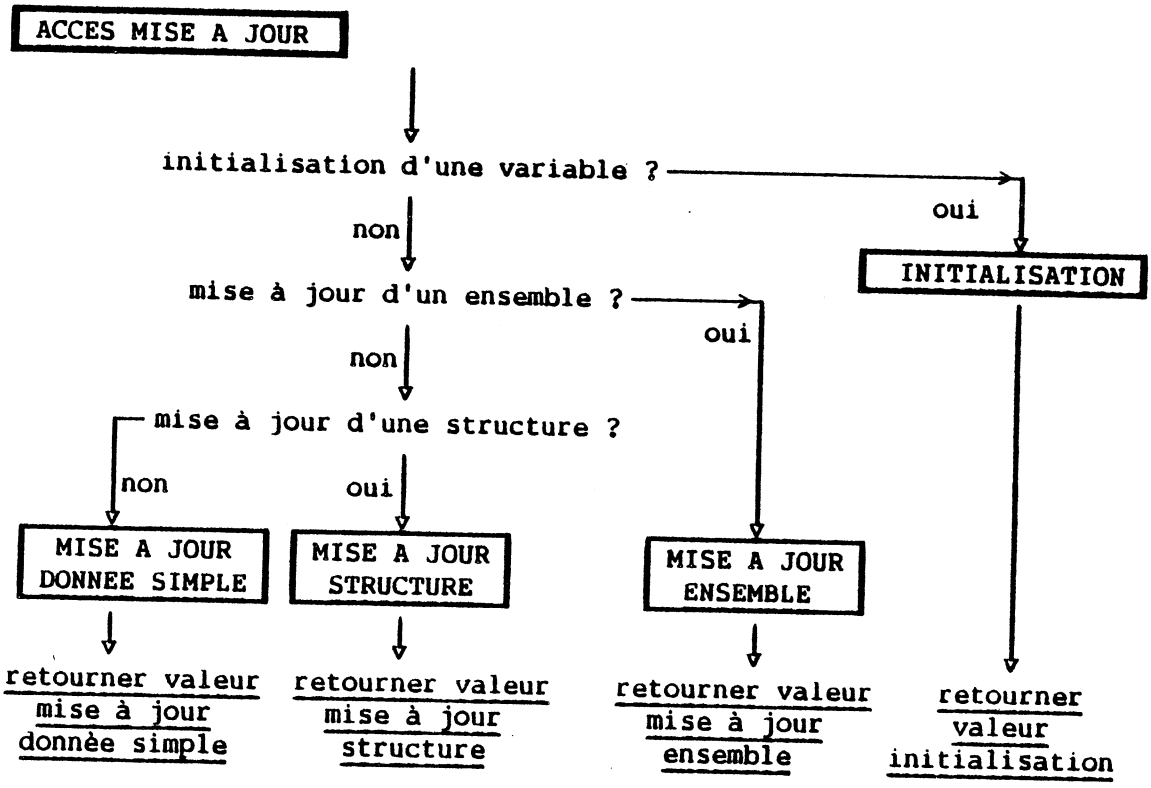
Le diagramme ACCES STRUCTURE, détaille les différentes opérations à effectuer selon le type d'accès à une structure : accès global ou accès à un champ.

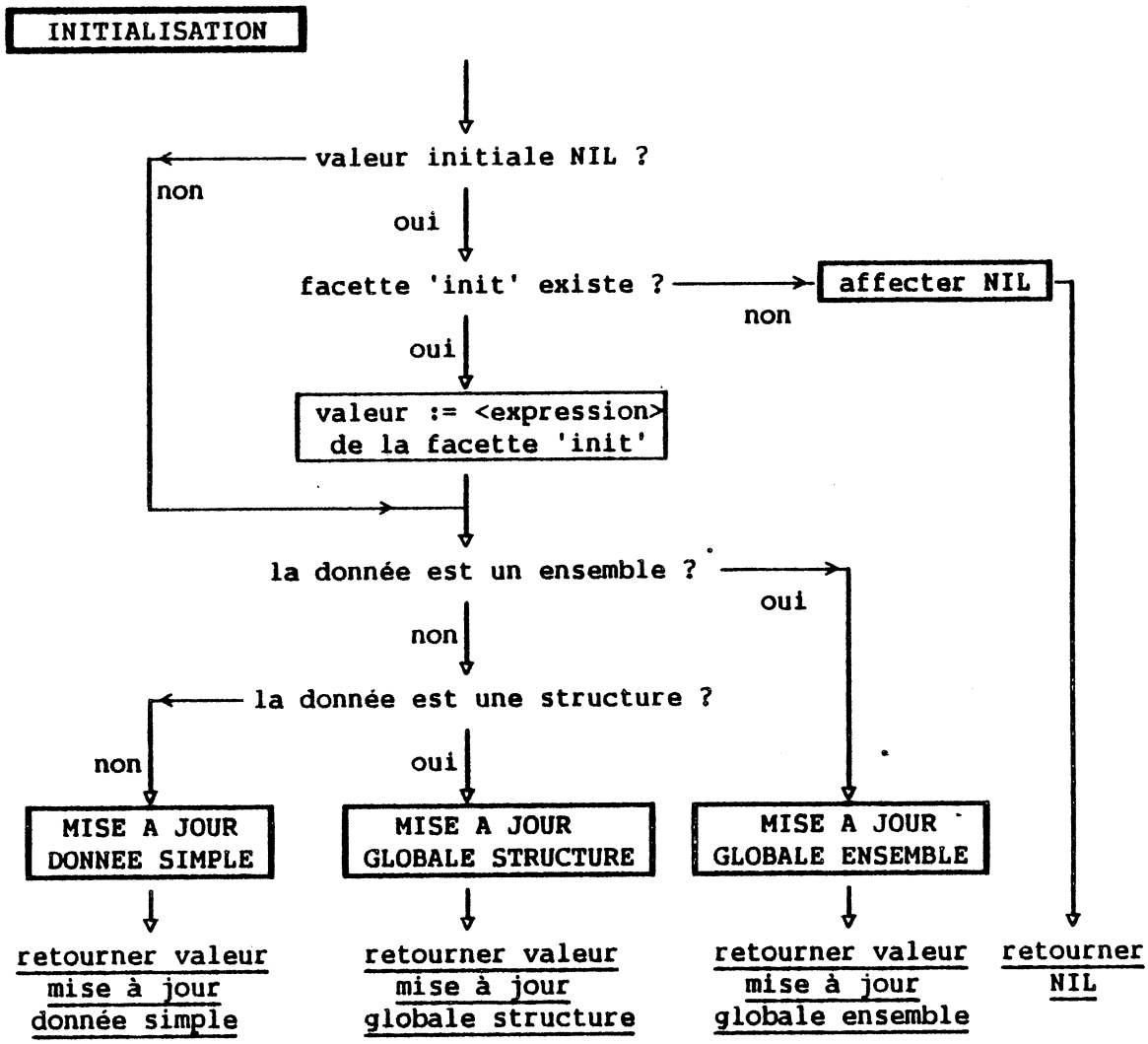
Le diagramme ACCES ENSEMBLE, différencie les cas de mises à jour d'un ensemble : mise à jour d'un élément, l'insertion ou la suppression d'un élément.

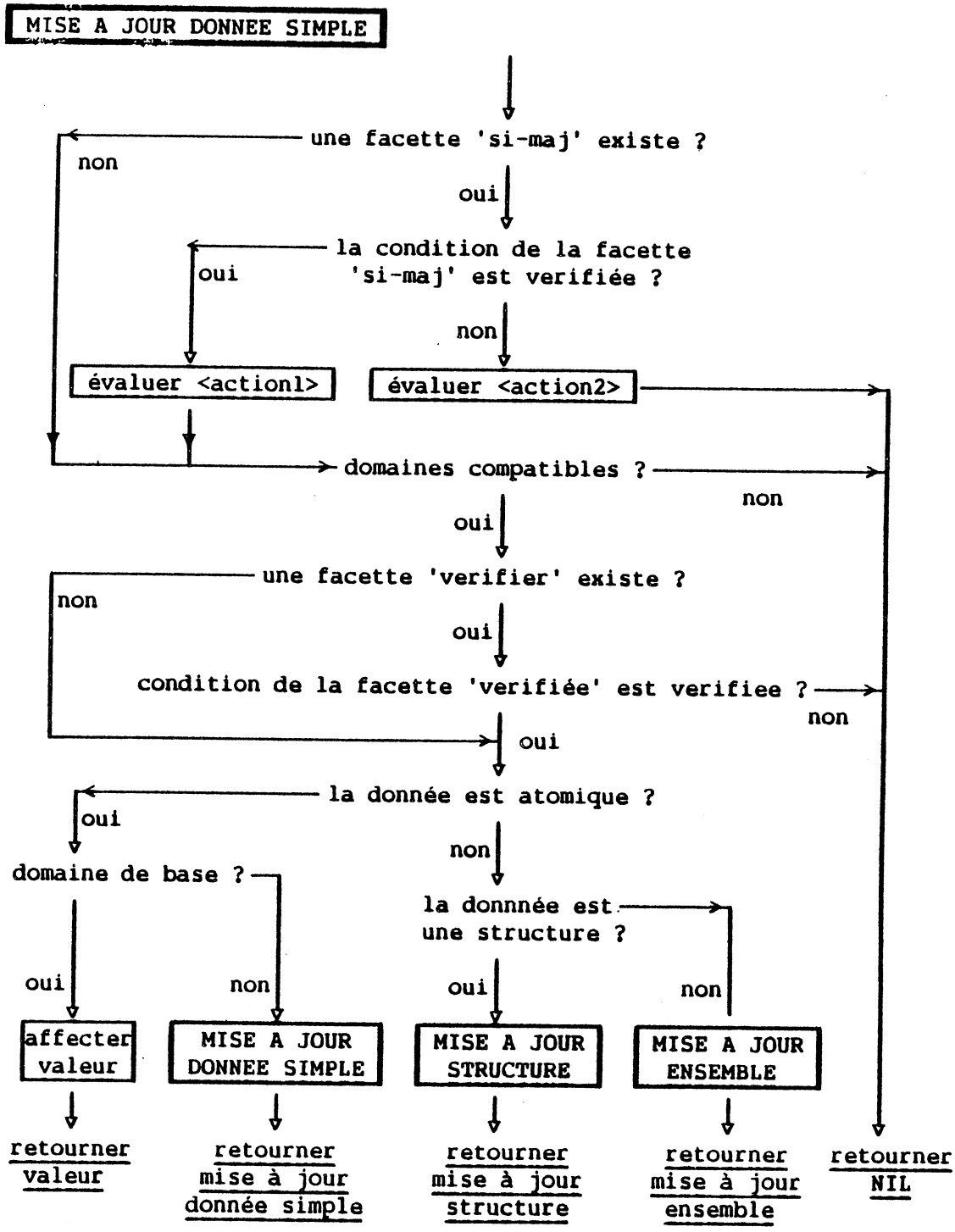
Enfin, les diagrammes INSERTION et SUPPRESSION traitent de l'insertion d'un élément dans un ensemble et de la suppression d'un élément d'un ensemble, respectivement. Les facettes 'si-ins' et 'si-sup' sont prises en compte dans ces diagrammes.

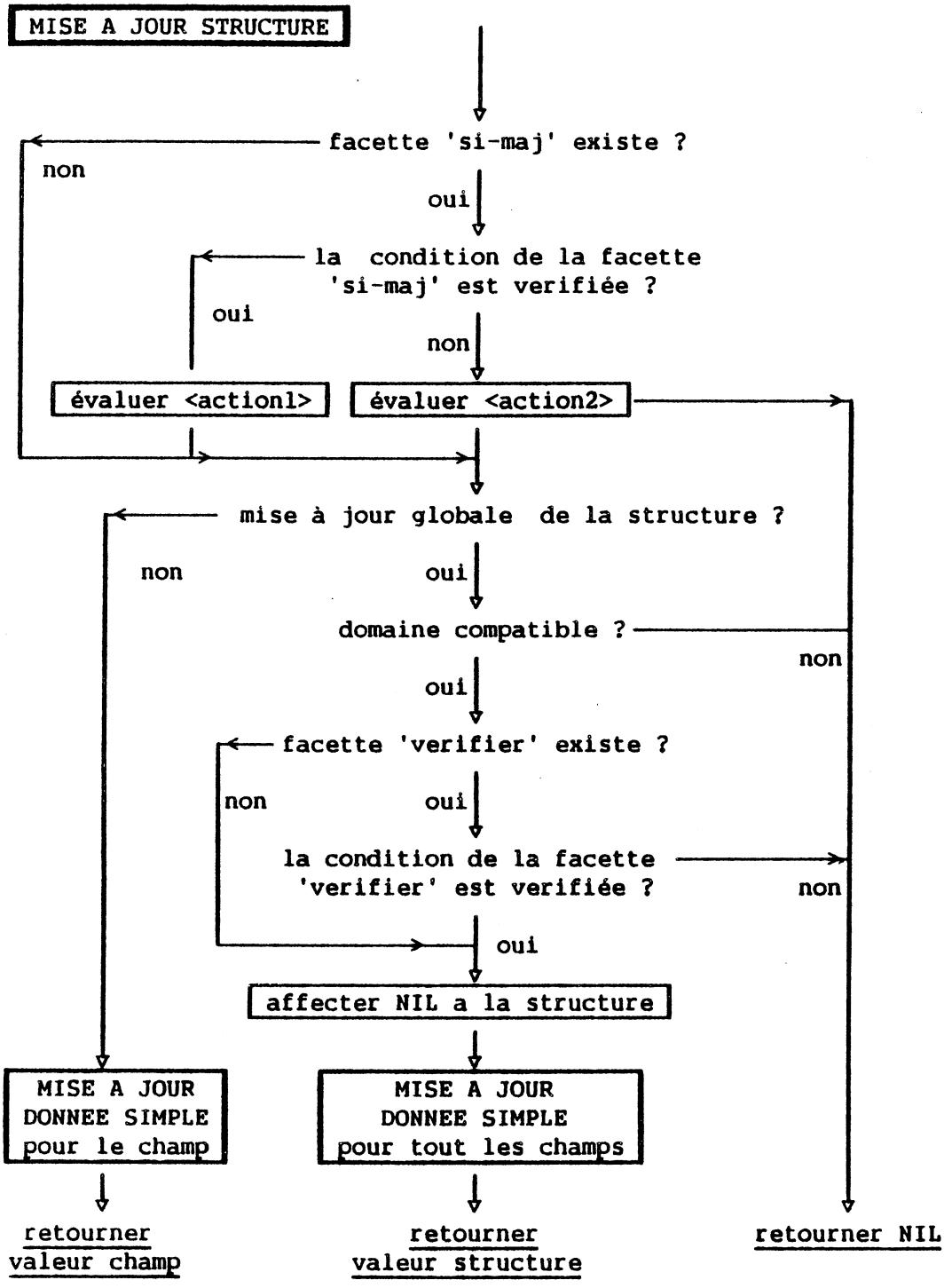

ACCES A UNE DONNEE
la donnée accédée est un champ ou un élement ?
oui
facette 'si-acc' existe pour la structure ou l'ensemble ?
non
non
oui
la condition de la facette 'si-acc' est verifiée ?
oui
évaluer <action1>
non
évaluer <action2>
facette 'si-acc' existe pour la donnée accédée ?
oui
la condition de la facette 'si-acc' est verifiée ?
non
oui
non
évaluer <action1>
évaluer <action2>
accès en lecture à la donnée ?
oui
ACCES EN LECTURE
non
ACCES EN MISE A JOUR
retourner valeur lecture
retourner valeur mise à jour
retourner NIL

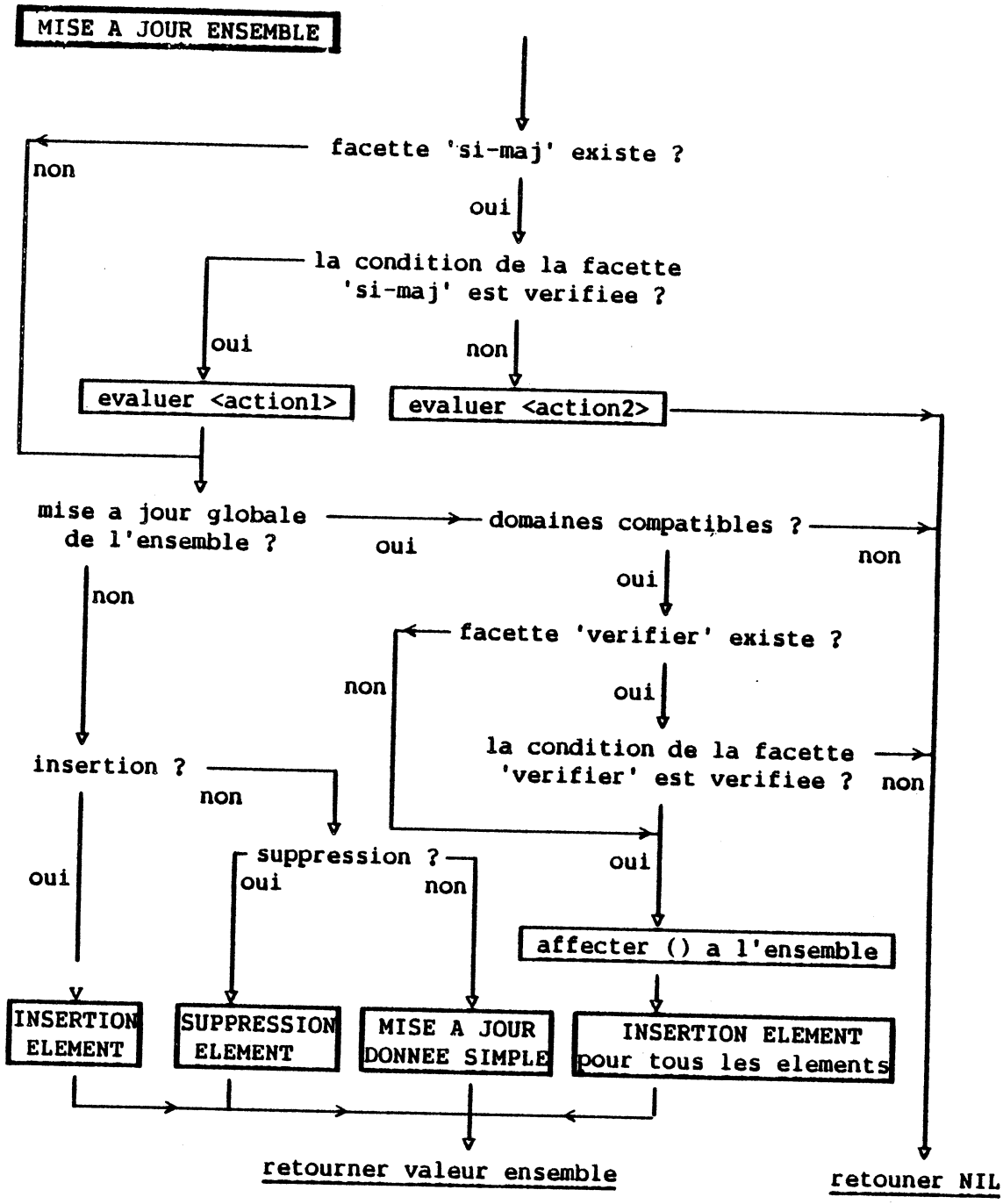

ACCES EN LECTURE
valeur NIL ?
oui
non
facette 'si-besoin' existe ?
oui
non
retourner NIL
retourner valeur <expression> facette
retourner valeur donnée accédée

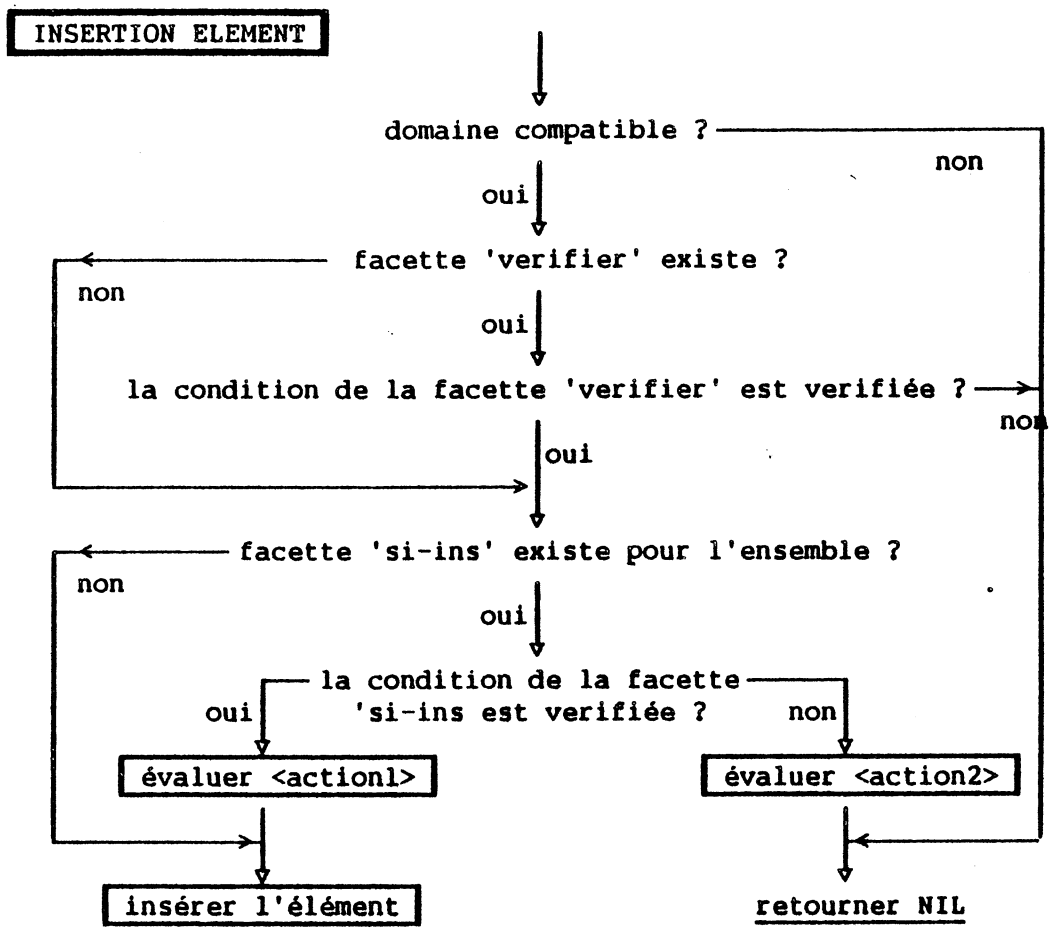

ACCES MISE A JOUR
initialisation d'une variable ?
oui
non
INITIALISATION
mise à jour d'un ensemble ?
oui
non
mise à jour d'une structure ?
non
oui
MISE A JOUR DONNEE SIMPLE
MISE A JOUR STRUCTURE
MISE A JOUR ENSEMBLE
retourner valeur mise à jour donnée simple
retourner valeur mise à jour structure
retourner valeur mise à jour ensemble
retourner valeur initialisation

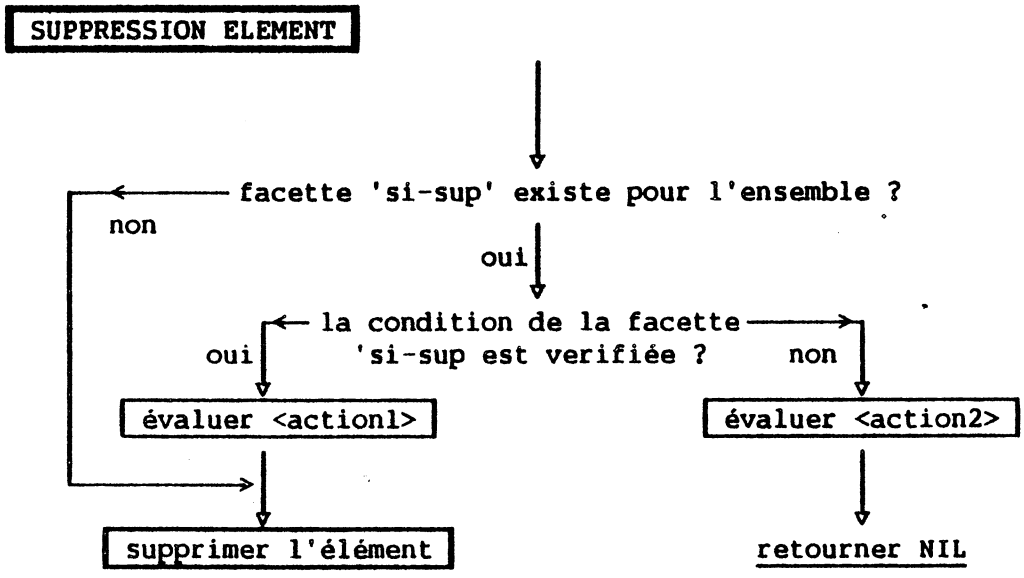

INITIALISATION
valeur initiale NIL ?
non
oui
facette 'init' existe ?
non
affecter NIL
oui
valeur := <expression>
de la facette 'init'
la donnée est un ensemble ?
oui
non
la donnée est une structure ?
non
oui
MISE A JOUR
DONNEE SIMPLE
MISE A JOUR
GLOBALE STRUCTURE
MISE A JOUR
GLOBALE ENSEMBLE
retourner valeur
mise à jour
donnée simple
retourner valeur
mise à jour
globale structure
retourner valeur
mise à jour
globale ensemble
retourner
NIL


MISE A JOUR DONNEE SIMPLE
une facette 'si-maj' existe ?
non
oui
la condition de la facette 'si-maj' est verifiée ?
oui
non
évaluer <action1>
évaluer <action2>
domaines compatibles ?
non
oui
une facette 'verifier' existe ?
non
oui
condition de la facette 'verifiée' est verifiee ?
non
oui
la donnée est atomique ?
oui
non
domaine de base ?
la donnnée est une structure ?
oui
non
oui
non
affecter valeur
MISE A JOUR DONNEE SIMPLE
MISE A JOUR STRUCTURE
MISE A JOUR ENSEMBLE
retourner valeur
retourner mise à jour donnée simple
retourner mise à jour structure
retourner mise à jour ensemble
retourner NIL


MISE A JOUR STRUCTURE
facette 'si-maj' existe ?
non
oui
la condition de la facette
'si-maj' est verifiée ?
oui
non
évaluer <action1>
évaluer <action2>
mise à jour globale de la structure ?
non
oui
domaine compatible ?
non
oui
facette 'verifier' existe ?
non
oui
la condition de la facette
'verifier' est verifiée ?
non
oui
affecter NIL a la structure
MISE A JOUR
DONNEE SIMPLE
pour le champ
MISE A JOUR
DONNEE SIMPLE
pour tout les champs
retourner
valeur champ
retourner
valeur structure
retourner NIL


MISE A JOUR ENSEMBLE
facette 'si-maj' existe ?
non
oui
la condition de la facette 'si-maj' est verifiee ?
oui
non
evaluer <action1>
evaluer <action2>
mise a jour globale de l'ensemble ?
oui
domaines compatibles ?
non
non
oui
facette 'verifier' existe ?
non
oui
la condition de la facette 'verifier' est verifiee ?
non
insertion ?
non
oui
suppression ?
oui
non
oui
affecter () a l'ensemble
INSERTION ELEMENT
SUPPRESSION ELEMENT
MISE A JOUR DONNEE SIMPLE
INSERTION ELEMENT pour tous les elements
retourner valeur ensemble
retouner NIL


INSERTION ELEMENT
domaine compatible ?
non
oui
facette 'verifier' existe ?
non
oui
la condition de la facette 'verifier' est verifiée ?
non
oui
facette 'si-ins' existe pour l'ensemble ?
non
oui
la condition de la facette
oui
'si-ins est verifiée ?
non
évaluer <action1>
évaluer <action2>
insérer l'élément
retourner NIL


SUPPRESSION ELEMENT
facette 'si-sup' existe pour l'ensemble ?
non
oui
la condition de la facette
oui
'si-sup est verifiée ?
non
évaluer <action1>
évaluer <action2>
supprimer l'élément
retourner NIL

- ANNEXE 2 - Les opérations de base de SHIVA -

La syntaxe générale d'appel d'une fonction SHIVA ou d'une fonction utilisateur est la suivante :

```
(<nom-fonction>  [ <arg-i> ] )
```

Où <nom-fonction> est le nom de la fonction et <arg-i> sont ses éventuels arguments.

Les arguments d'une fonction sont des expressions c'est à dire :

- des constantes,
- des noms de variables,
- ou des appels de fonctions.

La valeur effective d'un argument d'une fonction sera :

- la constante elle même si l'argument est une constante,
- la valeur de la variable si l'argument est un nom de variable,
- le résultat de l'évaluation de la fonction si l'argument est un appel à une fonction.

Toutes les fonctions de SHIVA retournent une valeur aprés leur évaluation. Ce qui leur permet d'être ainsi utilisées comme argument d'autres fonctions.

Accés aux données

- Affectation d'une valeur à une variable :

```
(affecter <nom-var> <expression>)
```

La fonction affecter donne la valeur de l'argument <expression> à la variable de nom <nom-var>. La valeur de la fonction 'affecter' est la valeur de la variable aprés affectation.

- Obtention de la valeur d'un champ :

```
(de <nom-var> <nom-champ>)
```

La fonction de permet d'obtenir la valeur d'un champ d'une variable composée. L'argument <nom-var> est le nom de la variable et l'argument <nom-champ> est le nom de la composante à accéder. La valeur de la fonction 'de' est la valeur de la composante accédée.

• Afffectation d'une valeur à un champ :

(dans <nom-var> <nom-champ> <expression>)

La fonction dans donne la valeur de l'argument <expression> à la composante de nom <nom-champ> de la variable structuée de nom <nom-var>. La valeur de la fonction 'dans' est la valeur affectée à la composante.

Fonctions sur les ensembles

• Obtention du nombre d'éléments d'un ensemble :

(long-e <expression>)

La fonction long-e retourne pour valeur le nombre d'éléments d'un ensemble. L'argument <expression> est une expression quelconque devant avoir une valeur d'ensemble.

• Insertion d'un élément dans un ensemble :

(ins-e <expression> <nom-var>)

La fonction ins-e permet d'ajouter un nouvel élément à un ensemble. La valeur de l'élément à ajouter, dans la variable de nom <nom-var>, est le résultat de l'évaluation de l'argument <expression>. La valeur de la fonction 'ins-e' est la valeur de l'ensemble aprés ajout. L'ajout d'un élément s'effectue toujours en «tête de liste».

L'argument <nom-var> est le nom d'une variable devant avoir une valeur d'ensemble et l'argument <expression> est une expression quelconque.

• Suppression d'un élément d'un ensemble :

(sup-e <expression> <nom-var>)

La fonction sup-e permet de supprimer la première occurence d'élément d'un ensemble, de nom <nom-var>, dont la valeur est égale au résultat de l'évaluation de l'argument <expression>. La valeur de la fonction 'sup-e' est la valeur de l'ensemble aprés suppression de l'élément.

L'argument <nom-var> doit être le nom d'une variable devant avoir une valeur d'ensemble et l'argument <expression> est une expression quelconque.

• Intersection de deux ensembles :

(int-e <expression-1> <expression-2>)

La fonction int-e permet d'effectuer l'intersection de deux ensembles. La valeur retournée est l'intersection des variables <expression-1> et <expression-2>. Les arguments sont des expressions quelconques devant avoir des valeurs d'ensemble.

• Union de deux ensembles :

(uni-e <expression-1> <expression-2>)

La fonction uni-e permet d'effectuer l'union de deux ensembles. La valeur retournée est l'union des ensembles <expression-1> et <expression-2>. Les arguments sont des expressions quelconques devant avoir des valeurs d'ensembles.

• différence de deux ensembles :

(diff-e <expression-1> <expression-2>)

La fonction diff-e permet d'effectuer la différence de deux ensemble. La valeur retournée est la valeur de la différence : <expression-1> moins <expression-2>. Les arguments sont des expressions quelconques devant avoir des valeurs d'ensemble.

Fonctions sur les chaînes de caractères

• Longueur d'une chaîne de caractères :

(long-c <expression>)

La fonction long-c retourne pour valeur la longueur d'une chaîne de caractères. L'argument est une expression quelconque devant avoir une valeur de chaîne de caractères.

• Concaténation de deux chaînes de caractères :

(cat-c <expression-1> <expression-2>)

La fonction cat-c retourne pour valeur le résultat de la concaténation de deux chaînes de caractères. Les arguments sont des expressions quelconques devant avoir des valeurs de chaînes de caractères.

• Insertion d'une sous-chaîne de caractère :

(ins-c <nom-var> <expression> <position>)

La fonction ins-c permet d'insérer une sous-chaîne dans une chaîne à une position donnée. L'argument <nom-var> est le nom de la chaîne réceptrice. La valeur de l'argument <expression> est, aprés évaluation, la sous-chaîne à insérer. Et la valeur de <position> détermine le rang où devra s'effectuer l'insertion.

• Suppression d'une sous-chaîne de caractères :

(sup-c <nom-var> <position> <nombre>)

La fonction sup-c permet de supprimer un ou plusieurs caractère d'une chaîne de caractères. L'argument <nom-var> est le nom de la variable chaîne de caractères où doit s'effectuer la suppression. La valeur de l'argument <expression> détermine le rang du premier caractère à supprimer. La valeur de l'argument <nombre> détermine le nombre de caractères à supprimer dans la chaîne <nom-var> à partir du rang <position>.

• Extraction d'une sous-chaîne de caractères :

(ext-ch <expression> <position> <nombre>)

La fonction ext-c permet d'extraire un ou plusieurs caractère d'une chaîne de caractères. La valeur de l'argument <expression> est chaîne de caractères dont on veut extraire une sous-chaîne. La valeur de l'argument <expression> détermine le rang du premier caractère d'extraction. La valeur de l'argument <nombre> détermine le nombre de caractère à extraire dans la chaîne <expression> à partir du rang <position>.

Les fonctions arithmétiques

Dans les fonctions suivantes, sauf indication contraire, tous les arguments sont des expressions quelconques devant avoir des valeurs de nombres entiers ou réels.

• Addition de deux nombres :

(plus <expression1> <expression2>)

La fonction plus retourne pour valeur le résultat de la somme des arguments <expression1> et <expression2>.

• Soustraction de deux nombres :

(moins <expression1> <expression2>)

La fonction moins retourne pour valeur le résultat de la soustraction : <expression1> moins <expression2>.

• Multiplication de deux nombres :

(mult <expression1> <expression2>)

La fonction mult retourne pour valeur le résultat du produit des arguments <expression1> et <expression2>.

• Division de deux nombres :

(div <expression1> <expression2>)

La fonction div retourne pour valeur le résultat de la division entière de <expression1> par <expression2>.

• Reste d'une division entière :

(mod <expression1> <expression2>)

La fonction mod retourne pour valeur le résultat du reste de la division entière de <expression1> par <expression2>.

• Conversion en entièr :

(entier <expression>)

La fonction entier retourne pour valeur le résultat de la conversion en un nombre entier de l'argument <expression>.

• Conversion en réel :

(reel <expression>)

La fonction reel retourne pour valeur le résultat de la conversion en un nombre reel de l'argument <expression>.

• Somme des éléments d'un ensemble :

(som-e <expression>)

La fonction som-e retourne pour valeur la somme arithmétique de tous les éléments d'un ensemble dont les éléments sont définis sur les domaines des nombres entiers ou réels. La valeur de l'argument <expression> est l'ensemble dont on veut calculer la somme des éléments. Si la valeur de <expression> est l'ensemble vide, la valeur 0 est retournée.

• Elément minimum d'un ensemble :

(min-e <expression>)

La fonction min-e retourne la valeur du plus petit élément de tous les éléments d'un ensemble dont les éléments sont définis sur les domaines des nombres entiers ou réels. La valeur de l'argument <expression> est l'ensemble dont on veut calculer le minimum. Si la valeur de <expression> est l'ensemble vide, la valeur NIL est retournée.

• Elément maximum d'un ensemble :

(max-e <expression>)

La fonction max-e retourne la valeur du plus grand élément (de tous les éléments) d'un ensemble dont les éléments sont définis sur les domaines des nombres entiers ou réels. La valeur de l'argument <expression> est l'ensemble dont on veut calculer le maximum. Si la valeur de <expression> est l'ensemble vide, la valeur NIL est retournée.

• Moyenne des éléments d'un ensemble :

(moy-e <expression>)

La fonction moy retourne la valeur de la moyenne arithmétique (de tous les éléments) d'un ensemble dont les éléments sont définis sur les domaines des nombres entiers ou réels. La valeur de l'argument <expression> est l'ensemble dont on veut calculer la moyenne. Si la valeur de <expression> est la liste vide, la valeur NIL est retournée.

Les fonctions logiques

• La conjonction :

(et <expr-1> <expr-2>... <expr-N>)

La fonction et retourne la valeur NIL si au moins une des expressions <expr-i> est la valeur NIL. Sinon, la valeur de <expr-N> est retournée.

• La disjonction :

(ou <expr-1> <expr-2>... <expr-N>)

La fonction ou retourne la valeur NIL si toutes les expressions <expr-i> ont la valeur NIL. Sinon la valeur de la première expression <expr-i> est retournée.

• La négation :

(non <expression>)

La fonction non retourne la valeur NIL si l'argument <expression> n'a pas la valeur NIL. Sinon la valeur 0 est retournée.

• Test de la liste vide :

(vide <expression>)

La fonction vide retourne la valeur NIL si la valeur de l'argument <expression> n'est la liste vide : (). Sinon la valeur 0 est retournée.

• L'égalité :

(egal <expression1> <expression2>)

La fonction egal retourne la valeur NIL si la valeur de <expression1> n'est pas égale à la valeur de <expression2>. Sinon la valeur de <expression1> est retournée.

• Test si zéro :

(zero <expression>)

• La fonction zero retourne la valeur NIL si la valeur de l'argument n'est pas un nombre entier ou réel égal à zéro. Sinon la valeur de <expression> est retournée.

• Test si positif ou nul :

(positif <expression>)

La fonction positif retourne la valeur NIL si la valeur de l'argument n'est pas un nombre entier ou réel supérieur ou égal à zéro. Sinon la valeur de <expression> est retournée.

• Test si strictement positif :

(positif+ <expression>)

La fonction positif+ retourne la valeur NIL si la valeur de l'argument n'est pas un nombre entier ou réel strictement supérieur à zéro. Sinon la valeur de <expression> est retournée.

• test si négatif ou nul :

(negatif <expression>)

La fonction negatif retourne la valeur NIL si la valeur de l'argument n'est pas un nombre entier ou réel inférieur ou égal à zéro. Sinon la valeur de <expression> est retournée.

• Test si strictement négatif :

(negatif- <expression>)

La fonction negatif retourne la valeur NIL si la valeur de l'argument n'est pas un nombre entier ou réel strictement inférieur à zéro. Sinon la valeur de <expression> est retournée.

• Test si pair :

(pair <expression>)

La fonction pair retourne la valeur NIL si la valeur de l'argument n'est pas un nombre pair. Sinon la valeur de <expression> est retournée.

• Test si impair :

(impair <expression>)

La fonction impair retourne la valeur NIL si la valeur de l'argument n'est pas un nombre impair. Sinon la valeur de <expression> est retournée.

• Test si strictement supérieur :

(sup <expression1> <expression2>)

La fonction sup retourne la valeur de <expression1> si les arguments sont des nombres entiers ou réels et si la valeur de <expression1> est strictement supérieure à la valeur de <expression2>. Sinon la valeur NIL est retournée.

• Test si strictement inférieur :

(inf <expression1> <expression2>)

La fonction inf retourne la valeur de <expression1> si la valeur des arguments sont des nombres entiers ou réels et si la valeur de <expression1> est strictement inférieure à la valeur de <expression2>. Sinon la valeur NIL est retournée.

• Test si supérieur ou égal :

(sup= <expression1> <expression2>)

La fonction sup= retourne la valeur de <expression1> si la valeur des arguments sont nombres entiers ou réels et si la valeur de <expression1> est supérieure ou égale à la valeur de <expression2>. Sinon la valeur NIL est retournée.

• Test si inférieur ou égal :

(inf= <expression1> <expression2>)

La fonction inf= retourne la valeur de <expression1> si la valeur des arguments sont nombres entiers ou réels et si la valeur de <expression1> est inférieure ou égale à la valeur de <expression2>. Sinon la valeur NIL est retournée.

• Test si chaîne plus petite :

(alpha- <expression1> <expression2>)

La fonction alpha- retourne la valeur de <expression1> si les arguments sont des chaînes de caractères et si la valeur <expression1> est inférieure ou égale lexicographiquement à la valeur de <expression2>. Sinon la valeur NIL est retournée.

• Test si sous-chaîne :

(cont-c <expression1> <expression2>)

La fonction cont-c retourne la valeur NIL si les arguments sont des chaînes de caractères et si <expression2> n'est pas une sous-chaîne de <expression1>. Sinon la valeur retournée est le rang dans la chaîne <expression1> où commence la sous-chaîne <expression2>.

• L'inclusion :

(inclus <expression1> <expression2>)

• La fonction inclus retourne la valeur de <expression1> si <expression2> est un ensemble et <expression1> une valeur quelconque telle que <expression1> est un élément de <expression2>. Sinon la valeur NIL est retournée.

La fonction conditionnelle SI

(si <expression1> <expression2> [<expression3>])

La fonction si retourne pour valeur le résultat de l'évaluation de <expression2> si la valeur de l'argument <expression1> n'est pas la valeur NIL. Sinon la valeur retournée est le résultat de l'évaluation de <expression3>.

Si l'argument <expression3> est absent, la valeur NIL est prise par défaut.

La fonction séquentielle FAIRE

```
(faire <expression1> <expression2> ... <expressionN>)
```

La fonction faire évalue séquentiellement tous ses arguments et retourne pour valeur, la valeur de la dernière expression évaluée.

La fonction itérative POUR

```
(pour <nom-var1> ( <nom-var2> [ <nom-var3> ] )
                 <expression>
                              [ <nom-iter> ] )
```

La fonction pour parcourt l'ensemble <nom-var1> et pour chaque élément parcouru, l'argument <expression> est évalué.

L'argument <nom-var2> est le nom d'une variable temporaire qui reçoit successivement les valeurs des différents éléments parcourus (cette variable est appelée «variable de parcours».

L'argument <nom-var3> est le nom d'une variable temporaire qui, si elle est présente, reçoit pour valeur le sous-ensemble des éléments restant à parcourir et ce à chaque pas de l'itération (cette variable est appellée «variable-suite»).

L'argument <nom-iter> est une expression optionnelle, dont la valeur doit être une chaîne de caractères et qui permet de «nommer» l'itération courante. Le rôle de cet argument sera précisé lors de la présentation de la fonction 'arret'.

Sauf indication contraire, la valeur retournée par la fonction 'pour' est toujours la valeur du dernier élément parcouru.

Si la variable de nom <nom-var1> n'a pas une valeur d'ensemble ou si l'ensemble est vide, la valeur NIL est retournée par la fonction 'pour' et l'argument <expression> n'est pas évalué.

Lors de la première évaluation de l'argument <expression> une variable de parcours, de nom <nom-var2>, est créée sur le domaine des éléments de l'ensemble <nom-var1> et est initialisée avec la valeur du premier élément de l'ensemble. Si une variable-suite est présente, une variable de nom <nom-var3> sera cré('ee sur le domaine de la variable <nom-var1> et sera initialisée avec l'ensemble de tous les éléments de la variable <nom-var1> sauf le premier.

A chaque pás de l'itération, la variable de parcours prend la valeur du premier élément du sous ensemble des éléments restants à parcourir et la variable-suite , si elle est présente prend pour valeur ce sous-ensemble diminué du premier élément.

A la fin de l'itération, la variable de parcours et, éventuellement la variable suite, seront détruites et ne seront plus accessibles.

Pendant toute la durée de l'itération, la variable de parcours ainsi que la variable-suite peuvent être modifiées. Dans ce cas, les modifications portent directement sur l'élément et le sous-ensemble courant de l'ensemble sur lequel le parcours s'effectue.

Exemple 1 :

```
(defdom UN-ENSEMBLE (ens entier))

(defvar ENS UN-ENSEMBLE (1 2 3 4))
--> (1 2 3 4)

(pour ENS (E) (affecter E (plus E 1)))
--> 5
```

La fonction 'pour' permet d'ajouter la valeur 1 à tous les éléments de l'ensemble ENS. La valeur de cet ensemble, aprés itération est :

(2 3 4 5)
et la valeur de la fonction 'pour' est la valeur 5 qui est la valeur du dernier élément de l'ensemble aprés mofication.

Exemple 2 :

```
(defvar E1 UN-ENSEMBLE (1 3 3 2 3 5 1))
--> (1 3 3 2 3 5 1)

(defvar E2 UN-ENSEMBLE () )
--> ()

(pour E1 (COUR SUITE)
    (si (inclus COUR SUITE) (sup-e SUITE COUR)))
--> 5
```

Cette expression permet de supprimer tous les éléments dupliqués de l'ensemble E1.

Détail de l'itération :

valeur de COUR	valeur de SUITE	valeur de E1
1	(3 3 2 3 5 1)	(1 3 3 2 3 5 1)
	(3 3 2 3 5)	(1 3 3 2 3 5)
3	(3 2 3 5)	(1 3 3 2 3 5)
	(2 5)	(1 3 2 5)
2	(5)	(1 3 2 5)
5	()	(1 3 2 5)

La rupture d'itération ARRET

(arret [nom-iter])

La fonction arret ne peut être utilisée que dans une expression associée à une fonction 'pour' et a pour but de provoquer l'arrêt de l'itération. Dans ce cas, la valeur de la fonction 'pour' sera la dernière valeur affectée à la variable de parcours.

L'argument <nom-iter> est une expression optionnelle qui, si elle existe, doit avoir pour valeur le nom donné à une fonction 'pour' en cours d'évaluation. Cet argument permet de rompre l'évaluation de nom <nom-iter> ainsi que de toutes les itérations qui y seraient imbriquées.

Si l'argument <nom-iter> est absent, c'est la dernière fonction 'pour' en cours d'évaluation, pour laquelle aucun nom n'a été donné qui sera arrêtée.

- ANNEXE 3 - Présentation de SHIMER -

SHIMER est un système relationnel, écrit entièrement en langage Lisp, qui offre aux utilisateurs de ce langage une "vision" relationnelle de leurs objets.

Le couplage Système Relationnel - Langage Lisp fait que SHIMER combine à la fois les fonctionnalités des systèmes relationnels :

- structures de données simples et faciles à utiliser,
- indépendance logique et physique des données,
- langage de manipulation de données de haut niveau;

et les immenses potentialités du langage Lisp :

- langage symbolique et fonctionnel,
- possibilité de programmation dirigée par les données,
- technique de filtrage,
- unicité des objets (les données et les fonctions sont de même nature).

Ce qui permet à Shimer d'offrir les services suivants :

- insertion, suppression et mise à jour de données,
- recherche de données satisfaisant à un critère sélectif,
- calculs arithmétiques et logiques,
- enchaînement de séquences d'actions,
- affectation des résultats pour impression ou valeurs de nouvelles relations.

Et du fait de sa programmation en langage Lisp, ces services s'utilisent et se comprennent comme tout objet de ce langage.

L'intérêt essentiel de SHIMER est qu'il offre, à l'intérieur de programmes Lisp, une méthode de stockage et de manipulation de données fiable et agréable conforme à la "vision" symbolique de ce langage. Grâce à l'apport relationnel, les objets Lisp (symboles, atomes, listes ou fonctions) ne seront plus stockés sous formes de fichiers mais sous formes de n-uplets conformément au modèle relationnel defini par CODD.

Un ensemble de fonctions Lisp permet la création, la consultation, la mise à jour et la suppression de relations, d'attributs et de n-uplets. A cet ensemble de fonctions, qui forment les primitives de base de SHIMER, est adjoint un ensemble de fonctions de plus haut niveau : les opérateurs de l'algèbre relationnelle.

SHIMER offre à l'utilisateur, sous forme de fonctions Lisp, les principaux opérateurs de l'algèbre relationnelle : la différence, le produit, le produit cartésien, la sélection la projection, l'antiprojection, la division, l'intersection, l'union et la jointure.

Pour l'accès aux données, deux méthodes sont actuellement proposées : le balayage séquentiel des n-uplets des relations et l'indexation (accés sur une clé).

Un avantage remarquable, apporté par l'utilisation des requêtes SHIMER sous forme de fonctions Lisp, est l'évaluation d'arborescences d'opérateurs relationnels directement par l'interpréteur Lisp.

Considérons, par exemple, les relations :

TRAIN (NO-TRAIN, NO-LIGNE, JOUR)

WAGON (NO-WAGON,TYPE-WAGON,POIDS,...)

La question "Donner les types des wagons du train 4002" dont l'expression en langage algébrique est :

R <--(((TRAIN : E)[NO-WAGON])*WAGON)[TYPE-WAGON]

(Si E désigne l'expression NO-TRAIN=4002)

peut s'exprimer, à l'aide des fonctions de Shimer, de la manière suivante :

```
(SELECT 'train '(EGAL no-train 4002))
--> tempo1

(PROJECT 'tempo1 '(no-wagon))
--> tempo2

(PRODUIT 'wagon 'tempo2)
--> tempo3

(PROJECT 'tempo3 '(type-wagon))
--> tempo4

(AFFECT 'r 'tempo4)
--> r
```

Où AFFECT, PROJECT, PRODUIT et SELECT sont les noms des fonctions SHIMER réalisant les opérateurs de l'algèbre relationnelle.

Mais aussi en une seule expression Lisp :

```
(AFFECT 'r
        (PROJECT
            (PRODUIT 'wagon
                     (PROJECT
                         (SELECT 'train
                                 '(egal no-train 4002))
                         '(no-wagon)))
            '(type-wagon)))
--> r
```

(Les lignes précédées d'une fêche (-->) sont les réponses de l'interpréteur Lisp au requêtes SHIMER).

Par cet exemple, on voit qu'il n'est nul besoin de gérer, stocker et d'interpréter des arborescences d'opérateurs algébriques; il suffit simplement d'énoncer la requête sous la forme d'une fonction Lisp.

La possibilité de créer des relations dérivées, d'effectuer des traitements arithmétiques ou logiques sur les données d'une relation et la notion de filtrage d'index sont d'autres avantages obtenus grâce au couplage Système Relationnel - Langage Lisp.

• Comme exemple de traitement arithmétique, considérons la relation WAGON-4002 qui décrit l'ensemble des wagons attaché au train 4002 :

WAGON-4002 (N-WAGON, POIDS-VIDE, POIDS-MARCHANDISE)

Pour obtenir le poids total en charge du train 4002 on formulera la requête SHIMER suivante :

```
(som-r 'WAGON-4002
       '(som-n 'POIDS-VIDE POIDS-MARCHANDISE)
        t)
```

• Comme exemple de relation dérivée, considérons la relation :

WAGON-A (N-WAGON, ETAT, POIDS-VIDE, POIDS-MARCHANDISE)

La requête SHIMER suivante :

```
(calcul-r 'WAGON-N        'WAGON-A

          '((N-WAGON    N-WAGON)
            (POIDS-TC   (som-n POIDS-VIDE POIDS-MARCHANDISE)))

          '(egal TYPE "charge")))
```

permet d'obtenir la relation dérivée :

WAGON-N (N-WAGON , POIDS-TC)

obtenue à partir des n-uplets de la relation WAGON-A qui vérifient la condition :

"type-wagon = charge"

et dont les valeurs des attributs N-WAGON et POIDS-TC sont obtenues, pour chaque n-uplet de la relation WAGON-N, de la manière suivante :

pour chaque n-uplet de la relation WAGON-A qui vérifie la condition énoncée, on a :

N-WAGON de WAGON-N = N-WAGON de WAGON-A

et

POIDS-TC de WAGON-N = POIDS-VIDE de WAGON-A
+ POIDS-MARCHANDISE de WAGON-A

• Comme exemple de filtrage considérons la relation TRAFIC :

TRAFIC (N-TRAIN ,JOUR, N-LIGNE)

telle que l'attribut JOUR est une liste de nombres entiers de la forme : (jour mois année); et les attributs LIGNE et N-TRAIN sont des nombres entiers.

Pour obtenir l'ensemble des trains qui circuleront sur la ligne numéro 12 le mois de mars 1980 et ce, à partir du 10 mars on formulera les requêtes SHIMER suivantes :

```
[1]     (creer-i 'TRAFIC 'ACCES 'prim '(N-LIGNE JOUR))

[2]     (sel-f          'TRAFIC 'ACCES
             '((egal 12)
               (telque (sup= 10) (egal 03) (egal 1980))))
```

La requête [1] permet de créer sur la relation TRAFIC un index secondaire de nom ACCES et dont la est composée des attributs N-LIGNE et JOUR;

La requête [2] permet de sélectionner dans la relation TRAFIC, en utilisant l'index ACCES, l'ensenble des wagons qui vérifient le filtre:

((egal 12) (telque (sup= 10) (egal 03) (egal 1980)))

Dans ce filtre, l'élément (egal 12) porte sur l'attribut N-LIGNE de la
de l'index,

et l'élément (telque (sup= 10) (egal 03) (egal 1980))
porte sur l'attribut JOUR de la .

- B I B L I O G R A P H I E -

[Adi82] "Bases de données et systèmes relationnels";
C. Delobel, M. Adiba; DUNNOD informatique; 1982.

[Aik83] "Prototypical knowledge in expert systems";
J.S. Aikins; Artificial Intelligence 20, pp 163-210; 1983.

[BD383] "Bases de données, nouvelles perspectives";
Groupe BD3; groupe BD3, INRIA; janvier 1983.

[Ban85] "Règles récursives dans les bases de données déductives";
F. Bancilhon; Journées Bases de Données Avancées, St. Pierre de Chartreuse; 6-8 mars 1985.

[Ben84] "SHIMER : un système relationnel en Lisp";
A. Bensaid; RR n. 464 ARTEMIS/IMAG; aout 1984.

[Bid84] "Un modèle de base de données relationnel non normalisé: algèbre et interprétation";
N. Bidoit; Thèse de 3ème cycle, université de Paris Sud, centre d'Orsay; juin 1984.

[Bon84] "L'intelligence artificielle, Promesses et réalités";
A. Bonnet; InterEditions, Paris; 1984.

[Bri83] "L'instanciation, dans les langages objets";
JP Briot.; BIGRE journées d'étude sur les langages orientés objets (pp 173,209); decembre 1983.

[Cha84] "Le_Lisp : le manuel de référence, version 14";
J. Chailloux; Documentation du système Le_Lisp, INRIA; mai 1984.

[Chr85] "Modèle agrégatif, langage de manipulation et interface multi-media";
G. Zurfluh, C. Chrisment; Journées Bases de Données Avancées, St. Pierre de Chartreuse; 6-8 mars 1985.

[Cod70] "A relational model for large shared bank";
E.F. Codd; Comm. ACM 13,6; 1970.

[Cod79] "Extending the database relational model to capture more meaning";
E.F. Codd; ACM trans. on database systems. 4,4; 1979.

[Coi83] "Une réalisation de SMALLTALK en VLISP";
P. Cointe; TSI Techniques et Sciences Informatiques, Vol 2, n=4; 1983.

[Dem85] "Une extension du modèle relationnel pour représenter et manipuler des objets structurés";
R. Demolombe; Journées Bases de Données Avancées, St. Pierre de Chartreuse; 6-8 mars 1985.

[Ers84] "A relational expert system for nursing management control";
C.J. Ersnt; North-Holland, Human Systems Management 4 (1984) p286-293; 1984.

[Gar84] "Bases de données. Les systèmes et leurs langages";
G. Gardarin; Editions EYROLLES; 1984.

[Goe83] "Data bases and knowlegde representation for literary and languistics studies";
N. Cercone, R. Goebel; Computers and hum. North Holland, 17. pp121-137; 1983.

[Gou84] "EDORA : un système intelligent d'aide à la modelisation en biologie";
P. Vignard, JL. Gouze; International 84 AMSE Conference Modelling and Simulation, Athen Greece; June 27-29 1984.

[Gra85] "Mises à jour des bases de données contenant de l'information incomplète";
S. Abiteboul, G. Grahme; Journées Bases de Données Avancées, St. Pierre de Chartreuse; 6-8 mars 1985.

[Gra82] "Evaluation des performances du langage PLASMA en intelligence artificielle";
C. Granger; Rapport ENSEEIHT; 1982.

[Gra84] "SHIRKA : des systèmes experts centrés objets";
F. Rechenmann, A. Bensaid, D. Granier; Les systèmes experts et leurs applications. Journées d'études et exposition, Avignon 1984; 2,3 et 4 mai 1984.

[Gut83] "Documentation processing in a relational database système";
M. Stonebraker, H. Stettner, N. Lynn, J. Kalash, A. Guttman; ACM trans. on office Inf. systems, vol 1,2, pp143-148; 1983.

[Hor81] "LISP";
PH. Winston, BKP Horn; Addison Wesley Publishing Company; 1981.

[Hul84] "CEYX, Version 4, Le manuel de référence";
JM Hullot; Note technique INRIA, (33 pages); janvier 1984.

[Kin77] "An overview of production systems";
R. Davis, J. King; Dans "Machine Intelligence 8", E.W Elcock et D. Michie Eds, Halsted Press; 1977.

[Lau82] "Removing restrictions in the relationnal database model : an application of problem solving techniques";
L. Siklossy J.L. Lauriere; AAAI. Pittsburg P310-313; 1982.

[Leo85] "A propos de systèmes de gestion de bases de données intégrant la conception";
M. Leonard; Journées Bases de Données Avancées, St. Pierre de Chartreuse; 6-8 mars 1985.

[Lip85] "une approche formelle aux valeurs nulles non-applicables";
N. Lerat, W. Lipski; Journées Bases de Données Avancées, St. Pierre de Chartreuse; 6-8 mars 1985.

[Mac79] "Sequel as a langage for document retrieval";
I.A. MacLeod; Journal of the American society for information science, Vol 30; 1979.

[Mad85] "Déduction dans la Machine Bases de Données SABRE";
J. Madelaine; Journées Bases de Données Avancées, St. Pierre de Chartreuse; 6-8 mars 1985.

[Mai85] "OFE : Un langage fonctionnel de manipulation de bases de données";
J. Le Maitre; Journées Bases de Données Avancées, St. Pierre de Chartreuse; 6-8 mars 1985.

[Mak77] "A consideration on normal form of not-necessarily-normalized relation in the relational data model";
A. Makinouchi; Proc. Inter. Conf. on VLDB, Tokyo; 1977.

[Met85] "SECSI : Un outil interactif de modélisation conceptuelle de bases de données";
M. Bouzeghoub, G. Gardarin, E. Metais; Journées Bases de Données Avancées, St. Pierre de Chartreuse; 6-8 mars 1985.

[Min75] "A framework for representing knowledge";
M. Minsky; Ph Winston Ed. McGraw-Hill; 1975.

[Oli85] "L'intégrité sémantique dans une Base de Données génralisées";
G.T. Nguyen, J. Olivares; Journées Bases de Données Avancées, St. Pierre de Chartreuse; 6-8 mars 1985.

[Oli84] "Le modèle de données et sa représentation dans un système de base de données généralisée";
J. Palazzo Oliveira; Thèse de 3ème cycle INPG; juin 1984.

[Pal85] "Notion de temps dans les bases de données généralisées";
M. ADIBA, Q.N. Bui, J. Palazzo; Journées Bases de Données Avancées, St. Pierre de Chartreuse; 6-8 mars 1985.

[Pin81] "Représentation des connaissances dans les systèmes experts";
S. Pinson; RAIRO Informatique Vol 15, n=4, pp. 343-367; 1981.

[Rec84] "Intelligence artificielle et construction de modèles dynamiques";
F. Rechenmann; Intelligence Artificielle et Productique 1984, Paris; 20-22 novembre 1984.

[Rie85] "Modèle et fonctionnalités d'un SGBD pour les applications CAO";
D. Rieu-Rialhe; Journées Bases de Données Avancées, St. Pierre de Chartreuse; 6-8 mars 1985.

[Rob80] "Using Frame in scheduling";
I. Goldstein, B. Robert; Artificial Intelligence an MIT perspective, vol 1, Winston and Brown editors, MIT press; 1980.

[Rob83] "SMALLTALK 80. The language and its implementation";
A. Goldberg, D. Robson; Addison Wesley Puplishing Company; 1983.

[Rou85] "Intégrité des Bases de Données logiques";
M. De Rougemont; Journées Bases de Données Avancées, St. Pierre de Chartreuse; 6-8 mars 1985.

[Sch82] "Remarks on the algebra of non first normal form relations";
G. Jaeschke, H.J. Schek; Proc. SIGACT-SIGMOD, Los Angeles; 1982.

[Sch84] "An algebra for the relational model with Relation-Valued attributes";
H.J. Schek, M.H. Scholl; Technical Report, DVSI-1984-T1, Technische Hochschule Darmstadtt, Fachbereich Informatik; 1984.

[Ser82] "Le langage FORMES, analyse et implémentation";
JP. Briot, BP. Serpette; Mémoire de stage de DEA, Paris VI; juillet 1982.

[Sim85] "SABRINA : Une évolution du SGBD SABRE vers un système intelligent et actif";
G. Gardarin, M. Bouzeghoub, B. Kerherve, F. Pasquier, E. Simon; Journées Bases de Données Avancées, St. Pierre de Chartreuse; 6-8 mars 1985.

[Ste82] "The LOOPS Manual";
DG. Bobrow, M. Stefik; Knowledge-Based VLSI Design Group, Memo KB-VLSI-82-22 (working paper) Xerox Corporation; 1982.

[Sto85] "Bases de données et systèmes experts";
M. Stonebraker; Journées Bases de Données Avancées, St. Pierre de Chartreuse; 6-8 mars 1985.

[Val85] "La compilation de contraintes d'intégrité exprimées dans un langage de haut niveau";
E. Simon, P. Valduriez; Journées Bases de Données Avancées, St. Pierre de Chartreuse; 6-8 mars 1985.

[Vas84] "An Optimizing Prolog Front-End to a Relational Query System";
M. Jarke, J. Clifford, Y. Vassiliou; ACM 1984 p296-306; 1984.

[Vel85] "La prise en compte de documents structurés dans un SGBD : Aspects modèle, langage et architecture du système";
F. Velez; Journées Bases de Données Avancées, St. Pierre de Chartreuse; 6-8 mars 1985.

[Vig84] "EDORA : an artificial intelligence approach to dynamic system modelling";
JL. Gouze, F. Rechenmann, P. Vignard; The management and modelling of dynamic systems, Bruges, Belgium; June 12-14 1984.

[Wei80] "Flavors : Messages passing in the Lisp Machine";
D. Moon, D. Weinreb; AI Memo n 602 (Working Paper), Artificial Intelligence Laboratory, MIT; November 1980.

[Win77] "An overwiew of KRL, a Knowlegde Representation Language";
D.G. Bobrow, T. Winograd; Cognitive Science, Voln=1; 1977.

[Yaz85] "Modèle complet, modèle irredondant pour un schéma de base de données relationnelle";
R. Demolombe, K. Yazdanian; Journées Bases de Données Avancées, St. Pierre de Chartreuse; 6-8 mars 1985.

[Ria83] "Bases de données et nouvelles applications";
M. Adiba, M. Burnier, C. Delobel, D. Rialhe; RR IMAG N.349; Fevrier 1983.

[Kui75] "A frame for frame : representation knowledge for recognition";
B.J. Kuipers; Representating and understanding, BG. Bobrow, A Collins (ed) Academic Press; 1975.

[Che76] "The entity-relationship model-toward a unified view of data ;
P.P. Chen; ACM Trans. on Data base syst. Vol 1.1 p9-36; March 1976.

AUTORISATION de SOUTENANCE

VU les dispositions de l'article 3 de l'arrêté du 16 avril 1974

VU les rapports de présentation de Messieurs
. J. MOSSIERE, Professeur
. M. SCHOLL, Ingénieur de recherche

Monsieur BENSAID Ali

est autorisé à présenter une thèse en soutenance en vue de l'obtention du diplôme de DOCTEUR-INGENIEUR, spécialité "Informatique".

Fait à Grenoble, le 24 avril 1985

Le Président de l'I.N.P.-G

D. BLOCH
Président
de l'Institut National Polytechnique
de Grenoble

P.O. le Vice-Président,