UNIVERSITÉ DE NANCY I
U.E.R. DE SCIENCES MATHÉMATIQUES

CENTRE DE RECHERCHE EN
INFORMATIQUE DE NANCY

UN LANGAGE DE SPÉCIFICATION DES SYSTÈMES D'INFORMATION

UN OUTIL POUR LEUR GESTION

THÈSE

Présentée pour l'obtention du

DOCTORAT D'INGÉNIEUR EN INFORMATIQUE

soutenue le 5 juin 1980

par

Sergio LEIFERT

Membres du Jury :

Président : J. LEGRAS
Mme : C. ROLLAND
MM. : J.C. DERNIAME
R. DEMOLOMBE
V.W. SETZER
J. MAROLDT

UNIVERSITÉ DE NANCY I
U.E.R. DE SCIENCES MATHÉMATIQUES

CENTRE DE RECHERCHE EN
INFORMATIQUE DE NANCY

UN LANGAGE DE SPÉCIFICATION DES SYSTÈMES D'INFORMATION

UN OUTIL POUR LEUR GESTION

THÈSE

Présentée pour l'obtention du

DOCTORAT D'INGÉNIEUR EN INFORMATIQUE

soutenue le 5 juin 1980

par

Sergio LEIFERT

Membres du Jury :

Président : J. LEGRAS
Mme : C. ROLLAND
MM. : J.C. DERNIAME
R. DEMOLOMBE
V.W. SETZER
J. MAROLDT

A Simone

et nos enfants

P L A N

AVANT PROPOS

Je voudrais par ces quelques mots témoigner ma profonde reconnaissance à tous ceux qui savent que ce travail n'aurait pas pu être réalisé sans le soutien, l'amitié, le conseil, la patience et la chaleur humaine qu'ils m'ont accordé.

A mes familles par la compréhension et l'appui qu'elles m'ont donné pendant mon séjour en France.

Aux amis que j'ai eu la chance de trouver.

A toute l'équipe REMORA qui m'a si bien accueilli, supporté et aidé dans la réalisation de ce travail et dans ma réalisation humaine.

Au CROUS et en particulier au service des étudiants étrangers par leur disponibilité et appui à tous les instants.

Aux membres du jury qui acceptent aujourd'hui de juger ce travail après m'avoir aidé par les moyens les plus différents à le réaliser.

A Colette ROLLAND, responsable pour mon travail et qui est à l'origine de la plupart des idées qui y sont developpées. Je veux aussi exprimer ma plus sincère gratitude pour tout le soutien et l'amitié qu'elle et sa famille nous ont donné et pour tout ce qu'elle m'a appris.

INTRODUCTION

I. L'IDEE DU PROCESSEUR DE LA DYNAMIQUE

I.1. LES SYSTÈMES D'INFORMATION DES ORGANISATIONS

L'analyse des systèmes d'information des organisations nous conduit à constater que (ROL 80) (ROL 79b) :

α. Le système d'information est un ensemble de représentations codées

Les gestionnaires ont besoin d'être renseignés en permanence sur l'état de l'organisation qu'ils gèrent. Ces renseignements peuvent être exigés à différents niveaux de détail et concernent des aspects variés de l'organisation et de son fonctionnement.

Afin de satisfaire un tel besoin et en tenant compte d'une part de l'impossibilité d'être partout présents et de tout retenir et d'autre part du besoin de disposer d'une vue commune et cohérente de l'organisation, les gestionnaires font une représentation de la réalité qui puisse les renseigner en permanence.

Le code associé à cette représentation doit permettre à tout gestionnaire de retrouver les informations qu'il demande.

Le système d'information représente les composants de la réalité en les identifiant par classes et en associant des propriétés aux classes pour les distinguer. Les différents composants de la réalité sont donc distingués dans une même classe par les différentes valeurs de leurs propriétés.

L'organisation est donc perçue, à tout instant, par l'ensemble de représentations codées qui reflète son état.

β Les représentations doivent évoluer au cours du temps

Le système d'information doit renseigner les gestionnaires sur l'état actuel, sur l'état passé et dans certains cas, sur l'état futur de l'organisation.

Il faut donc qu'il puisse renseigner les gestionnaires sur l'évolution de l'organisation au cours du temps.

Cette évolution est traduite par la réaction de l'organisation aux différents évènements qui y surviennent et qui induisent des changements.

Par exemple, l'arrivée d'une ligne de commande, la fin de la semaine, l'échéance d'une dette en sont des stimuli pour l'organisation qui réagit en activant des actions.

De même que l'organisation évolue le système d'information doit évoluer à son image de façon à ce que l'ensemble de représentations qu'il contient soient en permanence pertinentes.

I.2. LES SYSTÈMES D'INFORMATIONS AUTOMATISÉS

L'informatique a apporté des moyens pour stocker les représentations, pour les restituer et en produire de nouvelles.

Les systèmes d'information automatisés, comportent pour cette raison, des ensembles de données (les représentations) et des lots de programmes (pour restituer et produire les images).

Plusieurs travaux ont été développés dans le domaine de la conception de systèmes d'information automatisés (LUG 75)(TEY 73)(PEC 75)(WATT75) (COU 73)(DOY 76) (ROL 76)(ROL 78)(ROL 79a).

On retrouve dans la plupart d'eux l'application du précepte α.

Il impose que l'on réfléchisse à quelles représentations il faut mettre dans le SI avant de savoir comment le faire.

Pour qu'il puisse satisfaire le but pour lequel il est conçu, un système d'information doit assurer que l'ensemble de représentations de la réalité qu'il comprend soit, à tout instant, une image : fidèle (que traduit la réalité sans distorsion),
complète (tout utilisateur peut retrouver les représentations qu'il veut),
consistente (sans ambiguité et sans contradiction),
de la réalité.

Les travaux sur les bases de données vont dans ce sens. Ils ont introduit l'idée que pour disposer dans une base de données de "bonnes" images, il fallait identifier le problème de conception de la base à celui de la représentation des situations réelles (ou réel perçu (IFI 74, IFI 76, ANS 77, DAT 77).

On cherche à savoir définir correctement l'ensemble des représentations

Mais cela est insuffisant.

Le précepte β impose que l'on réfléchisse aux règles qui font évoluer les représentations et les conditions dans lesquelles elles s'appliquent et que l'on en ait aussi des représentations complètes, cohérentes, fidèles.

Nous ne considérons dans les systèmes d'information que l'évolution prévisible de la réalité, c'est-à-dire que l'évolution déterministe de l'organisation liée à son fonctionnement (sa cinématique).

En conséquence si on veut que l'ensemble des représentations de la réalité contenu dans un système d'information traduisent l'état réel en permanence, il faut que l'évolution de cet ensemble soit conforme à l'évolution de la réalité.

Il est bien clair qu'il existe des programmes autour des bases de données. Ces programmes remplissent le rôle précédent mais c'est d'une part

implicite et d'autre part confondu avec d'autres rôles notamment celui de la production d'images sous forme de messages.

Nous pensons qu'il est utile de l'expliciter

Si on le fait on est alors capable d'aller plus loin dans la gestion de l'évolution du SI puisqu'on est capable de reconnaître le type d'évènement concret déterminer la règle de gestion qui correspond à la réaction de l'organisation à cet évènement. Cette action peut être menée par un outil. C'est l'hypothèse que nous faisons.

Nous appelons cet outil le processeur de la dynamique.

Le processeur doit satisfaire les fonctions suivantes :

- reconnaître qu'un évènement est survenu dans la réalité
- identifier les règles d'évolution qui traduisent la réaction de la réalité à cet événement
- provoquer sur le SI l'exécution de ces règles et le faire évoluer conformément à la réalité.

2. SPECIFICATION DU PROCESSEUR DE LA DYNAMIQUE

Le processeur de la dynamique est un logiciel qui met en oeuvre des mécanismes opératoires pour analyser les changements d'état survenus dans le SI (image des changements d'état réels), décider des actions conséquentes (image des actions de gestion que l'organisation exécute en réponse aux événements venus de l'extérieur ou produits dans l'organisation) et contrôler leur exécution.

Il est évident qu'un tel logiciel doit intégrer la logique de fonctionnement du système réel puisqu'il ne fait qu'agir au niveau du SI de telle sorte que celui-ci évolue "comme" la réalité qu'il représente.

Nous pensons qu'il est profitable de dissocier la logique de fonctionnement du système réel que le processeur doit connaître des mécanismes opératoires qu'il met en oeuvre pour assurer l'évolution du SI.

Pour attendre le plus grand niveau de généralité, il s'impose de faire l'un et l'autre à un niveau indépendant de toute considération technique.

L'idée de l'expression des logiciels à un niveau abstrait est commune à tous les domaines de la construction du logiciel informatique. Elle est associée à celle de niveau conceptuel des bases de données (ANS 77),(BEN 79), (KEN 76),(CHEN 76) et à celle de spécification formelle que l'on retrouve dans les travaux sur la programmation (LIS 74)(LIS 77)(GUT 77)(ICH 79)(GOG 75) (WUL 77), sur les transformations de programmes (ARS 79)(BAU 79)(BE 79)(SET 79) sur la coopération de programmes (BM 75)(HOA 78)(DIJ 75) ou les systèmes d'exploitation (CRO 75)(DON 72)(SHA 74)(DON 74).

Ces remarques nous ont conduit à spécifier le processeur de la dynamique en deux étapes :

- un niveau conceptuel où le processeur de la dynamique est défini comme une machine abstraite utilisant la description du fonctionnement du système réel sous la forme d'une structure conceptuelle.
- un niveau physique où le processeur de la dynamique est défini comme un logiciel Réalisé sur une machine concrète conformément à sa spécification conceptuelle.

2.1. Spécification Conceptuelle du Processeur de la Dynamique

La spécification du processeur de la dynamique que nous présentons satisfait donc plusieurs objectifs :

- elle permet une distinction claire entre la description de la logique de fonctionnement du système réel et la description des mécanismes du processeur qui l'utilise.
- elle garantit que l'on ait correctement traduit la logique de fonctionnement du système réel comme une description de la logique de fonctionnement du système d'information.
- elle définit le processeur de la dynamique à un niveau conceptuel, c'est-à-dire, à un niveau d'abstraction où on ne tient pas compte des moyens techniques liés à son implémentation.
- elle définit des mécanismes généraux qui assurent l'évolution de tout système d'information pourvu que le système d'information soit définit par sa logique de fonctionnement.

Pour spécifier le processeur de la dynamique en satisfaisant ces objectifs il est indispensable de disposer

. d'un modèle de définition de la logique de fonctionnement d'un système (ou modèle de la dynamique)

. d'un langage de spécification
. d'une machine abstraite adaptée à la gestion des systèmes abstraits définis par leur structure de fonctionnement

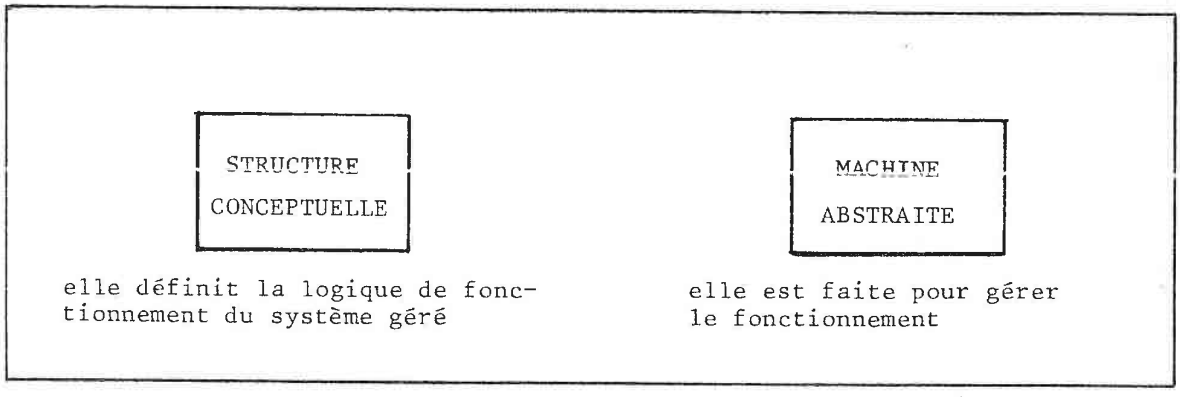
La spécification conceptuelle du processeur de la dynamique que nous proposons est la suivante :

1. représenter dans une structure conceptuelle unique le système réel et sa logique de fonctionnement.

2. associer à la description conceptuelle une machine abstraite définie une fois pour toutes. Cette machine comporte les mécanismes opératoires propres à assurer la gestion du fonctionnement d'un système défini par sa structure conceptuelle de fonctionnement.

La structure conceptuelle est décrite dans le langage de description associé à la machine abstraite. Elle correspond donc à la définition du système d'information et de ses règles d'évolution. Elle est basée sur les concepts du modèle de la dynamique.

La machine abstraite est indépendante de tout système représenté. Elle utilise la description de la structure conceptuelle, faite dans le langage de description qui lui est associé, pour gérer l'évolution du système abstrait qui lui correspond. Ses mécanismes sont donc fonction du modèle de la dynamique.



Spécification conceptuelle du processeur de la dynamique

2.1.1. LA STRUCTURE CONCEPTUELLE

Elle doit permettre une représentation correcte de la réalité et de son évolution.

Il nous semble que la meilleure façon d'atteindre ce but est d'exprimer la logique de fonctionnement de la réalité en termes de types, par une approche structuraliste où l'on cherche à définir les composants de cette réalité et leurs interrelations dynamiques.

La structure conceptuelle que nous proposons est analogue au concept de schéma conceptuel de données (ANS 77)(IFI 74)(IFI 76).

Le schéma conceptuel définit la structure de la base de données comme la représentation de la réalité par des types : les données sont la représentation des objets (c'est-à-dire, des constituants de la réalité) ; la structure de données est la représentation des liens structurels ou associations d'objets réels (KEN 76)(BEN 79)(CHEN 76).

La structure conceptuelle que nous devons définir est une structure plus complète. C'est une structure unique dans laquelle on décrit la structure statique (les composants de la Réalité) et la logique de fonctionnement de la réalité (les règles d'évolution et leur structure de synchronisation).

En conséquence il faut que le modèle de la dynamique nécessaire à cette approche comporte des concepts pour décrire la structure statique du SI et des concepts pour décrire sa structure dynamique.

Le modèle que nous avons retenu (ROL 78a)(ROL 78b) comporte trois catégories de phénomènes (OBJET, OPERATION, EVENEMENT) et trois catégories d'associations entre les catégories de phénomènes (MODIFIE (OPERATION, OBJET), CONSTATE (objet, événement), DECLENCHE (événement, opération)).

Il est présenté dans le chapitre I de ce travail.

L'expression de la structure conceptuelle doit être faite dans un langage approprié qui tienne compte de la spécificité de l'approche.

Ce langage appelé ISDEL, doit :

- être adapté à l'usage que la machine abstraite en fera
- être basé sur le modèle de la dynamique
- permettre que la description de la structure conceptuelle soit une spécification "fidèle", "complète", cohérente et non redondante du système réel.

Il est présenté dans le chapitre I en association avec le modèle de la dynamique.

2.1.2. LA MACHINE ABSTRAITE

La machine abstraite intègre les mécanismes qui assurent le fonctionnement de tout système réel décrit par une structure conceptuelle.

Elle fonctionne selon ses propres règles qui seront mises en oeuvre selon le schéma de fonctionnement du système à faire évoluer.

Les primitives de la machine abstraite sont :

- reconnaître les événements
- déclencher l'exécution des opérations
- mémoriser des représentations.

Le fonctionnement de la machine abstraite pour un système particulier peut être schématisé de la manière suivante : la machine abstraite reconnaît les événements qui se produisent, détermine les opérations qui doivent être déclenchées (en exploitant la structure conceptuelle correspondant au système), déclenche et contrôle leur exécution, vérifie si les changements d'état provoqués par les opérations sont des événements et, si oui, détermine et déclenche des actions conséquentes.

Nous introduisons la machine abstraite et sa spécification conceptuelle dans le chapitre II. La spécification conceptuelle est présentée dans le langage ISDEL.

3. REALISATION DU PROCESSEUR DE LA DYNAMIQUE

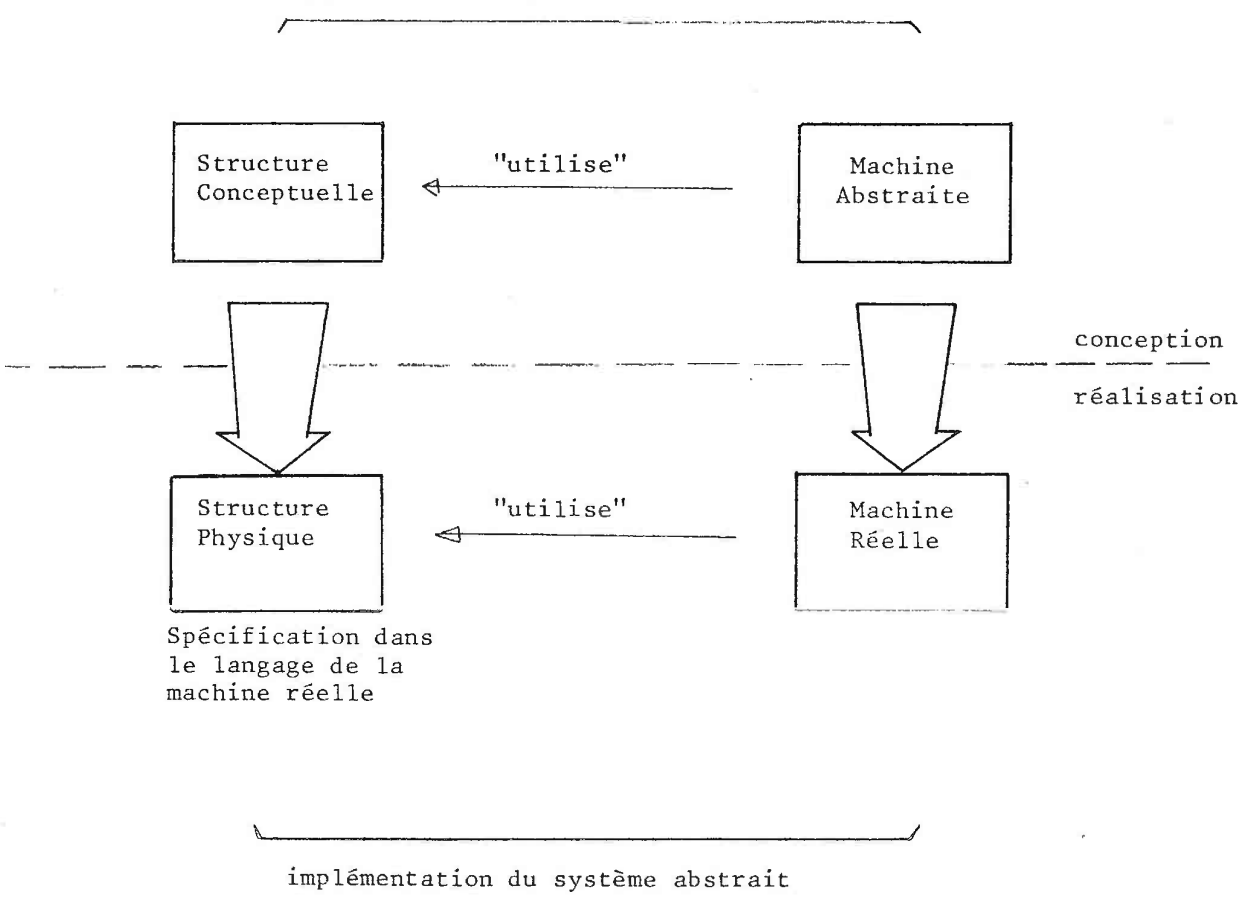
La réalisation de la machine abstraite correspond au passage de la spécification conceptuelle à la spécification physique.

Elle correspond :

. à la réalisation de la machine abstraite sur une machine réelle.
. l'implémentation de la structure conceptuelle (définie dans le langage associé à la machine abstraite)

Nous avons réalisé le processeur de la dynamique comme un programme COBOL qui utilise une base de données relationnelle. SYNTEX (DEM 75).

Cette réalisation est présentée dans le chapitre III.



4. L'APPROCHE STRUCTURALISTE DE CONCEPTION

Nous présentons dans le chapitre IV les conséquences que l'on peut attendre de la méthode structuraliste proposée et de l'outil.

Nous introduisons nécessairement l'intégration du processeur dans un outil de gestion de la dynamique, la politique de conception des systèmes en informatique de gestion et la généralisation de l'approche à la spécification du logiciel informatique.

I. LE LANGAGE DE LA MACHINE ABSTRAITE

I.1. INTRODUCTION AU LANGAGE DE LA MACHINE ABSTRAITE

I.1.1. LE ROLE DU LANGAGE

Le rôle du langage de la machine abstraite est de permettre la spécification conceptuelle d'un système d'information par l'expression de sa structure conceptuelle (c'est-à-dire, en termes de c-objets, c-opérations et c-événements).

Il doit constituer un système formel qui permettra au concepteur d'être rigoureux et précis et l'aide dans son travail en lui fournissant des concepts et des règles d'usage des concepts.

Il doit permettre une description complète en prenant en compte les aspects qui ne sont pas directement traduits par les concepts du modèle de la dynamique dans le formalisme associé. En d'autres termes, le langage doit donner en outre au concepteur la possibilité d'exprimer des contraintes d'intégrité sur les c-objets, c-opérations et c-événements.

I.1.2. LA SPECIFICITE DU LANGAGE

La dichotomie que l'on a toujours faite entre données et programmes a contribué au développement séparé d'une part des langages de programmation, d'autre part des langages de description de données.

Les langages de programmation sont orientés vers la description d'algorithmes complexes, mais ils n'admettent que des constructions pour définir des structures de données simples (tableaux, listes).

Les langages de données permettent la description et la manipulation de structures de données complexes mais en revanche ils limitent les possibilités d'écriture de programmes.

Le langage que nous proposons est un langage de spécification de systèmes d'information. Il est à la fois langage de programmation et langage de descriptions de données. Il a la même vocation que les langages présentés en (ABR 78) ou (BRE 79), mais il est défini pour s'adapter aux systèmes d'information conçus selon l'approche présentée.

Il intègre les aspects nécessaires à la description :

- des composants de l'organisation et leur structure,
- des règles de gestion de l'organisation et leur action sur les composants par :
  - . l'accès aux données qui représentent ces composants
  - . la manipulation de ces données
  - . les règles de calcul et les prédicats concernant ces données.
- de la synchronisation de l'exécution des règles de gestion.

I.1.3. LES CARACTERISTIQUES DU LANGAGE

Ce langage a des caractéristiques de type relationnel afin de :

- permettre la description des composants de la structure conceptuelle
- permettre l'expression des relations entre ces composants
- traduire l'accès aux données sans tenir compte de l'implémentation physique de ces données.
- exprimer des contraintes d'intégrité sur ces composants

Il intègre des caractéristiques procédurales afin de :

- décrire des expressions arithmétiques et logiques
- exprimer des procédures

I.2. QUE DÉCRIRE

Le langage que nous proposons est orienté pour la description des systèmes représentés selon les concepts du modèle conceptuel de la dynamique que nous présentons maintenant.

I.2.1. LE MODELE CONCEPTUEL DE LA DYNAMIQUE

Il s'agit d'un modèle conceptuel car il permet une description des systèmes réels et de leur fonctionnement indépendante des contraintes techniques.

Il a été défini par l'équipe de recherche et présenté notamment dans (ROL 78a) et (ROL79c) Sa définition est le résultat d'une analyse du système "organisation" en termes de catégories.

I.2.1.1. Les concepts du modèle de la dynamique

L'analyse des systèmes réels et de leur fonctionnement nous a conduit à identifier trois catégories de phénomènes (OBJET, OPERATION, EVENEMENT) et trois catégories d'associations entre les catégories de phénomènes : MODIFIE (opération, objet) ; CONSTATE (objet, évènement) et DECLENCHE (évènement opération).

A partir de cette analyse on définit ces phénomènes et ces associations de la façon suivante :

OBJET : est un constituant réel, durable, concret ou abstrait, d'un système réel qui peut être particularité (BEN 76)(CHE 76)(KEN 73)(LIN 74). Un objet traduit un aspect de l'état du système réel
exemple : le client "ROTENBERG", le produit "AI 35"

OPERATION est une action qui est unique et qui peut être exécutée seule, à un moment donné, dans le système réel et qui provoque le changement de la valeur d'une ou plusieurs caractéristiques d'un ou plusieurs objets. Une opération modifie l'état du système réel.
Exemple : l'opération "analyse de commande" numéro 327 crée l'objet "commande acceptée" numéro 208.

EVENEMENT est la constatation d'un changement d'état remarquable des valeurs d'une ou plusieurs propriétés d'un ou plusieurs objets. Ce changement d'état est remarquable car il entraîne l'évolution du système réel. Un événement est un changement d'état remarquable du système réel qui déclenche des opérations.

Exemple : l'évènement "arrivée de la commande" n° 316.

Nous remarquons la différence existante entre état, changement d'état et évènement.

état : l'état de l'objet est indiqué à un instant donné par les valeurs de ses propriétés. Il peut être durable

changement d'état : il est toujours instantanné. Il correspond au passage à de nouvelles valeurs d'une ou plusieurs propriétés de l'objet.

événement : c'est un changement d'état particulier qui entraîne une réaction dynamique du système réel. Il illustre le fait que tous les changements d'état d'un objet ne sont pas nécessairement des évènements.

Les définitions précédentes montrent que les trois catégories de phénomènes objets, opérations, évènements, ne sont pas indépendantes. Il existe trois catégories d'associations qui définissent leurs cas d'interconnections.

CONSTATE (EVENEMENT, OBJET) : est une association qui relie un évènement et un ou plusieurs objets. Elle exprime le fait que l'évènement est caractérisé par rapport au changement d'état d'objets.

DECLENCHE (EVENEMENT, OPERATION) : est une association qui relie un évènement et une ou plusieurs opérations. Elle exprime la réaction de l'organisations à un changement d'état particulier, elle indique quelles sont les opérations qui peuvent être déclenchées comme conséquence d'un évènement.

MODIFIE (OPERATION, OBJET): est une association qui relie une opération et un ou plusieur objets. Elle exprime quels sont les objets dont les propriétés ont des valeurs affectées par l'exécution de l'opération.

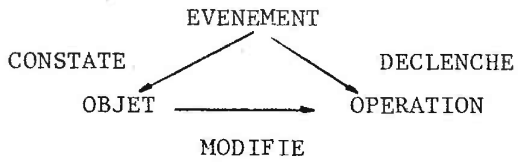
Exemples :

- l'évènement "arrivée de la commande" numéro 316 constate le changement d'état de l'objet "commande" numéro 417 puisqu'elle est passée de l'état non arrivée à l'état arrivée.

- l'évènement "arrivée de la commande" numéro 316 déclenche l'opération "analyse de commande" numéro 327.

- l'opération "analyse de commande" numéro 327 modifie l'objet "commande acceptée" numéro 208.

Nous pouvons résumer ces catégories de phénomènes et leurs associations par le schéma suivant :



La définition de la dynamique, traduite par ces associations, est de type causal : les évènements déclenchent l'exécution des opérations dont les effets sont les changements d'état d'objets qui peuvent être, à leur tour, des évènements.

I.2.1.2 Normalisation des concepts.

La définition donnée des différents phénomènes et de leurs associations est descriptive. Elle facilite la compréhension des phénomènes mais son tutilisation conduit à décrire les résultats de l'analyse d'un système réel par une collection d'énoncés qui peuvent être redondants par certains côtés, incomplets par d'autres et éventuellement ambigus.

Il est donc nécessaire de passer à une description correcte, c'est-à-dire de notre point de vue (ROL 78a)

fidèle (qui traduit la réalité de l'organisation sans distorsions)
complète (pour que tout questionnaire puisse retrouver l'image qu'il veut.
complète ne signifie pas exhaustive)
non redondante (sans répétition)
consistente (sans ambiguité et sans contradiction)

Ce passage correspond à la réelle tâche de conception. Il s'agit de transformer la collection initiale des énoncés en une collection sémantiquement équivalente mais complète, non redondante, non ambiguë. Pour le faire nous préconisons une normalisation des concepts du modèle et l'usage d'un formalisme qui permette leur expression, leur compréhension et leur vérification.

La collection finale est exprimée dans le formalisme relationnel qui décrit la version normalisée des concepts.

I.2.1.2.1. Contraintes sur les concepts

Ce sont celles que l'on introduit pour que le schéma conceptuel ait les qualités requises. Nous introduisons aussi une nouvelle forme de dépendance fonctionnelle (BER 78) pour les exprimer dans le formalisme relationnel : la dépendance fonctionnelle permanente (KRI 78)(ROL 79-c)

Dépendance Fonctionnelle Permanente : soient A et B des attributs de la relation R.

Soient Dom (A) et Dom (B) les domaines de A et B

soit f une fonction du temps telle que Dom (A) $\longrightarrow$ Dom (B)

f est une dépendance fonctionnelle permanente notée A $\dashrightarrow$ B si , à tout instant, depuis la date de création, à toute valeur a $\in$ Dom (A) il y a une unique et même valeur b $\in$ Dom (B). Une DFP est plus forte qu'une DF puisque si A $\dashrightarrow$ B, à tout instant b est le même et ne peut jamais changer.

On dit que a et b ont la même évolution au cours du temps.

Contraintes sur les évènements et les opérations : soient EVi une classe d'évènements, OPk une classe d'opérations, OBj une classe d'objets. Alors on a :

$\forall$ EVi $\longrightarrow$ $\exists$ OPk telle que $\exists$ (EVi, OPk)

$\forall$ EVi $\longrightarrow$ $\exists$ OBj tel que $\exists$ (EVi, OBj)

$\forall$ OPk $\longrightarrow$ $\exists$ OBj tel que $\exists$ (OPk, OBj)

$\forall$ OPk $\longrightarrow$ $\exists$ EVi tel que $\exists$ (EVi, OPk)

Contraintes sur la définition des associations : soient EVi une classe d'évènements, OPk une classe d'opérations, OBj et OBn des classes d'objets. Alors on a :

$\forall$ (EVi, OBj, OBn) $\longrightarrow \exists$ (EVi, OBj) et $\exists$ (EVi, OBn) $\Longrightarrow$ OBj = OBn

$\forall$ (OPk, OBj, OBn) $\longrightarrow \exists$ (OPk, OBj) et $\exists$ (OPk, OBn) $\Longrightarrow$ OBj = OBn

$\forall$ (EVi, OPk) $\longrightarrow \exists$ au plus une fonction booléenne $\mathcal{F}$ (OBi,..., OBj,...,OBm)
telle que étant donnés
evi $\in$ EVi
opk $\in$ OPk
alors (evi, opk) $\in$ (EVi, OPk) si et seulement si
$\mathcal{F}$ (OBi,..., OBj,..., OBm)=vrai ou $\nexists \mathcal{F}$(OBi,..., OBj,..., OBm)

Contraintes définissant la causalité : soient EVi une classe d'évènements OPk une classe d'opérations, OBj et OBn des classes d'objets (éventuellement OBn = OBj).

$\forall$ EVi $\longrightarrow \exists$ (EVi, OBj) $\Longrightarrow \exists$ OPk tel que (EVi, OPk)

$\forall$ OPk $\longrightarrow \exists$ (EVi, OPk) $\Longrightarrow \exists$ OBn tel que (OPk, OBn)

I.2.1.2.2. Définition normalisée des concepts

Le respect des contraintes introduites et l'emploi du formalisme relationnel nous conduisent à donner une définition normalisée aux concepts du modèle de la dynamique qui nécessite l'introduction de trois types désignés l'OBJET, C-OPERATION, C-EVENEMENT.

I.2.1.2.2.1. Le type C-OBJET

Définition : Une relation de type C-OBJET est une relation permanente, c'est-à-dire, une relation en troisième forme normale (BER 78) dont tous les constituants sont en dépendance fonctionnelle permanente avec l'identifiant-clé.

Exemple : Considérons la classe d'oobjets réels "FOURNISSEUR", caractérisée par les propriétés numéro d'INSEE (INSEE), non (NOM), adresse (ADRESSE) et chiffre d'affaires (CHIFFAFF).
La description de cette classe d'objets par l'intermédiaire d'une relation en troisième forme normale serait la suivante :

FOURNISSEUR (NFOUR, INSEE, NOM, ADRESSE, CHIFFAFF)

La description de cette classe en termes de c-objets conduit aux relations suivantes :

FOURNISSEUR-NOM (NFOUR, NOM, INSEE)
FOURNISSEUR-ADRESSE (NFOUR, DATE-ADRESSE, ADRESSE)
FOURNISSEUR-CHIFFAFF (NFOUR, DATE-CA, CHIFFAFF)

En effet les quatre propriétés de la classe FOURNISSEUR ont des comportements dynamiques distincts :

- le numéro INSEE et le nom sont constants
- l'adresse et le chiffre d'affaires sont des propriétés du fournisseur qui sont modifiées dans des circonstances distinctes avec des rythmes temporels distincts.

Chacune de ces trois relations représente un aspect de la classe FOURNISSEUR.

Sémantique : une relation de type C-OBJET représente un aspect dynamique d'une classe de phénomènes réels de la catégorie objet. En d'autres termes, il représente la plus grande collection de propriétés d'une classe d'objets du monde réel ayant un même comportement dynamique, c'est-à-dire, créé, modifié, supprimé en même temps : un C-OBJET représente l'évolution au cours du temps d'un certain aspect d'une classe d'objets du monde réel. Un objet est représenté dans le système d'information par une réalisation d'une relation de type C-OBJET. Une relation de type C-OBJET représente un état élémentaire du système réel.

I.2.1.2.2.2. Le type C-OPERATION

Définition : Une relation de type C-OPERATION est une relation en troisième forme normale dont tous les constituants sont en dépendance fonctionnelle permanente avec l'identifiant clé.

Exemple : Considérons la classe d'opérations "acceptation d'une ligne de commande". Elle est décrite par les trois schémas de relation suivants :

(1) acceptation-leg-com-per (nacligco, date-cre-acligco)
(2) acceptation-leg-com-texte (nacligco, date-texte-acligco,acligco-texte)
(3) acceptation-leg-com-mod (nacligco, date-mod-acligco, ligcoaliv)

La relation (1) introduit la classe d'opérations et sa date de création
La relation (2) introduit le texte de l'opération
La relation (3) décrit les modifications successives des objets.

Sémantique : une relation de type C-OPERATION représente un aspect temporel d'une classe d'opérations réelles. Elle décrit un aspect particulier et élémentaire de la classe représentée. Il faut remarquer qu'une C-OPERATION est définie par référence à un seul et unique C-OBJET et que la transformation qu'elle lui fait subir est traduite par un seul et unique texte dont l'existence est indispensable.

Les relations de type C-OPERATION représentent une action élémentaire qui provoque un changement d'état élémentaire dans le système réel : elle représente la plus petite transformation du système réel.

I.2.1.2.2.3 Le type C-EVENEMENT

Définition : Une relation de type C-EVENEMENT est une relation en troisième forme normale dont tous les constituants sont en dépendance fonctionnelle permanente avec la clé.

Exemple : Considérons la classe d'évènements "arrivée d'une ligne de commande". Elle est décrite par les cinq schémas de relation suivants :

(1) an-lig-com-per (nanléco, date-cre-anléco)
(2) an-lig-com-pred (nanléco, date-pred-anléco-pinit,anléco-pinit,anléco-pfin)
(3) an-lig-com-constate (nanléco, date-constate-anléco, nlicoin)
(4) an-lig-com-déclenche (nanléco, nacligco, date-def-déclenche, condition, facteur)
(5) an-lig-com-déclenche-réel (nanléco, date-constate-anléco, nacligco, date-modifie-acligo , date-déclenche-réel)

La relation (1) introduit la classe d'évènements et sa date de création
La relation (2) fournit le texte des prédicats
La relation (3) traduit les constatations successives d'évènements
La relation (4) traduit l'association DECLENCHE et ses propriétés
La relation (5) traduit les déclenchements successifs des opérations.

Sémantique : une relation de type C-EVENEMENT représente une classe d'évènements réels. Elle décrit un aspect élémentaire et particulier de la classe représentée. Il faut remarquer qu'un C-EVENEMENT est défini par référence à un seul et unique C-OBJET dont le changement d'état est décrit par un couple de prédicats (initial et final). L'existence de ces prédicats est indispensable pour caractériser la classe d'évènements. Par ailleurs, un C-EVENEMENT est défini par référence à une ou plusieurs C-OPERATIONS déterminées. La conditions qu'autorise le déclenchement d'une opération d'une C-OPERATION ainsi que le facteur qui indique s'il s'agit d'un déclenchement itératif sont des propriétés indispensables à la description d'un évènement.

Un C-EVENEMENT représente un changement d'état élémentaire remarquable du système réel, déclenchant des actions élémentaires.

I.2.1.3. Exemple

Nous présentons comme exemple la structure conceptuelle correspondant à un système de vente par correspondance. Nous utiliserons cet exemple pour illustrer les différentes caractéristiques de ISDEL au chapitre I.3.

Nous donnons l'expression graphique du schéma statique (système de C-classes d'objets), du schèma dynamique (comprenant les C-OBJETS, C-OPERATIONS, C-EVENEMENTS).

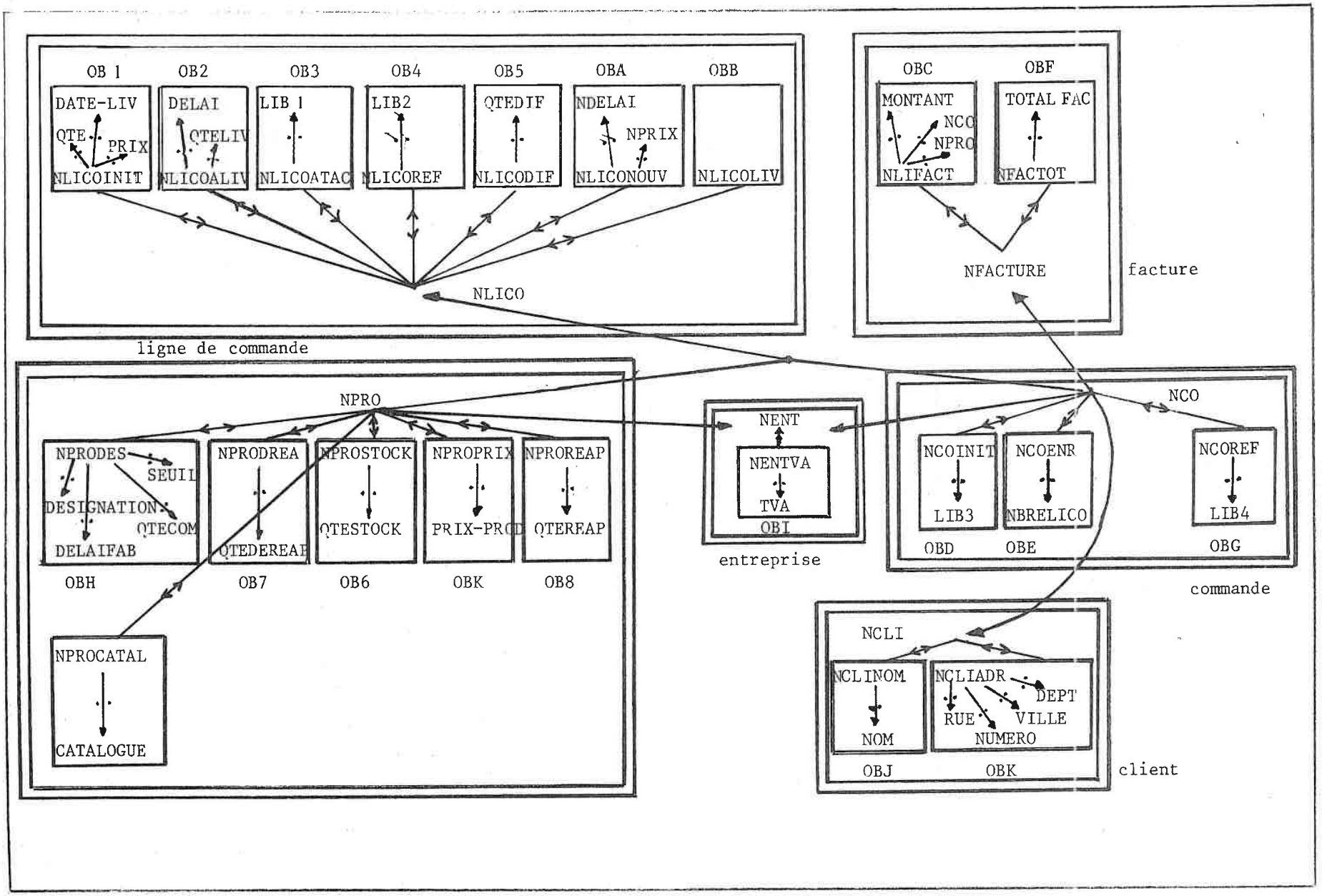
1.2.1.3.1. L'énoncé du problème

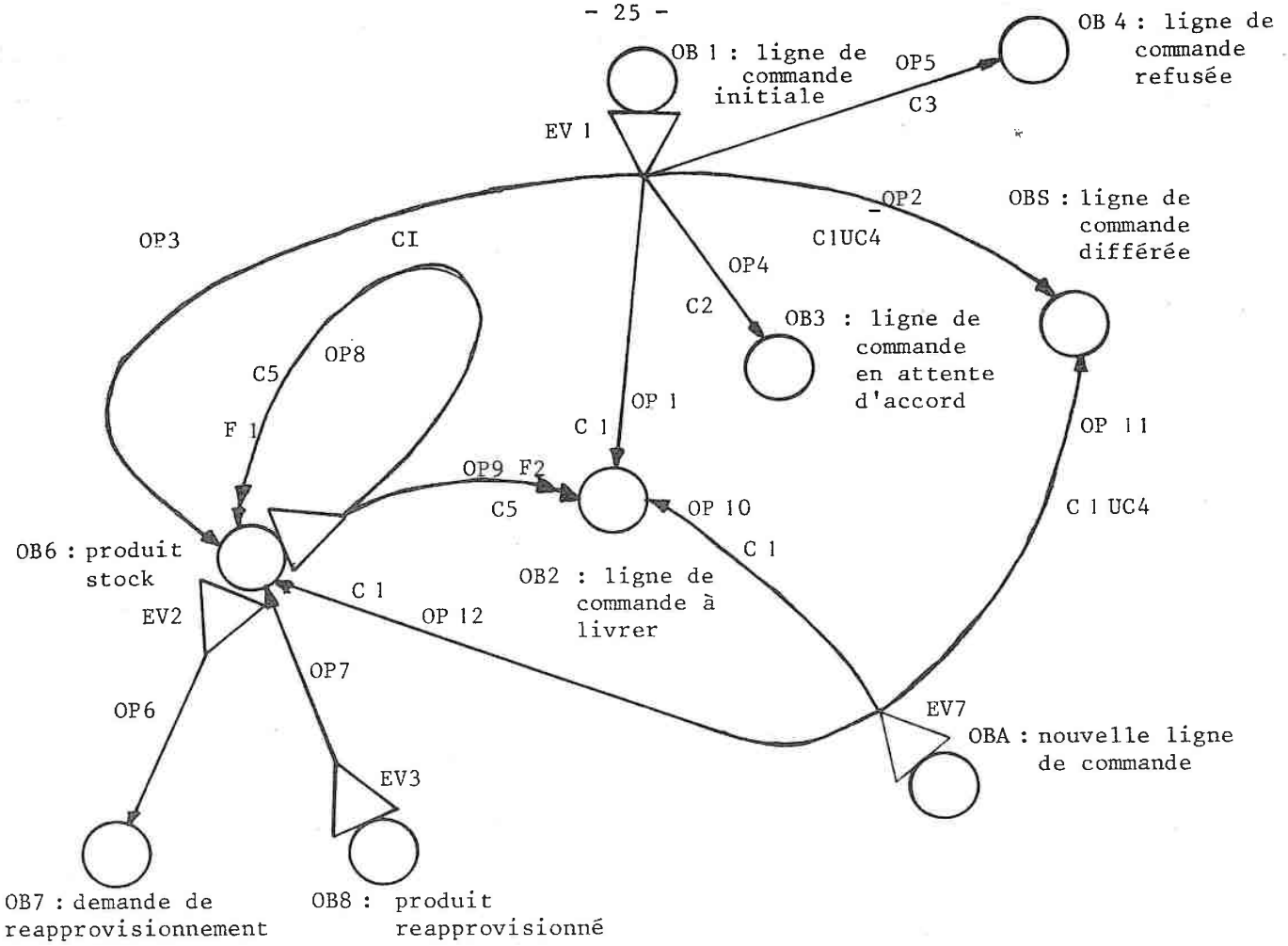
- Une entreprise vend 25000 articles catalogués à des entreprises industrielles :

- Une commande ne peut pas être enregistrée si un client n'est pas connu de l'entreprise.

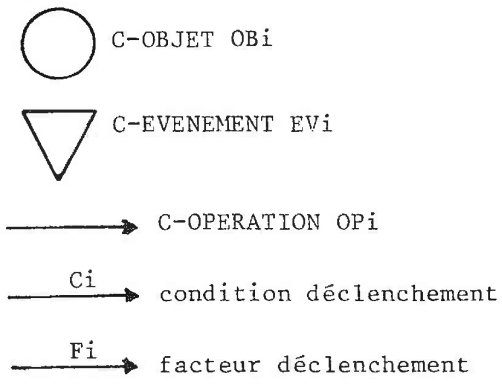
- Un numéro d'identification est donné à une commande par le service commercial.

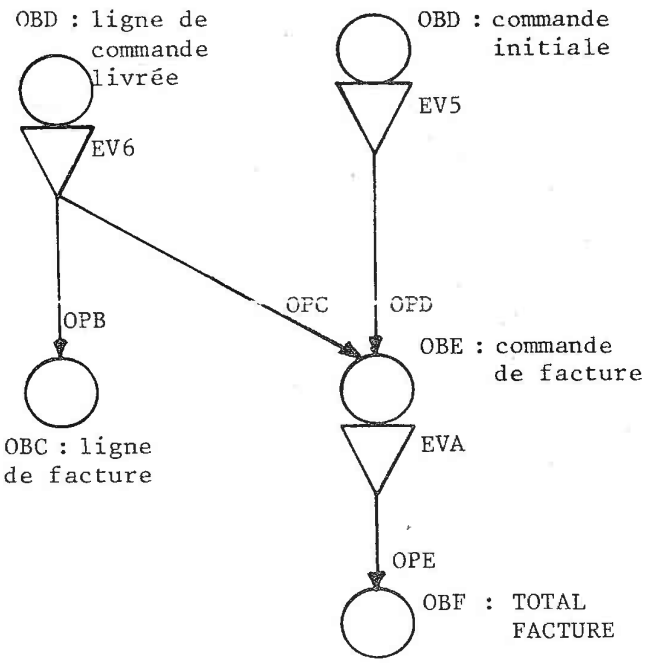
- Une commande enregistrée peut avoir plusieurs lignes de commande
- Chaque ligne de commande correspond à un produit différent
- Une ligne de commande peut être :
  - . refusée quand le produit commandé n'est pas "catalogué". Elle n'est plus jamais considérée dans la commande.
  - . différée jusqu'à consentement - sur le prix
    - sur la date de livraison
  - . acceptée quand il y a accord sur le prix et la date de livraison
- Au moment de la livraison, une commande peut être momentanément différée jusqu'à ce que le produit soit réapprovisionné. La facture de la commande est envoyée seulement après livraison complète de toutes les lignes de commande non refusées.


OB 1
OB2
OB3
OB4
OB5
OBA
OBB
DATE-LIV
QTE
PRIX
NLICOINIT
DELAI
QTELIV
NLICOALIV
LIB 1
NLICOATAC
LIB2
NLICOREF
QTEDIF
NLICODIF
NDELAI
NPRIX
NLICONOUV
NLICOLIV
NLICO
ligne de commande
OBC
OBF
MONTANT
NCO
NPRO
NLIFACT
TOTAL FAC
NFACTOT
NFACTURE
facture
NPRO
NPRODES
SEUIL
DESIGNATION
QTECOM
DELAIFAB
OBH
NPRODREA
QTEDEREAL
OB7
NPROSTOCK
QTESTOCK
OB6
NPROPRIX
PRIX-PRCD
OBK
NPROREAP
QTEREAP
OB8
NPROCATAL
CATALOGUE
NENT
NENTVA
TVA
OBI
entreprise
NCO
NCOINIT
LIB3
OBD
NCOENR
NBRELICO
OBE
NCOREF
LIB4
OBG
commande
NCLI
NCLINOM
NOM
OBJ
NCLIADR
DEPT
RUE
VILLE
NUMERO
OBK
client



Convention graphique





LEXIQUE DES OPERATIONS

OP1 : création de la ligne de commande à livrer
OP2 : création de la ligne de commande différée
OP3 : mise à jour du stock (pour servir la ligne de commande à livrer)
OP4 : mise en attente d'une ligne de commande (à cause de manque de stock)
OP5 : refus d'une ligne de commande
OP6 : création d'une demande de réapprovisionnement d'un produit
OP7 : mise à jour du stock (suite à l'arrivée d'un réapprovisionnement)
OP8 : mise à jour du stock (pour servir la ligne de commande à servir correspondant à la ligne de commande différée qui peut être servie)
OP9 : création d'une ligne de commande à servir (correspondant à la ligne de commande différée qui peut être servie)
OP10 : création d'une ligne de commande à livrer (correspondant à la ligne de commande en attente d'accord)
OP11 : mise en attente d'une ligne de commande (correspondant à la ligne de commande en attente d'accord)
OP12 : mise à jour du stock (pour servir la ligne de commande à servir correspondant à la ligne de commande en attente d'accord).

LEXIQUE DES EVENEMENTS

EV1 : arrivée d'une ligne de commande
EV2 : rupture de stock d'un produit
EV3 : arrivée d'un réapprovisionnement
EV4 : augmentation du stock
EV5 : arrivée d'une commande
EV6 : la ligne de commande a été livrée
EV7 : arrivée de l'accord du client
EVA : la commande a été complètement livrée.

LEXIQUE DES CONDITIONS DE DECLENCHEMENT

C1 : $\overline{C2}$ U $\overline{C3}$

C2 : Prix ou délai incorrect

C3 : Produit provisoirement mis hors du catalogue

C4 : Le stock du produit est supérieur à la quantité à servir

C5 : La quantité en stock du produit peut satisfaire la ligne de commande différée.

LEXIQUE DES FACTEURS DE DECLENCHEMENT

F1 : définit les mises à jour du stock correspondant aux lignes de commande à servir qui seront créées (parce qu'il y a eu réapprovisionnement du stock et il y avait des lignes de commande différées).

F2 : définit les lignes de commande à servir qui seront créées (à cause de l'existence de lignes de commande différées) après réapprovisionnement du stock.

I.3. COMMENT DÉCRIRE

1.3.1. CHOIX DE CONCEPTION DE ISDEL

Les choix que nous avons fait dans la définition du langage ISDEL tiennent compte du formalisme relationnel associé au modèle et ont pour but de faciliter la tâche du concepteur, l'écriture des spécifications. Ils conduisent à un langage intégré de description des structures d'information, de manipulation de données et de calcul.

Nous présentons successivement les caractéristiques liées à l'aspect relationnel, à l'aspect procédural et les clauses de base permettant la description d'une structure conceptuelle.

1.3.2. ASPECT RELATIONNEL

L'approche conceptuelle que nous utilisons nous conduit à décrire un système d'information par des relations de différents types.

L'aspect relationnel de ISDEL doit permettre :

- de reconnaître ces relations
- d'exprimer des prédicats sur les attributs de ces relations
- d'exprimer des prédicats entre les relations

Cela rend possible d'une part l'expression des accès aux composants du SI et leur manipulation et d'autre part, l'expression des contraintes d'intégrité qu'ils doivent respecter.

La syntaxe choisie est basée sur le calcul relationnel (COD 72) par sa non-procéduralité , la rigueur qu'il permet et le nombre réduit de concepts qu'il nécessite d'employer pour construire une expression.

Le désavantage de ce choix est lié au fait que certains considèrent difficile l'utilisation du formalisme voisin du calcul des prédicats du premier ordre par l'utilisateur occasionnel.

I.3.2.1. Rappel du calcul relationnel

Nous nous inspirons de (CODD 72) et de(PIR 76) pour présenter ce rappel.

Alphabet du calcul relationnel :

constantes individuelles	: a1, a2, a3,...
constantes indice	: 1, 2, 3, 4,...
variables tuple	: r1, r2, r3,...
prédicats	
"nomadic"	: P1, P2, P3
"dyadic"	: =, <, >, = >, =<, ≠
symboles logiques	: ∃, ∀, ∧, ∨, ¬
délimiteurs	: [ ] ( ),

Termes du calcul relationnel

Il y a deux types possibles de termes

Domaine : défini par un prédicat "monadic" suivi d'une variable tuple Le terme domaine Pj r signifie que la variable r a la relation Rj de la base de données comme domaine.

Un tuple indexé de la forme r [N] où r est une variable tuple et N une constante indice identifie le n-ième composant du tuple r.

Jonction : soit θ un prédicat dyadic. Soient x et y des tuples indexés et w une constante individuelle. Alors x θ y et x θ w sont des termes jonction.

Formules : les formules proprement définies (WFF) du calcul relationnel sont construites récursivement de la façon suivante :

1. Tout terme est une WFF
2. Si Γ est une WFF alors ¬Γ l'est
3. si Γ1, Γ2 sont WFFs, alors (Γ1 ∧ Γ2) et (Γ1 ∨ Γ2) le sont.
4. Si Γ est une WFF où r est une variable libre, alors
   ∃r (Γ) et ∀r (Γ) le sont.
5. Il n'y a pas d'autres WFF

Domaine WFF est une formule WFF sans quantificateurs dont tous les termes sont des termes domaines

domaine WFF sur r est un domaine WFF dont la seule variable libre est r

domaine WFF propre sur r est un domaine WFF sur r tel que :

a. soit $\neg$ n'apparaît pas soit il suit immédiatement un $\wedge$
b. si r est présent dans deux termes domaine ou plus, alors les prédicats domaine dans ces termes doivent être associés à des relations qui sont "union-compatible".

Ces deux contraintes obligent les variables tuples à avoir comme domaine soit les relations définies soit des relations qui peuvent être genenées à partir d'elles par l'application des opérateurs ensemblistes d'union, intersection et différence à des couples de relation "union-compatible".

quantificateurs "liés aux domaines" sont définis de la façon suivante :

. soit Δ une WFF ayant r comme variable libre mais sans avoir de terme domaine sur r
. soit Γ un domaine WFF propre sur r

$\exists \Gamma (\Delta) = \exists r (\Gamma \wedge \Delta)$

$\forall \Gamma (\Delta) = \forall r (\neg\Gamma \vee \Delta)$

On peut donc définir une classe de WFF ayant les domaines de leurs variables précisément définis.

Une WFF est séparable en domaine si elle est une conjonction de la forme :

U1 $\wedge$ U2 $\wedge$ ... $\wedge$ Un $\wedge$ V où

1. n -> 1
2. U1 à Un sont domaine WFF propre sur n variables tuples distinctes
3. V est soit nul soit une WFF ayant les propriétés :

   a. tout quantificateur dans V est "lié au domaine"
   b. toute variable libre dans V appartient à l'ensemble dont les domaines sont spécifiés par U1, U2,..., Un
   c. V ne contient pas de terme domaine.

expression alpha a la forme (ta, tz,..., tn) : w

où a. w est une WFF séparable en domaine

b. t1, t2,..., tk sont des termes distincts, chacun d'entre eux étant soit une variable tuple soit une variable tuple indexée

I.3.2.2. Expression du calcul relationnel en ISDEL

Nous établissons la correspondance suivante entre l'aspect relationnel d'ISDEL et le calcul relationnel :

a) Constantes indice : en ISDEL on désigne l'attribut de la relation par son nom (ou par une variable dont la valeur est un nom d'attribut).

b) terme domaine : en ISDEL on le désigne par le nom d'une relation (ou par une variable dont la valeur est un nom de relation) suivi d'une variable tuple entre les démimiteurs < >.

c) tuple indexé : il a la forme r. N ou r est une variable tuple et N le nom d'un attribut d'une relation

d) symboles logiques : on utilise AND au lieu de $\wedge$
: on utilise OR au lieu de $\vee$
on utilise ALL au lieu de $\forall$
on utilise EXIST au lieu de $\exists$
on utilise NOT au lieu de $\neg$

I.3.2.3. Exemples d'utilisation

Considérons le schéma composé par les "schémas de relation" suivants :

Fournisseur (NFOUR, NOM, ADRESSE)
Article (NART, COULEUR, POIDS)
Projet (NPROJ, DESIGNATION, ADRESSE)
Approvisionnement (NFOUR, NART, NPROJ, QUANTITE)

Nous présentons l'emploi de ISDEL sur des expressions qui s'appliquent à ce schéma et qu'on appelle : expressions relationnelles ISDEL.

a) Donner les détails des fournisseurs

(x) : fournisseur <x>

b) Donner les numéros des fournisseurs qui approvionnent le projet P1 avec les articles A1

(x. NFOUR) : Approvisionnement < x > AND x. NART = 'A1' AND x NPROJ = 'P1'

c) Donner les numéros des fournisseurs qui approvisionnent les projets P1 et P2

(x. NFOUR) : Fournisseur < x > AND Δ approvisionnement < y > AND
EXIST <y> (y. NFOUR = x.NFOUR AND y. NPROJ = 'P') AND
EXIST <y> (y. NFOUR = x.NFOUR AND y. NPROJ = 'P2')

d) Donner les numéros de projet qui ne sont pas approvisionnés en articles verts par aucun fournisseur dont l'adresse est Paris

(x.NPROJ) : Projet <x> AND Fournisseur <y> AND Article <z> AND
Approvisionnement <w>
AND NOT EXIST <w> (w.NPROJ = x.NPROJ
AND EXIST <y> (y.NFOUR = w NFOUR AND y. ADRESSE = 'PARIS'
AND EXIST <z> (z.NART = w NART AND z. COULEUR = VERT)))

e) Donner le numéro des fournisseurs qui approvisionnent tous les projets avec une même pièce

(x. NFOUR) : Fournisseur <x> AND Projet < y > AND Article <z> Approvisionnement <w> AND EXIST <Z> (ALL <y> (EXIST <w>
(w. NFOUR = x. NFOUR AND
w. NART = z. NART AND
w. NPROJ = y. NPROJ)))

I.3.3. ASPECT PROCEDURAL

L'aspect procédural est impliqué par la nécessité de décrire en ISDEL des calculs qui peuvent être complexes bien que "explicites" représentant les règles de gestion de l'organisation, leurs conditions d'exécution, leurs facteurs de déclenchement et les changements d'état événementiels.

Pour décrire cet aspect nous sommes inspirés du langage PASCAL (WIR 71) et des différentes extensions qui lui ont été proposées, telles que PASCAL-R (SCH 77) PLAIN (wAS 78) en l'intégrant le mieux possible à l'aspect relationnel.

Nous ne présentons dans la suite que les modifications que nous introduisons par rapport à l'usage courant de PASCAL. Une présentation complète de ce langage PASCAL se trouve dans (WIR 71) et dans (WIR 73) et (WIR 76)

I.3.3.1. Aspects concernant les structures de données et de contrôle

Ils concernent la définition d'un nouveau type, son utilisation dans la description des variables et la structure de contrôle liéé au traitement des variables de ce type.

I.3.3.1.1. Type RELATION

Le type RELATION décrit un domaine où tout élément est structuré comme une liste de domaines appelés champs. Chaque champ est défini par un identificateur et par le type du champ.

Les valeurs du domaine décrit par le type RELATION sont un sous ensemble du produit cartésien des champs qui le composent.

La valeur d'une variable d'un type relation particulier est donc un ensemble de tuples. Ils ont la même structure que celle qui est définie par la liste de champs dans la déclaration du type relation en question.

L'utilisation des variables de type relation permet de construire des relations locales à partir de celles qui composent la base de données et de les manipuler sans modifier la base.

Syntaxe : la structure d'un type relation peut être déclarée soit comme une liste de champs soit comme étant associée à l'identificateur d'un type RECORD préalablement déclaré.

<type-relation> :: = RELATION <structure-relation> END
<structure-relation> :: = <liste-de-champs> <identificateur-type>
<liste-de-champs> :: = <identificateur> : <type>
<identificateur> : <type> ; <liste-de-champs>

Exemples :

a) considérons le cas où la structure du type relation est définie par une liste de champs

type prod-nel = RELATION npro : integer ; prix : real END

b) considérons le cas où la structure du type relation est définie à partir d'un identificateur de type RECORD.

type produit-nec = RECORD npro : integer ; prix = real END
produit-rel = RELATION produit-rec END

La déclaration d'une variable de type relation est analogue à celle des variables d'autres types. (voir annexe I)

On a deux constructions possibles =

a) <identificateur-variable > : <identificateur-type >

Par exemple : type prod-nel = RELATION npro : integer ; prix : neal END ;
var produit-var : prod-nel ;

b) <identificateur-variable> : <type-relation>

Exemples : 1. var produit-var : RELATION npro : integer ; prix : real END ;

2. type produit-rec = RECORD npro = integer ; prix : real END ;
var produit-var : RELATION produit-rec END

I.3.3.1.2. La stucture FOREACH

Elle a été introduite pour exprimer la manipulation des tuples qui composent le domaine associé à une variable d'un type relation donné. Son utilisation est obligatoire si on veut accéder et traiter un ou plusieurs tuples d'une telle variable.

La structure FOREACH à la forme générale suivante :

FOREACH <variable> IN <relation-ident> [UNTIL <expression>]
DO <instruction> END

Cette structure traduit la répétition de <instruction> soit autant de fois qu'il y a de tuples en <relation-ident> soit jusqu'à ce que la condition d'arrêt décrite par <expression> soit vérifiée.

Pour chaque répétition de <instruction> un nouveau tuple de la relation identifiée par <relation-ident> est affecté à <variable>

On considère que <variable> a le même type relation que la relation identifiée par <relation-ident>

La portée de <variable> est l' <instruction> qui suit la clause DO.
On désigne par <relation-ident> :

- soit une relation qui est définie dans ISDEL
- soit une variable de type relation

Dans le premier cas, la désignation des champs qui composent la structure de <variable> est la même que celle qui a été utilisée pour définir la relation.

Dans le deuxième cas, la désignation des champs est celle employée lors de la définition du type de la <variable>.

Dans les deux cas, la référence à un champ est faite par la notation suivante :

<variable> . <désignation du champs>

Avant de présenter des exemples d'utilisation de la structure FOREACH nous introduisons les opérateurs qui permettent la manipulation des variables de type relation.

I.3.3.1.3 Opérateurs associés aux variables de type relation

Les variables de type relation ont comme valeur un ensemble de tuples et peut leur associer des opérateurs qui permettent la manipulation de ces tuples.

I.3.3.1.3.1. Affectation

Il permet l'affectation d'un ensemble de tuples à une variable de type relation.

Il est noté par : < et on l'utilise dans deux cas.

Affectation par transfert : on affecte à la variable de type relation un ensemble de tuples défini par une expression relationnelle ISDEL appliquée à des variables de type relation et à des relations de la base de données.

Exemples : (nous utilisons la base de données définie par les relations présentées en I.3.2.3 : Fournisseur, Article, Projet, Approvisionnement dans tous les exemples qui suivent)

a) soit un extrait de description

VAR prod-poids : relation num-ant : integer ; val-poids : real end ;
begin
prod-poids : <(x.nant, x.poids):article <x> AND x.poids => 10;

On affecte à "prod-poids" l'ensemble de tuples obtenus à partir de la relation article tels que le poids de l'article soit supérieur ou égal à 10.

b) soit un extrait de description

```
VAR fournis : relation num-four : integer; nom-four : char end ;
    new-fournis : relation neuw-four : integer ; new-nom-four : char end ;
begin
    fournis : <(x.nfour, x.nom) : fournisseur <x> AND x. adresse ='PARIS';
    new-fournis : <(x) : fournis <x> AND x. num-four > 300;
```

On affecte à "fournis" le numéro et le nom des fournisseurs qui sont à Paris et on affecte à "new-fournis" le numéro et le nom des fournisseurs dont le numéro est supérieur ou égal à 300.

Affectation par calcul : dans ce cas l'affectation à la variable de type relation d'un ensemble de tuples est faite tuple par tuple, par adjonctions successives.

Exemples

a) soit un extrait de sescription

```
VAR prod-couleur : relation pc-nart:integer ; pc-couleur : char ;
    pc-poids : real end ; prod-poids : relation pp-mart:integer ;
    pp-poids : real end ; new-poids : real ;
begin
    prod-couleur : <(x) : article <x> AND (x.couleur ='vert' OR
    x.couleur = 'GRIS');
    foreach prod in prod-couleur do
    begin
       new-poids : = prod. pc - poids * 1.2 ;
       prod-poids : < (prod. pc-nart, new-poids)
    end
end
```

On affecte à "prod-couleur" tous les tuples correspondant aux articles dont la couleur est VERT ou GRIS.

Pour chacun de ces tuples on calcule le nouveau poids et on ajoute un nouveau tuple à "prod-poids".

Finalement, "prod-couleur" et "prod-poids" ont un même nombre de tuples.

I.3.3.1.3.2. Suppression

Elle traduit la suppression d'un tuple de l'ensemble de tuples associé à la variable.

On utilise la notation : - pour l'indiquer

Exemple :

a) soit un extrait de description

```
var prod-poids : relation pp-nart : integer ; pp-poids : real end ;
begin
    prod-poids : <(x.nart, x.poids) : article <x> ;
    prod-poids : - (10,500)
```

On élimine de "prod-poids" le tuple correspondant au produit 10 dont le poids est 500.

I.3.3.1.4. Exemples d'utilisation de la structure FOREACH

Le schéma conceptuel considéré est le suivant :

Fournisseur (nfour, nom, adresse)
Article (nart, couleur, poids)
Projet (nprof, désignation, adresse)
Approvisionnement (nfour, nart, nproj, date-approvionnement, quantité)

a) exploitation d'une relation appartenant au schéma conceptuel, sans condition d'arrêt

```
var total : real ;
begin
total : = 0 ; foreach art in article do total : = total+art.poids end ;
```

On considère que art a la même structure que la relation "article". En conséquence on peut écrire art. poids. On n'a pas besoin de décrire "article" comme une variable de type relation puisqu' "article" est une relation du schéma. Toutes les tuples de la relation "article" sont traitées par l'expression FOREACH donnée.

b) exploitation d'une relation appartenant au schéma conceptuel avec condition d'arrêt

```
var total : real end ; compteur : integer ;
begin
total : = 0 ; compteur := 0 ;
foreach art in article until compteur =  15 do
 begin
       total:= total + art.poids ; compteur:= compteur + 1
 end end ;
```

c) exploitation d'une variable de type relation

```
var prod-poids : relation num-art : integer ; poids-ant : real end ;
total : real ;
begin
prod-poids : <(x.nprod, x.poids) : article<x> AND fournisseur <y> AND
   approvisionnement  < z > AND EXIST < z >(x.nart=z.nart AND EXIST < y >
   (y.nfour=z.nfour AND y. adresse = 'PARIS')) ;
   total : = 0 ;
   foreach prod in prod-poids do total := total + prod.poids end ;
   .
   .
   .
```

Dans cet exemple, prod-prix comprend les tuples qui indiquent le code et le poids des articles fournis par les fournisseurs de PARIS.

On utilise chacune de ces tuples pour évaluer le poids total qu'on aurait à transporter si on prenait une unité de chaque article fourni par les fournisseurs de PARIS.

d) imbrication de structures FOREACH

On peut avoir autant de niveaux d'inbrication de structures FOREACH qu'on le désire. Les règles concernant la portée des variables (quel que soit leur type) sont les mêmes que celles de PASCAL.

```
var approv : relation num-ant : integer ; qte-four : real end ;
    prod-poids : relation num-ant : integer ; poids-ant : real end ;
    total : real
begin
    approv : <(x.nart, x.quantité) : approvisionnement <x> ;
    prod-poids : <(x.nart, x.poids) : article <x>  AND x.COULEUR='VERT' ;
    total : = 0 ;
foreach ap in approv do
begin
   foreach  propo in prod-poids do
     begin if ap. num-ant = propo. num-ant then
        total:= propo. poids-ant *  ap. qte-four + total
           end
     end end
end end ;
```

Cette expression calcule la valeur de "total" comme cumul du poids de tous les articles verts fournis.

I.3.3.2. Aspects concernant la manipulation des données et les traitements

Ils concernent les opérateurs de manipulation du SI, les cas particuliers d'utilisation de fonctions et procédures, les fonctions relationnelles pré-définies et l'expression sur l'ordre des tuples d'une relation.

I.3.3.2.1. Expression de la manipulation des relations

On a choisi, d'une part pour des raisons de clarté d'écriture et d'autre part pour permettre au compilateur des contrôles distincts, d'introduire des opérateurs spécifiques pour la création, la suppression et la mise-à-jour du SI.

I.3.3.2.1.1. Insertion

L'opérateur d'insertion a la structure de la forme :

insert (liste-de-champs) in nom-de-relation

où liste-de-champs : identifiée soit par des variables qui contiennent les valeurs du nuple qu'on veut introduire soit par des valeurs

nom-de-relation : représente soit par le nom d'une relation du SI soit par une variable. Cette dernière possibilité est employée dans la spécification de la machine abstraite (chapitre III).
On utilise le plus souvent la première construction.

L'exécution d'un INSERT conduit à l'adjonction d'un nuple de la relation "nom-de-relation" dans le SI.

Exemples : considérons la relation : article (nart, couleur, poids) du SI

a) soit un extrait d'um programme

```
begin
     insert (10, 'VERT', 13572) in article
end
```

b) soit un extrait d'un programme

```
var num-art : integer ; couleur-art : char ; poids : real ;
    nom-rel : char ;
begin
    num-art := 10 ; couleur-art := 'VERT' ; poids:= 13572 ;
    nom-rel := 'ARTICLE' ;
    insert (num-ant, couleur-ant, poids) in nom-rel
end ;
```

Dans cet exemple on insère le nuple (10, 'VERT', 13572) dans la relation article (désignée par nom-rel).

I.3.3.2.1.2. Actualisation

L'opérateur actualisation a la structure de la forme :

update (liste-de-champs) in nom-de-relation

où liste-de-champs : identifiée soit par des variables qui contiennent les valeurs du nuple qui remplace un nuple du SI soit par des valeurs.

nom-de-relation : représenté soit par le nom d'une relation du SI soit par une variable.

L'exécution d'un "update" conduit à la mise-à-jour du nuple de la relation "nom-de-relation" du SI qui a la même valeur pour l'identifiant clé que le nuple définit par "liste-de-champs".

Exemples : soit la relation : approvisionnement (nfoor, nart, nproj, date-ap, qte)

a) soit un extrait d'un programme

```
begin
    update (12, 10, 3, 19800104, 35) in approvisionnement
end
```

Dans ce cas le nuple de la relation approvisionnement dont l'identifiant clé est composé par (12, 10, 3, 19800104) aura la valeur de l'attribut qte changée pour 35.

b) soit un extrait de programme

```
var ndate : date : nqte : real ; nom-nel : char ;
begin
    ndate:= 19800104 ; nqte:=35 ; nom-nel:= 'approvisionnement' ;
    update (12, 10,3, ndate, nqte) in nom-nel
end ;
```

Dans cet exemple on a la mise-à-jour du nuple de la relation désignée par nom-nel (ici, approvisionnement) dont l'identéfiant clé est (12,10,3, ndate).

I.3.3.2.1.3 Elimination

Il a la structure de la forme :

delete (liste-de-champs) from nom-de-relation

où liste-de-champs : identifiée soit par des variables qui contiennent les valeurs du nuple qu'on veut éliminer soit par des valeurs

nom-de-relation : représenté soit par le nom d'une relation su SI soit par une nouvelle variable.

L'exécution d'un"delete" conduit à l'élimination du nuple de la relation "nom-de-relation" du SI qui a la même valeur pour l'identifiant clé que le nuple définit par "liste-de-champs".

Exemples : soit la relation : approvisionnement (nfour, nart, nprof, date-ap, qte)

a) soit un extrait de programme

```
begin
      delete (12, 10, 3, 19800104, 35) from approvisionnement
```

le nuple de la relation approvisionnement dont l'identifiant clé est composé par (12,10, 3, 19800104) est éliminé.

b) soit un extrait de programme

```
var ndate : date ; nqte : real ; nom-nel : char ;
begin
   ndate:=19800104 ; nqte:=35 ; nom-nel:= 'approvisionnement' ;
   delete (12, 10,3, ndate, nqte) from nom-nel
end ;
```

Dans cet exemple on supprime le nuple de la relation désignée par nom-nel (ici, approvisionnement) dont l'identifiant clé est (12, 10, 3, ndate).

I.3.3.2.2. Utilisation des procédures et fonctions

Outre les règles concernant la définition et l'emploi des procédures et fonctions en PASCAL, on introduit les possibilités suivantes :

a) utilisation d'un nom de relation comme paramètre formel

On a considéré que cette façon de faire était plus simple que de passer en paramètre formel la liste des attributs de la relation.

Exemple : considérons la relation approvisionnement (nfour, nart, nproj, date-ap, poids) et un extrait de description.

```
procedure calcul (approvisionnement ; var total : real);
   begin
   total:=0 ;
   :
   begin
        total:= approvisionnement.poids *1.2
   end
end ;
```

Cette procédure a en entrée un tuple dont la structure est la même que celle des tuples de la relation approvisionnement.

b) utilisation d'une variable, de type relation, comme paramètre formel

Il est nécessaire dans ce cas de déclarer la variable et son type. Cela implique que dans le corps de la procédure, lorsqu'on veut faire référence à un attribut de la relation on utilise la notation : <variable.> <attribut relation>. On verra l'utilité de cette convention dans le chapitre III.

Exemple : considérons un extrait de description portant sur la relation : approvisionnement (nfour, nart, nproj, date-ap, poids)

```
type var-relation=relation  vnfour : integer ; vnart : integer ; vnprof :
integer ; vdate-ap : date ; vpoids : real end ;

var id-relation : var-relation ; total-poids : real ;

/ * introduction de la procédure cumul * /

procedure cumul (nom-relation : var-relation ; var total : real) ;
   begin
          total :=0 ;
          foreach prod in nom-relation do
          begin
              total:= total + prod.vpoids
          end end end ;
```

/ * cette procédure calcule le total des valeurs de l'attribut vpoids des nuples de la relation désignée par nom-relation * /

/ * introduction du début du corps du programme * /

```
begin
        id-relation : <(x) : approvisionnement < x > ;
        cumul (id-relation, total-poids)
```

/ * dans ce cas la procédure cumul calcule le poids total de tous les produits déjà fournis à tous les projets par tous les fournisseurs et cette valeur est passée comme variable par total-poids * /

c) utilisation d'une variable, dont la valeur est un identificateur de procédure, comme identificateur de procédure : cette construction permet d'exprimer l'appel d'une procédure (ou d'une fonction) dont on ne connaît pas à l'avance l'identificateur. On en verra l'utilité dans le chapitre III. Une conséquence de cette construction est qu'on considère que toute procédure (ou fonction) appelée est définie, même si cette définition n'est pas faite dans le texte.

Exemple : considérons un extrait d'un texte d'une opération de la machine abstraite. Nous n'introduisons que les éléments nécessaires à la compréhension de l'exemple.

```
var cod-pred-init-fin : relation identifiant-cevper  : char ;
                                 date-cevpred        : date ;
                                 cod-penit           : char ;
                                 cod-pfin            : char ;
                                 desig-id-cev        : char end ;

    val-att-cob-precedant : relation p-val-at : char end ;
    val-att-cob           : relation c-val-at : char end ;
...
    procédure identification-évènement
*/ cette procédure lance l'évaluation des prédicats initial et final que caractérisent un évènement */
    begin
      foreach predicat in cod-pred-init-fin until test=true do
```

*/ la variable cod-pred-init-fin a été affectée par une autre procédure */

if predicat.cod-pinit (val-att-cob-precedant)= true then

*/ predicat-cod-pénit fournit le nom de la procédure qui évalue le prédicat initial */

if predicat.cod-pfin (val-att-cob) = true then

* predicat-cod-pfin fournit le nom de la procédure qui évalue le prédicat final */

begin teste := true ; ....

I.3.3.2.3. Fonctions relationnelles pré-définies

Ce sont des fonctions standard qui s'appliquent à des relations ou à des variables de type relation et qu'on retrouve dans la plupart des langages relationnels (DAT 77)(PIR 76).

Nous n'avons retenu que celles qui sont le plus souvent utilisées car on peut exprimer d'autres fonctions au moyen de ISDEL.

Toutes les fonctions présentées sont de la forme :
nom-de-fonction (expression-relationnelle)

où expression-relationnelle peut être :

. un nom de relation
. une variable de type relation
. une expression relationnelle ISDEL
. un attribut indexé d'une relation

TOTAL (expression relationnelle)

TOTAL est une fonction d'addition des valeurs de l'attribut défini par "expression relationnelle".

TOTAL cumule toutes les valeurs obtenues par l'évaluation de "expression relationnelle" avant l'élimination des doubles.

On rappelle qu'une relation est un ensemble de tuples. Il n'y a pas de répétition d'éléments.

Exemples : considérons la Relation approvisionnement (nfour, nart, nproj, date-ap, poids)

TOTAL((x. poids) : approvisionnement <x> AND x. nproj = 30)
donne le poids total de tous les articles déjà fournis par tous les fournisseurs au projet 30.

COUNT (expression relationnelle)

COUNT est une fonction de comptage d'occurences de la relation définie par "expression relationnelle". On compte les occurences qui satisfont "expression relationnelle" après élimination des doubles (car il s'agit d'une relation et non pas d'une liste de valeurs comme pour le cas de TOTAL)

Exemple : considérons la relation : approvisionnement (nfour, nart, nproj, date-ap, poids)

COUNT ((x-nfour) : approvisionnement <x> AND x. nproj = 30)
donne le nombre de fournisseurs qui ont déjà approvisionné le projet 30.

AVERAGE (expression relationnelle)

AVERAGE calcule la moyenne arithmétique d'une liste de valeurs obtenue par l'évolution de "expression relationnelle".

Exemple : AVERAGE ((x.poids) : approvisionnement < x > AND x.nproj = 30)
conne le poids moyen des articles déjà fournis au projet 30

MAX (expression relationnelle

MAX retrouve la plus grande valeur d'un attribut d'une relation défini par "expression relationnelle"

Exemple MAX ((x.poids) : approvisionnement <x> AND x.nproj = 30) donne le poids de l'article le plus lourd déjà fourni au projet 30.

MIN (expression relationnelle)

MIN retrouve la plus petite valeur d'un attribut d'une relation défini par "expression relationnelle"

Exemple : MIN ((x.poids) : approvisionnement < x > AND x.nproj = 30
donne le poids de l'article le plus léger déjà fourni au projet 30.

I.3.3.2.4. Ordre

Les expressions relationnelles à utiliser ne présupposent aucun ordre dans la prise en compte des tuples d'une relation. Dans certains cas, cela peut être nécessaire d'ordonner la manipulation des tuples et pour cette raison on introduit la notion de relation d'ordre, qui est exprimée par la clause :

ordre :: = {<sens> <attribut>}*

où <sens > est égal à :
- UP pour croissant
- DOWN pour décroissant

Une expression relationnelle ISDEL a donc la forme générale :
(ti, te, ..., tn) : u [AND ordre]

Exemples : considérons les relations

article (nart, couleur, poids)
approvisionnement (nfour, nart, nproj, date-ap, quantité)

a) (x.nart, x.poids) : article < x > AND UP x. poids
donne en ordre croissant de poids tous les articles et leur poids.

b) (x. nfour, x.nart) : approvisionnement < x > AND UP nfour DOWN nart donne en ordre croissant de fournisseurs la liste en ordre décroissant des articles qu'ils ont déjà fourni aux projets.

I.3.4. Description ISDEL d'une structure conceptuelle

Nous présentons les différentes clauses qui permettent la description des relations qui composent la structure conceptuelle.

I.3.4.1 Clauses introduisant les types des relations

La structure conceptuelle est composée par des relations dont le type traduit quelle catégorie de phénomène de la réalité elle représente.

Plusieurs relations du même type sont nécessaires pour décrire l'intégralité du phénomène, chacune décrit un aspect temporel particulier du phénomène.

On utilise pour décrire ces clauses les conventions suivantes :

```
description ::= IS-TYPE is-type-ident spécification-relation END
is-type-ident::=COBJECT MODE COB | COPERATION MODE cop | CEVENT MODE cev
cop ::= COPPER | COPTEXTE | COPMODIFIE
cev ::= CEVPER | CEVCONSTATE | CEVPRED | CEVDECLENCHE | CEVDECEFF
```

1.3.4.1.1. IS-TYPE

La clause IS-TYPE d'une relation indique quel est la catégorie de phénomène qu'elle décrit. On a trois types de IS-TYPE :

COBJECT : indique que la relation décrit un aspect d'un c-objet de la structure conceptuelle

COPERATION : indique que la relation décrit un aspect d'une c-opération de la structure conceptuelle.

CEVENT : indique que la relation décrit un aspect d'un c-évènement de la struture conceptuelle

I.3.4.1.2. MODE

La clause MODE complète la clause IS-TYPE en indiquant quel est l'aspect de la catégorie de phénomène que la relation décrit. Un MODE peut donc être considéré comme un sous-type de la relation et chaque MODE est associé à un seul IS-TYPE.

MODE COB : il est associé au IS-TYPE OBJECT. Une relation de ce mode décrit un aspect temporel particulier d'une classe d'objets du monde réel.

Exemples : extraits de II.3.5.

IS-TYPE COBJECT MODE COB produit-per (npro, date-cre-pro)
IS-TYPE COBJECT MODE COB produit-prix (npro, date-prix, prix)

MODE COPPER : il est associé au IS-TYPE COPERATION et il indique que la relation décrit les propriétés, d'une c-opération qui sont variantes au cours du temps. A chaque c-opération de la structure conceptuelle correspond une et une seule relation de ce mode.

Exemples :

IS-TYPE COPERATION MODE COPPER aclicodif-per (nop9, date-op9-per)
IS-TYPE COPERATION MODE COPPER majsto-per (nop 8, date-op8-per)

MODE COPTEXTE : il est associé au IS-TYPE COPERATION et il indique que la relation introduit le texte de la c-opération. Toute relation de ce mode est à schéma de relation constante. A chaque c-opération de la structure conceptuelle ne correspond qu'une relation de ce mode.

Exemples :

IS-TYPE COPERATION MODE COPTEXTE aclécodif-texte(nop9,date-nop9tex, texte-op9)
IS-TYPE COPERATION MODE COPTEXTE majsto-texte (nop8, date-nop8 tex, texte-op8)

MODE COPMODIFIE : il est associé au IS-TYPE COPERATION et il indique quel est le c-objet modifié par la c-opération. Toute relation de ce mode est à schéma de relation constant. A chaque c-opération de la structure conceptuelle ne correspond qu'une relation de ce mode. Il représente l'association MODIFIE.

Exemples :

IS-TYPE COPERATION MODE COPMODIFIE aclicodif-mod (nop9-texte, date-nop9-modifie, nlicoaliv
IS-TYPE COPERATION MODE COPMODIFIE majsto-mod (nop8-texte, date-nop8-modifie, nprostock)

MODE CEVPER : il est associé au IS-TYPE CEVENT et il indique que la relation décrit les propriétés d'un c-évènements au cours du temps. A chaque c-évènement qui sont de la structure conceptuelle correspond une et une seule relation de ce mode.

Exemples :

IS-TYPE CEVENT MODE CEVPER ev1-per (nev1, date-cre-ev1)
IS-TYPE CEVENT MODE CEVPER ev4-per (nev4, date-cre-ev4)

MODE CEVPRED : il est associé au IS-TYPE CEVENT et il indique que la relation introduit les textes de prédicats initial et final d'un c-évènement. Toute relation de ce mode est à schéma de relation constant. A chaque c-évènement de la structure conceptuelle ne correspond qu'une relation de ce mode.

Exemples :

IS-TYPE CEVENT MODE CEVPRED ev1-pred (nev1, date-ev1-pred, pinit-ev1, pfin-ev1)
IS-TYPE CEVENT MODE CEVPRED ev4-pred (nev4, date-ev4-pred,pinit-ev4, pfin-ev4)

MODE CEVCONSTATE : il est associé au IS-TYPE CEVENT et il indique quel est le c-objet dont le changement d'état est constaté comme un c-évènement. Toute relation de ce mode est à schéma de relation constant. A chaque c-évènement de la structure conceptuelle correspond une et une seule relation de ce mode. Il décrit l'association CONSTATE.

Exemples :

IS-TYPE CEVENT MODE CEVCONSTATE ev1-constate (nev1-pred,date-ev1-constate, nlicoin).
IS-TYPE CEVENT MODE CEVCONSTATE ev4-constate (nev4-pre, date-ev4-constate nprostock)

MODE CEVDECLENCHE : il est associé au IS-TYPE CEVENT et il indique quelles sont la condition de déclenchement et le facteur de déclenchement d'une c-opération associée à un c-évènement. Toute relation de ce mode est à schéma de relation constant. A chaque c-évènement de la structure conceptuelle, il y a autant de relations de ce mode qu'il y a de c-opération associées à ces c-évènements. Ce mode représente les propriétés de l'association DECLENCHE.

Exemples :

IS-TYPE CEVENT MODE CDECLENCHE ev1-dec (nev1, nop1, date-ev10p1-dec, cond-ev1-op1, facteur-ev1-op1).
IS-TYPE CEVENT MODE CDECLENCHE ev4-dec (nev4, nop8, date-ev4 op8-dec; cond-ev4-op8, facteur-ev4-op8)

MODE CEVDECEFF : il est associé au IS-TYPE CEVENT et il indique quelle est la c-opération qui a été déclenchée par un c-évènement. Toute relation de ce mode est à schéma de relation constante. A chaque évènement de la structure conceptuelle il y a autant de relations de ce mode qu'il y a de c-opérations associées à ce c-évènement. Ce mode représente le déclenchement effectif d'une c-opération par un c-évènement.

Exemples :

IS-TYPE CEVENT MODE CEVDECEFF (nev1-constate, nop1-modifie)
IS-TYPE CEVENT MODE CEVDECEFF (nev4-constate, nop8-modifie)

I.3.4.2. Définition des clauses qu'introduisent les attributs d'une relation

I.3.4.2.1. DOMAIN

Ce sont les clauses que l'on utilise pour décrire les relations et les domaines associés à leurs attributs.

A chaque attribut d'une relation correspond un domaine qui caractérise l'ensemble des valeurs que l'attribut peut prendre.

On dispose les domaines pré-définis suivants :

CHAR : ensemble des listes de caractères alphanumériques
INTEGER : ensemble des entrées
REAL : ensemble des réels
BOOLEAN : ensemble des valeurs true et false
DATE : ensemble des dates

On peut aussi décrire le domaine d'un attribut par la clause SAME OF. Elle a la forme : SAME OF nom-de-relation. Cette clause indique que le domaine de l'attribut auquel elle s'applique est le même que celui qu'on a défini dans la relation désignée pour "nom-de-relation".

On utilise la syntaxe suivante pour décrire ces clauses.

```
Spécification-relation:= spec-rel/spec-rel ; spécification-relation
spec-rel::= nom-relation (liste-nom-champs) spec-domaine [spec-assertion]
nom-relation ::= indentificateur
liste-nom-champs::= nom-champs/nom-champs ; liste-nom-champs
nom-champs ::= identificateur/variable
spec-domaine ::= DOMAIN spec-domaine-dif
spec-domaine-dif ::= domaine-dif/domaine-dif ; spec-domaine-dif
domaine-dif ::= nom-champs : domaine-exp/nom-champs : SAME OF nom-relation
domaine-exp ::= CHAR [(entier-sans-signe)]
                INTEGER [(entier-sans-signe)]
                REAL [ (précision)]
                DATE [(entier-sans-signe)]
                BOOLEAN
précision ::= entier-sans-signe / entier-sans-signe, entier-sans-signe
```

Nous rappelons que les symboles entre [ ] sont optionnels.

Exemples :

a) IS-TYPE COBJECT MODE COBVAR ligne-com-ihit (nco, npro, date-leçoin, qte, date-liv, prix)

```
DOMAIN nco : SAME OF ligne-de-commande-per ;
       npro : SAME OF ligne-de-commande-per ;
       date-lecoin : date (8) ;
       qte  : real (6,2) ;
       date-liv : date (8) ;
       prix : real (6,2) ;
```

Dans cet exemple nous indiquons que :

nco : a été défini dans la relation ligne-de-commande-per

npro : a été défini dans la relation ligne-de-commande-per

date-lecoin : a huit positions, c'est-à-dire, elle comprend l'année, le mois, le jour

qte : est une valeur de type réel ayant au plus six positions et avec deux positions décimales.

date-liv : à huit positions

prix : est une valeur de type réel ayant au plus six positions et avec deux positions décimales

b) IS-TYPE COPERATION MODE COPTEXTE majsto-texte (nop8, date-cre-nop8, texte-op8)

```
DOMAIN nop 8 : char ;
       date-cre-nop 8 : date (8) ;
       texte-op8 : char ;
```

Dans cet exemple, nous indiquons que :

nop8 : est défini par une chaîne de caractères

date-cre-nop8 : comprend l'année, le mois, le jour

texte-op8 : est défini comme une chaîne de caractères. Nous rappelons qu'il s'agit de l'indicatif du texte de l'opérateur

I.3.4.2.1.1. DATE

Il est défini par rapport à un calendrier dont on donne la définition suivante :

. Soit D l'ensemble des ensembles Di ($1 \leq i \leq n$) finis totalement ordonné d'entiers.

. Soit M un ensemble de mots tel que card (M) = n. A chaque mot $i \in M$ on associe une unité de temps distincte.

. Soit μ une application bijective définie de D sur M

$$\mu : D \longrightarrow M$$
$$Di \longmapsto \mu(Di)$$

. Soit $Vi = \{(di, \mu(Di)) \text{ tel que } di \in Di\}$. On pourrait dire $Vi = Di \times \{\mu(Di)\}$

. Soit $V1 \times Va \times \ldots \times Vn$ le produit cantésian des Vi ($1 \leq i \leq n$)

On définit un calendrier comme un sous-ensemble de $V1xV2x..xVn$ tel que :

. $\forall i,j$ si $i \neq j$ alors $Vi \cap Vj = \emptyset$

. $\forall i,j$ ($1 \leq i,j \leq n$) si $1 < j$ alors
$(\forall (di, \mu(Di)) \in Vi) \wedge (\forall (dj, \mu(Dj)) \in Vj)$ alors
$\exists k \in N^{*}$ tel que $dj = k * di$

. soit $Pi : V1 \times V2x \ldots \times Vn \longrightarrow Vi, Vi+1, \ldots, Vn$
$(v1,v2,\ldots,vn) \longmapsto Pi(v1,v2,\ldots,vn) = (vi,vi+1,\ldots,vn)$
la projection d'ordre i sur $V1 \times V2x,\ldots xVn$

Une date est exprimée par une des projections possibles sur le calendrier

Le domaine date définit l'ensemble des parties des projections d'un calendrier.

La notation retenue pour les valeurs du domaine date est la suivante : dn dn-1...di tels que $di \in Dn$, $dn-1 \in Dn1, \ldots$, $du \in Di$. On laisse donc implicites des $\mu(Di)$. On dit que la date dn dn-1...di est d'ordre i.

Opérateurs et fonctions pré-définis sur le domaine date

Comparaison de deux dates :

a) si les deux dates sont du même ordre, la comparaison est analogue à celle des entiers

b) si les deux dates sont d'ordre différent la comparaison s'effectue selon la règle suivante :

. soit vi la date de plus petit ordre et soit i son ordre
. soit vj la date de plus grand ordre et soit j son ordre
. la comparaison est effectuée entre vi et la projection d'ordre i de vj

Par exemple : vi = (1980010523)
vj = (19800105230402)

la comparaison est faite entre vi et Pi (vj), où Pi (vj) = (1980010523)

c) les opérateurs de comparaison sont : = , >, < ,=> , =< , ≠ . L'égalité (=) n'est définie que si les deux dates sont du même ordre.

Décomposition d'une date : soit $\mathcal{E}$ l'ensemble des applications injectibles $\mathcal{E}j$ $(1 \leq j \leq n)$ applicables à un calendrier $\mathcal{C}$, tels que :

$\mathcal{E}j : \mathcal{C} \longrightarrow N^*$ ou $\mathcal{E}j = Vi \times Vi+1 \times \ldots \times Vn \longrightarrow N^*$ pour $1 \leq i \leq n$ et $i \leq j \leq n$

$(v1, vi+1, \ldots, vn) \longmapsto \mathcal{E}j(vi, vi+1, \ldots, vn) = dj$

On utilise la valeur de $\mu(Dj)$ pour remplacer $\mathcal{E}j$ et définir l'identificateur de l'application.

Exemple : month (19800504) = 05
day (19800504) = 04

Création d'une date est décrite par la fonction time

time : $N^* \longrightarrow \mathcal{C}$
$n \longmapsto$ time (n) = (vn, vn-1,..., vi)

Exemple : time (6) = (198004)
time (2) = indéfini

Composition d'une date est décrite par la fonction concat

$$concat : N^* \times N^* \times \ldots \times N^* \longrightarrow \mathbb{C}$$

$$(n1,n2,\ldots,nm) \longmapsto concat\ (n1,n2,\ldots,nm) = (vn,vn-1,\ldots,vn-m+1)$$

Exemple : concat (1980,05,04) = (19800504)
concat (1980,03,02,05) = (1980030205)

I.3.4.3. Définition des contraintes d'intégrité associées aux relations

I.3.4.3.1. Classification des contraintes d'intégrité

Nous nous sommes inspirés de (BEN 79) et de (DAT 77) pour établir cette classification.

1. Contraintes sur un attribut d'une relation

   - Existence : . Valeurs prises par l'attribut définies par - extension
     - algorithme
     . Valeurs prises par l'attribut appartiennent à un interval
     . Format : si on considère que chaque valeur est une chaine de caractères on peut avoir des sous-chaines obligatoires.

2. Contraintes entre attributs d'une relation

   - Existence : . Valeurs prises par un attribut sont définies par rapport aux valeurs des autres attributs - par un algorithme
     - par un prédicat
     . Format
     . Clé
     . Dépendance fonctionnelle permanente
   - Cardinalité de la relation

3. Contraintes entre attributs de plusieurs relations

   - Existence : . Valeurs prises par l'attribut d'une relation sont définies par les attributs d'autre relation - par un algorithme
     - par extension
     - par un prédicat
   - Cardinalité

On remarque que ces contraintes sont statiques dans le sens qu'elles définissent des propriétés que les valeurs des attributs doivent respecter pour être introduites dans le SI.

Les contraintes liées à la dynamique sont exprimées par les textes des opérations, prédicats, conditions et facteurs de déclenchement. On remarque aussi que dans l'évaluation de ces textes la notion d'ordre d'occurence joue un rôle important et en général on aura besoin de la dernière occurence chronologique associée à une clé logique d'une relation (on la désignera par CURRENT) ou de l'occurence qui précède une occurence donnée (on l'indique par l'opérateur de comparaison PREDECESSOR).

I.3.4.3.2. Expression des contraintes d'intégrité en ISDEL

Les contraintes d'intégrité sont introduites selon la syntaxe suivante :

```
spec-rel::= nom-relation (liste-nom-champs)spec-domaine [spec-assertion]
spec-assertion::=ASSERT ident-spec-list
ident-spec-list::= spec-list-elem / spec-list-elem, ident-spec-list
spec-list-elem::=identificateur:nature-assert /idendificateur:LIKE identi-
                 ficateur
nature-asset::=assert-elementaire/assert-complexe:specification-texte-assert
```

La clause ASSERT indique que l'on fournit la liste de contraintes d'intégrité associées à la relation.

Ces contraintes sont divisées en deux catégories : élémentaires et complexes.

I.3.4.3.2.1. Contraintes d'intégrité élémentaires

Elles sont celles exprimées par des clauses élémentaires ni n'exigent pas d'expressions ISDEL mais seulement l'usage de mots clés.

Leur syntaxe est la suivante :

```
Assert-élémentaire::= liste-nom-champs COMPOSKEY [nom-champs]
                      nom-champs assert-desc
assert-desc ::= FINAL/INITIAL/ACTUAL/KEY/DATECRE
```

La clause ACTUAL sera développée dans II.3.4.3.2.2.4.

I.3.4.3.2.1.1. COMPOSKEY

La clause COMPOSKEY est utilisée pour indiquer les attributs qui composent l'identifiant clé d'une relation. Elle peut éventuellement indiquer le nom-de-champs qu'on donne à cet identifiant clé composé et qu'on appelle : nom de l'identifiant clé composé.

Il est donc possible d'utiliser indifféremment le nom de l'identifiant clé composé ou la liste de nom-de-champ au sein d'une expression relationnelle ISDEL.

Si on veut connaître la valeur de l'identifiant clé d'un tuple d'une relation on utilise la fonction idcle. Elle a la forme suivante :

idcle (nom-de-relation, tuple)

où nom-de-relation est le nom de la relation en question

tuple est la variable qui a comme valeur un tuple de la relation

Exemple : basé sur I.3.5.

```
IS-TYPE COBJECT MODE COB
ligne-com-init (nco, npro, date-lecoin, qte, date-liv, prix)
DOMAIN nco : SAME OF cclasse -ligne-commande ;

ASSERT
licoinit I = nco,npro, date-licoin COMPOSKEY nlicoinit ;
```

Dans cet exemple nlicoinit est le nom de l'identifiant clé composé. On l'utilise, par exemple, dans la relation :

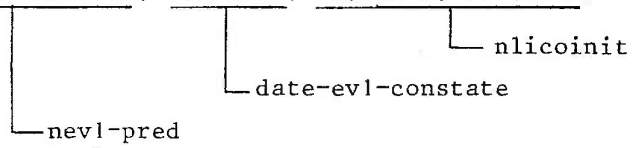
IS-TYPE CEVENEMENT MODE CEVCONSTATE

evl-constate (nevl-pred, date-evl-constate, nlicoinit)

où nevl-pred lui aussi est le nom d'un identifiant clé composé

Si on veut insérer un nuple dans la relation evl-constate, on le ferait par exemple pour :

insert (evl01, 19800104, 19800105, 30, 127A, 19800302)



I.3.4.3.2.1.2. KEY

La clause KEY indique quel attribut est l'identifiant clé (dans ce cas simple) de la relation.

Exemple : basé sur I.3.5.

```
IS-TYPE COPERATION MODE COPPER
   dif-nouv-lico (nopll, date-difnoulico)
DOMAIN  nopll : char ;
        date-difnoulco : date (8) ;
ASSERT
   difnoulicol : nopll KEY ;
```

I.3.4.3.2.1.3. INITIAL, FINAL

Ce sont des contraintes d'intégrité qui indiquent quel attribut correspond au prédicat initial d'un évènement (INITIAL) et quel attribut correspond au prédicat final d'un c-évènement (FINAL).

Exemple : basé sur I.3.5.

```
IS-TYPE CEVENT MODE CEVPNED
   evl-pred (nevl, date-evl-pred, pinit-evl, pfin-evl)
DOMAIN nevl            : SAME OF evl-per ;
       date-evl-pred   : date (8) ;
       pinit-evl       : char ;
       pfin-evl        : char ;
```

```
ASSERT
   evl-predl : nevl, date-evl-pred COMPOSKEY nevl-pred ;
   evl-pred2 : pinit-evl : INITIAL ;
   evl-pred3 : pfin-evl : FINAL ;
```

I.3.4.3.2.1.4. DATECRE

La clause DATECRE indique quel attribut de type date de la relation est utilisé pour exprimer l'évolution dans le temps des occurences de la relation par leur date de création.

Cette clause permet l'évaluation d'une fonction pré-définies de ISDEL (datecle) aussi bien que l'utilisation des clauses CURNENT et PREDECESSOR.

La fonction datecle a la forme suivante :

datecle (nom-de-relation, tuple)

où nom-de-relation est le nom de la relation à laquelle on applique la fonction.tuple (variable qui a comme valeur un tuple de la relation en question).

La fonction datecle donne comme résultat la valeur de l'attribut du tuple auquel on a appliqué la clause DATECLE. En d'autres termes, cette fonction fournit la date de création d'un tuple.

Exemples : En reprenant les exemples donnés pour les autres clauses on en peut rajouter les contraintes d'intégrité suivantes :

```
a) ...
   ligne-com-init (nco, npro, date-licoin, qte, date-liv, prix)
   ...
   licoinit 2 : date-licoin DATECRE ;
   ...

b) ...
   dif-nouv-lico (nopll, date-difnoulico)
   ...
   dif-noulico 2 : date-difnoulico DATECRE
```

c) ...

ev1-pred (nev1, date-ev1-pred, pinit-ev1, pfin-ev1)

...

ev1-pred4 : date-ev1-pred : DATECRE ;

Il n'est pas nécessaire de spécifier la clause DATECRE s'il n'y a qu'un attribut de type date dans la relation décrite. Cet attribut est considéré comme celui qui exprime la date de création de l'occurence de la relation.

I.3.4.3.2.2. Contraintes d'intégrité complexes

Elles expriment à l'aide d'expressions ISDEL. Ce sont les contraintes d'intégrité qui définissent les propriétés des attributs des relations et les textes des prédicats, opérations et des conditions et facteurs de déclenchement.

Elles ont la syntaxe suivante (développée suite à V.3.4.2.)

```
nature-assert::=assert-elementaire/assert-complexe:spécification-texte-assert
assert-complexe::=nom-de-champ  texte-identif /
                  CONSTRAINT (liste-variables-specif)
texte-identif::=texte-id (liste-variables-specif)
texte-id ::= OPERATION / FACTOR / PREDICATE / CONDITION
Specification-texte-assert::=specification-procedurale /modele-de-predicat /
                             LIKE indentificateur
liste-variables-specif ::= variable-specif /
                           variable-specif ; liste-variables-specif
variables-specif::=ident-relation/ident-relation-det/identificateur :
                   identificateur-type
ident-relation-dit::=ident-relation . nom-de-champ . / nom-champ
ident-relation ::= nom-relation / nom-relation-var
nom-relation-var::=nom-relation <variable-relation> / variable-relation
variable-relation ::= identificateur
```

I.3.4.3.2.2.1. CONSTRAINT : pour les attributs

Cette clause a la syntaxe suivante

ident : CONSTRAINT (liste-variables-specif) : specification-texte-assert

où liste-variables-specif : identifie les attributs sur lesquels s'applique la contrainte d'intégrité

specification-texte-assert : identifie le texte de la contrainte d'intégrité

ident : est le nom que l'on donne à la contrainte d'intégrité. Il est assimilé à une fonction booléenne définie par "spécification-texte-assert" et ayant comme paramètres formels "liste-variables-specif".

Exemples : extraits de I.3.5.

a) IS-TYPE COBJECT MODE COB
produit-prix (npro, date-prodprix, prod-prix)

```
DOMAIN npro ...
      ...
ASSERT
   propri1 : npro, date-prodprix COMPOSKEY nproprix ;
   propri2 : CONSTRAINT (prod-prix) :
             begin
                 if prod-prix = > 0 then propri2 :=true
                 else propri 2 := false end
             end ;
```

La contrainte d'intégrité "propri 2" est satisfaite si le prix du produit est supérieur ou égal à zéro.

b) IS-TYPE COBJECT MODE COB
ligne-com-nouvelle (nco,npro,date-nouv, ndélai,nprix)

```
DOMAIN nco ...
      ...
```

```
ASSERT
   liconouv1 : nco, npro, date-nouv COMPOSKEY nliconouv ;
   liconouv2 : CONSTRAINT (npro, nprix) :
     type rech-prix = relation proprix : real end ;
     var test-prix : rech-prix ;
     begin
       test-prix : <(x.prod-prix) : produit-prix < x > AND x.npro=npro
                                    AND x.nprostock CURRENT
       foreach element in test-prix do
          if element . proprix = nprix then  liconouv2:= true
          ilse liconouv2 := false
       end end
     end ;
```

La contrainte d'intégrité "liconouv2" est satisfaite si le prix (nprix) du produit (npro) de la ligne de commande nouvelle présentée est égal au prix actuel du produit indiqué dans la relation produit-prix. On remarque que test-prix n'a qu'un nuple comme valeur.

I.3.4.3.2.2.2. PREDICATE : pour les prédicats des c-événements

Cette clause a la syntaxe suivante :

ident : nom-de-champ PREDICATE (ident-relation) : specification-texte-assert

où ident-relation : identifie l'occurence de la relation de type c-objet qui est liéee à son changement d'état

nom-de-champ : identifie l'attribut qui désigne le prédicat initial ou le prédicat final associés au c-évènement

specification-texte-assert ; identifie le texte du prédicat

ident : est le nom que l'on donne à la contrainte d'intégrité

On considère nom-de-champ comme le nom d'une fonction booléenne ayant comme paramètre formel une occurence du c-objet qui a changé d'état.

S'il s'agit du prédicat final l'occurence en question est celle qui a provoqué le changement d'état et s'il s'agit du prédicat initial l'occurence en question est la dernière existante avant que le changement d'état se produise.

Exemples : extraits de I.3.5.

```
a) IS-TYPE CEVENT MODE CEVPNED
      ev1-pred (nev1, date-ev1-pred, pinit-ev1, pfin-ev1)
   DOMAIN nev1 ...
          ...
ASSERT ev1-pred1 : nev1, date-ev1-pred COMPOSKEY nev1-pred ;
   ev1-pred2 : pinit-ev1     : INITIAL ;
   ev1-pred3 : pfin-ev1      : FINAL ;
   ev1-pred4 : date-ev1-pred: DATECLE ;
   ev1-pred5 : pinit-ev1 PREDICATE (ligne-com-init) :
      begin pinit-ev1 := true end ;
   ev1-pred6 : pfin-ev1 PREDICATE (ligne-com-init) :
      begin pfin-ev1 :=true end ;
```

Dans cet exemple on considère que la création d'une occurence de la relation ligne-com-init est toujours un évènement du c-évènement ev1. En conséquence pinit-ev1 et pfin-ev1 sont toujours vrais.

```
b) dans ce deuxième exemple l'évènement est associé au c-objet décrit par :
produit-stock (npro, date-prostock, qte-stock)
   IS-TYPE CEVENT MODE CEVPRED
      ev2-pred (nev2, date-ev2-pre, pinit-ev2, pfin-ev2)
   DOMAIN nev2 ...
          ...
   ASSERT ev2-pred1 : nev2, date-ev2-pred COMPOSKEY nev2-pred
      ev2-pred2 : pinit-ev2   : INITIAL ;
      ev2-pred3 : pfin-ev2    : FINAL ;
      ev2-pred4 : date-ev2-pred : DATECLE ;
      ev2-pred5 : pinit-ev2 PREDICATE (produit-stock) :
```

```
            begin
      if produit-stock.qte-stock >0 then pinit-ev2 := true
            else pinit-ev2 := false
            end
         end
/* on vérifie si la quantité en stock avant le changement d'état était positive*/
/* on remarque que produit-stock n'a qu'une occurence */
   ev2-pred6 : pfin-ev2 PREDICATE (produit-stock) :
               var quantité-seuil : relation val-seuil : real end ;
               begin
                 foreach prod in produit-stock do
/* on rappelle qu'il n'y a qu'une occurence associée à prod */
      quantité-seuil : < (x.seuil) : produit-désignation < x >AND
      x.npro = produit-stock.npro AND x. nprodes CURRENT ;
/* on dispose en quantité-seuil de la valeur du seuil de sécurité du stock */
                foreach seuil-cale in quantité-seuil do
/* il n'y a qu'une occurence en seuil-cale */
                 if prod.qte-stock <seuil-cale. val-seuil
                 then pfin-ev2 := true
                 else pfin-ev2 := false
                 end
               end
          end
       end ;
/* le prédicat pfin-ev2 est vrai si la quantité en stock est inférieure au
   seuil de sécurité */
```

I.3.4.3.2.2.3. CONDITION : pour les conditions de déclenchement des c-opérations

Cette clause a la syntaxe suivante :

ident : nom-de-champ CONDITION (ident-relation) : spécification-texte-assert

où ident-relation : identifie l'occurence de l'objet qui est à l'origine du déclenchement.

nom-de-champ : identifie l'attribut que désigne la condition de déclenchement.

specification-texte-assert : identifie le texte de la condition de déclenchement

ident : c'est le nom que l'on donne à la contrainte d'intégrité.

Le texte d'une condition de déclenchement décrit l'état du SI qui est nécessaire au déclenchement de l'opération.

Ce texte définit une fonction booléenne, dont la désignation est "nom-de-champ ", ayant comme paramètre en entrée l'occurence de l'objet dont le changement d'état constitue l'évènement.

Exemples : extraits de I.3.5.

a) dans cet exemple on utilise la relation produit-stock (npro, date-prostock, qtestock)

```
IS-TYPE CEVENT MODE CEVDECLENCHE
   ev2-op6-prodst-dec (nev2, nop6, date-ev2-op6-dec, cond-ev2-op6,
                       facteur-ev2-op6)
DOMAIN nev2  : SAME OF ev2-per ;
       nop6  : SAME OF derepro-per ;
       date-ev2-op6-dec : date (8) ;
       cond-ev2-op6 : char ;
       facteur-ev2-op6 : char ;
ASSERT
ev2-op6-prod1 : nev2, nop6, date-ev2-op6-dec COMPOSKEY ev2-ip6-dec;
ev2-op6-prod2 : cond-ev2 op6 CONDITION (produit-stock) :
/* on rappelle que produit-stock n'a qu'une occurence */
   begin cond-ev2-op6 := true end ;
```

Dans cet exemple le déclenchement de l'opération OP6 est inconditionnel.

b) dans cet exemple on utilise en plus la relation :

```
ligne-com-enit  (nco, npro, date-licoin, qte, date-liv, prix)

IS-TYPE CEVENT MODE CEVDECLENCHE
   ev1-op2- licoin-dec (nev1, nop2, date-ev1-op2-dec, cond-ev1-op2,
                        facteur-ev1-op2)
DOMAIN nev1 : same of ev1-per ;
       nop2 : same of maleco-per ;
       date-ev1-op2-dec : date (8) ;
       cond-ev1-op2 : char ;
       facteur-ev1-op2 : char ;
ASSERT
   ev1-op4-li1 : nev1, nop2, date-ev1 op2-dec COMPOSKEY nev1 op2-dec ;
   ev1-op4-li2 : cond-ev1 op  CONDITION (ligne-com-init) :

/* on rappelle que ligne-com-init n'a qu'une occurence */

   var niveau-stock : relation val-stock : real end ;
   begin

/* il n'y a qu'une occurence associée à ligne-com-init */
   niveau-stock : < (x. qtestock) : produit-stock < x >
                  AND x. npro = ligne-com-init. npro
                  AND x. nprostock CURRENT ;

/* niveau-stock a comme valeur le 1-uple correspondant à la quantité en
   stock du produit */

   foreach val in niveau-stock do
        if val. val-stock  <ligne-com-init.qte then cond-v1-op2:=true ;
      else cond-ev1-op2 := false end
   end
end end ;

/* la condition de déclenchement est vraie si la quantité de la ligne de
   commande en question  est supérieure à la quantité disponible en stock
   (val.val-stock) en conséquence ont doit créer une occurence de ligne de
   commande différée */
```

I.3.4.3.2.2.4. FACTEUR : pour les facteurs de déclenchement

Cette clause a la syntaxe suivante :

ident : nom-de-champ FACTOR (ident-relation): specification-text-assert

où ident-relation : identifie l'occurence de l'objet qui est à l'origine du déclenchement

nom-de-champ : identifie l'attribut qui désigne le facteur de déclenchement

specification-text-assert : identifie le texte du facteur de déclenchement

ident : c'est le nom que l'on donne à la contrainte d'intégrité.

Cette contrainte d'intégrité décrit le texte du facteur de déclenchement. Ce texte est une procédure qui définit l'ensemble des tuples dont chaque élément est un paramètre du déclenchement.

Il y a autant de déclenchements qu'il y a de tuples définis par le facteur de déclenchement. L'opération s'exécute chaque fois sur un tuple différent.

La procédure qui évalue le facteur de déclenchement a comme paramètre en entrée l'occurence du c-objet qui a provoqué l'évènement.

Le texte du facteur de déclenchement affecte à "nom-de-champ " l'ensemble de tuples qu'elle évalue.

Exemples : extraits de I.3.5.

a) dans cet exemple on utilise la relation

produit-stock (npro, date-prostock, qtestock)

```
IS-TYPE CEVENT MODE CEVDECLENCHE
    ev1-op6-prodst-dec(nev2, nop6, date-ev2-op6-dec, cond-ev2-op6,
                       facteur-ev2-op6)
DOMAIN nev2 ...
       ...
       facteur-ev2-op6 : char ;
```

```
ASSERT
ev2-op6-prod-1 : nev2...
ev2-op6-prod-2 : cond-ev2-op6 ...
ev2-op6-prod-3 : facteur-ev2-op6 FACTOR (produit-stock) : ACTUAL ;
```

La clause ACTUAL est utilisée pour indiquer que le facteur de déclenchement est composé par l'occurence de "ident-relation" (dans le cas produit-stock) passée en paramètre. Il n'est donc pas nécessaire de décrire le texte du facteur de déclenchement.

b) dans cet ensemble on utilise les relations

```
ligne-com-a-liv (nco, npro, date- aliv , délai, qtealiv)
ligne-com-differée (nco, npro, date-licodif, qtedif)
produit-stock (npro, date-prostock, qtestock)

IS-TYPE CEVENT MODE CEVDECLENCHE
ev4-op9-prodst-dec (nev4, nop9, date-ev4-op9-dec, cond-ev4-op9,
                    facteur-ev4-op9)

DOMAIN nev4 ...
       ...
ASSERT
ev4-op9-pro1 : nev4,...
ev4-op9-pro2 : cond-ev4-op9 CONDITION ...
ev4-op9-pro3 : facteur-ev4-op9 FACTOR (produit-stock) :

/* on rappelle que produit-stock n'a qu'une occurence */
var dif : relation dif-nco : integer ; dif-npro : integer ; dif-qte :
                                real end ;
   cumul: real ;

begin
   dif :  <(y.nco, y.npro, y.qte)
            ligne-com-a-live <x> AND ligne-com-differee  <y> AND
            EXIST <y> (y.npro = produit-stock.npro AND

/* y est un tuple de la relation ligne-com-différée tel qu'il est même numéro
   de produit npro que l'occurence de produit-stock passée en paramètre */
```

```
      NOT EXIST < x > (x.npro = y.npro AND x. nco = y.nco )) ;
/* il n'existe pas une ligne de commande livrée x qui corresponde à la ligne
   de commande différée y */
/* dif contient l'ensemble des tuples qui correspondent aux lignes de commande
   différées non encore servies*/
      cumul :=0 ;
      foreach ligne in dif until cumul > produit-stock.qtestock do
      begin
        cumul := cumul + ligne.dif-qte
/* on calcule le cumul des quantités différées que l'on peut servir avec la
   quantité disponible en stock */
      if cumul = <produit-stock . qtestock
      then facteur-ev4-op9 : < (ligne.dif-nco, ligne.dif-npro,
           ligne.dif-qte)
      end
/* on affecte à facteur-ev4-op9 les tuples correspondant aux lignes de com-
   mande différées que l'on va servir */
          end
        end
      end ;
```

I.3.4.3.2.2.5. OPERATION : pour le texte des opérations

Cette clause a la syntaxe suivante :

```
ident : nom-de-champ  OPERATATION (liste-variables-specif) : specifica-
        tion-texte-assert
```

où liste-variables-spécif : identifie les valeurs passées en paramètre au texte de l'opétation

nom-de-champ : identifie l'attribut qui désigne le texte de l'opération

specification-texte-assert : identifie le texte de l'opération

ident : c'est le nom donné à la contrainte d'intégrité.

Cette contrainte d'intégrité décrit le texte d'une opération. Il représente la description de la règle de gestion que l'on exécute dans le SI pour le faire passer d'un état à l'autre.

Le texte d'une c-opération a comme entrée le tuple défini par le facteur de déclenchement. Dans le cas où ce tuple appartient à une relation de is-type cobject on déclare comme paramètre formel le nom de la relation et dans le cas où ce tuple est composé par le facteur de déclenchement on déclare comme paramètres formule les attributs du tuple et leurs types.

Exemples : extraits de I.3.5.

a) dans cet exemple on utilise les relations :

produit-demande-reap (npro, date-proderea, qtedereap)
produit-stock (npro, date-prodsctock, qte stock)
produit-désignation (npro, date-desig, délaidisp, qtecom, seuil)

```
IS-TYPE COPERATION MODE COPTEXTE
   derepro-texte (nop6, date-nop6-tex, texte-op6)
DOMAIN nop6 : SAME OF demande-reap-prod ;
        date-nop6-tex : date (8) ;
        texte-op6 : char ;
ASSERT
dereprotex-1 : nop6, date-nop6-tex COMPOSKEY nop6tex
dereprotex-2 : texte-op6 OPERATION (produit-stock) :
/* cette opération crée une demande de réapprovisionnement */
var ndate : date
    qte-dema : relation nouv-qte : real end ;
begin
    qte-dema : < (x.qtecom) : produit-désignation < x > AND
        x.npro=produit-stock.npro AND x.nprodes CURRENT ;
/* qte-dema contient un tuple qui indique la quantité à commander du produit */
foreach element in qte-dema do
   ndate := time (8)
  insert (prod.npro, ndate, élément.nouv-qte) in
          produit-demande-reap
          end
       end
   end ;
```

b) dans cet exemple on utilise les relations :

produit-stock (npro, date-prostock, qtestock)
ligne-com-init (nco, npro, date-licoin, qte, date-liv, prix)
ligne-com-a-liv (nco, npro, date-aliv, délai, qte aliv)

```
IS-TYPE COPERATION MODE COPTEXTE
   aclicodif-texte (nop9, date-op9tex , texte-op9)
DOMAIN nop9 : SAME OF activ-licodif ;
       date-op9tex : date (8) ;
       texte-op9 : char ;
ASSERT
   aclicodif-tex-1 : nop9, date-op9tex COMPOSKEY nop9tex ;
   aclicodif-tex-2 : texte-op9 OPERATION (inco : integer ; inpro : integer;
                                          iqte : real)
/* les paramètres (formels définis correspondent aux valeurs créées par fac-
   teur-ev4 op9 que l'on a défini en I.3.4.3.2.2.4. */
var lico-init : relation date-livinit : date end ;
    newdélai : integer ; ndate : date
begin
   lico-init : <(x.date-liv) : ligne-com-init < x > AND
                x.nco = inco AND x.npro = inpro AND
                x.nlicoin CURRENT ;
/* lico-init a un seul tuple dont la valeur est la date de livraison de la
   ligne-de-commande */
   ndate := time (8 ;
   foreach element in lico-init do
   newdelai := tsubday (ndate, element.date-livinit)
/* on calcule le délai de livraison de la ligne de commande */
   insert (inco, inpro, ndate, newdelai, iqte) in ligne-com -a-liv
   end
  end
end ;
```

I.3.4.3.2.2.6. Clause LIKE

Cette clause est employée selon la syntaxe suivante :

assert-complexe : LIKE identificateur

où assert-complexe introduit une contrainte d'intégrité d'une relation

identificateur est l'identificateur duquel on obtient le texte de la contrainte d'intégrité décrite par assert-complexe

On associe à la contrainte d'intégrité décrite par"assert-complexe" le texte associé au "identificateur" qui suit la clause LIKE.

L'usage de cette clause est analogue à celui de la clause SAME OF. Seulement au lieu d'identifier le domaine d'un attribut on identifie le texte d'une contrainte d'intégrité.

Exemple : Considérons l'exemple b présenté en I.3.4.3.2.2.4. Il s'agit de décrire la relation.

ev4-op9-prodst-dec (nev4, nop9, date-ev4 op9-dec, cond-ev4-op9, facteur-ev4-op9).

Nous décrivons en particulier, le texte de facteur-ev4-op9 par la contrainte ev4-op9-pro3.

Considérons maintenant la relation

ev4-op8-prodst-dec (nev4,nop8, date-ev4op8-dec, cond-ev4-op8, facteur-ev4-op8)

introduite en II.3.5..

Le texte associé à facteur-ev4-op8 est le même que celui associé à facteur-ev4-op9. Nous utilisons la clause LIKE pour le décrire.

IS-TYPE CEVENT MODE CEVDECLENCHE
ev4op8-prodst-dec (nev4,nop8,date-ev4 op8-dec, cond-ev4 op8,facteur-ev4op8)

```
DOMAIN ...
ASSERT

ev4-op8-pro1 : nev4,nop8,date-ev4-op8-dec COMPOSKEY nev4-op8-dec ;
ev4-op8-pro2 : facteur-ev4-op8 FACTOR (produit-stock) : LIKE
...     facteur ev4-op9
```

I.3.5. PRESENTATION DE L'EXEMPLE EN ISDEL

```
is-type cobject mode cob
CCLASSE-PRODUIT (NPRO, DATE-CRE-PRO, NENT)
       domain NPRO  : integer (4) ;
              DATE-CRE-PRO : date (8) ;
              NENT : same of CCLASSE-ENTREPRISE .
       assert CCPRO-1 : NPRO key ;

CCLASSE-LIGNE-COMMANDE (NCO, NPRO, DATE-CRE-LIGCO)
       domain NCO : same of CCLASSE-COMMANDE ;
              NPRO: same of CCLASSE-PRODUIT ;
              DATE-CRE-LIGCO = date (8) ;
       assert CCLLIG-CO-1 : NCO, NPRO composkey NLICO ;
CCLASSE-CLIENT ( NCLI,DATE-CRE-CLI)
       domain NCLI  : integer (4) ;
              DATE-CRE-CLI : date (8) ;
       assert CCLI-1 : NCLI key ;
CCLASSE-ENTREPRISE (NENT, DATE-CRE-ENT)
       domain NENT : integer (2) ;
              DATE-CRE-ENT : date (8);
       assert CCENT-1 : NENT key ;
CCLASSE-FACTURE (NFACTURE, DATE-CRE-FAC)
       domain NFACTURE : integer (4) ;
              DATE-CRE-FAC : date (8) ;
       assert CCFAC-1 : NFACTURE key ;
CCLASSE-COMMANDE (NCO, DATE-CRE-COM, NFACTURE, NCLI, NENT)
       domain NCO          : integer (4) ;
              DATE-CRE-COM : date (8) ;
              NFACTURE     : same of CCLASSE-FACTURE .
              NCLI         : same of CCLASSE-CLIENT ;
              NENT         : same of CCLASSE-ENTREPRISE ;
       assert CCO-1 : NCO key ;
```

```
LIGNE-FACTURE (NFACTURE, DATE-LIFAC, NCO, NPRO, MONTANT)
       domain NFACTURE    : same of CCLASSE-FACTURE ;
              DATE-LIFAC  : date (8) ;
              NCO         : same of CCLASSE-COMMANDE ;
              NPRO        : same of CCLASSE-PRODUIT ;
              MONTANT     : real (10,2) ;
       assert NFACTURE, DATE-LIFAC composkey NLIFACT ;

TOTAL-FACTURE (NFACTURE, DATE-TOTFAC, TOTAL-FAC)
       domain NFACTURE     : same of CCLASSE-FACTURE ;
              DATE-TOTFAC  : date (8) ;
              TOTAL-FAC    : real (12,2) ;
       assert TOTFAC-I : NFACTURE, DATE-TOTFAC composkey NFACTOT ;

COMMANDE-INIT (NCO, DATE-COM-INIT, LIB 3), NUMLI)
       domain NCO : same of CCLASSE-COMMANDE .
              DATE-COM-INIT : date (8) ;
              LIB 3 : char (20) ; NUMLI : integer (2);
       assert CI-1 : NCO, DATE-COM-INIT composkey NCOINIT ;

COMMANDE-ENREG (NCO, DATE-ENREG, NBRELICO)
       domain NCO : same of CCLASSE-COMMANDE .
              DATE-ENREG : date (8) ;
              NBRELICO : integer (2) ;
       assert CE-1 : NCO, DATE-ENREG composkey NCOENREG ;
```

```
PRODUIT-DESIGNATION (NPRO, DATE-DESIG, DESIGNATION,DELAIFAB,QTECOM, SEUIL)
       domain NPRO        : same of CCLASSE-PRODUIT .
              DATE-DESIG  : date (8) ; DESIGNATION : char (20) ;
              DELAIFAB    : integer (2) ;
              QTECOM      : real (8,2) ;
              SEUIL       : real (8,2) ;
       assert PRODE-1 : NPRO, DATE-DESIG composkey NPRODES ;

PRODUIT-DEMANDE-REAP (NPkO, DATE-PRODEREA, QTEDEREAP)
       domain NPRO : same of CCLASSE-PRODUIT ;
              DATE-PRODEREA : date (8) ;
              QTEDEREAP : real (B,2) ;
       assert PRODEREAP-1 : NPRO, DATE-PRODEREA composkey NPRODREA ;

PRODUIT-STOCK (NPRO, DATE-PROSTOCK, QTESTOCK) ;
       domain NPRO           : same of CCLASSE-PRODUIT ;
              DATE-PROSTOCK  : date (8) ;
              QTESTOCK       : real (10,2) ;
       assert PROSTO-1 : NPRO, DATE-PROSTOCK composkey NPROSTOCK

PRODUIT-PRIX (NPRO, DATE-PRODPRIX, PRIX-PROD)
       domain NPRO  : same of CCLASSE-PRODUIT; DATE-PRODPMY : date (8) ;
              PRIX-PROD : real (8,2) ;
       assert PROPRI-1 : NPRO, DATE-PRODPRIX composkey NPROPRIX ;
              PROPRI-2 : constraint (PROD-PRIX) .
                         begin
                            if PROD-PRIX => 0 then PROPRI 2 :=true
                          else PRODRI 2 := false end
                          end ;

PRODUIT-REAPPRO (NPRO, DATE-PROREAP, QTEREAP)
       domain NPRO          : same of CCLASSE-PRODUIT ;
              DATE-PROREAP  : date (14) ;
              QTEREAP       : real (8,2) ;
       assert PROREAP-1 : NPRO, DATE-PROREAP composkey NPROREAP
              PROREAP-2 : constraint (QTEREAP)
                          begin
                               if QTEREAP >0 then PROREAP 2:=true
                             else PROREAP 2:= false end
                          end ;
```

```
ENTREPRISE-TAXE (NENT,DATE-TVA, TVA)
       domain NENT      : same of CCLASSE-ENTREPRISE .
              DATE-TVA : date (8) ;
              TVA : real (4,2) ;
       assert ENTVA-1 : NENT, DATE-TVA composkey NENTVA ;

PRODUIT-CATALOGUE (NPRO, DATE-CATAL, CATALOGUE)
       domain NPRO        : same of CCLASSE-PRODUIT ;
              DATE-CATAL : date (8),
              CATALOGUE  : integer (1) ;
       assert PROCAT-1 : NPRO, DATE-CATAL composkey NPROCATAL ;

LIGNE-COM-INIT (NCO, NPRO, DATE-LICOIN, QTE, DATE-LIV, PRIX)
       domain NCO          : same of CCLASSE-LIGNE-COMMANDE ;
              NPRO         : same of CCLASSE-LIGNE-COMMANDE ;
              DATE-LICOIN : date (8) ;
              QTE          : real (6,2) ;
              DATE-LIV     : date (8) ;
              PRIX         : real (6,2) ;
       assert LICOINIT-1 : NCO, NPRO, DATE-LICOIN composkey NLICOINIT1 ;
              LICOINIT-2 : DATE-LICOIN : dateclé ;

LIGNE-COM-A-LIV (NCO, NPRO, DATE-ALIV, DELAI, QTEALIV )
       domain NCO        : same of CCLASSE-LIGNE-COMMANDE
              NPRO       : same of CCLASSE-LIGNE-COMMANDE
              DATE-ALIV : date (8) ;
              DELAI      : integer (2) ; QTEALIV : real (6,2)
       assert LICOALIV-1 : NCO, NPRO, DATE-ALIV composkey NLICOALIV ;

LIGNE-COM-EN-ATT-AC (NCO, NPRO, DATE-ATAC, LIB1)
       domain NCO        : same of CCLASSE-LIGNE-COMMANDE
              NPRO       : same of CCLASSE-LIGNE-COMMANDE
              DATE-ATAC : date (8)
              LIB1       : char (40)
       assert LICOATAC-1 : NCO, NPRO, DATE-ATAC composkey NLICOATAC
```

```
LIGNE-COM-REFFUSEE (NCO, NPRO, DATE-REF, LIB 1)
       domain NCO       : same of CCLASSE-LIGNE-COMMANDE ;
              NPRO      : same of CCLASSE-LIGNE-COMMANDE ;
              DATE-REF  : date (8) ;
              LIB 2     : char (40) ;
       assert LICOREF-1 : NCO, NPRO, DATE-REF composkey NLICOREF

LIGNE-COM-NOUVELLE (NCO, NPRO, DATE-NOUV, NDELAI, NPRIX)
       domain NCO        : same of CCLASSE-LIGNE-COMMANDE ;
              NPRO       : same of CCLASSE-LIGNE-COMMANDE ;
              DATE-NOUV  : date (8) ;
              NDELAI     : integer (2) ;
              NPRIX      : real (6,2) ;
       assert LICONOUV-1 : NCO, NPRO, DATE-NOUV composkey NLICONOUV ;
              LICONOUV-2 : constraint (NPRO, NPRIX) ;
                           type rech-prix = relation proprix : real end ;
                           var  test-prix : rech-prix ;
                           begin test-prix : <(x.PROD-PRIX) : PRODUIT-PRIX
                                                < x > AND
                                   x.NPRO=NPRO AND x. NPRO CURRENT ;
                               foreach element in test-prix do
                               if element.proprix = NPRIX then
                               LICONOUV-2 :=true else LICONOUV-2:=false end
                           end end ;

LIGNE-COM-LIVREE (NCO, NPRO, DATE-LICOLI)
       domain NCO          : same of CCLASSE-LIGNE-COMMANDE ;
              NPRO         : same of CCLASSE-LIGNE-COMMANDE ;
              DATE-LICOLI  : date (8) ;
       assert LICOLIV-1 : NCO, NPRO, DATE-LICOLI composkey NLICOLIV ;

LIGNE-COM-DIFFEREE (NCO, NPRO, DATE-LICODIF, QTEDIF)
       domain NCO           : same of CCLASSE-LIGNE-COMMANDE ;
              NPRO          : same of CCLASSE-LIGNE-COMMANDE ;
              DATE-LICODIF  : date (8) ;
              QTEDIF        : real (6,2) ;
       assert LICODIF-1 : NCO, NPRO composkey NLICODIF ;
```

```
LIGNE-COM-FACTUREE (NFACTURE, DATE-LICOFACT, NCO, NPRO, MONTANT)
       domain NFACTURE : same of CCLASSE-FACTURE ; DATE-LICOFACT : date (8) ;
              NCO      : same of CCLASSE-COMMANDE ;
              NPRO     : same of CCLASSE-PRODUIT ;
              MONTANT  : real (10,2) ;
       assert LICOFAC-1: NFACTURE, DATE-LICOFACT composkey NLIFACT ;

CLIENT-NOM (NCLI, DATE-NOM, NOM)
       domain NCLI     : same of CCLASSE-CLIENT ;
              DATE-NOM : date (8) ;
              NOM      : char (20) ;
       assert CLINO-1 : NCLI, DATE-NOM composkey NCLINOM ;

CLIENT-ADRESSE (NCLI, DATE-ADRESSE, RUE, NUMERO, VILLE, DEPT)
       domain NCLI         : same of CCLASSE-CLIENT ;
              DATE-ADRESSE : date (8) ;
              RUE          : char (20) ;
              NUMERO       : integer (3) ;
              VILLE        : char (20) ;
              DEPT         : integer (2) ;
       assert CLIAD-1 : NCLI, DATE-ADRESSE composkey NCLIADR ;

FACTURE-DATE (NFACTURE, DATE-FAC)
       domain NFACTURE : same of CCLASSE-FACTURE
              DATE-FAC : date (8) ;
       assert FACDA1 : NFACTURE key ;
```

```
is-type c-opération sort c-op-per

CREATION-LIGCOM-A-LIVRER (NPO11, DATE-CLELICOALIV)
       domain NOP1              : char (5) ;
              DATE-CLELICOALIV : date (8) ;
       assert CRELCALIV-1 : NOP1  key ;

CREATION-LIGCOM-DIF (NOP2 , DATE-CLELICODIF)
       domain NOP2             : char (5) ;
              DATE-CRELICODIF : date (8) ;
       assert CRELCDIF 1 : NOP2  key;

ACTUAL-STOCK (NOP3, DATE-ACTSTOCK)
       domain NOP3          : char (5) ;
              DATE-ACSTOCK : date (8) ;
       assert ACTSTO 1 : NOP3  key ;

MISE-ATTENTE-LIGCO (NOP4, DATE-MATLICO)
       domain NOP4           : char (5) ;
              DATE-NMATLICO : date (8) ;
       assert MALICO-1 : NOP4 key ;

OPB-PER (NOPB, DATE-OPB)
       domain NOPB     : char (5) ;
              DATE-OPB : date (8) ;
       assert OPBP-1 : NOPB key ;

OPC-PER (NOPC, DATE-OPC)
       domain NOPC     : char (5) ;
              DATE-OPC : date (8) ;
       assert OPCP-1 : NOPC key ;

OPD-PER (NOPD, DATE-OPD)
       domain NOPD     : char (5) ;
              DATE OPD : date (8) ;
       assert OPDP-1 : NOPD key ;

OPE-PER (NOPE, DATE-OPE)
       domain NOPE     : char (5) ;
              DATE-OPE : date (8) ;
       assert OPEP-1 : NOPE key ;
```

```
MAJ-STOCK (NOP8, DATE-MASTO)
       domain NOP8       : char (5) ;
              DATE-MASTO : date (8) ;

       assert MAJSTO-1 : NOP8 key ;

ACTIV-LICODIF (NOP9, DATE-ACLICODIF)
       domain NOP9            : char (5) ;
              DATE-ACLICODIF : date (8) ;
       assert ACLICODIF-1 : NOP9 key ;

ACTIV-LICOAT (NOP10 , DATE-ACLICOAT)
       domain NOP10         : char (5) ;
              DATE-ACLICOAT : date (8) ;
       assert ACLICOAT-1 : NOP10  key ;

DIF-NOUV-LICO (NOP11 , DATE-DIFNOULCO)
       domain NOP11           : char (5) ;
              DATE-DIFNOULCO : date (8) ;
       assert DIFNOULICO-1 : NOP11  key ;

REFUS-LIGCOM (NOP5, DATE-REFLICO)
       domain NOP5          : char (5) ;
              DATE-REFLICO : date (8) ;
       assert REFLICO 1 : NOP5 key ;

DEMANDE-REAP-PROD (NOP6, DATE-DEREPRO)
       domain NOP6          : char (5) ;
              DATE-DEREPKO : date (8) ;
       assert DEREPRO-1 : NOP6 key ;

REAPROV-STOCK (NOP7, DATE-RESTO)
       domain NOP7 :      : char (5) ;
              DATE-RESTO : date (8) ;
       assert RESTO-1 : NOP 7 key ;

ACTUAL-NOULICO-STOCK (NOP12 , DATE-ACNLCSTO,
       domain NOP12         : char (5) ;
              DATE-ACNLCSTO : date (8) ;
       assert ACNLSTO-1 : NOP12  key ;
```

```
is-type coperation mode coptexte

CRELCALIV-TEXTE (NOP-1, DATE-NOP1TEX, TEXTE-OP1)
       domain NOP-1         : same of CREATION-LIGCOM-A-LIVRER ;
              DATE-NOP1TEX  : date (8) ;
              TEXTE-OP1     : char (10) ;
       assert CRE1-CALIV-TEX-1 :NOP-1, DATE-NOP1TEX composkey NOP1TEX ;

CRELCALIVTEX-2 : TEXTE-OP1 operation (LIGNE-COM-INIT) :
       var ndate : date ; delai : integer ;
       begin ndate :=time (8) ;
             delai :=tsubday (LIGNE-COM-INIT.DATE-LIV, ndate)
             insert (LIGNE-COM-INIT.NCO,LIGNE-COM-INIT.NPRO, ndate,delai,
                     LIGNE-COM-INIT.QTEALIV) in LIGNE-COM-A-LIV end ;

/* tsubday est une fonction standard qui calcule la différence entre deux
   dates et l'exprime comme un nombre entier de jours */

CRELCDIF-TEXTE (NOP2, DATE-NOP2TEX, TEXTE-OP2)
       domain NOP2          : same of CREATION-LIGCOM-DIF ;
              DATE-NOP2 TEX : date (8) ;
              TEXTE-OP2     : char (10) ;
       assert CRELCDIFTEX-1 : NOP2, DATE-NOP2TEX composkey NOP2TEX ;

CRELCDIFTEX 2 : TEXTE-OP2 operation (LIGNE-COM-INIT) :
       var ndate : date ; lib-1 : char ;
       begin lib-1 :='attente réapprovisionnement' ;
             ndate:=time (8) ;
             insert (LIGNE-COM-INIT.NCO,LIGNE-COM-INIT.NPRO,ndate,lib-1)
             in  LIGNE-COM-DIFFEREE
       end ;

ACTSOT-TEXTE (NOP3, DATE-NOP3TEX, TEXTE-OP3)
       domain NOP3          : same of ACTUAL-STOCK ;
              DATE-NOP3TEX  : date (8) ;
              TEXTE-OP3     : char (10) ;
       assert ACTSTOTEX-1 : NOP3,DATE-NOP3TEX composkey  NOP3TEX ;
```

```
ACTSTOTEX-2 : TEXTE-OP3 operation (LIGNE-COM-INIT) :
      var ac-stock : relation qte-st : real end ; val-stock : real ;
          ndate : date ;
      begin  ac-stock : <(x.QTESTOCK) : PRODUIT-STOCK < x > AND
             x.NPRO=LIGNE-COM-INIT.NPRO AND
             x.NPROSTOCK CURRENT ;
      foreach element in ac-stock do
          begin
               val-stock:=element.qte-st - LIGNE-COM-INIT.QTE
          end end ;
      ndate:= time (8) ;
      insert (LIGNE-COM-INIT.NPRO, ndate, val-stock) in PRODUIT-STOCK
     end ;

MALICO-TEXTE (NOP4, DATE-NOP4TEX, TEXTE-OP4)
      domain NOP4  : same of MISE-ATTENTE-LIGCO ;
             DATE-NOP4TEX : date (8) ;
             TEXTE-OP4 : char (10) ;
      assert MALICOTEX 1 : NOP4TEX composkey NOP4TEX ;

MALICOTEX 2 : TEXTE-OP4 operation (LIGNE-COM-INIT) :
      var ndate : date ; lib-2 : char ;
      begin lib-2 := 'attente accord prix/délai' ;
            ndate:= time (8) ;
            insert (LIGNE-COM-INIT.NCO, LIGNE-COM-INIT.NPRO, ndate,lib2)
                   in LIGNE-COM-EN-ATT-AC
      end ;

REFLICO-TEXTE (NOP5, DATE-NOP5TEX, TEXTE-OP5)
      domain NOP5          : same of REFUS-LIGCOM ;
             DATE-NOP5TEX : date (8) ;
             TEXTE-OP5     : char (10) ;
      assert REFLICOTEX 1 : NOP5, DATE-NOP5TEX composkey NOP5TEX ;

REFLICOTEX 2 : TEXTE-OP5 operation (LIGNE-COM-INIT) :
      var ndate : date
      begin ndate:=time (8) ;
      insert (LIGNE-COM-INIT.NCO, LIGNE-COM-INIT-NPRO,ndate) in
             LIGNE-COM-REFFUSEE
      end ;
```

```
DEREPRO-TEXTE (NOP6, DATE-NOP6TEX, TEXTE-OP6)
       domain NOP6          : same of DEMANDE-REAP-PROD ;
              DATE-NOP6TEX : date (8) ;
              TEXTE-OP6 : char (10) ;
       assert DEREPROTEX-1 : NOP6, DATE-NOP6TEX composkey NOP6TEX ;

DEREPROTEX 2 : TEXTE-OP6 operation (PRODUIT-STOCK) :
       var ndate : date ; qte-dem : real ;
           qte-adem : relation nouv-qte : real end ;
       begin
           qte-adem : <(X.QTECOM) : PRODUIT-DESIGNATION < x > AND
                       X.NPRO = PRODUIT-STOCK.NPRO AND
                       X.NPRODES CURRENT ;
           foreach element in qte-adem do qte-dem:=element.nouv-qte end ;
           ndate:= time (8) ;
           insert (PRODUIT-STOCK.NPRO, ndate, qte-dem) in
                   PRODUIT-DEMANDE-REAP
       end ;

RESTO-TEXTE (NOP7, DATE-NOP7TEX, TEXTE-OP7)
       domain NOP7  : same of REAPPROV-STOCK ;
              DATE-NOP7TEX : date (8) ;
              TEXTE-OP7    : char (10) ;
       assert RESTOTEX-1 : NOP7, DATE-NOP7TEX composkey NOP7TEX ;

RESTOTEX 2 : TEXTE-OP7 operation (PRODUIT-REAPPRO) :
       var ac-stock : relation qte-st : real end ; val-stock : real ;
                      ndate : date ;
       begin
            ac-stock : <(X.QTESTOCK) : PRODUIT-STOCK < x > AND
                        X.NPRO=PRODUIT-REAPPRO.NPRO AND X.NPROSTOCK CURRENT ;
       foreach element in ac-stock do
       begin val-stock := element.qte-st + PRODUIT-REAPPRO.QTEREAP end end
           ndate := time (8) ;
       insert (PRODUIT-REAPPRO.NPRO, ndate, val-stock) in PRODUIT-STOCK
     end ;
```

```
MAJSTO-TEXTE (NOP8, DATE-NOP8TEX, TEXTE-NOP8)
        domain NOP8 : same of MAJ-STOCK .
               DATE-NOP8TEX : date (8) ;
               TEXTE-NOP8 : char (10) ;
        assert MAJSTOTEX-1 : NOP8, DATE-NOP8 composkey NOP8 TEX ;
               MAJSTOTEX-2 : TEXTE-NOP8 operation (fact-npro : integer ;
                                                   fact-qte : real) :
            var ndate : date ;
                ac-stock : relation qte-st : real end ; val-stock : real ;
            begin
                ac-stock : < (x.QTESTOCK) : PRODUIT-STOCK < x > AND
                             x.NPRO = fact-npro AND
                             x.NPROSTOCK CURRENT ;
                foreach element in ac-stock do
                begin
                     val-stock := element.qte-st - fact-qte
               end end
            ndate := time (8) ;
            insert (fact-npro, ndate,val-stock) in PRODUIT-STOCK
        end ;
ACLICODIF-TEXTE (NOP9, DATE-OP3TEX, TEXTE-OP9)
        domain NOP9     : same of ACTIV-LICODIF ;
               DATE-OP9TEX : date (8) ;
               TEXTE-OP9 : char (10) ;
        assert ACLICODIFTEX-1 : NOP9 , DATE-OP9 TEX composkey NOP9 TEX ;
               ACLICODIFTEX-2 : TEXTE-OP9 operation (INCO:integer.INPRO :
                                integer ; IQTE:real)
            var lico-init : relation date-livinit : date end ; newdelai :
                            integer ; ndate : date ;
            begin
                lico-init : < (x.DATE-LIV) : LIGNE-COM-INIT < x > AND x.NCO=INCO
                            AND x. NPRO = INPRO AND x. NLICO CURRENT ;
                ndate := time (8) ;
            foreach element in lico-init do
                newdelai := tsubday (ndate, element.date-livinit)
            end ;
             insert (INCO,INPRO,ndate,newdelai,IQTE) in LIG-COM-A-LIV
          end ;
```

```
ACLICOAT-TEXTE (NOP10, DATE-NOP10, TEXTE-OP10)
       domain NOP10      : same of ACTIV-LICOAT ;
              DATE-NOP10 : date (8) ;
              TEXTE-NOP10 : prog ;      )
       assert ACLICOATTEX-1 : NOP10 , DATE-NOP10 composkey NOP10TEX ;
              ACLICOATTEX-2 : TEXTE-OP10 operation (INCO:integer ; INPRO :
                              integer ; DATE-NOUV : date ; NDELAI : integer ;
                              NPRIX : real)
           var ndate : date ;
           begin
                ndate := time (14) ;
                insert (INCO, INPRO, ndate, NDELAI) in LIG-COM-A-LIV
           end ;

DIFNOULICO-TEXTE (NOP11, DATE-NOP11TEX, TEXTE-NOP11)
       domain NOP11      : same of DIF-NOUV-LICO ;
              DATE-NOP11TEX : date (8) ;
              TEXTE-NOP11 : char (10) ;
       assert DIFNOULICO-1 : NOP11, DATE-NOP11 composkey NOP11TEX ;
              DIFNOULICO-2 : TEXTE-NOP11 operation (INCO:integer;INPRO:integer;
                                    DATE-NOUV:date;NDELAI:integer;
                                    NPRIX : real)
           var ndate : date ;
           begin
                ndate : time (8) ;
                insert (INCO, INPRO, ndate) in LIGNE-COM-DIFFEREE
           end ;

ACNLSTO-TEXTE (NOP12, DATE-NOP12TEX, TEXTE-NOP12-
       domain NOP12  : same of ACTUAL-NOULICO-STOCK .
              DATE-NOP12 : date (8) ;
              TEXTE-NOP12 : char (10) ;
       assert ACNLSTOTEX-1 : NOP12, DATE-NOP12TEX composkey NOP12TEX ;
              ACNLSTOTEX-2 : TEXTE-NOP14 operation (INCO:integer;INPRO:integer;
                             DATE-NOUV : date; NDELAI:integer; NPRIX :real)
           var lico-init : relation qte-licoin :real end ; ndate : date ;
               as-stock : relation qte-st:date end ; val-stock : real ;
           begin
               ac-stock :<(x.QTESTOCK):PRODUIT-STOCK < x > AND
                         x.NPRO = INPRO AND
                         x.NPRO CURRENT ;
```

```
OPB-TEXTE (NOPB, DATE-OPB-TEXTE, TEXTE-OPB)
       domain NOPB  : same of NOPB-PER ;
              DATE-OPB-TEXTE : date (8) ;
              TEXTE-OPB : char (10) ;
       assert
       OPBT-1 : NOPB, DATE-OPB-TEXTE composkey NOPBTEX ;
       OPBT-2 : TEXTE-OPB operation (LIGNE-COM-LIVREE) :
           var prix : relation val-prix : real end ;
               tva  : relation val-tva : real end ;
               fac  : relation num-fac : integer end ;
           lico-mont-fac:real ; mont-lies : relation qte : real end ;
           begin
             prix : <(x.prix-prod) : produit-prix < x > AND
                    x.npro=LIGNE-COM-LIVREE.NPRO AND x.NPROSTOCK CURRENT ;
             tva : < (y.TVA):ENTREPRISE-TAXE < y > AND y.NENTVA CURRENT ;
             fac : < (z.NFACTURE) : CCLASSE-COMMANDE < z > AND
                      z.NCO = LIGNE-COM-LIVREE.NCO ;
             mont-lico : <(w.QTEALIV) : LIGNE-COM-A-LIV <w > AND w.NCO =
                                        LIGNE-COM-LIVREE.NCO AND w.NPRO =
                                        LIGNE-COM-LIVREE.NPRO ;
           foreach tva in tva do
             foreach vfac in fac do
               foreach vprix in prix do
                 foreach vmont-lico in mont-lico do
               begin ndate := time (8) ;
               lico-mont-fac:=(vmont-lico.qte * vprix.val-prix) *
                              (1 + vtva.val-tva) ;
           insert (vfac.num -fac, ndate,LIGNE-COM-LIVREE.NCO, LIGNE-COM-LIVREE.
                   NPRO, lico-mont-fac)
           in LIGNE-FACTURE
           end end end end end end end ;
OPC-TEXTE (NOPC, DATE-OPC-TEXTE, TEXTE-OPC)
      domain NOPC    : same of OPC-PER
             DATE-OPC-TEXTE : date (8) ;
             TEXTE-OPC : char (10) :
```

```
        assert
        OPCT-1 : NOPC, DATE-OPC-TEXTE compskey NOPCTEX ;
        OPCT-2 : TEXTE-OPC operation (LIGNE-COM-LIVREE) ;
            var ndate : date ; cumul : integer ;
                num-lig : relation cum : integer end ;
            begin
                ndate := time (8) ;
                num-lig:= (x.NBNELICO) : COMMANDE-ENREG < x> AND
                          x. NCO = LIGNE-COM-LIVRE. NCO ;
            foreach numero-ligne in num-lig do
               cumul := numero-ligne.cum + 1 ;
            insert (LIGNE-COM-LIVREE.NCO ndate, cumul) in COMMANDE-ENREG
            end end end ;
OPD-TEXTE (NOPD, DATE-OPD-TEXTE, TEXTE-OPD)
        domain NOPD : same of OPD-PER ;
               DATE-OPD-TEXTE : date (8) ;
               TEXTE-OPD : char (10);
        assert
        OPDT-1 : NOPD, DATE-OPD, TEXTE composkey NOPDTEX ;
        OPDT-2 : TEXTE-OPD operation (COMMANDE-INIT) :
            var ndate : date ;
            begin ndate := time (8)
            insert (COMMANDE-INIT.NCO, ndate, COMMANDE-INIT.NUMLI)
            in COMMANDE-ENREG
               end ;
OPE-TEXTE (NOPE, DATE-OPE-TEXTE, TEXTE-OPE)
        domain NOPE  : same of OPE-PER .
               DATE-OPE-TEXTE : date (8) ;
               TEXTE-OPE : char (10) ;
        assert
        OPET-1 : NOPE, DATE-OPE-TEXTE composkey NOPETEX ;
        OPET-2 : TEXTE-OPE operation (COMMANDE-ENREG) :
            var ndate : date ; cumul : real ;
                fac : relation num-fac : integer end ;
```

```
begin
  begin ndate := time (8) ;
    fac : <(x.NFACTURE) : CCLASSE-COMMANDE < x > AND
            x.NCO = COMMANDE-ENREG.NCO
  foreach vfac in fac do
  begin cumul := TOTAL (y. MONTANT) : LIGNE-FACTURE < y >
              AND y. NFACTURE = vfac.num-fac;
  insert
       (vfac.num-fac, ndate, cumul) in TOTAL-FACTURE
end end end end end ;
```

```
is-type coperation mode copmodifie

CRECLCALIV-MOD (NOP1TEX, DATE-OP1-MOD, NLICOALIV)
       domain NOP1TEX      : same of CRELCALIV-TEXTE ;
              DATE-OP1-MOD : date (8) ;
              NLICOALIV    : same of LIGNE-COM-A-LIV ;
       assert CREALIV-MOD-1 : NOP1TEX, DATE-OP1-MOD composkey NOP1MOD ;

CRELCDIF-MOD (NOP2TEX, DATE-OP2-MOD, NLICODIF)
       domain NOP2TEX      : same of CRELCDIF-TEXTE .
              DATE-OP2-MOD : date (8) ;
              NLICODIF     : same of LIGNE-COM-DIFFEREE ;
       assert CRELCDIFMO-1 : NOP2TEX, DATE-OP2-MOD composkey NOP2 MOD ;

ACTSOT-MOD (NOP3TEX, DATE-OP3-MOD, NPROSTOCK)
       domain NOP3TEX      : same of ACTSOT-TEXTE .
              DATE-OP3-MOD : date (8) ;
              NPROSTOCK    : same of PRODUIT-STOCK ;
       assert ACTSOTMO-1 : NOP3TEX, DATE-OP3-MOD composkey NOP3MOD ;

MALICO-MOD (NOP4TEX, DATE-OP4-MOD, NLICOATAC)
       domain NOP4TEX      : same of MALICO-TEXTE ;
              DATE-OP4-MOD : date (8) ;
              NLICOATAC    : same of LIGNE-COM-EN-ATT-AC ;
       assert MALICOMO-1 : NOP4TEX, DATE-OP4-MOD composkey NOP4 MOD ;

REFLICO-MOD (NOP5TEX, DATE-OP5-MOD, NLICOREF)
       domain NOP5TEX      : same of REFLICO-TEXTE ;
              DATE-OP5-MOD : date (8) ;
              NLICOREF     : same of LIGNE-COM-REFFUSEE ;
       assert REFLICOMO-1 : NOP5TEX, DATE-OP5-MOD composkey NOP5MOD ;

DEREPRO-MOD (NOP6TEX, DATE-OP6-MOD, NPRODREA)
       domain NOPGTEX : same of DEREPRO-TEXTE ;
              DATE-OP6-MOD : date (8) ;
              NPRODREA  same of PRODUIT-DEMANDE-REAP ;
       assert DEREPROMO-1 : NOP6TEX, DATE-OP6-MOD composkey NOP6MOD ;

RESTO-MOD (NOP7TEX, DATE-OP7-MOD, NPROSTOCK)
       domain NOP7TEX      : same of RESTO-TEXTE .
              DATE-OP7-MOD : date (8) ;
              NPROSTOCK    : same of PRODUIT-STOCK ;
       assert RESTOMO-1 : NOP7TEX, DATE-OP7-MOD composkey NOP7MOD ;
```

```
MAJSTO-MOD (NOP8TEX, DATE-OP8-MOD, NPROSTOCK)
       domain NOP8TEX      : same of MAJSTO-TEXTE ;
              DATE-OP8-MOD : date (8) ;
              NPROSTOCK    : same of PRODUIT-STOCK ;
       assert MAJSTOMO-1 : NOP8TEX, DATE-OP8-MOD composkey NOP8 MOD ;

ACLICODIF-MOD (NOP9TEX, DATE-OP9-MOD, NLICOALIV)
       domain NOP9TEX      : same of ACLICODIF-TEXTE .
              DATE-OP9-MOD : date (8)
              NLICOALIV    : same of LIGNE-COM-A-LIV ;
       assert ACLICODIFMO-1 : NOP9TEX, DATE-OP9-MOD composkey NOP9 MOD ;

ACLICOAT-MOD (NOP10TEX, DATE-OP10-MOD, NLICOALIV) ;
       domain NOP10TEX      : same of ACLICOAT-TEXTE ;
              DATE-OP10-MOD : date (8) ;
              NLICOALIV     : same of LIGNE-COM-A-LIV .
       assert ACLICOATMO-1 : NOP10TEX, DATE-OP10-MOD composkey NOP10MOD ;

DIFNOULICO-MOD (NOP11TEX, DATE-OP11-MOD, NLICODIF)
       domain NOP11 TEX     : same of DIFNOULICO-TEXTE .
              DATE-OP11-MOD : date (8) ;
              NLICODIF      : same of LIGNE-COM-DIFFEREE ;
       assert DIFNOULICOMO-1 : NOP11TEX, DATE-OP11-MOD composkey NOP11MOD ;

ACNLSTO-MOD (NOP12TEX, DATE-OP12-MOD, NPROSTOCK)
       domain NOP12TEX      : same of ACNLSTO-MOD ;
              DATE-OP12-MOD : date (8) ;
              NPROSTOCK     : same of PRODUIT-STOCK ;
       assert ACNLSTOMO-1 : NOP12TEX, DATE-OP12-MOD composkey NOP12MOD ;

OPB-MOD (NOPBTEX, DATE-OPB-MOD, NLIFACT)
       domain NOPBTEX      : same of OPB-TEXTE ;
              DATE-OPB-MOD : date (8) ;
              NLIFACT      : same of LIGNE-FACTURE ;
       assert OPBM-1 : NOPBTEX, DATE-OPB-MOD composkey NOPBMOD ;

OPC-MOD (NOPCTEX, DATE-OPC-MOD, NCOENREG)
       domain NOPCTEX      : same of NOPC-TEXTE
              DATE-OPC-MOD : date (8) ;
              NCOENREG     : same of COMMANDE-ENREG ;
       assert OPCM-1 : NCOTEX, DATE-OPC-MOD composkey NOPCMOD ;
```

```
OPD-MOD (NOPDTEX, DATE-OPD-MOD, NCOENREG)
       domain NOPDTEX       : same of OPD-TEXTE .
              DATE-OPD-MOD  : date (8) ;
              NCOENREG      : same of COMMANDE-ENREG ;
       assert OPDM-1 : NOPDTEX, DATE-OPD-MOD composkey NOPDMOD ;

OPE-MOD (NOPETEX, DATE-OPE-MOD, NFACTOT)
       domain NOPETEX       : same of OPE-TEXTE ;
              DATE-OPE-MOD  : date (8) ;
              NFACTOT       : same of TOTAL-FACTURE ;
       assert OPEM-1 : NOPETEX, DATE-OPE-MOD composkey NOPEMOD ;
```

```
is-type cevent mode cevper

EV1-PER (NEV1, DATE-CRE-EV1)
       domain NEV1          : char (5) ;
              DATE-CRE-EV1  : date (8) ;
       assert EV1PER1 : NEV1 key ;

EV2-PER (NEV2, DATE-CRE-EV2 )
       domain NEV2          : char (5) ;
              DATE-CRE-EV2  : date (8) ;
       assert EV2PER2 : NEV2 key ;

EV3-PER (NEV3, DATE-CRE-EV3)
       domain NEV3          : char (5) ;
              DATE-CRE-EV3  : date (8) ;
       assert EV3PER1 : NEV3 key ;

EV4-PER (NEV4, DATE-CRE-EV4)
       domain NEV4          : char (5) ;
              DATE-CRE-EV4  : date (8) ;
       assert EV4PER1 : NEV4 key ;

EV5-PER (NEV5, DATE-CRE-EV5)
       domain NEV5          : char (5) ;
              DATE-CRE-EV5  : date (8) ;
       assert EV5PER1 : NEV5 key ;

EV6-PER (NEV6, DATE-CRE-EV6)
       domain NEV6          : char (5) ;
              DATE-CRE-EV6  : date (8) ;
       assert EV6PER1 : NEV6 key ;

EV7-PER (NEV7, DATE-CLE-EV7)
       domain NEV7          : char (5) ;
              DATE-CRE-EV7  : date (8) ;
       assert EV7PER1 : NEV7 key ;

EVA-PER (NEVA, DATE-CRE-EVA)
       domain NEVA          : char (5) ;
              DATE-CRE-EVA  : date (8) ;
       assert EVAPER1 : NEVA key ;
```

```
is-type cevent mode cevpred

EV1-PRED (NEV1, DATE-EV1-PNED, PINIT-EV1, PFIN-EV1)
       domain NEV1          : same of EV1-PER
              DATE-EV1-PRED : date (8) ;
              PINIT-EV1     : char (5) ;
              PFIN-EV1      : char (5) ;
       assert
       EV1PRED1 : NEV1, DATE-EV1-PRED composkey NEV1-PRED
       EV1PRED2 : PINIT-EV1 : initial ;
       EV1PRED3 : PFIN-EV1 : final; EV1PRED4 : DATE-EV1-PRED : datecré ;
       EV1PRED5 : PINIT-EV1 predicate (ligne-com-init) :
                  begin pinit-ev1:=true end ;
       EV1PRED6 : PFIN-EV1 predicate (ligne-com-init) :
                  begin   pfin-ev1:= true end ;

EV2-PNED (NEV2, DATE-EV2-PRED, PINIT-EV2, PFIN-EV2)
       domain NEV2          : same of EV2-PER ;
              DATE-EV1-PRED : date (8) ;
              PINIT-EV2     : char (5) ;
              PFIN-EV2      : char (5) ;
       assert
       EV2PRED1 : NEV2, DATE-EV2-PNED composkey NEV2-PNED ;
       EV2PRED2 : PINIT-EV2 : initial
       EV2PRED3 : PFIN-EV2  : final
       EV2PRED4 : DATE-EV2-PRED : datecre ;
       EV2PRED5 : PINIT-EV2 predicate (produit-stock) :
                  begin
                    foreach prod in produit-stock do
                      if  prod. qtestock > 0 then pinit-ev1 := true end
                      else pinit-ev1 := false end
                  end ;

EV2PNED6 : PFIN-EV2 predicate (produit-stock) :
           var quantite-seuil : relation val-seuil : real end ;
           begin
             foreach prod in produit-stock do
```

```
        quantite-seuil : <(x.seuil) : produit-designation <x> AND
                         x.npro=prod.npro AND x.nprostock CURRENT ;
        foreach seuil-cale in quantite-seuil do
        if  prod.qte stock <seuil-cale.val-seuil
        then pfin-ev2 := true
        else pfin-ev2 :=false
        end end end end ;

EV3-PNED (NEV3, DATE-EV3-PNED, PINIT-EV3, PFIN-EV3)
        domain NEV3          : same of EV3-PER ;
               DATE-EV3-PNED : date (8) ;
               PINIT-EV3     : char (5) ;
               PFIN-EV3      : char (5) ;
        assert
        EV3PNED1 : NEV3, DATE-EV3-PER composkey NEV3-PNED
        EV3PNED2 : PINIT-EV3 : initial ;
        EV3PNED3 : PFIN-EV3 : final, EV3PNED4 : DATE-EV3-PNED : datecre ;
        EV3PNED5 : PINIT-EV3 predicate (produit-reappro) ;
               begin pinit-ev3 := true end ;
        EV3PNED6 : PFIN-EV3 predicate (produit-reappro) :
               begin pfin-ev3 := true end ;

EV4-PNED (NE4, DATE-EV4-PNED, PINIT-EV4, PFIN-EV4)
        domain NEV4          : same of EV4-PER .
               DATE-EV4-PRED : date (8) ;
               PINIT-EV4     : char (5) ;
               PFIN-EV4      : char (5) ;
        assert
        EV4PNED1 : NEV4, DATE-EV4-PNED composkey NEV4-PNED
        EV4PNED2 : PINIT-EV4 : initial ;
        EV4PNED3 : PFIN-EV4  : final ;
        EV4PNED4 : DATE-EV4-PRED : datecre ;
        EV4PNED5 : PINIT-EV4 predicate (produit-stock) :
               begin pinit-ev4 :=true end ;
        EV4PNED6 : PFIN-EV4 predicate (produit-stock) :
               var qte-stock : relation qte : real end
               begin
```

```
        qte-stock :  <(x.qtestock) : produit-stock <x> AND
                     x.npro=produit-stock.npro AND
                     x.date-prostock PREDECESSOR produit-stock.date-prostock
        foreach val in qte-stock do
            if val.qte-stock < produit-stock.qtestock
          then pfin-ev4 := true
          else pfin-ev4 := false
          end end end ;

EV5-PRED (NEVS, DATE-EV5-PRED, PINIT-EV5, PFIN-EV5)
      domain NEV5          : same of EV5-PER ;
             DATE-EV5-PRED : date (8) ;
             PINIT-EV5     : char (5) ;
             PFIN-EV5      : char (5) ;
      assert
      EV5PNED1 : NEV5, DATE-EV5-PNED composkey NEV5-PNED ;
      EV5PNED2 : PINIT-EV5 : initial ;
      EV5PNED3 : PFIN-EV5  : final ;
      EV5PNED4 : DATE-EV5-PNED : datecre ;
      EV5PNED5 : PINIT-EV5 predicate (cclasse-commande) ;
          begin pinit-ev5 := true end ;
      EV5PNED6 : PFIN-EV5 predicate (cclasse-commande) :
          begin pfin-ev5 := true end

EV6-PNED (NEV6, DATE-EV6-PNED, PINIT-EV6, PFIN-EV6)
      domain NEV6          : same of EV6-PER
             DATE-EV6-PNED : date (8) ;
             PINIT-EV6     : char (5) ;
             PFIN-EV6      : char (5) ;
      assert
      EV6PRED1 : NEV6, DATE-EV6-PNED composkey NEV6-PNED
      EV6PRED2 : PINIT-EV6 : initial ;
      EV6PRED3 : PFIN-EV6 : final ;
      EV6PRED4 : DATE-EV6-PNED : datecre ;
      EV6PRED5 : PINIT-EV6 predicate (ligne-com-livree) :
          begin pinit-ev6 := true end ;
      EV6PRED6 : PFIN-EV6 predicate (ligne-com-livree) :
          begin pinit-ev6 := true end ;
```

```
EV7-PNED (NEV7, DATE-EV7-PNED, PINIT-EV7, PFIN-EV7)
       domain NEV7          : same of EV7-PER .
              DATE-EV7-PNED : date (8) ;
              PINIT-EV7     : char (5) ;
              PFIN-EV7      : char (5) ;
       assert
       EV7PNED1 : NEV7, DATE-EV7-PNED composkey NEV7-PNED ;
       EV7PNED2 : PINIT-EV7 : initial ;
       EV7PNED3 : PFIN-EV7  : final ;
       EV7PNED4 : DATE-EV7-PRED : datecre ;
       EV7PNED5 : PINIT-EV7 predicate (ligne-com-nouvelle) :
           begin pinit-ev7 := true end ;
       EV7PNED6 : PFIN-EV7 predicate (ligne-com-nouvelle) :
           begin pinit-ev7 := true end ;
EVA-PNED (NEVA, DATE-EVA-PNED, PINIT-EVA,PFIN-EVA)
       domain NEVA          : same of EVA-PER ;
              DATE-EVA-PNED : date (8) ;
              PINIT-EVA     : char (5) ;
              PRIN-EVA      : char (5) ;
       assert
       EVAPNED1 : NEVA, DATE-CRE-EVA composkey NEVA-PRED ;
       EVAPNED2 : PINIT-EVA : initial ;
       EVAPNED3 : PFIN-EVA : final ;
       EVAPNED4 : DATE-EVA-PRED : datecre ;
       EVAPNED5 : PINIT-EVA predicate (commande-enreg)
           begin pinit-eva := true end ;
       EVAPNED6 : PFIN-EVA predicate (commande-enreg)
           var nombre-ligne : integer
           begin
               nombre-ligne:= COUNT ((x.nco):ligne-com-livree < x > AND
                                      x.nco= commande-enreg.nco)
           if  nombre-ligne = commande-renreg.nbrelico
           then pfin-eva := true
           else pfin-eva := false
           end end ;
```

```
is-type cevent mode cevconstate

EV1-CONSTATE (NEV1-PRED, DATE-EV1-CONSTATE, NLICOINIT)
       domain NEV1-PRED          : same of EV1-PRED ;
              DATE-EV1-CONSTATE  : date (8) ;
              NLICOINIT          : same of LIGNE-COM-INIT ;
       assert EVICON1 : NEV1-PRED, DATE-EV1-CONSTATE composkey NEV1-CONSTATE ;

EV2-CONSTATE (NEV2-PRE, DATE-EV2-CONSTATE, NPROSTOCK)
       domain NEV2-PRED          : same of EV2-PRED ;
              DATE-EV2-CONSTATE  : date (8) ;
              NPROSTOCK          : same of PRODUIT-STOCK
       assert EV2CON1 : NEV2-PNED, DATE-EV2-CONSTATE composkey NEV2-CONSTATE ;

EV3-CONSTATE (NEV3-PRED, DATE-EV3-CONSTATE, NPROREAP ;
       domain NEV3-PNED          : same of EV3-PRED ;
              DATE-EV3-CONSTATE  : date (8) ;
              NPROREAP           : same of PRODUIT-REAPPRO ;
       assert EV3CON 1 : NEV3-PRED, DATE-EV3-CONSTATE composkey NEV3-CONSTATE ;

EV4-CONSTATE (NEV4-PRED, DATE-EV4-CONSTATE, NPROSTOCK)
       domain NEV4-PRED          : same of EV4-PRED ;
              DATE-EV4-CONSTATE  : date (8) ;
              NPROSTOCK          : same of PRODUIT-STOCK ;
       assert EV4CON1 : NEV3-PRED, DATE-EV4-CONSTATE composkey NEV4-CONSTATE ;

EV6-CONSTATE (NEV6-PRED, DATE-EV6-CONSTATE, NLICOLIV)
       domain NEV6-PNED          : same of EV6-PRED ;
              DATE-EV6-CONSTATE  : date (8) ;
              NLICOLIV           : same of LIGNE-COM-LIVREE ;
       assert EVECON1 : NEV5-PRED, DATE-EV6-CONSTATE composkey NEV6-CONSTATE ;

EV7-CONSTATE (NEV7-PRED, DATE-EV7-CONSTATE, NLICONOUV)
       domain NEV7-PRED          : same of NEV7-PRED ;
              DATE-EV7-CONSTATE  : date (8) ;
              NLICONOUV          : same of LIGNE-COM-NOUVELLE ;
       assert EV7CON1 : NEV7-PRED, DATE-EV7-PRED composkey NEV7-CONSTATE ;

EVA-CONSTATE (NEVA-PRED, DATE-EVA-CONSTATE, NCOENREG)
       domain NEVA-PRED          : same of EVA-PER
              DATE-EVA-CONSTATE  : date (8) ;
              NCOENREG           : same of COMMANDE-ENREG ;
       assert EVACON1 : NEVA-PRED, DATE-EVA-CONSTATE composkey NEVA-CONSTATE ;
```

```
is-type cevent mode cevdeclenche

EV1-OP1-LICOIN-DEC (NEV1,NOP1, DATE-EV1OP1-DEC,COND-EV1OP1,FACTEUR-EV1OP1)
       domain NEV1   : same of EV-PER ;
              NOP1   : same of CREATION-LIG-COM-A-LIVRER ;
              DATE-EV1OP1-DEC : date (8) ;
              COND-EV1OP1   : char (5) ;
              FACTEUR-EV1OP1 : char (5) ;
       assert
       EV1OP11: NEV1, NOP1, DATE-EVOP-DEC composkey NEV1OP1-DEC ;
       EV1OP12: DATE-EV1OP1-DEC  : datecle ;

       EV1OP13: COND-EV1OP1 condition (LIGNE-COM-INIT) :
           var prodprix : relation prix-exist : real end ; ncompte: integer ;
                          ndate : date ;
                prodesig : relation delai-exist : integer end ; procat :
                          relation cat : integer end ;
           begin
              prodprix : < (x.PRIX-PROD) : PRODUIT-PRIX < x > AND
                         x.NPRO = LIGNE-COM-INIT.NPRO AND x.NPROPRIX CURRENT ;
              prodesig : < (y.DELAIFAB) : PRODUIT-DESIGNATION < y > AND
                          y.NPRO=LIGNE-COM-INIT.NPRO AND y.NPRODES CURRENT ;
              procat   : < (z.CATALOGUE) : PRODUIT-CATALOGUE < z > AND
                          z.NPRO - LIGNE-COM-INIT.NPRO AND z.NPROCATAL CURRENT ;
              foreach pprix in prodprix do
              if      pprix.prix-exist = LIGNE-COM-INIT.PRIX
              then foreach pdesig in prodesig do
                   begin
                        ndate := time (8) ;
                        ncompte := tsubday (LIGNE-COM-INIT.DATE-LIV,ndate) ;
                   if  ncompte = < pdesig.delai-exist
                   then foreach pcat in procat do
                     if  pcat.cat = 1
                    then COND-EV1OP1 := true
                    else COND-EV1OP1 := false
                       end end
              else COND-EV1OP1 := false
                   end end end
           else COND-EV1OP1 := false
           end end end
           end ;
```

```
EV1OP14 : FACTEUR-EV1OP1 factor (LIGNE_COM-INIT) : ACTUAL ;

EV1-OP2-LICOIN-DEC (NEV1, NOP2, DATE-EV1OP2-DEC,COND-EV1OP2, FACTEUR-EV1OP2
       domain NEV1 : same of EV1-PER ;
              NOP2 : same of CREATION-LIGCOM-DIF ;
              DATE-EV1OP2-DEC : date (8) ;
              COND-EV1OP2 : char (5) ;
              FACTEUR-EV1OP2 : char (5) ;
       assert
       EV1OP2-1 : NEV1, NOP2, DATE-EV1OP2-DEC composkey NEV1-OP2-DEC ;
       EV1OP2-2 : DATE-EV1OP2-DEC : datecle ;
       EVAOP2-3 : COND-EV1OP2 condition (LIGNE-COM-INIT) :
           var prodprix : relation prix-exist : real end ; ncompte : integer ;
                          ndate : date ;
               prodesig : relation delai-exist : integer ; delai-seuil : real end;
               qtexist  : relation qte-exstock : real end ;
               procat   : relation cat : integer end ;
           begin
           prodprix : <(x.PRIX-PROD) : PRODUIT-PRIX < x > AND
                      x.NPRO=LIGNE-COM-INIT.NPRO AND x.NPROPRIX CURRENT ;
           prodesig : <(y.DELAIFAB, y.SEUIL) : PRODUIT-DESIGNATION < y > AND
                       y.NPRO=LIGNE-COM-INIT . NPRO AND y.NPRODES CURRENT ;
           procat   : <(z.CATALOGUE) : PRODUIT-CATALOGUE < z > AND
                       z.NPRO=LIGNE-COM-INIT . NPRO AND z.NPROCATAL CURRENT ;
           qtexist  : <(w.QTESTOCK) : PRODUIT-STOCK < x > AND
                       x.NPRO = LIGNE-COM-INIT . NPRO AND x.NPROSTOCK CURRENT ;
           foreach  pprix in prodprix do
           if   pprix-prix-exist = LIGNE-COM-INIT . PRIX
           then foreach pdesig in prodesig do
                begin
                     ndate := time (8) ;
                     ncompte := tsubday (LIGNE-COM-INIT . DATE-LIV, ndate) ;
                if   ncompte = <pdesig.delai-exist
                then foreach pcat in procat do
                     if pcat-cat = 1
                     then foreach pqte in qtexist do
                          if (pqte.qte)exstock-LIGNE-COM-INIT . QTE) >
                               pdesig.delai-seuil
```

```
                    then COND-EV1OP2 := true
                    else COND-EV1OP2 := false
                    end end
               else COND-EV1OP2 := false
               end end
          else COND-EV1OP2 := false
          end end
     else COND-EV1OP2 := false
     end end end
     end ;
EV1OP2-4 : FACTEUR-EV1OP2   factor (LIGNE-COM-INIT) : ACTUAL ;
```

```
EV1-OP3-LICOIN-DEC (NEV1, NOP3, DATE-EV1OP3-DEC,COND-EV1OP3,FACTEUR-EV1OP3)
      domain NEV1 : same of EV1-PER .
             NOP3 : same of ACTUAL -STOCK ;
             DATE-EV1OP3-DEC : date (8) ;
             COND-EV1OP3 : char (5) ;
             FACTEUR-EV1OP3 : char (5) ;
      assert
      EV1OP3-1 : NEV1, NOP3, DATE-EV1OP3-DEC composkey NEV1-OP3-DEC ;
      EV1OP3-2 : DATE-EV1OP3-DEC : datecre ;
      EV1OP3-3 : COND-EV1OP3-DEC condition (LIGNE-COM-INIT): like COND-EV1OP1 ;
      EV1OP3-4 : FACTEUR-EV1OP3 factor (LIGNE-COM-INIT) : ACTUAL ;

EV1-OP4-LICOIN-DEC (NEV1, NOP4, DATE-EV1OP4-DEC, COND-EV1OP4,
                    FACTEUR-EV1OP4)
      domain NEV1 : same of EV1-PER .
             NOP4 : same of MISE-ATTENTE-LIGCO ;
             DATE-EV1OP4-DEC : date (8) ;
             COND-EV1OP4     : char (5) ;
             FACTEUR-EV1OP4  : char (5) ;
      assert
      EV1 OP4 1 : NEV1, NOP4, DATE-EV1OP4-DEC composkey NEV1OP4 - DEC ;
      EV1 OP4 2 : DATE-EV1OP4-DEC : datecre ;
      EV1 OP4 3 : COND-EV1 OP4-DEC condition (LIGNE-COM-INIT) :
          var prodprix : relation prix-exist : real end ; ncompte : integer ;
              prodesig : relation delai-exist : integer end ; ndate : date ;
          begin
          prodprix : <(x.PRIX-PROD : PRODUIT-PRIX <x> AND
                       x.NPRO=LIGNE-COM-INIT.NPRO AND x.NPROPRIX CURRENT ;
          prodesig : < (y. DELAIFAB) : PRODUIT-DESIGNATION < y > AND
                       y.NPRO=LIGNE-COM-INIT.NPRO AND y.NPRODES CURRENT ;
          foreach pprix in prodprix do
          if not (pprix.prix-exist = LIGNE-COM-INIT.PMX)
          then COND-EV1OP4 := true
          else foreach pdesig in prodesig do
               begin
                    ndate := time (8) ;
                   ncompte:= tsubday (LIGNE-COM-INIT.DATE-LIV, ndate) ;
               if ncompte > pdesig. delai-exist
               then COND-EV1OP4 := true
               else COND-EV1OP4 := false
               end end end
      end end end end ;
EV1OP4-4 :FACTEUR-EV1OP4 factor (LIGNE-COM-INIT) : ACTUAL ;
```

```
EV1-OP5-LICOIN-DEC (NEV1, NOP5, DATE-EV1OP5-DEC, COND-EV1OP5,FACTEUR-EV1OP5)
    domain NEV1 : same of EV1-PER ;
           NOP5 : same of REFUS-LIGCOM ;
           DATE-EV1OP5-DEC : date (8) ;
           COND-EV1OP5 : char (5) ;
           FACTEUR-EV1OP5 : char (5) ;
      assert
      EV1OP5-1 : NEV1, NOP5, DATE-EV1OP5-DEC composkey NEV1OP5-DEC ;
      EV1OP5-2 : DATE-EV1OP5-DEC : datecre ;
      EV1OP5-3 : COND-EV1OP5 condition (LIGNE-COM-INIT) :
           var procat : relation cat : integer end ;
           begin
             foreach pcat in procat do
             if not (pcat = 1)
             then COND-EV1OP5 := true
             else COND-EV1OP5 := false
           end end end ;
      EV1OP5-4 : FACTEUR-EV1OP5 factor (LIGNE-COM-INIT) : ACTUAL ;

    EV2-OP6-PRODST-DEC (NEV2, NOP6, DATE-EV2OP6-DEC, COND-EV2OP6,
                        FACTEUR-EV2OP6)
      domain NEV2 : same of EV2-PER ;
             NOP6 : same of DEMANDE-REAP-PROD ;
             DATE-EV2OP6-DEC : date (8) ;
             COND-EV2OP6  : char (5) ;
             FACTEUR-EV2OP6 : char (5) ;
      assert
      EV2OP6-1 : NEV2, NOP6, DATE-EVAOP6-DEC composkey NEV2OP6-DEC ;
      EV2OP6-2 : DATE-EV OP2-DEC6 : datecre ;
      EV2OP6-3 : COND-EV2OP6 condition (PRODUIT-STOCK) :
           begin COND-EV2OP6 3 := true end
      EV2OP6-4 : FACTEUR-EV2OP6 factor (PRODUIT-STOCK) : ACTUAL ;
```

```
EV3-OP7-PRODST-DEC (NEV3, NOP7, DATE-EV3OP7-DEC, COND-EV3OP7,
                    FACTEUR-EV3OP7)
   domain NEV3 : same of EV3-PER .
          NOP7 : same of REAPPROV-STOCK .
          DATE-EV3OP7-DEC : date (8) ;
          COND-EV3OP7 : char (5) ;
          FACTEUR-EV3OP7 : char (5) ;
   assert
   EV3OP7-1 : NEV3, NOP7, DATE-EV3OP7-DEC composkey NEV3-OP7-DEC ;
   EV3OP7-2 : DATE-EV3OP7-DEC : datecre ;
   EV3OP7-3 : COND-EV3OP7 condition (PRODUIT-REAPPRO) :
              begin COND-EV3OP7 := true end ;
   EV3OP7-4 : FACTEUR-EV3OP7 factor (PRODUIT-REAPPRO) : ACTUAL ;
```

```
EV4-OP8-PRODST-DEC (NEV4, NOP8, DATE-EV4 OP8-DEC, COND-EV4OP8,
                    FACTEUR-EV4OP8)
   domain NEV4 : same of EV4-PER ;
          NOP8 : same of MAJ-STOCK ;
          DATE-EV4OP-DEC date (8) ;
          COND-EV4OP8 : char (5) ;
          FACTEUR-EV4OP8  : char (5) ;
   assert
   EV4OP8-1 : NEV4, NOP8, DATE-EV4OP8-DEC composkey NEV4OP8-DEC ;
   EV4OP8-2 : DATE-EV-OP-DEC : datecre ;
   EV4OP8-3 : COND-EV4OP8 condition (PRODUIT-STOCK) :
       var qte-dif : relation val-qte : real end ;
           teste : boolean ;
       begin
           qte-dif : <(x.QTEDIF) : LIGNE-COM-DIFFEREE < x > AND
                     LIGNE-COM-A-LIV < y > AND x. NPRO=PRODUIT-STOCK.NPRO
                     AND NOT EXIST   < y > (y.NPRO=x.NPRO AND
                                            y.NCO=x.NCO) ;
           teste := false
           foreach valeur in qte-dif until teste =true do
                if valeur.val-qte <PRODUIT-STOCK.QTESTOCK
              then teste := true end end
              if teste := true then COND-EV4OP8 := true
              else COND-EV4OP8 := false
              end
          end ;
   EV4OP8-4 : FACTEUR-EV4OP8 factor (PRODUIT-STOCK) : like EV4OP9 4 ;

EV4-OP9-PRODST-DEC (NEV4, NOP9, DATE-EV4OP9-DEC, COND-EV4OP9,
                    FACTEUR-EV4OP9)
   domain NEV4 : same of EV2-PER .
          NOP9 : same of ACTIV-LICODIF ;
          DATE-EV4OP9-DEC : date (8) ;
          COND-EV4OP9 : char (5) ;
          FACTEUR-EV4OP9 : char (5) ;
```

```
assert
EV4OP9-1 : NEV4, NOP9, DATE-EV4OP9-DEC composkey NEV4OP9-DEC ;
EV4OP9-2 : DATE-EV4OP9-DEC : datecle ;
EV4OP9-4 : COND-EV4OP9 condition (PRODUIT-STOCK): like COND-EV4OP8 ;
EV4OP9-3 : FACTEUR-EV4OP9 factor (PRODUIT-STOCK) :
    var dif : relation dif-nco : integer, dif-npro : integer ;
              dif-qte : real end ;
        cumul : real ;
    begin
        dif : <(y.nco, y.npro, y.qte) :
                LIGNE-COM-A-LIV <x> AND LIGNE-COM-DIFFEREE <y> AND
                EXIST <y> (y.NPRO = PRODUIT-STOCK.NPRO AND
                NOT EXIST <x> (x.npro=y.npro AND x.nco=y.nco))
    cumul :=0
    foreach ligne in dif until cumul > PRODUIT-STOCK.QTESTOCK do
       begin
            cumul := cumul + ligne . dif-qte
       if cumul = < produit-stocker.qtestock
     then
          facteur-ev4op9 : < (ligne.dif-nco, ligne.dif-npro,
                              ligne.dif-qte)
     end
 end end end ;

EV7-OP10-NOULIC-DEC (NEV7, NOP10, DATE-EV7OP10-DEC, EV7OP10,
                     FACTEUR-EV7OP10)
   domain NEV7 : same of EV7-PER ;
          NOP10 : same of ACTIV-LICOAT ;
          DATE-EV7OP10-DEC : date (8) ;
          COND-EV7OP10 : char (5) ;
          FACTEUR-EV7OP10 : char (5) ;
   assert
   EV7OP10-1 : NEV7, NOP10, DATE-EV7OP10-DEC composkey NEV7OP10-DEC ;
   EV7OP10-2 : DATE-EV7-OP10-DEC : datecre ;
   EV7OP10-3 : COND-EV7-OP10 condition (LIGNE-COM-NOUVELLE) ;
       var prodprix : relation prix-exist : real end ;
           prodesig : relation delai-exist : integer end ;
           ncompte :  integer ; ndate : date ; procat : relation cat :
                              integer end ;
```

```
        begin
            prodprix : <(x.PRIX-PROD) : PRODUIT-PRIX < x > AND
              x.NPRO = LIGNE-COM-NOUVELLE.NPRO AND x.NPROPRIX CURRENT ;
            prodesig : <(y.DELAIFAB) : PRODUIT-DESIGNATION < y > AND
              y.NPRO=LIGNE-COM-NOUVELLE . NPRO AND y.NPRODES CURRENT ;
        foreach pprix in prod prix do
        if  pprix.prix-exist = ligne-prix
        then foreach pdesig in prodesig do
             begin
                  ndate := time (8) ;
                 ncompte:= tsubday (LIGNE-COM-NOUVELLE.DATE-LIV,ndate)
             if  ncompte = < pdesig.delai-exist
             then foreach pcat in procat do
                  if pcat.cat = 1
                  then COND-EV1OP1 := true
                  else COND-EV1OP1 := false
                  end end
            else COND-EV1OP1 := false
            end end end
        else COND-EV1OP1 := false
        end end end
    EV7OP10 4 : FACTEUR-EV7OP10 factor (LIGNE-COM-NOUVELLE): ACTUAL ;

EV7-OP11-NOULIC-DEC (NEV7, NOP11-DEC, COND-EV7OP11, FACTEUR-EV7OP11)
    domain NEV 7 : same of EV7-PER ;
           NOP11 : same of DIF-NOUV-LICO ;
           DATE-EV7OP11-DEC : date (8) ;
           COND-EV7OP11 : char (5) ;
           FACTEUR-EV7OP11 : char (5) ;
    assert
    EV7OP11-1 : NEV7, NOP11, DATE-EV7OP11-DEC composkey NEV7OP11-DEC ;
    EV7OP11-2 : DATE-EV7OP11-DEC : datecle ;
    EV7OP11-3 : COND-EV7OP11 condition (LIGNE-COM-NOUVELLE) : like
                                        COND-EV7OP1 ;
    EV7OP11-4 : FACTEUR-EV7OP11 condition (LIGNE-COM-NOUVELLE) : ACTUAL ;
```

```
EV7-OP12-NOULIC-DEC (NEV7, NOP12, DATE-EV7OP12-DEC, DOND-EV7OP12,
                     FACTEUR-EV7OP12)
   domain NEV7 : same of EV7-PER ;
          NOP12: same of ACTUAL-NOULICO ;
          DATE-EV7012-DEC  : date (8) ;
          COND-EV7OP12 : char (5) ;
          FACTEUR-EV7OP12 : char (5) ;

   assert
   EV7OP12 1 : NEV7, NOP12, DATE-EV7OP12-DEC composkey NEV7OP12-DEC ;
   EV7OP12-2 : DATE-EV7OP12-DEC : datecre ;
   EV7OP12-3 : COND-EV7OP12 condition (LIGNE-COM-NOUVELLE) :
       var prodprix : relation prix-exist : real end ; ncompte : integer;
                      ndate : date ;
           prodesig : relation delai-exist : integer ; delai-seuil :
                                             real end ;
           qtexist  : relation qte-exstock : real end ;
           procat   : relation cat : integer end ;
       begin foreach
       prodprix : <(x.PRIX-PROD):PRODUIT-PRIX  <x >AND
                    x.NPRO=LIGNE-COM-NOUVELLE.NPRO AND x.NPROPRIX CURRENT ;
       prodesig : <(y.DELAIFAB, y.SEUIL) : PRODUIT-DESIGNATION <y >AND
                    y.NPRO=LIGNE-COM-NOUVELLE.NPRO AND y.NPRODES CURRENT ;
       procat   : <(z.CATALOGUE) : PRODUIT-CATALOGUE <z >AND
                    z.NPRO=LIGNE-COM-NOUVELLE.NPRO AND z.NPROCATAL CURRENT;
       qtexist  : <w.QTESTOCK) : PRODUIT-STOCK <x >AND
                   w.NPRO=LIGNE-COM-NOUVELLE.NPRO AND w.NPROSTOCK CURRENT ;
       foreach pprix in prodprix do
        if   pprix.prix-exist =  LIG-COM-NOUVELLE.PRIX
        then foreach pdesig in prodesig do
             begin
                  ndate := time (8) ;
               ncompte := tsubday (LIGNE-COM-NOUVELLE.DATE-LIV, ndate) ;
             if ncompte  = < pdesig.delai-exist
            then foreach pcat in procat do
                 if pcat-cat = 1
                 then foreach pqte in qtexist do
                      if (pqte.qte-exstock=LIGNE-COM-NOUVELLE.QTE) >
                           pdesig.delai-seuil
```

```
                then COND-EV1OP12 := true
                else COND-EV1OP12 := false
                end end
            else COND-EV1OP12 := false
            end end
        else COND-EV!OP12 := false
        end end
    else COND-EV1OP2 := false
    end end end
    end ;
    EV7OP12-4 : FACTEUR-EV1OP2   factor (LIGNE-COM-NOUVELLE) : ACTUAL ;

EV6-OPB-LICOLIV-DEC (NEV6, NOPB, DATE-EV6OPB-DEC, COND-EV6OPB,
                     FACTEUR-EV6OPB)
    domain NEV6  : same of EV6-PER ;
           NOPB  : same of OPB-PER ;
           DATE-EV6OPB-DEC : date (8) ;
           COND-EV6OPB  : char (5) ;
           FACTEUR-EV6OPB : char (5) ;
    assert
    EV6OPB-1 : NEV6, DATE-EV6OPB-DEC composkey NEV6OPB-DEC ;
    EV6OPB-2 : DATE-EV6OPB-DEC : datecle ;
    EV6OPB-3 : COND-EV6OPB condition (LIGNE-COM-LIVREE) :
        begin COND-EV6OPB := true end ;
    EV6OPB-4 : FACTEUR-EV6OPB (LIGNE-COM-LIVREE) : ACTUAL ;

EV6-OPC-LICOLIV-DEC (NEV6, NOPC, DATE-EV6OPC-DEC, COND-EV6OPC,
                     FACTOR-EV6OPC)
    domain NEV6 : same of EV6-PER ;
           NOPC : same of OPC-PER ;
           DATE-EV6OPC-DEC : date (8) ;
           COND-EV6OPC  : char (5) ;
           FACTEUR-EV6OPC : char (5) ;
    assert
    EV6OPC-1 : NEV6, NOPC, DATE-EV6OPC-DEC composkey NEVOPC-DEC .
    EV6OPC-2 : DATE-EV6OPC-DEC : datecle ;
    EV6OPC-3 : COND-EV6OPC condition (LIGNE-COM-LIVREE) :
               begin COND-EV6OPC := true end ;
    EV6OPC-4 : FACTEUR-EV6OPC factor (LIGNE-COM-LIVREE) : ACTUAL
```

```
EV5-OPD-COMINIT-DEC (NEV5, NOPD, DATE-EV5OPD-DEC, COND-EV5OPD,
                     FACTEUR-EV5OPD)
   domain NEV5  : same of EV5-PER ;
          NOPD  : same of OPD-PER ;
          DATE-EV5OPD-DEC : date (8) ;
          COND-EV5OPD     : char (5) ;
          FACTEUR-EV5OPD  : char (5) ;
   assert
   EV5OPD 1 : NEV5, NOPD, DATE-EV5OPD-DEC composkey NEV5OPD -DEC ;
   EV5OPD 2 : DATE-EV5OPD-DEC : datecle ;
   EV5OPD 3 : COND-EV5OPD condition (COMMANDE-INIT) :
              begin COND-EV5OPD := true END ;
   EV5OPD 4 : FACTEUR-EV5OPD factor (COMMANDE-INIT) : ACTUAL ;

EVA-OPE-COMENREG-DEC (NEVA,NOPE, DATE-EVAOPE-DEC,COND-EVAOPE,
                      FACTEUR-EVAOPE)
   domain NEVA : same of EVA-PER ;
          NOPE : same of OPE-PER ;
          DATE-EVAOPE-DEC : date (8) ;
          COND-EVAOPE     : char (5) ;
          FACTEUR-EVAOPE : char (5) ;
   assert
   EVAOPE-1 : NEVA, DATE-EVAOPE-DEC coposkey NEVAOPE-DEC ;
   EVAOPE-2 : DATE-EVAOPE-DEC : datecle ;
   EVAOPE-3 : COND-EVAOPE condition (COMMANDE-ENREG) ;
              begin COND-EVAOPE := true end ;
   EVAOPE-4 : FACTEUR-EVAOPE factor (COMMANDE-ENREG) : ACTUAL ;

is type cevent mode cevdeceff

EV1-OP1-DECEFF (NEV1-CONSTATE, NOP1-MODIFIE)
   domain NEV1-CONSTATE : same of EV1-CONSTATE ;
          NOP1-MODIFIE  : same of OP1-MODIFIE ;
   assert EV1OP1EF1 : NV1-CONSTATE, NOP1-MODIFIE composkey NEV1OP1-DECEFF;

EV1-OP2-DECEFF (NE1-CONSTATE, NOP2-MODIFIE)
   domain NEV1-CONSTATE : same of EV1-CONSTATE ;
          NOP2-MODIFIE  : same of OP2-MODIFIE ;
   assert EV1OP2EF1 : NEV1-CONSTATE, NOP2-MODIFIE composkey NEV1OP2-
                      DECEFF ;
```

```
EV1-OP3-DECEFF (NEV1-CONSTATE,NOP3-MODIFIE) ;
   domain NEV1-CONSTATE : same of EV1-CONSTATE ;
          NOP3-MODIFIE  : same of OP3-MODIFIE ;
   assert EV1OP3EF1 : NEV1-CONSTATE, NOP3-MODIFIE composkey NEV1OP3-
                      DECEFF ;

EV1-OP4-DECEFF (NEV1-CONSTATE, NOP4-MODIFIE)
   domain NE1-CONSTATE : same of EV1-CONSTATE ;
          NOP4-MODIFIE : same of OP4-MODIFIE ;
   assert EV1OP4EF1 : NEV1-CONSTATE, NOP4-MODIFIE composkey NEV1OP4-
                      DECEFF ;

EV1-OP5-DECEFF (NEV1-CONSTATE, NOP5-MODIFIE)
   domain NEV1-CONSTATE : same of EV1-CONSTATE .
          NOP5-MODIFIE  : same of OP5-MODIFIE .
   assert EV1OP5EF1 : NEV1-CONSTATE, NOP5-MODIFIE composkey
                      NEV1OP5-DECEFF ;

EV2-OP6-DECEFF (NEV2-CONSTATE, NOP6-MODIFIE)
   domain NEV2-CONSTATE  : same of EV2-CONSTATE ;
          NOP6-MODIFIE   : same of OP6-MODIFIE ;
   assert EV2OP6EF1 : NEV2-CONSTATE, NOP6-MODIFIE composkey
                      NEV2OP6-DECFF ;

EV3-OP7-DECEFF (NEV3-CONSTATE, NOP7-MODIFIE)
   domain NEV3-CONSTATE : same of EV3-CONSTATE ;
          NOP7-MODIFIE  : same of OP7-MODIFIE ;
   assert EV3OP7EF1 : NEV3-CONSTATE,NOP7-MODIFIE composkey
                      NEV3OP7-DECFF ;

EV4-OP8-DECEFF (NEV4-CONSTATE, NOP8-MODIFIE)
   domain NEV4-CONSTATE  : same of EV4-CONSTATE ;
          NOP8-MODIFIE   : same of OP8-MODIFIE ;
   assert EV4OP8EF1 : NEV4-CONSTATE,NOP8-MODIFIE composkey
                      NEV4OP8-DECFF ;

EV4-OP9-DECEFF (NEV4-CONSTATE, NOP9-MODIFIE)
   domain NEV4-CONSTATE : same of EV4-CONSTATE ;
          NOP9-MODIFIE  : same of OP9-MODIFIE ;
   assert EV4OP9EF1 : NEV4-CONSTATE, NOP9-MODIFIE composkey
                      NEV4OP9-DECFF ;
```

```
EV6-OPB-DECEFF (NEV6-CONSTATE, NOPB-MODIFIE)
   domain NEV6-CONSTATE : same of EV6-CONSTATE ;
          NOPB-MODIFIE  : same of OPB-MODIFIE ;
   assert EV6OPB EF1 : NEV6-CONSTATE, NOPB-MODIFIE composkey
                       NEV6OPB-DECFF ;

EV6-OPC-DECEFF (NEV6-CONSTATE, NOPC-MODIFIE)
   domain NEV6-CONSTATE : same of EV6-CONSTATE ;
          NOPC-MODIFIE  : same of OPC-MODIFIE ;
   assert EV6OPCEF1 : NEV6-CONSTATE, NOPC-MODIFIE composkey
                      NEV6OPC-DECFF ;

EV5-OPD-DECEFF (NEV5-CONSTATE, NOPD-MODIFIE)
   domain NEV5-CONSTATE : same of EV5-CONSTATE ;
          NOPD-MODIFIE  : same of OPD-MODIFIE ;
   assert EV5OPDEF1 : NEV5-CONSTATE, NOPD-MODIFIE composkey
                      NEV5OPD-DECFF ;

EV7-OP10-DECEFF (NEV7-CONSTATE, NOP10-MODIFIE)
   domain NEV7-CONSTATE : same of EV7-CONSTATE ;
          NOP10-MODIFIE : same of OP10-MODIFIE ;
   assert EV7OP10EF1 : NEV7-CONSTATE,NOP10-MODIFIE composkey
                       NEV7OP10-DECEFF ;

EV7-OP11-DECEFF (NEV7-CONSTATE, NOP11-MODIFIE)
   domain NEV7-CONSTATE : same of EV7-CONSTATE ;
          NOP11-MODIFIE : same of OP11-MODIFIE ;
   assert EV7OP11EF1 : NEV7-CONSTATE, NOP11-MODIFIE composkey
                       NEV7OP11-DECEFF ;

EV7-OP12-DECEFF (NEV7-CONSTATE, NOP12-MODIFIE)
   domain NEV7-CONSTATE : same of EV7-CONSTATE ;
          NOP12-MODIFIE : same of OP12-MODIFIE ;
   assert EV7OP12EF1 : NEV7-CONSTATE, NOP12-MODIFIE composkey
                       NEV7OP12-DECEFF ;

EVA-OPE-DECEFF (NEVA-CONSTATE,NOPE-MODIFIE)
   domain NEVA-CONSTATE : same of EVA-CONSTATE ;
          NOPE-MODIFIE  : same of OPE-MODIFIE .
   assert EVAOPEEF1 : NEVA-CONSTATE, NOPE-MODIFIE composkey
                      NEVAOPE-DECEFF ;
```

II. LA MACHINE ABSTRAITE

II.1. INTRODUCTION A LA MACHINE ABSTRAITE

II.1.1. LE ROLE DE LA MACHINE ABSTRAITE

L'approche conceptuelle qui a été retenue pour la conception des systèmes d'information conduit à les définir par leur structure conceptuelle de fonctionnement décrite par le langage de spécification précédemment introduit.

Cette description comporte la définition des composants du SI (les c-objets) et la définition de sa logique de fonctionnement (les c-opération, les c-évènements, avec les c-objets et leurs associations). Elle contient ainsi tous les renseignements nécessaires pour gérer l'évolution du SI.

Nous avons choisi de gérer le fonctionnement du SI par un outil appelé, au stade de spécification abstraite ou nous nous plaçons, "machine abstraite".

La machine abstraite est définie à un niveau abstrait qui est indépendant de toute considération technique et de toute machine réelle sur laquelle on peut envisager son implémentation.

Le rôle de la machine est :

1. d'assurer automatiquement l'évolution du SI, c'est-à-dire, le passage d'un etat Ei à l'etat Ej
2. de faire en sorte que cette évolution soit conforme à l'évolution du système réel que le SI représente. Ainsi quand un évènement se produit dans le système réel la machine abstraite doit en prendre compte et contrôler l'exécution dans le SI des transformations conséquentes.

La machine abstraite doit contenir des mécanismes qui assurent la gestion de l'évolution de tout système d'information décrit par sa structure conceptuelle.

II.1.2. LES MECANISMES DE LA MACHINE ABSTRAITE ET LE SUPERVISEUR

Pour assurer son rôle, la machine abstraite met en oeuvre des mécanismes qui lui sont propres.

Ils correspondent à des prémitives de base qui sont :

- reconnaître si le changement d'état d'un objet est un évènement.
- définir quelles sont les opérations conséquentes et contrôler leur exécution.
- gérer des représentations.

Ces mécanismes agissent de façon synchronisée sous le contrôle d'un superviseur qui les active, selon sa propre logique de fonctionnement, de façon à respecter la chronologie des évènements du système réel.

Schématiquement, la logique de fonctionnement du superviseur est la suivante : la machine abstraite reconnaît les évènements qui se produisent en conséquence du changement d'état des objets, détermine les opérations qui doivent être déclenchées, déclenche et contrôle leur exécution, actualise le SI.

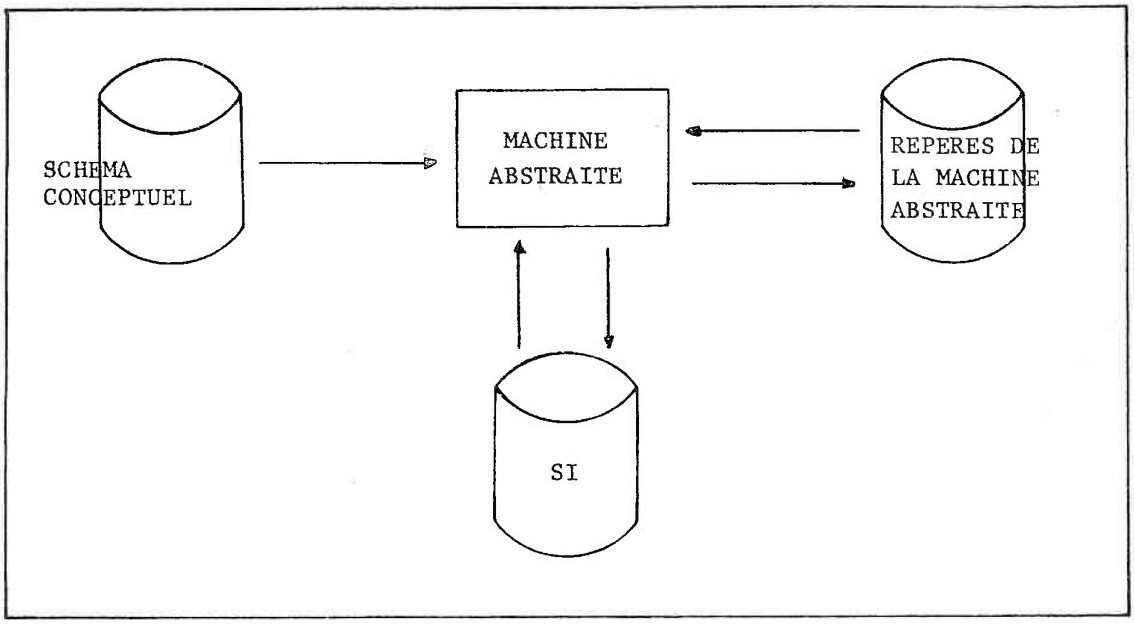
II.1.3. L'ARCHITECTURE DE LA MACHINE ABSTRAITE

Pour assurer son rôle la machine abstraite utilise la description conceptuelle du SI qu'elle gère et des informations qui lui sont propres et que l'on appelle : repères de la machine abstraite.

La description conceptuelle du SI fournit la description de sa logique de fonctionnement sous la forme d'une structure conceptuelle.

Les repères de la machine abstraite sont des repères de l'état du SI.

Ainsi, l'architecture de la machine abstraite est la suivante.



II.1.4. SPECIFIER LA MACHINE ABSTRAITE

Pour spécifier la machine abstraite il nous faut :

i) définir la structure des informations qui constituent toute structure conceptuelle, c'est-à-dire, définir la metastructure conceptuelle dont les occurences définissent la structure conceptuelle d'un SI particulier.
ii) définir la structure des repères de la machine abstraite
iii) définir les mécanismes de base de la machine abstraite et leur enchaînement logique.

Nous nous proposons de donner cette spécification au niveau conceptuel et dans le langage ISDEL.

De cette façon nous disposerons de la spécification tout-à-fait générale d'une machine abstraite elle-même générale.

En effet la machine abstraite est indépendante d'un SI particulier. Elle ne dépend que de la méthode de spécification retenue :

. du modèle de structuration d'une réalité dynamique
. du langage de spécification associé.

L'indépendance de cette spécification vis-à-vis d'un environnement technique particulier permet d'envisager son implémentation sous des formes variées et dans des contextes différents : manuel, fichiers, base de données.

Par ailleurs, comme nous le verrons au chapitre IV, la généralité de cette spécification dépasse le cadre des machines à gérer l'évolution des systèmes d'information.

II.2. SPÉCIFICATION CONCEPTUELLE DE LA MACHINE ABSTRAITE

Nous présentons dans un premier temps le schéma dynamique du fonctionnement conceptuel de la machine abstraite en commentant les informations (repères) qu'il utilise et les opérations qu'il met en oeuvre.

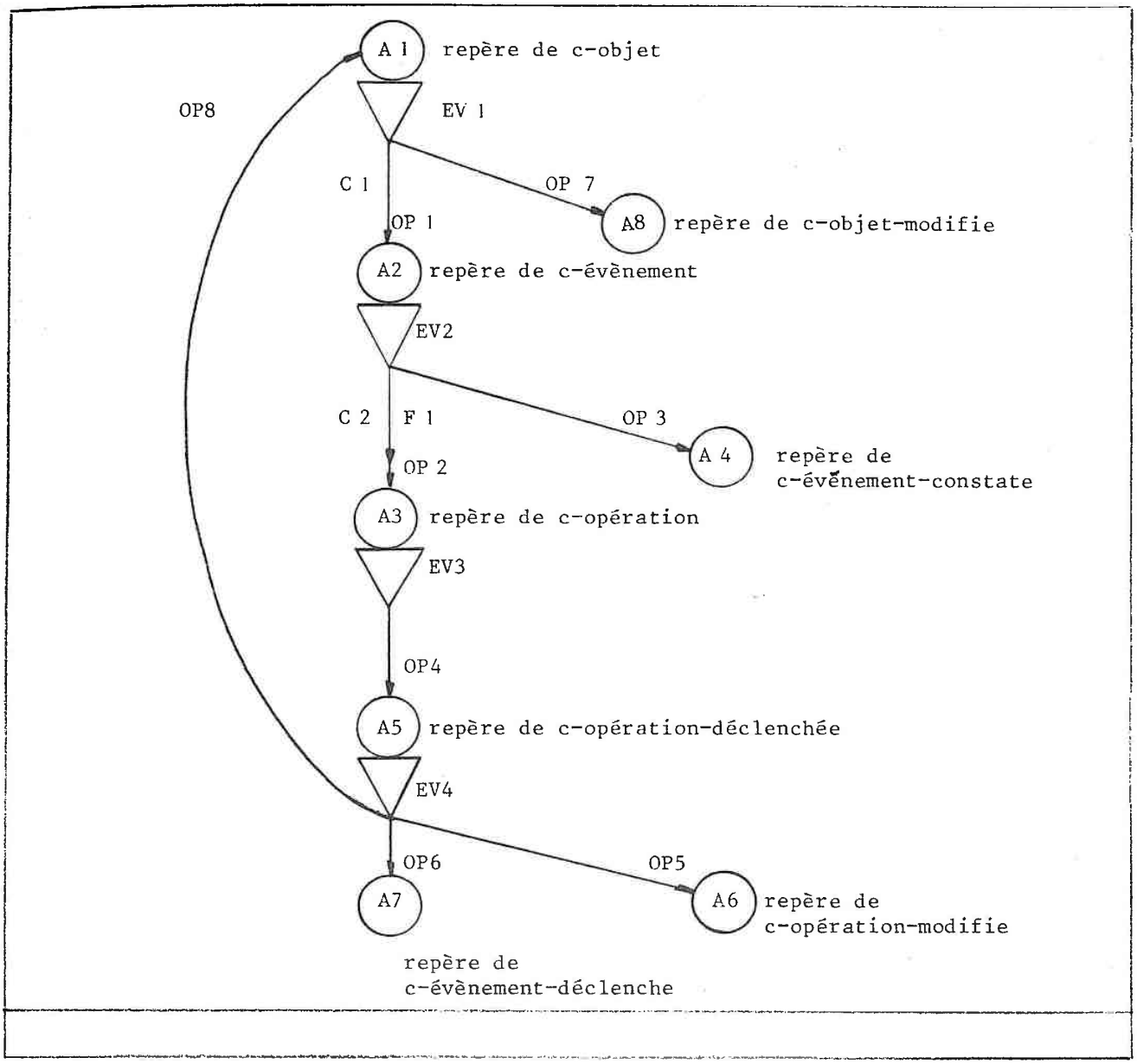
Nous spécifions ensuite la machine abstraite dans le langage ISDEL en donnant :

i) la metastructure conceptuelle
ii) les repères de la machine abstraite (a-objets)
iii) les opérations de la machine abstraite (a-opérations)
iv) les événements de la machine abstraite (a-évènements)

II.2.1. LE SCHEMA DYNAMIQUE DE LA MACHINE ABSTRAITE

Nous en donnons une description graphique selon les conventions introduites dans II.1.3.1. (figure II.1.3.1.1.a)

Le schéma décrit les différentes a-opérations de la machine abstraite et leur synchronisation.


A 1
repère de c-objet
OP8
EV 1
C 1
OP 7
OP 1
A8
repère de c-objet-modifie
A2
repère de c-évènement
EV2
C 2
F 1
OP 3
OP 2
A 4
repère de
c-événement-constate
A3
repère de c-opération
EV3
OP4
A5
repère de c-opération-déclenchée
EV4
OP6
OP5
A7
A6
repère de
c-opération-modifie
repère de
c-évènement-déclenche

II.2.2. LES COMPOSANTS DU SCHEMA DYNAMIQUE DE LA MACHINE ABSTRAITE

II.2.2.1. Les a-objets : les objets de la machine abstraite

Les a-objets doivent indiquer à la machine abstraite les différents états de l'évolution dynamique du SI qu'elle gère.

Ces différents états reflètent les différentes étapes de la logique de fonctionnement de la machine abstraite.

Les a-objets doivent donc fournir des indications qui traduisent :

- dans quelle étape du déroulement de sa logique de fonctionnement se trouve la machine abstraite
- quel est l'état du SI associé à cette étape
- quel est la conséquence sur le SI de l'action de la machine abstraite

La sémantique des a-objets de la machine abstraite est la suivante :

repère de c-objet (A1) : définit l'occurence d'un c-objet à introduire dans le SI

repère de c-évènement (A2) : définit l'occurence d'un c-évènement constaté dans le SI associé à une occurence déterminée de c-objet.

repère de c-opération (A3) : définit l'occurence d'une c-opération choisie pour être déclenchée, associée à une occurence déterminée de c-événement

repère de c-opération (A5) : définit l'occurence d'une c-opération déclenchée, associée à une occurence déterminée de c-opération.

repère de c-évènement-constate (A4): définit l'occurence d'une relation de mode cevconstate à introduire dans le SI, associée à une occurence déterminée de c-événement.

repère de c-opération-modifie (A6) : définit l'occurence d'une relation de mode copmodifié à introduire dans le SI, associée à une occurence déterminée de c-opération-déclenchée.

repère-de c-événement-déclenche (A7) : définit l'occurence d'une relation de mode cevdeclenche à introduire dans le SI, associée à une occurence déterminée de c-opération-déclenchée

repère de c-objet-modifie (A8) : définit l'occurence d'une relation de mode cob, associée à une occurence déterminée de repère de c-objet.

II.2.2.2. Les a-événements : les événements de la machine abstraite

Les a-événements doivent indiquer à la machine abstraite quelle situation de fonctionnement a été atteinte.

Il existe quatre a-événements :

création d'un repère de c-objet (EV1) : la création d'un A1 est un événement pour la machine abstraite. Il traduit le fait qu'une occurence de c-objet a été introduite dans le SI et que la machine abstraite doit le prendre en compte

création d'un repère de c-évènement (EV2) : la création d'un A2 est un événement pour la machine abstraite. Il traduit le fait que le changement d'état d'un c-objet est un évènement du SI et que la machine abstraite doit le prendre en compte.

création d'un repère de c-opération (EV3) : la création d'un A3 est un événement pour la machine abstraite. Il traduit le fait qu'une c-opération du SI doit être déclenchée et que la machine abstraite doit la prendre en compte.

création d'un repère de c-opération-déclenchée (EV4) : la création d'un A5 est un événement pour la machine abstraite. Il traduit le fait qu'une c-opération du SI a été déclenchée et que la machine abstraite doit en prendre compte.

II.2.2.3. Les a-opérations : les opérations de la machine abstraite

Les a-opérations représentent les actions exécutées par la machine abstraite.

Elles représentent les mécanismes de la machine abstraite, c'est-à-dire, les primitives qui assurent son fonctionnement.

Il existe huit a-opérations associées aux quatre a-évènements précédemment introduits.

reconnaissance de c-événements (OP1) : OP1 crée une occurence de repère de c-événement. Elle n'est déclenchée que si la condition C1 est vraie, c'est-à-dire, si le changement d'état du SI décrit par le repère A1 est un événement du SI.

recherche des c-opérations à déclencher (OP2) : OP2 crée une occurence de repère de c-opération. Elle n'est déclenchée que si la condition C2 est vraie, c'est-à-dire, s'il y a des c-opérations du SI associées au c-évènement du SI décrit par le repère A2 et dont la condition de déclenchement est satisfaite.

Par ailleurs, le déclenchement de OP2 peut être multiple : OP2 sera déclenchée autant de fois qu'il y a de c-opérations (identifiées par le facteur de déclenchement F1 de OP2) associées au c-évènement décrit par A2 et dont la condition du déclenchement est satisfaisante.

Cela montre bien que les dépendances chronologiques entre c-évènements (RIC 79) sont respectées car toutes les c-opérations associées à un c-évènement sont prises en compte par la machine abstraite avant la prise en compte des c-opérations associées à d'autres c-événements identifiés eux aussi par la machine abstraite.

actualisation du SI (OP3) : OP3 crée une occurence d'un repère de c-événement-constate (A4). OP3 traduit la création d'une occurence de relation de mode cevconstate associée à l'évènement décrit par le repère A2.

déclenchement des c-opérations (OP4) : OP4 crée une occurence de repère de c-opération-déclenchée (A5). OP4 traduit le lancement de l'exécution de la c-opération du SI décrite par le repère A3.

actualisation du SI (OP5) : OP5 crée une occurence d'un repère de c-opération-modifie (A6). OP5 traduit la création d'une occurence de la Relation de mode copmodifie correspondant à l'opération décrite par le repère A5.

actualisation du SI (OP6) : OP6 crée une occurence d'un repère de c-évènement-déclenche (A7). OP6 traduit la création d'une occurence de la relation de mode cevdeceff correspondant à l'évènement décrit par le repère A2 et l'opération décrite par le repère A5.

actualisation du SI (OP7) : OP7 crée une occurence d'un repère de c-objet-modifie (A8). OP7 traduit la création d'une occurence de la relation de mode cob correspondant à l'objet décrit par le repère A1.

reconnaissance d'un c-objet (OP8) : OP8 crée une occurence d'un repère de c-objet (A1). OP8 traduit la création d'une occurence de un c-objet dans le SI.

II.2.3. DESCRIPTION RELATIONNELLE DES COMPOSANTS DE L'ARCHITECTURE DE LA MACHINE ABSTRAITE

Elle comprend l'introduction sous forme d'ensembles de relations normalisées des structures de données qui sont utilisées par la machine abstraite :

- description de la metastructure
- description des a-objets

II.2.3.1. Description Relationnelle de la Metastructure conceptionnelle

La metastructure conceptuelle définit la structure de l'ensemble informationnel qui constitue la structure conceptuelle de SI

La métastructure conceptuelle est définie une fois pour toutes à partir des schémas de relation utilisés pour décrire toute structure conceptuelle de SI.

Comme on l'a vu au chapitre II, cette description est faite selon 3 types de relation et les 9 modes qui leur correspondent :

COBJECT - COB

COPERATION - COPPER
COPTEXTE
COPMODIFIE

CEVENT - CEVPER
CEVPRED
CEVCONSTATE
CEVDECLENCHE
CEVDECEFF

La metastructure conceptuelle est définie par la collection de relations suivante :

ms-cob	(ms-id-cob, ms-nom-rel-cob)
ms-copper	(ms-id-copper, ms-nom-rel-copper, ms-date-creation-copper)
ms-coptexte	(ms-id-coptexte, ms-nom-rel-coptexte, ms-texte-coptexteq
ms-copmodifie	(ms-id-copmodifie,ms-nom-rel-copmodifie, ms-id-cob)
ms-cevper	(ms-id-cevper,ms-nom-rel-cevper, ms-date-creation-cevper)
ms-cevpred	(ms-id-cevpred,ms-nom-rel-cevpred,ms-pinit,ms-pfin)
ms-cevconstate	(ms-id-cevconstate,ms-nom-rel-cevconstate,ms-id-cob)
ms-cevdeclenche	(ms-id-cevdeclenche,ms-nom-rel-cev-declenche,ms-condition, ms-facteur)
ms-cevdeceff	(ms-id-cevdeceff ms-nom-rel-cevdeceff)
ms-constituants-ob	(ms-id-const-ob,ms-const-ob-type,ms-const-ob-format)
ms-composition-ob	(ms-id-cob, ms-id-const-ob)
ms-composition-clé	(ms-id-clé, ms-id-constituant-clé)

Cette liste de relations est soumise aux contraintes d'intégrité suivantes :

ms-id-copper, ms-date-coptexte composkey ms-id-coptexte
ms-id-coptexte, ms-date-copmodifie composkey ms-id-copmodifie
ms-id-cevper, ms-date-cevpred composkey ms-id-cevpred
ms-id-cevpred, ms-date-cevconstate composkey ms-id-cevconstate
ms-id-cevper, ms-id-copper, ms-date-cevdeclenche composkey ms-id-cevdeclenche
ms-id-cevconstate, ms-id-copmodifie composkey ms-id-cevdeceff

Ces relations sont interprétées de la façon suivante :

1) ms-cob : décrit les relations de mode cob
   - ms-id-cob : est l'identifiant clé de la relation décrite
   - ms-nom-rel-cob : est le nom de la relation décrite

2) ms-copper : décrit un aspect des relations de mode cob
   - ms-id-copper : est l'identifiant clé de la relation décrite
   - ms-nom-rel-copper : est le nom de la relation décrite
   - ms-date-creation-copper : est l'attribut "date de création" de la relation décrite

3) ms-coptexte : décrit les relations de mode coptexte
   - ms-id-coptexte : est l'identifiant de la relation décrite
   - ms-nom-rel-coptexte : est le nom de la relation décrite
   - ms-texte-coptexte : est l'attribut "texte" de la relation décrite

4) ms-copmodifie : décrit les relations de mode copmodifie
   - ms-id-copmodifie est l'identifiant clé de la relation décrite
   - ms-nom-rel-copmodifie : est le nom de la relation décrite
   - ms-id-cob : est l'attribut "identifiant clé du c-objet" de la relation décrite

5) ms-cevper : décrit les relations de mode cevper
   - ms-id-cevper : est l'identifiant clé de la relation décrite
   - ms-nom-rel-cevper : est le nom de la relation décrite
   - ms-date-creation-cevper : est l'attribut "date de création" de la relation décrite.

6) ms-cevpred : décrit les relations de mode cevpred
   ms-id-cevpred : est l'identifiant clé de la relation décrite
   ms-nom-rel-cevpred : est le nom de la relation décrite
   ms-pinit : est l'attribut "prédicat initial" de la relation décrite
   ms-pfin : est l'attribut "prédicat final" de la relation décrite

7) ms-cevconstate : décrit les relations de mode cevconstate
   ms-id-constate : est l'identifiant clé de la relation décrite
   ms-nom-rel-cevconstate : est le nom de la relation décrite
   ms-nom-cob : est l'attribut "identifiant clé du c-objet" de la relation décrite

8) ms-cevdeclenche : décrit les relations de mode cevdeclenche
   ms-id-cevdeclenche : est l'identifiant clé de la relation décrite
   ms-nom-rel-cevdeclenche : est le nom de la relation décrite
   ms-condition : est l'attribut "condition de déclenchement" de la relation décrite
   ms-facteur : est l'attribut "facteur de déclenchement" de la relation décrite

9) ms-cevdeceff : décrit les relations de mode cevdeceff
   ms-id-cevdeceff : est l'identifiant clé de la relation décrite
   ms-nom-rel-cevdeceff : est le nom de la relation décrite

10) ms-constituants-ob : décrit les attributs des relations de mode cob et leur rôle.
   ms-id-const-ob : identifie un attribut de relation de mode cob
   ms-const-ob-type : indique s'il est l'identifiant clé ou s'il en fait partie
   ms-const-ob-format : indique le format d'un attribut de relation de mode cob

11) ms-composition-ob : décrit la composition des relations de mode cob
   ms-id-cob : est l'identifiant de la relation de mode cob
   ms-id-const-ob : est un attribut de la relation de mode cob

12) ms-composition-clé : décrit la composition des identifiants clés des relations
   ms-id-clé : est l'identifiant clé d'une relation
   ms-id-constituant-clé : est l'attribut qui compose la clé

SPECIFICATION CONCEPTUELLE DE LA METASTRUCTURE CONCEPTUELLE
IS-TYPE : C-OBJET
MODE : COBPER

```
ms-cob      (ms-id-cob, ms-nom-rel-cob)
      DOMAIN ms-id-cob       : CHAR ;
             ms -nom-rel-cob : CHAR ;
      ASSERT mscob-1 : ms-id-cob KEY ;

ms-copper  (ms-id-copper, ms-nom-rel-copper,ms-date-creation-copper)
      DOMAIN ms-id-copper            : CHAR ;
             ms-nom-rel-copper       : CHAR ;
             ms-date-creation-copper : CHAR ;
     ASSERT  ms-copper-1 : ms-id-copper KEY ;

ms-coptexte (ms-id-copper, ms-date-coptexte, ms-nom-rel-coptexte,
             ms-texte-coptexte)
      DOMAIN ms-id-copper : SAME OF ms-copper ;
             ms-date-coptexte : CHAR ;
             ms-nom-rel-coptexte : CHAR ;
             ms-texte-coptexte : CHAR ;
      ASSERT
      ms-coptex1:ms-id-copper, ms-date-coptexte COMPOSKEY ms-id-coptexte;

ms-copmodifie (ms-id-coptexte, ms-date-copmodifie, ms-nom-rel-copmodifie,
               ms-id-cob)
      DOMAIN ms-id-coptexte : SAME OF ms-coptexte ;
             ms-date-copmodifie : CHAR ;
             ms-nom-rel-copmodifie : CHAR ;
             ms-id-cob : SAME OF ms-cob ;
      ASSERT
      ms-copmod1 : ms-id-coptexte, ms-date-copmodifie COMPOSKEY
                   ms-id-copmodifie ;

ms-cevper (ms-id-cevper, ms-nom-rel-cevper, ms-date-creation-cevper)
      DOMAIN ms-id-cevper            : CHAR ;
             ms-nom-rel-cevper       : CHAR ;
             ms-date-creation-cevper : CHAR ;
      ASSERT ms-cevper-1 : ms-id-cevper KEY ;

ms-cevpred (ms-id-cevper, ms-date-cevpred, ms-nom-rel-cevpred,
            ms-pinit, ms-pfin)
      DOMAIN ms-id-cevper : SAME OF ms-cevper ;
             ms-date-cevpred : CHAR ;
             ms-nom-rel-cevpred : CHAR ;
             ms-pinit           : CHAR ;
             ms-pfin            : CHAR ;
      ASSERT ms-cevpred1 : ms-id-cevper, ms-date-cevpred COMPOSKEY
                           ms-id-cevpred ;
```

```
SPECIFICATION CONCEPTUELLE DE LA METASTRUCTURE CONCEPTUELLE
IS-TYPE : C-OBJET
MODE : COBPER

ms-cevconstate (ms-id-cevpred, ms-date-cevconstate, ms-nom-rel-cevconstate,
                ms-id-cob)
    DOMAIN ms-id-cevpred : SAME OF  ms-cob ;
           ms-date-cevconstate : CHAR ;
           ms-nom-rel-cevconstate : CHAR ;
           ms-id-cob : SAME OF ms-cob ;

    ASSERT
    mscevconstate1 : ms-id-cevpred, ms-date-cevconstate COMPOSKEY
                     ms-id-cevconstate ;

ms-cevdeclenche (ms-id-cevper, ms-id-copper, ms-date-cevdeclenche ;
                 ms-nom-rel-cevdeclenche, ms-condition, ms-facteur)
    DOMAIN ms-id-cevper : SAME OF ms-cevper ;
           ms-id-copper : SAME OF ms-copper ;
           ms-date-cevedeclenche : CHAR ;
           ms-nom-rel-cevdeclenche : CHAR ;
           ms-condition            : CHAR ;
           ms-facteur              : CHAR ;
    ASSERT
    mscevdeclel : ms-id-cevper, ms-id-copper, ms-date-cevdeclenche
                  COMPOSKEY ms-id-cevdeclenche ;

ms-cevdeceff (ms-id-cevconstate, ms-id-copmodifie, nom-rel-cevdeceff)
    DOMAIN ms-id-cevconstate : SAME OF CONS-CEVCONSTATE ;
           ms-id-copmodifie  : SAME OF ms-copmodifie ;
           nom-rel-cevdeceff : CHAR ;
    ASSERT
    mscevdeceff : ms-id-cevconstate, ms-id-copmodifie COMPOSKEY
                  ms-id-cevdeceff ;

ms-constituants-ob (ms-id-const-ob, ms-const-ob-type, ms-const-ob-format)
     DOMAIN      ms-id-const-ob    : CHAR ;
                 ms-const-ob-type  : CHAR ;
                 ms-const-ob-format: CHAR ;
     ASSERT      ms-coob-1 : ms-id-const-ob KEY

ms-composition-ob (ms-id-cob, ms-id-const-ob)
     DOMAIN      ms-id-cob      : CHAR ;
                 ms-id-const-ob : CHAR ;
     ASSERT      ms-compob-1 : ms-id-cdo, ms-id-const-ob COMPOSKEY ;

ms-composition-clé (ms-id-cle, ms-id-constituant-cle)
     DOMAIN      ms-id-cle            : CHAR ;
                 ms-id-constituant-cle: CHAR ;
     ASSERT      ms-cocle-1 : ms-id-cle, ms-id-constituant-cle  COMPOSKEY
```

II.2.3.2. Description Relationnelle des a-objets

Nous présentons la description de l'ensemble de relations sous forme normalisée qui correspond aux a-objets de la machine abstraite et nous donnons leur interprétation.

repère-de-c-objet (id-rep-cob, id-cob, val-att-cob)
repère-de-c-événement (id-rep-cev, id-cevconstate, val-id-cevper, val-date-cevpred, val-date-cevconstate, id-rep-cob)
repère-de-c-opération (id-rep-cop, id-copper, val-id-copper, id-rep-cev)
repère-de-c-opération-déclenchée (id-rep-copdec, id-cobmod, val-att-cobmod, val-date-coptexte, id-rep-cop)
repère-de-c-événement-constate (id-rep-cevconstate, id-rep-cev)
repère-de-c-opération-modifie (id-rep-copmodifie, id-rep-copdec)
repère-de-c-objet-modifie (id-rep-cobmodifie, id-rep-cob)
repère-de-c-évènement-declenche (id-rep-cevdeclenche , id-rep-copdec)

Ces relations sont interprétées de la façon suivante :

1) repère-de-c-objet

id-rep-cob : indique l'identifiant clé du repère

id-cob : indique le nom de l'identifiant clé du c-objet qui a changé d'état

val-att-cob : indique la valeur du tuple de la relation R qui a produit le changement d'état

2) repère-de-c-événement

id-rep-cev : indique l'identifiant clé du repère

id-cevconstate : indique le nom de l'identifiant clé de la relation R de mode cevconstate

val-id-cevper : indique la valeur de l'identifiant id-cevper qui compose id-cevconstate

val-date-cevpred : indique la valeur de "date-cevpred", c'est-à-dire la date des textes de prédicats qui définissent l'événement.

val-date-cevconstate : indique la valeur de la date du changement d'état qui a donné origine à ce repère-de-c-événement

id-rep-cob : indique la valeur de l'identifiant clé des repère-de-c-objet associé.

3) repère-de-c-opération

id-rep-cop : indique l'identifiant clé du repère

id-copper : indique le nom de l'identifiant clé de la relation R de mode copper

val-rel-copper : indique la valeur de l'identifiant id-copper

id-rep-cev : indique la valeur de l'identifiant clé du repère-de-c-évènement associé

4) repère-de-c-opération-déclenchée

id-rep-copdec : indique l'identifiant clé du repère

id-cobmod : indique le nom de l'identifiant clé du c-objet modifié par la c-opération déclenchée

val-date-coptexte : indique la valeur de la date du texte de la c-opération

val-att-cobmod : indique la valeur du tuple du c-objet qui a été modifié

id-rep-cop : indique la valeur de l'identifiant clé du repère-de-c-opération associé

5) repère-de-c-événement-constate

id-rep-cevconstate : indique l'identifiant du repère

id-rep-cev : indique la valeur de l'identifiant clé du repère-de-c-événement associé

6) repère-de-c-opération-modifie

id-rep-copmodifie : indique l'identifiant clé du repère

id-rep-copdec : indique la valeur de l'identifiant clé du repère-de-c-opération-déclenchée associé.

7) repère-de-c-objet-modifie :

id-rep-cobmodifié : indique l'identifiant clé du repère

id-rep-cob : indique la valeur de l'identifiant clé du repère-de-c-objet associé.

8) repère-de-c-évènement-declenche

id-rep-cevdeclenche : indique l'identifiant clé du repère

id-rep-copdec : indique la valeur de l'identifiant clé du repère-de-c-opération-déclenchée associé.

II.3. SPÉCIFICATION CONCEPTUELLE DE LA MACHINE ABSTRAITE EN ISDEL

Nous présentons la spécification des a-objets, des a-opérations et des a-événements de la machine abstraite dans le langage ISDEL.

Nous décrivons les a-objets par des relations de is-type cobject, les a-opérations par des relations de is-type coperation et les a-évènements par des relations de is-type cevent.

II.3.1. LES a-objets DE LA MACHINE ABSTRAITE

Leur spécification conceptuelle est faite à partir de leur description relationnelle présentée en II.2.3.2.

II.3.2. LES a-opérations DE LA MACHINE ABSTRAITE

Leur spécification conceptuelle est faite à partir de leur description donnée en II.2.2.3.

Nous présentons successivement les relations de mode copper , de mode coptexte et de mode copmodifie

II.3.2.1. Mode copper

Les relations de ce mode introduisent les a-opérations de la machine abstraite.

II.3.2.2. Mode coptexte

Les relations de ce mode introduisent le texte des a-opérations de la machine abstraite.

II.3.2.3. Mode copmodifié

Les relations de ce mode expriment les a-objets modifiés par chaque a-opération .

II.3.3. LES a-évènements DE LA MACHINE ABSTRAITE

Leur spécification conceptuelle est faite à partir de leur description donnée en II.2.2.2..

Nous présentons successivement les relations de mode cevper, de mode cevpred, de mode cevconstate, de mode cevdeclenche et de mode cevdeceff.

II.3.3.1. Mode cevper

Les relations de ce mode introduisent les a-évènements de la machine abstraite.

II.3.3.2. Mode cevpred

Les relations de ce mode introduisent les textes des prédicats qui caractérisent a-événement de la machine abstraite.

II.3.3.3. Mode cevconstate

Les relations de ce mode expriment le a-objet associé à chaque a-évènement de la machine abstraite.

II.3.3.4. Mode cevdeclenche

Les relations de ce mode introduisent le texte du facteur de déclenchement et de la condition de déclenchement de chaque a-opération associée à un a-évènement de la machine abstraite.

II.3.3.5. Mode cevdeceff

Les relations de ce mode expriment le déclenchement successif des a-opérations associées aux a-évènements de la machine abstraite.

SPECIFICATION CONCEPTUELLE DES A-OBJETS
IS-TYPE : C-OBJECT
MODE : COBPER


```
IS-TYPE C-OBJECT MODE COBPER
   repère-de-c-objet (id-rep-cob, id-cob, val-att-cob)
   DOMAIN  id-rep-cob : INTEGER ;
           id-cob     : CHAR ;
           val-att-cob: CHAR ;
   ASSERT  rco1 : id-rep-cob KEY ;

   repère-de-c-évènement (id-rep-cev,id-cevconstate,val-id-cevper, val-
                          date-cevpred, val-date-cob, id-nep-cob)
   DOMAIN  id-rep-cev       : INTEGER      ;
           id-cevconstate   : CHAR         ;
           val-id-cevper    : CHAR         ;
           val-date-cevpred : DATE         ;
           val-date-cob     : DATE         ;
           id-rep-cob       : SAME OF repère-de-c-objet ;
   ASSERT  rcev1 : id-rep-cev KEY ;
           rcev2 : val-id-cevper, val-date-cevpred, val-date-cob
                   COMPOSE val-id-cev ;

   repère-de-c-opération (id-rep-cop, id-copper,val-id-copper,id-rep-cev)
   DOMAIN  id-rep-cop : INTEGER ;
           id-copper  : CHAR  ;
           val-id-copper : CHAR ;
           id-rep-cev : SAME OF repère-de-c-évènement ;
   ASSERT  rcop1 : id-rep-cop KEY ;

   repère-de-c-opération-déclenche (id-rep-copdec, id-cobmod,val-att-cobmod
                   id-repcop)
   DOMAIN  id-rep-copdec  : INTEGER ;
           id-cobmod      : CHAR    ;
           val-att-cobmod : CHAR    ;
           id-repcop      : SAME OF repère-de-c-opération ;
   ASSERT  ropdec1 : id-nep-copdec KEY

   repère-de-c-évènement-constate (id-rep-cevconstate, id-rep-cev)
   DOMAIN  id-rep-cevconstate : INTEGER ;
           id-rep-cev         : SAME OF repère-de-c-évènement ;
   ASSERT rcevcon 1 : id-rep-cevconstate KEY ;

   repère-de-c-opération-modifié (id-rep-copmodifié,id-rep-copdec)
   DOMAIN  id-rep-copmodifié : INTEGER ;
           id-rep-copdec     : SAME OF repère-de-c-opération-déclenche ;
   ASSERT  rcopmod 1 : id-rep-copmodifie KEY ;

   repère-de-c-évènement-déclenche (id-rep-cevdéclenche,id-rep-copdec)
   DOMAIN  id-rep-cevdeclenche  : INTEGER ;
           id-rep-copdec        : SAME OF repère-de-c-opération-déclenche;
   ASSERT rcevdec1 : id-rep-cevdeclenche KEY ;

   repère-de-c-objet-modifie (id-rep-cobmodifie, id-rep-copdec)
   DOMAIN  id-rep-cobmodifié : INTEGER .
           id-rep-copdec     : SAME OF repère-de-c-opération-déclenche ;
   ASSERT  rcobmod 1 : id-rep-cobmodifie KEY ;
```

SPECIFICATION CONCEPTUELLE DES A-OPERATIONS
IS-TYPE : C-OPERATION
MODE : COPPER

Page 1 de 1

```
IS-TYPE C-OPERATION MODE COPPER
   Reconnaissance-évènement-per (id-op1, date-cre-op1)
   DOMAIN id-op1       : CHAR  ;
          date-cre-op1 : DATE  ;
   ASSERT op1per1 : id-op1 KEY ;

   Recherche-opération-per (id-op2, date-cre-op2)
   DOMAIN id-op2       : CHAR ;
          date-cre-op2 : DATE ;
   ASSERT op2-per1 : id-op2 KEY ;

   Constatation-évènement-per (id-op3, date-cre-op3)
   DOMAIN id-op3       : CHAR ;
          date-cre-op3 : DATE ;
   ASSERT op3-per-1 : id-op3 KEY ;

   Déclenchement-opération-per (id-op4, date-cre-op4)
   DOMAIN id-op4       : CHAR ;
          date-cre-op4 : DATE ;
   ASSERT op4-per1 : id-op4 KEY ;

   Opération-modifie-per (id-op5, date-cre-op5)
   DOMAIN id-op5       : CHAR  ;
          date-cre-op5 : DATE  .
   ASSERT op5-oer1 : ud-op5 KEY ;

   Déclenchement-effectif-per (id-op6, date-cre-op6)
   DOMAIN id-op6       : CHAR ;
          date-cre-op6 : DATE ;
   ASSERT op6-per-1 : id-op6 KEY ;

   Modification-objet-per (id-op7, date-cre-op7)
   DOMAIN id-op7       : CHAR ;
          date-cre-op7 : DATE ;
   ASSERT op7-per-1 : id-op7 KEY ;

   Reconnaissance-objet-per (id-op8, date-cre-op8)
   DOMAIN id-op8       : CHAR ;
          date-cre-op8 : DATE ;
   ASSERT op8-per-1 : id-op8 KEY ;
```

SPECIFICATION CONCEPTUELLE DES A-OPERATIONS IS-TYPE : C-OPERATION MODE : COPTEXTE	Page 1 de 17 Item 1 de 8

```
IS-TYPE C-OPERATION MODE COPPER
   reconnaissance-évènement-texte (id-op1,dateop1-texte, texte-op1)
   DOMAIN  id-op1          : SAME OF reconnaissance-évènement-per ;
           date-op1-texte : DATE  ;
           texte-op1       : CHAR  ;
   ASSERT
   rcevte-1 : id-op1, date-op1-texte COMPOSKEY id-op1-texte ;
   rcevte-2 : texte op1 OPERATION (repère-de-c-objet) :

/* texte-op1 est le nom du texte de 1'a-opération OP1 */
       var val-ms idev : relation nom-id-cevpred:char ; nom-pinit : char ;
                                  nom-pfin : char ;
                                  nom-rel-cevpred : char ;
                                  nom-ed-cevconstate : char  end ;

/* la variable val-ms idev contient des attributs qui correspondent aux
   c-événements qui peuvent être associés à un changement d'état d'un
   c-objet */
       cod-pred-init-fin : relation sc-id-cevper : char ;
                                    sc-date-cevper : date ;
                                    cod-pinit : char ;
                                    cod-pfin  : char ;
                                    desig-id-cev : char end ;

/* la variable cod-pred-init-fin contient les nuples qui contiennent les
   indicatifs des textes actuels des prédicats associées aux noms de pré-
   dicats de la variable "val-ms idev" */
       val-att-cob-précédant : relation pval-at : char end ;

/* la variable val-att-cob-précédant contient la valeur de l'occurence du
   c-objet avant le changement d'état.*/
       val-id-cob : char ; val-date-cob : date ;
       teste : boolean ;
       nom-rel-cob : relation nom-rel : char
                              nom-idcob : char end ;

/* la variable nom-rel-cob contient le nom de la relation qui identifie
   le c-objet qui a changé d'état */
       procedure analyse-objet-modifie ;
       begin

       nom-rel-cob:<(x.ms-nom-rel-cob,x.ms-id-cob):ms-cob <x >
       AND x.ms-id-cob = repère-de-c-objet . id-cob ;

/* nom-rel-cob a comme valeur le nom de la relation qui représente le
   c-objet qui a changé d'état */
       foreach nome in nom-rel-cob-do
               val-id-cob:=idcle (nome.nom-rel,repère-de-c-objet
                          val-att-cob) ;
               val-date-cob:=datecle (nome.nom-rel,repère-de-c-objet.
                             val-att-cob) ;
               val-att-cob-precedant:  (x):nome.nom-rel <x> AND
                             x.nome.nom-idcob PREDECESSOR val-id-cob ;
```

SPECIFICATION CONCEPTUELLE DES A-OPERATIONS IS-TYPE : C-OPERATION MODE : COPTEXTE	Page 2 de 17 Item 1 de 8

```
/* val-att-cob-précédant est le tuple de la relation "nome.nom-rel" dont
   la valeur de l'identifiant clé le caractérise comme celui qui précédait
   le tuple identifié par le repère-de-c-objet */

      procedure identification-evenements-associes ;
      begin  val-ms idev : <(y.ms-id-cevpred, y.ms-predinit,y.ms-predfin,
                             y.ms-nom-cevpre,x.ms-id-cevconstate) :
                  ms-cevpred   < y > AND ms-  cevconstate < x > AND
                  EXIST <y> (x.ms-id-cobmod=repère-de-c-objet.id-cob AND
             y.ms-id-cevpred = x.ms-id-cevpred)
      end ;

/* Cette procédure affecte à "val-ms idev" les nuples correspondant aux ca-
   ractéristiques des c-évènements associés au c-objet qui a "repère-de-c-
   objet.id-cob" comme nom de son identifiant clé */

      procedure evenement-predicats ;
      begin
      foreach element in val-ms idev do
      begin
      cod-pred-init :< ((x): element.nom-rel-cevpred < x > AND
                        x.element.nom-id-cevpred CURRENT),
                        element.nom-id-cevconstate)
      end end end ;

/* cette procédure affecte à "cod-pred-init-fin", pour chaque c-évènement
   de "ms-idcev", la tuple qui correspond à l'occurence la plus récente
   de la relation de mode cevpred de ce c-évènement*/

      procedure creation-repere-evènement (n-id-cev : char ; val-id-cevper :
                char ; val-dt-cevpred    : date) ;
      var ancien-rep-ev : relation anc-id-repcev : char end ;
      begin


      ancien-rep-ev := max ((x.id-rep-cev) : repère-de-c-évènement < x >)

          foreach rep in ancien-rep-ev do
          begin
                rep.anc-id-repcev := rep.anc-id-repcev + 1 ;
          insert (rep.anc-id-repcev, n-id-cev, val-id-cevper,val-dt-cevpred,
                  val-date-cob, repère-de-c-objet, id-rep-cob)
          in repère-de-c-évènement
          end end end ;

/* Cette procédure introduit un nouveau repère-de-c-évènement */
```

SPECIFICATION CONCEPTUELLE DES A-OPERATIONS IS-TYPE : C-OPERATION MODE : COPTEXTE	Page 3 de 17 Item 1 de 8

```
    procedure identification-evenement ;
     begin
      foreach predicat in cod-pred-init-fin until test = true do
       begin
        if    predicat.cod-pinit (val-att-cob-precedant)=true then
           if  predicat.cod-pfin (repere-de-c-objet.val-att-cob)=true then
             begin teste := true ;
                  creation-repere-evenement (predicat.desig-id-cev,
                                             predicat.sc-id-cv-per,
                                             predicat.sc-date-cev-pred)
            end
           end
          end
         end
        end
       end;

/* Cette procédure déclenche, pour chaque c-évènement de "cod-pred-pinit-
   pfin", l'exécution de ses prédicats initial jusqu'à ce qu'un c-évènement
   ait ses prédicats satisfaits.
   L'utilisation de la condition d'arrêt est due au fait qu'il y a au plus
   un c-évènement lié à un changement d'état d'un c-objet */

        begin
           teste := false ;
           analyse-objet-modifie ;
           identification-evenements-associes;
           evenements-predicats ;
           identification-evenement
        end ;

/* Ce bloc correspond au corps du texte de l'a-opération OP 1  */
```

SPECIFICATION CONCEPTUELLE DES A-OPERATIONS IS-TYPE : C-OPERATION MODE : COPTEXTE	Page 4 de 17 Item 2 de 8

```
recherche-operation-texte (id-op2, date-op2-texte, texte-op2)
    DOMAIN  id-op2          : CHAR ;
            date-op2-texte  : DATE ;
            texte-op2       : CHAR ;
    ASSERT
    reopte-1 : id-op2, date-op2-texte COMPOSKEY id-op2-texte ;
    reopte-2 : texte-op2 OPERATION (n-id-cop : char ; val-id-cop : char ;
                                    val-id-nep-cev : char):

/* les paramètres formels correspondent à ceux qui sont évalués pour le
   facteur de déclenchement de cette a-opération (facteur-ev2-op2). Ils
   sont le nom de l'identifiant clé de la c-opération, sa valeur et la
   valeur du repère-de-c-évènement à son origine */

    var ancien-rep-op : relation anc-id-repop : char end ;

    begin
    ancien-rep-op  :=max ((x.id-rep-cop) : repere-de-c-operation < x > );

/* on retrouve la valeur actuelle de l'identifiant clé de repère-de-c-
   opération */

    foreach rep in ancien-rep-op do
        begin
            rep.anc-id-repop := rep. anc-id-repop + 1 ;
        insert (rep.anc-id-repop, n-id-cop, val-id-cop,val-id-rep-cev)
        in  repère-de-c-opération  ;

        end
      end
      end

/* on introduit un nouveau tuple dans repère-de-c-opération  */
```

SPECIFICATION CONCEPTUELLE DES A-OPERATIONS
IS-TYPE : C-OPERATION
MODE : COPTEXTE


```
constatation-événement-texte (id-op3, date-op3-texte, texte-op3)
    DOMAIN  id-op3          : CHAR ;
            date-op3-texte  : DATE ;
            texte-op3       : CHAR :
    ASSERT
    constevt-1 : id-op3    , date-op3-texte COMPOSKEY id-op3-texte
    constevt-2 : texte-op3 OPERATION (repere-de-c-evenement) :
                 var val-repob relation nom-cevconstate : char ;
                                        val-att-cob : char ;
                                        nom-id-cob : char end
                 nom-rel-ob : relation nom-rel-cob : char end ;
/* nom-rel-ob indique le c-objet qui a changé d'état ; val-repob
   l'occurence qui a produit ce changement d'état, le nom de la relation
   de mode cevconstate du c-évènement associé au c-objet et le nom de l'i-
   dentifiant clé du c-objet */
                 val-id-cob : char ;
/* valeur de l'identifiant clé du c-objet */
                 val-rep-cevconstate : relation ancien-id-repevconst:char
                                       end
/* valeur de l'identifiant clé du repère-de-c-évènement-constate */
   begin
        val-repob : <(y.ms-id-cevconstate,x.val-att-cob,x.id-cob) :
            repere-de-c-objet < x > AND ms-cevconstate   < y > AND
                                (y.ms-id-cevconstate=repere-de-c-evenement.
            id-cevconstate AND x.id-rep-cob = repere-de-c-evenement.
            id-rep-cob) ;
/* on affecte une occurence à "val-repob" */
   foreach element in val-repob do
   nom-rel-ob : <(x.ms-nom-rel-cob) : ms-cob < x >AND
                 x.ms-id-cob = element.nom-id-cob ;
/* on affecte à nom-rel-ob le nom du c-objet qui a changé d'état */
   foreach nom in nom-rel-ob do
   val-id-cob := idcle (nom.nom-rel-cob, element.val-att-cob) ;
/* on affecte à "val-id-cob" la valeur de l'identifiant clé du c-objet
   qui a changé d'état */
        end
        insert (repere-de-c-evenement. val-id-cevper ,
                repere-de-c-évenement.val-date-cevpred,
                repere-de-c-evenement.val-date-cob,
                val-id-cob)
        in element.val-nom-cevconstate
       end
```

SPECIFICATION CONCEPTUELLE DES A-OPERATIONS IS-TYPE : C-OPERATION MODE : COPTEXTE	Page 6 de 17 Item 3 de 8

```
/* on introduit une nouvelle occurence dans la relation de mode cevcons-
   tate du c-évènement associé au c-objet qui a changé d'état */

   val-rep-cevconst := max ((x.id-rep-cevconstate) :
                repere-de-c-evenement-constate < x > )

      foreach rep in val-rep-cevconst do
      update (rep.anc-id-repevconst,
              repere-de-c-evenement. id-rep-cev)

      in repere-de-c-evenement-constate
      end

   end ;
/* on affecte a "val-rep-cevconst" la valeur de l'identifiant clé de la
   relation repère-de-c-évènement-constate et on met à jour la valeur
   qui lui est associée */
```

SPECIFICATION CONCEPTUELLE DES A-OPERATIONS IS-TYPE : C-OPERATION MODE : COPTEXTE	Page 7 de 17 Item 4 de 8

```
declenchement-operation-texte (id-op4, date-op4-texte,texte-op4)
  DOMAIN id-op4 : SAME OF déclenchement-opération-per ;
         date-op4-texte : DATE
         texte-op4      : CHAR
  ASSERT
  decopt-1 : id-op4, date-op4-texte COMPOSKEY id-op4-texte ;
  decopt-2 : texte-op4 OPERATION (repère-de-c-opération):

  var, facteur-dec : relation nom-fac-dec : char ;
                              nom-rel-dec : char ;
                              nom-id-cevdec : char end ;
       code-facteur-dec : relation cod-fac-dec : char end ;

/* la variable "facteur-dec" contient la description des noms du facteur
   de déclenchement, de la relation de mode cevdeclenche et de son iden-
   tifiant qui sont associés à la c-opération décrite par repère-de-c-
   opération. La variable "code-facteur-dec" contient l'indicatif du texte
   du facteur de déclenchement */

   texte-operation : relation nom-texte-op : char ;
                              nom-rel-optexte : char ;
                              nom-id-optexte : char end ;
   code-texte-op : relation val-id-coper      : char ;
                            val-date-coptexte : date ;
                            cod-op            : char end ;

/* la variable "texte-opération"contient la description des noms des at-
   tributs de la relation de mode coptexte de la c-opération décrite par
   repère-de-c-opération. La variable "code-texte-op" contient la valeur
   de l'identifiant clé de la relation de mode coptexte et l'indicatif du
   texte actuel de la c-opération */

       nom-id-obmod : relation id-obmod : char end ;
       id-ob-modif : char

/* ces variables contiennent le nom de l'identifiant clé du c-objet asso-
   cié à la c-opération décrite par repère-de-c-opération */
       op-val-att-ob : relation val-rep-att-ob : char end

/* la variable "val-att-ob" contient la valeur de l'occurence de c-objet
   associée au repère-de-c-objet à l'origine du c-évènement qui a conduit
   au déclenchement de la c-opération */

       rel-defini -facteur : relation att-rel-def-fact : char end ;

/* la variable "rel-définie-facteur" contient les nuples obtenus par
   l'évaluation du texte du facteur de déclenchement */

       val-att-obmod : relation val-att-cobmod : char end ;

/* cette variable contient le nuple du c-objet modifié par l'exécution de
   la c-opération décrite par repère-de-c-opération */
```

SPECIFICATION CONCEPTUELLE DES A-OPERATIONS IS-TYPE : C-OPERATION MODE : COPTEXTE	Page 8 de 17 Item 4 de

```
procedure retrouver-facteur ;
   begin facteur-dec : < (x.ms-facteur-declenche,x.ms-nom-cevdeclenche,
         x.ms-id-cevdeclenche)  = ms-c-ev-declenche < x > AND
         repere-de-c-évènement < y > AND ms-c-ev-constate < z > AND
         y.id-rep-cev = repere-de-c-operation-id-rep-cev AND
         EXIST < z > (y.id-cev = z.ms-id-cevconstate AND
         EXIST < x > (x.ms-id-cevper = z.ms-id-cevper AND
                      x.ms-id-copper = repère-de-c-opération. id-cop)
   end ;
/* cette procédure affecte à "facteur-dec" le nuple correspondant à la
   c-opération associée au c-évènement décrit par le repère-de-c-évène-
   ment associé au repère-de-c-opération */

   procedure retrouver-code-facteur ;
       begin
         foreach element in facteur-dec do
         code-facteur-dec : <(x.element.nom-fac-dec) :
                             element. nom-rel-dec < x > AND
                             x.element.nom-id-cevdec CURRENT
       end
    end ;
/* cette procédure affecte à "code-facteur-dec" l'indicatif du texte
   courant du facteur de déclenchement */

   procedure retrouver-texte-operation ;
   begin texte-operation : <(x.ms-texte-coptexte, x.ms-nom-rel-coptexte
         x.ms-id-coptexte) : ms-c-op-texte < x > AND x.ms-id-copper
         = repère-de-c-opération. id-cop
   end :
/* cette procédure affecte à "texte-opération" le nuple correspondant à
   la c-opération décrite par repère-de-c-opération */

   procedure retrouver-code-texte ;
   begin
       foreach élément in texte-opération do
         code-texte-op : <(x) : element.nom-rel-optexte < x > AND
                             x. element.nom-id-optexte CURRENT
       end end ;
/* cette procédure affecte à "code-texte-op" le nuple correspondant à la
   c-opération décrite par repère-de-c-opération */
```

SPECIFICATION CONCEPTUELLE DES A-OPERATIONS IS-TYPE : C-OPERATION MODE : COPTEXTE	Page 9 de 17 Item 4 de 8

```
procedure creation-repere-c-operation-declenche (n-id-cob, val-att-cob,
          val-date-coptexte)

var ancien-rep-opd : relation anc-id-repopd : char end ;
begin ancien-rep-op := max ((x,id-rep-copdec) : repere-de-c-operation-
      declenche < x > )
    foreach rep in ancien-rep-opd do
        begin
            rep.anc-id-repopd := rep.anc-id-repopd + 1 ;
        insert (rep.anc-id-repopd, n-id-cob, val-att-cob,val-date-coptexte,
                repère-de-c- operation. id-rep-cop)
        in repere-de-c-operation-declenchee ;
        end
      end
    end ;

/* cette procédure introduit un nouveau repère-de-c-opération-déclenche */

   procédure identification-objet-modifie (var nom-obmod : char)
        begin
          nom-id-obmod : <(x.ms-md-cob) : ms-copmodifie < x > AND
          ms-cob < y >AND x.ms-id-copper = repere-de-c-operation.id-cop AND
          EXIST < y >(y.ms-id-cob = x. ms-id-cob) ;

          foreach ob in nom-id-obmod do
                  nom-ob-mod := ob.rel-obmod
          end
        end ;

/* cette procédure retrouve le nom de l'identifiant clé du c-objet associé
   à la c-opération décrite par repère-de-c-opération */

   procédure recherche-val-ob
     begin
        op-val-att-ob : <(y.val-att-cob) : repere-de-c-objet < y > AND
          repere-de-c -evenement < z > AND
          z. id-rep-cev = repere-de-c-operation.id-rep-cev AND EXIST <y >
          (z. id-rep-cob = y. id-rep-cob) ;
   end ;

/* cette procédure affecte à "op-val-att-ob" le tuple correspondant au
   c-objet décrit par repère-de-c-objet */
```

SPECIFICATION CONCEPTUELLE DES A-OPERATIONS IS-TYPE : C-OPERATION MODE : C-OPTEXTE	Page 10 de 17 Item 4 de 8

```
procedure declencher-operation ;
 begin
      identification-ob-modifie (id-ob-modifie)
 foreach facteur-dec in code-facteur-dec do
      rel-defini -facteur := facteur-dec.cod-fac-dec (op-val-att-ob)
/* on affecte à "rel-defini -facteur" les nuples obtenus par l'évaluation
   de"facteur-dec.cod-fac-dec"
 foreach element-fact  in rel-defini-facteur do
/* on déclenche la c-opération autant de fois qu'il y a de tuples dans
   "rel-defini -facteur" */
   foreach oper in code-texte-op do
      oper.cod-op (element-fact, val-att-obmod)
/* il n'y a qu'un tuple en code-texte-op. La c-opération est déclenchée
   comme un appel de procédure et elle retourne la valeur du c-objet
   modifié par la variable "val-att-ob mod"  */
   foreach det-ob in val-att-ob mod do
/* on utilise le tuple du c-objet modifié pour créer un repère-de-c-
   opération-déclenche  */
      création-repere-c-operation-declenche (id-ob-modif,
                                    det-ob.val-att-cobmod,
                                    oper.val-date-coptexte)
      end
     end
    end
   end
  end ;
   begin
        retrouver-facteur ;
        retrouver-code-facteur ;
        retrouver-texte-operation ;
        retrouver-code-texte ;
        recherche-val-ob ;
        declenche-operation
   end
/* corps du texte de l'a-opération OP4  */
```

SPECIFICATION CONCEPTUELLE DES A-OPERATIONS IS-TYPE : C-OPERATION MODE : COPTEXTE	Page 11 de 17 Item 5 de 8

```
operation-modifie-texte (id-op5, date-op5-texte, texte-op5)
  DOMAIN id-op5          : SAME OF operation-modifie-per ;
         date-op5-texte  : DATE                          ;
         texte-op5       : CHAR                          ;
  ASSERT
  opmodt 1 : id-op5, date-op5-texte COMPOSKEY id-op5-texte ;
  opmodt 2 : texte-op5 OPERATION (repere-de-c-operation-declenche) :
         var
           ident-caract-opmod : relation nom-rel-opmod : char ;
                                         val-id-oppar  : char end ;

/* cette variable contient le nom de la relation de mode compodifie et la
   valeur de l'identifiant de la relation de mode copper qui le corres-
   pond */
           calc-val-id-cob   : char ;
           calc-val-date-cob : date ;

/* ces variables contiennent respectivement la valeur de l'identifiant
   clé et de la date de modification du c-objet modifié par la c-
   opération décrite par repère-de-c-opération-déclenchée */

           nom-rel-cobmod : relation  nom-rel-ob : char end ;

/* cette variable contient le nom de la relation de mode cob correspon-
   dant au c-objet modifié */

           val-rep-opmod relation anc-id-repopmod : char end ;

/* cette variable contient la valeur de l'identifiant clé actuel de
   repère-de-c-opération-modifie */

    begin
       ident-caract-opmod : <(x.ms-nom-copmodifie,y.val-id-cop) :
         ms-copmodifie   < x > AND repere-de-c-operation < y > AND
         y.id-rep-cop = repere-de-c-operation-declenche.id-rep-copAND
         EXIST < x> (x.ms-id-copper = y.id-cop)

/* cette expression retrouve le tuple qu'on affecte à "ident-caract-
   opmod" */
       nom-rel-cobmod :  <(x.ms-nom-rel-cob) : ms-cob < x > AND
            x.ms-id-cob=repère-de-c-operation-declenche.id-cobmod ;

/* cette expression retrouve le nom de la relation de mode cob modifiée */

    foreach nom in nom-rel-cobmod  do
       calc-val-id-cob := idcle (nom.nom-rel-ob ,
                                 repere-de-c-operation-déclenche.val-att-
                                 cobmod) ;
       calc-val-date-cob := datecle (nom.nom-rel-ob,
                                 repere-de-c-operation-declenche .val-att-
                                 cobmod) ;
    end ;

/* cette expression calcule les valeurs de l'identifiant clé du c-objet
   modifié et de la date de modification */
```

SPECIFICATION CONCEPTUELLE DES A-OPERATIONS IS-TYPE MODE : COPTEXTE	Page 12 de 17 Item 5 de 8

```
    foreach element in ident-caract-opmod do
    insert (element.val-id-opper,
            repere-de-c-operation-declenche.val-date-cop-texte,
            calc-val-date-cob,
            calc-val-id-cob)

    in  element.nom-rel-opmod
    end

/* on introduit un nouveau tuple dans la relation de mode copmodifie
   désignée par "element.nom-rel-opmod".*/

    val-rep-opmod := max ((x.id-rep-copmodifie) : repere-de-c-operation-
    modifie < x>)

/* on retrouve la valeur de l'identifiant clé actuel de repère-de-c-
   opération-modifié */

    foreach rep in val-rep-opmod do
      repdate (rep.anc-id-repopmod,
               repere-de-c-operation-declenche.id-rep-copdec)
      in repere-de-c-operation-modifie
    end
  end ;
/* on actualise repère-de-c-opération-modifie */
```

SPECIFICATION CONCEPTUELLE DES A-OPERATION IS-TYPE : C-OPERATION MODE : COPTEXTE	Page 13 de 17 Item 6 de 8

```
declenchement-effectif-texte (id-op6, date-op6-texte, texte-op6)
  DOMAIN id-op6         : SAME OF déclenchement-effectif-per ;
         date-op6-texte : DATE
         texte-op6      : CHAR
  ASSERT
  deceft-1 :id-op6, date-op6-texte COMPOSKEY id-op6-texte ;
  deceft-2 :texte-op6 OPERATION (repère-de-c-operation-declenche) :
           var id-caract-deceff : relation nom-rel-cevdeceff : char ;
                                           val-ident-cevper  : char ;
                                           val-date-cevpred  : date ;
                                           val-date-cob      : date ;
                                           val-id-copper     : char ;
                                  end ;
/* cette variable contient le nom de la relation de mode cevdeceff que l'on
   va actualiser, les valeurs de l'identifiant clé de la relation de mode
   cob modifie associée et la valeur de l'identifiant clé de la relation de
   mode copper associé */
           val-rep-evdec : relation anc-id-repevdec : char end ;
/* cette variable contient la valeur de l'identifiant clé actuel de repère-
   de-c-évènement-déclenche  */
           calc-val-date-ob : date ;
           nom-rel-cob-mod  : relation nom-rel-ob : char end ;
/* ces variables contiennent respectivement la date de modification du c-
   objet et le nom de la relation qui le décrit */
    begin
        id-caract-deceff : <(z.ms-nom-rel-cevdeceff,x.val-id-cevper,
                             x.val-date-cevpred, x.val-date-cob,
                             y.val-id-cop) :
        repere-de-c-evenement <x >AND repere-de-c-operation <y >AND
        ms-cevdeceff  <z > AND
        y.id-rep-cop = repere-de-c-operation-declenche.id-rep-cop
        AND EXIST <x >  (x.id-rep-cev = y.id-rep-cev  AND EXIST <z >
        (z.ms-id-cev = x.id-cev AND
         z.ms-id-copper = y.id-cop))
/* cette expression affecte un nuple à "id-caract-deceff" */
    nom-rel-cobmod : <(x.ms-nom-rel-cob) ms-cob <x >AND
           x.ms-id-cob=repere-de-c-operation-declenche.id-cobmod ;
    foreach nom in nom-rel-cob-mod do
        calc-val-date-cob := datecle  (nom.nom-rel-ob,
                  repere-de-c-operation-declenche.val-att-cobmod)
    end
/* ces expressions permettent l'évaluation de la date de modification du
   c-objet */
    foreach rep in id-caract-deceff do
    insert  (rep.val-ident-cevper, rep.val-date-cevpred,
             rep.val-date-cob, rep-val-id-copper,
             repere-de-c-operation-declenche.val-date-coptexte
             calc-val-date-cob)
    in rep. nom-rel-cevdeceff  end ;
```

SPECIFICATION CONCEPTUELLE DES A-OPERATIONS IS-TYPE : C-OPERATION MODE : COPTEXTE	Page 14 de 17 Item 6 de 8

```
/* cette expression introduit un tuple dans la relation de mode cevdeceff
   désignée par "rep. nom-rel-cevdeceff" */

     val-rep-evdec := max ((x.id-rep-cevdeclenche) : repere-de-c-évenement-
                      declenche < x > )

/* on retrouve la valeur de l'identifiant clé actuel de repère-de-c-
   évènement-déclenche */

     foreach rep in val-rep-evdec do
     update (rep.anc-id-repevdec,
             repere-de-c-operation-declenche . id-rep-copdec)
     in repere-de-c-evènement-declenche
     end
   end ;

/* on actualise repère-de-c-évènement-déclenche */
```

SPECIFICATION CONCEPTUELLE DES A-OPERATIONS IS-TYPE : C-OPERATION MODE : COPTEXTE	Page 15 de 17 Item 7 de 8

```
modification-objet-texte (id-op7, date-op7-texte, texte-op7)
  DOMAIN  id-op7          : SAME OF modification-objet-per ;
          date-op7-texte  : DATE                           ;
          texte-op7       : CHAR                           ;
  ASSERT
  moobt 1 : id-op7, date-op7-texte COMPOSKEY id-op7-texte ;
  moobt 2 : texte-op7 OPERATION (repère-de-c-objet)
            var
              ident-rel-cob : relation nom-rel-cob : char end ;

/* cette variable contient le nom de relation de mode cob qui décrit le
   c-objet modifié */
            val-rep-obmod : relation anc-id-repobmod : char end ;

/* cette variable contient la valeur actuelle de l'idendifiant clé de
   repère-de-c-objet-modifié */

    begin
        ident-rel-cob : < (x.ms-nom-cobvar) :  ms-cob-var < x > AND
                    x.ms-id-cob     = repère-de-c-objet . idcob

/* on retrouve le nom du c-objet */

    foreach relat-ob in ident-rel-cob do

    insert
         repere-de-c-objet . val-att-cob)

    in relat-ob . nom-rel-ob
    end

/* on introduit un tuple dans la relation de mode cob désignée par
   "relat-ob . nom-rel-ob */
         val-rep-obmod := max ((x.id-op-cobmodifie) : repère-de-c-objet-
                                  modifie < x >)

/* on retrouve la valeur de l'identifiant clé actuel de repère-de-c-objet
   modifié */

   foreach rep in val-rep-obmod do
   update (rep. anc-id-repobmod,
           repere-de-c-objet . id-rep-cob)

   in repere-de-c-objet-modifie

   end

/* on actualise repère-de-c-objet-modifié */
```

SPECIFICATION CONCEPTUELLE DES A-OPERATIONS IS-TYPE : C-OPERATION MODE : COPTEXTE	Page 16 de 17 Item 7 de 8

```
modification-objet-texte (id-op7,date-op7-texte, texte-op7)
  DOMAIN  id-op7          : SAME OF modification-objet-per ;
          date-op7-texte  : DATE                           ;
          texte-op7       : CHAR                           ;
  ASSERT
  moobt 1 : id-op7, date-op7-texte COMPOSKEY id-op7-texte ;
  moobt 2 : texte-op7 OPERATION (repère-de-c-objet)
             var
                 ident-rel-cob : relation nom-rel-cob : char end ;
/* cette variable contient le nom de relation de mode cob qui décrit le
   c-objet modifié */
          val-rep-obmod : relation anc-id-repobmod : char end ;
/* cette variable contient la valeur actuelle de l'identifiant clé de
   repère-de-c-objet-modifié */
     begin
          ident-rel-cob : <(x.ms-nom-cobvar) : ms-cob -var <x> AND
                   x.ms-id-cob    = repere-de-c-objet . id-cob
/* on retrouve le nom du c-objet */
     foreach relat-ob in ident-rel-cob do
     insert
          (repère-de-c-objet . val-att-cob)
     in relat-ob. nom-rel-ob
     end
/* on introduit un tuple dans la relation de mode cob désignée par
   "relat-ob . nom-rel-ob" */
     val-rep-obmod := max ((x.id-rep-cobmodifie) : repere-de-c-objet-
                           modifie <x>)
/* on retrouve la valeur de l'identifiant clé actuel de repère-de-c-
   objet-modifié */
     foreach rep in val-rep-ob mod do
     update (rep. anc-id-repobmod,
             repere-de-c-objet.id-rep-cob)
     in repere-de-c-objet-modifie
     end
/* on actualise repère-de-c-objet-modifie */
```

SPECIFICATION CONCEPTUELLE DES A-OPERATIONS IS-TYPE : C-OPERATION MODE : COPTEXTE	Page 17 de 17 Item 8 de 8

```
reconnaissance-objet-texte (id-op8, date-op8-texte, texte-op8)
   DOMAIN id-op8          : SAME OF reconnaissance-objet-per ;
          date-op8-texte  : DATE                             ;
          texte-op8       : CHAR                             ;
   ASSERT
   reobt 1 : id-op8, date-op8-texte COMPOSKEY id-op8-texte ;
   reobt 2 : texte-op8 OPERATION (repere-de-c-operation-texte) :
             var
                val-repob : relation anc-id-repob : char end ;

             begin
             val-repob := max ((x.id-rep-ob) : repere-de-c-objet < x >

/* on retrouve la valeur actuelle de l'identifiant de repere-de-c-
   objet */
             foreach rep in val-repob do
              rep.anc-id-repob := rep.anc-id-repob + 1 ;
             insert (rep.anc-id-repob,
                     repere-de-c-operation-declenchée . id-cob
                     repere-de-c-operation-declenchée . val-att-cob
             in repere-de-c-objet
             end
          end ;
/* on introduit une nouvelle occurence en repère-de-c-objet */
```

SPECIFICATION CONCEPTUELLE DES A-OPERATIONS IS-TYPE : C-OPERATION MODE : COPMODIFIE	Page 1 de 2

```
IS-TYPE C-OPERATION MODE COPMODIFIE
   reconnaissance-evenement-modifie (id-op1-texte, date-op1-modifie,
                                     id-rep-cev)
     DOMAIN id-op1-texte     : SAME OF reconnaissance-évènement-texte ;
            date-op1-modifie : DATE                                   ;
            id-rep-cev       : SAME OF repère-de-c-évènement          ;
     ASSERT re1 : id-op1-texte, date-op1-modifie COMPOSKEY id-op1-modifie;
            re2 : date-op1-modifie : DATECRE ;
   recherche-operation-modifie (id-op2-texte, date-op2-modifie,id-rep-cop)
     DOMAIN id-op2-texte     : SAME OF recherche-opération-texte ;
            date-op2-modifie : DATE                              ;
            id-rep-cop       : SAME OF repere-de-c-operation     ;
     ASSERT ro1 : id-op2-texte, date-op2-modifie COMPOSKEY id-op2-modifie;
            ro2 : date-op2-modifie : DATECRE ;
   constatation-evenement-modifie (id-op3-texte,date-op3-modifie,
                                   id-rep-cevconstate)
     DOMAIN id-op3-texte       : SAME OF constatation-evènement-texte ;
            date-op3-modifie   : DATE                                 ;
            id-rep-cevconstate : SAME OF repere-de-c-évènements-constate;
     ASSERT ce1 : id-op3-texte, date-op3-modifie COMPOSKEY id-op3-modifie;
            ce2 : date-op3-modifie : DATECRE ;
   déclenchement-operation-modifie (id-op4-texte,date-op4-modifie,
                                    id-rep-copdec)
     DOMAIN id-op4-texte     : SAME OF déclenchement-opération-texte ;
            date-op4-modifie : DATE                                  ;
            id-rep-opdec     : SAME OF repere-de-c-operation-declenchee;
     ASSERT dom1 : id-op4-texte,date-op4-modifie COMPOSKEY id-op4-modifie;
            dom2 : date-op4-modifie : DATECRE ;
   operation-modifie-modifie (id-op5-texte,date-op5-modifie,
                              id-rep-copmodifie)
     DOMAIN id-op5-texte       : SAME OF operation-modifie-texte ;
            date-op5-modifie   : date                            ;
            id-rep-copmodifie : SAME OF repere-de-c-operation-modifie ;
     ASSERT om1 : id-op5-texte,date-op5-modifie COMPOSKEY id-op5-modifie ;
            om2 : date-op5-modifie : DATECRE .
   declenchement-effectif-modifie (id-op6-texte,date-op6-modifie,
                                   id-rep-cevdeclenche)
     DOMAIN id-op6-texte        : SAME OF declenchement-effectif-texte ;
            date-op6-modifie    : DATE                                 ;
            id-rep-cevdeclenche : SAME OF repere-de-c-evenement-declenche;
     ASSERT de-1 : id-op6-texte, date-op6-modifie COMPOSKEY id-op6-modifie ;
            de-2 : date-op6-modifie : DATECRE ;
   modification-objet-modifie (id-op7-texte, date-op7-modifie,
                               id-rep-cob modifie)
     DOMAIN id-op7-texte      :SAME OF modification-objet-texte ;
            date-op7-modifie  :DATE                             ;
            id-rep-cobmodifie :SAME OF repere-de-c-objet-modifie ;
     ASSERT mo-1 : id-op7-texte,date-op7-modifie COMPOSKEH id-op7-modifie ;
            mo-2 : date-op7-modifie : DATECRE ;
```

SPECIFICATION CONCEPTUELLE DES A-OPERATIONS IS-TYPE : C-OPERATION MODE : COPMODIFIE	Page 2 de 2

```
reconnaissance-objet-modifie (id-op8-texte, date-op8-modifie,
                              id-rep-cob)
  DOMAIN id-op8-texte     : SAME OF reconnaissance-objet-texte ;
         date-op8-modifie : DATE                               ;
         id-rep-cob       : SAME OF repère-de-c-objet ;
  ASSERT ro 1 : id-op8-texte, date-op8-modifie COMOSKEY id-op8-modifie ;
         ro 2 : date-op8-modifie : DATECLE ;
```

SPECIFICATION CONCEPTUELLE DES A-EVENEMENTS IS-TYPE : CEVENT MODE : CEVPER	Page 1 de 1

```
IS-TYPE CEVENT MODE CEVPER
   creation-rep-ob-per (id-ev 1, date-cre-ev 1)

     DOMAIN  id-ev 1        : CHAR ;
             date-cre-ev 1  : CHAR ;

     ASSERT crpob 1 : id-ev 1 KEY :

   creation-rep-ev-per (id-ev 2, date-cre-ev 2)

     DOMAIN id-ev 2       : CHAR ;
            date-cre-ev2  : CHAR ;

     ASSERT crpev 1 : id-ev2 KEY ;

creation-rep-op-per (id-ev3, date-cre-ev 3)

     DOMAIN id-ev3        : CHAR ;
            date-cre-ev3  : CHAR ;

     ASSERT crpop 1 : id-ev3 KEY .

création-rep-opdec-per (id-ev4, date-cre-ev4)

     DOMAIN id-ev4        : CHAR ;
            date-cre-ev4  : CHAR ;

     ASSERT cropdec 1 : id-ev4 KEY ;
```

SPECIFICATION CONCEPTUELLE DES A-EVENEMENTS IS-TYPE : CEVENT MODE : CEVPRED	Page 1 de 2

```
IS-TYPE CEVENT MODE CEVPRED
   creation-rep-ob-pred (id-ev1-per, date-ev1-pred, pinit,pfin-ev1 )

     DOMAIN id-ev1-per    : SAME OF creation-rep-ob-per ;
            date-ev1-pred: date ;
            pinit-ev1     : char ;
            pfin-ev1      : char ;

     ASSERT crobpred1 : id-ev1-per, date-ev1-pred COMPOSKEY id-ev1-pred ;
            crobpred2 : pinit-ev1 : INITIAL ;
            crobpred3 : pfin-ev1 :  FINAL ;
            crobpred4 : pinit-ev1 PREDICATE (repère-de-c-objet) :
                  begin
                       pinit-ev1:=true
                  end ;
            crpbpred5 : pfin-ev1 PREDICATE (repère-de-c-objet) :
                  begin
                       pfin-ev1 := true
                  end ;


   creation-rep-ev-pred (id-ev2-per, date-ev2-pre, pinit-ev2,pfin-ev2)

     DOMAIN id-ev2-per     : SAME OF creation-rep-ev-per ;
            date-ev2-pred : date                        ;
            pinit-ev2      : char .                     ;
            pfin-ev2       : char                       ;

     ASSERT crevpred_1 : id-ev2-per, date-ev2-pred COMPOSKEY id-ev2-pred ;
            crevpred-2 : pinit-ev2 : INITIAL ;
            crevpred-3 : pfin-ev2  : FINAL ;
            crevpred-4 : pinit-ev2 PREDICATE (repère-de-c-évènement) :
                         begin
                              pinit-ev2 :=true
                         end ;
            crevpred-5 : pfin-ev2 PREDICATE (repère-de-c-évènement) :
                         begin
                              pfin-ev2:= true
                         end ;
   creation-rep-op-pred (id-ev3-per,date-ev3-pred, pinit-ev3, pfin-ev3)

     DOMAIN id-ev3-per     : SAME OF creation-rep-op-per ;
            date-ev3-pred : date                        ;
            pinit-ev3      : char                       ;
            pfin-ev3       : char                       ;
```

SPECIFICATION CONCEPTUELLE DES A-EVENEMENTS IS-TYPE : CEVENT MODE : CEVPRED	Page 2 de 2

```
ASSERT croppred-1 : id-ev3-per, date-ev3-pred COMPOSKEY id-ev3-pred;
       croppred-2 : pinit-ev3 : INITIAL ;
       croppred-3 : pfin-ev3 : FINAL ;
       croppred-4 : pinit-ev3 PREDICATE (repère-de-c-opération) :
                    begin
                        pinit-ev3 :=true
                    end ;
       croppred-5 : pfin-ev3 PREDICATE (repère-de-c-opération) :
                    begin
                        pfin-ev3 :=true
                    end ;

creation-rep-opdec-pred (id-ev4-per, date-ev4-pred, pinit-ev4, pfin-ev4)

DOMAIN id-ev4-per    : SAME OF creation-rep-opdec-per ;
       date-ev4-pred : date                           ;
       pinit-ev4     : char                           ;
       pfin-ev4      : char                           ;

ASSERT cropdecpred-1 : id-ev4-per, date-ev4-pred COMPOSKEY
                       id-ev4-pred ;
       cropdecpred-2 : pinit-ev4 : INITIAL ;
       cropdecpred-3 : pfin-ev4 : FINAL ;
       cropdecpred-4 : pinit-ev4 PREDICATE (repère-de-c-opération-
                                            déclenche) :
                    begin
                        pinit-ev4 := true
                    end ;
       cropdecpred 5 : pfin-ev4 PREDICATE (repère-de-c-opération-
                                           declenche) :
                    begin
                        pfin-ev4:=true
                    end ;
```

SPECIFICATION CONCEPTUELLE DES A-EVENEMENTS IS-TYPE : CEVENT MODE : CEVCONSTATE	Page 1 de 1

```
IS-TYPE CEVENT MODE CEVCONSTATE

   creation-rep-ob-constate (id-ev1-pred, date-ev1-const, id-rep-cob)

     DOMAIN id-ev1-pred    : SAME OF creation-rep-ob-pred ;
            date-ev1-const : DATE                         ;
            id-rep-cob     : SAME OF repère-de-c-objet    ;
     ASSERT evct 1 : id-ev1-pred, date-ev1-const COMPOSKEY id-ev1-const ;

   creation-rep-ev-constate (id-ev2-pred, date-ev2-const,id-rep-cev)

     DOMAIN id-ev2-pred    : SAME OF creation-rep-ev-pred ;
            date-ev2-const : DATE                         ;
            id-rep-cev     : SAME OF repère-de-c-évènement;
     ASSERT evct 2 : id-ev2-pred, date-ev2-const COMPOSKEY id-ev2-const ;

   creation-rep-op-constate (id-ev3-dred, date-ev3-const, id-rep-cop)

     DOMAIN id-ev3-pred    : SAME OF creation-rep-op-pred ;
            date-ev3-const : DATE                         ;
            id-rep-cob     : SAME OF repère-de-c-opération ;
     ASSERT evct 3 : id-ev3-pred, date-ev3-const COMPOSKEY id-ev3-const ;

   creation-rep-opdec-constate (id-ev4-pred, date-ev4-const, id-rep-copdec)

     DOMAIN id-ev4-pred    : SAME OF creation-rep-opdec-pred ;
            date-ev4-const : DATE                           .
            id-rep-opdec   : SAME OF repère-de-c-opération-déclenche ;
     ASSERT evct 4 : id-ev4-pred, date-ev4-const COMPOSKEY id-ev4-const ;
```

SPECIFICATION CONCEPTUELLE DES A-EVENEMENTS IS-TYPE : CEVENT MODE : CEVDECLENCHE	Page 1 de 7

```
IS-TYPE CEVENT MODE CEVDECLENCHE
cre-rep-ev-dec (id-ev1-per, id-op1-per, date-ev1 op1-dec, cond-ev1-op1,
                facteur-ev1-op1)
   DOMAIN id-ev1-per      : SAME OF création-rep-ob-per ;
          id-op1-per      : SAME OF reconnaissance-évènement-per ;
          date-ev1op1-dec : date ;
          cond-ev1-op1    : char ;
          facteur-ev1-op1 : char ;
   ASSERT crerevdec-1 : id-ev1-per, id-op1-per, date-ev1 op1-dec COMPOSKEY
                        id-ev1 op1-dec ;
          crerevdec-2 : facteur-ev1-op1 FACTOR (repere-de-c-objet) : ACTUAL;
          crerevdec-3 : cond-ev1-op1 CONDITION (repere-de-c-objet) :
            var val-msidev : relation nom-id-cevpred : char ; nom-pinit:char;
                                      nom-pfin : char ;
                                      nom-rel-cevpred : char ;
                                      nom-id-cevconstate : char end ;
/* la variable val-ms idev contient des attributs qui correspondent aux
   c-évènements qui peuvent être associés à un changement d'état d'un
   c-objet */
     cod-pred-init-fin : relation sc-id-cevper : char ;
                                  sc-date-cevpred : date ;
                                  cod-pinit : char ;
                                  cod-pfin : char ;
                                  desig-id-cev : char end ;
/* la variable cod-pred-init-fin contient les nuples qui contiennent les
   indicatifs des textes actuels des prédicats associées aux noms de pré-
   dicats de la variable "val-msidev" */

     val-att-cob-précedant : relation pval-at : char end ;
/* la variable val-att-cob-précédant contient la valeur de l'occurence du
   c-objet avant le changement d'état */
     val-id cob : char ;
     texte : boolean
     nom-rel-cob : relation nom-rel : char ;
                            nom-idcob : char end ;
/* la variable nom-rel-cob contient le nom de la relation qui identifie le
   c-objet qui a changé d'état */
     procedure analyse-objet-modifie ;
     begin
     nom-rel-cob <(x.ms-nom-rel-cob, x.ms-id-cob) : ms-cob <x >
     AND x. ms-id-cob = repere-de-c-objet . id-cob ;
/* nom-rel-cob a comme valeur le nom de la relation qui représente le
   c-objet qui a changé d'état */
          foreach nome in nom-rel-cob do
                  val-idcob := idcle (nome.nom-rel, repere-de-c-objet.
                               val-att-cob) ;
            x.nome-nom-idcob PREDECESSOR val-idcob
          end end end ;
```

SPECIFICATION CONCEPTUELLE DES A-EVENEMENTS IS-TYPE : CEVENT MODE : CEVDECLENCHE	Page 2 de 7

```
/* val-att-cob-précédant est le tuple de la relation "nome-nom-rel" dont la
   valeur de l'identifiant clé le caractérise comme celui qui précédait le
   tuple identifié par le repère-de-c-objet */

     procedure identification-evenements-associes ;
     begin val-msidev : <(y.ms-id-cevpred, y.ms-pinit, y.ms-pfin ;
                          y.ms-nom-cevpred, x.ms-id-cev constate) :
               ms-cevpred   <y> AND ms-cevconstate   <x> AND
               EXIST <y> (x.ms-id-cobmod = repère-de-c-objet.id-cob AND
           y.ms-id-cevpred = x.ms-id-cevpred)
     end ;

/* cette procédure affecte à "val-msidev" les nuples correspondant aux ca-
   ractéristiques des c-évènements associés au c-objet qui a "repère-de-c-
   objet . id-cob" comme nom de son identifiant clé */

     procedure évènements-predicats ;
      begin
       foreach element in val-ms idev do
        begin
          cod-pred-init-fin : + (( (x) : element.nom-rel-cevpred < x > AND
                                  x.element-nom-id-cevpred CURRENT),
                                  element.nom-id-cevconstate)
        end end end ;

/* cette procédure affecte à "cod-pred-init-fin", pour chaque c-évènement
   de "ms-idcev", le tuple qui correspond à l'occurence la plus récente de
   la relation de mode cevpred de ce c-évènement */

     begin
          teste := false ;
          cond-evl-opl := false ;
          analyse-objet-modifie ;
          identification-evenement-associes ;
           évènement-predicats ;
          foreach predicat in cod-pred-init-fin until teste = true do
          begin
             if predicat . cod-pinit (val-att-cob-precedant) = true
             then if predicat.cod-pfin (repere-de-c-objet.val-att-cob)
                                        = true
                  then cond-evl-opl := true
   end end end end end ;

/* corps du texte de la condition de déclenchement cond-evl opl. La condi-
   tion est vrai si les prédicats qui définissent un c-évènement associé
   au c-objet qui a changé d'état sont vrais */
```

SPECIFICATION CONCEPTUELLE DES A-EVENEMENTS IS-TYPE : CEVENT MODE : CEVDECLENCHE	Page 3 de 7

```
cre-rep-op-dec (id-ev2-per, id-op2-per, date-ev2 op2-dec, cond-ev2-op2,
                facteur-ev2-op2)
    DOMAIN id-ev2-per       : SAME OF création-rep-ev-per ;
           id-op2-per       : SAME OF recherche-opération-per ;
           date-ev2 op2-dec : date ;
           cond-ev2-op2     : char ;
           facteur-ev2-op2  : char ;
    ASSERT
    crereopd-1 : id-ev2-per, id-op2-per, date-ev2 op2-dec COMPOSKEY
                 id-ev2 op2-dec ;
    crereopd-2 : facteur-ev2-op2 FACTOR repère-de-c-évènement) :

    var val-msop : relation nom-id-cevdec : char ;
                            nom-rel-evdec : char ;
                            nom-conddec : char ;
                            nom-id-copper : char end ;

/* la variable "val msop" contient des attributs qui correspondent aux
   c-opérations qui sont associées au c-évènement identifié par repère-
   de-c-évènement */

    sc-cev-dec : relation val-id-cevper : char ;
                          val-id-copper : char ;
                          val-conddec   : char ;
                          desig-id-copper : char end ;

/* la variable "sc-cev-dec" contient les nuples qui contiennent les indica-
   tifs des textes actuels des conditions de déclenchement donnés par "val-
   msop" */

    val-c-ob : relation n-val-ob : char end ;

    procedure identification-operation-associees ;
    begin  val-msop : <(x.ms-id-cevdeclenche, x.ms-condition
                        x.ms-nom-cevdeclenche, x.ms-id-copper ):
                       ms-cevdeclenche   <x > AND ms-cevconstate   < y > AND
                       EXIST < x > (y.ms-id-cevconstate=repere-de-c-evenement.
           id-cevconstate AND x.ms-id-cevper = y.ms-id-cevper)
    end ;

/* cette procédure affecte à "val-msop" les nuples correspondants aux carac-
   téristiques des relations de mode cevdeclenche associées au c-évènement
   qui a "repère-de-c-évènement. id-cevconstate" comme nom de l'identifiant
   clé de sa relation de mode cevconstate */
      procedure condition-declenche ;
        foreach element in val-msop do
          begin
               sc-cev-dec:<(((x):element.nom-rel-evdec < x > AND
                             x.element.nom-id-cevdec CURRENT), element.nom-
                                                               id-copper)
          end ;
```

SPECIFICATION CONCEPTUELLE DES A-EVENEMENTS
IS-TYPE : CEVENT
MODE : CEVDECLENCHE


```
/* cette procédure affecte à "sc-cev-dec", pour chaque tuple de "val-msop",
   le tuple qui correspond à l'occurence actuelle de la relation de mode
   cevdeclenche identifiée par "val-msop.nom-rel-evdec" */

     procedure recherche-val-ob ;
        val-c-ob : <(x.val-att-cob) : repere-de-c-objet < x > AND
                     x. id-rep-cob = repere-de-c-evènement.id-rep-cob ;

/* cette procédure affecte à val-c-ob le nuple correspondant au c-objet
   identifié par le repère-de-c-objet qui a "repère-de-c-évènement.id-rep-
   cob" comme identifiant clé */

     begin
          identification-operations-associees ;
          condition-declenche ;
          recherche-val-ob ;

             foreach condition in sc-cev-dec do
             if  condition-val-conddec (val-c-ob) = true then
             facteur-ev2-op2 :< (condition.desig-id-copper,
                                 condition.val-id-copper,
                                 repère-de-c-évènement.id-rep-cev)
     end
   end
 end ;

/* corps du texte du facteur de déclenchement "facteur-ev2 op2". Il y aura
   autant d'occurences dans facteur-ev2 op2 qu'il y a de c-opérations dont
   les conditions de déclenchement sont satisfaites. Chaque occurence con-
   tient les valeurs nécessaires à l'exécution de OP2 */

   crereopd-3 :    cond-ev2-op2    CONDITION (repere-de-c-evènement) :

   var val-msop : relation nom-id-cevdec : char ;
                           nom-rel-evdec : char ;
                           nom-conddec :   char ;
                           nom-id-copper : char end ;

/* la variable "val-msop" contient des attributs qui correspondent aux
   c-opérations qui sont associées au c-évènement identifié par repère-
   de-c-évènement */

    sc-cev-dec  : relation val-id-cevper : char ;
                           val-id-copper : char ;
                           val-conddec   : char ;
                           desig-id-copper : char end ;

/* la variable "sc-cev-dec" contient les nuples qui contiennent les indi-
   catifs des textes actuels des conditions de déclenchement donnés par
   "val-msop" */

         teste : boolean ;
```

SPECIFICATION CONCEPTUELLE DES A-EVENEMENTS IS-TYPE : CEVENT MODE : CEVDECLENCHE	Page 5 de 7

```
   val-c-ob : relation n-val-ob : char end ;

procédure identification-opération-associes ;
 begin  val-msop : <(x.ms-id-cevdeclenche, x.ms-condition-declenche,
                     x.ms-nom-cevdeclenche, x.ms-id-copper) :
                  ms-cevdeclenche  <x> AND ms-cevconstate  <y> AND
                  EXIST(x)(y.ms-id-cevconstate = repère-de-c-évènement.
        id-cevconstate AND x.ms-id-cevper = y.ms-id-cevper)
 end ;

/* cette procédure affecte à "val-msop" les nuples correspondantes aux
   caractéristiques des relations de mode cevdeclenche associées au c-
   évènement qui a "repère-de-c-évènement. id-cevconstate" comme nom de
   l'identifiant clé de sa relation de mode cevconstate */

procédure condition-declenche
   foreach element in val-msop do
     begin
         sc-cev-dec:< (((x) : element.nom-rel-evdec <x> AND
                      x.element.nom-id-cevdec CURRENT), element-nom-id-copper)
     end ;

/* cette procédure affecte à "sc-cev-dec", pour chaque tuple de "val-msop",
   le tuple qui correspond à l'occurence actuelle de la relation de mode
   cevdeclenche identifiée par "val-msop.nom-rel-evdec" */

procédure recherche-val-ob ;
     val-c-ob : <(x.val-att-cob) : repere-de-c-objet <x> AND
                  x.id-rep-cob = repère-de-c-evenement.id-rep-cob ;

/* cette procédure affecte à val-c-ob le nuple correspondant au c-objet
   identifié par le repère-de-c-objet qui a "repère-de-c-évènement.id-
   rep-cob" comme identifiant clé */

   begin
        teste:=false ; cond-ev2-op2 :=false ;
        identification-operations-associees ;
        condition-declenche ;
        recherche-val-ob ;
        foreach condition in sc-cev-dec until teste=true do
            if  condition.val-conddec (val-c-ob) = true
           then teste:=true ;
                cond-ev2-op2:=true
           end end end ;

/* corps du texte de la condition de déclenchement "cond-ev2op2". La con-
   dition est vraie s'il y a au moins une c-opération dont la condition
   de déclenchement est satisfaite */
```

SPECIFICATION CONCEPTUELLE DES A-EVENEMENTS
IS-TYPE : CEVENT
MODE : CEVDECLENCHE


```
cre-rep-ev-const-dec (id-ev2-per,id-op3-per,date-ev2op3-dec,cond-ev2op3,
                      facteur-ev2op3)
   DOMAIN id-ev2-per       : SAME OF création-rep-ev-per ;
          id-op3-per       : SAME OF constatation-evenement-per ;
          date-ev2op3-dec  : date ;
          cond-ev2op3      : char ;
          facteur-ev2op3   : char ;
   ASSERT
   crerevcod-1 : id-ev2-per,id-op3-per,date-ev2op3-dec COMPOSKEY
                 id-ev2op3-dec;
   crerevcod-2 : cond-ev2op3 CONDITION (repere-de-c-evenement) :
                 begin
                      cond-ev2op3:=true
                 end ;
   crerevcod-3 : facteur-ev2op3 FACTOR (repere-de-c-evenement): ACTUAL ;

cre-rep-opdec-dec (id-ev3-per,id-op4-per,date-ev3op4-dec,cond-ev3op4,
                   facteur-ev3op4)
   DOMAIN id-ev3-per       : SAME OF création-rep-op-per ;
          id-op4-per       : SAME OF declenchement-operation-per ;
          date-ev3op4-dec  : date                                ;
          cond-ev3op4      : char                                ;
          facteur-ev3op4   : char                                ;
   ASSERT
   crereopdd-1 : id-ev3-per,id-op4-per,date-ev3op4-dec COMPOSKEY
                 id-ev3op4-dec
   crereopdd-2 : cond-ev3op4 CONDITION (repere-de-c-opération) :
                 begin
                      cond-ev3op4:=true
                 end ;
   crereopdd-3 : facteur-ev3op4 FACTOR (repere-de-c-opération) : ACTUAL ;

cre-rep-opmod-dec (id-ev4-per, id-op5-per, date-ev4op5,cond-ev4op5,
                   facteur-ev4op5)
   DOMAIN id-ev4-per       : SAME OF création-rep-opdec-per ;
          id-op5-per       : SAME OF operation-modifie-per ;
          date-ev4op5-dec  : date ;
          cond-ev4op5      : char ;
          facteur-ev4op5   : char ;
   ASSERT
   crereopmodec-1 : id-ev4-per, id-op5-per,date-ev4op5-dec COMPOSKEY
                    id-ev4op5-dec ;
   crereopmodec-2 : cond-ev4op5 CONDITION (repere-de-c-opération-declenche):
                    begin
                         cond-ev4op5:=true
                    end ;
   crereopmodec-3 : facteur-ev4op5 FACTOR (repere-de-c-operation-declenche):
                    ACTUAL ;

cre-rep-ev-dec-dec (id-ev4-per,id-op6-per, date-ev4op6-dec,cond-ev4op6,
                    facteur-ev4op6)
   DOMAIN id-ev4-per : SAME OF creation-rep-op-dec-per ;
          id-op6-per : SAME OF declenchement-effectif-per ;
```

SPECIFICATION CONCEPTUELLE DES A-EVENEMENTS IS-TYPE : CEVENT MODE : CEVDECLENCHE	Page 7 de 7

```
            date-ev4op6-dec : date ;
            cond-ev4op6     : char ;
            facteur-ev4-op6 : char ;
    ASSERT
    crerevdedec-1 : id-ev4-per, id-op6-per,date-ev4op6-dec COMPOSKEY
                    id-ev4op6-dec ;
    crerevdedec-2 : cond-ev4op6-CONDITION (repere-de-c-operation-declenche):
                    begin
                         cond-ev4op6:=true
                    end ;
    crerevdedec-3 : facteur-ev4op6 FACTOR (repere-de-c-operation-declenche):
                    ACTUAL ;

cre-rep-ob-mod-dec (id-ev1-per,id-op7-per,date-ev1op7-dec,cond-ev1op7,
                    facteur-ev1op7)
    DOMAIN id-ev1-per      : SAME OF creation-rep-opdec-per ;
           id-op7-per      : SAME OF modification-objet-per ;
           date-ev1op7-dec : date ;
           cond-ev1 op7    : char ;
           facteur-ev1op7  : char ;
    ASSERT
    crerobmod-1 : id-ev1-per,id-op7-per, date-ev1op7-dec COMPOSKEY
                  id-ev1op7-dec ;
    crerobmod-2 : cond-ev op7 CONDITION (repere-de-c-operation-declenche) :
                  begin
                       cond-ev1op7:=true
                  end ;
    crerobmod-3 : facteur-ev1op7 FACTOR (repere-de-c-operation-declenche):
                  ACTUAL ;

cre-rep-ob-dec (id-ev4-per,id-op8-per,date-ev4op8-dec,cond-ev4op8,
                facteur-ev4op8)
    DOMAIN id-ev4-per      : SAME OF creation-rep-opdec-per ;
           id-op8-per      : SAME OF reconnaissance-objet-per ;
           date-ev4op8-dec : date ;
           cond-ev4op8     : char ;
           facteur-ev4op8  : char ;
    ASSERT
    crereobdec-1 : id-ev4-per,id-op-per,date-ev4op8-dec COMPOSKEY
                   id-ev4op8-dec ;
    crereobdec-2 : cond-ev4op8 CONDITION (repere-de-c-operation-declenche):
                   begin
                        cond-ev4op8:=true
                   end ;
    crereobdec-3 : facteur-ev4op8 FACTOR (repere-de-c-operation-declenche):
                   ACTUAL ;
```

SPECIFICATION CONCEPTUELLE DES A-EVENEMENTS
IS-TYPE : CEVENT
MODE : CEVDECEFF

Page 1 de 1

```
   IS-TYPE CEVENT MODE CEVDECEFF

cre-rep-ev-decef (id-ev1-constate,id-op1-modifie)
   DOMAIN id-ev1-constate : SAME OF creation-rep-ob-constate ;
          id-op2-modifie :  SAME OF reconnaissance-evenement-modifie;
   ASSERT crobdf 1 : id-ev1-constate,id-op1-modifie COMPOSKEY
                     id-ev1op1-decef ;

cre-rep-op-decef (id-ev2-constate,id-op2-modifie )
   DOMAIN id-ev1-constate : SAME OF creation-rep-ev-constate ;
          id-op2-modifie  : SAME OF recherche-operation-modifie ;
   ASSERT crevdf 1 : id-ev2-constate,id-op2-modifie COMPOSKEY
                     id-ev2op2-decef ;

cre-rep-ev-const-decef (id-ev2-constate, id-op3-modifie)
   DOMAIN id-ev2-constate : SAME OF creation-rep-ev-constate ;
          id-op3-modifie  : SAME OF constatation-evenement-modifie ;
   ASSERT ev2op3 : id-ev2-constate,id-op3-modifie COMPOSKEY
                   id-ev2op3-decef ;

cre-rep-opdec-decef (id-ev3-constate, id-op4-modifie)
   DOMAIN id-ev3-constate : SAME OF creation-rep-op-constate
          id-op4-modifie  : SAME OF declenchement-operation-modifie ;
   ASSERT ev3op4 : id-ev3-constate,id-op4-modifie COMPOSKEY
                   id-ev3op4-decef ;

cre-rep-opmod-decef (id-ev4-constate, id-op5-modifie)
   DOMAIN id-ev4-constate : SAME OF creation-rep-opdec-constate ;
          id-op5-modifie  : SAME OF operation-modifie-modifie ;
   ASSERT ev4op5 : id-ev4-constate,id-op5-modifie COMPOSKEY
                   id-ev4op5-decef ;

cre-rep-ev-dec-decef (id-ev4-constate,id-op6-modifie-
   DOMAIN id-ev4-constate : SAME OF creation-rep-opdec-constate ;
          id-op6-modifie  : SAME OF declenchement-effectif-modifie ;
   ASSERT ev4op6 : id-ev4-constate,id-op6-modifie COMPOSKEY
                   id-ev4op6-decef ;

cre-rep-obmod-decef (id-ev1-constate,id-op7-modifie)
   DOMAIN id-ev1-constate : SAME OF creation-rep-opdec-constate ;
          id-op7-modifie  : SAME OF modification-objet-modifie ;
   ASSERT ev1 op7: id-ev1-constate,id-op7-modifie COMPOSKEY
                   id-ev1op7-decef ;

cre-rep-ob-decef (id-ev4-constate,id-op8-modifie)
   DOMAIN id-ev4-constate : SAME OF creation-rep-opdec-constate ;
          id-op8-modifie  : SAME OF reconnaissance-objet-modifie ;
   ASSERT ev4op8 : id-ev4-constate, id-op8-modifie COMPOSKEY
                   id-ev4op8-decef ;
```

III. LE PROCESSEUR DE LA DYNAMIQUE

L'objectif de ce chapitre est de montrer comment on réalise le processeur de la dynamique à partir de la spécification conceptuelle précedemment introduite .

Nous introduisons la correspondance entre l'architecture de la machine abstraite et celle du processeur de la dynamique.

Nous présentons le SGBD SYNTEX dont nous utilisons les possibilités pour l'écriture du processeur de la dynamique et puis nous présentons les solutions techniques retenues.

III.1. ARCHITECTURE DU PROCESSEUR

Elle se déduit de l'architecture de la machine abstraite compte tenu de l'environnement technique de réalisation choisi.

Dans notre cas nous avons choisi de réaliser le processeur de la dynamique d'une part sur l'ordinateur IRIS 80 de l'IUCA de Nancy d'autre part en utilisant les possibilités du SGBD Relationnel SYNTEX.

La correspondance que nous avons retenu entre l'architecture de la machine abstraite et celle du processeur de la dynamique est schématisée par la figure IV.1.a

Elle se traduit de la façon suivante :

- le schéma conceptuel est implémenté comme une base de données relationnelle SYNTEX. Nous l'appelons métabase puisque en effet les occurrences de la métabase définissent les schémas de relation du SI.
- le SI est implémenté comme une base de données relationnelle SYNTEX.
- les repères de la machine abstraite sont implémentés sous forme de fichiers COBOL.
- la machine abstraite est réalisée comme un programme COBOL avec une interface SYNTEX. Ce programme utilise les occurrences des repères des fichiers COBOL pour exécuter les primitives de la machine abstraite selon sa logique de fonctionnement, décrite dans le chapitre III. Ces primitives utilisent la description de la structure conceptuelle du SI contenue dans la métabase SYNTEX pour gérer l'évolution de la base de données SYNTEX qui est le SI.

III.2. LE SGBD SYNTEX

Nous introduisons le SGBD SYNTEX, qui a été développé à l'ONERA-CERT à Toulouse, de façon succinte. On peut trouver des informations complémentaires dans (ART 74)(DEM 75)(DEM 78)(DEM 79).

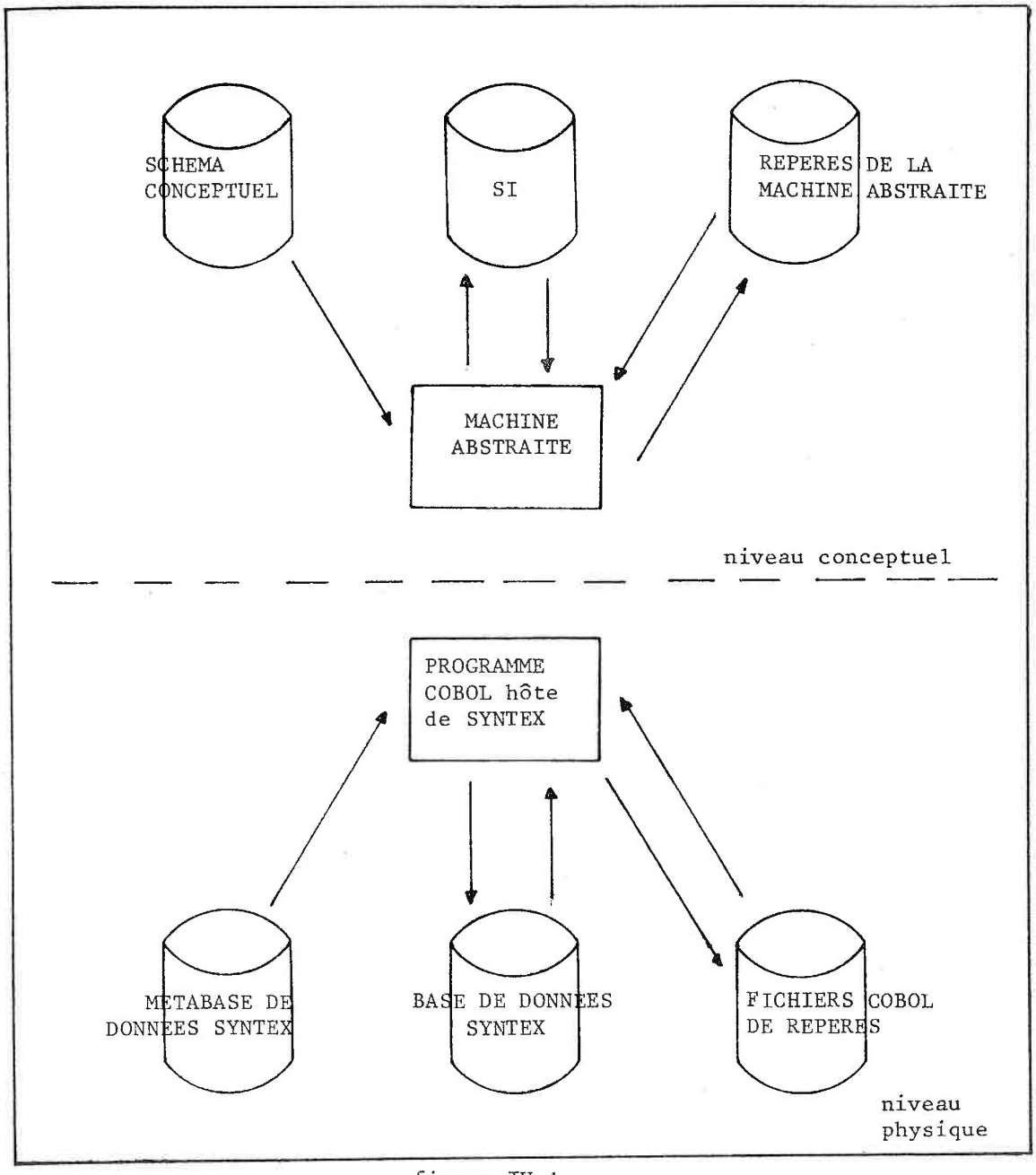


figure IV.1.a.

Nous introduisons successivement les élèments d'une base SYNTEX et le langage SYNTEX.

III.2.1. LES ELEMEMENTS D'UNE BASE SYNTEX

Une base SYNTEX est définie comme des nuples de valeurs (atomes ou molécules) interconnectés par des RDA (relations sur atomes) et des RDM (relations sur molécules).

III.2.1.1. Atomes

Un atome est une valeur individuelle d'une base SYNTEX. Son existence est indépendante de toute autre valeur. Elle peut être manipulée individuellement.

Un atome peut être numérique (de type entier ou réel) ou non-numérique (de type objet ou texte).

Exemples : entier : 1234
réel : 327.57
OBJET : JEAN
JEAN-LUC
JEAN 23
texte : EXTRAIT DE TEXTE

III.2.1.2. Molécules

Une molécule est un nuple ordonné de valeurs d'une base SYNTEX. Les valeurs qui composent une molécule ne sont manipulables qu'en tant que composants de la molécule. Contrairement aux atomes leur existence est interdépendante.

Chacune des valeurs d'une molécule peut être numérique (de type entier ou réel) ou non-numérique (de type chaine de caractères).

Exemples : (ABCD, 1234, 5678)
(123, AB12, JEJE)

III.2.1.3. Relations Définies sur Atomes (RDA)

Soit D_1, D_2, ... Dn une collection d'ensembles (non nécessairement disjoints).

R est une relation sur ces n ensembles si R est un ensemble de n-tuples ordonnées $< d_1, d_2, \ldots, dn >$ tel que $d1 \in D_1$, $d2 \in D2, \ldots, dn \in$ Dn.

On appelle Di un domaine de R.

Soit A l'ensemble des atomes de même type d'une base SYNTEX.

Une RDA est une relation sur la collection d'ensembles (A1, A2,..., An).

Une RDA est l'ensemble des n-tuples ordonnées <a1, a2, ..., an > où a i ∈ Ai.

"Une RDA est une relation dont les arguments sont uniquement de type ATOME" (DEM 75).

III.2.1.4. Relations définies sur Molécules (RDM)

Soit A l'ensemble des atomes de même type d'une base SYNTEX.

Soit M l'ensemble des molécules m1, m2,...,mn d'une base SYNTEX telles que :

- ∀i,j degré (mi)= degré (mj)
- ∀i,j si v_k est la valeur d'ordre k de mi
  et μ_k est la valeur d'ordre k de mj
  alors v_k et μk ont le même type (entier, réel, chaine de caractères)

Une RDM est une relation R sur la collection d'ensembles (A1, A2, ...; An, M).

Une RDM est un sous-ensemble de l'ensemble de n-tuples ordonnées $<a_1, a_2, \ldots a_{n-1}, m_n >$ où : ∀i, ai ∈ Ai et mn ∈ M;

"Une RDM est une relation dont un argument et un seul est de type molécule. Cette molécule a une structure moléculaire donnée"(DEM 75).

III.2.1.5. Relations définies sur ALGORITHMES (RDALG)

Soit A l'ensemble des atomes d'une base SYNTEX.

Une RDALG est une relation R sur la collection d'ensembles (A1, A2,..., An).

En d'autres termes, une RDALG est l'ensemble des n-tuples ordonnées
< a1, a2,...,an > où ai∈ Ai et tel qu'il existe une fonction boolène f.

$$f : A1, A2,...,An \longrightarrow \{0,1\}$$

$$(a1, a2,...,an) \longrightarrow f(a1, a2,...,an) = \begin{cases} 1 \text{ si } < a1,a2,...,an> \in R \\ 0 \text{ si } < a1,a2,...,an> \notin R \end{cases}$$

III.2.2. LANGAGE SYNTEX

Le langage SYNTEX associé au SGBD SYNTEX peut être utilisé selon les trois modes présentés, le passage d'un mode à un autre ne nécessitant pas de précaution particulière. La forme de la phrase suffit au système pour déterminer le mode employé.

III.2.2.1 Mode commande

Ce mode permet à l'utilisateur de créer (ou supprimer) les modèles de relation qu'il manipule.

Modèle de relation : Il désigne les constructions que l'on utilise pour définir une relation. Il équivaut à la notion de schéma de relation (BER 78).

Il est composé d'une suite de mots et de variables.
Une variable est désignée par son nom (au plus 8 caractères alphanumériques entre les symboles < et > .

A chaque variable correspond un argument de la relation désignée pour le modèle de relation.

La valeur d'une variable peut être soit un atome soit une molécule.

III.2.2.1.1. Création d'un modèle de relation RDA

Un modèle de relation RDA décrit une relation définie sur atomes.
On le crée selon la syntaxe :

CRE : modèle de relation, type 1, type 2,..., type n !

où ; CRE est la clause qui indique que l'on est en mode commande
. modèle de relation décrit la relation RDA
. le type de la ième variable est défini par type i
. ce type peut être - atome objet (Ø)
- atome numérique : entier (E)
réel (R)
- atome texte (c)

Exemples :
CRE : PRODUIT-STOCK <NPRO > <DATE > <QTE >, E, Ø, R !
On décrit la relation ternaire entre numéro produit, date de mise à jour du stock, quantité en stock.

CRE : <x > EST EMPLOYE AU SERVICE <Y >, Ø, Ø !
On décrit la relation binaire entre employé et service

CRE : PRODUIT-PRIX A NPRO <X > DT-PRIX <Y > PRIX <Z > ,E, Ø, R !
On décrit la relation ternaire entre numéro de produit, date du prix du produit et prix du produit.

CRE : PROJET <PROJ > UTILISE PRODUIT <X > FOURNIT PAR <Y > ,Ø, E, Ø !
On décrit la relation ternaire entre projet, article et fournisseur

III.2.2.1.2. Création d'un modèle de relation RDM

Un modèle de relation RDM décrit une relation définie sur molécules.

On le crée selon la syntaxe :

CRE : modèle de relation, type 1,..., type n, RDM : nom de fichier, taille,
{ (nom de champ attribut, type attribut [,table de codification]) } * !

où . CRE est la clause qui indique que l'on est en mode commande
. modèle de relation décrit la relation RDM
. type i est le type de la ième variable
. une variable (et une seule) est de type molécule (M)

. le "nom de champ attribut" est une suite de mots pouvant inclure les séparateurs'et-

. type attribut : C(n) : chaine de n caractères
E : entier
R : réel

. pour l'utilisation de table de codification voir (DEM 75)

. nom de fichier permet au SGBD SYNTEX de construire le nom de fichier qui sera utilisé par le système d'exploitation.

. taille donne le nombre de molécules qui pourra contenir le fichier.

Exemples :

CRE : PRODUIT <x >A LES CARACTERISTIQUES < Y > ,Ø, M, RDM : PROD, 100,
(POIDS, R) (COULEUR, C (10)(USINE, C (10)) !

On décrit la relation entre un numéro de produit et le triplet qui décrit ses caractéristiques : son poids, sa couleur, son usine de fabrication.

CRE : EMPLOYE < X > DU PROJET < Y > EST DEFINI PAR < Z > ,Ø,Ø, M, RDM :
EMPLOY,200, (SUPERVISEUR, C (20))(TACHE, C(15)) !

On décrit la Relation entre un employé, le projet sur lequel il travaille et le couple qui décrit les propriétés d'un employé dans un projet : son superviseur et sa tâche.

III.2.2.1.3. Création d'un modèle de relation RDALG

Un modèle de relation RDALG décrit une relation définie sur algorithme.

On le crée selon la syntaxe.

CRE : modèle de relation [,type1,..., tupe n], ALG : nom d'algorithme !

où . CRE est la clause qui indique que l'on est en mode commande

. type i est le type de la ième variable.

. le type peut être : atome numérique : entier (E)
: réel (R)

. nom d'algorithme est prédéfini dans SYNTAX. Cela peut être :
EG, NEG, INFE, INF, SUPE, SUP

Exemple

CRE : < X > EST SUPERIEUR A < Y >, ALG : SUP !
On décrit la relation entre deux valeurs où la première est supérieure à la deuxième.

III.2.2.1.4. Suppression de modèles de relation

Elle s'applique à tous les modèles de relation.
Elle a la syntaxe suivante :

SUP : modèle de relation !

III.2.2.2. Mode interrogation

Une question est constituée de deux parties, la seconde étant optionnelle :

1) une expression caractérisant les informations que l'on veut obtenir (on l'appelle LIF)
2) les spécifications de sortie

Deux niveaux de langage interviennent dans la formation d'une expression LIF. Le premier correspond à la déclaration des modèles de relation. Le deuxième correspond aux valeurs des attributs des variables de type molécule déclarées dans le premier niveau.

III.2.2.2.1. Premier niveau : expressions sur modèles de relations RDA et RDM

Ces expressions sont de deux types : expressions simples et expressions complexes.

Expressions simples

Une expression simple est un modèle de relation dans lequel les arguments sont soit des variables soit des valeurs imposées. Une variable peut éventuellement être quantifiée existentiellement.

On décrit les valeurs imposées d'atomes numériques entre ✱ et ✱. Les atomes de type objet (∅) sont représentés entre ≠ et ≠.

Exemples : on utilise les modèles de relation introduits précédemment

1) donner la liste des employés et des services où ils travaillent
. <X> EST EMPLOYE AU SERVICE <Y>?

2) donner la liste des projets qu'utilisent le produit 15 et sont approvisionnés par le fournisseur DUPONT.
. PROJET <X> UTILISE PRODUIT ✱15✱ FOURNIT PAR ≠ DUPONT ≠ ?

3) donner la quantité en stock de tous les produits le 15 mars 1980
. PRODUIT-STOCK <x> ≠19800315≠ <B> ?

4) donner les caractéristiques du produit 15 du catalogue
. PRODUIT ✱15✱ A LES CARACTERISTIQUES <Y> ?

Quantificateur existentiel : il est déclaré avant le modèle de relation qui compose la question.

Il a la forme

Ex. variable

où variable indique l'argument de la relation que l'on quantifie. Il peut y avoir plusieurs déclarations de variables quantifiées existentiellement dans une expression simple.

L'utilisation du quantificateur existentiel a pour effet sur la réponse la suppression des valeurs correspondantes aux variables quantifiées.

5) donner la liste des employés
. Ex. X, <X> EST EMPLOYE AU SERVICE <Y> ?

6) donner la liste des produits dans le catalogue
. Ex. Y, PRODUIT <X> A LES CARACTERISTIQUES <Y> ?

Expressions composées

Elles font intervenir les opérations logiques : ET, OU, MAIS PAS, NON. Une variable peut être quantifiée existentiellement au niveau d'une expression. Le domaine réponse d'une expression ayant n variables libres (c'est-à-dire, non quantifiées) est constitué de l'ensemble des nuples d'éléments (atomes et molécules) qui substitués aux variables libres conduisent à une expression vraie.

Exemples

1) quelles sont les caractéristiques des produits fournis au projet P12 ?
   Ex. X,(EX.Z, PROJET ≠P12≠ UTILISE PRODUIT < X > FOURNIT PAR < Z >
   ET (PRODUIT < X > A LES CARACTERISTIQUES < W >) ?

2) quelles sont les caractéristiques des employés qui travaillent dans le projet P12 ou dans un projet fourni par DUPONT et à quel service ils appartiennent.
   . Ex. X,(< X > EST EMPLOYE AU SERVICE < Y >)ET
   ((EMPLOYE < X > OU PROJET ≠P12≠ EST DEFINI PAR < Z >)OU
   (EX.X, (EMPLOYE < X > DU PROJET < W > EST DEFINI PAR < Z >)ET
   (EX.R,(PROJET < W > UTILISE PRODUIT < R > FOURNIT PAR ≠DUPONT≠)))) ?

III.2.2.2. Deuxième niveau : expressions sur attributs de molécules

Elles sont de deux types : simples et composées.

La variable molécule qui comparait dans une de ces expressions doit être déclarée comme variable libre dans une expression du premier niveau à laquelle l'expression du deuxième niveau est rattachée par un ET ou un MAIS PAS.

i Expressions simples sur attributs (ESA)

On appelle attribut le nom de champ attribut donné lors de la définition d'une molécule.

Ces expressions permettent d'exprimer des prédicats sur les valeurs associés aux attributs d'une molécule.

La forme générale en est la suivante :

"attribut de molécule" opérateur de comparaison "contrainte".

Un attribut est dénoté par : nom de variable molécule.nom de champ attribut

Un opérateur peut être : =, ¬ =, >, = >, < , =<

Une contrainte peut être :

- un attribut
- une valeur d'attribut
- liste de valeurs d'attributs dans laquelle chacun des constituants sont séparés par des barres verticales.

Exemples : nous utilisons les modèles de relation RDM introduits précédemment

1) donner les caractéristiques des produits dont la couleur est bleue ou verte. (PRODUIT < y > Δ LES CARACTERISTIQUES < Y >)ET (Y.COULEUR = ≠BLEUE≠ / ≠VERT≠) ?

2) donner les produits dont le poids est supérieur à 30 kg et ses caractéristiques (PRODUIT < X > A LES CARACTERISTIQUES < Y >) ET (Y.POIDS= > 30) ?

ii Expressions composées sur Attributs

Elles sont obtenues par composition de ESA à l'aide de connecteurs ET, OU, MAIS PAS.

Exemples : on utilise les modèles de relation introduits précedemment.

1) Quels sont les projets approvisionnés par le fournisseur DUPONT avec une pièce rouge qui n'est pas fabriquée à l'usine U2 et quelles sont les caractéristiques de ces pièces
EX-Y,(PROJET < X > UTILISE PRODUIT < Y > FOURNIT PAR ≠DUPONT≠) ET ((PRODUIT < Y > A LES CARACTERISTIQUES < Z >) ET Z.USINE ¬ =≠U2≠) ET (Z.COULEUR =≠ROUGE≠)))) ?

2) Quels sont les caractéristiques des produits qui avaient, le 15 juin 1979, un prix égal à 300 ou un poids supérieur à 300 mais ne sont pas ni bleus ni verts.

(PRODUIT-PRIX A NPRO < X > DT-PRIX≠19790615≠*300*) OU
((PRODUIT < X > A LES CARACTERISTIQUES < Y >)ET
((Y.POIDS 300) MAIS PAS (Y.COULEUR =≠VERT≠/≠BLEUE≠))) ?

III.2.2.2.3. Spécifications de Sortie

Les spécifications de sortie permettent de préciser les demandes suivantes :

i) Lister

Lorsqu'une variable molécule apparaît libre dans une expression (ce qui est toujours le cas pour les ESA ou les ECA) tous les champs des molécules sélectionnées seront listés lors de la formulation de la réponse. Une clause LS permet de n'obtenir que l'impression de certains champs.

La forme de cette clause est la suivante :

LS : nom de variable molécule (nom de champ$_1$,..., nom de champ Pn)

Exemples : On utilise les exemples de ECA introduits dans le paragraphe précédent.

1) Quels sont les projets approvisionnés par le fournisseur DUPONT avec une pièce rouge qui n'est pas fabriquée à l'usine U2 ?

EX. Y,((PROJET < X > UTILISE PRODUIT < Y > FOURNIT PAR ≠DUPONT≠) ET
((PROUIT < Y > A LES CARACTERISTIQUES < Z >)ET
((Z.USINE = ≠U2≠) ET (Z.COULEUR= ≠ROUGE≠)))), LS:Z () ?

2) Quel est l'usine ou sont fabriqués les produits qui avaient, le 15 juin 1979, un prix égal à 300 ou un poids supérieur à 300 mais ne sont pas ni bleus ni verts.

EX. X,(PRODUIT-PRIX A NPRO <X> DT-PRIX ≠19790615≠ *300*) OU
((PRODUIT < X > A LES CARACTERISTIQUES < Y >) ET
((Y. POIDS > 300) MAIS PAS (Y.COULEUR =≠VERT≠ / ≠BLEUE≠))),
LS : Y(USINE) ?

ii FONCTIONS

Il est possible de demander la valeur d'un certain nombre de fonctions appliquées aux valeurs correspondant à un variable atome ou à un champ donné d'une variable molécule.

La forme de la commande est la suivante :

nom de fonction : nom de variable atome
ou nom de fonction : nom de variable molécule (nom de champ)

Les fonctions disponibles sont :

MIN (minimum)
MAX (maximum)
MOY (moyenne)
SOM (somme)

Exemples

1) Quel est la date de la dernière mise à jour du stock du produit 15
EX. Y,EX.Z,(PRODUIT-STOCK ⁎15⁎ < Y > < Z > , MAX : Y ?

2) Quel est le poids du produit plus léger ?
EX. X,(PRODUIT < X > A LES CARACTERISTIQUES < Y >), LS:Y ( ),
MIN : Y (POIDS) ?

III.2.3. MODE MISE A JOUR

Il permet la mise à jour d'atomes et de modèles de relations RDA et RDM

i) Atomes Ils sont créés par CRE : modèle d'atome !
supprimés par SUP : modèle d'atome !

Exemples : CRE : ≠JEAN≠ !
CRE : ⁎40⁎ !
SUP : ≠JEAN≠

ii) RDA Une occurrence d'une RDA est créée par l'attribution de valeurs à son modèle de relation. Les atomes non-existant sont créés en même temps que l'occurrence.

Exemples : on utilise les modèles de relation introduits précedemment

1) PRODUIT-STOCK *15* ≠19800604≠ *3000*
   le produit 15 a la quantité 3000 en stock le 04 juin 1980

2) ≠JEAN≠ EST EMPLOYE AU SERVICE ≠COMPTABILITE≠ !
   on introduit les atomes JEAN et COMPTABILITE ainsi que le 2-uple qu'ils composent.

iii) RDM

Pour créer une occurrence de RDM on attribue des valeurs aux variables atomes et on fournit la liste des valeurs de la molécule.

Exemples : on utilise les modèles de relation introduits précedemment

1) PRODUIT *15* A LES CARACTERISTIQUES < Y > : *3000*,≠VERT≠, ≠U201≠ !

2) EMPLOYE ≠JEAN≠ DU PROJET ≠P2≠ A LES CARACTERISTIQUES < x >:
   ≠DURAND≠, ≠U201 ≠ !

Pour mettre à jour une ou plusieurs molécules d'un modèle de relation RDM on utilise la forme suivante :

modèle de RDM, MAJ : nom de champ attribut : valeur */,TOUS/ !

ou modèle de RDM peut avoir comme argument des variables libres ou des variables imposées

MAJ est la clause qui indique que l'on effectue une mise à jour,
nom de champ attribut indique l'attribut de la molécule que l'on veut actualiser.
TOUS s'il y a plus d'une molécule concernée par la mise à jour celle-ci ne sera faite que si la clause TOUS est présente.

Exemples : on utilise les modèles de relation introduits précedemment.

1) (PRODUIT < X > A LES CARACTERISTIQUES < Y > ET (Y.COULEUR= ≠VERT≠) ,
   MAJ : USINE : ≠U2≠, TOUS !

   on valorise l'attribut correspondant à USINE avec "U2" pour toutes les molécules dont l'attribut correspondant à COULEUR a la couleur VERT.

2) (EMPLOYE ≠JEAN≠ DU PROJET ≠P31≠ EST DEFINI PAR < Z >)ET((Z.SUPERVISEUR=, ≠DURAND≠) ET (Z.TACHE=≠COMPTABLE≠)), MAJ: TACHE:≠SUPERVISEUR≠ !
l'attribut correspondant à la tâche de l'employé est modifié.

i(x) Suppression

La suppression est réalisée par la commande : ECIE [,TOUS] //

où ECIE désigne : un modèle de RDA dont les arguments sont des variables libres ou des valeurs imposées ou un modèle de RDM soumis à ces mêmes règles et suivi d'une ECA.

La clause TOUS est utilisée quand plus d'un nuple est concerné.

Exemples : on utilise les modèles de relation introduits précédemment.

1) supprimer l'employé Jean du projet P31
EMPLOYE ≠JEAN≠ DU PROJET ≠P31≠ EST DEFINI PAR < X > //

2) supprimer tous les produits dont le prix est égal à 300
PRODUIT-PRIX <X> < Y > *300*, TOUS //

3) supprimer les produits de couleur verte
(PRODUIT <X > A LES CARACTERISTIQUES < Y >)ET
(Y.COULEUR =≠VERT≠), TOUS //

III. 3. META -STRUCTURE ET META-BASE SYNTEX

III.3.1. META-STRUCTURE SYNTEX

Nous avons défini la méta-structure conceptuelle au chapitre III.2.3.1. comme une collection de relations munies de leurs contraintes d'intégrité et décrites en ISDEL.

La Meta-structure SYNTEX que nous utilisons dans la réalisation du processeur de la dynamique est une collection de modèles de relation qui traduit la même sémantique que la meta-structure conceptuelle.

Nous la définissons par mapping. Ce mapping correspond à l'application de règles de correspondance entre le modèle relationnel ISDEL et le modèle SYNTEX.

III.3.1.1. Règles de correspondance entre les modèles

règle 1 : à chaque mode de la description conceptuelle correspond un atome.

règle 2 : à chaque attribut d'une relation de la description conceptuelle correspond un atome.

règle 3 : à chaque nom de relation de la description conceptuelle correspond un atome

règle 4 : l'appartenance d'une relation de la description conceptuelle à un mode est traduite par une RDA.

III.3.1.2. Mapping

Le mapping s'appuie sur les règles précédentes de correspondance entre les modèles.

Il conduit à l'ensemble de modèles de relation suivant :

1) modèles de relation qui traduisent l'appartenance d'une relation de la description conceptuelle à un mode

(TR1) : RELATION < X > DU MODE COB < Y > , ØØ !
(TR2) : RELATION < X > DU MODE COPPER < Y > , ØØ !
(TR3) : RELATION < X > DU MODE COPTEXTE < Y > , ØØ !
(TR4) : RELATION < X > DU MODE COPMODIFIE < Y > , ØØ !
(TR5) : RELATION < X > DU MODE CEVPER < Y > , ØØ !
(TR6) : RELATION < X > DU MODE CEVPRED < Y > , ØØ !
(TR7) : RELATION < X > DU MODE CEVCONSTATE < Y > , ØØ !
(TR8) : RELATION < X > DU MODE CEVDECLENCHE < Y > , ØØ !
(TR9) : RELATION < X > DU MODE CEVDECEFF < Y > , ØØ !

2) Modèle de relation qui décrit la composition d'une relation en attributs.

(CR) : COMPOSITION RELATION <X> <Y> <W> <Z> ,Ø,Ø,Ø,Ø !

où <X> désigne un nom de relation de la description conceptuelle
<Y> désigne le nom d'un de ses attributs
<Z> désigne l'ordre de l'attribut dans la relation
<W> indique le format de l'attribut.

3) modèle de relation qui décrit la composition d'un identifiant d'une relation

(CC) : COMPOSITION CLE <X> <Y> <Z>

où <X> désigne un nom de relation de la description conceptuelle
<Y> désigne le nom de l'identifiant clé de la relation
<Z> désigne le nom de l'attribut qui compose l'identifiant clé

4) modèles de relation qui traduisent les associations entre les modes de la description conceptuelle :

(MS1) COB <X> DU CEVCONSTATE <Y> <X> est le nom d'une relation de mode COB
<Y> est le nom d'une relation de mode CEVCONSTATE
<Y> constate le changement d'état de <X>

(MS2) COB <X> DE LA COPMODIFIE <Y> X> est le nom d'une relation de mode COB
<Y> est le nom d'une relation de mode COPMODIFIE
<Y> modifie <X>

(MS3) CEVCONSTATE <X>
COPMODIFIE <Y>
DU CEVDECEFF <Z>
<X> est le nom d'une relation de mode CEVCONSTATE
<Y> est le nom d'une relation de mode COPMODIFIE
<Z> est le nom d'une relation de mode CEVDECEFF
<Z> indique que <Y> a été exécuté après <X>

(MS4) : CEVPRED < X > DU CEVPER < Y > < X > est le nom d'une relation de mode CEVPRED
< Y > est le nom d'une relation de mode CEVPER
< X > introduit les prédicats < Y >

(MS5) : CEVCONSTATE < X > DU CEVPER < Y > < X > est le nom d'une relation de mode CEVCONSTATE
< Y > est le nom d'une relation de mode CEVPER
< X > indique l'objet associé à < Y >

(MS6) : CEVCONSTATE < X > DU CEVPRED < Y > < X > est le nom d'une relation de mode CEVCONSTATE
< Y > est le nom d'une relation de mode CEVPRED
< X > utilise les prédicats de < Y >

(MS7) : COPMODIFIE < X > DU COPPER < Y > < X > est le nom d'une relation de mode COPMODIFIE
< Y > est le nom d'une relation de mode COPPER
< X > indique l'objet associé à < Y >

(MS8) : COPTEXTE < X > DU COPPER < Y > < X > est le nom d'une relation de mode COPTEXTE
< Y > est le nom d'une relation de mode COPPER
< X > introduit le texte de < Y >

(MS9) : COPMODIFIE < X > DU COPTEXTE < X > < X > est le nom d'une relation de mode COPMODIFIE
< Y > est le nom d'une relation de mode COPTEXTE
< X > utilise le texte de < Y >

(MS10) : CEVPER < X > COPPER < Y > DU CEVDECLENCHE < W > < X > est le nom d'une relation de mode CEVPER
< Y > est le nom d'une relation de mode COPPER
< W > est le nom d'une relation de mode CEVDECLENCHE
< W > indique que < X > déclenche < Y >

III.3.1.3. Expression graphique de la meta-structure conceptuelle SYNTEX

Nous reprenons le graphisme introduit dans (DEM75) pour représenter les concepts de SYNTEX. Il est le suivant :


expression : indique un atome.Exemple


COMPTABILITE


expression : indique un modèle. Exemple de relation


<x >EST EMPLOYE AU SERVICE <y >


expression : indique le"type atome". . Exemple


SERVICE

L'expression graphique de la meta-structure conceptuelle SYNTEX est faite par les schémas IV.3.1.3.a et IV 3.1.3 b

Nous indiquons les modèles de relation de la meta-structure conceptuelle SYNTEX par leur désignation en abrégé.

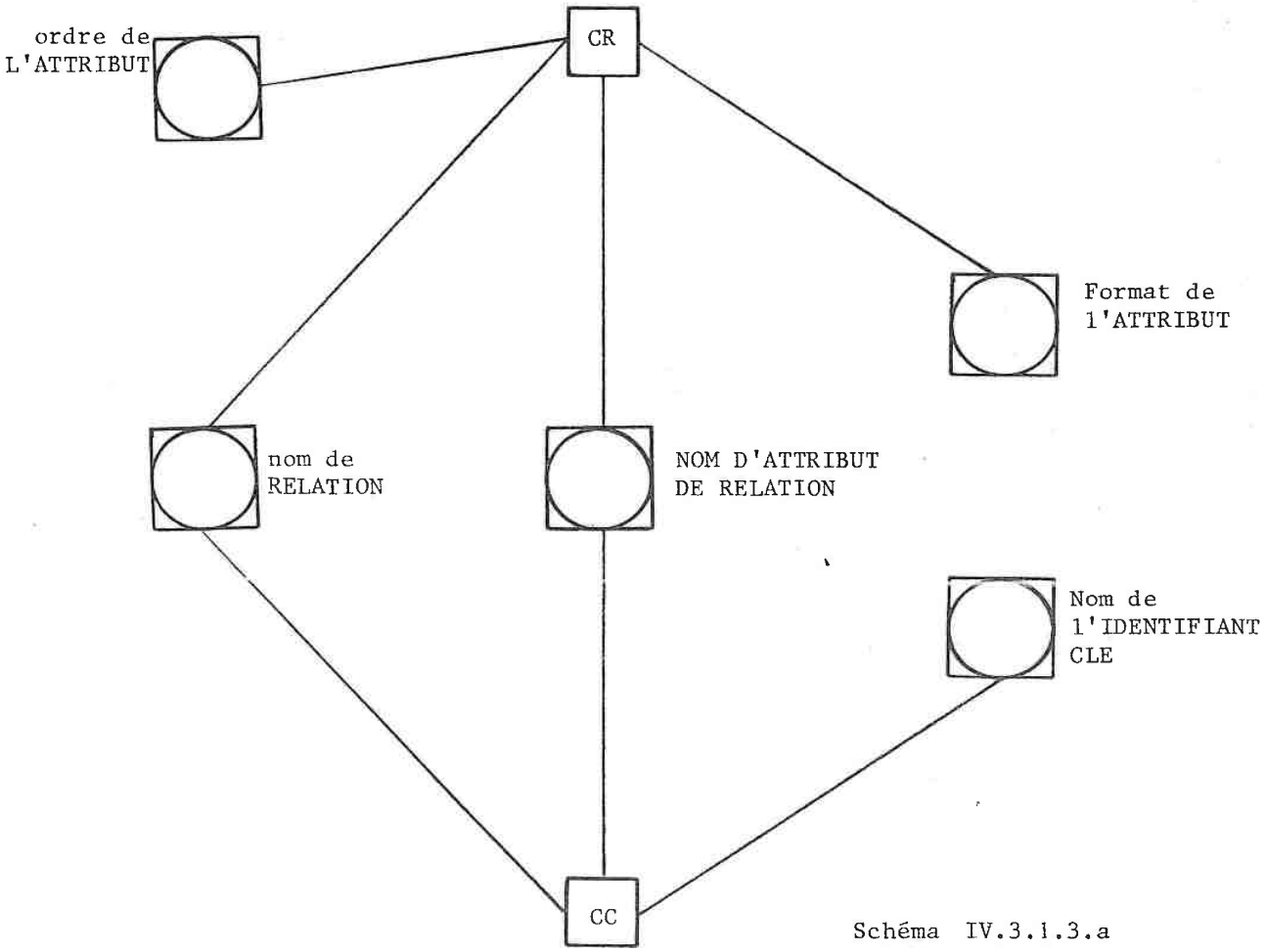
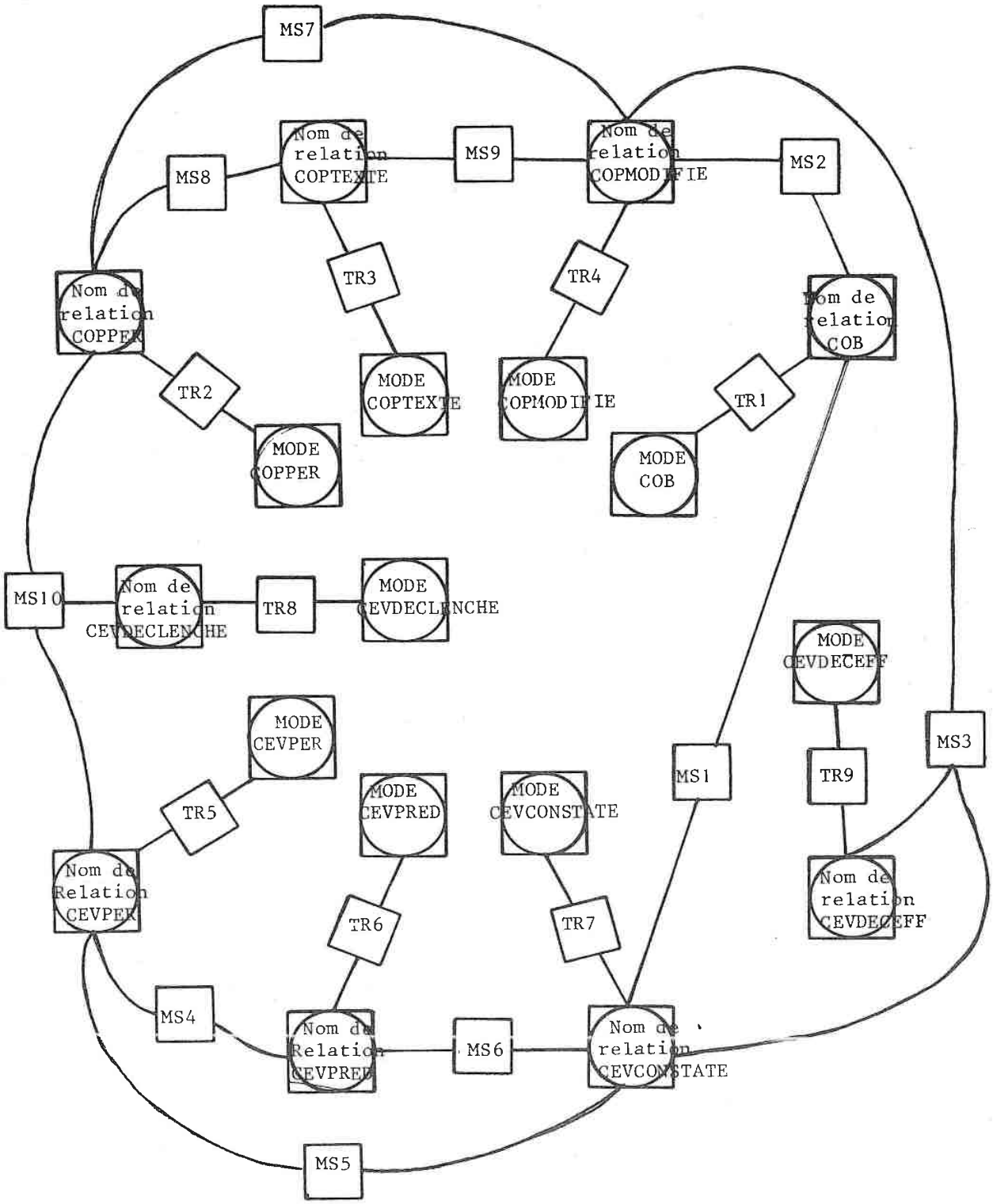


Schéma IV.3.1.3.a


MS7
MS8
Nom de relation COPTEXTE
MS9
Nom de relation COPMODIFIE
MS2
TR3
TR4
Nom de relation COPPER
Nom de relation COB
TR2
MODE COPTEXTE
MODE COPMODIFIE
TR1
MODE COPPER
MODE COB
MS10
Nom de relation CEVDECLENCHE
TR8
MODE CEVDECLENCHE
MODE CEVDECEFF
MODE CEVPER
MS3
MS1
TR9
TR5
MODE CEVPRED
MODE CEVCONSTATE
Nom de Relation CEVPER
Nom de relation CEVDECEFF
TR6
TR7
MS4
Nom de Relation CEVPRED
MS6
Nom de relation CEVCONSTATE
MS5

III.3.2. METABASE SYNTEX

La metabase SYNTEX correspond aux nuples d'atomes liés aux modèles de relations de la meta structure conceptuelle SYNTEX.

Ces nuples sont composés à partir des Relations de la description conceptuelle ISDEL d'un SI.

A chaque mode de relation de la description conceptuelle on établit la liste des modèles de relation de la meta structure conceptuelle qui sont concernés.

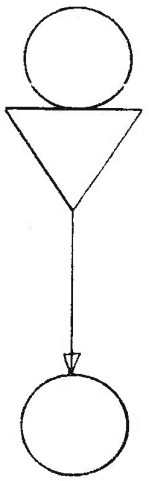
La création des occurrences de nuples de la metabase est faite selon les règles présentées en III.3.1.2.

Nous considérons que chaque ensemble de relations de le description conceptuelle pris en compte traduit un cycle dynamique.

Un cycle dynamique est défini comme l'ensemble de relations qui décrivent un c-événement que constate le changement d'état d'un c-objet, de toutes les c-opérations directement déclenchées par ce c-événement et des c-objets qui sont modifiés par les C-OPERATIONS (figire IV.3.1.4.a). (BAN 79).

III.3.2.1. Exemple de création d'occurrences de la metabase

Nous présentons comme exemple un cycle dynamique extrait de l'exemple présenté en II.3.5.



OB6 : produit en stock

EV2 : quantité en stock inférieure au seuil

OP6 : création d'une demande de réapprovisionnement

OB7 : demande de réapprovisionnement

Nous rappelons les relations ISDEL qui le décrivent

produit-stock (npro, date-prostock, qtestock)
ev 2-per (nev2, date-ev2-per)
ev 2-pred (nev2, date-ev2-pred, pinit-ev2, pfin-ev2)
ev 2-constate (nev2-pred, date-ev2-constate, nprostock)
ev 2-op6-dec (nev2,nop6,date-ev2op6-dec,cond-ev2op6, facteyr-ev2op6)
ev 2-op6-deceff (nev2-constate, nop6-modifie)
op6-per (nop , date-op6-per)
op6-texte (nop6, date-op6-texte, texte-op6)
op6-modifie (nop6-texte, date-op6-modifie, nproderea)
produit-demande-reap (npro,date-proderea, qtedereap)

La création des occurrences de la metabase est faite par les expressions du langage SYNTEX suivantes :

relation ≠produit stock≠ du mode ≠cob≠ !
relation ≠produit demande-reap≠ du mode ≠cob≠ !
relation ≠ev2 per ≠ du mode ≠cevper≠ !
relation ≠ev2 pred≠ du mode ≠cevpred≠ !
relation ≠ev2 constate≠ du mode ≠cevconstate≠ !
relation ≠ev2op6 dec≠ du mode ≠cevdec≠ !
relation ≠ev2op6 deceff≠ du mode ≠cevdeceff≠ !
relation ≠op6 per≠ du mode ≠copper≠ !
relation ≠op6 texte≠ du mode ≠ coptexte≠ !
relation ≠op6 modifie≠ du mode ≠copmodifie ≠ !

composition relation ≠produit stock≠ ≠npro≠ ≠1≠ ≠E0400≠ !
composition relation ≠produit stock≠≠date-prostock≠≠2≠≠C0800≠ !
composition relation ≠produit stock≠≠qte-stock≠≠3≠R1002≠ !
composition relation ≠produit demande reap≠≠npro≠≠1≠≠E0400≠ !
composition relation ≠produit demande reap≠≠date proderea≠≠2≠≠C0800≠ !
composition relation ≠produit demande reap≠≠qtedereap≠≠3≠≠R0802≠ !
composition relation ≠ev2 per ≠≠nev2≠≠1≠≠co500≠ !
composition relation ≠ev2 per≠≠date-ev2-per≠≠2≠≠C0800≠ !
composition relation ≠ev2 pred≠≠nev2≠≠1≠≠C0500 ≠ !
composition relation ≠ev2 pred≠≠date ev2 pred≠≠2≠≠C0800≠ !

```
composition relation ≠ev2 pred##pinit ev2##3##C1000≠ !
composition relation ≠ev2 pred##pfin ev2##4##C1000≠ !
composition relation ≠ev2 constate##nev2##1##C0500≠ !
composition relation ≠ev2 constate##date ev2 pred##2##C0 00≠ !
composition relation ≠ev2 constate##date ev2 constate##3##C0800≠ !
composition relation ≠ev2 constate##nprostock ##4##C1200≠ !
composition relation ≠ev2 op6 dec##nev2##1##C0500≠ !
composition relation ≠ev2 op6 dec##nop6##2##C0500≠ !
composition relation ≠ev2 op6 dec##date ev2op6 dec##3##C0800≠ !
composition relation ≠ev2 op6 dec##cond ev2op6##4##C1000≠ !
composition relation ≠ev2 op6 dec##facteur ev2op6##5##C1000≠ !
composition relation ≠ev2 op6 deceff ≠nev2##1##C0500≠ !
composition relation ≠ev2 op6 deceff##date ev2pred##2##C0800≠ !
composition relation ≠ev2 op6 deceff##date ev2 constate##3##C0800≠ !
composition relation ≠ev2 op6 deceff##nop6##4##C0500≠ !
composition relation ≠ev2 op6 deceff##date op6 texte##5##C0800≠ !
composition relation ≠ev2 op6 deceff##date op6 modifie##6##C0800≠ !
composition relation ≠op6 per##nop6##1##C0500≠ !
composition relation ≠op6 per ##date op6 per ##2##C0800≠ !
composition relation ≠op6 texte##nop6##1##C0500≠ !
composition relation ≠op6 texte##date op6 texte##2##C0800 ≠ !
composition relation ≠op6 texte##texteop6##3##C1000≠ !
composition relation ≠op6 modifie##nop6##1##C0500≠ !
composition relation ≠op6 modifie##date op6 texte##2##C0800≠ !
composition relation ≠op6 modifie##date op6 modifie##3##C0800≠ !
composition relation ≠op6 modifie##nproderea##4##C1200≠ !

composition cle ≠produit demande reap##nproderea##npro##E0400≠ !
composition cle ≠produit demande reap##nproderea##date proderea##C0800≠ !
composition cle ≠produit stock##npro##E0400≠ !
composition cle ≠produit stock##date prostock##C0800≠ !
composition cle ≠nev2 pred##nev2##C0500≠ !
composition cle ≠nev2 pred##date ev2pred##C0800≠ !
composition cle ≠nev2 constate##nev2##C0500≠ !
composition cle ≠nev2 constate##date ev2 pred##C0800≠ !
composition cle ≠nev2 constate##date ev2 constate##C0800≠ !
```

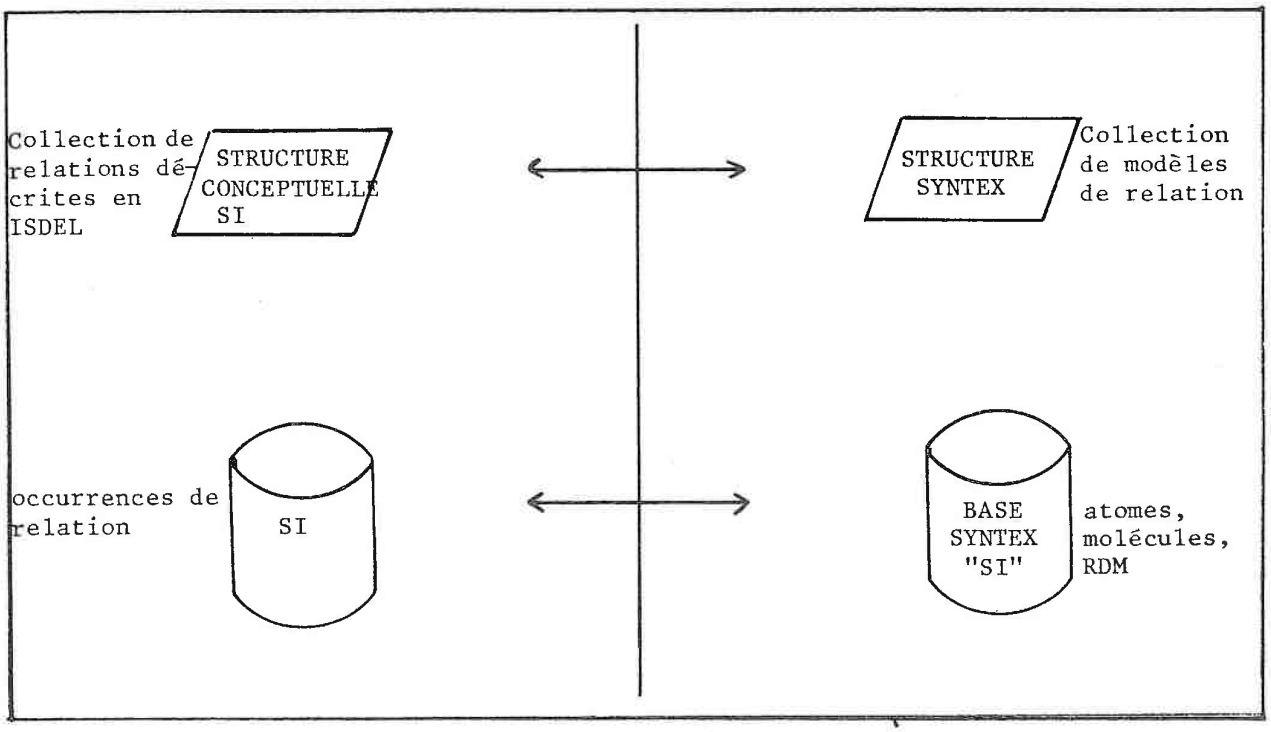
```
composition cle ≠nev2 op6 dec≠≠nev2≠≠CO500≠ !
composition cle ≠nev2 op6 dec≠≠nop6≠≠CO500≠ !
composition cle ≠nev2 op6 dec≠≠dateev2op6 dec≠≠CO800 !
composition cle ≠nev2 op6 deceff≠≠nev2=≠/CO500≠ !
composition cle ≠nev2 op6 deceff≠≠date ev2 pred≠≠CO800≠ !
composition cle ≠nev2 op6 deceff≠≠date ev2 constate≠≠CO800≠ !
composition cle ≠nev2 op6 deceff≠≠nop6≠≠CO500≠ !
composition cle ≠nev2 op6 deceff≠≠date op6 texte≠≠ CO8000≠ !
composition cle ≠nev2 op6 deceff≠≠date op6 modifie≠≠CO800≠ !
composition cle ≠nop6 texte≠≠nop6≠≠CO500≠ !
composition cle ≠nop6 texte≠≠date op6 texte≠≠CO800≠ !
composition cle ≠nop6 modifie≠≠nop6≠≠CO500≠ !
composition cle ≠nop6 modifie≠≠date op6 texte≠≠CO800≠ !
composition cle ≠nop6 modifie≠≠ date op6 modifie≠≠CO800≠ !

cob≠produit-stock≠du cevconstate≠ev2-constate≠ !
cob≠produit-demande-reap≠ de la copmodifie≠op6-modifie≠ !
cevconstate≠ev2-constate≠copmodifie≠op6 modifie≠du cevdeceff
 ≠ev2 op6 cevdeceff≠ !
cevpred≠ev2-pred≠ du cevper≠ev2 per≠ !
cevconstate≠ev2 constate≠ du cevper≠ev2 per≠ !
cevconstate≠ev2 constate≠ du cevpred≠ev2 pred≠ !
copmodifie≠op6 modifie≠du copper≠op6 per≠ !
copmodifie≠op6 modifie≠du coptexte≠op6 texte≠ !
coptexte≠op6 texte≠du copper≠op6 per≠ !
cevper≠ev2 per≠copper≠op6 per≠du cevdeclenche≠ev2 op6 dec≠ !
```

III.4 STRUCTURE ET SI SYNTEX

III.4.1. STRUCTURE SYNTEX

Nous définissons le SI par l'ensemble des Relations ISDEL de sa structure conceptuelle. Nous l'implémentons comme une base de données SYNTEX.

De la même façon que nous avons associé à la meta-structure conceptuelle une meta-structure SYNTEX décrivant la meta-base SYNTEX nous associons à la structure conceptuelle ISDEL du SI une structure SYNTEX qui définit l'image de la base SYNTEX.



Nous définissons la structure SYNTEX par mapping. Ce mapping correspond à l'application de règles de correspondance entre le modèle ISDEL et le modèle SYNTEX.

III.4.1.1. Règles de correspondance entre les modèles

règle 1 à chaque nom de relation de la description conceptuelle d'un SI correspond un atome, quelque soit son mode.

règle 2 à chaque attribut d'une relation de la description conceptuelle correspond un champ attribut d'une molécule.

III.4.1.2. Mapping

Le mapping s'appuie sur les règles de correspondance entre les modèles présentées en III.4.1.1.

Il conduit à représenter toute relation de la description conceptuelle par un modèle de relation de type RDM qui obéit à la forme générale :

nom-de-relation < x > < y >, Ø, M, RDM : FICH, SPEC où

. nom-de-relation : c'est le nom de la relation de la description conceptuelle

< X > : est une variable atome qui représente le nom de la relation de la description conceptuelle

< Y > : est une variable molécule dont chaque champ attribut correspond à un attribut de la description conceptuelle

FICH : repere le fichier RDM et définit sa taille

SPEC : correspond à la description des champs attribut de la variable molécule < Y >. Le type du champ attribut est obtenue à partir de la description du domaine de l'attribut de la relation de la description conceptuelle.

III.4.1.3. Expression graphique de la structure SYNTEX

Nous reprenons le graphisme introduit en III.3.1.3. en ajoutant les notations suivantes :

val 1	val 2	...	valn

indique un nuple de valeur (val 1, val 2, ..., val n) qui constitue une molécule

exp 1	exp 2 ...	exp n

chaque exp i indique le type du "champ attribut" de la molécule.

La structure SYNTEX comporte autant de modèles de relation RDM qu'il y a de relations dans la description conceptuelle. Ils ont tous l'expression graphique générale présentée sur le schéma IV.4.1.3.a.

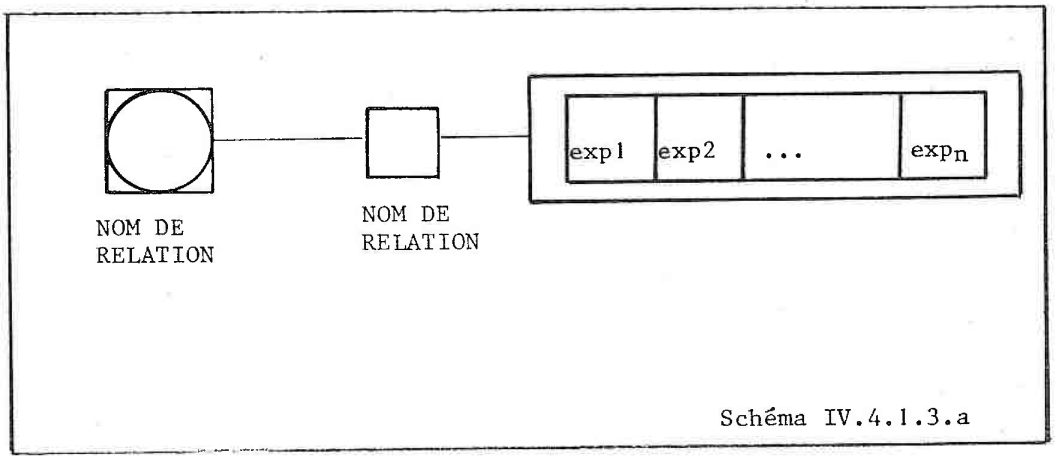


Schéma IV.4.1.3.a

III.4.2. SYSTEME D'INFORMATION OU BASE SYNTEX "SI"

Elle est faite par l'utilisation du langage SYNTEX.

Nous présentons deux exemples de création d'occurrences du "SI" en utilisant les relations de la description conceptuelle du cycle dynamique présenté en III.3.2.1.

Exemple 1 : considérons la relation de mode cob présentée en I.3.5.

```
produit-stock (npro, date-prostock, qtestock)
  DOMAIN  npro          : SAME OF cclasse-produit ;
          date-prostock : date (8) ;
          qte-stock     : real (10,2) ;
  ASSERT  ...
```

A cette relation correspond le modèle de relation de type RDM du SI :

produit stock < x > < y >, Ø, M, RDM : ØB06, 100, (npro,E)
(date-prostock,C(08)) (qtestock,R)

Une occurrence de ce modèle de relation est donnée par l'expression

produit stock ≠produit stock≠ < X > : 15, ≠19800104≠, 3000 !

Exemple

Exemple 2 : considerons la relation de mode cob texte présentée en II.3.5.

```
derepro-texte (nop6, date-op6-texte, texte-op6)
  DOMAIN  nop6           : SAME OF demande-reap-prod ;
          date-op6-texte : date (8) ;
          texte-op6      : char (8) ;
  ASSERT  ...
```

A cette relation correspond le modèle de relation de type RDM du SI :

derepro texte < X > < Y >, Ø, M, RDM : ØP06T, 10, (nop6, C(5))(date-op6-texte, C(8)) (texte-op6, C(8))

Une occurrence de ce modèle de relation est donnée par l'expression

derepro texte ≠derepro texte≠ < X >: ≠nopo6≠, ≠19800104≠, ≠nopo601≠ !

Remarquons que dans l'implémentation choisie on ne mémorise dans le SI que le nom mnémonique des textes des opérations, conditions et facteurs de déclenchement et des prédicats des événements.

La description de ces textes ainsi que des textes des contraintes d'intégrité est contenue dans une bibliothèque qui peut être consultée par des macro-enquêtes à partir du nom mnémonique contenu dans le SI.

III.5. Les repères

Les repères ont été implémentés sous forme de fichiers COBOL en accès direct.

Il y a un fichier COBOL par repère. Sa structure est dérivée à partir de celle du a-objet correspondant de la machine abstraite.

Nous avons choisi cette solution (au lieu d'une implémentation sous forme d'une base de données SYNTEX) car la manipulation des repères est simple, locale au superviseur et temporaire.

Le " overhead " de l'utilisation de SYNTEX n'était pas justifié

La gestion des repères procède selon l'une des deux techniques suivantes :

liste circulaire : le premier registre du fichier est composé par le numéro de bloc du dernier repère inclu et le numéro de bloc du dernier repère traité. Il y a un repère par bloc.

Cette technique est utilisée pour les fichiers correspondant aux repères de c-objet, de c-événement, de c-opération, de c-opération-déclenche.

élément unique : le fichier n'a qu'une occurrence. Celle du dernier repère inclu. Elle occupe un bloc.

C'est la technique utilisée pour les repères de c-événement-constate, de c-opération-modifie, de c-objet-modifie, de c-événement-déclenche.

III.5.1. DESCRIPTION DES FICHIERS

Fichier repère-de-c-objet

DERNIER-OB INCLU	DERNIER-OB-TRAITE

premier bloc

NOM-REL-COB	ATTRIBUTS-OB-ACTUEL

autres blocs

DERNIER-OB-INCLU : valeur du bloc du dernier repère de c-objet inclu
DERNIER-OB-TRAITE: valeur du bloc du dernier repère de c-objet traité

NOM-REL-OB : nom du c-objet qui a changé d'état
ATTRIBUTS-OB-ACTUEL : valeur de l'occurrence qui a produit le changement d'état

Fichier repère-de-c-évènement

DERNIER-EV-INCLU	DERNIER-EV-TRAITE

premier bloc

NOM-REL-EV	VAL-ID-CEVPER	VAL-DATE CEVPRED	VAL-DATE-CEVCONSTATE	ATTRIBUTS-OB ACTUEL	REP-OB

autres blocs

DERNIER-EV-INCLU : valeur du bloc du dernier repère-de-c-évènement inclu
DERNIER-EV-TRAITE : valeur du bloc du dernier repère-de-c-évènement traité

NOM-REL-EV : nom de la relation de mode cevconstate du c-évènement constaté
VAL-ID-CEVPER : valeur de l'attribut id-cevper qui compose la clé de NOM-REL-EV
VAL-DATE-CEVPRED : date associé aux textes des prédicats du c-évènement
VAL-DATE-CEVCONSTATE : date de la constatation du c-évènement
ATTRIBUTS-OB-ACTUEL: valeur de l'occurrence de c-objet qui a produit l'évènement
REP-OB : numéro de bloc du repère-de-c-objet associé.

Fichier repère-de-c-opération

DERNIER-OP INCLU	DERNIER-OP TRAITE

premier bloc

NOM-REL-OPADEC	VAL-ID COPPER	DESC-REP-EV	DESC-REP-OB	REP-EV

autres blocs

DERNIER-OP-INCLU : valeur du bloc du dernier repère-de-c-opération inclu
DERNIER-OP-TRAITE : valeur du bloc du dernier repère-de-c-oparation traité

NOM-REL-OPADEC : nom de la relation de mode copper de la c-opération à déclencher
VAL-ID-COPPER : valeur de l'identifiant id-copper qui compose la clé de NOM-REL-OPADEC
DESC-REP-OB : comprend : NOM-REL-COB, ATTRIBUTS-OB-ACTUEL
DESC-REP-EV : comprend : NOM- EL-EV,VAL-ID-CEVPER,VAL-DATE-CEVPRED, VAL-DATE-CEVCONSTATE
REP-EV : numéro de bloc du repère-de-c-évènement associé.

Fichier repère-de-c-opération-déclenche

DERNIFR-OPEX-INCLU	DERNIER OPEX TRAITE

premier bloc

NOM-REL-OPEX	VAL-ID-COPPER	VAL-DATE-OPEX-TEXTE	DESG-REP-EV	NOM-REL-OB-OPEX	ATTRIBUTS-REL-OB-OPEX	REP-OPADEC

autres blocs

DERNIEP-OPEX-INCLU : valeur du bloc du dernier repère-de-c-opération-déclenche inclu
DERNIER-OPEX-TRAITE : valeur du bloc du dernier repère-de-c-opération-déclenche traite

NOM-REL-OPEX : nom de la relation de mode copmodifié de la c-opération déclenchée

VAL-I-COPPER : valeur de l'identifiant id-copper qui compose la clé de NOM-NEL-OPEX

VAL-DATE-OPEX-TEXTE : date du texte de la c-opération déclenchée

DESC-REP-EV : comprend : NOM-REL-EV, VAL-ID-CEVPER, VAL-DATE-CEVPRED, VAL-DATE-CEVCONSTATE

NOM-REL-OB-OPEX : nom de la relation de mode cob du c-objet modifié.

ATTRIBUTS-REL-OB-OPEX : valeur de l'occurence du tuple du c-objet qui a été modifié

REP-OPADEC : numéro de bloc du repère-de-c-opération associé.

Fichier repère-de-c-événement-constate

REP-EV	unique bloc

REP-EV : numéro de bloc du repère-de-c-évènement associé

Fichier repère-de-c-opération-modifie

REP-OPEX	unique bloc

REP-OPEX : numéro de bloc du repère-de-c-opération-déclenche associé

Fichier repère-de-c-objet-modifie

REP-OPEX	unique bloc

REP-OPEX : numéro de bloc du repère-de-c-opération-déclenchee associé

Fichier repère-de-c-évènement-déclenche

REP-OPEX	unique bloc

REP-OPEX : numéro de bloc du repère-de-c-opération-déclenchée associé

III.6 . LE PROGRAMME PROCESSEUR DE LA DYNAMIQUE

III.6.1. STRUCTURE DU PROGRAMME

Le processeur de la dynamique a été réalisé comme un programme COBOL.

L'architecture du programme se déduit de la spécification conceptuelle de la machine abstraite présentée au chapitre I.

Nous associons à chaque cycle dynamique de la spécification conceptuelle de la machine abstraite une "section" COBOL.

Il y a donc quatre "section COBOL" appelées :

- prendre-événement
- prendre-opération
- déclencher-opération
- actualisation-SI

Nous associons à chaque a-opération d'un cycle dynamique de la machine abstraite trois paragraphes COBOL :

- paragraphe DEBUT-(a-opération) : correspond à l'évaluation de la condition de déclenchement de la a-opération et de son facteur de déclenchement.
- paragraphe OPERATION-(a-opération) : correspond à l'évaluation du texte de la a-opération
- paragraphe FIN-(a-opération) : correspond à l'actualisation du fichier de repères de a-objets associé au a-événement qui a déclenché les a-opérations du cycle dynamique. Ce paragraphe n'est défini que pour une a-opération du cycle dynamique.

La structure du programme est une traduction de l'ordonnancement des cycles dynamiques de la machine abstraite. Cet ordonnancement est exprimé par les dépendances chronologiques (RIC 79) entre les a-événements.

$EV1 \xrightarrow{c} EV2$, $EV2 \xrightarrow{c} EV3$, $EV3 \xrightarrow{p} EV4$, $EV4 \xrightarrow{p} EV1$

où $EVi \xrightarrow{c} EVj$ indique que EVj peut suivre EVi (il le fait si la condition C est satisfaite).

$EVi \xrightarrow{p} EVj$ indique que EVj suit toujours EVi

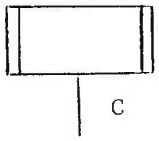
Nous le représentons par l'ordonnancement des sections COBOL :

1- prendre-événement (correspond à EV1)
2- prendre-opération (correspond à EV2)
3- déclencher-opération (correspond à EV3)
4- actualisation-SI (correspond à EV4)

Il oriente la définition du fonctionnement du programme et il est exprimé par l'arbre programmatique de la figure III.6.1.a.

Plus précisement la structure du programme construite avec les conventions de la programmation structurée suivante :

désigne un paragraphe de traitement



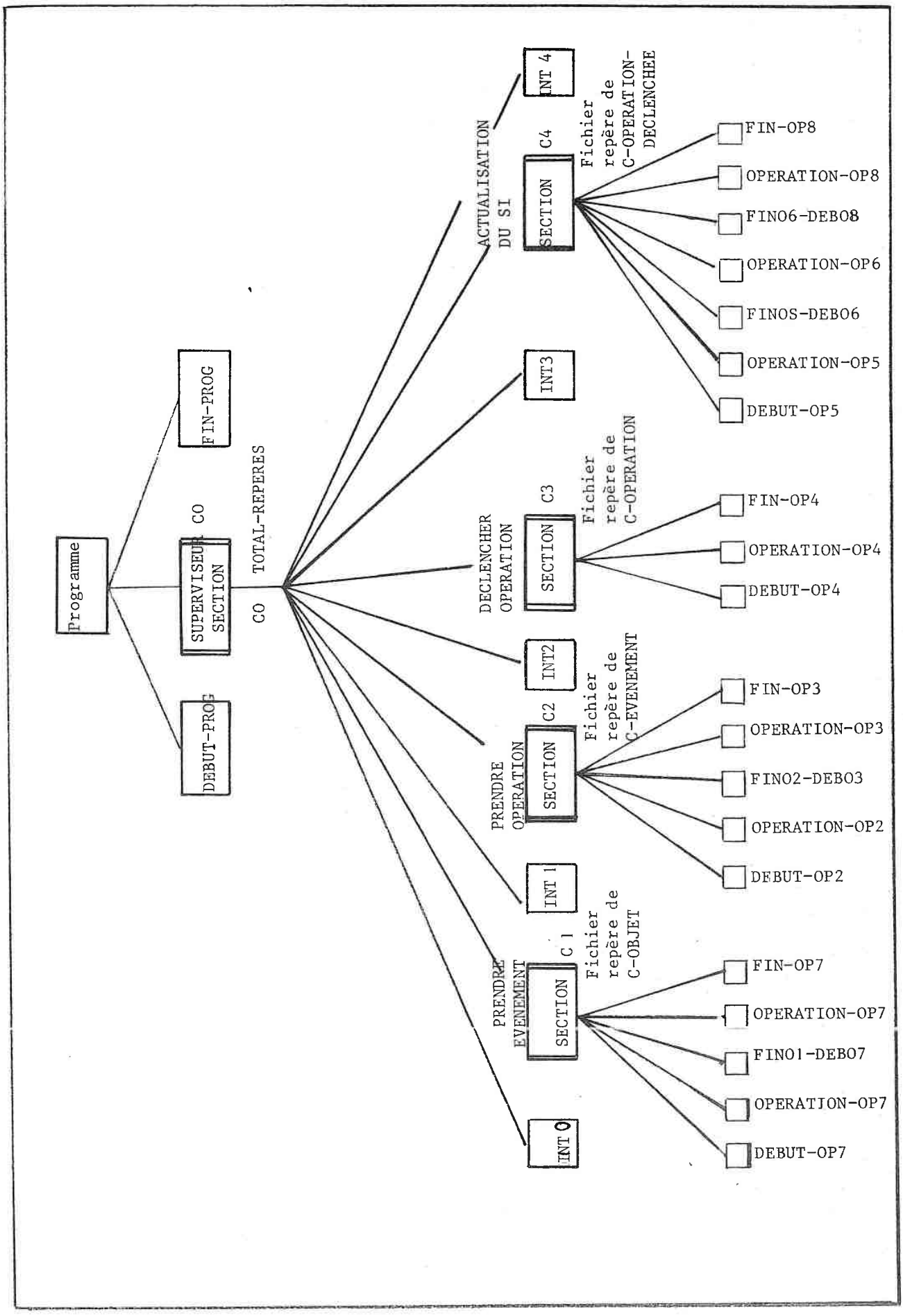
indique un paragraphe de traitements itératifs dont C est la condition d'arrêt

L'arbre programmatique doit être lu de haut en bas et de gauche à droite.

Les éléments nouveaux que l'on introduit sont :

TOTAL-REPERES : est un fichier COBOL qui contient le nombre total de repères à traiter.

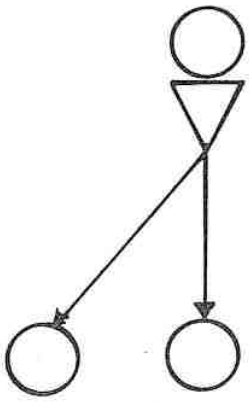
C0 : nombre total de repères à traiter est égal à zéro
C1 : nombre de repères de c-objet à traiter est égal à zéro
C2 : nombre de repères de c-événement à traiter est égal à zéro
C3 : nombre de repères de c-opération à traiter est égal à zéro
C4 : nombre de repères de c-opération-déclenchée à traiter est égal à zéro
INT0, INT1, INT2, INT3, INT4 : actualisent TOTAL-REPERES


Programme
SUPERVISEUR SECTION
CO
CO
TOTAL-REPERES
FIN-PROG
DEBUT-PROG
ACTUALISATION DU SI
INT 4
C4
SECTION
Fichier repère de C-OPERATION-DECLENCHEE
FIN-OP8
OPERATION-OP8
FINO6-DEBO8
OPERATION-OP6
FINOS-DEBO6
OPERATION-OP5
DEBUT-OP5
INT3
DECLENCHER OPERATION
C3
SECTION
Fichier repère de C-OPERATION
FIN-OP4
OPERATION-OP4
DEBUT-OP4
INT2
PRENDRE OPERATION
C2
SECTION
Fichier repère de C-EVENEMENT
FIN-OP3
OPERATION-OP3
FINO2-DEBO3
OPERATION-OP2
DEBUT-OP2
INT 1
PRENDRE EVENEMENT
C 1
SECTION
Fichier repère de C-OBJET
FIN-OP7
OPERATION-OP7
FINO1-DEBO7
OPERATION-OP7
DEBUT-OP7
INT 0

III.6.2. ANALYSE DES SECTIONS

III.6.2.1. Section prendre-événement

Cette section : correspond aucycle dynamique décrit par le schéma suivant



A1 repère de c-objet

EV1 création d'un repère de c-objet

C1 il existe un c-événement

OP7 : actualisation du SI

OP1 reconnaissance de c-événement

A8-repère-de-c-objet-modifie

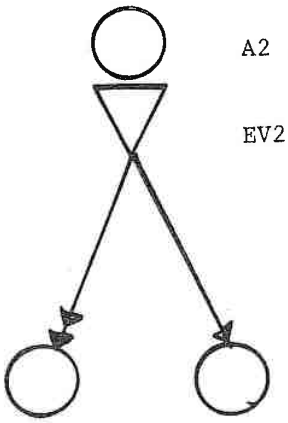
A2 repère de c-événement

Cette section contient les paragraphes suivants :

. DEBUT-OP1 : évaluation de la condition de déclenchement C1
. OPERATION-OP1 : évaluation du texte de la a-opération OP1
. FIN-OP1 : a un rôle mnémonique
. DEBUT-OP7 : a un rôle mnémonique
. OPERATION-OP7 : évaluation du texte de la a-opération OP7
. FIN-OP7 : actualisation de A 1

III.6.2.2. Section prendre-opération

Cette section correspond au cycle dynamique décrit par le schéma suivant :



A2 : repère de c-événement

EV2 : création d'un repère de c-événement

C2 : il existe des des c-opérations à déclencher

F1 : les c-opérations à déclencher

OP2 : recherche des c-opérations à déclencher

A3 : repère de c-opération

OP3:actualisation du SI

A4:repère de c-événement-constate

Cette section contient les paragraphes suivants

. DEBUT-OP2 : évaluation de la condition de déclenchement C2 et le facteur de déclenchement F1.

. OPERATION-OP2 : évaluation du texte de la a-opération OP2 et

. FIN-OP2 : a un rôle mnémonique

. DEBUT-OP3 : a un rôle mnémonique

. OPERATION-OP3 : évaluation du texte de la a-opération OP3

. FIN-OP3 : actualisation de AZ

III.6.2.3. Section déclencher-opération

Cette section correspond au cycle dynamique décrit par le schéma suivant

A3 : repère de c-opération

EV3 : création d'un repère de c-opération

OP4 : déclenchement des c-opérations

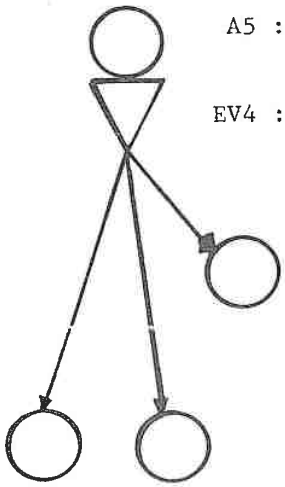
A5 : repère de c-opération-déclenche

Cette section contient les paragraphes suivants :

DEBUT-OP4 : a un rôle mnémonique
OPERATION-OP4 : évaluation du texte de la a-opération OP4
FIN-OP4 : actualisation de A3

III.6.2.4. Section actualisation-SI

Cette section correspond au cycle dynamique suivant :



A5 : repère de c-opération-déclenche

EV4 : création d'un repère de c-opération-déclenchée

OP5 : actualisation du SI

A6 : repère de c-opération-modifie

OP6 : actualisation du SI

A7 : repère de c-événement-déclenche

OP8 : reconnaissance d'un c-objet

A1 : repère de c-objet

Cette section contient les paragraphes suivants :

DEBUT-OP5 : a un rôle mnémonique
OPERATION-OP5 : évaluation du texte de la a-opération OP5
FIN-OP5 : a un rôle mnémonique
DEBUT-OP6 : a un rôle mnémonique
OPERATION-OP6 : évaluation du texte de la a-opération OP6
FIN-OP6 : a un rôle mnémonique
DEBUT-OP8 : a un rôle mnémonique
OPERATION-OP8 : évaluation du texte de la a-opération OP8
FIN-OP8 : actualisation de A5

IV. COMMENTAIRES SUR L'APPROCHE CONCEPTUELLE

Le but de ce chapitre est de présenter les principales conséquences que l'on peut attendre de la méthode structuraliste proposée et de l'outil que l'on a associé.

Ces commentaires concernent le système de gestion de systèmes d'information, la politique de conception des systèmes en informatique de gestion et la généralisation de l'approche structuraliste à la spécification du logiciel informatique.

IV. 1. SYSTÈME DE GESTION DE SYSTÈMES D'INFORMATION

IV.1.1. LE ROLE DU SGSI

Pour augmenter l'aide aussi bien à la conception que pendant l'exploitation du SI il est possible d'imaginer une extension des solutions proposées et en particulier d'intégrer le processeur de la dynamique dans un outil plus large qui remplirait les fonctions suivantes :

. réaliser automatiquement le SI, c'est-à-dire, créer la base de données et générer les programmes à partir du langage ISDEL
. gérer l'évolution du SI par le biais du processeur de la dynamique, c'est-à-dire, commander et contrôler l'exécution des programmes
. permettre l'utilisation du SI, c'est-à-dire, rendre aux gestionnaires les moyens d'intreface avec la base de données SI.

Le SGSI satisfait les objectifs suivants :

i) réduire le travail du concepteur à celui de l'analyse et de la représentation des faits réels.

Le concepteur ne doit pas se préoccuper des choix techniques liés à l'implémentation physique du SI qu'il décrit. Le SGSI doit être capable de réaliser l'implémentation du SI à partir de sa description conceptuelle.

ii) assurer la cohérence du SI par une gestion automatique de son évolution

iii) faire en sorte que le SI soit en permanence l'image de la réalité qu'il représente

Le SGSI doit prendre en compte tout événement survenu dans le système réel et d'exécuter dans le SI les opérations qui en sont la conséquence.

Nous remarquons que le rôle du SGSI est analogue à celui d'un Système de Gestion de Bases de Données Relationnel (SGBD Relationnel)

Les SGBD Relationnels ((AST 76)(CHA 76)(CLE 75)(CZA 75)(HEL 75) (SCH 76)(ZCO 77)(STO 76) entre d'autres) proposés dans la littérature (KIM 79) indiquent que les caractéristiques suivantes doivent faire partie d'un SGBD relationnel "idéal" :

"- interface avec un langage non-procédural de haut niveau qui permet aux utilisateurs, spécialisés ou non, de : définir les données, manipuler les données, interroger les données, contrôler l'accès aux données et leur intégrité.

- structures de fichiers et d'accès efficaces
- un optimiseur des expressions du langage non-procédural
- possibilité de "vues" et de "fenêtres" sur la base de données (DAT 77, STO 76, NYLO 75)
- contrôle d'intégrité sur les états de la base ou sur "transactions" (ESCU 76, BUN 77, DAT 77)
- contrôle d'accès sélectif

- récupération de pannes de logiciel ou machine
- "report generator" de très grande souplesse "

Si le rôle du SGSI est analogue à celui des SGBD Relationnels il faut reconnaître que les fonctions qu'il assure sont plus complètes. Il ne se charge pas seulement de la manipulation cohérente des données, il contrôle les opérations sur les données et il contrôle les événements qui déclenchent ces opérations.

Pour satisfaire ces objectifs nous pensons que le SGSI doit satisfaire les fonctions suivantes :

- génération de la structure interne
- création du système d'information
- gestion de l'évolution
- utilisation

La réalisation du SGSI dépasse les objectifs de ce travail. Toutefois nous nous proposons d'illustrer le principe de ses fonctions et de son architecture pour bien montrer d'une part son utilité et d'autre part le rôle que doit y jouer le processeur de la dynamique.

IV.1.1.1. Génération de la structure interne

Cette fonction récouvre plusieurs sous-fonctions.

Il s'agit de choisir le mode d'implémentation physique de la collection de relations qui constitue la description conceptuelle d'un SI. Les c-objets, c-opérations et c-événements dans le SI sont en effet décrits de la même façon dans le langage ISDEL : par des relations. On peut donc les implémenter de la même façon.

La structure interne du SI est la structure physique choisie pour l'implémentation des relations. Le SGSI utilise un modèle de structuration physique et les règles de mapping entre le modèle externe (relationnel) et le modèle interne.

Evidemment plusieurs autres facteurs, comme le nombre d'attributs des relations ainsi que la fréquence et le volume des modifications qui peuvent se produire doivent également être prise en compte dans la génération de la structure interne.

Il s'agit ensuite de générer les textes des opérations, les prédicats définissant d'une part les conditions et facteurs de déclenchement et d'autre part les états initiaux et finaux des événements ainsi que les contraintes d'intégrité dans un langage exécutable par le système et compatible avec la structure d'implémentation choisie.

Si nous nous plaçons dans le cas du prototype du processeur de la dynamique nous considérons que les relations de la structure conceptuelle du SI sont implémentés selon le choix interne fait par le SGBD SYNTEX. Les textes sont générés directement en COBOL, comme langage compatible avec SYNTEX à l'heure actuelle.

La figure IV.1.1.1.a. nous montre d'une façon globale quels sont les éléments de l'architecture du SGSI liés à la fonction génération de la structure interne.

IV.1.1.2. Création du système d'information

Cette fonction comprend le stockage des occurrences du SI conformément à sa structure.

Elle est contrôlée par l'exécution des contraintes d'intégrité générées par la fonction précédente.

Les occurrences sont introduites dans le SI à partir de son extérieur et elles doivent être prises en compte par le processeur de la dynamique. Il faut donc disposer d'une interface entre le module de saisie et le processeur de la dynamique.

Dans le cas du prototype du processeur de la dynamique implémenté nons considérons que la communication entre ces deux fonctions est faite par le biais du fichier COBOL de repères de c-objet.

Chaque fois que le responsable de la création du SI insère une occurrence de c-objet dans le SI il le signale au processeur de la dynamique par l'actualisation du fichier de repères de c-objet.

La figure IV.1.1.2.a nous montre quels sont les éléments de l'architecture du SGSI liés à la fonction création du SI.

IV.1.1.3. Gestion de l'évolution

Cette fonction doit assurer une évolution du SI conforme à l'évolution de la réalité qu'il représente.

Elle est associée par le processeur de la dynamique que nous avons présenté dans ce travail.

La figure IV.1.1.3.a. nous montre quels sont les éléments de l'architecture du SGSI liés à la fonction gestion de l'évolution.

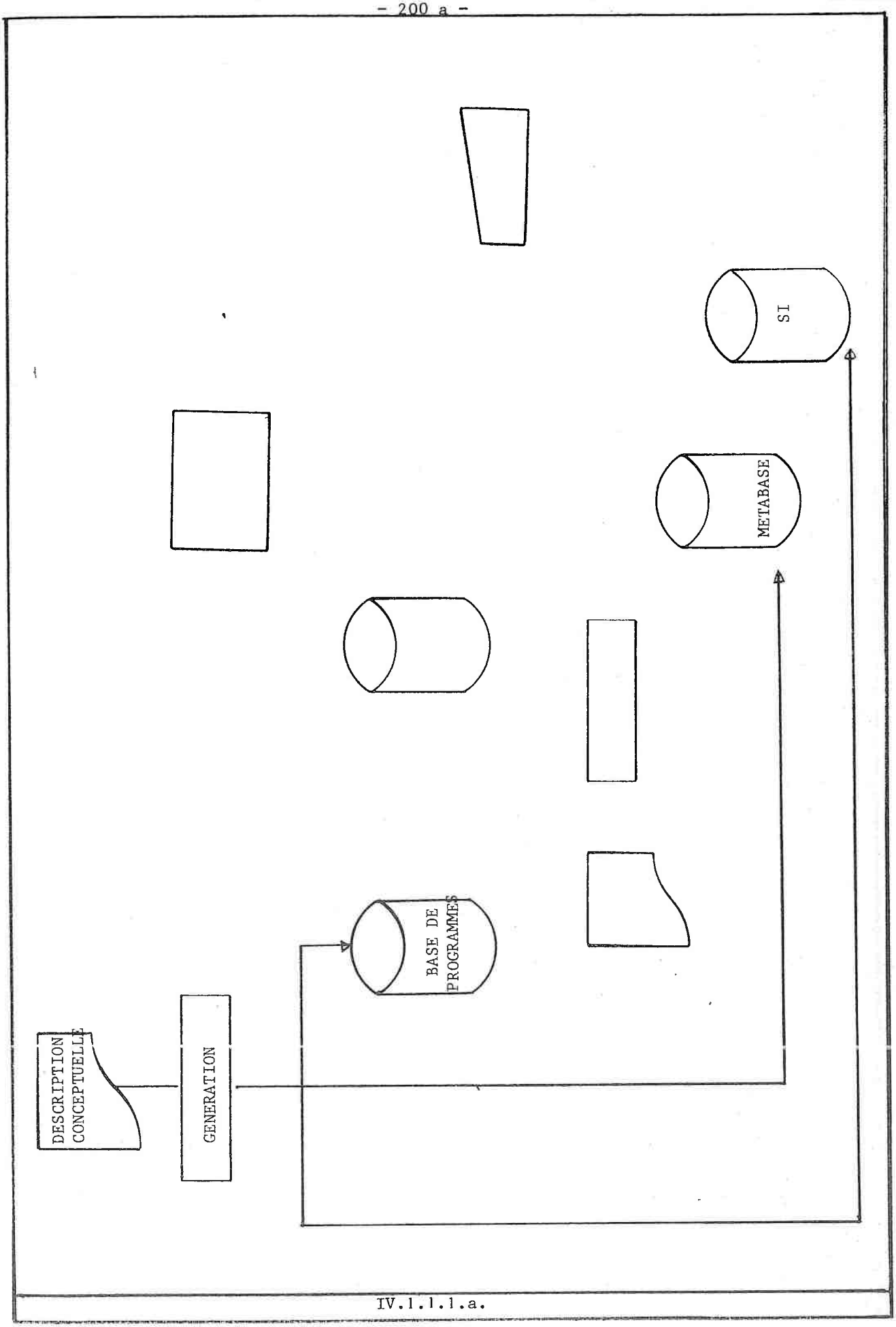
IV.1.1.4. Utilisation

Cette fonction doit assurer aux utilisateurs du SI, qu'ils soient spécialisés ou non, la possibilité de l'interroger et d'exploiter ses données.

Il faut noter que le passage de la base de données au SI étend considérablement le champ de renseignements que l'utilisateur peut obtenir sur l'organisation.

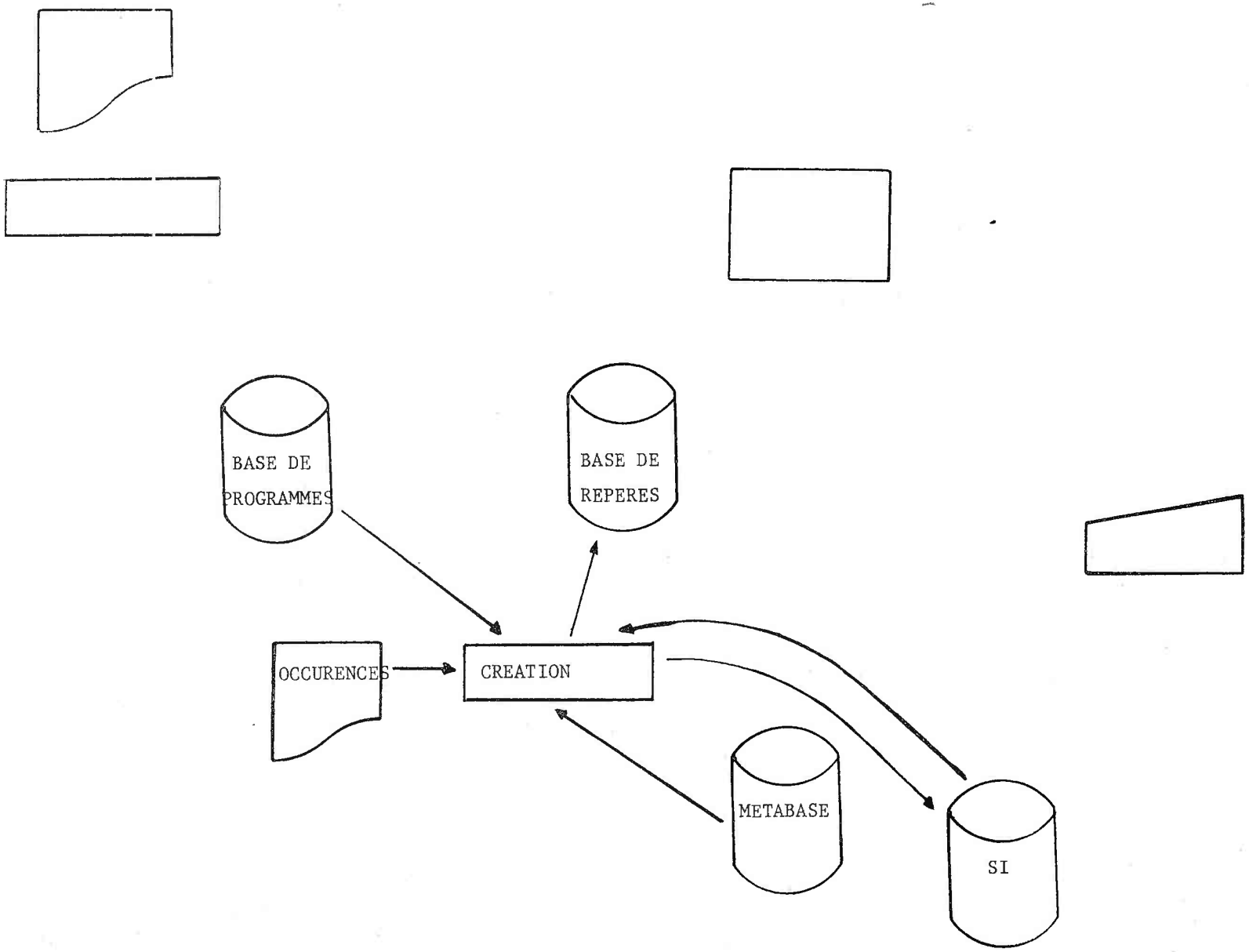
Il peut en effet être informé indifféremment sur l'état des objets, sur l'exécution d'une opération, sur l'arrivée d'un événement et plus généralement sur la séquence des causes et des conséquences de tout changement d'état du système.

Il peut, en plus, être informé sur la description conceptuelle du SI

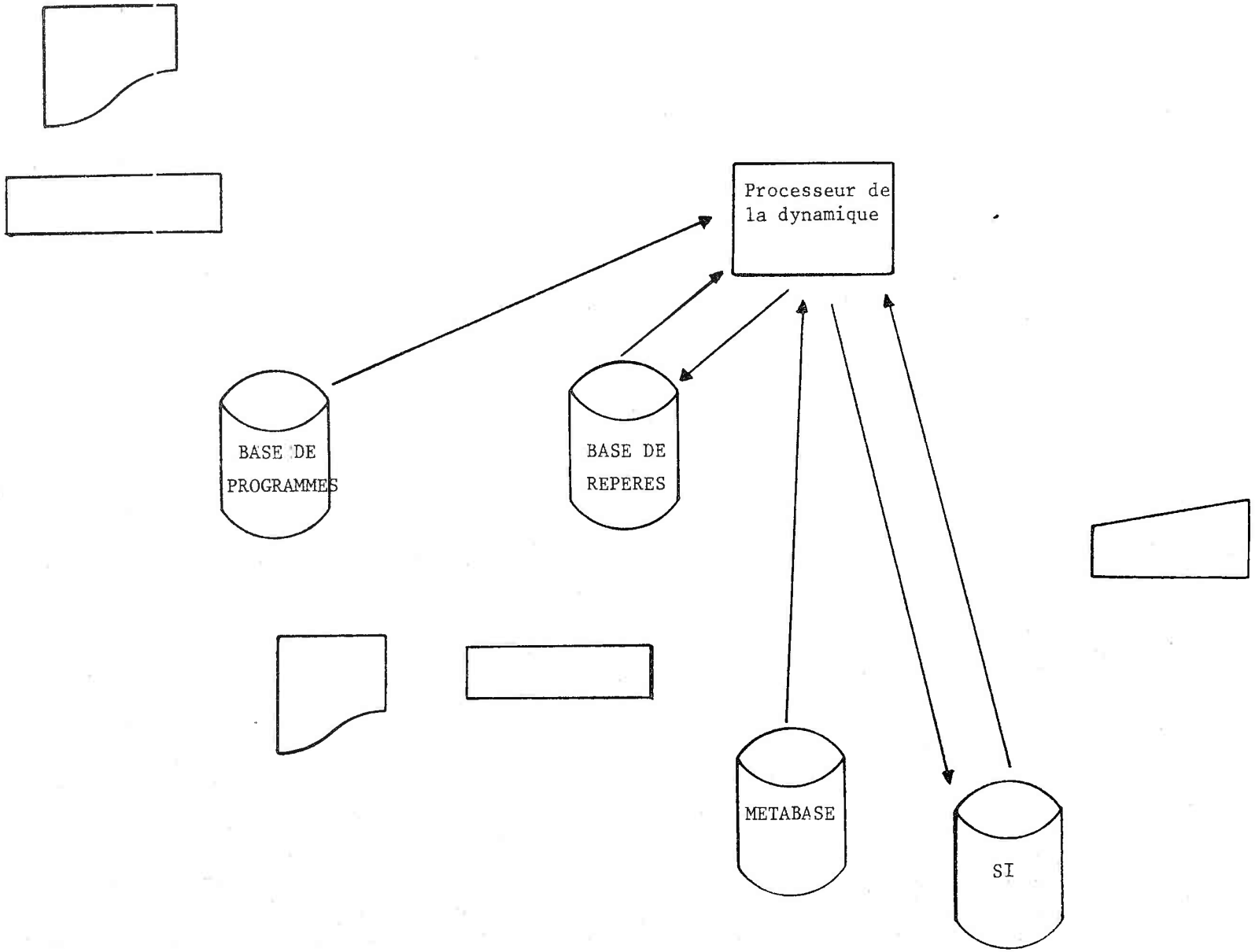

SI
METABASE
BASE DE PROGRAMMES
GENERATION
DESCRIPTION CONCEPTUELLE

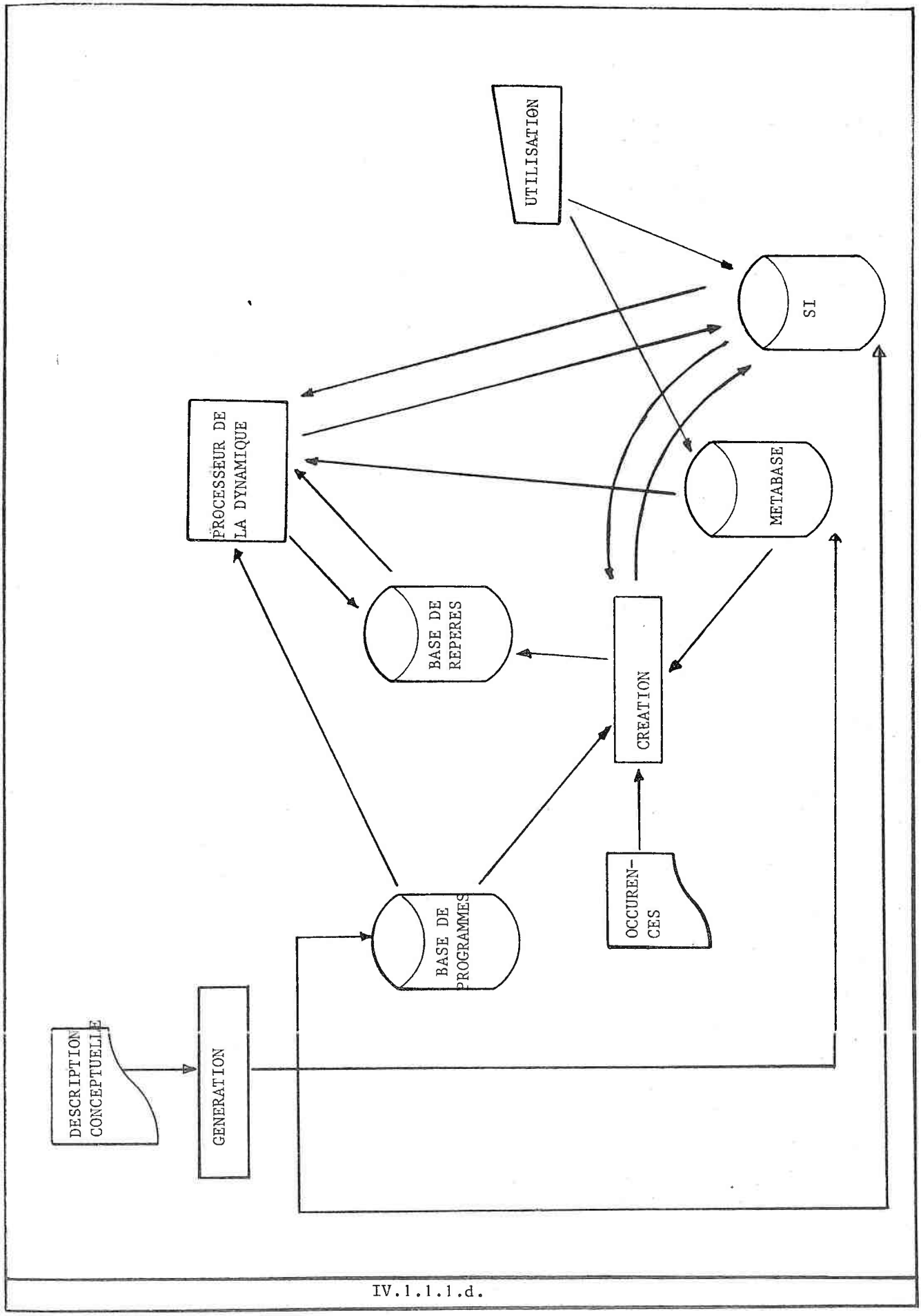
IV.1.1.1.a.

IV.1.1.1.b.


BASE DE PROGRAMMES
BASE DE REPERES
OCCURENCES
CREATION
METABASE
SI

IV.1.1.1.c.


Processeur de
la dynamique
BASE DE
PROGRAMMES
BASE DE
REPERES
METABASE
SI


UTILISATION
SI
PROCESSEUR DE LA DYNAMIQUE
METABASE
BASE DE REPERES
CREATION
OCCUREN-CES
BASE DE PROGRAMMES
DESCRIPTION CONCEPTUELLE
GENERATION

IV.1.1.1.d.

Le passage de la base de données à une représentation plus complète des phénomènes réels élargit l'information distribuée et augmente le pouvoir résolutoire des gestionnaires.

Dans le cas du prototype du processeur de la dynamique le SI peut être exploité comme une base de données SYNTEX soit directement par le langage SYNTEX soit par un programme COBOL.

La figure IV.1.1.4.a. nous montre l'architecture complète du SGSI.

IV.2. UNE POLITIQUE POUR LA CONCEPTION DES SYSTEMES EN INFORMATIQUE DE GESTION

Le système de gestion de systèmes d'information ne représente pas seulement une extension des systèmes de gestion de bases de données relationnelles.

Il est l'outil informatique qui permet la réalisation d'un système d'information dont la conception et l'utilisation est faite dans un cadre différent de celui proposé par l'informatique classique.

IV.2.1. La politique classique

Un système d'information est un système artificiel que les gestionnaires construisent pour disposer à tout instant des informations dont ils ont besoin pour gérer l'entreprise.

En conséquence la plupart des méthodes de conception de systèmes d'information ont comme point de départ l'analyse des besoins des utilisateurs. Le système d'informations est construit comme un ensemble de données et un ensemble de programmes nécessaires à la satisfaction de ces besoins.

Nous considérons que cette approche présente deux inconvénients principaux :

- "introduction de bruit" : pour optimiser les programmes et pour s'adapter aux besoins des gestionnaires on ajoute de nouvelles données, on

introduit des synonymies (les mêmes données ont différents noms), des polysemies (différentes données ont le même nom). On accepte la redondance des données qui genère au cours des actualisations successives des incohérences .

- introduction de "confusion" : on perd la correspondance entre le système réel et sa représentation par les données car on ajoute des propriétés aux données que n'ont pas de correspondance dans la réalité afin de "mieux utiliser le système d'information".

L'utilisation des bases de données a beaucoup réduit la plupart des problèmes liés à l'introduction de "bruits" et celle des "confusions".

Les travaux actuels sur la dynamique dans les bases de données et sur la conception conjointe des données et des programmes (BOD 79)(BRA 78)(BRE 79)(KOU 79)(LIN 79)(ROC 78) vont dans le sens de réduire "l'introduction de confusion".

Nous adhérons à cette démarche et notre travail en est la preuve.

IV.2.2. Proposition d'une politique

Nous pensons que la conception de systèmes en informatique de gestion doit aboutir à :

i) un système d'information conçu avec soin, par des concepteurs spécialisés, en termes de représentation du système réel

ii) un système d'utilisation flexible, adaptable, souple,conçu et réalisé par les utilisateurs au fur et à mesure des besoins.

Nous postulons qu'un système d'information doit être conçu comme l'ensemble minimal des données et des règles qui le font évoluer. Cet ensemble doit être à tout instant une représentation correcte, fidèle et cohérente de l'organisation.

Nous nous proposons donc de définir un système d'information comme une représentation codée de la réalité au lieu de le définir à partir de la liste des besoins, à un moment donné, des gestionnaires.

Par définition, si la représentation a les qualités présentées alors elle permet de satisfaire tous les besoins de n'importe quel gestionnaire par extraction du SI.

Cette extraction doit être possible à l'aide d'un ou plusieurs langages qui permettent que le gestionnaire, seul ou avec l'aide d'un spécialiste, définisse les solutions qui permettent la satisfaction de ses besoins à partir des informations contenues dans le SI.

Evidemment on peut envisager plusieurs outils, tels que générateurs d'états, interfaces graphiques et d'autres qui rendent plus adaptés aux besoins des gestionnaires les informations qu'ils extraient par le langage d'utilisation.

Dans le prototype que nous proposons ce langage d'utilisation c'est le langage SYNTEX ou COBOL dans les cas d'extractions et formattage plus complexes.

La principale caractéristique de la politique proposée est de bien distinguer la représentation de l'organisation, des besoins que cette représentation doit satisfaire.

Cela nous permet d'identifier les impacts dus aux changements de la représentation et ceux qui sont liés à l'évolution des besoins.

L'évolution de la représentation est due soit à l'évolution des états de l'organisation soit à l'évolution de la structure de base de l'organisation. La première est traduite par les occurrences du SI et la deuxième par l'évolution de sa structure conceptuelle.

L'évolution des besoins des gestionnaires est permanente et imprévisible. Elle est responsable, dans la politique classique, des côuts d'entretien prohibitifs des systèmes informatiques développés.

Nous pensons que notre approche simplifie les problèmes de maintenance et permet un traitement à la fois plus rapide et plus économique quand on considère l'utilisation globale des systèmes informatiques.

IV.3. GÉNÉRALITÉS DE L'APPROCHE DE SPÉCIFICATION CONCEPTUELLE

IV.3.1. APPROCHE CONCEPTUELLE

Le processeur de la dynamique est un logiciel qui gère un ensemble de représentations. A ce titre c'est un outil tout-à-fait général, de la même nature que la plupart des logiciels informatiques.

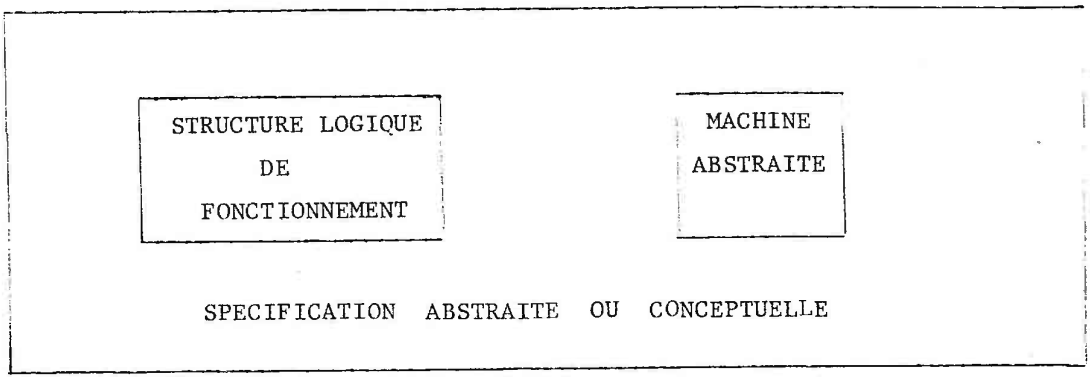
Indépendemment de la façon dont il est construit, le rôle d'un logiciel consiste :

- soit à gouverner (gérer, contrôler)un système physique dont il devient l'organe de commande en prenant automatiquement toutes les décisions relatives à son fonctionnement. Il est évident que pour cela il importe que le logiciel intègre de manière appropriée la logique de fonctionnement du système physique dont on lui confie le contrôle.
- soit à gouverner un système abstrait (ensemble de représentation d'un système réel) de telle façon que l'évolution du système abstrait soit conforme (équivalente à quelques décalages temporels près puisqu'il y a des temps morts inévitables ou voulus) à l'évolution du système réel représenté. Ici aussi il est évident que le logiciel doit intégrer de manière appropriée la logique de fonctionnement du système réel.

On est ainsi naturellement conduit à distinguer dans le logiciel

- les mécanismes opératoires d'une part
- la logique de fonctionnement d'autre part

et pour atteindre le niveau de généralité souhaité par rapport aux moyens et solutions techniques liés à l'implémentation du logiciel on aboutit à la solution que nous avons retenue pour le processeur de la dynamique.



IV.3.2. L'APPROCHE CONCEPTUELLE EST DE TYPE STRUCTURALISTE

Cette approche est de type structuraliste puisqu'on essaye de spécifier le logiciel à partir de l'analyse de la structure de fonctionnement et non de la collection de fonctions que le logiciel utilise pour agir. Il faut donc analyser et décrire tous les aspects du système (physique ou abstrait) que le logiciel gére : les aspects statiques par les constituants du système et leur structure ; les aspects dynamiques relatifs à l'évolution du système à travers les règles de fonctionnement qui le font passer d'un état à l'autre.

Elle rejoint l'approche des automaticiens (VAL 77)(VAL 80) ou l'on retrouve une séparation entre la description des composants du système que l'on gère (la partie données) et la description du mécanisme de gestion de l'évolution du système (la partie commande). Plusieurs techniques et modèles ont été développés pour décrire la partie données et la partie commande, et on peut comme en (VAL 80) distinguer parmi elles le graphe de données (ROU 78) pour la partie données et les réseaux de Petri pour la partie commande (AGE 73).

Le travail développé dans (RIC 79) a montré que tout système réel modelisé sous forme d'un réseau de Petri à ares inhibiteurs (AGE 73)(AGE 74) est exprimable par une structure conceptuelle basée sur le modèle conceptuel de la dynamique et que la notion de marquage d'une place (PET 77)(PETRI 74) (DEN 70) est exprimable par les mécanismes de la machine abstraite. Ainsi, dans le cas de l'analyse et de la modélisation d'un système réel, le modèle conceptuel de la dynamique a la même puissance de modélisation qu'un réseau de Pétri à arcs inhibiteurs, c'est-à-dire en fait la puissance de la machine de Turing (AGE 73)(AGE 74)(PET 77).

Cela nous conduit à dire que l'utilisation de l'approche préconisée est valable pour la spécification conceptuelle d'un logiciel qui gère un système physique ou abstrait que l'on peut modéliser par un réseau de Pétri à arcs inhibiteurs.

Le logiciel ainsi spécifié est un processeur discret d'information (GLU 73) composé par une structure conceptuelle que définit la partie contrôle du processeur et une machine abstraite qui définit sa partie opératoire.

Nous le montrons dans le cas du processeur de la dynamique, après un bref rappel de la théorie des processeurs discrets (GLU 73).

IV.3.2.1. Le processeur de la dynamique : un processeur discret

IV.3.2.1.1. Eléments de la théorie des algorithmes et des processeurs discrets

Une des principales idées de la théorie des processeurs discrets est celle d'un schéma général pour la représentation des algorithmes basée sur le concept de traitement discret de l'information. Le concept de processeur discret exprime, d'une manière générale, une méthode de spécifier la séquence des actions effectuées par un algorithme pendant le déroulement d'un processus et il est défini par rapport aux automates discrets.

. soient A, X, Y trois ensembles. A est appelé un X-automate s'il existe une application (en général partielle) A :

$$\delta_A : A \times Y \longrightarrow A$$
$$(a,x) \longmapsto A(a,x) = ax$$

Cette application δ_A de l'ensemble AxX dans A s'appelle la fonction transition de l'automate.

. Automate de Mealy : un X, Y-automate (ou automate de mealy) est le non donné à un X-automate pour lequel il existe une fonction de soutie (fonction "output") λ_A :

$$\lambda_A : A \times X \longrightarrow Y$$
$$(a,x) \longmapsto \lambda_A (a,x) = y$$

Cette fonction λ_A est définie sur le sous-ensemble de A x X où δ_A est définie. Elle est donc définie pour les couples (a,x) pour lesquels δ_A est définie et seulement pour ces couples.

. Automate de Moore : un X, Y-automate A est appelé un automate de Moore si A (a,x) ne dépend pas de x. Dans ce cas on a une fonction de sorie μA :

$$\mu_A : A \longrightarrow Y$$
$$a \longmapsto \mu_A (a) = y$$

Les éléments des ensembles A, X, Y sont appelés, respectivement, les états, les signaux en entrée, les signaux en sortie.

. Un automate est dit complétement défini si la fonction transition est définie pour tous les pairs (a,x).
Un automate A est dit initial si on y peut identifier un état initial.

. Processeur Discret :

Soit A un automate de Mealy.
Soient X sont alphabet en entrée et Y son alphabet en sortie
Soient a0 son état initial et a^* sont état final.

Dans la théorie des automates abstraits les éléments de X et Y n'ont pas de sens spécial. Si on veut utiliser les concepts de la théorie des automates pour décrire des processus algorithmiques il faut interpréter les signaux en entrée de l'automate comme les signaux concernant l'information qui doit être manipulée et les signaux en sortie de l'automate comme les actions réalisées par l'algorithme. Pour satisfaire cela on définit l'ensemble B, que l'on appelle ensemble "informationnel", dont les éléments, que l'on appelle objets "informationnels", sont l'information à être traitée par l'algorithme.

Pour chaque signal en sortie y ($y \in Y$) on établit la correspondance avec une transformation f_y (en général partielle) de l'ensemble B en lui-même. Pour certains éléments de B on établit une correspondance avec les signaux en entrée $x = \mu(b) \in X$, de l'automate A.

Si on donne la correspondance entre les signaux en sortie de l'automate A et les transformations de l'ensemble B et aussi la correspondance entre les éléments de l'ensemble B et les signaux en entrée de l'automate A, on dit que l'on a donné une interprétation de l'automate A.

Un automate avec un état final est appelé un processeur discret si on lui donne une interprétation. Dans ce cas on dit que le processeur discret agit sur l'ensemble B et que les transformations f_y sont appelées les opérateurs élémentaires de l'automate A.

Si les ensembles X, Y, et A sont finis, le processeur discret est dit fini.

Tout processeur discret d'information A définit une transformation f_A de l'ensemble B. Cette transformation est calculée par l'application de A aux éléments de l'ensemble B.

. Fonctionnement d'un processeur discret

Afin d'obtenir la valeur $f_A(b')$, c'est-à-dire, de connaître la valeur résultant de l'action du processeur A pur un élément $b \in B$, il est nécessaire d'initialiser l'automate A.

Alors à l'instant initial on applique le signal $x_1 = \mu(b)$ en entrée de l'automate. L'automate fait la transition à l'état $a_1 = a_o x_1$ et au même temps fournit le signal en sortie $y_1 = (a_o, x_1)$.

On applique la transformation f_{y1} à l'élément b et par ce moyen on obtient le nouvel élément $b_1 = f_{y1}(b)$.

A l'instant suivant le signal $x_2 = \mu(b_1)$ est appliqué en entrée de l'automate, lequel continue à opérer de manière analogue (figure I.4.1.a)

Le résultat est la génération des séquences $x_1, x_2, \ldots; y_1, y_2, \ldots, a_o, a_1, \ldots; b, b_1, \ldots$

Ces séquences peuvent être infinies ou se terminer après un certain nombre de cycles.

Si l'automate finit son travail quand il arrive à son état terminal après n étapes ($a^* = a_n$) alors on considère que la valeur de $f_A(b)$ est égale au dernier élément de la séquence $b, b_1, \ldots, b_n$. On dit que le processeur discret est applicable à l'élément b.

Dans le cas contraire on dit que le processeur discret n'est pas applicable à l'élément b.

Si les éléments de l'ensemble B sont des objets constructibles et les interprétations des fonctions de transition et de sortie de l'automate A sont données, le processeur discret A définit l'opération d'un algorithme alors que l'opérateur f_A définit la fonction calculée par cet alogorithme.

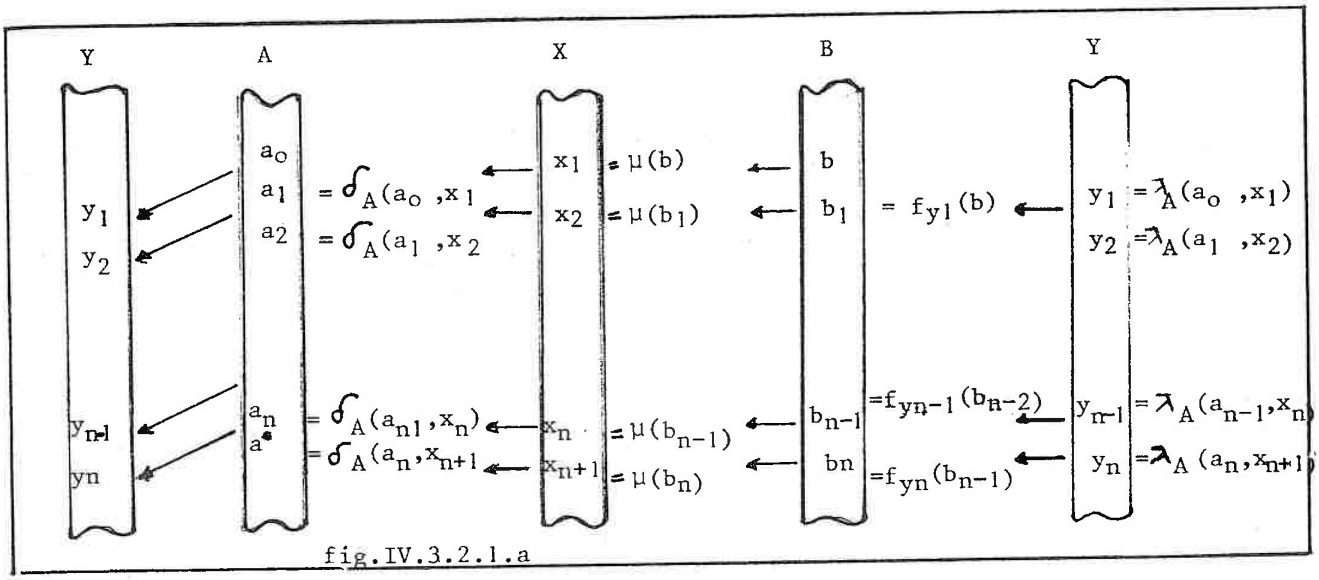


fig.IV.3.2.1.a

. Représentation de l'opération du processeur discret

Il est convenable de considérer l'opération du processeur discret dans le calcul de la fonction fA comme l'opération de deux automates A et B, connectés selon le schéma de la figure I.4.1.b.

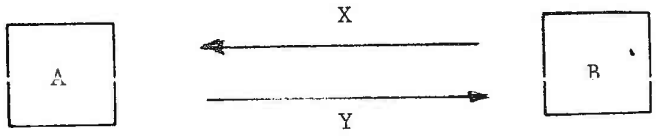


fig.IV.3.2.1.6.

L'ensemble B peut être considéré comme un automate de Moore avec alphabet en entrée Y et alphabet en sortie X si on définit sa fonction de transition δ_B par la relation $b_y = f_y(b)$ et si on prend μ comme sa fonction de sortie.

Si l'automate B est initialisé à l'état b alors le système des deux automates commence à fonctionner. Si l'automate A fait la transition à son état terminal à un instant donné, alors la valeur de la fonction $f_A(b)$ est donnée par l'état de l'automate B à cet instant.

L'automate A est appelé l'automate de contrôle (ou partie contrôle)

On dit que B est l'automate opératoire (ou partie opératoire). Quelquefois il est intéressant de prendre un automate de Mealy comme automate opératoire au lieu d'un automate de Moore. Dans ce cas, le processeur discret A est un automate de Moore. En plus, la fonction μ dépend non seulement des éléments de l'ensemble "informationnel" mais aussi des signaux de sortie du processeur discret A.

IV.3.2.1.2. Processeur de la dynamique et la théorie des processeurs discrets

La correspondance entre notre proposition pour concevoir le processeur de la dynamique et celle des processeurs discrets peut s'établir du fait que l'on peut exprimer le processeur de la dynamique comme un processeur discret.

On fait correspondre la structure conceptuelle à la partie contrôle et la machine abstraite à la partie opératoire d'un processeur discret représenté par deux automates A et B (fig IV.3.2.1.b.)

La structure conceptuelle correspond à un automate de Mealy et la machine abstraite à un automate de Moore.

L'automate de la partie contrôle est défini de la façon suivante :

. les états de l'automate sont définis par les éléments de la structure conceptuelle, c'est-à-dire, les C-OBJETS, C-EVENEMENTS, C-OPERATIONS.

. la fonction de transition δ_A est définie de la façon suivante :

- soit a un état de l'automate A
- si x correspond à un C-OBJET alors $\delta_A(a,x) = a'$ et a' est le C-EVENEMENT qui correspond au changement d'état du C-OBJET indiqué par x
- si x correspond à un C-EVENEMENT $\delta_A(a,x)=a'$ et a' est un élément de l'ensemble des parties défini sur les C-OPERATIONS associées au C-EVENEMENT indiqué par x.
- si x correspond à une C-OPERATION $\delta_A(a,x)=a'$ et a' est le C-OBJET qui correspond à la C-OPERATION indiquée par x

. la fonction de sortie A est définie de la façon suivante :

$$\lambda_A : A \times X \rightarrow Y$$
$$(a, x) \mapsto \lambda_A(a,x) = \delta_A(a,x)$$

L'automate de la partie opératoire est défini de la façon suivante :

. les états de l'automate sont définis par les occurrences du système abstrait de représentation associées aux éléments de la structure conceptuelle.

. la fonction de transition B est définie de la façon suivante :

- soit b un état de l'automate B
- si y correspond à un C-EVENEMENT alors $\delta_B(b,y) = b'$ et b' est la représentation de la constatation du C-EVENEMENT y
- si y correspond à une C-OPERATION alors $\delta_B(b,y) = b'$ et b' est la représentation du déclenchement de la C-OPERATION y
- si y correspond à un C-OBJET alors $\delta_B(b,y)=$ 'b' et b' est la représentation de la modification du C-OBJET y

. la fonction de sortie μ B est définie de la façon suivante :

- si δ_B (b,y) = b' et b' est la représentation de la constatation du C-EVENEMENT y alors μ_B (b') indique le C-EVENEMENT représenté par b'
- si δ_B (b,y) = b' et b' est la représentation du déclenchement de la C-OPERATION y alors μ_B (b') indique la C-OPERATION représentée par b'
- si δ_B (b,y) = b' et b' est la représentation de la modification du C-OBJET y alors μ_B (b') indique le C-OBJET représenté par b'.

Les définitions données respectent la définition d'un processeur discret car :

. il y a une correspondance entre les signaux de sortie de A (c'est-à-dire y $\in$ alphabet Y) et la fonction transition δ_B de l'automate B.
. il y a une correspondance entre les signaux en entrée de A (cest-à-dire, x $\in$ alphabet X) et la fonction sortie μ_B de l'automate B.
. la fonction de transition B est définie par la relation $b_y = f_y$ (b) et f_y (b) représente les mécanismes de la machine abstraite.

On constate que le processeur de la dynamique spécifié par la structure conceptuelle (qui décrit un système réel) et la machine abstraite (qui gère sa représentation) correspond à un processeur discret d'information, au sens présenté en (GLU 73).

Nous remarquons que la définition donnée à la partie opératoire, qui est spécifiée par la machine abstraite, montre bien son indépendance par rapport à une structure conceptuelle particulière. La fonction de transition de la partie opératoire est définie par rapport aux concepts du modèle de la dynamique et elle correspond aux transformations sur l'ensemble de représentation géré par la machine abstraite. La fonction sortie de la partie opératoire est elle aussi indépendante d'une structure conceptuelle particulière car elle est définie par rapport aux concepts du modèle.

La machine abstraite est donc spécifiée une fois pour toutes et si on veut caractériser le processeur de la dynamique d'un système que l'on peut décrire par une structure conceptuelle, il suffira de fournir la description de cette structure.

L'intérêt d'établir la correspondance entre le processeur de la dymique et les processeurs discrets est dû au fait qu'il nous est, en conséquence, possible d'appliquer des propriétés des processeurs discrets au processeur de la dynamique.

IV.3.3. INTERET DE L'APPROCHE STRUCTURALISTE

Notons que l'approche structuraliste s'oppose aux approches les plus souvent retenues pour la spécification et la réalisation de logiciels informatiques et qui sont de type fonctionnel.

Dans ces approches le logiciel est défini par l'ensemble des fonctions qu'il doit assurer. Elles ont été utilisées le plus souvent pour la conception des systèmes opératoires, systèmes de gestion de bases de données, compilateurs.

Notons que dans la plupart des cas elles conduisent à un raisonnement en termes de processus (fonctions) et de synchronisation entre processus.

Nous croyons que l'utilisateur de l'approche conceptuelle, surtout dans le domaine des organisations, correspond mieux aux exigences de la conception de logiciels, et cela pour plusieurs raisons :

Le nombre de fonctions, surtout lorsque l'on passe de la base de données au système d'information, est très grand et il devient difficile de s'assurer de l'exhaustivité du recensement.

En revanche, le nombre d'éléments composant le système d'informations est plus réduit et donc plus facile à reconnaître et à décrire.

L'intégration entre les fonctions est difficile et l'expérience des méthodes d'analyse a montré qu'on n'arrivait pas à la maîtriser.

Au contraire, les liens structurels entre les éléments qui composent le SI sont plus réduits.

La sémantique du SI est mieux explicitée lorsque l'on raisonne en termes de représentation du système réel.

Dans d'autres domaines, tels que les compilateurs, où il ne s'agit pas de représenter un système réel, les fonctions du logiciel ont été largement recensées et il nous semble que l'approche fonctionnelle est préférable.

En outre, remarquons qu'un intérêt de l'approche conceptuelle est de permettre une spécification unique de la machine abstraite pour traiter une certaine classe de problèmes.

Cela simplifie la tâche du concepteur car il n'a plus qu'à décrire la structure conceptuelle du système réel.

Enfin notons que la distinction entre la spécification conceptuelle du logiciel et sa réalisation présente un intérêt considérable par la flexibilité et l'adaptabilité qu'elle peut proportionner.

CONCLUSION

Le système d'information est un ensemble de représentation codée de l'organisation.

Cet ensemble évolue au cours du temps et nous avons présenté l'idée d'un outil, le processeur de la dynamique, qui assure que l'évolution du SI est l'image de la réalité.

L'intérêt de réaliser la spécification de cet outil à un niveau abstrait ainsi que la conviction que son action devrait être basée sur la logique de fonctionnement de la réalité nous ont conduit :

- à représenter la réalité à l'aide d'un modèle capable de décrire ses aspects dynamiques et statiques à un niveau conceptuel
- à proposer un langage pour réaliser la description de la représentation ainsi faite comme la structure conceptuelle d'un système d'information.

Nous avons conçu le processeur de la dynamique par le couple (machine abstraite, structure conceptuelle) car cela nous a permis d'isoler d'une part, les mécanismes qui réalisent l'évolution du SI et d'autre part la description de sa structure conceptuelle.

Le choix de la spécification de la machine abstraite par sa structure de fonctionnement nous a conduit à la réaliser dans le langage ISDEL. Cela a montré la généralité du langage proposé ainsi que l'intérêt que l'on peut avoir à utiliser l'approche structuraliste pour la conception de logiciels complexes destinés à la gestion de systèmes de représentations.

L'idée du processeur de la dynamique nous a conduit à proposer son intégration aux SGBD existants de façon à disposer d'une nouvelle classe d'outils pour la réalisation et la gestion des systèmes d'information : les Systèmes de Gestion des Systèmes d'Information. Nous avons essayé d'en présenter l'intérêt en développant un prototype basé sur le SGBD Syntex. Bien que nous n'ayons réalisé que l'aspect concernant le processeur de la dynamique nous croyons que l'intérêt porté par ce genre d'outil est tout-à-fait justifiable surtout par des conséquences sur la politique existante pour la conception de systèmes d'information et leur utilisation.

Nous croyons que la réalisation physique d'un logiciel conçu selon l'approche conceptuelle peut lui donner une très grande portabilité (le changement de machine réel n'altère pas la description conceptuelle du système représenté), faciliter sa maintenance (on identifie dans la structure des conséquences des changements), isoler au niveau de l'implémentation de la machine abstraite les problèmes liés à l'environnement (tel est le cas des bases de données réparties (ROL80).

Finalement, nous pensons qu'il y a encore plusieurs aspects à développer et nous considérons que les plus importants concernent la réalisation en vraie grandeur d'un SGSI, du compilateur ISDEL associé et la recherche des limites d'application de l'approche conceptuelle dans la spécification des logiciels.

B I B L I O G R A P H I E

AGE 74 - AGERWALA T "A complete model for representing the coordination of asynchronous processes".
Computer reserch report 32, John Hopkins University, Baltimore (Md), 1974.

AGE 73 - AGEWALAT, FLYNNM.
"Comments on capabilities, limitations and correctress of Petri Nets' Procedings of the 1rst annual Symposium on Computer Architecture Lipovski G.J. and Szygenda S.A. editors, A.C. M.; New York 1973.

- ABR 78 ABRIAL, RJ : A specification language
IFIP 1978, Tokyo - Japon

- ARS 79 ARSAC J., KODRATOFFY : Some methods for transformation of recursive procedures into iterative ones, Rapport 79-2, LITP, Paris 1979

- ART 74 ARTAUD, NICOLAS J.N. : An experimental query system : SYNTEX.
Proc. Int. Computing Symposium-Northe-Holland.

- AST 76 ASTRAHAM and All : System R : a relational approach to data base management ACM TODS-1976

- BAN 79 BANON D : Projet REMORA : un outil pour la gestion des systèmes d'information. Thèse de 3ème cycle, Nancy.

- BAU 79 BAUER FL, PEPPER P, WOSSMER H. : "Notes on the project CIP : outline of a transformation system, in Program Construction, FL BAUER et N BROY (ed). Lecture Notes in Computer Science 69, Springer Verlag - 1979.

- BEL 79 BELLEGARDE F, QUERE A, REMY JL : "construction et transformation systématiques de programmes" - rapport CRIN 79-R-045 - 1979.

- BER 78 BEERI C., BERNSTEIN A, GOODMAN N : "A sophisticate's introduction to data base normalization theory.
VLOB, Berlin - Allemagne - 1978.

- BEN 76 BENCI G, and all : "concepts for the design of a conceptual schema"
IFIP Conference - Freudenstadt - Allemagne - 1976.

- BEN 76 b BENCI G., ROLLAND C. : Cours de conception des Bases de Données.
Revue Technique de l'Ingénieur.

- BEN 79 BENCI G, ROLLAND C : "les bases de données : conception canonique pour une réalisation extensible" - SCM Editeur - Paris 1979.

- BRE 79 BREUTMAN B, FALKENBERG E, MAUER R : "CSL : a language for defining conceptual schemas" - IFIP TC2 Working conference on data base architecture, Venise, Italie 1979.

- BRI 75 BRINCH-HANSEN P ; "the programming language concurrent Pascal" - IEEE-SE 1,2 (June 75)

- CHA 76 CHAMBERLIN DD and all : "SEQUEL 2 : A unified approach to data definition manipulation and control".
IBM Journal - Res. Develop. 20-6 - 1976.

- CHE 76 CHEN PPS : The entity relationship model toward a unified view of data - ACM/TODS Vol 1, N°1 - 1976.

- CLE 75 CLEOD D.J. : HELDMANN M.J. : "A generalized minicomputers relational data base management system"
Proceedings NCC - 1975.

- COD 70 CODD E.F. : A relational model of data for large shared data banks . CACM, vol. 13, n° 6.

- COD 71 CODD E.F. : Normalized Data base structure : a brief tutorial - ACM SIGFIDET Workshop on data description, acces and control, San Diego, California.

- COD 74 DODD E.F. : Recent investigation in Relational Database systems.
Proc. IFIP Congress 74, North-Holland Publication, Amsterdam.

- COU 73 COUGER D. : Evolution of business system analysis techniques.
Computing surveys vol 5, n° 3, septembre , pp 190-198.

- CRO 75 CROCUS : "Systèmes d'exploitation" Dunod 1975.

- CZA 75 CZARNIK B. and all : ZFTA : a relational data base management system
Proceedings of ACM - Pacific conference - 1975.

- DAT 77 DATE C.J. : An introduction to database systems. Addison Wesley

- DEL 75 DELOBEL C. : Contribution théorique à la conception et à la définition d'un système d'information appliqué à la gestion. Thèse de doctorat ès Sciences, Grenoble.

- DEM 75 DEMOLOMBE R., LEMAITRE M., NICOLAS J.M. : SYNTEX 2 - Rapport DRME - n° 94222.

- DEM 78 DEMOLOMBE R., LEMAITRE M., NICOLAS J.M. : The language of SYNTEX 2 an implemented relationnel like DBMS. Proc. JCI 73. Monita Ed. August. North-Holland.

- DEM 79 DEMOLOMBE R. : Semantic checking of questions expressed in predicate calculus language. Proc. 5th Int. Conf. on VLDB, Rio de Janeiro.

- DEN 70 DENNIS J.B. (editor) "Record of the project MAC conference on concurrent Systems and parallel computation" A.C.M. ; New York, 1970.

- DIJ 75 DIJKSTRA EW : guarded commands - non determinacy and formal derivation of programs CACM - Vol 18 - N°8 - 1975.

- DON 72 DONAVAN J.J. : "systems programming - Mac Graw Hill - 1972

- DON 74 DONAVAN J.J., NADNICK S : Operation systems - Mac Graw Hill - 1974.

- DOY 76 DOYLE P. : Every object is a system, a cognitive basis for system description. Patrick Doyle Publisher.

- FLO 77 FLORY A. : Un modèle et une méthode pour la conception d'une Base de Données. Thèse de doctorat ès Sciences, Lyon.

- GOG 75 GOGUEN J.A., THATCHER J.W., WAGNER E.G., WRIGHT J.B. : "Abstract data types as initial algebras and the correctness of data representation - Proc. Conference on computer graphics, pattern recognition and data structure.

- GUT 77 GUTTAG J. : "Abstract data types and the development of data structures - Communication ACM vol 20, N° 6 - 1977.

- HEC 80 HECKENROTH H., TARDIEU H., ESPINASSE B. : Niveaux de modèles, formalismes et outils pour la dynamique dans les systémes d'informations. Séminaire Inforsid Groupe II, Aix-en-Provence, Janvier

- HEL 75 HELD G.D. and all : INGRES : "a relational data base system" - Proceedings NCC 1975

- HOA 78 HOARE CAR : "A model for communicating sequential processes" - Oxford University 1978.

- ICH 79 ICHBIAH J.D. et all : Rationate for the design of the ADA programming language. SIGPLAN NOTICES 14,6 (June 79).

- IFIP74 IFIP Working Conference on data structure for information system". NAMUR, BELGIUM.

- IFIP76 IFIP Working Conference on "Modelling in data base management systems", Freudenstadt, GERMANY.

- KEN 73 KENT W. : Describing information (not data reality). Technical report TRO 3012.

- KIM 79 KIM W. : Relational Database Systems. CACM, vol. 11, n° 3, September.

- KRI 78 KRIEGUER M. : Projet REMORA : modèle de représentation et méthode de construction des composants statiques d'un SI : Thèse de 3ème cycle. Nancy.

- LIN 74 LINDGREEN P. : Basic operations on information as a basis for Database design. Proceedings of IFIP WORKING Group TC-2, North Holland Publication, Amsterdam.

- LIS 74 LISKOV B., ZILLES SS : Programming with abstract data types - Very high language symposium - Santa Monica 1974.

- LIS 77 LISKOV B. etal. Abstraction mechanisms in CLV - CACM, vol 20, p. 564-576.

- LUG 75 LUGUET J. : SCAPFACE. Thèse d'Etat. Toulouse.

- MYL075 MYLOPOULOS J., SCHUSTER S., TSICHRITZIS D. : A multi-level relational system. Proc. AFIPS-NCC, vol 44, A FIPS Press, Montvale, NJ.

- PEC 75 PECOUD F. : MACSI. Thèse d'Etat, Grenoble.

- PET 77 PETERSON J.L. "Petri Nets". Computing Surveys, Vol 9 Nb 3, 1977.

- PE 73 PETRI C.A. "Concepts of Net theory"
Proceedings of Symposium and summer school on mathematical foundations of Computer Science, High Tatras, Sept. 3-8-1973, Mathematics Institute of Slovak Academy of Science, 1973.

- PIR 76 PIROTTE A. : Explicit description of intities and their manipulation in languages for the relationel data base model.
Thèse de Docteur en Sciences Appliquées. Univ. Libre de Bruxelles Belgique.

- PIR 77 PIROTTE A. : The entity-association model : an information oriented data base mode. Proc. ACM Int. Computing Symp, April

- RIC 79 RICHARD C. : Une approche conceptuelle des problèmes de synchronisation. Thèse de 3ème cycle, Nancy.

- ROL 76 ROLLAND C. : "Bilan de deux années de réflexion sur le projet REMORA". Colloque INFORSIDCaen, Public. IMA pp. 203-221.

- ROL 78 a ROLLAND C., FOUCAUT O. : Concepts for design of an information system conceptual schema and its utilisation in the REMRA project. Proceeding 4 th International Conference on VLDB , BERLIN, Septembre pp. 342-350.

- ROL 78b ROLLAND C : le projet Remora
Proceedings conference ATP CNRS - Rennes, France - 1978

- ROL 79a ROLLAND C. : Rapport final du contrat SESORI 76168 : PROJET REMORA, CRIN 79-R-040, Mars

- ROL 79b ROLLAND C., RICHARD C. : Concepts for a distributed computing system design - Seminary on distributed data sharing system : IRIA/IGDD - Aix-en-Provence - France.

- ROL 79c ROLLAND C., FOUCAUT O., LEIFERT S. : A proposal for information systems design and management. Rapport interne CRIN, Nancy. France.

- ROL 80 ROLLAND C., LEIFERT S. ; Un outil de gestion des systèmes d'information. Soumis à publication.

- ROU 78 ROUCAIROL G., Contribution à l'étude des équivalences syntaxiques et transformations de programmes parallèles. Thèse de Doctorat d'Etat, Univ. Pierre et Marie Curie, PARIS.

- ROU 75 ROUSSOPOULOS N., : A Semantic Network Model of Data Bases. Ph. D Thesis, Techn. Rep. TR 104, Univ. of TORONTO, September.

- SEN 75 SENKO M.E., : Spécification of Stored Data Structures and desined output results in DIAM II with FORAL, Proc. 1th Int. Conf. on VLDB, September.

- SCH 76 SCHID M.A. and all : The relational data base system OMEGA Progress Report - University of Toronto - 1976.

- SCH 77 SCHMIDT J.W. : Some high level language constructs for data of type relation. ACM-TODBS, vol 2, N°3

- SET 79 SETZER V.W. : Program development by transformations applied to relational data base queries. 5th Int. Conf. on VLDB. Rio de Janeiro - BRASIL.

- SHA 74 SHAWC : "The logical design of operating systems. Prentice Hall. Englewood Cliffs - N° 5 - 1974.

- SOC 77 SOCRATE II : Manuel d'utilisation
Ref F 24742 - CII HONEYWELL BULL - 1977.

- STO 76 STONEBAKER M. and all : The design and implementations of INGRES ACM TODS, Vol 1, 1976.

- SUN 75 SUNDGREN B. : Theory of Data Bases. Petnocelli/Charter.

- TEI 73 TEICHRDEW D. : An introduction to computer based information processing system, ISDOS Working paper 72.

- WAS 78 WASSERMAN A.I. : A software engineering view of data base management. Proc. 4th International Conference on VLDB, Berlin, September.

- WAT 75 WATTERS : "Methodologie assistée par ordinateur dans la conception des systèmes informatiques. L'Informatique Nouvelle, Janvier.

- WIR 71 WIRTH N. : The programming language PASCAL. Acta Informatica 1, p. 35-63.

- WIR 73 WIRTH N. : Systematic Programming : An Introduction. Prentice-Hall Series in Automatic Computation.

- WIR 76 WIRTH N. : Algorithmes + Data Structures = Programs - Prentice-Hall Series in Automatic Computation.

- WUL 77 WULF N, SHAWN, LONDON R.L. : "abstraction and verification in Alphard defining and specifying interactions and generators"- CACM, Vol 20, n° 8 - 1977.

- ZLO 77 ZLOFF MM. : Query-by-exemple : a data-base language,"IBM Systems Journal",n° 16, 4.

Annexe I

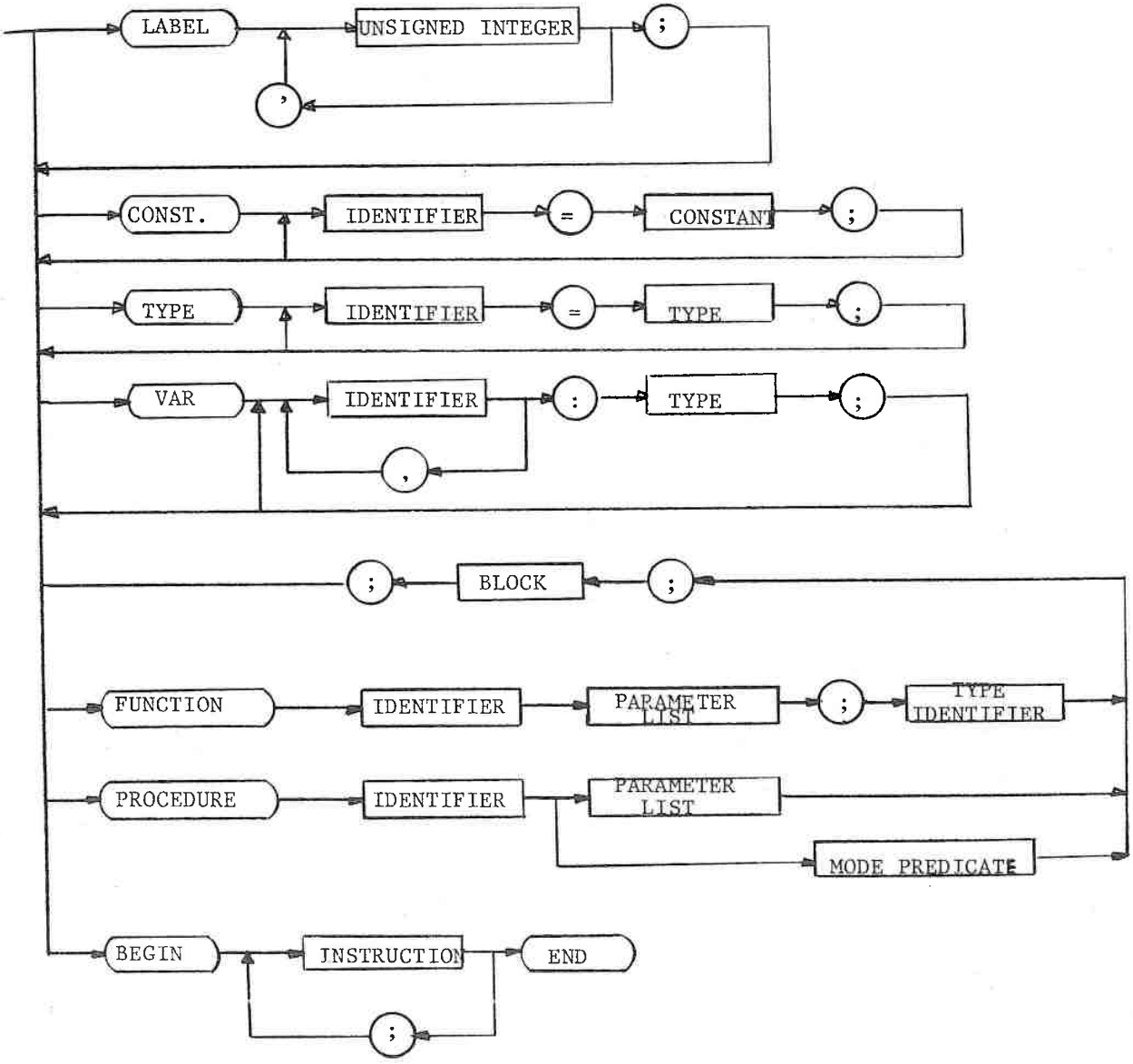
Nous présentons la syntaxe d'ISDEL sous une version de la forme normale de Backus (BNF) et sous la forme de diagrammes syntaxiques.

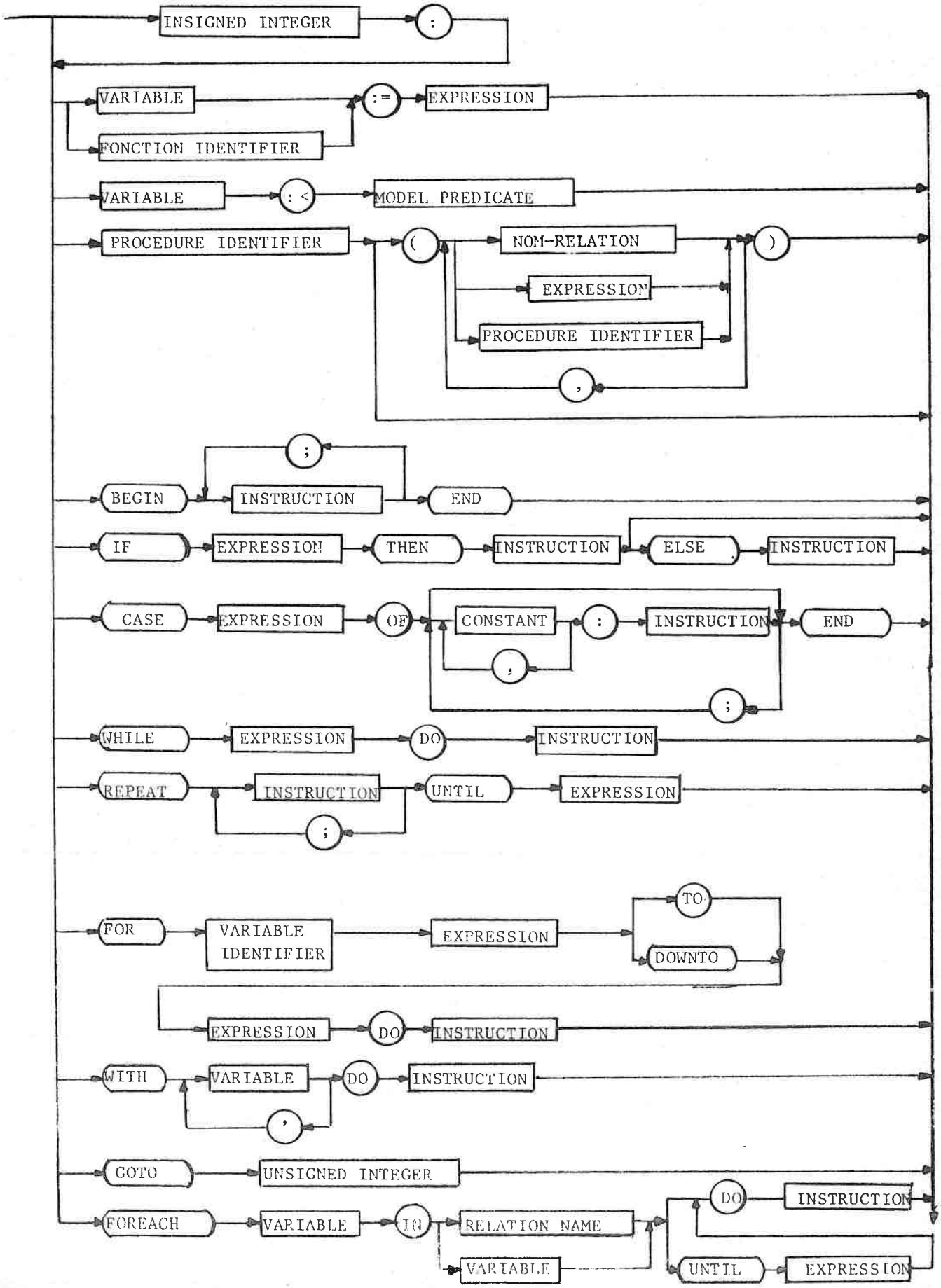
```
expression-ideal ::= requete/description
description ::= IS-TYPE is-type- ident specification-relation END
is-type-ident ::= COBJECT MODE COB/COPERATION MODE cop/
                  CEVENT MODE cev
cop ::= COPPER/COPTEXTE/COPMODIFIE
cev ::= CEVPER/CEVPRED/CEVCONSTATE/CEVDECLENCHE/CEVDECEFF
spécification-relation ::= spec-rel/spec-rel, specification-relation
spec-rel ::= nom-relation (liste-nom-champ) spec-domaine /spec-assertion/
nom-relation ::= identifier
liste-nom-champ ::= nom-champ /nom-champ ; liste-nom-champ
nom-champ ::= identifier/variable
spec-domaine ::= DOMAIN spec-domaine-def
spec-domaine-def ::= domaine-def/domaine-def ; spec-domaine-def
domaine-def ::= nom-champ : domaine-exp/nom-champ : SAME OF
                nom-relation
domain-exp ::= CHAR [(entier-sans-signe)] /
               INTEGER [(entier-sans-signe)] /
               REAL [(precision)] /
               DATE [(entier-sans-signe)] /
               BOOLEAN
precision ::= entier-sans-signe/entier-sans-signe, entier-sans-signe
spec-assertion ::= ASSERT ident-spec-list
ident-spec-list ::= spec-list-elem/spec-list-elem, ident-spec-list
spec-list-elem ::= identifier : nature-assert
nature-assert ::= assert-elementaire/assert-complexe : specification-
                  texte-assert
assert-elementaire ::= liste-nom-champ COMPOSKEY [nom-champ] /
                  nom-champ assert-desc
assert-desc ::= FINAL/INITIAL/ACTUAL/KEY/DATECRE
assert-complexe ::= nom-de-champ texte-identif /
                  CONSTRAINT (liste-variables-specif)
```

```
texte-identif ::= texte-id (liste-variables-specif)
texte-id ::= OPERATION/FACTOR/PREDICATE/CONDITION
specification-texte-assert ::= specification procedurale /
                               modele-de-predicat /
                               LIKE identificateur
liste-variables-specif ::= variable-specif /
                           variable-specif ; liste-de-variables-specif
variable-specif ::= ident-relation /ident-relation-det/
                    identifier : type identifier
ident-relation-det ::= ident-relation.nom-champ
ident-relation ::= nom-relation /
                   identifier
modele-de-predicat ::= (liste-elements) : predicat-def
liste-elements ::= element-simp / element-simp, liste-elements
element-simp ::= variable . nom-champ
predicat-def ::= definition-domaine /definition-domaine AND formula
definition-domaine ::= definition-relation /
                       definition-relation AND definition-domaine
definition-relation ::= nom-relation  <variable>
formula ::= predicat-facteur / disjonction
disjonction ::= predicat-facteur OR formula /
                predicat-facteur MAIS PAS formula
predicat-facteur ::= simple / conjunction
conjunction ::= simple AND predicat-facteur
simple ::= [NOT] boolean-primaire
boolean-primaire ::= predicat-expr / (formula)
predicat-expr ::= EXIST  <variable>(formula) /
                  ALL    <variable>(formula) /
                  comparaison-pred
comparaison-pred ::= expr comp-op element / element comp-op expr /
                     expr comp-op expr
expr ::= ident-relation-det / fonction-sur-relation (expr)
         ident-relation-det CURRENT
comp-op ::= PREDECESSOR / comparaison
```

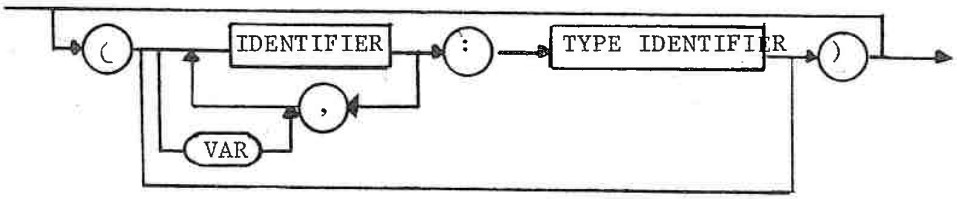
SPECIFICATION PROCEDURALE

BLOCK

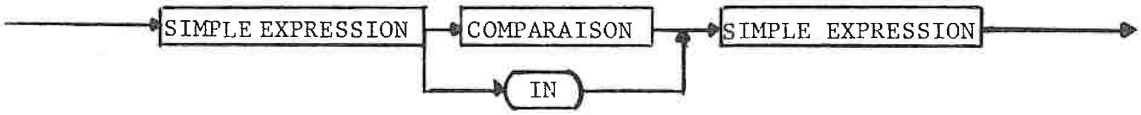



INSIGNED INTEGER
:
VARIABLE
:=
EXPRESSION
FONCTION IDENTIFIER
VARIABLE
:<
MODEL PREDICATE
PROCEDURE IDENTIFIER
(
NOM-RELATION
)
EXPRESSION
PROCEDURE IDENTIFIER
,
;
BEGIN
INSTRUCTION
END
IF
EXPRESSION
THEN
INSTRUCTION
ELSE
INSTRUCTION
CASE
EXPRESSION
OF
CONSTANT
:
INSTRUCTION
END
,
;
WHILE
EXPRESSION
DO
INSTRUCTION
REPEAT
INSTRUCTION
UNTIL
EXPRESSION
;
FOR
VARIABLE IDENTIFIER
EXPRESSION
TO
DOWNTO
EXPRESSION
DO
INSTRUCTION
WITH
VARIABLE
DO
INSTRUCTION
,
GOTO
UNSIGNED INTEGER
FOREACH
VARIABLE
IN
RELATION NAME
VARIABLE
DO
INSTRUCTION
UNTIL
EXPRESSION

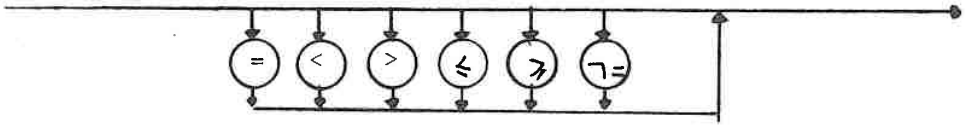
PARAMETER LIST



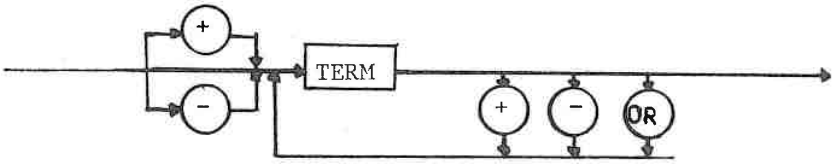
EXPRESSION



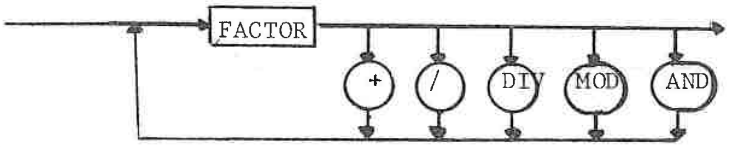
COMPARAISON



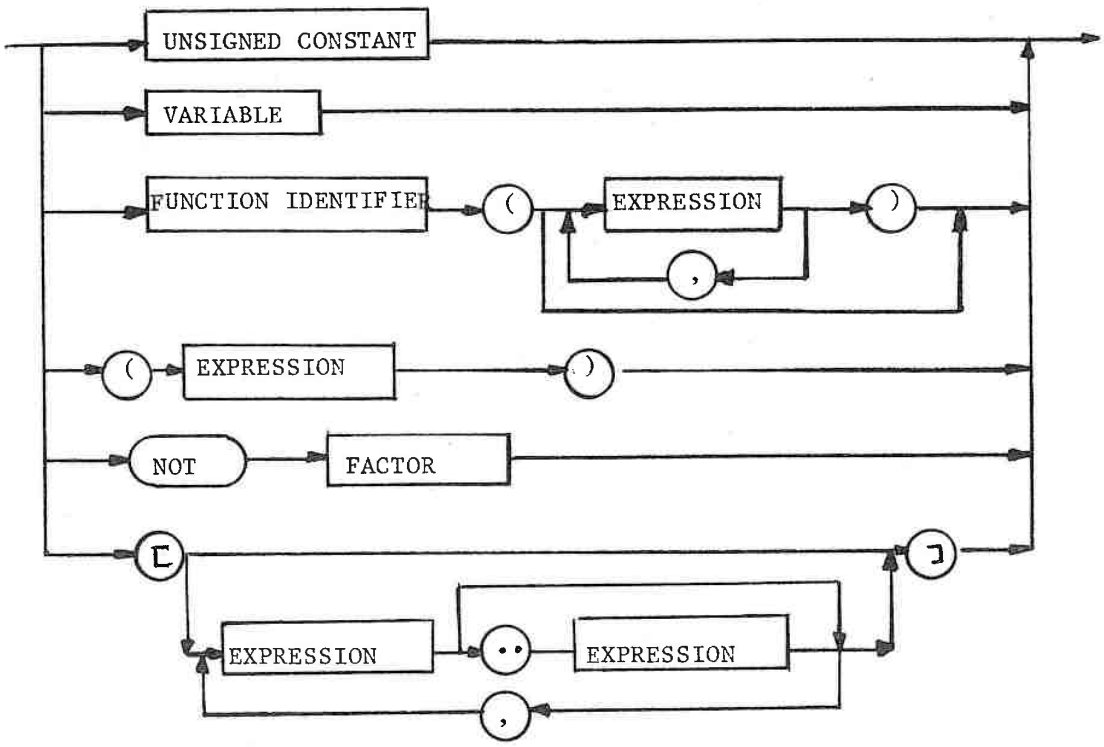
SIMPLE EXPRESSION



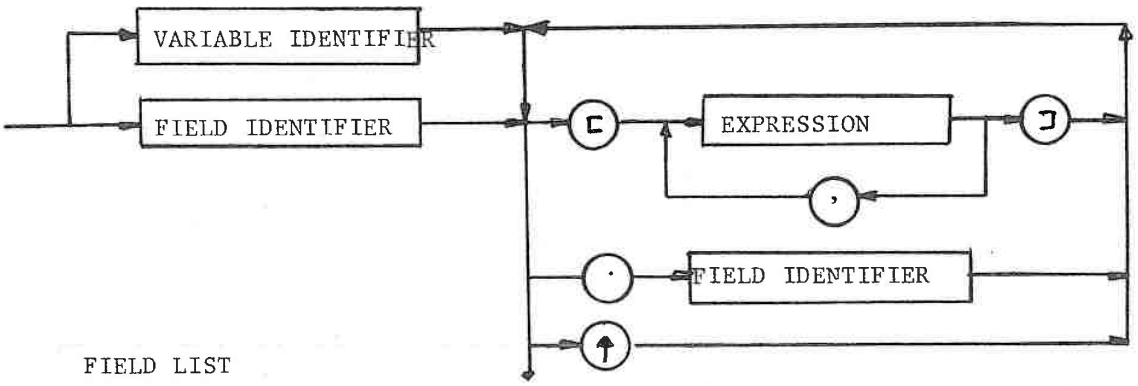
TERM



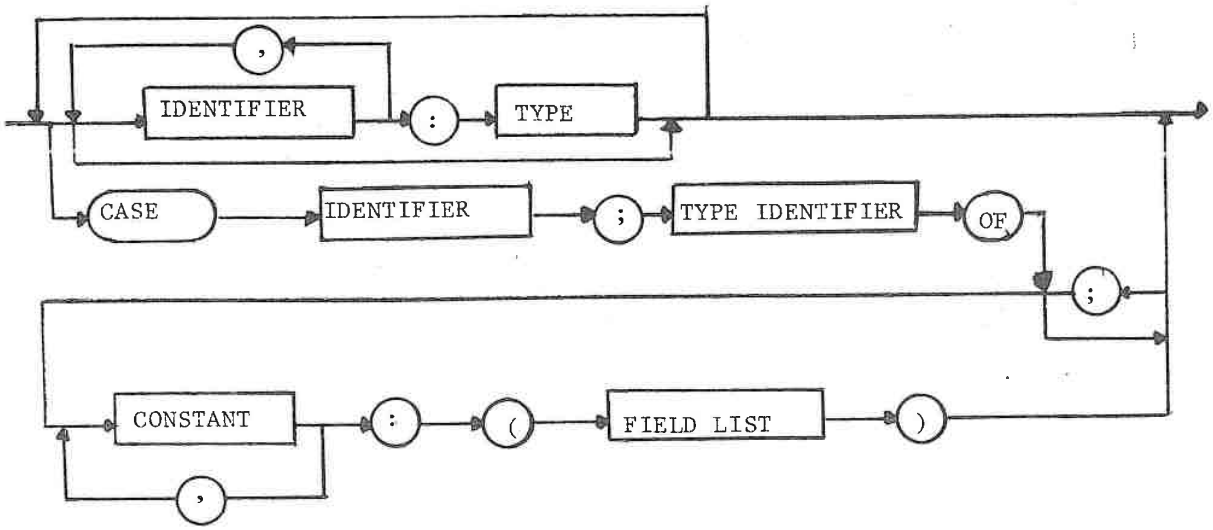
FACTOR



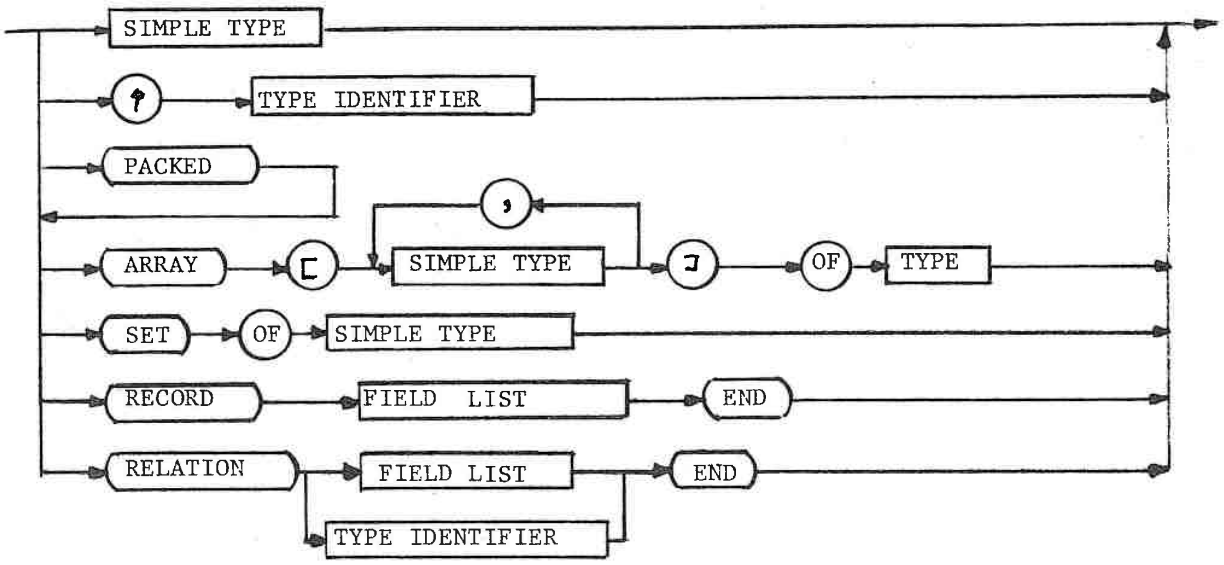
VARIABLE



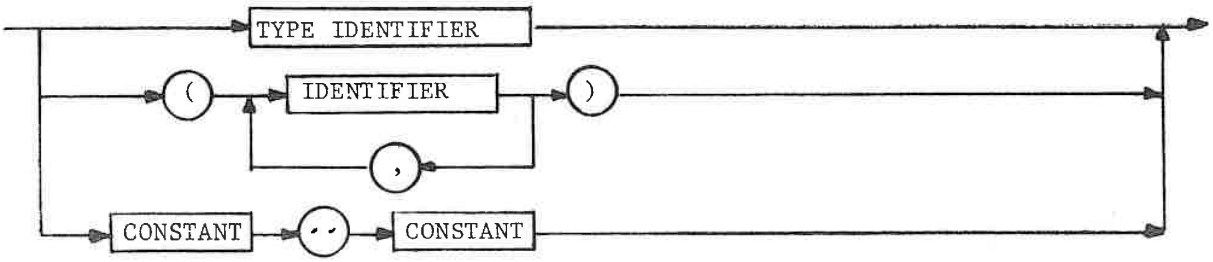
FIELD LIST



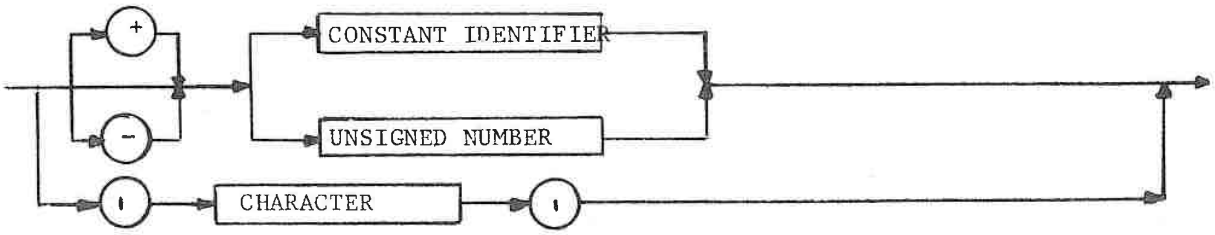
TYPE



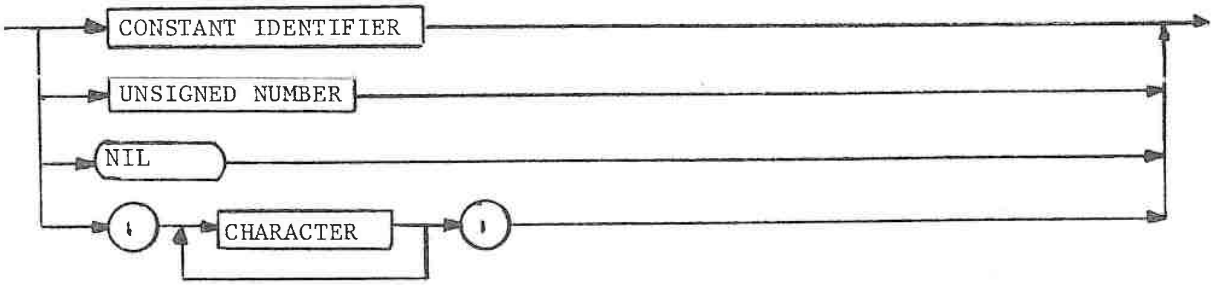
SIMPLE TYPE



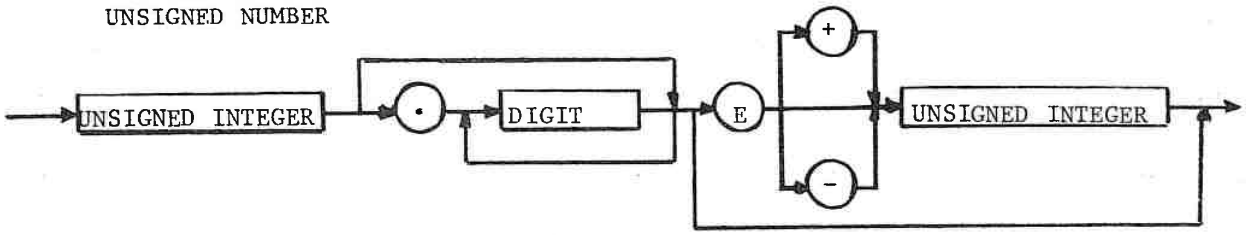
CONSTANT



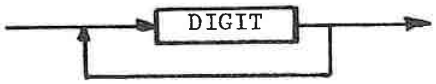
UNSIGNED CONSTANT



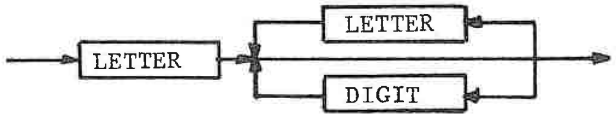
UNSIGNED NUMBER



UNSIGNED INTEGER



IDENTIFIER



NOM DE L'ETUDIANT : *LEIFERT Sergio*

NATURE DE LA THESE : *DOCTORAT INGENIEUR Informatique*

VU, APPROUVE

et PERMIS D'IMPRIMER

NANCY LE [illegible] 1980

LE PRESIDENT DE L'UNIVERSITE DE NANCY I

UNIVERSITE DE NANCY I
LE PRESIDENT

M. BOULANGE