l'intégrité et la mise à jour dans un système de gestion de bases de données réparties : projet POLYPHEME

Juan Manuel Andrade

THESE

présentée à

l' Institut National Polytechnique de Grenoble

pour obtenir le grade de
DOCTEUR INGENIEUR

par

Juan Manuel ANDRADE

L'INTEGRITE ET LA MISE A JOUR DANS UN SYSTEME DE GESTION DE BASES DE DONNEES REPARTIES.

PROJET POLYPHEME

Thèse soutenue le 29 octobre 1980 devant la commission d'examen

Président : C. DELOBEL

Examinateurs : Mme A. RECOQUE
MM. M. ADIBA
P. DECITRE
S. KRAKOWIAK
J. LE BIHAN

Je tiens à remercier

Monsieur Claude DELOBEL, Professeur à l'Université de Grenoble, de m'avoir fait l'honneur de présider le jury de cette thèse. Je veux aussi lui exprimer ma plus profonde gratitude pour l'accueil dans son équipe, la confiance qu'il m'a toujours témoignée et ses nombreux conseils et encouragements qui m'ont permis de mener à bien ce travail.

Madame Alice RECOQUE, Ingénieur de recherche à la Compagnie CII-Honeywell Bull, et Messieurs Jean LE BIHAN, Directeur du projet SIRIUS à l'IRIA et Sacha KRAKOWIAK Professeur à l'Université de Grenoble, pour leurs critiques et conseils concernant le manuscrit de ce travail et pour leur participation au jury.

Messieurs Michel ADIBA, animateur et responsable du projet POLYPHEME et Paul DECITRE, Ingénieur de recherche à la Compagnie CII-Honeywell Bull, qui m'ont beaucoup aidé de leurs conseils dans le développement de mon travail. Toute ma gratitude pour les fastidieuses lectures et critiques de mon manuscrit.

Je suis également reconnaisant

à Monsieur Christian Euzet qui a lu et relu (ouff!!) mon manuscrit pour permettre d'améliorer sa rédaction. Bien entendu il est très difficile de tout corriger et le lecteur voudra bien excuser les erreurs qu'il pourra trouver;

à tous les membres de l'équipe POLYPHEME, en particulier: Edouard ANDRE et Gilles BOGO avec lesquels nous avons eu des discussions très intéressantes et encourageantes; Andrée STIERS et NGUYEN GIA TOAN pour l'accueil qu'ils m'ont témoigné;

à un certain TTX 80 qui m'a permis d'éditer et de corriger facilement le français de mon manuscrit;

au Service de Reprographie qui à terminé la réalisation matérielle de ce travail.

"La dernière démarche de la raison est de reconnaître qu'il y a une infinité de choses qui la surpassent."

Pensées

Blaise Pascal

Résumé:

L'objet de cette thèse est double; premièrement, il s'agit de faire le point sur l'ensemble des travaux qui ont été menés dans le cadre du projet POLYPHEME, en essayant d'en dégager les principales conclusions. Nous présentons l'architecture et les caractéristiques de la maquette qui a été réalisée. Deuxièmement, nous étudions les problèmes d'intégrité qui se posent lorsque l'on désire faire coopérer des données dans un environnement réparti; nous abordons en particulier l'intégrité sémantique et le traitement des opérations de mise à jour. Un formalisme est proposé pour exprimer le comportement des mises à jour sur une relation abstraite construite sur d'autres relations; cette analyse considère les différents critères de fragmentation d'une relation abstraite.

CHAPITRE 1

"...Quelle douleur t'accable, Polyphème, et pourquoi dans la nuit immortelle as-tu poussé ces cris, qui nous ont réveillés? Est-ce qu'un mortel entraîne malgré toi tes troupeaux, ou cherche-t-on à te tuer par ruse ou violence?..."

L'Odyssée, Chant IX

Homère

Chapitre 1

INTRODUCTION

1.1 Rappels sur POLYPHEME 1.1

1.1.1 L'homogénéisation 1.4

1.1.2 La vue de la coopération 1.7

1.1.3 Traitement des requêtes 1.8

1.1.4 Exécution des requêtes 1.10

1.2 POLYPHEME et la configuration ANSI/SPARC 1.13

1.3 L'intégrité dans un SGBDR 1.17

1.4 Conclusions 1.21

CHAPITRE 1 INTRODUCTION

Dans ce chapitre, nous introduisons les principaux apports du projet POLYPHEME qui a servi de cadre à ce travail.

1.1 Rappels sur POLYPHEME:

POLYPHEME est le nom d'un projet d'études sur la coopération de bases de données réparties sur un réseau d'ordinateurs. Ce projet a été mené conjointement par l'IMAG et le Centre Scientifique CII-Honeywell Bull. Il a reçu un apport financier du projet pilote SIRIUS de l'IRIA et à ce titre une maquette a été développée et expérimentée depuis Juin 1979 sur le réseau CYCLADES.

POLYPHEME a été un travail d'équipe qui a fait l'objet de plusieurs publications scientifiques, rapports et thèses. Notre contribution à POLYPHEME s'est située sur le plan du développement des logiciels mais aussi a consisté à étudier les problèmes d'intégrité des données dans un environnement réparti.

En ce qui concerne la maquette nous avons réalisé principalement tout le logiciel qui permet de contrôler et synchroniser l'exécution répartie des requêtes. Nous avons également accordé une large part à la mise en forme du logiciel pour faire un prototype vraiment opérationnel dans l'environnement du réseau CYCLADES.

En ce qui concerne l'aspect plus théorique de notre travail, nous avons étudié les problèmes d'intégrité qui se posent lorsque l'on désire faire coopérer des données dans un environnement réparti.

L'objet de notre travail est double. Premièrement, il s'agit de faire le point sur l'ensemble des travaux qui ont été menés dans le cadre du projet en essayant d'en dégager les principales conclusions. Deuxièmement, nous voulons compléter certains aspects qui jusqu'à présent ont reçu une attention moindre compte tenu de l'ampleur de la tâche à réaliser. Il s'agit en particulier des aspects concernant l'intégrité et le traitement des opérations de mise à jour dans un environnement réparti pour lesquelles nous ferons un certain nombre de propositions.

Dans POLYPHEME nous avons défini un ensemble d'hypothèses et suppositions qui fixent le cadre de la coopération des Bases de Données (BD): (ADI78,POL79)

- on dispose de plusieurs Systèmes de Gestion de Bases de Données (SGBD) hétérogènes connectés à un réseau d'ordinateurs,
- une BD par SGBD participe à la coopération,
- les modèles des BD peuvent être hétérogènes,
- les BD sont déjà en exploitation,
- on ne veut pas restructurer les BD,
- il n'est pas acceptable de bouleverser les accès lo-

caux à un SGBD lors de son introduction sur le réseau.

Ces hypothèses nous ont confrontés aux problèmes suivants:

a) homogénéisation des SGBD hétérogènes pour permettre un traitement standard des informations (ADI78, ACE78),

b) définition et description des données de la coopération (ADI78,ADA79),

c) analyse, décomposition et optimisation des requêtes sur ces données (CAL78,NGU79,FER79,PAD79,ADA79),

d) définition d'un mécanisme d'exécution et de synchronisation dans un environnement réparti (ADE78,DEA80, EUZ79,ADA79).

Nous allons développer chacun de ces problèmes en donnant les solutions que nous avons retenues. Il faut noter l'approche ascendante que nous avons prise pour la conception de l'architecture du SGBDR et pour la conception de la Base de Données Répartie (BDR), par rapport à l'approche descendante utilisée par ailleurs (ROT80,STO76).

1.1.1 L'homogénéisation:

Nous avons utilisé le modèle relationnel (COD70) et l'algèbre relationnelle pour homogénéiser les SGBD. La méthode consiste à assimiler les SGBD à des machines relationnelles logiques qui peuvent être considérées comme homogènes (ACE78) . Chaque machine se compose essentiellement de trois éléments:

a) un espace de noms qui correspond à la description logique des relations et des attributs.
b) un espace d'objets formé par les occurrences des relations,
c) une correspondance entre les deux espaces précédents.

La machine fournit en outre à l'usager des langages de définition et de manipulation des données.

Cette transformation a été réalisée tout en respectant la configuration du SGBD. L'espace de noms de la machine relationnelle logique (appelée Machine Locale ou ML) construite sur le SGBD, correspond au niveau externe du SGBD, selon la configuration ANSI/SPARC (ANS75). Ce niveau concerne seulement le point de vue d'un usager particulier. Il peut supporter plusieurs vues individuelles où chaque usager voit la même construction sémantique représentée sous des formes différentes. Un schéma externe cor-

respond alors à l'information que l'administrateur du SGBDR a décidé de considérer pour la coopération dans ce SGBD particulier. Dans la terminologie de POLYPHEME, nous appellons ce schéma externe Vue Locale. Compte tenu de nos hypothèses, on aura alors, une vue locale par SGBD coopérant qui décrit les données faisant partie de cette coopération. Le modèle d'une vue locale est le modèle relationnel, ce qui suppose la transformation des schémas réseaux et hiérarchiques en un schéma relationnel. (LEO76,ADL76, ADI78)

La correspondance entre l'espace des noms et l'espace d'objets est réalisée par la description des opérations (obtenir, insérer, supprimer, et modifier) disponibles sur les relations. Nous avons retenu deux solutions pour mettre en oeuvre cette correspondance:

a) Disposer de programmes réalisant les opérations primitives des langages relationnels, auquel cas chaque programme est directement interprétable par le SGBD correspondant. (CAS77,ADI78)

b) Avoir un système permettant un interface relationnel avec le SGBD (comme URANUS (NGU77) avec SOCRATE). C'est cette solution qui se trouve intégrée dans la maquette. On dispose ainsi de plusieurs opérateurs relationnels et on peut faire travailler URANUS comme un système relationnel indépendant de SOCRATE.

Pour réaliser la coopération des divers SGBD au travers des machines locales, l'administrateur considère l'ensemble des vues locales et les intègre en une vue unique: la vue de la coopération (voir paragraphe 1.3). Cette vue doit considérer tous les liens sémantiques et les incohérences possibles entre les diverses les vues locales.

Mais comment implanter cette vue de la coopération? La solution que nous avons prise consiste à utiliser de nouveau le concept de machine relationnelle logique. Nous allons créer une autre machine -appelée Machine Globale ou (MG)- qui contiendra dans son espace de noms la description de la vue de la coopération. L'espace d'objets se trouve réparti sur les différentes machines locales et la correspondance entre cet espace et l'espace de noms prend en compte cette notion de répartition.

L'usager du SGBDR adresse ses requêtes à la machine globale. Nous n'avons pas fait d'hypothèses sur la localisation des différentes machines qui ne sont que logiques. En fait, certaines machines (ou toutes) peuvent résider sur le même ordinateur.

La description de cette architecture qui est intégrée dans la maquette sera discutée au chapitre 4. Le lecteur pourra remarquer la nécessité d'un mécanisme d'exécution et de synchronisation dans un environnement réparti, où le SGBDR est considéré comme un ensemble de SGBDs indépen-

dants coopérant entre eux à travers le réseau d'ordinateurs. La base de données répartie peut être ainsi considérée comme un ensemble de bases. (LIT79)

1.1.2 La vue de la coopération:

Dans le paragraphe précédent, nous avons défini la vue de la coopération comme une vue° unique formée à partir des vues locales. Cette vue doit considérer tous les liens sémantiques et incohérences possibles entre les vues locales; ceci revient à dire que la vue de la coopération correspond au schéma conceptuel du SGBDR, sur lequel plusieurs schémas externes- chacun correspondant à une application différente- peuvent être définis. Ces schémas externes apparaisent dans la terminologie de POLYPHEME comme vues locales.

Nous avons dû faire quelques restrictions sur la notion d'usager pour simplifier la mise en oeuvre de la maquette, considérant que la Machine Globale est mono-usager. En conséquence on pourrait penser que la vue globale et la vue de la coopération (schéma conceptuel) se confondent, mais ceci n'est pas vrai si le système admet la possibi-

---

°NOTA: Nous utiliserons le mot vue à la place du mot schéma. Dans notre contexte, ils sont équivalents.

lité de plusieurs machines globales travaillant sur des vues globales différentes mais construites sur les mêmes vues locales. Dans ce cas les vues globales doivent être contruites à partir du schéma conceptuel pour éviter les incohérences dûes au partage des données. Initialement dans POLYPHEME nous avons limité le système à une seule machine globale, ce qui fait que nous ne considérions qu'une seule vue globale.

Pour construire la vue de la coopération, il a fallu développer des outils de modélisation afin de déterminer les liens sémantiques et de gérer les incohérences pouvant provenir de cette coopération. MOGADOR (ADI77,ADI78), fondé sur le modèle "Data Semantics" (ABR74), est l'outil de modélisation proposé. Il est resté un modèle conceptuel et dans la maquette, la vue globale et les vues locales ont été considérées comme un ensemble de descriptions de relations n-aires .

Le lecteur trouvera dans le Chapitre 2 une description plus approfondie des problèmes concernant la conception d'une vue globale.

1.1.3 Traitement des requêtes:

Nous avons défini un ensemble d'opérations (obtenir, insérer, modifier, supprimer) sur les n-uplets des relations de chaque machine. Ces opérations s'expriment par

des requêtes écrites dans un langage qui utilise les concepts de l'algèbre relationnelle. Rappelons qu'un usager du SGBDR s'adresse à une machine globale qui est la seule qu'il voit. La correspondance global-local est assurée automatiquement par POLYPHEME. L'usager travaille en fait comme sur un SGBD centralisé.

C'est cette analyse des requêtes de la machine globale qui doit prendre en compte la répartition de l'espace d'objets sur plusieurs machines locales, et avoir une stratégie d'accès optimal aux données (CAL78,NGU79,PAR78, FER79,PAD79).

Pour chaque requête soumise par l'usager, la machine globale effectue une analyse en plusieurs étapes:

a) Une analyse syntaxique avec la création d'un arbre binaire.

b) Prise en compte de la fragmentation (cf 2.1.1) des relations de la vue globale. Chaque relation est remplacée par sa fonction de définition (cf. 2.1.2, 2.1.4). Le critère de fragmentation peut être rajouté pour des raisons d'optimisation (cf Chapitre 3).

c) Une phase d'optimisation sur les opérateurs algébriques qui consiste à remonter, descendre ou réduire les opérateurs de l'arbre (CHS75,CAL78).

d) Prise en compte de la localisation des relations pour découper l'arbre en sous-arbres; cette étape doit tenir compte de l'optimisation de l'accés aux machines locales et de la minimisation du transfert d'informations entre les machines.

Il faudra faire dans cette dernière étape une distinction entre l'optimisation statique (CAL78) qui décide lors de la compilation de la requête d'une stratégie fixe d'exécution et localisation des traitements, par rapport à l'optimisation dynamique (NGU79) qui permet de localiser dynamiquement les traitements pendant l'exécution de la requête.

Les opérations de modification (insérer,modifier,supprimer) (ADA79) exigent un traitement différent des opérations d'obtention (CAL78, NGU79) ; c'est ce traitement que nous allons détailler dans le chapitre 3.

1.1.4 Exécution des requêtes:

Dans le paragraphe précédent nous signalons l'importance de la stratégie d'accés optimal aux données: parallélisme des traitements et minimisations des transferts entre les machines pendant l'exécution de la requête. La synchronisation de ces traitements parallèles est mise en oeuvre à deux niveaux:

1) Le premier consiste à construire un module développé sur les niveaux de communication, transport et contrôle des messages du réseau de communication (ADE78,DEC79). Ce module, appelé le moniteur d'exécution répartie (MER), assure le séquencement et le contrôle de flux des messages à travers le réseau. Il permet, en utilisant PL/1, la mise en oeuvre d'applications réparties (p.ex: un SGBDR). MER donne au programmeur d'applications réparties des services de base d'activation et synchronisation de procédures à distance. C'est alors MER qui assure la communication entre les différentes machines au travers du réseau. La synchronisation à ce niveau consiste à:

a) activer des procédures (PL/1) à distance; c'est à dire:

- transformer ces activations en messages (un protocole spécial a été défini),
- envoyer les messages vers les sites correspondants,
- recevoir des messages,
- transformer les messages en activations de procédures.

b) contrôler l'asynchronisme de l'envoi et de la réception des messages. MER assure le séquencement et le contrôle de flux des messages au travers du réseau.

2) Le deuxième niveau de synchronisation dépend directement du concepteur du SGBDR, qui programme son application en utilisant les services et facilités de synchronisation données par MER. Le concepteur se trouve confronté à des décisions sur la synchronisation du flot de données entre les différents opérateurs relationnels qui composent la requête:

- exécution des opérateurs n-uplet par n-uplet (mode sériel ou "pipeline"),
- retarder l'exécution d'un opérateur jusqu'à la fin de la relation intermédiaire, (mode total),
- combiner les deux modes précédents, (mode mixte).

Une des solutions à ce problème utilise les réseaux de Petri et est présenté dans la thèse de C.Euzet (EUZ79). Dans la maquette nous avons pris la décision d'utiliser le mode mixte: une exécution sérielle ("pipe-line") des sous-arbres locaux dans une machine locale, pour former une relation temporaire; cette relation est transférée à la machine globale qui attend l'arrivée de toutes les relations provenant des sous-arbres locaux pour continuer l'exécution de la requête (mode total). Le problème de la synchronisation des mises à jour (ADA79) sera considéré dans le chapitre 3.

Le concepteur du SGBDR doit aussi faire face au traitement d'erreurs: erreurs dans l'exécution d'un sous-arbre

local, panne d'une machine locale, etc. Les services de MER permettent de contrôler à ce niveau les problèmes causés par la déconnexion d'une machine.

1.2 POLYPHEME et la configuration ANSI/SPARC:

Dans ce paragraphe nous voulons montrer comment la notion de la BDR définie par une vue globale, s'intégre avec de légères modifications à l'architecture proposée dans le rapport ANSI/SPARC (ANS75). Rappelons les trois niveaux de description d'un SGBD:

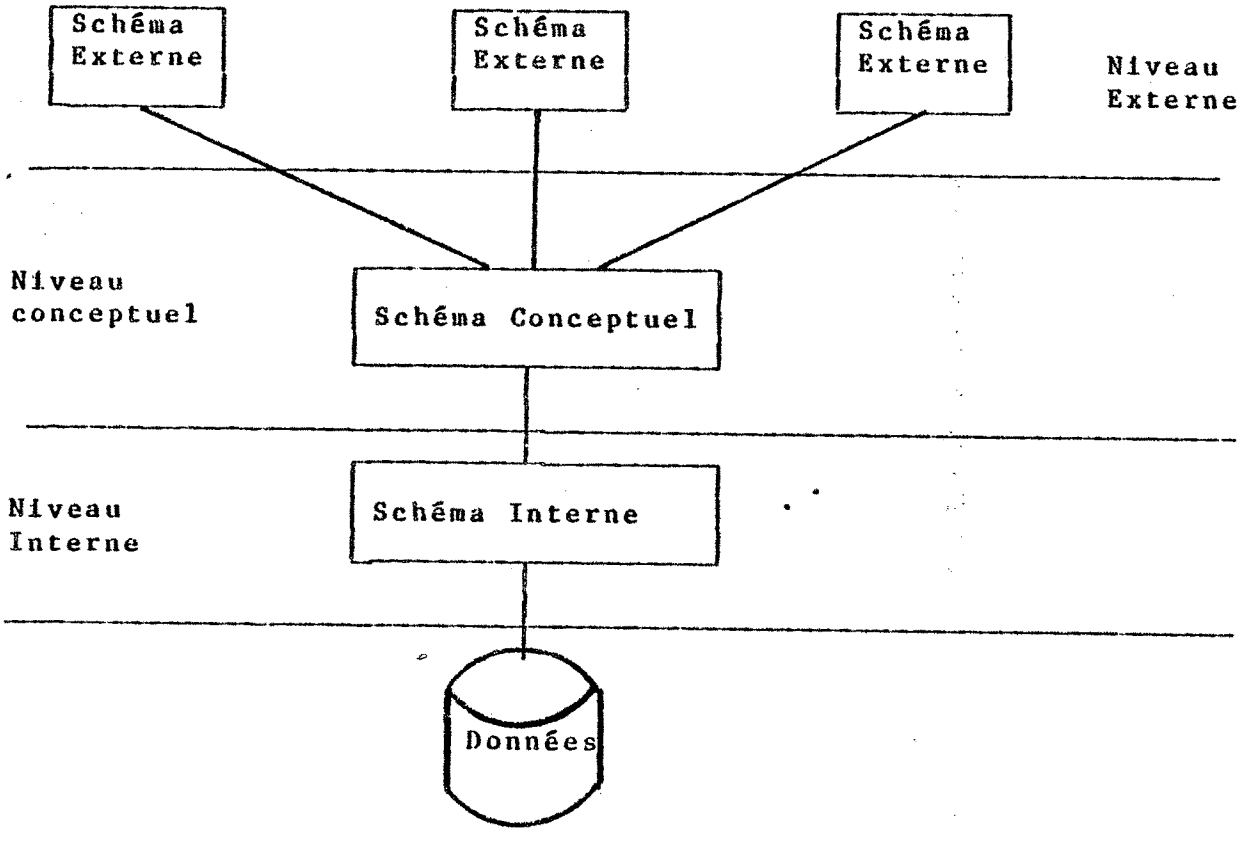


Figure 1.1 Niveaux d'un SGBD (ANSI/SPARC)

a) Le niveau interne ou physique qui concerne la structure des fichiers qui constituent la base. C'est la représentation physique des données, considérée comme un schéma: le schéma interne.

b) Le niveau conceptuel qui décrit l'ensemble des informations qui constituent la base de données. Le schéma conceptuel résume les abstractions réalisées sur les données et il représente le point de vue de la communauté où tous les concepts des différents usagers sont intégrés. C'est l'administrateur de la base qui se charge de sa description.

c) Le niveau externe ou de l'usager qui concerne seulement le point de vue d'un usager en particulier. Le schéma externe correspond généralement à une partie de la base de données et il s'adresse à une application particulière. Ce niveau peut supporter plusieurs vues individuelles (schémas externes), où chaque usager voit la même construction sémantique représentée sous différentes formes.

Dans les paragraphes 1.1.1 et 1.1.2 nous avons remarqué que POLYPHEME peut être considéré comme un ensemble de SGBDs indépendants, et la base de données de la coopération peut être à son tour considérée comme un ensemble de bases. Ceci est provoqué par l'approche ascendante que nous avons prise.

La figure 1.2 montre les différents niveaux de l'architecture du SGBDR POLYPHEME. Nous y retrouvons la configuration ANSI/SPARC à deux niveaux: le niveau local qui correspond aux descriptions des différentes bases de données coopérantes (Bases Locales) et le niveau global qui correspond aux informations concernant la coopération.

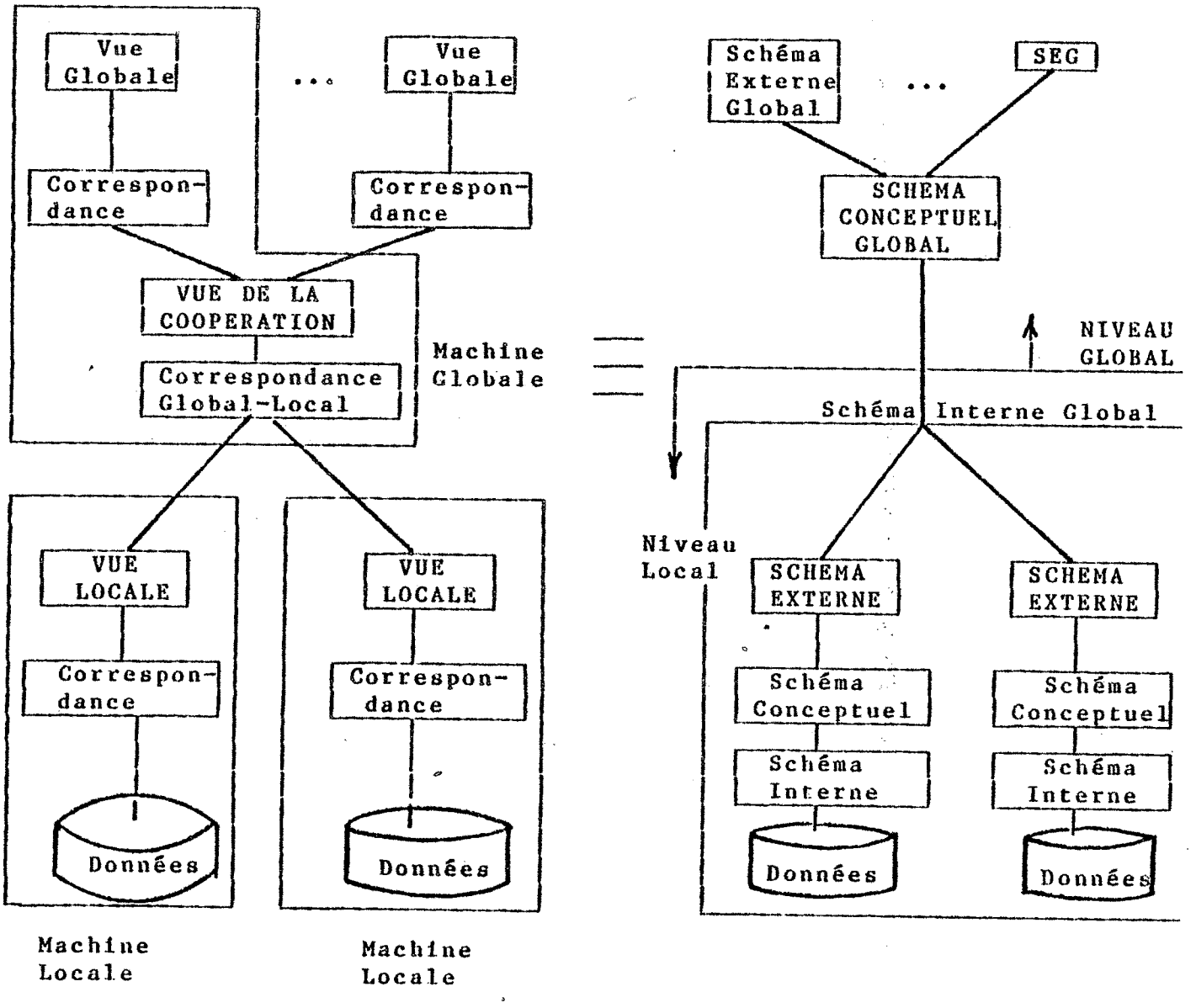


Figure 1.2 Architecture de POLYPHEME

Nous avons fait plusieurs restrictions à ce modèle. L'une d'elles se situe au niveau des schémas externes locaux (ou vues locales): nous avons considéré seulement un schéma externe par SGBD coopérant, ce qui permet de construire facilement un schéma conceptuel de coopération (schéma conceptuel global ou vue de coopération). S'il y a plusieurs schémas externes par SGBD, il faudra alors former le schéma conceptuel de toutes les coopérations possibles.

Dans la maquette de POLYPHEME, nous n'avons pas mis en oeuvre de mécanisme pour le traitement des relations définies à partir d'autres. Ceci implique que la vue de coopération- qui doit être un ensemble de descriptions de relations basées sur les descriptions des vues locales- est réduite à l'ensemble de toutes les relations qui figurent dans les vues locales. Sur cette vue de coopération- qui reste pour nous au niveau conceptuel- l'administrateur peut former une vue globale en utilisant seulement un sous-ensemble de relations. Il peut également imaginer des relations "virtuelles ou abstraites" (p.ex: en utilisant des projections, restrictions, unions, compositions, etc) mais comme la maquette n'a pas de mécanisme de gestion automatique de ce type de relations, l'usager doit gérer lui même la correspondance; pour cela, il a à sa disposition des requêtes cataloguées. Les opérations de modification se transforment dans cet environnement en des opérations ponctuelles, sur une seule machine et une seule relation.

1.3 L'intégrité dans un SGBDR:

Rappelons tout d'abord quelques concepts importants: (POL79)

a) Sécurité: signifie pour nous, la protection des données contre la destruction, la modification et l'accès non autorisés (soit accidentellement, soit intentionnellement). Cette protection est assurée par l'identification et l'autorisation d'un usager, l'intégrité et la confidentialité des données.

b) Identification et autorisation: correspondent à l'ensemble de techniques permettant de reconnaitre un individu et de vérifier que c'est vraiment lui qui veut utiliser le système.

c) Intégrité: c'est pour nous l'ensemble des techniques et des mécanismes maintenant l'existence et la qualité de l'information.

d) Confidentialité: c'est l'ensemble des techniques et des mécanismes garantissant l'information contre toute destruction ou accès non autorisés.

Dans (POL79) ont été étudiés d'une façon très sommaire les problèmes concernant l'intégrité dans un environnement réparti. Dans notre travail nous voulons compléter cet aspect.

Définition 1: une donnée est valide quand elle satisfait toutes les règles de description du modèle.

Définition 2: une donnée est intègre si elle est restée valide pendant un fonctionnement dégradé du système.

Définition 3: une donnée est cohérente si elle est restée intègre à l'interaction simultanée (concurrente) de deux ou plusieurs processus.

Ces définitions nous permettent de classer les mécanismes pour maintenir l'intégrité en trois catégories: (ESC75,HAM75)

a) L'intégrité sémantique ou (interne) qui considère la maintenance de la qualité des données par la vérification des contraintes sémantiques (dépendances fonctionnelles et autres, cf 2.2) définies implicitement ou explicitement sur le schéma de description de la base;

c) L'intégrité externe qui prend en compte tous les aspects relatifs à la synchronisation des interactions de processus concurrents. Ces interactions peuvent occasionner des erreurs sémantiques qui introduisent des incohérences sur les données partagées.

c) La fiabilité qui se préoccupe de maintenir l'existence des données et de les rendre dans un état intègre en cas de fonctionnement anormal (pannes) du système.

Dans un SGBDR on retrouve ces trois classes de mécanismes mais avec des problèmes supplémentaires dûs à la répartition et aux conditions anormales suivantes: (une machine = une machine logique = un ensemble de programmes)

- panne de la ligne de communication: implique que les deux machines liées par la ligne continuent à travailler de façon indépendante,
- panne d'un ordinateur: implique l'abandon des machines travaillant sur cet ordinateur,
- "plantage" d'une machine: implique l'abandon de la machine,
- abandon volontaire d'une machine quand le système est en opération,
- réinsertion d'une machine dans le système,
- formation de sous-réseaux travaillant indépendamment.

Ces conditions anormales introduisent des indécisions dans le comportement d'une machine qui est obligée de s'adresser aux autres pour lever ces indécisions. Si l'indécision n'est pas levée, la machine est obligée de bloquer toutes les informations concernées (cela conduit parfois au blocage de la machine elle même) jusqu'à la reprise après panne. Ces types de conditions anormales montrent clairement que des mécanismes d'intégrité externe et de fiabilité sont importants et prioritaires à développer dans un SGBDR. Les solutions à ces problèmes sont com-

plexes et de nombreuses études ont été réalisées dont citerons ici plus particulièrement: (ROT80,ROS78,STO78,LIN79 THO77,WIL79,WIL80,...).

Dans ce travail nous ne considérerons pas les techniques et mécanismes qui garantissent l'intégrité externe ou la fiabilité. Nous suggèrons au lecteur de se reporter aux références citées ci-dessus. Notre travail portera plutôt sur l'intégrité sémantique, les problèmes concernant la conception de la vue globale (v. Chapitre 2) et le traitement des opérations de mise à jour sur une vue globale (Chapitre 3).

Dans la maquette nous n'avons pas considéré la mise en oeuvre de ces techniques; cependant deux mécanismes simples ont été prévus: annuler une requête en cas d'erreur et effectuer une copie de la BDR au démarrage du système (cf Chapitre 4).

L'optimisation du traitement réparti d'une requête suggère l'introduction de copies pour accélérer l'accés aux données. Ceci entraîne deux nouveaux problèmes: la cohérence des copies et un problème d'optimisation sur leur allocation qui dépend de la topologie du réseau (POL79, PAD79).

La confidentialité a été l'objet d'études ponctuelles à côté du projet POLYPHEME dont nous citerons (RIC78,EUZ80).

1.4 Conclusions

Ce chapitre nous a permis de rappeler l'environnement dans lequel se situe notre travail et de décrire en même temps les aspects plus les importants de notre recherche:

-intégrité sémantique (Chapitre 2),

-traitement des mises à jour (Chapitre 3),

-description de la maquette (Chapitre 4),

-expériences acquises de POLYPHEME (Chapitre 5).

CHAPITRE 2

"Prudence est mère de sûreté"

(Proverbe populaire)

"L'intégrité de l'organisme est indispensable aux manifestations de la conscience"

A.Carrel

CHAPITRE 2

L'INTEGRITE D'UNE BASE DE DONNEES REPARTIE

2.1 Les problèmes de la conception d'une vue globale 2.1

2.1.1 Définitions 2.2

2.1.2 Le problème de la correspondance entre schémas 2.5

2.1.3 La transformation des opérations de mise à jour 2.8

2.1.3.1 Changement admissible (Condition 1) 2.8

2.1.3.2 Mise à jour sémantiquement intègre (Condition 2) 2.10

2.1.3.3 Mise à jour sans interférences (Condition 3) 2.10

2.1.4 Généralisation de la définition d'un schéma 2.13

2.1.5 La correspondance de schémas d'un SGBDR 2.15

2.1.5.1 L'approche descendante 2.15

2.1.5.2 L'approche ascendante 2.21

2.1.6 MOGADOR 2.27

2.1.7 L'usager et la correspondance 2.32

2.2 L'intégrité sémantique 2.38

2.2.1 Types de contraintes 2.41

2.2.2 Prise en compte des contraintes 2.44

2.2.3 Validation à la compilation 2.46

2.2.4 Validation pendant l'exécution 2.48

2.2.5 Validation aprés l'exécution 2.49

2.2.6 Validation des contraintes dans un SGBDR 2.49

2.3 L'importance de la notion de transaction 2.53

2.4 Conclusions 2.55

2.1 Les problèmes de la conception d'une vue globale:

L'architecture multi-niveaux de POLYPHEME (cf§1.2) est bâtie sur la notion de "vue relationnelle" (COD74,CHA75, STO75). Une vue relationnelle est une relation virtuelle (ou abstraite) qui n'existe pas physiquement dans la base de données mais qui peut être définie en termes d'autres relations (les relations de base). Les relations de base sont les seules relations dont une occurrence est stockée dans la base de données.

L'algèbre relationnelle permet d'appliquer une approche récurrente dans la définition des relations: le résultat d'un opérateur algèbrique ou d'une expression formée par l'imbrication d'opérateurs algèbriques est, toujours, une relation qui à son tour pourra être utilisée pour définir d'autres relations. Cette récurrence est particulièrement importante dans une approche coopérative ascendante de conception de bases de données réparties. Elle pose, cependant, des problèmes qui conduisent à des restrictions (parfois très sévères) dans la définition et l'utilisation des vues.

Pour nous, le terme "vue" est plus général: c'est un ensemble de relations abstraites.

2.1.1 Définitions:

Nous allons donner une série de définitions fixant la terminologie que nous utiliserons.

1.Schème d'une relation: il correspond à la description de la relation, c'est à dire d'une part son nom, et d'autre part les noms, types et portées des attributs. Un ensemble de contraintes d'intégrité décrit l'ensemble d'états valides de la relation.

2.Schéma relationnel: c'est l'ensemble des schèmes des relations. Il considère en outre l'ensemble des contraintes d'intégrité inter-relationnelles.

3.Vue: elle considérée comme la partie visible d'un schéma relationnel. Cette partie correspond à une espèce de "fenêtre" qui montre les données accessibles par l'usager de la vue. Une vue peut être un sous-ensemble d'un schéma relationnel.

4.Relation de Base: c'est une relation décrite dans le schéma conceptuel (cf. définition 13). Une relation de base peut être découpée en fichiers ou relations dites relations internes.

5.Relation interne: elle est décrite dans le schéma interne et caractérise la structure du fichier qui contient l'occurrence physique de la relation.

6.Relation Abstraite: c'est une relation définie soit sur une relation de base, soit sur une autre relation abstraite. Elle est caractérisée par l'expression qu'il faut appliquer pour créer son occurrence.

7.Relation fragmentée ou partitionnée: c'est une relation abstraite composée par une ou plusieurs relations ou sous-relations appelées fragments. Dans POLYPHEME, un fragment est toujours une relation.

8.Partionnement ou fragmentation: c'est l'action de séparer ou découper une relation abstraite en plusieurs fragments. (On ne tient pas compte de la répartition des fragments)

9.Répartition: elle s'applique à l'action de distribuer les fragments sur les différentes ressources du système. La répartition implique une localisation.

10.Localisation: c'est l'action de prendre en compte la situation des fragments.

11.Relation répartie: est une relation fragmentée dont les fragments se trouvent sur les différents sites du système.

Les trois schémas de ANSI/SPARC (ANS75) sont définis comme suit:

12.Schéma interne: il décrit l'accés physique aux occurrences des relations. Il est composé par l'ensemble des

descriptions de fichiers et/ou par un ensemble de schèmes de relations internes.

13.Schéma conceptuel: composé de l'ensemble des schèmes des relations de base et de l'ensemble des contraintes d'intégrité; il est considéré comme la vue de toute la base de données.

14.Schéma externe: c'est un sous-schéma du schéma conceptuel. Il est composé par l'ensemble de schèmes de relations abstraites définies sur les relations de base du schéma conceptuel. Un usager peut ne voir seulement qu'une partie du schéma: sa vue.

Dans le texte nous utiliserons le mot vue pour désigner une ou plusieurs relations abstraites. Une vue est alors définie par un ensemble de schèmes de relations abstraites et par la spécification de la correspondance des opérations sur ces relations en opérations sur les relations qui les composent. L'usager ne voit que la déclaration (le schème) des relations abstraites; la spécification de la correspondance est considérée comme une partie de la mise en oeuvre de la vue (cf.Chapitre 3).

Dans §1.2, nous avons défini deux niveaux dans l'architecture de POLYPHEME: le niveau global qui est celui des informations concernant la coopération et le niveau local qui correspond aux systèmes de bases de données coopé-

rants. C'est pour cela que dans la terminologie de POLYPHEME, nous trouvons les concepts de relation globale, relation locale, vue locale, et vue globale. Pour nous, ces concepts ont la signification suivante:

15.Relation Locale: c'est une relation abstraite ou une relation de base d'une base de données coopérante. Dans POLYPHEME, elle (ou une partie d'elle) est considérée comme un fragment d'une relation globale.

16.Relation Globale: c'est une relation abstraite répartie. Elle est définie sur une ou plusieurs relations locales.

La vue locale est alors la partie visible d'une base de données coopérante et elle est formée par un ensemble de schèmes de relations locales. La vue de la coopération est considérée comme un ensemble de schèmes de relations globales et une vue globale comme un sous-ensemble de la vue de la coopération où plusieurs relations globales peuvent être redéfinies donnant lieu à une nouvelle hiérarchie de relations abstraites (cf figs 1.2, 2.6, 2.7).

2.1.2 Le problème de la correspondance entre schémas:

L'architecture multi-schéma permet d'obtenir une indépendance de l'interface utilisateur vis à vis des changements possibles qui peuvent apparaître dans la structure

logique ou physique de la base de données. A n'importe quel niveau sur une relation, il faut pouvoir appliquer des opérations d'obtention et de modification des nuplets.

Les opérations sur une relation abstraite doivent être traduites par des opérations correspondantes sur les relations qui la composent. Cette transformation est toujours possible pour toute opération d'obtention de n-uplets mais elle présente des problèmes trés sérieux pour les opérations de modification.

Le problème de la spécification de la correspondance entre schémas est un problème ouvert dont les meilleures solutions proposées impliquent la connaissance de toute la sémantique des données. Plusieurs études ont jeté les fondations d'une théorie sur les mises à jour de vues relationnelles (DAB78a,DAY79,BAS78,PAP77,TOD77). Notre approche, définie au Chapitre 3, est basée sur l'analyse de la définition des différentes formes de fragmentation d'une relation abstraite (par des opérations algébriques) afin de trouver une correspondance adéquate pour chaque cas. Une approche semblable est définie dans (FUS77 et OSM79). D'autres solutions adoptent l'approche des types abstraits, où associée à la définition des relations, il y a une description des opérations possibles (d'obtention ou de modification de l'occurrence) ainsi que de la transformation de ces opérations en des opérations sur les rela-

tions de base (PPB78,SEF78,ROS79). Notre approche utilise aussi cette solution.

En §2.1.1, nous avons défini un schéma relationnel comme un ensemble de schémes de relations et un ensemble de contraintes d'intégrité; nous ajoutons à cette définition, un ensemble d'opérations sur les relations. Considérons la notation suivante:

SR l'ensemble des schèmes des relations

OP l'ensemble d'opérations que l'on peut réaliser: obtenir, insérer, supprimer ou modifier un nuplet d'une relation.

CI l'ensemble des contraintes d'intégrité.

Un schéma est alors représenté par un triplet: (SR,OP,CI).

Si Schéma-2(SR2,OP2,CI2) est un schéma où SR2 décrit une relation abstraite, elle même définie sur une ou plusieurs relations d'un autre schéma (le Schéma-1(SR1,OP1, CI1)) la correspondance entre les deux schémas est définie par:

a) La fonction de définition de SR2 sur SR1.
b) La transformation des opérations OP2 sur SR2 en des opérations OP1 sur SR1.
c) La répercussion des contraintes d'intégrité de SR2 sur SR1.

2.1.3 La transformation des opérations de mise à jour

La transformation de l'opération d'obtention est définie par l'application directe de la fonction de définition, mais la transformation des opérations de mise à jour est liée aux conditions suivantes:

- Toute modification sur SR2 doit se transformer en une modification équivalente sur les relations de SR1 (l'effet est le même et pas plus).

- Une mise à jour sur SR2 doit vérifier les contraintes d'intégrité CI2 et sa transformation sur SR1 doit vérifier les contraintes d'intégrité CI1.

- Il n'y a pas d'interférence (cf §2.1.3.3) entre les opérations de plusieurs schémas qui ont été définis sur le Schéma-1.

Nous allons regarder en détail chacune de ces conditions.

2.1.3.1 Changement admissible (Condition 1):

Pour l'usager du Schéma-2, SR2 est une relation avec une occurrence dans la base de données. En realité, SR2 est une relation abstraite définie sur des relations du Schéma-1, et donc son occurrence est seulement dérivable à partir des occurrences des relation du Schéma-1. (Consi-

dérons pour l'instant que le Schéma-1 est le schéma conceptuel et que la correspondance vers le schéma interne est directe, c'est à dire qu'une relation de base est un fichier du schéma interne).

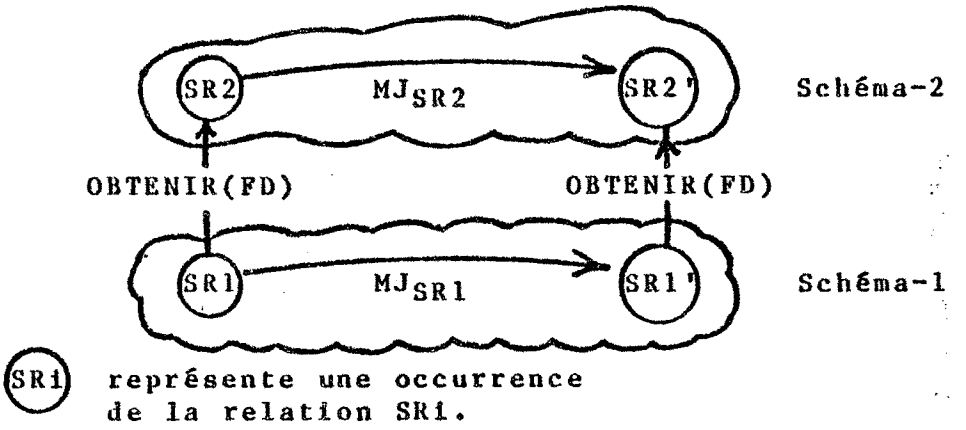


fig 2.1 Changement Admissible

Pour maintenir cette transparence, tout changement (insertion, suppression ou modification) de " l'occurrence" de SR2 doit être transformé en un changement équivalent sur SR1, c'est à dire: si d'une part FD(SR1) est la fonction de définition de SR2 sur SR1, et OBTENIR(FD(SR1)) permet de matérialiser l'occurrence de SR2; et si d'autre part MJ_{SR1} est la traduction dans le Schéma-1 de l'opération de mise à jour (MJ_{SR2}) sur le Schéma-2, alors:

$$MJ_{SR2}(\text{OBTENIR}(\text{FD}(\text{SR1}))) = \text{OBTENIR}(\text{FD}(MJ_{SR1}(\text{SR1}))))$$

ce qui est représenté par la figure 2.1. Si cette condition est satisfaite par l'opération de mise à jour alors le changement est admissible.

2.1.3.2 Mise à jour sémantiquement intègre:(Condition 2)

Toute opération de mise à jour doit satisfaire les contraintes d'intégrité portant sur la relation à modifier, c'est à dire: tout changement doit maintenir les données conformes aux règles de description. Sur le Schéma-1, l'ensemble des contraintes d'intégrité contient les restrictions ou contraintes fixant la politique de l'entreprise, les autorisations d'utilisation des données et la cohérence des données. Le Schéma-2 est une vue du Schéma-1 sur laquelle, l'usager peut définir un ensemble de contraintes exprimant des conditions sur les liens sémantiques entre les relations du schéma. Ces contraintes sont exprimées en termes de relations du Schéma-2.

Une opération de mise à jour sur SR2 doit alors satisfaire CI2, et sa traduction sur SR1 doit satisfaire CI1.

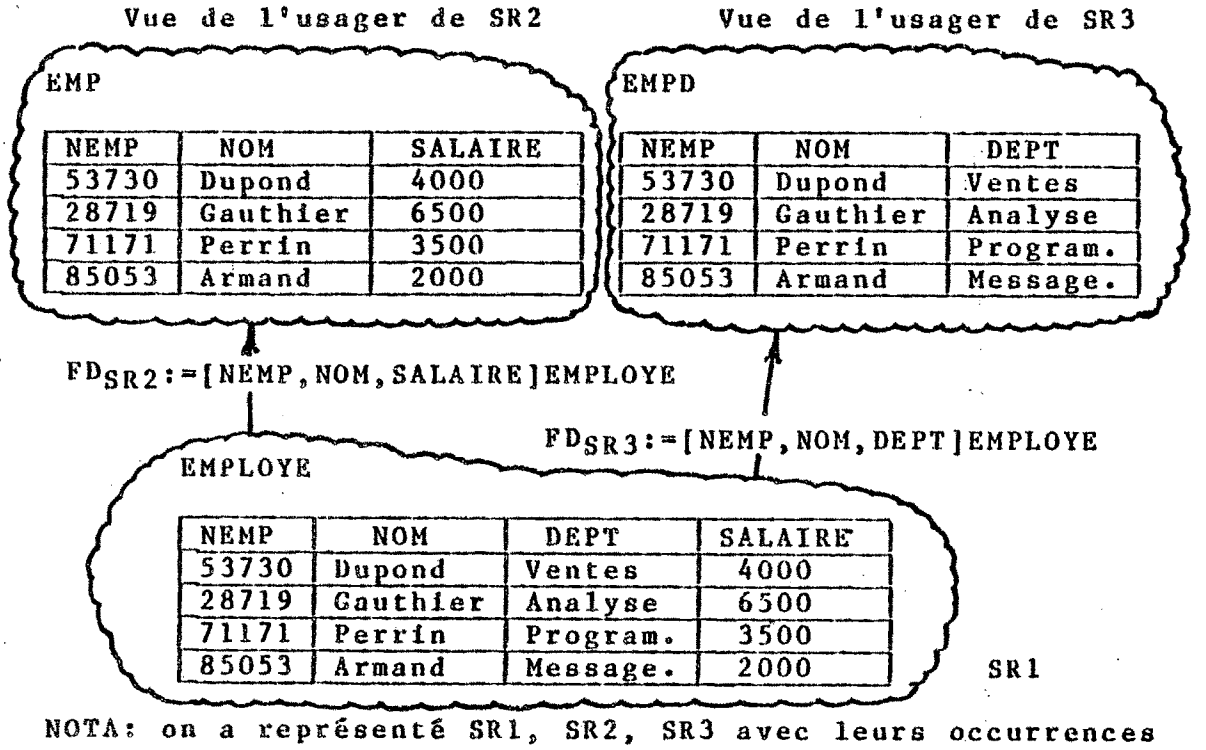
SATISFAIRE(CI2) ET SATISFAIRE(CI1) = VRAI

Condition 2

2.1.3.3 Mise à jour sans interférences: (Condition 3)

L'approche multi-schéma permet la définition d'un autre schéma, le Schéma-3, sur les relations du Schéma-1. Considérons que SR3 est une relation abstraite définie sur les mêmes relations que SR2 mais avec une fonction de défini-

tion différente. Un exemple est donnée dans la figure 2.2, où nous avons défini SR2 (EMP) et SR3 (EMPD) comme des projections de SR1 (EMPLOYE).



NOTA: on a représenté SR1, SR2, SR3 avec leurs occurrences pour mieux visualiser l'exemple.

Fig 2.2 Interférence de schémas

Une opération d'insertion sur EMP ou EMPD est traduite en une opération d'insertion sur la relation EMPLOYE; les valeurs des attributs manquants dans la définition des relations EMP ou EMPD doivent être complétées avec la valeur "indéfini" (notée "?"). Le lecteur peut se rendre compte qu'une insertion sur EMP crée une valeur indéfinie sur l'attribut DEPT de la relation EMPLOYE et donc sur "l'occurrence" de la relation EMPD; et vice versa, une insertion sur EMPD crée des valeurs indéfinies pour EMP.

Un problème se présente si les usagers de EMP et EMPD font l'insertion simultanée du même employé, p.ex: l'usager de SR2 fait INSERER(EMP,[83251,Dupont,7000]) et l'usager de SR3 fait INSERER(EMPD,[83251,Dupont,Ventes]). Supposons que l'insertion de SR2 ait été réalisée la première créant ainsi un n-uplet [83251,Dupont,?,7000] sur la relation EMPLOYE. Donc, pour réaliser l'insertion de SR3 deux solutions se présentent:

i) Rejeter l'insertion parce que la clé est déjà existante, en obligeant l'usager de SR3 à réaliser une modification du champ inconnu.

ii) Accepter l'insertion en la transformant automatiquement en modification pour laquelle il faudra obéir à des restrictions, p.ex: accepter la modification si les valeurs définies sont les mêmes et compléter les valeurs indéfinies, ou accepter la modification dès que la valeur de la clé est la même (solution un peu plus dangereuse mais moins restrictive). Dans l'exemple, l'opération INSERER(EMPD,[83251,Dupont,Ventes]) est transformée en MODIFIER(EMPLOYE,[83251],[83251,Dupont,Ventes,?]) ce qui donne le nuplet [83251,Dupont,Ventes,7000] sur la relation EMPLOYE.

Ce problème est connu sous le nom d'interférence entre schémas (PAP78). La troisième condition pour réaliser la transformation est alors: une opération de mise à jour sur

le schéma SR2 ne doit pas occasionner d'interférences avec les contraintes d'intégrité du schéma SR3 (ou d'autres schémas).

Ceci implique une augmentation dynamique de l'ensemble CI2 avec les contraintes provenant de l'interférence avec l'ensemble CI3. (CI2=Union[CI2,INTERFERENCE(CI3)]).

SATISFAIRE(CI2) et SATISFAIRE(contraintes de CI3 en interférence) = VRAI.

Condition 3

Un mécanisme pour résoudre le problème de la correspondance, ainsi que divers exemples sont donnés au Chapitre 3.

2.1.4 Généralisation de la définition d'un schéma:

Nous avons évoqué plusieurs fois dans le texte la possibilité d'une définition récurrente de schémas. La figure 2.3 représente cette récursivité avec une restriction pour des raisons de confidentialité et de contrôle: tout schéma d'un niveau plus haut est complètement défini sur un et un seul schéma du niveau immédiatement inférieur (voir fig 2.4). Cette règle doit être appliquée à partir du schéma conceptuel de la base de données.

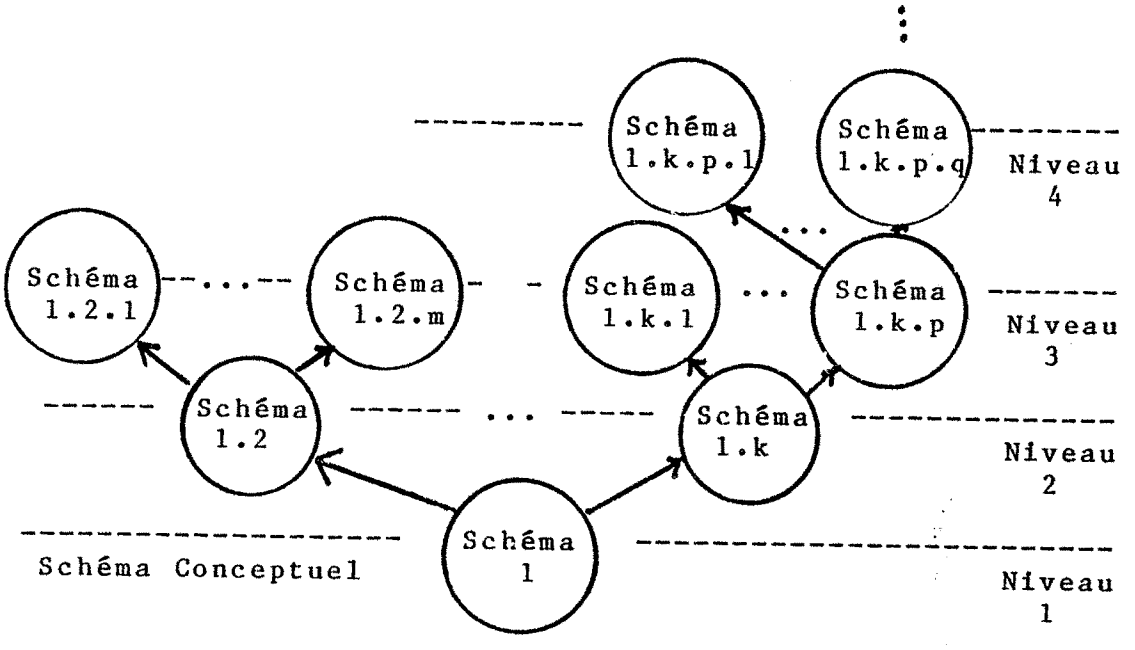
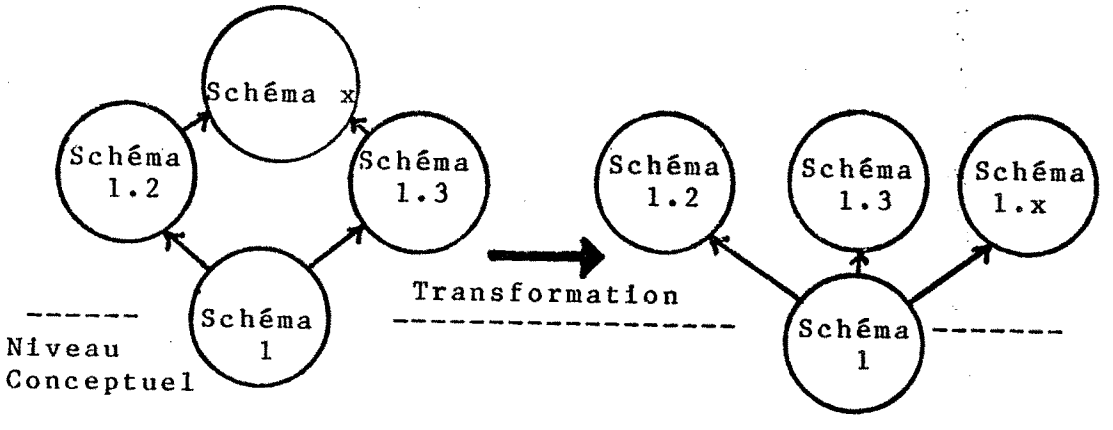


Figure 2.3 Définition récurrente de schémas

Le passage des opérations d'un schéma vers des opérations sur les schémas inférieurs doit tenir compte des conditions définies dans les paragraphes précédents. La transformation finale permet d'exprimer les opérations d'un niveau k par des opérations sur les relations de base du schéma conceptuel.



a)Définition incorrecte b)Définition correcte

Figure 2.4 Définition d'un schéma

Le lecteur aura remarqué que jusqu'à présent nous n'avons pas abordé le problème d'un schéma conceptuel

constitué de relations fragmentées et réparties sur plusieurs sites d'un réseau d'ordinateurs. Dans le paragraphe suivant, nous allons définir le problème de la correspondance dans un SGBDR.

2.1.5 La correspondance de schémas d'un SGBDR:

L'approche que nous venons de décrire s'applique directement dans un SGBDR à partir du schéma conceptuel. La correspondance définie en § 2.1.2, 2.1.3, 2.1.4 se situe entre le niveau externe et le niveau conceptuel de la base de données répartie. Nous allons voir comment la correspondance entre le schéma conceptuel et le schéma interne peut être considérée comme un problème typique des SGBDR (mais aussi des SGBD centralisés dans certaines limites).

Considérons le schéma conceptuel formé de relations fragmentées où une relation est composée d'une ou plusieurs sous-relations. La fragmentation des relations peut être réalisée selon deux approches: l'approche descendante ou l'approche ascendante.

2.1.5.1 L'approche descendante: (ROT80,ESP78)

Les relations du schéma conceptuel ont été conçues en respectant toutes les règles de la normalisation; cependant pour des raisons de performances, les relations sont découpées horizontalement par des restrictions simples ou verticalement par des projections.

Exemple: Soit la relation EMPLOYE(Nemp,Nom,Dept,Salaire) une relation du schéma conceptuel avec comme occurrence:

EMPLOYE

	Nemp	Nom	Dept	Salaire	
Frag.1	53730	Dupond	Ventes	4000	Dept=Ventes
	28719	Gauthier	Ventes	2000	
	71171	Perrin A	Ventes	3500	
Frag.2	85053	Perrin B	Product.	6500	Dept=Prod.
	78719	Rey	Product.	4700	
Frag.3	15971	Perrin C	Program.	3500	Dept=Prog.

Cette relation est découpée horizontalement en trois fragments qui séparent les employés par département. La relation EMPLOYE se retrouve par l'opération:

$$UNION_{i=1,3}(Fragment_i).$$

La relation peut aussi être découpée verticalement de deux manières:

i) Par des projections qui incluent la clé de la relation:

Nemp	Nom	Salaire
53730	Dupond	4000
28719	Gauthier	2000
71171	Perrin A	3500
85053	Perrin B	6500
78719	Rey	4700
15971	Perrin C	3500

Nemp	Dept
53730	Ventes
28719	Ventes
71171	Ventes
85053	Product.
78719	Product.
15971	Program.

La relation EMPLOYE est retrouvée par l'opération de composition ("JOIN") des deux fragments. Dans l'exemple l'opération de composition porte sur la clé des deux rela-

tions; elle est donc réalisée sans perte d'information (RID73). Tout découpage vertical doit prendre en compte les propriétés des opérations de composition sans perte d'information (RID73,ABU79,DAB78b,etc).

ii) Par des projections qui peuvent ne pas inclure la clé de la relation, ce qui correspond au cas d'un partionnement de fichiers.

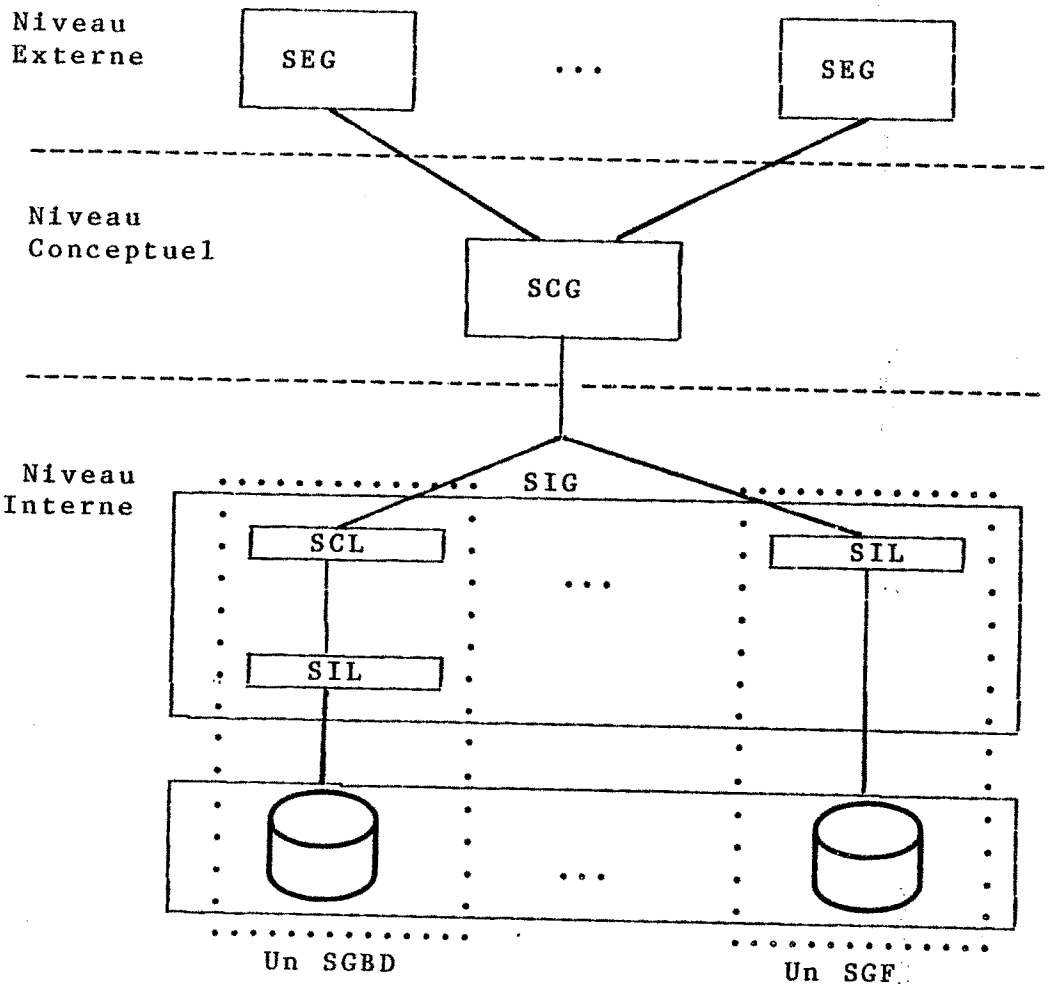
id.	Nemp	Salaire
1	53730	4000
2	28719	2000
3	71171	3500
4	85053	6500
5	78719	4700
6	15971	3500

id. uplet	Nom	Dept
1	Dupond	Ventes
2	Gauthier	Ventes
3	Perrin A	Ventes
4	Perrin B	Product.
5	Rey	Product.
6	Perrin C	Program.

Dans ce cas la relation EMPLOYE peut être retrouvée de deux façons:

a) Par la définition d'une opération de concaténation de fichiers.

b) Par l'addition d'un nouveau champ à chaque fragment indiquant l'identificateur de chaque nuplet. L'identificateur est unique et généré automatiquement par le système; ce champ est donc considéré comme la clé du fragment. Pour retrouver la relation EMPLOYE, on réalise l'opération de composition sur ce champ. SDD-1 et INGRES ont utilisé cette solution (ROT80, ESP78).



SEG: schéma externe global
SCG: schéma conceptuel global
SIG: schéma interne global
SCL: schéma conceptuel local
SIL: schéma interne local
SGF: système de gestion de fichiers
SGBD: système de gestion de bases de données

figure 2.5 L'approche descendante

L'approche descendante permet de mieux contrôler la fragmentation d'une relation et de placer les fragments (et leurs copies éventuelles) dans les différents sites du réseau d'ordinateurs. Ceci se ramène à un problème d'allocation de données pour lequel plusieurs solutions ont été proposées (CHU73,ADD78,PAD79).

La figure 2.5 montre une architecture possible pour une base de données répartie conçue de manière descendante et dont les fragments peuvent être accédés soit au travers d'un SGBD soit au travers d'un système de gestion de fichiers. La correspondance entre le schéma conceptuel (SCG) et le schéma interne (SIG) du SGBDR est définie directement par la fonction de définition de chaque relation du SCG et par sa fonction inverse (cf. Chapitre 3).

Cette approche de fragmentation descendante rejoint les problèmes de normalisation des relations et de l'équivalence de schémas (BBG78,DEL79). Dayal (DAY79) propose une fragmentation verticale en sous-relations indépendantes sans perte d'information (RIS77). Ce type de fragmentation permet d'éliminer toutes les dépendances fonctionnelles inter-relationnelles (considérées comme des contraintes d'intégrité) et de résoudre ainsi le problème du test de ces dernières. Une telle fragmentation doit satisfaire les conditions suivantes: (RIS77)

1) Il existe une fonction bijective entre les fragments et la relation fragmentée. Ceci signifie que la relation fragmentée est définie par l'opération de composition ("Natural Join") et que la fonction inverse (retrouver directement un fragment) correspond à des projections sur la relation fragmentée. Ces projections doivent être des fonctions bijectives

(correspondance 1-1 des nuplets) pour assurer la récupération de la relation fragmentée par l'opération de composition.

ii) La deuxième condition impose une préservation des dépendances fonctionnelles: étant donnée, F^*= l'ensemble de toutes les dépendances fonctionnelles de la relation fragmentée (y compris les dépendances dérivées par les règles d'inférence) et F1, F2 les ensembles de dépendances fonctionnelles respectivement des fragments FRAG1 et FRAG2, la condition:

$$F^* = (F1 \cup F2)^*$$

doit être satisfaite, pour assurer la préservation des dépendances fonctionnelles. Dans ce cas on dit que FRAG1 et FRAG2 sont deux projections indépendantes.

iii) L'opération de composition doit se réaliser sans perte d'information: si X1 et X2 sont respectivement les ensembles d'attributs de FRAG1 et FRAG2, et si les dépendances fonctionnelles $X1 \cap X2 \rightarrow X1$ ou $X1 \cap X2 \rightarrow X2$ se retrouvent dans F^*, alors la relation fragmentée est une représentation sans perte d'information des fragments FRAG1 et FRAG2.

Comme $F1 \cap F2 = \phi$, la préservation des dépendances fonctionnelles peut être réalisée sur chaque fragment et sans communication avec les autres sites. Cependant, ce type de

fragmentation en composantes indépendantes ne peut être utilisé pour la préservation des dépendances fonctionnelles multivaluées (DEL79,DAY79).

2.1.5.2 L'approche ascendante:(ADD77,ADI78)

Dans l'approche ascendante, les relations du schéma conceptuel sont définies à partir des schémas (externes ou conceptuels) de bases de données déjà en exploitation et que l'on ne veut pas modifier. Les deux mêmes types de fragmentation horizontale et verticale peuvent être utilisés pour former les relations du schéma conceptuel. La différence la plus importante avec l'approche précédente est que la fragmentation est conditionnée par la structure des fragments. On rencontre deux problèmes importants: des duplications partielles de n-uplets et les valeurs indéfinies de certains attributs.

L'exemple suivant illustre ces cas:
Considérons trois relations avec leurs occurrences:
(Nota: "Toute ressemblance ... coïncidence")

CHERCHEUR1

NOM	SALAIRE	PROJET
Fernandez	2000	Microbe
Diehl	2500	Microbe
Adiba	6000	Polyphème
Paik	2000	Polyphème
Stiers	5000	Bureautique
Nguyen	5600	Microbe
Euzet	5000	Polyphème

Cette relation contient les chercheurs de l'équipe Bases de Données.

La relation CHERCHEUR2 contient les chercheurs qui ont travaillé dans des projets déjà finis:

CHERCHEUR2

NOM	SALAIRE	PROJET
Adiba	6000	Polyphème
Euzet	5000	Polyphème
Caleca	2500	Polyphème
Paik	2000	Polyphème
Decitre	7000	Polyphème
Cristian	2500	Sésame
Briat	5000	Ours
Forestier	2500	Ours

et la relation CHERCHEUR3 contient une partie des chercheurs de POLYPHEME et l'activité développée:

CHERCHEUR3

NOM	ACTIVITE
Adiba	Modélisation
Euzet	Interprétation
Caleca	Optimisation
Nguyen	Optimisation
Decitre	Exécution répartie

Maintenant, on considère que dans le schéma conceptuel doit apparaître une relation CHERCHEUR qui intègre les schèmes des trois relations. L'occurrence de cette relation lors de l'application de l'opération d'obtention est donc donnée par:

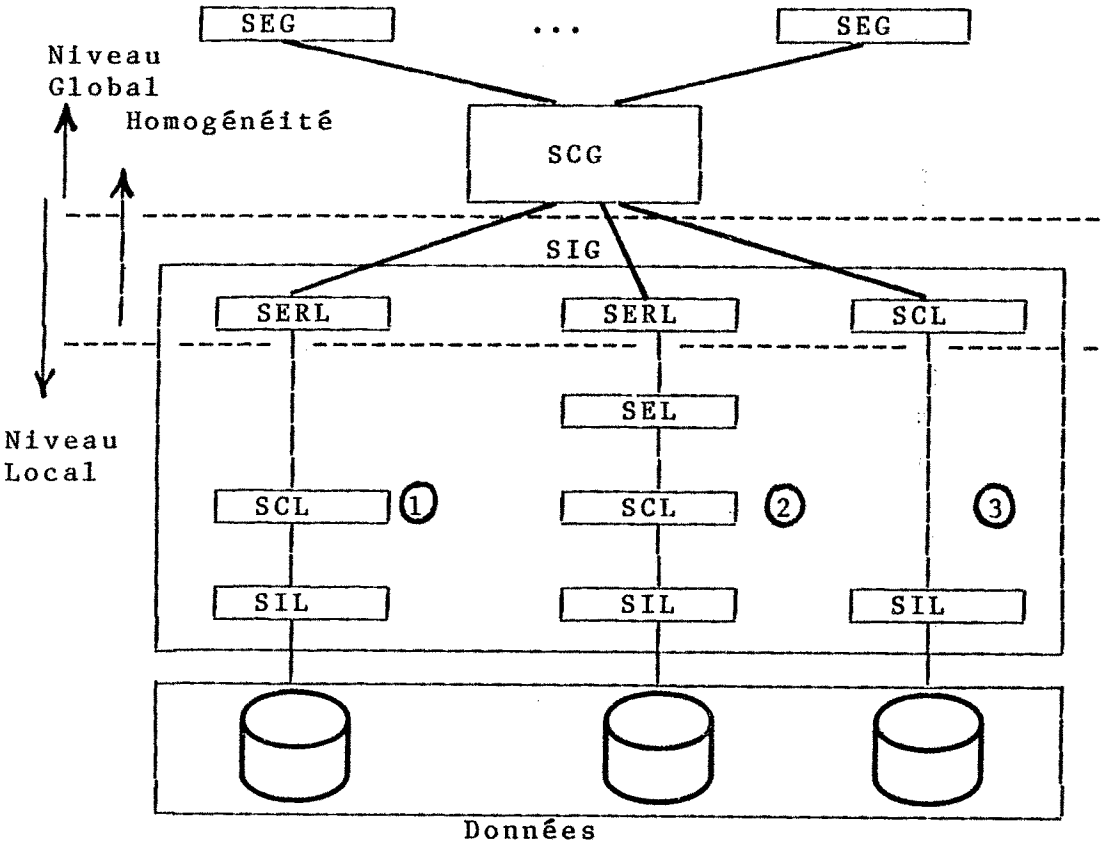
CHERCHEUR

	SALAIRE	PROJET	NOM	ACTIVITE	
	2000	Microbe	Fernandez	?	
	2500	Microbe	Diehl	?	
	5000	Bureautiq.	Stiers	?	
Cher-cheur1	5600	Microbe	Nguyen	Optimisation	
	2000	Polyphème	Paik	?	
	6000	Polyphème	Adiba	Modélisation	
	5000	Polyphème	Euzet	Interprétation	Cher-cheur3
	2500	Polyphème	Caleca	Optimisation	
Cher-cheur2	7000	Polyphème	Decitre	Exécution rep.	
	2500	Sésame	Cristian	?	
	2500	Ours	Forestier	?	
	5000	Ours	Briat	?	

Cette relation est formée par une fragmèntation verticale de deux fragments: la relation CHERCHEUR3 et une relation formée par l'UNION des relations CHERCHEUR1 et CHERCHEUR2 qui est une fragmentation horizontale de ces deux dernières relations.

L'approche ascendante est difficile à mettre en pratique étant donné qu'il faut considérer tous les liens sémantiques des fragments pour éliminer toutes les incohérences possibles. Ceci amène à des restrictions sévères pour la formation des relations du schéma conceptuel (cf Chapitre 3). La figure 2.6 résume l'architecture d'une base de données répartie (BDR) conçue de façon ascendante; il faut noter dans cette architecture:

1.Un niveau d'homogénéisation où l'on retrouve le même modèle de données (le modèle relationnel).



SEG: Schéma externe global
SCG: Schéma conceptuel global
SIG: Schéma interne global
SEL: Schéma externe local
SCL: Schéma conceptuel local
SIL: Schéma interne local
SERL: Schéma externe relationnel local

Figure 2.6 Approche ascendante

2.Deux niveaux (global et local) comme nous l'avons souligné au paragraphe 1.2; le niveau global correspond à la structure de la BDR et le niveau local aux structures des bases de données existantes ou Bases locales.

3.Le schéma externe relationnel local (SERL) qui correspond à une vue homogénéisée d'une Base Locale dont le modèle ne serait pas relationnel. Il représente soit tout

ou partie du schéma conceptuel (SCL) de la Base Locale (c'est le cas 1 de la fig. 2.6), soit tout ou partie d'un schéma externe (SEL) d'une Base Locale (c'est le cas 2 qui suppose un SGBD capable de gérer des schémas externes, cf §2.1.2).

4.Le cas 3 de la fig. 2.6, correspond à une Base Locale relationnelle.

5.Les cas 1 et 2 soulignent une différence très importante avec l'architecture de la BDR obtenue à partir d'une approche descendante (cf fig. 2.5).

Cette figure nous permet de distinguer trois types de correspondances entre les différents schémas de l'architecture ascendante d'une BDR:

i) entre un SEG et le SCG; cette correspondance doit respecter toutes les conditions du paragraphe 2.1.3,

ii) entre le SCG et le SIG qui se trouve réparti sur les Bases Locales,

iii) entre le SERL et la Base Locale; cette correspondance provient de la phase d'homogénéisation qui introduit un niveau supplémentaire.

La correspondance entre le SCG et le SIG dépend directement de la nature des relations du niveau d'homogénéisation local. Nous voudrions insister sur plusieurs points importants de cette approche:

- La valeur indéfinie est très importante car elle permet de représenter soit des nuplets supplémentaires à rajouter à une relation, soit des valeurs qui doivent rester inconnues ou indéfinies. Par exemple, si une des relations locales est une restriction d'une des relations du SCL, alors on ne pourra pas rajouter de nuplets en dehors de cette restriction.

- D'autre part, il faut résoudre le problème de l'unicité de la clé des relations et le problème des duplications partielles des nuplets.

- La notion de répartition liée à la notion de fragmentation permet de distinguer un nouveau type de fragmentation: la fragmentation par voisinage avec une autre relation du SCG (ADI78). Voyons ceci avec un exemple.

Exemple:

Soit la relation DEPARTEMENT(Dept,Nom-dept,Budget), une relation du SCG qui est fragmentée horizontalement en 2 fragments:

i) DEPT1(Dept,Nom-dept,Budget) qui contient les départements avec un Budget $>$ 10 MF.

ii) DEPT2(Dept,Nom-dept,Budget) avec les départements ayant un Budget $<$ 10 MF.

Soit la relation EMPLOYE(Nemp,Nom,Salaire,Dept) du SCG avec la caractéristique suivante: elle est fragmentée par

le critère de fragmentation de la relation DEPARTEMENT; c'est à dire qu'elle est fragmentée horizontalement en 2 fragments:

i) EMP1(Nemp,Nom,Salaire,Dept) avec tous les employés des départements de Budget $>$ 10 MF.

ii) EMP2(Nemp,Nom,Salaire,Dept) avec tous les employés travaillant dans des départements de Budget $\leq$ 10 MF.

La répartition par voisinage indique que l'on veut placer les fragments correspondants des relations DEPARTEMENT et EMPLOYE (p.ex. DEPT1 et EMP1) dans le même site (ou dans la même base). Le lecteur pourra noter que la fonction de définition de la relation EMPLOYE est donnée par l'UNION des fragments EMP1 et EMP2, mais que la fonction inverse doit faire intervenir la relation DEPARTEMENT pour déterminer sur quel fragment l'opération de mise à jour doit être réalisée.

Dans le Chapitre 3, nous analyserons la correspondance entre le SCG et le SIG dans l'approche ascendante, et donc ce cas particulier de fragmentation.

2.1.6 MOGADOR:

Rappelons que dans POLYPHEME, nous avons pris une approche ascendante dans la conception de la BDR avec une hypothèse d'hétérogénéité des Bases Locales (cf §1.1 et

cas 1 et 2 de la fig. 2.6). MOGADOR (ADI77), est un modèle de données réparties qui permet d'adapter les schémas hétérogènes des Bases Locales à des schémas relationnels. Le modèle, qui repose sur la théorie des ensembles et sur le modèle "Data Semantics" (ABR74), permet d'établir les liens sémantiques entre les données en supprimant ainsi certaines ambiguïtés. Le processus de transformation d'un schéma MOGADOR en un schéma relationnel en 3FN est donnée dans (ADI78).

Dans MOGADOR, les objets du monde réel- qui existent à priori dans l'univers- sont caractérisés sous forme d'ensembles statiques (appelés Catégories Abstraites) qui ne changent pas avec le temps. Cette notion correspond directement à la notion de Domaine dans le modèle relationnel (COD70). Une catégorie abstraite est identifiée par un nom et un type de données (entier, réel, logique ou chaîne) qui représente une caractéristique particulière des données. Sur les catégories abstraites (que l'on suppose finies) on ne peut réaliser que deux types d'opérations: énumérer les éléments de l'ensemble ou regarder (tester) si un élément de l'ensemble existe déjà.

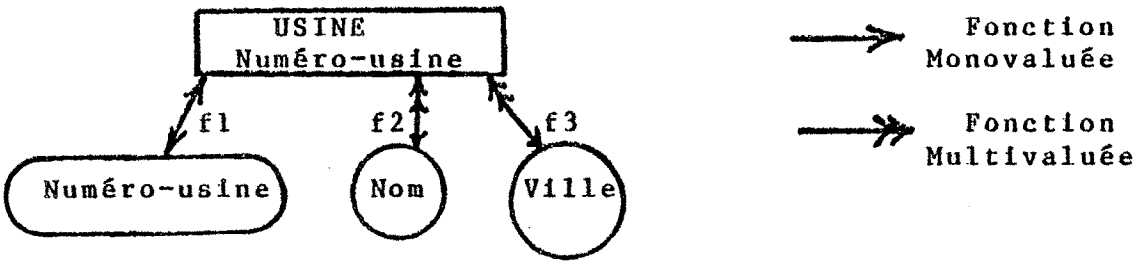
Pour caractériser des entités qui représentent une abstraction dynamique de l'univers d'objets, c'est à dire des ensembles d'objets qui peuvent changer avec le temps, MOGADOR dispose des Catégories Concrètes. Elles représen-

tent les deux formes d'abstraction de (SMI77) : généralisation et agrégation . La première permet de considérer les objets de même nature comme faisant partie du même ensemble générique (catégorie concrète). La deuxième regroupe des entités (objets ou catégories) de natures différentes dans une seule catégorie concrète; plusieurs des caractéristiques des entités peuvent être ignorées.

Une catégorie concrète de MOGADOR est définie par son nom et le nom d'un identificateur qui permet la distinction des objets de l'ensemble. Une catégorie concrète correspond, dans la majorité des cas, à une généralisation d'objets et à une agrégation d'autres catégories (abstraites ou concrètes) qui indiquent les caractéristiques des objets généralisés.

Exemple:

USINE est une généralisation d'un ensemble d'usines, et si Numéro-usine, Nom, Ville sont des catégories abstraites qui dénotent des caractéristiques communes entre les usines, le graphe suivant est une agrégation:



Les flèches du graphe représentent des fonctions (monovaluées ou multivaluées) entre catégories et elles matérialisent les agrégations. Les fonctions permettent alors de relier des ensembles (catégories abstraites et concrètes) et d'exprimer ainsi leurs liens sémantiques.

Les fonctions acceptent les opérations suivantes:

- accès: obtenir f(a), si "f" est le nom de la fonction et "a" l'ensemble source.
- lier: relier deux éléments entre eux par une fonction.
- délier: déconnecter deux éléments des ensembles.
- graphe: obtenir toutes les valeurs (éléments) sur lesquelles la fonction est définie.

Ces quatre opérations et les quatre autres qui s'appliquent sur les catégories concrètes (énumérer, tester, rajouter ou supprimer des éléments de l'ensemble) peuvent être considérées comme des règles d'évolution qui permettent à l'administrateur de décrire la sémantique de la manipulation associée à chaque catégorie et à chaque fonction. Après le passage à un schéma relationnel, on aura pour chaque relation quatre règles d'évolution indiquant la définition et les contraintes des opérations d'obtention, d'insertion, de suppression et de modification des n-uplets d'une relation.

Le processus de conception de la BDR avec MOGADOR est résumé de la façon suivante:

i). Homogénéiser les schémas des Bases Locales. Ceci implique de transformer ces schémas en des schémas MOGADOR et de spécifier la correspondance. Adiba (ADI77,CAS77) propose pour cela l'utilisation de programmes (appelés "programmes locaux") écrits dans le langage de manipulation du SGBD local et activables par l'usager de la base locale.

ii) Rapprocher (généraliser) ces schémas dans un seul schéma: le schéma conceptuel de la BDR. L'administrateur est obligé d'expliciter les règles d'évolution qui indiquent la solution aux ambiguïtés sémantiques. La correspondance entre le schéma conceptuel et les schémas locaux est donnée par des règles (appelées règles globales) explicitées par un programme global (ADI78).

iii) Un schéma MOGADOR est facilement transformable en un schéma relationnel binaire et donc en un schéma relationnel n-aire mais avec quelques restrictions (cf. page 4.22 de ADI78). Les schémas MOGADOR sont alors transformés en schémas relationnels n-aires qui sont en 3FN. La correspondance entre schémas se réduit alors, à la spécification des règles d'évolution des opération d'obtention, insertion, suppres-

sion, et modification des n-uplets de la relation, toujours explicitées par un programme. Dans la correspondance entre le niveau global (schéma conceptuel de la BDR) et le niveau local (schémas des Bases Locales) on peut identifier un traitement standard ou non-standard dépendant du mode de fragmentation des relations globales (cf. Chapitre 3).

La correspondance entre le schéma relationnel des Bases Locales et le schéma dans le modèle propre aux SGBD locaux peut être réalisée soit en utilisant la technique des programmes locaux soit en utilisant les mécanismes développées dans le projet URANUS (NGU77), approche que nous avons utilisée dans la maquette et qui consiste à réaliser une transformation des informations (cf. Chapitre 4).

2.1.7 L'usager et la correspondance:

Un usager est celui qui s'adresse au système soit pour définir différents schémas et leur correspondance (l'administrateur), soit pour manipuler les données au travers d'un schéma particulier (l'utilisateur). Dans l'architecture multi-schémas d'un SGBD, l'utilisateur agit au niveau du schéma conceptuel ou du schéma externe. Cependant, les manipulations de l'usager sur le schéma externe doivent satisfaire aux conditions du paragraphe 2.1.2. Par contre, tout changement réalisé directement sur le schéma concep-

tuel se traduit sans ambiguïté en des changements équivalents sur n'importe quel schéma externe; ceci vient du fait que les relations du schéma externe sont définies sur des relations du schéma conceptuel par des fonctions.

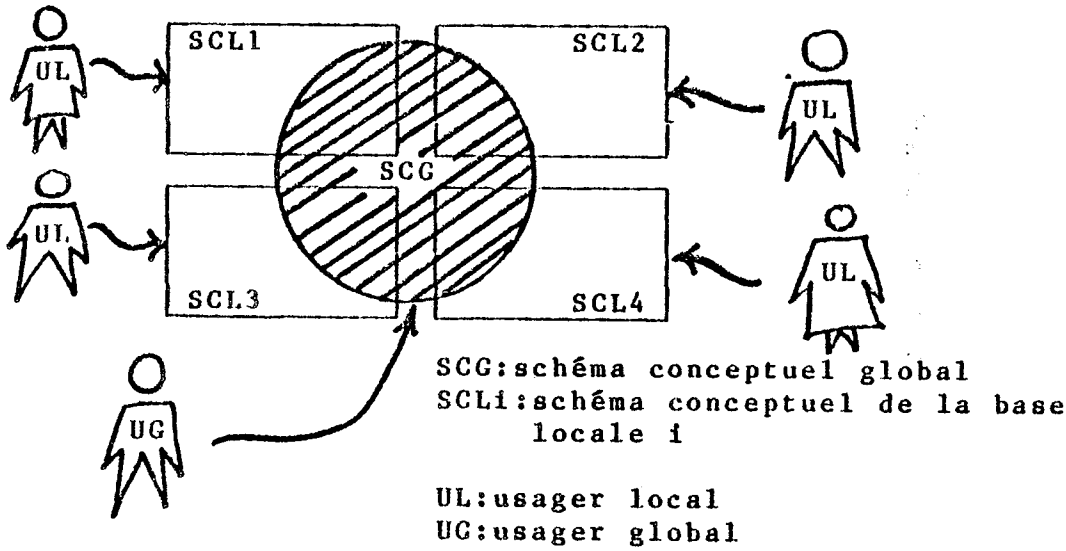
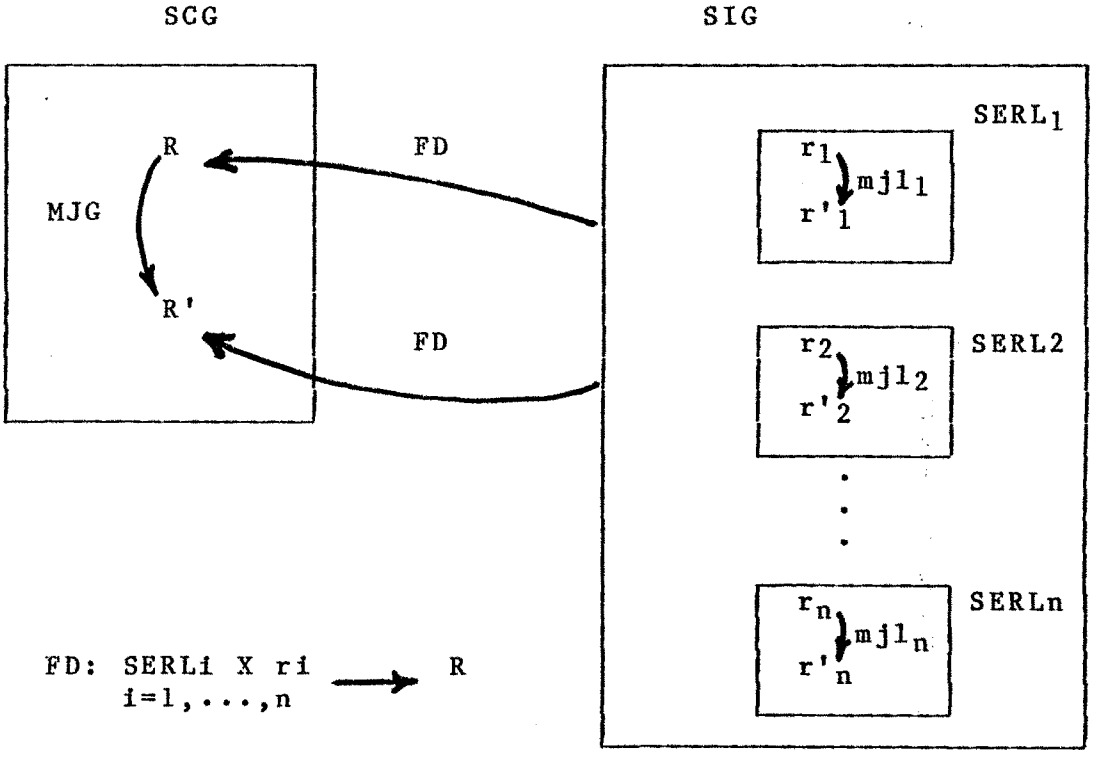


figure 2.7 Les utilisateurs de POLYPHEME

Dans l'architecture de POLYPHEME, nous avons identifié deux types d'usagers: (voir fig. 2.7) l'usager du SGBDR (ou global) et l'usager d'une Base Locale (ou local). Le premier s'adresse au schéma conceptuel global ou à un schéma externe global, et indirectement il s'adresse aux vues locales (SERL de la fig. 2.6). L'usager de la Base Locale s'adresse au schéma conceptuel local (cf cas 1 et 2 de la fig. 2.6) sur lequel il doit pouvoir faire des opérations d'obtention et de modification (hypothèse de départ de POLYPHEME, cf. §1.1).

Nous allons analyser rapidement la correspondance entre le schéma conceptuel global et le schéma interne global.

Dans la figure 2.6, le schéma interne global (SIG) est composé par des SERLs et par des parties propres aux SGBD dont nous ne tiendrons pas compte ici. Le SIG est donc considéré comme un ensemble de SERLs. Une relation globale (qui appartient au SCG) est construite à partir des relations des SERLs, et donc sa fonction de définition est du type: FD: $SERL_i$ X r_i ⟶ R avec i=1,2,...,n et où r_i est un ensemble de relations du schéma $SERL_i$ et R est la relation globale.



FD: fonction de définition
R : relation globale
R': relation R modifiée
r_i: relation locale du schéma $SERL_i$
r'_i: relation r_i modifiée
SCG: schéma conceptuel global
SIG: schéma interne global
SERL: schéma externe relationnel local
MJG: mise à jour globale
mjl_i: mise à jour dans le schéma $SERL_i$

Figure 2.8 Correspondance SCG-SIG

Une modification (de "l'occurrence") d'une relation globale réalisée par l'usager global, doit se traduire dans un changement équivalent sur le SIG: (voir fig.2.8)

$$\underset{R'}{MJG\ (\ \underset{R}{FD\ (\ SERL_i\ ,\ r_i\ )}\ )} \quad = \quad \underset{R'.}{FD\ (\ SERL_i\ ,\ \underset{r'_i}{mjl_i\ (\ r_i\ )}\ )}$$

L'ensemble des contraintes d'intégrité du SCG, CIG, est formé en partie par l'union des ensembles des contraintes d'intégrité des $SERL_i$. Donc toute mise à jour globale qui satisfait les contraintes de CIG , doit satisfaire également les contraintes des SERL. Aucune interférence entre schémas ne peut être détectée car le SCG est un schéma unique. La correspondance entre le SCG et le SIG ne présente donc pas de problèmes majeurs. Cependant, la correspondance entre les changements réalisés par un usager local sur son schéma de travail (SCL ou SEL) et ceux équivalents sur le SCG, peut être source d'incohérences. Il n'est pas possible d'assurer que tout changement sur la partie correspondante à un SERL se traduit sans ambiguïtés en changements equivalents sur le SCG, car la définition d'une relation globale crée des dépendances entre les relations des SERL que l'on supposait indépendantes.

Au niveau des contraintes d'intégrité, une mise à jour locale peut satisfaire seulement les CI_{SERLi}. Ces contraintes peuvent se trouver généralisées dans l'ensemble CIC du schéma conceptuel global, c'est à dire qu'elles

peuvent être communes à d'autres SERLs, et donc une interférence peut être générée. Nous allons donner un exemple.

Exemple: Unicité de la clé d'une relation globale.

Soit la relation globale

PROJET (Numéro, Nom, Description, Date-fin, Budget)

constituée de deux fragments: PROJ1 avec les projets de Budget $<$ 200.000, et PROJ2 qui contient les projets avec Budget $\geqslant$ 200.000. Chaque fragment est sur un SERL différent et selon les hypothèses de départ un usager local peut réaliser, indépendamment de l'usager global et des autres usagers locaux, des opérations de modification. Prenons le cas d'une insertion d'un nouveau n-uplet: [P4,SD3,Systèmes Discrets,31-01-80, 74000] dans la relation PROJ1. Cette opération est cohérente dans SERL-1 mais elle crée des incohérences au niveau du SCG: le projet P4 existe déjà sur PROJ2 avec des caractéristiques différentes. L'unicité de la clé est alors rompue et en conséquence il y a une violation des dépendances fonctionnelles (contraintes d'intégrité) de la relation globale PROJET.

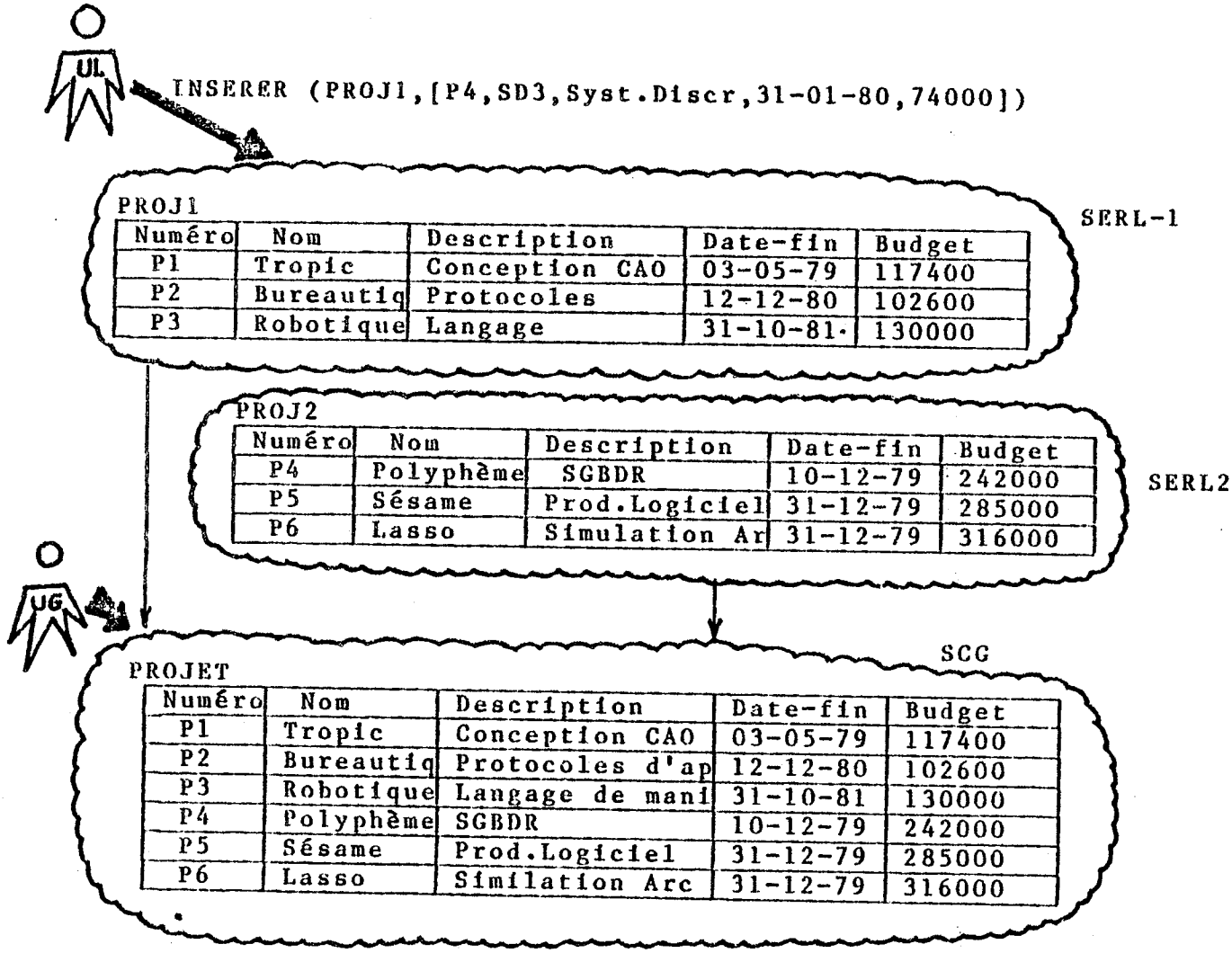


Figure 2.9 Exemple des incohérences produites par un usager local

Plusieurs solutions à ce problème sont envisageables:

i) Eliminer l'usager local: solution trop restrictive qui ne peut pas être appliquée dans tous les cas parce qu'une partie de la Base Locale ne participe pas à la coopération, et peut donc être manipulée indépendamment du SCG (cf. fig.2.7).

ii) Interdire à l'usager local toute mise à jour dans le SERL: ce qui contrôle les modifications au niveau du

SCG. Une modification des droits d'accès sur la partie correspondante du SERL dans le SCL doit être réalisée; cependant cela n'est pas toujours possible et dépend des capacités du SGBD local à prendre en compte ce type de contrôle.

iii) Eliminer les dépendances entre les SERL: solution difficilement applicable car elle va contre la notion de généralisation préconisée par POLYPHEME.

iv) Modifier le SGBD pour faire passer toute opération de modification de l'usager local portant sur le SERL, au travers du SCG ce que l'on s'est refusé dans POLYPHEME. C'est en fait la solution la moins "restrictive".

En résumé, la coexistence des usagers globaux et locaux introduit des restrictions sur les droits d'accès des usagers locaux ou sur la conception de la vue de la coopération. Une solution à ce problème dans le cas de la maquette de POLYPHEME sera discutée dans le Chapitre 4.

2.2 L'intégrité sémantique:

L'intégrité sémantique (cf §1.3) dans un SGBDR, loin de considérer de nouveaux mécanismes, se présente comme l'application dans un environnement réparti des concepts classiques des SGBD centralisés.

Une base de données sert à modéliser une certaine partie du monde réel; son contenu représente une configuration particulière de la partie du monde modélisée. Un ensemble de règles sert à contrôler la validité des états de la base. Un état est valide s'il satisfait ces règles; si une valeur ne leur obéit pas, il y a une erreur dans la base. On dira que la base de données a perdu son "intégrité".

L'intégrité est liée au problème de la validité des états (qualité des données) par la prévention des erreurs. Ceci implique la possibilité de détecter les erreurs et d'être capable de corriger leurs effets. L'intégrité sémantique considère des mécanismes pour la prévention des erreurs sémantiques: erreurs occasionnées accidentellement par des usagers qui font des modifications à la légère ou en ignorant certaines parties de la structure du schéma sur lequel ils travaillent.

L'idée fondamentale des différents mécanismes proposés est de définir l'ensemble des règles contrôlant la validité des données sous forme de <u>contraintes</u> qui limitent les transitions entre les différents états valides de la base de données (ESW75,HAM75,STO75,FLO74,DAT77,AST76). Une contrainte est une assertion ou prédicat que doivent vérifier les données à certains instants (états stables). Tous les mécanismes proposés s'accordent pour l'utilisation

d'un langage non-procédural pour exprimer ces contraintes. Généralement, la déclaration d'une contrainte est faite en deux parties:

i) l'expression de la contrainte à l'aide du langage de manipulation,

ii) la description des conditions ("quand et comment") dans lesquelles une contrainte doit être testée: à la compilation de la requête? A l'exécution? A la fin de l'exécution mais avant la validation (ou "commitment") des opérations de modification? Pour une opération de modification spécifique? Etc.

Pour maintenir l'intégrité sémantique dans un SGBDR on se trouve confronté aux problèmes suivants:

- spécifier le langage de définition des données,
- définir le type et la nature de ces contraintes,
- définir le niveau et la forme d'application des contraintes,
- établir la correspondance des contraintes du schéma conceptuel global avec le schéma interne global,
- vérifier l'interférence des contraintes entre les différents schémas externes (cf. §2.1.2).

Dans POLYPHEME, nous avons envisagé deux possibilités pour le langage de définition des contraintes: faire une extension au langage LADORE (ACA78), ou utiliser l'option

prédicat d'URANUS (NGU77). Ces deux possibilités n'ont pas pu, à ce jour, être mises en oeuvre car nous avons réalisé une première version de la maquette ne contenant pas de telles fonctions. Des exemples de spécification de contraintes à l'aide d'un langage particulier peuvent être trouvés dans: (AST76,STO75,ROS79,DAT77).

2.2.1 Types de contraintes:

Les contraintes d'intégrité peuvent être classées de la façon suivante: (ESC75,DAT77,HAM75)

a) Dans la déclaration de la structure des données:

i) Le type et la portée des domaines, par exemple:

DOMAINE Nom = CHAINE(16).

ii) Restriction des valeurs possibles d'un attribut; exemple:

Couleur= CHAINE(5) AVEC-VALEURS('rouge','noir')

b) Dans la définition des dépendances sémantiques entre les données:

i) La clé: elle permet l'unicité des n-uplets d'une relation. Elle est identifiée par un ou plusieurs attributs de la relation.

Exemple: Nom CLE .

ii) Les clés secondaires qui peuvent être maintenues par exemple avec l'opération UNIQUE (DAT77).

iii) Les dépendances fonctionnelles qui sont liées à

la conception de la structure des relations. Si la relation est en 3FN, ces dépendances sont garanties implicitement avec la maintenance de la clé et des clés secondaires.

iv) Assertions explicites:

- sur des n-uplets individuels,

  Exemple: Salaire < 20000

- sur des ensembles de n-uplets,

  Exemple: Les employés doivent gagner moins que 10 fois le volume des recettes si le département a des recettes positives.

Ces contraintes peuvent être statiques ou dynamiques. Elles sont dynamiques quand elles s'appliquent à la transition d'un état de la base à un autre.

Exemple:

Le salaire d'un employé doit être croissant quand il y a des augmentations: VIEUX.Salaire $\ll$ NOUVEAU.Salaire.

Les contraintes peuvent aussi être soit intra-relation (applicables sur un ou plusieurs n-uplets, sur un ou plusieurs attributs de la relation), soit inter-relation (définies entre deux ou plusieurs relations indiquant certains liens sémantiques existant entre elles et applicables généralement sur un ou plusieurs attributs des relations).

Dans un SGBDR on retrouve ces mêmes types de contraintes, auxquelles on peut rajouter les contraintes suivantes:

a) Sur la fragmentation des relations globales: conditions ou restrictions pour former des relations globales. Exemple: dans une fragmentation horizontale un prédicat peut être associé à chaque fragment (cf. Chapitre 3).

b) Sur la correspondance des opérations sur une relation globale en opérations sur les relations locales (cf.Chapitre 3).

Exemple: en fonction du type de fragmentation, des contraintes (dites de répartition) standard ou non-standard pourront être définies pour chaque opération de mise à jour (ADA79,ADI78). Les contraintes non-standard doivent être décrites explicitement.

c) Sur la localisation: ce sont des contraintes qui permettent de définir la situation des relations locales et de leurs copies.

Exemple: en respectant les notions de localité et de priorité d'accès à une copie pour la définition de nouvelles relations locales.

Nous considérons la définition d'une relation globale comme indépendante de la localisation des relations loca-

les et de l'existence d'éventuelles copies.

2.2.2 Prise en compte des contraintes:

La validation des contraintes d'intégrité peut être réalisée (pour chaque requête ou transaction qui modifie les données) de différentes façons:

- à la compilation, c'est à dire avant l'exécution,
- à l'exécution,
- après l'exécution mais avant (ou pendant) la phase de validation des mise à jour.

La violation d'une ou plusieurs contraintes entraîne le refus de l'opération de modification. L'utilisateur doit en être informé avec la liste des contraintes non satisfaites et des n-uplets qui les invalident (ce qui n'est pas toujours possible). Un système de reprise et de journalisation doit être mis en place car dans le cas des transactions, il faudra défaire les effets des mises à jour réalisées jusqu'à l'instant de l'erreur (ESC75,VER78)

Un autre problème qui apparaît dans la validation des contraintes est leur traitement face à la valeur "indefinie", interprétable sémantiquement de deux façons: inconnue (?) ou impossible (#). Regardons un exemple inspiré de (VAS79):

Soit la relation EMPLOYE(Nom,Age,Etat,Nom-épouse) avec comme occurrence:

NOM	AGE	ETAT-CIVIL	NOM-EPOUSE
Alain	?	marié	Gloria
Robert	?	marié	Andrée
Edouard	36	?	?
Chris	26	celibataire	#

L'âge d'Alain et de Robert est "inconnu" de même que l'état civil et le nom de l'épouse d'Edouard; par contre, Chris est célibataire donc la valeur de l'attribut Nom-épouse est "impossible". (COD75) présente une extension à l'algèbre relationnelle pour incorporer la manipulation des valeurs inconnues. Avec les contraintes d'intégrité on peut contrôler la valeur "impossible" étant donnée qu'elle est le résultat d'un lien sémantique avec les valeurs des autres attributs du n-uplet:

exemple: Si Etat-civil='célibataire' alors Nom-épouse='#'

Dans les cas où l'on considère au moins la valeur "inconnue", il faudra utiliser une logique tri-valuée (COD75) pour l'évaluation des conjonctions, disjonctions et négations des conditions (une contrainte pouvant être interprétée comme une condition). Donc, une contrainte évaluée "inconnue" ou "vraie" est considérée comme satisfaite; dans les cas de contraintes utilisant des opérateurs statistiques (p.ex. MOYENNE, CARDINALITE,etc) ou arithmétiques la valeur "indéfinie" peut être ignorée. (C'est la meilleure politique dans la majorité des cas).

Il y a plusieurs façons de mettre en oeuvre les contraintes dont les trois principales sont:

i) Considérer le sous-système d'intégrité comme composé d'un ensemble de conditions pour chaque relation; le sous système est un moniteur qui est appelé pour chaque opération de mise à jour et qui évalue les conditions pour la relation concernée.

ii) Utiliser le mécanisme de modification de requêtes (STO75) pour rajouter les contraintes sous la forme de conditions.

iii) Considérer les contraintes comme des programmes qui sont appelés automatiquement. Dans la littérature on connait ces programmes sous le nom "d'actions spontanées" ("Triggers" (AST76) ou "alerters" (HAS78,BCL79)).

Dans les paragraphes suivants nous allons voir les différentes formes de validation des contraintes.

2.2.3 Validation à la compilation:

Plusieurs contraintes statiques peuvent être validées à la compilation (ou à l'invocation) d'une opération de modification. Par exemple, sur la relation EMPLOYE(Nom,Age, Salaire,Titre) existent les contraintes d'intégrité:

a) $e \in$ EMPLOYE: e.Salaire > 0

b) $e \in$ EMPLOYE: e.Titre='Chercheur' $\rightarrow$ e.Salaire < 4000

Lors de l'invocation de l'opération d'insertion:

INSERER (EMPLOYE,t) avec t=[Dupont,36,5000,chercheur], les contraintes peuvent être appliquées (p.ex. en utilisant les mécanismes i) ou ii) du §2.2.2) sur les attributs correspondants; la contrainte b) est fausse et donc, l'insertion est rejetée. Si cette contrainte avait été vraie, alors l'opération d'insertion aurait pu être exécutée. Notons que les tests de cette contrainte supposent que les valeurs des attributs Titre et Salaire sont données avec les autres valeurs à modifier.

Hammer et Sarin (HAS78) proposent une validation à la compilation par la synthèse de "tests" (interactions avec la base de données) efficaces qui permettent de déterminer si une contrainte est invalidée par une opération particulière. Un test obtient de la base les valeurs nécessaires pour l'évaluation de la contrainte (cas d'une opération de modification, ou d'une opération d'insertion avec des contraintes d'intégrité sur des ensembles). Si la contrainte est vraie, la requête ou la transaction peut être exécutée. Il y a un test pour chaque contrainte; cependant, les tests peuvent être combinés pour former des tests qui doivent être appliqués pour chaque opération de mise à jour (insertion, modification ou suppression). Un test est considéré comme une "action spontanée" (cf. §2.2.2).

L'avantage de ce type de validation est de ne pas nécessiter une reprise de la requête ou de la transaction

lors de la violation d'une contrainte. Cependant, la validation et l'exécution sont sérielles et un verrouillage des n-uplets considérés est nécessaire pendant ces deux phases; en ce sens, cette technique peut s'avérer très coûteuse dans un SGBDR.

2.2.4 Validation pendant l'exécution:

Ce type de validation considère une évaluation des contraintes faite simultanément avec l'exécution des requêtes ou transactions.

La technique de modification des requêtes d'INGRES (STO75) permet de rajouter les contraintes comme de simples restrictions évaluées à l'exécution de la requête. Le traitement des contraintes qui utilisent des opérateurs statistiques ou arithmétiques (p.ex. MOYENNE ou CARDINALITE) doit être considéré comme une exception; leur évaluation est réalisée aprés l'exécution de la requête.

Badal (BAD79) propose de considérer l'exécution des transactions comme formée d'une séquence d'événements de lecture et de calcul (SL) suivie d'une séquence d'événements d'écriture (SE). L'évaluation des contraintes est réalisable aprés l'exécution de la séquence SL car le résultat (les modifications) de la transaction est déjà connu à ce moment là. Si les contraintes sont satisfaites, alors la séquence SE est exécutée. Ce système ne nécessite

pas de mécanisme de reprise car les contraintes sont testées avant toute opération d'écriture.

2.2.5 Validation aprés l'exécution:

Cette méthode considère l'exécution d'une transaction ou requête comme réalisée en deux phases: l'exécution proprement dite (lecture, et "écriture partielle" des nouvelles valeurs) et la validation des modifications (rendre permanentes les nouvelles valeurs). La validation des contraintes d'intégrité est réalisée entre les deux phases, et s'il y a violation de l'une d'elles, les nouvelles valeurs seront ignorées.

Un verrouillage des n-uplets à modifier est nécessaire pendant les "trois" phases d'exécution de la transaction; l'existence d'un système de reprise est obligatoire. Le Système R (AST76) utilise cette technique quand le mot "IMMEDIATE" n'est pas spécifié pour les assertions (dans ce cas la validation a lieu pendant l'exécution).

2.2.6 Validation des contraintes dans un SGBDR:

La prise en compte des contraintes dans un SGBDR est un facteur déterminant pour les performances du système, et elle dépend de son architecture. Nous pensons que tout "filtrage" possible doit être réalisé sur la machine qui reçoit directement les requêtes de l'usager. C'est pour

cela que nous proposons la classification suivante pour les contraintes d'intégrité:

a)Contraintes globales: toutes les contraintes qui peuvent être testées globalement (sur la machine globale) avant le lancement sur les machines locales des répercussions des opérations de mise à jour. Par exemple:

i) Le type et le portée des attributs,

ii) Contraintes statiques du type:

- $e \in$ EMPLOYE: e.Salaire > 0
- $e \in$ EMPLOYE: e.Etat-civil='celibataire' $\rightarrow$ e.Epouse='#'
- $e \in$ EMPLOYE: e.Titre='chercheur' $\rightarrow$ e.Salaire < 4000

(Nota: Toutes les valeurs sont supposées connues)

Il n'y a qu'un cas pour lequel toutes les contraintes du SCG peuvent être validées globalement après l'exécution de l'opération de modification: celui de l'obtention de la relation globale suivi de l'exécution de l'opération de modification et des tests des contraintes. Si les tests sont corrects, les modifications définitives sont envoyées aux machines locales correspondantes.

b)Contraintes Locales: toutes les contraintes qui peuvent être validées dans une machine locale avant ou après l'exécution de l'opération de modification. On peut penser également à des contraintes locales activées sous un contrôle global, par exemple: tester l'unicité de la clé.

Ces contraintes sont activées soit par une instruction du type: VERIFIER(Machine Locale,Noms-contraintes), soit par l'incorporation des appels aux contraintes dans l'expression de la correspondance entre l'opération sur le SCG et les opérations sur les SERLs (cf. Chapitre 3). Sur une machine locale les contraintes sont activées soit pendant l'exécution d'une requête (p.ex. test de l'unicité de la clé), soit après l'exécution de la requête (p.ex. cas des contraintes dynamiques).

Le problème des contraintes inter-relationnelles (c.à.d entre deux ou plusieurs relations) est particulièrement intéressant. Dayal (DAY79) propose une technique pour la fragmentation des relations globales en sous-relations indépendantes permettant ainsi d'éliminer (au moins au niveau des dépendances fonctionnelles) les contraintes inter-relationnelles. Cependant, ceci ne peut pas être réalisé dans tous les cas (p.ex. POLYPHEME); donc, il faudra utiliser soit la technique de Badal (BAD79), soit la technique des "actions spontanées" (ESC75,AST76) declenchées sous contrôle de la machine globale ou directement à partir d'une machine locale ("communication machine locale-machine locale").

Dans POLYPHEME (approche ascendante), une approche statique dans la conception du SCG est souhaitable en utilisant des techniques d'analyse comme celles proposées dans

(SEF78,FLO74,DOF79,LEL78) qui permettent d'identifier toutes les opérations de mise à jour autorisées en contrôlant tous leurs effets. Avec MOGADOR (ADI78), la définition des règles d'évolution (cf.§2.1.5) et des fonctions permettent de réaliser un contrôle total (en incorporant tous les aspects sémantiques) des opérations de mise à jour sur un schéma particulier; cependant, il reste un gros travail à réaliser: exprimer la correspondance entre des opérations et des contraintes sur un schéma considéré comme une généralisation (SMI77), en des opérations et des contraintes sur les schémas qui servent à cette généralisation.

La solution adoptée dans MOGADOR pour l'expression des règles d'évolution et des fonctions est la même que celle des "actions spontanées" (programmes) de "Data Semantics" (ABR74) et du modèle de Hammer et Sarin (HAS78). Ces modèles ont été développés pour couvrir des aspects sémantiques que le modèle relationnel ne peut pas exprimer. Donc, lors du passage d'un schéma MOGADOR à un schéma relationnel certains de ces aspects sont trés difficiles à mettre en oeuvre (dans certains cas impossible) et parfois il faudra utiliser un mécanisme "d'actions spontanées" analogue à celui du Système R (AST76) pour pouvoir les contrôler.

2.3 L'importance de la notion de transaction:

Dans un système multi-utilisateur, l'action simultanée des usagers peut introduire des problèmes de cohérence du type perte d'une mise à jour ou lecture de données impropres (p.ex. résultats partiels d'une mise à jour) (GRA78). Ces incohérences ne peuvent être résolues que par la définition d'un mécanisme de synchronisation permettant de "sérialiser" les requêtes: l'effet de l'exécution concurrente des requêtes doit être le même que celui d'une exécution sérielle de ces mêmes requêtes (ESW76). Presque toutes les solutions proposées utilisent la notion de transaction comme l'unité de synchronisation et de reprise. Une transaction est une séquence de requêtes ou d'opérations primitives (p.ex. Lire ou Ecrire) avec les caractéristiques suivantes:

i) l'exécution de la transaction préserve l'intégrité de la base (c.à.d. qu'elle satisfait toutes les contraintes d'intégrité);

ii) si l'exécution de la transaction est arrêtée de façon anormale, ses effets (i.e. ses mises à jour) ne seront pas pris en compte dans la base. Ceci implique que le système doit garder une trace des actions réalisées par la transaction (i.e. mécanisme des journaux).

Par ses caractéristiques, la transaction est l'élément principal pour résoudre les problèmes de concurrence et de reprise. Dans un SGBDR ces deux problèmes sont prioritaires, et la transaction se présente donc comme le meilleur moyen pour leur résolution (LIN79,BER77,WIL79,TRA78,SCO80)

La transaction est considérée en plus comme l'unité logique de travail pour un usager. Elle permet d'alléger l'interface avec les usagers de la base de données pour leur fournir un système "presse-boutons". C'est alors à l'administrateur de l'application de construire et de fournir les transactions; elles sont bâties généralement en utilisant soit le langage de manipulation de la base, soit un interface avec un langage de haut niveau (AST76), ou encore un langage spécialement conçu à ce propos (BAN79,ANB80).

Traiger et al (TRA78), Gray (GRA79) proposent la nécessité d'un modèle transactionnel dans un système réparti, où une transaction est une séquence de requêtes manipulant des entités (représentées par des objets). Ces requêtes sont traduites automatiquement par le système en actions (lire, écrire, verrouiller, déverrouiller, etc) sur les objets. Ces actions sont considérées comme des actions primitives exécutées sur les différents sites du système.

(BAN79) et (ANB80,SCO80) utilisent une approche différente: la coopération des transactions. Chaque site a

un ensemble de transactions de base sur lesquelles sont définies d'autres transactions. Ces transactions font des appels ou activent des transactions qui peuvent être sur différents sites. L'importance de la notion de transaction comme un outil pour la modélisation d'un comportement est beaucoup plus sensible dans cette approche.

Au niveau des contraintes d'intégrité, la notion de transaction a un rôle très important: c'est l'unité de reprise et l'unité de cohérence. La définition que nous avons donnée de transaction considère qu'à la fin de l'exécution, les résultats doivent satisfaire toutes les contraintes d'intégrité. Dans (BAP79) on trouve une analyse très intéressante sur le coût de la prise en compte des contraintes d'intégrité dans une transaction. Dans le Système R (AST76) toutes les contraintes n'ayant pas le mot "IMMEDIATE" sont évaluées pendant la phase de validation ("commitment") de la transaction. La notion "d'actions spontanées" ("triggers ou alerters") (ABR74,AST76,BCL79, HAS78) correspond en quelque sorte à la notion de transaction appelée (ou activée) par une autre transaction. Dans certains systèmes, les transactions sont directement construites dans un langage du système (PL/1,etc); les contraintes d'intégrité sont alors rajoutées sous une forme algorithmique incorporée dans la transaction elle même.

La conception des transactions et ses rapports avec les opérations de mise à jour constitue un aspect très impor-

tant. Dans (DOF79) une approche statique est adoptée et elle est basée sur la notion de type abstrait. Dans (ANB80) une approche incrémentale basée aussi sur la notion de type abstrait (le langage ADA) a été choisie. Dans le système SDD-1 (ROT80) la conception statique des transactions joue un rôle trés important pour les mécanismes de concurrence; les transactions sont décomposées automatiquement en ensembles de lecture et d'écriture qui sont analysés pour déterminer tous les conflits possibles. Ceci permet de classer les transactions en différents groupes pour lesquelles un protocole spécifique de synchronisation est défini.

L'analyse que nous venons de faire, nous permet de mettre en évidence l'importance de la notion de transaction dans un système réparti. Le lecteur peut se rapporter à (LIN79,TRA78,SCO80,ANB80) où des modèles transactionnels pour des systèmes de bases de données réparties sont proposés.

2.4 Conclusions:

Dans ce chapitre nous avons décrit une série de problèmes concernant l'intégrité dans un SGBDR. Nous avons défini la correspondance entre schémas et souligné les différences entre l'approche ascendante et l'approche descendante de conception d'une BDR. Nous avons vu comment la

notion d'usager local introduit des incohérences qui amènent à des limitations dans ce qu'un tel usager est autorisé à faire. Enfin, nous avons présenté l'intégrité sémantique dans un SGBDR et souligné l'importance de la notion de transaction.

Dans le Chapitre 3, nous allons voir une méthode de résolution des problèmes de correspondance entre le schéma conceptuel global et le schéma interne global.

CHAPITRE 3

Courage bon lecteur...

"A coeur vaillant rien

d'impossible"

(Proverbe populaire)

CHAPITRE 3

FRAGMENTATION ET MISE A JOUR

3.1 Fonction de définition et fonction inverse	3.1
3.2 Opérations et formalisme	3.7
3.2.1 Opérations	3.8
3.2.2 Formalisme	3.11
3.2.3 Syntaxe du formalisme	3.16
3.3 Obtention d'une relation globale	3.20
3.4 Fragmentation et mise à jour	3.27
3.4.1 Fragmentation horizontale	3.31
3.4.2 Fragmentation verticale	3.39
3.4.3 Fragmentation par voisinage	3.46
3.4.4 Fragmentation combinée	3.52
3.4.5 Autres types de fragmentation	3.54
3.5 Conclusions	3.56

3.1 Fonction de définition et fonction inverse:

Dans le paragraphe 2.1.2, nous avons défini la correspondance entre deux schémas (A $\xrightarrow{FD}$ B) comme étant exprimée par les fonctions de définition des relations, les transformations des opérations sur une relation en opérations sur les relations (du schéma A) qui la composent, et par les répercussions des contraintes d'intégrité.

La fonction de définition (FD) est généralement une fonction qui préserve les arguments lors de son application (cf. changement admissible, §2.1.2). Si FD est bijective, c'est à dire si c'est un isomorphisme, son inverse (FD^{-1}) est une fonction dite "monovaluée" (1→1). Si FD n'est pas bijective, FD^{-1} n'est pas une fonction (au sens strict du terme). Le comportement de FD^{-1} face aux opérations de modification (insérer, supprimer, modifier) est ambiguë dans la majorité des cas; c'est à dire qu'il existe plus d'une transformation qui peut satisfaire la condition de changement admissible.

Les solutions aux ambiguïtés imposent soit d'interdire toute opération de modification, soit de restreindre la fonction FD à un isomorphisme (CHA75,AST76), soit encore d'établir des conditions théoriques que les opérations de modification doivent satisfaire pour trouver une transformation non ambigue (DAB78a,DAY79,PAP77), soit enfin de considérer une transformation en fonction de la politique

de l'entreprise (SEF78,BAS78,CLE78). Nous avons opté pour analyser la sémantique de fragmentation d'une relation abstraite. Un formalisme pour exprimer les transformations a été développé dans (ADA79); notre but a été de fixer à partir de ce formalisme, une transformation standard dépendant du type de fragmentation. Nous allons compléter ici cette analyse en considérant uniquement la correspondance entre le schéma conceptuel global et le schéma interne global (§2.1.1).

Le schéma conceptuel global contient la description des relations globales qui sont des relations abstraites formées à partir des relations locales d'un SERL (cf§2.1.4.2) Une relation globale peut être considérée comme une généralisation (SMI77) d'une ou plusieurs relations locales (cf exemple §2.1.6): elle est donc une relation fragmentée dont les fragments (relations locales) peuvent être stockés sur des sites différents. Une relation locale est entièrement stockée sur un site du réseau.

Nous allons utiliser l'algèbre relationnelle pour exprimer la définition des relations globales et des relations locales d'une façon symétrique (FD et FD^{-1}): la fonction FD est une expression de l'algèbre relationnelle donnée en termes de relations locales, et FD^{-1} est un ensemble de fonctions (une par fragment) qui expriment les relations locales en termes de la relation globale qu'elles composent.

Exemple:

Considérons une relation globale PROJET(Numéro,Nom, Date-fin,Budget,Ville), formée par deux fragments (relations locales): PROJ1 qui contient les projets menés à Paris et PROJ2 avec les projets menés à Grenoble.

La fonction FD est:

PROJET:= UNION(PROJ1, PROJ2)

La "fonction" FD^{-1} est:

PROJ1 := PROJET: Ville='Paris'

PROJ2 := PROJET: Ville='Grenoble'

Le symbole ":=" signifie "est exprimée par".

La transformation de l'opération d'obtention de la relation globale, n'offre pas de problèmes majeurs parce qu'elle correspond à l'application de la fonction FD, ce qui est toujours possible; il y a un aspect d'optimisation à considérer, cf§1.1.3 et §3.3). La "fonction" FD^{-1} est utilisée pour déterminer partiellement le comportement des opérations de modification car c'est elle qui contient le critère de fragmentation (cf§3.4). En général, FD^{-1} peut être considérée comme un ensemble de contraintes d'intégrité dont une contrainte (prédicat) est associée à chaque fragment de la relation globale. Ce prédicat est appliqué à chaque n-uplet à modifier, insérer ou supprimer pour déterminer le (ou les) fragment (s) sur lequel (lesquels) l'opération doit être réalisée.

Relation globale nom-relation-g

a) Attributs:
-Noms, types et portées des attributs
-Identification des attributs qui composent la clé de la relation.

b) Critère de fragmentation:
('horizontal-disjoint','horizontal-avec-duplications' 'vertical','voisinage-horizontal','arbitraire', 'néant')

c) Fragments:
nom-relation-1 de id.schéma SERL avec prédicat

d) Opérations autorisées:
('toutes','obtention','insertion','suppression', 'modification')

e) Traitement opérations:
('standard','non-standard')
Si 'non-standard' alors spécifier la description des opérations (p.ex. en utilisant le formalisme §3.2)

f) Description opérations:
Pour chaque opération autorisée spécifier sa transformation en opérations sur les fragments.

g) Contraintes d'intégrité:
Pour chaque opération autorisée spécifier les contraintes et les effets de bord associés.

Figure 3.1 Eléments de la définition d'une relation globale

La définition d'une relation globale comporte plusieurs éléments: son nom, les noms, types et portées des attributs, le critère de fragmentation, les caractéristiques des fragments: identification du schéma d'où ils proviennent et les prédicats de fragmentation associés, la description des opérations autorisées, le type de traitement (standard ou non-standard), la description de la transfor-

mation des opérations si le traitement est non-standard, les contraintes d'intégrité auxquelles seront rajoutés les effets de bord occasionnés par une opération particulière, et les contraintes sur la localisation (cf. §2.2.1). La figure 3.1 résume tous ces éléments.

Un traitement standard indique que le système détermine automatiquement la transformation des opérations. Le but de notre analyse dans le paragraphe 3.4 est de déterminer ce traitement standard. Une fragmentation arbitraire conduit à la description de la transformation des opérations (traitement 'non-standard'). La fonction de définition (FD) est implicite dans la description de la transformation de l'opération d'obtention et la "fonction" FD^{-1} est décrite implicitement dans la définition des fragments.

Soit la relation EMPLOYE(Numéro,Nom,Salaire,Dept#) une généralisation des fragments (EMP1 et EMP2); EMP1 qui contient les employés avec un salaire inférieur ou égal à 3000, et EMP2 les employés de salaire supérieur à 2000. Les fragments EMP1 et EMP2 se trouvent sur les schémas SERL-1 et SERL-2 respectivement. Nous allons permettre uniquement l'opération d'obtention par souci de simplification.

```
Relation globale: EMPLOYE

a) Attributs:
    Numéro Clé entier de 0 à 999999
    Nom Chaîne(16)
    Salaire réel de 0. à 15000.
    Dept# Clé de DEPARTEMENT

b) Critère de fragmentation:
    'horizontal-avec-duplications'

c) Fragments:
    EMP1 de SERL-1 avec  Salaire ≤ 3000
    EMP2 de SERL-2 avec  Salaire > 2000

d) Opérations autorisées:  'obtention'

e) Traitement opérations: 'non-standard'

f) Description opérations:
    OBTENIR (EMPLOYE) := (P_ET  (OBTENIR ( EMP1),
                                 OBTENIR ( EMP2))
                          =▶   UNION(EMP1,EMP2);

g) Contraintes d'intégrité:
    SUR INSERTION ET MODIFICATION:
       C1: Salaire > 0
       C2: ANCIEN.Salaire < NOUVEAU.Salaire;
    C3:EFFET DE BORD RELATION DEPARTEMENT:
      SUR INSERTION: Nombre-employés + 1
      SUR SUPPRESSION: Nombre-employés - 1
      SUR MODIFICATION:SI ANCIEN.Dept# = NOUVEAU.Dept#
              ALORS
               POUR ANCIEN.Dept#: Nombre-employés-1
               POUR NOUVEAU.Dept#: Nombre-employés+1
```

Il n'est pas de notre propos de développer un nouveau langage de définition des relations sinon de montrer les différents éléments qui interviennent dans la définition d'une relation globale. Le langage LADORE, proposé par M.Adiba et J.Y.Caleca (ACA78), n'exprime pas toute la sémantique liée à la définition des relations globales, et à notre avis il faudra faire encore des recherches sur ce

sujet (peut être pour un langage orienté vers les types abstraits).

La localisation des relations locales (fragments) n'intervient pas pour la détermination de la transformation des opérations (cf§3.4). Nous considérons cette notion comme indépendante de notre analyse et nous avons utilisé la notation "id_schéma_SERL.nom-relation", pour distinguer des relations locales identifiées par le même nom. Deux tables permettent de décrire cette localisation.

TABLE1

SERL	MACHINE
SERL-1	ML1
SERL-2	ML2

TABLE2

MACHINE	SITE
MG	GRENOBLE
ML1	TOULOUSE
ML3	GRENOBLE

Figure 3.2 La localisation

Une table décrit la relation entre un schéma externe relationnel local (SERL) et la machine sur laquelle il se trouve. La deuxième table, permet d'établir la situation des machines logiques (fig.3.2).

3.2 Opérations et formalisme:

Dans (ADA79) nous avons proposé un formalisme pour définir une mise à jour globale (sur une relation globale) par une expression en termes d'opérations sur les fragments (en indiquant leur parallélisme et leur séquencement). L'expression permet de donner un résultat par des

fonctions booléenes. Nous pensons que ce formalisme permet d'exprimer d'une façon simple la transformation des opérations de mise à jour. Dans le paragraphe §3.4, nous allons l'utiliser pour caractériser par critère de fragmentation et par opération de mise à jour:

a) les opérations sur chaque fragment,
b) la forme d'exécution des opérations: sériel ou en parallèle,
c) le résultat de l'opération globale sous une forme qui peut aider à préserver la cohérence du SCG.

3.2.1 Opérations:

Nous allons considérer quatre opérations de manipulation qui peuvent être appliquées sur les relations globales ou locales:

a) OBTENIR(R): obtenir tous les n-uplets de R, (où R peut être soit un nom de relation soit une expression relationnelle algébrique).

b) INSERER(nom-rel,t): insérer le n-uplet t dans la relation nom-rel.

c) SUPPRIMER(nom-rel,t): supprimer le n-uplet t de la relation spécifiée par nom-rel.

d) MODIFIER(nom-rel,t,t'): remplacer le n-uplet t par le n-uplet t' dans la relation nom-rel. La valeur de

la clé doit rester invariante, c'est à dire: si k est la valeur de la clé de t, alors t' a le même k.

Nous allons associer au résultat effectif d'une opération (ensemble de n-uplets pour l'obtention ou changements dans les occurrences des relations locales pour les opérations de mise à jour) un résultat booléen qui a la valeur:

"VRAI" si l'opération a été bien réalisée

"FAUX" sinon.

La figure 3.3 indique quand le résultat booléen des opérations définies précédemment prend la valeur "VRAI".

Opération	Re	Résultat Booléen
OBTENIR(R)	En	VRAI si En=R
INSERER(R,t)	$\emptyset$	VRAI si $t \in R$
SUPPRIMER(R,t)	$\emptyset$	VRAI si $t \notin R$
MODIFIER(R,t,t')	$\emptyset$	VRAI si $t \notin R \wedge t' \in R$

Re est le résultat effectif de l'opération
R est une relation
En est un ensemble de n-uplets
t est un n-uplet
$\emptyset$ est l'ensemble vide

figure 3.3 Définition des opérations

Nous supposons les opérations de mise à jour réalisées selon la technique des fichiers différentiels: la mise à jour est effectuée sur un fichier temporaire si elle satisfait les contraintes d'intégrité devant être validées lors de l'exécution de l'opération; une valeur logique est envoyée indiquant le succès ou l'échec (contraintes

d'intégrité invalidées). Après cette première phase, une deuxième commence pour la déclaration des modifications réalisées comme "modifications permanentes" (écrire les valeurs sur le fichier de la base). C'est la phase de validation ('commitment') de la mise à jour (si une panne arrive entre les deux phases, les modifications ne seront pas prises en compte). Après la phase de validation, le fichier temporaire est détruit.

Pendant la première phase, nous vérifions essentiellement deux contraintes: la clé de la relation et les prédicats associés à la fragmentation. Pour cela, deux nouvelles opérations sont définies: soit R(K,X) une relation où K est l'ensemble d'attributs formant la clé et X est l'ensemble de tous les autres attributs; alors:

CLE(R,t): pour un n-uplet t (donné par l'opération de mise à jour), projeter K de t. Le résultat est comparé avec les valeurs de la clé dans la relation R pour vérifier s'il y est déjà (la réponse est alors "VRAI") ou s'il n'est pas défini (la réponse est "FAUX").

PRED(R,t): si un ensemble de prédicats (contraintes d'intégrité) est associé à R, appliquer chaque prédicat à t. Pour l'instant, nous allons considérer uniquement le prédicat associé à la fragmentation (R dans ce cas est une relation locale). Le résultat dépend de l'évaluation du prédicat ("VRAI" ou "FAUX").

3.2.2 Formalisme:

Le formalisme, dont la syntaxe est décrite dans §3.2.3, permet d'exprimer le séquencement (conditionnel ou non) et le parallélisme des opérations. Nous allons utiliser la notation suivante: Si OP est une opération, alors OP représente le résultat booléen provenant de l'exécution de cette l'opération; le complément de OP sera noté par NON OP'.

1.Expressions booléennes simples: elles sont exprimées en utilisant des opérateurs booléens: ET,OU,XOU (OU exclusif), et leur négations NET,NOU,NXOU. Par exemple:

OP1 ET OP2 OU OP3

signifie qu'après exécution des trois opérations, l'expression évaluée pour obtenir un résultat booléen ("VRAI" ou "FAUX").

2.Expressions booléennes conditionnelles: elles ont la forme suivante:

Expression1: (Expression2 / Expression3)

qui indique une notion de contrôle associée à l'exécution des expressions et une valeur booléenne associée à l'évaluation de l'expression conditionnelle. Chaque expression rend toujours une valeur booléenne comme résultat en plus de leur action sur les données. Une expression

conditionnelle booléenne est équivalente à l'instruction SI de l'ALGOL:

A:= SI Expression1 ALORS Expression2 SINON Expression3

où l'Expression i représente le résultat booléen associé avec l'exécution de cette expression; l'instruction SI rend un résultat booléen indiquant la bonne ou mauvaise exécution de l'expression 2 ou de l'expression 3. Cette instruction SI peut en fait être considérée comme un SI booléen, et nous voulons le distinguer du SI classique qui indique seulement une notion de contrôle. Quand la partie SINON n'existe pas, l'expression conditionnelle a la forme suivante:

Expression1: (Expression2)

qui signifie:

A:=SI Expression1 ALORS Expression2 SINON FAUX

3.Expressions de parallélisme: le parallélisme d'exécution des opérations (ou expressions) est exprimé à l'aide de six opérateurs: P-OU, P-ET, P-XOU, et les négations P-NOU, P-NET, P-NXOU. Ils sont définis dans la figure 3.4, avec deux expressions (a et b) comme arguments. En général, ces opérateurs sont n-aires et nous utiliserons la notation: P-@ (Exp_i) avec i=1,2,...,n pour désigner P-@ (Exp1,Exp2,...,Expn), qui indique un parallélisme

d'exécution des expressions Exp1,Exp2,...,Expn; l'opérateur renvoie un résultat booléen dépendant de l'évaluation de l'expression:

Exp1 @ Exp2 @ ... @ Expn

où;

- Exp i est le résultat booléen de l'exécution de l'expression i,
- @ est un des opérateurs booléens: OU,ET,XOU,NOU,NET, NXOU,NON,
- une expression est soit une constante "VRAI" ou "FAUX" soit une opération (Obtenir, Insérer, Supprimer, Modifier, Pred, Clé) soit une expression booléenne simple ou conditionnelle.

Opérateur	Signification: a,b sont exécutées en parallèle et résultat booléen sera calculé par:
P-OU(a,b)	a OU b
P-ET(a,b)	a ET b
P-XOU(a,b)	(a ET (NON b)) OU (b ET (NON a))
P-NOU(a,b)	NON(a OU b)
P-NET(a,b)	NON(a ET b)
P-NXOU(a,b)	NON(a XOU b)

Figure 3.4 Opérateurs de parallélisme

4.Expressions conditionnelles de contrôle: elles correspondent plus ou moins à l'instruction SI classique, et nous allons les caractériser par la forme suivante:

OP:= Expression ⇒ Opérateur Relv ;

où:

- OP est une opération: Obtenir, Insérer, Modifier, ou Supprimer,
- Expression: est une expression simple ou conditionnelle booléene, ou une expression de parallélisme. Noter que le souligné représent le résultat booléen rendu lors de l'exécution de l'expression,
- Opérateur Relv: correspond soit à un opérateur relationnel (p.ex: UNION, COMPOSITION, etc) soit à l'opération de validation des mises à jour (p.ex: VALIDER).

La signification est la suivante:

SI Expression ALORS exécuter l'opérateur Relv
SINON signaler une erreur

L'opération OP a comme toute opération un résultat booléen qui correspond à celui donné par l'évaluation de l'expression. Si OP est une opération d'obtention, le résultat effectif est celui donné par l'exécution de l'opérateur Relv. Si OP est une mise à jour, les changements réalisés (résultat effectif d'OP) sont matérialisés (rendus permanents) par l'exécution de l'opération de validation.

Exemples:

a) OBTENIR (Employé):= P-ET (OBTENIR(EMP1),
OBTENIR(EMP2))
=▶ UNION(EMP1,EMP2);

qui signifie: pour obtenir la relation Employé, il faut obtenir tous les n-uplets des fragments EMP1 et EMP2 (opérations qui peuvent être réalisées en parallèle). Si l'obtention des fragments a bien été effectuée (P-ET est évalué "VRAI") alors l'opérateur d'UNION peut être exécuté.

b)(cf§3.4.1 Fragmentation horizontale) Si RG est une relation globale fragmentée horizontalement en forme disjointe, et si t est un n-uplet que l'on veut insérer dans RG, alors:

```
INSERER(RG,t):= P-XOU ( PRED (RLi,t):
                          (CLE(RLi,t):
                            (FAUX/INSERER(RLi,t)))
                          /(CLE(RLi,t):(FAUX))))
                  => VALIDER ;
```

qui signifie:

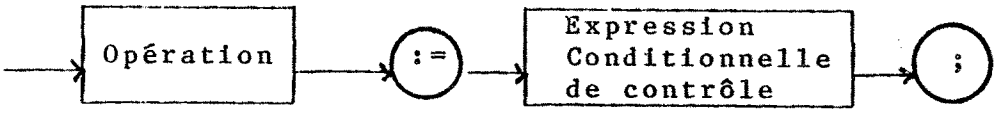
-i) Exécuter en parallèle pour chaque fragment:
SI le prédicat de fragmentation est VRAI ALORS
SI la clé n'existe pas ALORS l'insertion peut s'exécuter
SINON rendre la valeur FAUX
SINON il faut quand même tester l'existence de la clé sur les fragments dont le prédicat est FAUX:
SI la clé existe ALORS rendre la valeur FAUX

-ii) Evaluer l'expression P-XOU en utilisant les résultats booléens du premier pas. Si P-XOU a la valeur "VRAI" alors la validation de la mise à jour est réalisée,et sinon une erreur sera signalée.

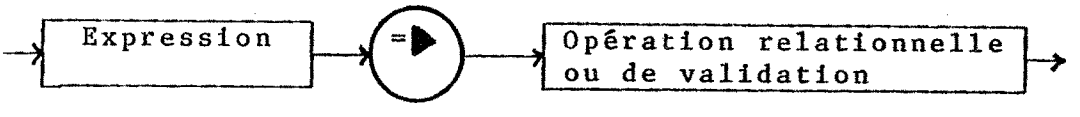
3.2.3 Syntaxe du formalisme:

La syntaxe du formalisme (ADA79) est décrite facilement en utilisant les diagrammes syntaxiques de (WIR76):

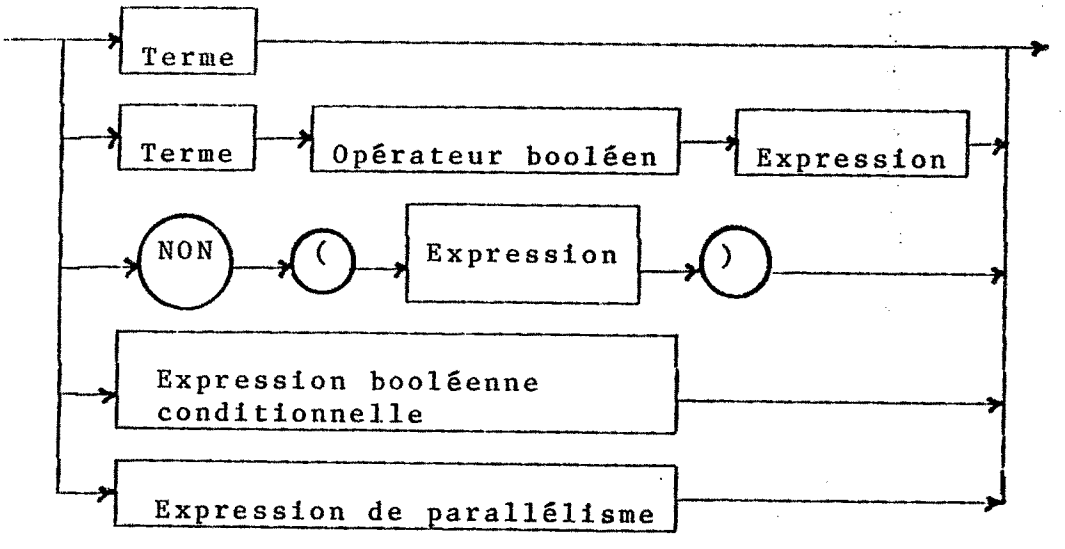
1.Transformation: une opération sur une relation abstraite est transformée en une expression appliquée aux relations composantes:



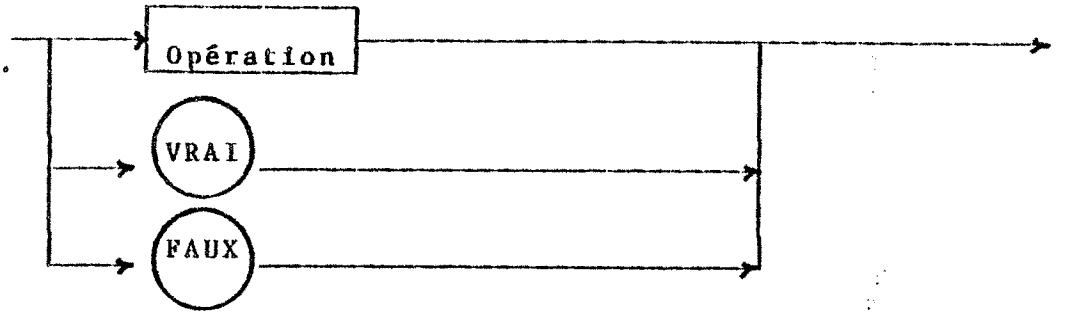
2.Expression conditionnelle de contrôle:



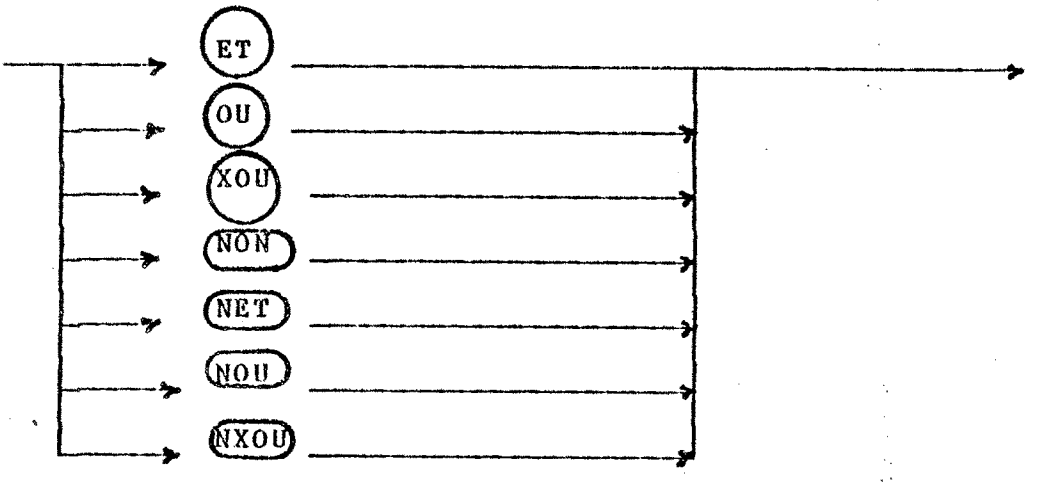
3.Expression



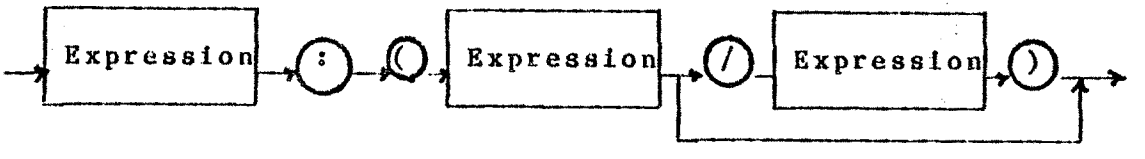
4.Terme



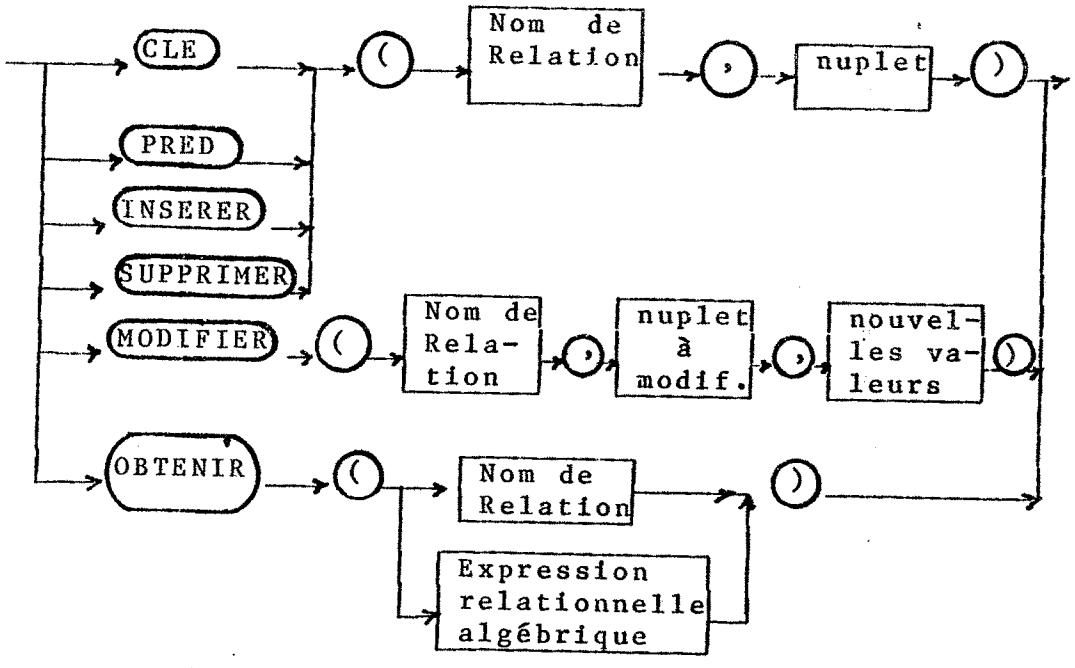
5.Opérateur booléen:



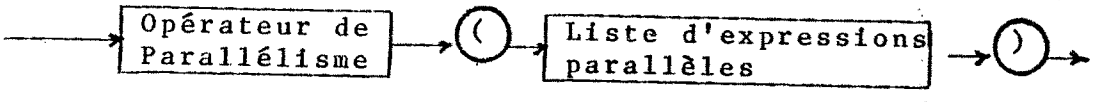
6.Expression conditionnelle booléenne:



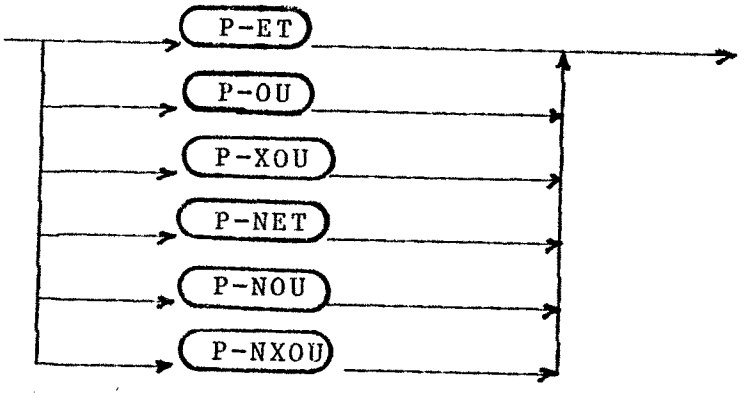
7.Opération:



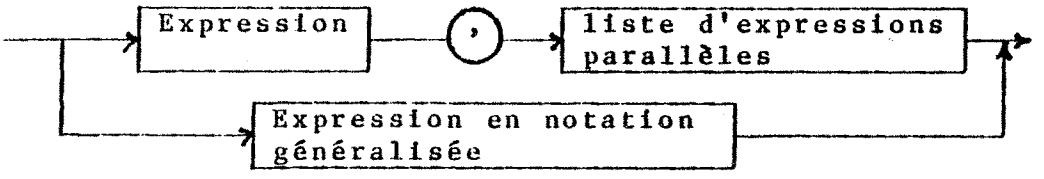
8.Expression de parallélisme:



9.Opérateur de parallélisme:

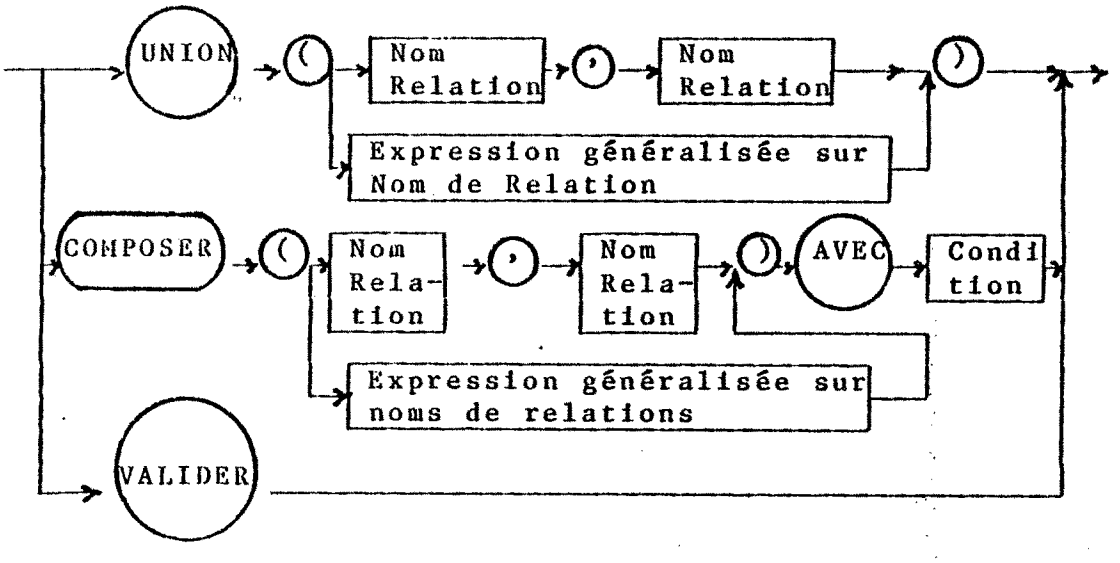


10.Liste d'expression parallèles:



11.Expression en notation généralisée: nous utiliserons exp i avec i=1,2,...,n pour désigner expl,exp2,...,expn

12.Opérateur relationnel ou de validation: nous considérons principalement les opérateurs:



En utilisant ce formalisme nous allons donner dans le paragraphe §3.4 l'expression de la transformation des opérations sur les relations globales en opérations sur les relations locales correspondantes. Dans le paragraphe suivant nous analysons le cas particulier de l'opération d'obtention.

3.3 Obtention d'une relation globale:

Obtenir une relation abstraite consiste essentiellement en l'application de la fonction de définition. Toutefois, une relation globale est composée par des fragments qui peuvent être sur des sites distants; il faudra donc prendre en compte cette répartition et définir une stratégie d'accès optimale aux données. Dans le paragraphe §1.1.3 nous avons souligné quatre étapes dans l'analyse d'une requête d'obtention (une expression relationnelle algébrique) appliquée sur une relation globale.

Ces quatre étapes sont: (voir fig. 3.5)

i) Analyse syntaxique,
ii) Décomposition (prise en compte de la fragmentation),
iii) Réduction des opérateurs algébriques,
iv) Localisation et découpage.

Le but de cette analyse est d'obtenir des sous-requêtes qui s'appliquent sur les relations locales qui composent la relation globale initiale, tout en tenant compte du parallélisme possible lors de leur exécution. L'étape iv) définit une statégie de répartition et d'exécution qui tient compte de l'optimisation de l'accès aux machines locales et de la minimisation du transfert d'informations entre les machines.

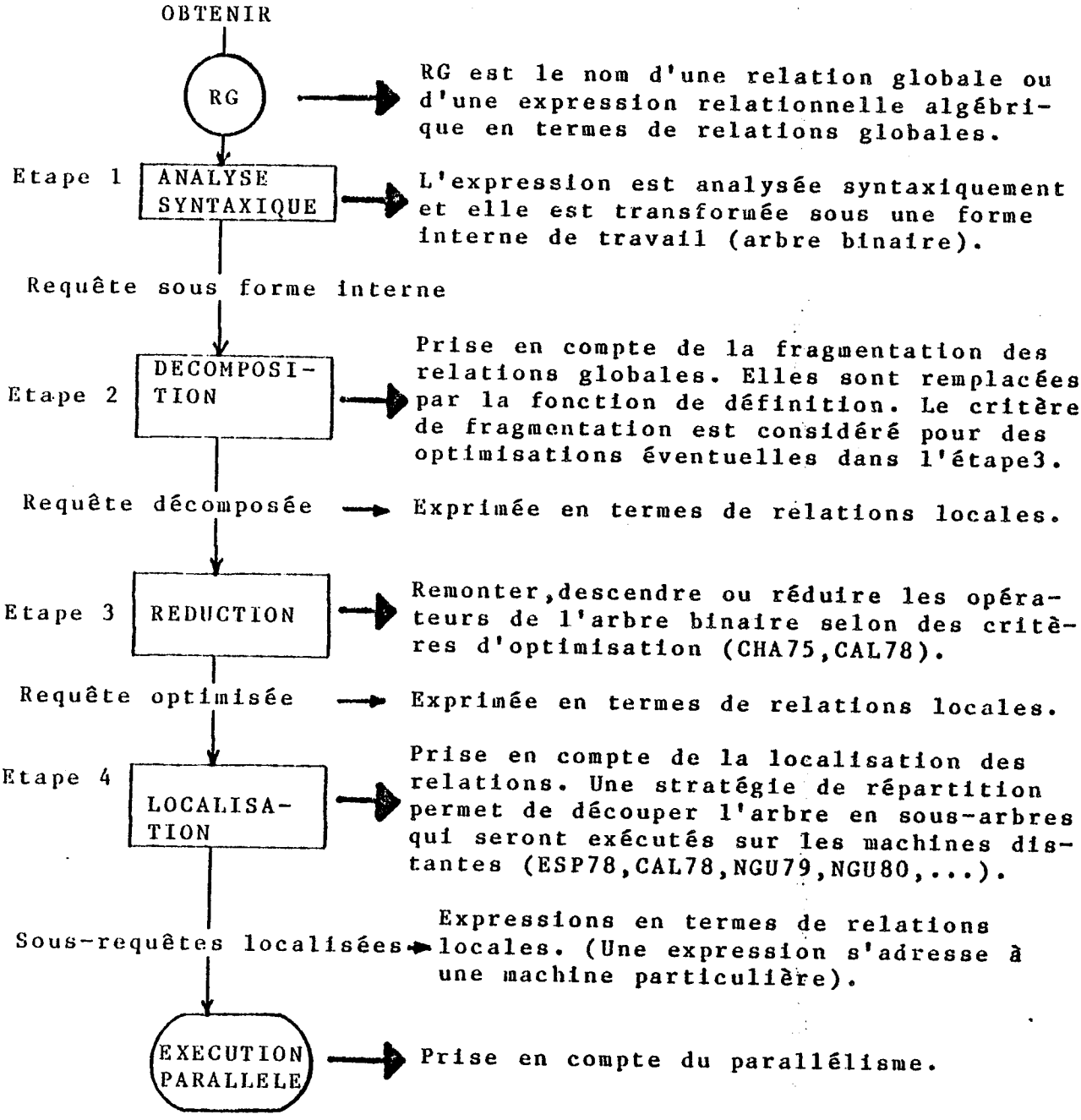


Figure 3.5 Analyse d'une requête d'obtention

Deux approches ont été définies au cours du projet: l'optimisation statique (CAL78) qui définit une stratégie fixe d'exécution et de localisation des traitements, et l'optimisation dynamique (NGU79,NGU80) qui permet de loca-

liser les traitements au fur et à mesure de l'exécution de la requête d'obtention.

Nous allons voir l'analyse d'une requête sur un exemple emprunté de (CAL78). Nous insistons sur la différence méthodologique de l'étape ii) avec les autres présentations (cf. CAL78) de cette approche analytique. Nous faisons en effet une distinction entre la prise en compte de la fonction de définition et la prise en compte des critères de fragmentation (les prédicats de définition des relations composantes, cf.§3.1).

Exemple:

Soit USINE(Numéro,Nom,Ville,Budget,Code) une relation globale définie de la façon suivante: (voir fig. 3.6)

Relation globale USINE

a.Attributs:
  Numéro Clé entier de 0 à 99
  Nom Chaîne(16)
  Ville Chaîne(8)
  Budget Entier de 0 à 9999999
  Code Entier de 0 à 9999

b.Critère de fragmentation:
  'combiné: horizontal et vertical'

c.Fragments:
 Usine de SERL-1 avec (Numéro,Nom,Ville)
                 et Ville='Lyon ^ Grenoble'
 Fabrique de SERL-2 avec Ville='Paris ^ Lille'

d.Opérations autorisées: 'toutes'

e.Traitement des opérations: 'non-standard'

g.Contraintes d'intégrité:
C1:Pour toute opération:
Budget=? ^ Code=? quand Ville='Lyon ^ Grenoble'

(remarque: "?" indique la valeur indéfinie.)

Fin définition relation USINE;

Nous avons délibérément omis la description de la transformation des opérations pour analyser particulièrement l'obtention.

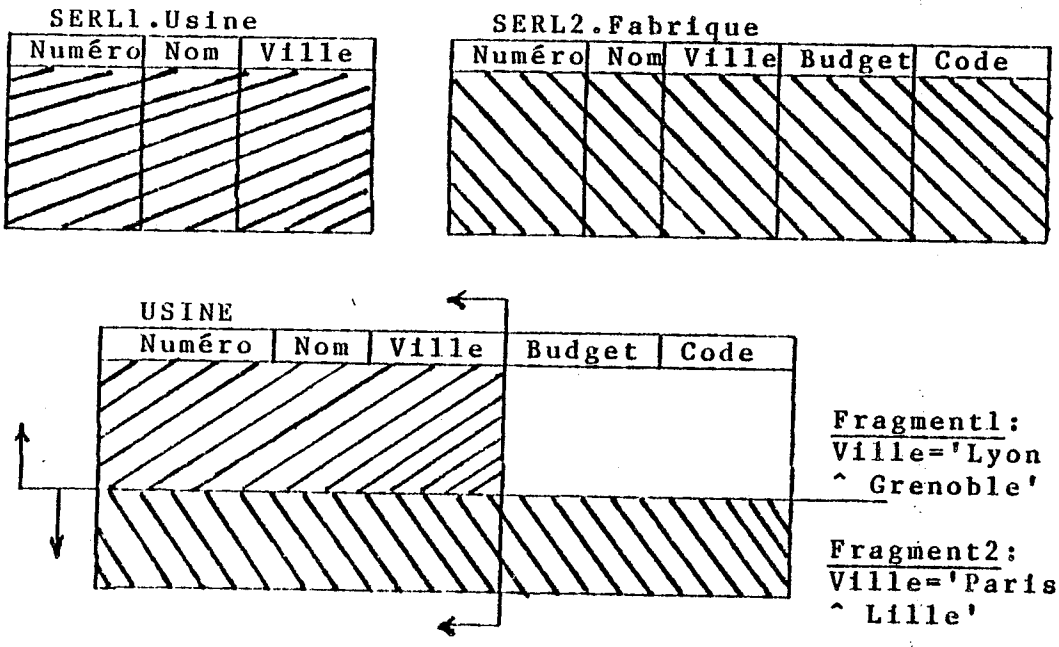


Figure 3.6 La relation globale Usine et ses 2 fragments

La fonction d'obtention est définie par l'expression relationnelle:

USINE:=(Usine U [Numéro,Nom,Ville] Fabrique)

[Numéro U* Numéro]

([Numéro,Budget,Code] Fabrique)

et où U est l'opérateur UNION de deux relations, U* est l'opérateur UNION-COMPOSITION ('Union-Join') qui porte ici sur l'attribut Numéro des deux relations. Cette fonction est présentée plus schématiquement dans la figure 3.7.

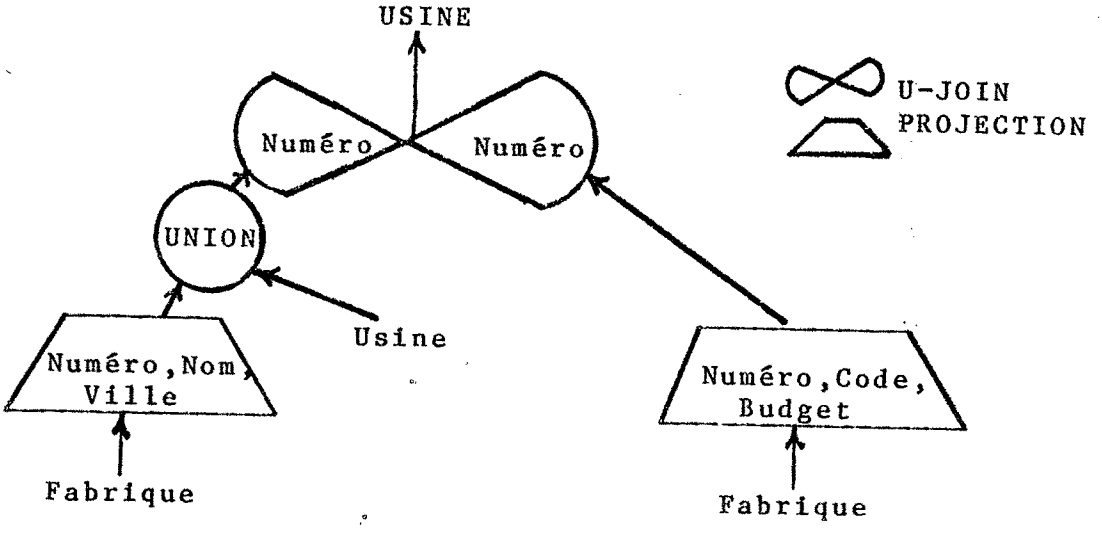


Figure 3.7 Fonction de définition de la relation globale USINE

Un usager global veut savoir le numéro et le budget des usines situées à Paris, et il écrit alors la requête d'obtention suivante:

OBTENIR([Numéro,Budget] USINE: Ville='Paris');

qui est représentée dans la figure 3.8 par un arbre.(C'est l'étape 1)

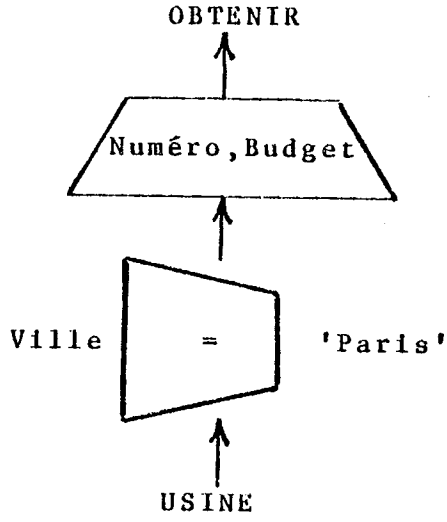


Figure 3.8 Obtenir le numéro et le budget des usines situées à Paris

La deuxième étape prend en compte la définition de la relation USINE. Le nom de la relation est remplacé par la fonction de définition (voir fig. 3.9a), et si l'on ne tient pas compte du critère de fragmentation les étapes 3 (Réduction) et 4 (Localisation et découpage) permettent de réaliser l'optimisation présentée dans la figure 3.9b.

La considération du critère de fragmentation (voir la définition des fragments) permet de réaliser une optimisation plus efficace car plusieurs opérateurs peuvent être éliminées en simplifiant ainsi le découpage en sous-requêtes. La figure 3.10 montre le résultat: la partie a) correspond à la prise en compte de la fonction de définition avec le critère de fragmentation inclus, et la partie b) représente les étapes de réduction, localisation, et découpage. La différence avec l'optimisation de la figure 3.9b est évidente.

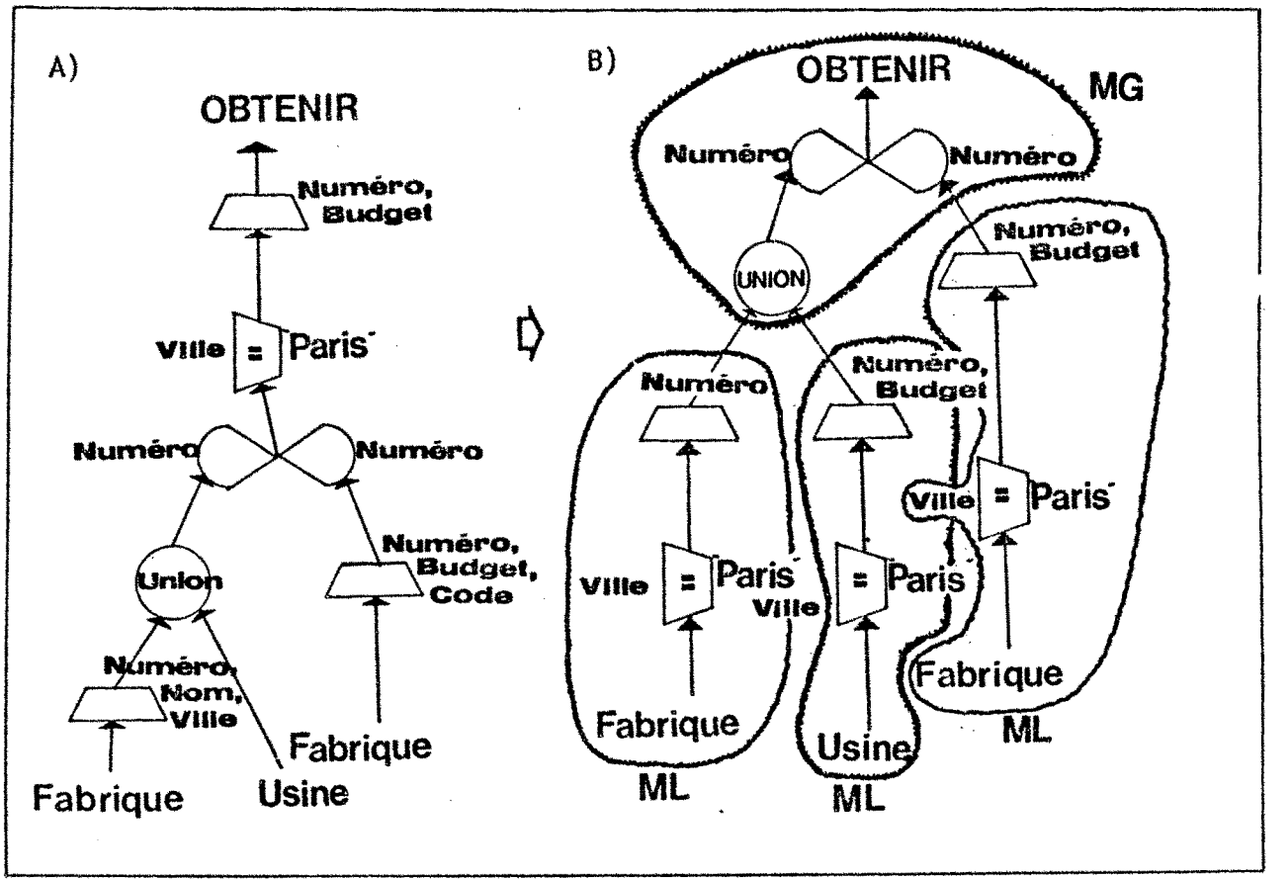


Figure 3.9 Optimisation sans tenir compte du citère de fragmentation

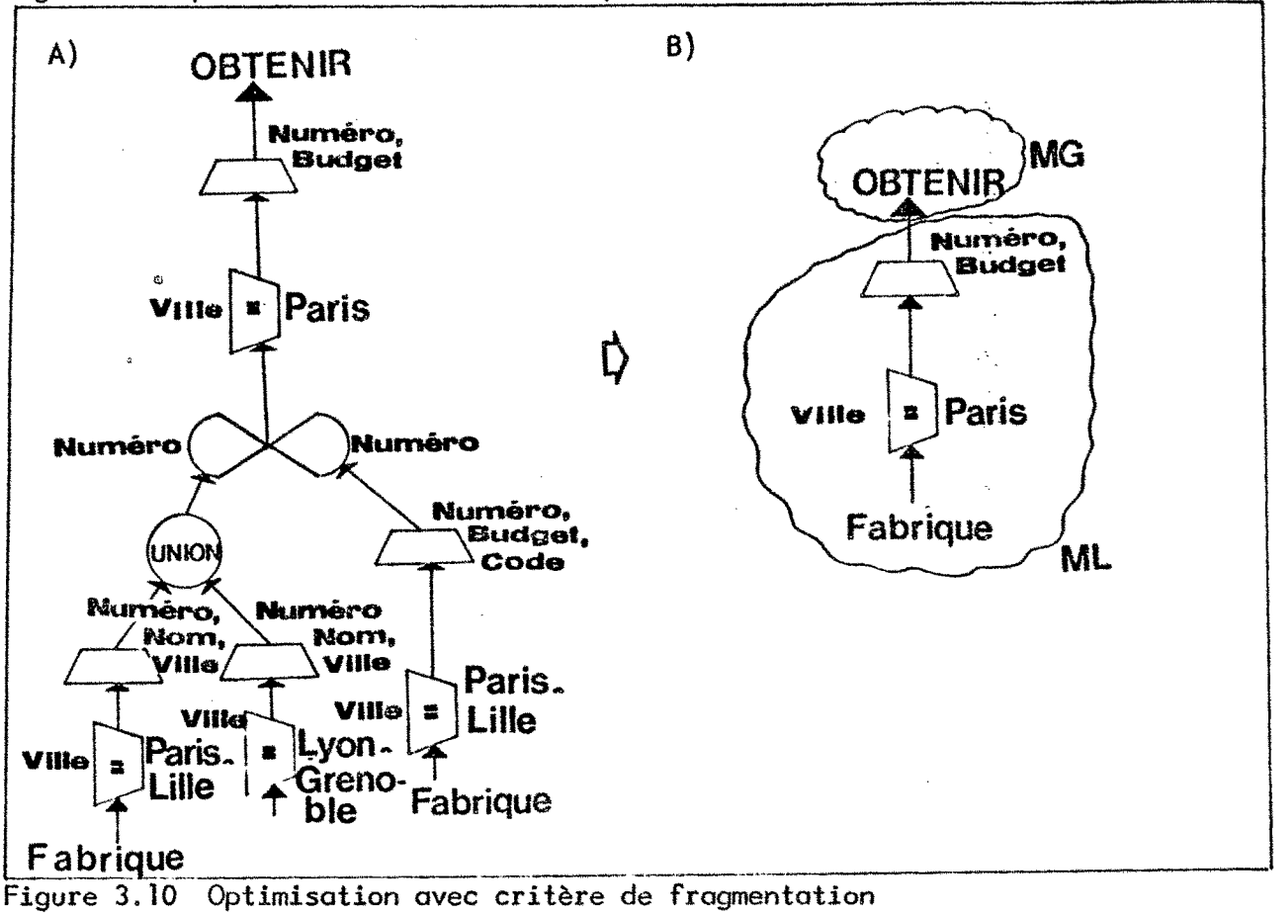


Figure 3.10 Optimisation avec critère de fragmentation

Toutefois, il faut remarquer que ce type d'optimisation n'est pas valable dans tous les cas; cependant, dans cet exemple particulier, il peut être utilisé avec toute requête ayant des restrictions (sur l'attribut Ville) et/ou des projections des attributs.

Le formalisme exprime également le parallélisme d'obtention des informations (n-uplets) des relations locales. Les opérateurs de parallélisme aident à définir une stratégie d'obtention, par exemple: si la relation RG est définie par

RG:=UNION(R11,R12,...,R1n)

alors l'opération OBTENIR(RG) peut être transformée en

OBTENIR(RG):= P-ET(OBTENIR(R11)) =▶ UNION(R1i) ;

ce qui assure que tous les fragments ont été obtenus. Logiquement, si RG est une expression relationnelle elle est soumise aux optimisations précédentes.

3.4 Fragmentation et mise à jour:

En utilisant notre formalisme pour chaque cas de fragmentation et pour chaque opération de mise à jour, nous allons définir la transformation des opérations sur une relation globale en opérations sur les relations locales qui la composent. Dans (ADA79), nous avons appelé cette correspondance: "la propagation des mises à jour". Nous allons aussi considérer les incohérences qui peuvent survenir.

Le formalisme permet de définir le critère de répartition et d'exprimer des contraintes de cohérence (dans §2.2.1, nous les avons appelées des contraintes de répartition) qui doivent être garanties pendant la mise à jour des relations globales. Les opérateurs de parallélisme sont utilisés pour indiquer les opérations exécutables en parallèle. Chaque opérateur permet de définir un point de validation de la cohérence de la mise à jour, interprété de la façon suivante:

P-ET: toutes les opérations lancées en parallèle doivent être "vraies". C'est la première phase du concept classique de validation d'une mise à jour('commitment'). (ESW76,GRA78,LIN79,....)

P-XOU: une et seulement une opération doit être "vraie"

P-OU: une ou plusieurs opérations doivent être "vraies"

P-NET,P-NXOU, et P-NOU sont les négations des opérations précédentes.

Pour renforcer la notion de contrôle associée aux opérateurs de parallélisme, il est possible de considérer une condition d'erreur "armée" quand une opération produit une erreur; mais, ceci est directement dépendant de la mise en oeuvre du formalisme et ne sera pas considéré par la suite dans la définition des transformations.

Associée à la définition des transformations, nous allons donner une solution aux problèmes de migration d'un n-uplet et d'unicité de la clé. Le premier problème (la migration) se présente quand l'opération MODIFIER est appliquée sur une relation fragmentée horizontalement (cf. §3.4.1). Le deuxième, l'unicité de la clé, est directement lié à l'opération INSERER (c.à.d. insertion d'un n-uplet dans une relation globale).

Plusieurs possibilités existent pour donner une solution au problème de l'unicité:

i) Obtenir toute la relation globale sur une machine et tester l'existence de la clé.

ii) Rajouter un identificateur unique du n-uplet ('tid') au champ de la clé. Cet identificateur peut être formé par le numéro du n-uplet concatené avec l'identificateur du schéma (ou de la machine) sur lequel se trouve la relation locale correspondante. Une machine ne peut jamais générer le même identificateur. C'est la solution choisie dans les systèmes SDD-1 (ROT80) et INGRES (EPS78). Dans INGRES, un nuplet est sur un seul site (ou machine) et les n-uplets qui n'appartiennent pas à un site sont considérés comme violation d'une contrainte d'intégrité. Cette solution d'identificateur change la notion de clé donnée par l'usager et fait que la répartition n'est pas transparente aux différents usagers.

iii) Contrôler l'existence de la clé par échange de messages entre machines.

C'est cette dernière solution que nous allons analyser. L'unicité de la clé est considérée comme une contrainte d'intégrité qui peut être validée soit globalement (dans la machine globale) soit localement (dans la machine locale correspondant au frâgment). Dans le premier cas, un message de vérification de la clé est envoyé aux machines locales; la machine globale collecte les résultats des validations et détermine si l'unicité est garantie pour envoyer l'opération d'insertion aux relations locales correspondantes ("répercussion de la mise à jour"). Pendant ces deux phases, un verrouillage des relations locales est nécessaire pour garantir l'unicité vis à vis de la concurrence des différents usagers. Le deuxième cas de validation (locale) consiste à inclure le test de la clé dans la répercussion de la mise à jour sur les relations locales correspondantes. Si la clé n'existe pas l'opération d'insertion peut être effectuée. Cette deuxième possibilité élimine la première phase de validation; des mécanismes de contrôle de la concurrence du type "estampilles" (LAM76,KAN79) sont utilisables pour garantir la cohérence de la relation globale vis à vis des manipulations des autres usagers.

Nous allons, maintenant, étudier les différents cas de fragmentation en essayant de définir un traitement stan-

dard pour les opérations de mise à jour.

3.4.1 Fragmentation horizontale:

Une relation globale est fragmentée horizontalement quand elle est formée par l'union de fragments (relations locales) qui peuvent être considérés comme des restrictions de la relation globale. C'est à dire, si RL1(C,A), RL2(C,A),...,RLn(C,A) sont des relations locales dont (C,A) représente l'ensemble d'attributs de définition avec C=[ensemble d'attributs formant la clé], et A=[le reste des attributs], si RG(C,A) est une relation globale avec comme fonction de définition:

RG:=UNION(RL1,RL2,...,RLn)

et si RLi(C,A) pour i=1,2,...,n est une restriction de la relation RG:

RLi:= RG : $Prédicat_i$

alors, la relation RG est fragmentée horizontalement.

Si $Prédicat_1 \cap Prédicat_2 \cap \ldots \cap Prédicat_n = \emptyset$, alors la relation RG est formée par une fragmentation horizontale disjointe. Si $\bigcap_{i=1}^{n} Prédicat_i \neq \emptyset$, alors RG est fragmentée horizontalement avec des duplications partielles. Si $Prédicat_i = Prédicat_j$ avec $i \neq j$, alors les fragments RL_i et RL_j sont identiques (l'un est une copie de l'autre ou vice versa); dans ce cas là, il faudra définir une copie primaire qui sera utilisée dans la fonction d'obtention. Nous ne considérerons pas ce cas spécial de fragmentation.

Dans une fragmentation horizontale plusieurs incohérences peuvent se produire quand il n'y a pas un contrôle de l'unicité de la clé, de la migration des n-uplets et de la validité des prédicats de fragmentation. Dans §3.4 nous avons considéré deux manières de valider la contrainte d'intégrité "clé de la relation globale": avant ou pendant la répercussion des mises à jour. Considérons une relation globale RG fragmentée horizontalement en plusieurs fragments (RL_i,i=1,2,...,n) tels que les prédicats de fragmentation sont disjoints ($\bigcap_{i=1}^{n}$ $Prédicat_i=\emptyset$); alors, en utilisant le formalisme, la transformation d'une opération d'insertion sur RG en des opérations d'insertion sur les RLi correspondantes (dans ce cas, seulement une) est exprimée par:

i) Si le test de l'unicité de clé est réalisé <u>avant</u> la répercussion de la mise à jour:

```
INSERER(RG,t):=
        P-NOU(CLE(RLi,t)):      *Si la clé n'existe pas
        (P-XOU(PRED(RLi,t):     *insérer dans le fragment
              (INSERER(RLi,t)))) *correspondant
      => VALIDER ; *valider si l'expression est VRAI *
```

L'opérateur P-NOU permet d'assurer la non-existence de la valeur de clé donnée dans le n-uplet t. L'opérateur P-XOU assure que l'insertion se réalise sur un seul des fragments étant donné que dans une fragmentation horizon-

tale disjointe,un seulement des prédicats de fragmentation devra avoir la valeur "VRAI". Si l'évaluation de $PRED(RL_i,t)$ pour tout i=1,2,...,n donne la valeur "FAUX", alors le n-uplet t ne satisfait aucun des prédicats de fragmentation.

ii) Si l'unicité de la clé est testée pendant la répercussion des mises à jour:

```
INSERER(RG,t):=
   P-XOU(PRED(RLi,t):   *Si le prédicat est VRAI alors
         (CLE(RLi,t):    *Si la clé n'existe pas
          (FAUX/INSERER(RLi,t)) *alors insérer le nuplet
                                *sinon rendre "FAUX"
        /(CLE(RLi,t):    *il faut quand même tester l'exis-
             (FAUX))))   *tence de la clé sur les fragments
     => VALIDER;         *dont le prédicat est "FAUX" *
```

Le test des prédicats de fragmentation permet de vérifier les opérations qui se réalisent sur la "fenêtre" (partie visible par l'usager) de la relation globale (définie par la conjonction des prédicats de fragmentation) et implicitement sur les "fenêtres" des relations locales correspondantes (un prédicat de fragmentation est associé à chacune des relations).

Le problème de la migration d'un n-uplet est occasionné par une opération MODIFIER qui change la valeur d'un at-

tribut (ou des attributs) associé à un prédicat de fragmentation; la nouvelle valeur étant associée à un autre prédicat. Ceci implique qu'un n-uplet est déplacé d'un fragment vers un autre. L'opération MODIFIER est alors transformée en une opération de suppression sur le fragment qui contient le n-uplet et en une opération d'insertion sur l'autre fragment.

Exemple:

La relation globale EMPLOYE(Numéro,Nom,Dept,Salaire) se trouve fragmentée horizontalement en trois relations locales:

EMP1 de SERL-1 avec Dept='Ventes'

EMP2 de SERL-2 avec Dept='Production'

EMP3 de SERL-3 avec Dept='Programmation'

et avec l'occurrence suivante:

EMPLOYE

Numéro	Nom	Dept	Salaire	
53730	Dupont	Ventes	4000	
28719	Gauthier	Ventes	2000	EMP1
71171	Perrin	Ventes	3500	
85053	Perrin	Production	6500	
78719	Rey	Production	4700	EMP2
15917	Armand	Programm.	3500	EMP3

L'opération MODIFIER(EMPLOYE,t,t') avec:

t:=(28719,Gauthier,Ventes,2000) et

t':=(28719,Gauthier,Production,3500)

occasionne une modification du n-uplet t de la relation

EMP1 en un n-uplet t' qui doit en fait être déplacé vers la relation EMP2.

Il y a deux façons de contrôler une migration:

i) Obtenir toute la relation globale, la modifier et la décomposer à nouveau en fragments correspondants envoyés vers les machines respectives. Cette solution est un peu coûteuse car le nombre de n-uplets à transmettre à travers le réseau peut être trés grand, et toute la relation globale (c.à.d. les fragments) doit être verrouillée pendant l'opération.

ii) Utiliser les prédicats de fragmentation pour déterminer les relations locales à modifier. Dans ce cas, seule la répercussion sur ces relations est transférée entre les machines globale et locales. La contrainte d'intégrité CLE peut être utilisée pour déterminer l'existence du n-uplet à modifier sur le fragment correspondant.

Nous allons exprimer avec notre formalisme la deuxième forme de contrôle de migration dans le cas général:

```
MODIFIER(RG,t,t'):=
   P-OU(PRED(RLi,t'): *Pour tout frag. qui valide le préd.
      (CLE(RLi,t):    *Si la clé existe dans ce frag.
         (MODIFIER(RLi,t,t') *alors modifier le nuplet
```

/INSERER(RLi,t')) *sinon, alors insérer le nuplet

/ *et pour tout frag qui ne valide pas le préd.

(CLE(RLi,t): *si la clé existe, alors

(SUPPRIMER(RLi,t)))) *supprimer le nuplet *

=▶ VALIDER;

Nous avons considéré deux cas: si t existe dans le fragment, alors l'opération est une modification du fragment qui n'occasionne pas de migration. Si t n'existe pas alors c'est une migration et le n-uplet t' doit être inséré; en conséquence le n-uplet t est supprimé du fragment où il se trouve.

Nous allons donner, maintenant, la définition standard d'une relation globale fragmentée horizontalement. Nous considérons le cas général, c'est à dire: horizontal avec duplications partielles de n-uplets ($\bigcap_{i=1}^{n}$ Prédicat$_i \neq \phi$).

Relation globale RG

a.Attributs

Noms, type et portée des attributs;

identification des attributs qui forment la clé de la relation.

b.Critère de fragmentation: "horizontal avec duplications "

c.Fragments:

RL$_i$ de id.SERL avec Prédicat$_i$

(Pour tout fragment, il y aura un prédicat de

fragmentation associé. Dans ce cas $\bigcap_{i} \text{Prédicat}_i \neq \phi$, où l'intersection indique la zone de duplication des n-uplets)

d.Opérations autorisées: "toutes"

e.Traitement-opérations: "standard"

f.Description-opérations:

1.OBTENIR(RG):=

P-ET(OBTENIR(RL_i)) => UNION(RL_i) ;

Obtenir une relation globale, c'est obtenir les relations locales (fragments) et réaliser l'UNION de toutes ces relations (l'opérateur P-ET le garantit) en éliminant les n-uplets dupliqués.

2.SUPPRIMER(RG,t):=

P-OU(PRED(RL_i,t): *Pour tout fragment dont le
*n-uplet valide le prédicat
(SUPPRIMER(RL_i,t))) *supprimer le n-uplet t
=> VALIDER; *Si P-ET est VRAI alors valider*

3.INSERER(RG,t):= *Pour tout i
P-OU(PRED(RL_i,t): *Si t valide le prédicat
(CLE(RL_i,t): *et la valeur-clé n'existe pas
(FAUX/INSERER(RL_i,t)) *insérer t dans la RL
/ *et si t ne valide pas le prédicat
(CLE(RL_i,t):(FAUX)))) *regarder si la clé
*existe sur ce fragment; si oui,

*l'insertion n'est pas possible.

=▶ VALIDER; *Si P-OU est VRAI alors valider *

4.MODIFIER(RG,t,t'):=

P-OU(*Pour tout fragment

PRED(RL_i,t'): *Si t' valide le prédicat

(CLE(RL_i,t): *et si t existe déjà

(MODIFIER(RL_i,t,t') *alors, modifier t par t'

/(INSERER(RL_i,t') *sinon, alors insérer t'.

/ *Si t ne valide pas le prédicat

(CLE(RL_i,t): *vérifier si t existe sur ce frag

(SUPPRIMER(RL_i,t)))) *Si oui, supprimer t.

=▶ VALIDER; *Si P-OU est VRAI alors valider *

Il faut remarquer que l'opération de modification ne doit pas changer la valeur de la clé. C'est à dire, que la condition CLE(RG,t)=CLE(RG,t') doit être maintenue.

g.Contraintes d'intégrité:

C1:SUR TOUTE OPERATION

P-OU(PRED(RL_i,t))="VRAI"

C2:SUR MODIFICATION

CLE(RG,t)=CLE(RG,t')

C3:SUR INSERTION: l'unicité de la clé

(implicite dans la description de l'opération d'insertion)

Fin définition relation RG;

La description des opérations permet de définir le traitement standard d'une relation globale fragmentée de façon horizontale. Il faut noter que l'opération d'obtention peut être soumise à des optimisations (cf.§3.3).

3.4.2 Fragmentation verticale:

Une relation globale est fragmentée verticalement quand elle est formée par une composition de plusieurs (≥ 2) relations locales. Nous allons considérer que l'opération de composition se réalise sans perte d'information (RID73, cf.§2.1.4.1); ceci peut être facilement obtenu si l'attribut qui contrôle la composition est la clé primaire ou une clé secondaire d'au moins une des relations locales (considérant l'opération de composition comme une opération binaire).

Dans POLYPHEME, nous avons fait une restriction très sévère sur ce type de fragmentation: la composition généralisée de plusieurs relations locales RL1,RL2,...,RLn est possible seulement si toutes les relations ont la clé définie sur le même domaine, l'opération se réalisant sur ce domaine (ADI78,ADA79). Ainsi, les ambiguïtés peuvent être éliminées car la fontion inverse, dans ce cas, est bijective, et donc l'effet des opérations de mise à jour est déterminé avec précision.

Exemple:

Les relations locales PROD1(Numéro,Nom,Qte-stock) et

PROD2(Numéro,Prix-vente) contiennent les produits d'une usine; la relation PROD1 décrit la quantité en stock des produits et la relation PROD2 donne leur prix de vente. Les deux relations sont composées pour former la relation globale PRODUIT(Numéro,Nom,Qte-stock,Prix-vente); cette relation est définie comme suit:

Relation globale PRODUIT

a.Attributs:

Numéro	Clé entier de 0 à 999999
Nom	Chaîne(16)
Qte-stock	Entier de 0 à 99999
Prix-vente	Réel dé 0.00 à 99999.99

b.Critère de fragmentation: "vertical simple"

c.Fragments:

PROD1 de SERL-1 avec (Numéro,Nom,Qte-stock)

PROD2 de SERL-2 avec (Numéro,Prix-vente)

d.Opérations autorisées: "toutes"

e.Traitement opérations: "standard"

f.Description opérations:

1.Obtenir:

```
OBTENIR(PRODUIT):=
        P-ET(OBTENIR(PROD1),OBTENIR(PROD2))
    =>  COMPOSER(PROD1,PROD2) AVEC Numéro=Numéro;
```

Cette expression peut être soumise à des optimisations, p.ex: transfert de la relation PROD1 sur la machine qui

contient la relation PROD2 (ou vice versa) et exécuter l'opération de composition sur cette machine, etc.

2.Insérer:

Dans ce cas le traitement standard est d'insérer le n-uplet t sur les deux relations (si la valeur de la clé n'existe pas). Nous allons considérer que t_i représente la projection de t sur les attributs du $fragment_i$ (définis par le prédicat, cf. partie c. de la définition).

```
INSERER(PRODUIT,t):=
   P-ET(                        *pour tout fragment
     CLE(PROD_i,t):   *si la valeur de la clé existe
       (FAUX          *l'insertion est rejetée
      /INSERER(PROD_i,t_i)) *sinon, alors insérer t_i.
   =>  VALIDER;    *Si P-ET est VRAI alors valider*
```

Fin définition relation PRODUIT;

Pour la suppression et la modification, nous ne donnerons pas leur description; la sémantique est la même que pour l'opération d'insertion: repécuter sur tous les fragments l'opération correspondante.

Dans cet exemple, nous avons considéré l'opérateur de composition-intersection (ou composition naturelle) avec l'hypothèse: [Numéro]PROD1=[Numéro]PROD2; c'est à dire, que les valeurs de la clé sont dupliquées sur les deux relations. Si [Numéro]PROD1 $\neq$ [Numéro]PROD2 la composition

se réalise uniquement sur le sous-ensemble commun aux deux fragments; alors, une opération de modification peut s'avérer trés utile pour compléter l'information qui manque sur PROD1. L'opérateur Union-composition doit être utilisé de la même manière.

Quand la composition se réalise sur la clé d'une des relations (les relations ayant une clé différente), la "fonction inverse" peut ne pas être unique. Donc, des ambiguïtés et des effets de bord se présentent lorsque des opérations de mise à jour sont réalisées sur cette relation globale. Ce problème a été étudié formellement dans (DAB78a,DAB78b,DAY79,BAS78,..). Nous donnons ici des exemples de ces incohérences et ambiguïtés, et l'utilisation du formalisme pour tester l'unicité de la clé.

Exemple:

Nous prenons l'exemple classique des employés et des départements. Considérons que les deux relations locales EMP(Nemp,Nom,Salaire,Ndept) et DEPT(Ndept,Etage) sont composées sur Ndept pour former la relation globale: EMPLOYE(Nemp,Nom,Salaire,Ndept,Etage). Les occurrences des relations sont représentées dans la figure 3.11.

EMP

Nemp	Nom	Salaire	Ndept
53730	Dupond	4000	Systèmes
28719	Toto	2000	Systèmes
71171	Duc	3500	Langages
85053	Abel	6000	Graphiques

DEPT

Ndept	Etage
Systèmes	B2
Langages	D3
Bases de Données	B3

EMPLOYE

Nemp	Nom	Salaire	Ndept	Etage
53730	Dupond	4000	Systèmes	B2
28719	Toto	2000	Systèmes	B2
71171	Duc	3500	Langages	D3

Figure 3.11 Exemple de fragmentation verticale-complexe.

L'opération de composition n'entraine pas de perte d'information (cf.§2.1.4.1) mais toutefois certaines ambiguïtés et des effets de bord se présentent. Considérons, l'insertion du n-uplet t=[43280,Dupont,5700,CAO,C2] sur la relation globale EMPLOYE; il faudra alors définir la transformation de cette opération en opérations équivalentes sur les relation locales EMP et DEPT. Cependant, cette définition implique de bien connaître la sémantique de l'opération et les liens entre les relations locales.

Deux cas sont identifiables:

a) Il existe un lien hiérarchique entre les relations DEPT et EMP, tel que un employé ne peut être inséré que si le département dans lequel il travaille existe déjà. Cette condition est une contrainte d'intégrité qui est vérifiée en regardant l'existence de la valeur de la clé. Ceci est donné par l'expression:

```
INSERER(EMPLOYE,t):=
      CLE(EMP,t_EMP):    *Si le nouvel employé existe déjà
       (FAUX             *l'insertion n'est pas possible
      /CLE(DEPT,t_DEPT): *sinon, si le département existe
       (INSERER(EMP,t_EMP)))  *alors, insérer l'employé
                *et sinon, l'insertion n'est pas possible.
  =>   VALIDER; *Si l'expression est VRAI alors valider *
```

où:

$t=[43280,Dupont,5700,CAO,C2]$

$t_{EMP}=[43280,Dupont,5700,CAO]$

$t_{DEPT}=[CAO,C2]$

b) Si le département n'existe pas, il faudra alors l'insérer:

```
INSERER(EMPLOYE,t):=
   CLE(EMP,t_EMP):       *Si l'employé existe alors
    (FAUX                *l'insertion n'est pas possible;
   /P-ET(                *sinon réaliser en parallèle
     (INSERER(EMP,t_EMP),    *l'insertion de l'employé et
     CLE(DEPT,t_DEPT):   *l'insertion du département s'il
      (INSERER(DEPT,t_DEPT)))              *n'existe pas.
  =>  VALIDER;  *Valider si l'expression est évaluée VRAI*
```

Si t, t_{EMP}, et t_{DEPT} ont les mêmes valeurs que dans le cas a), l'insertion est réalisée sur les relations EMP et DEPT; un nouveau département est alors créé: CAO qui se

trouve au deuxième étage du bâtiment C. Cependant, la relation EMPLOYE présente des inconvenients; prenons le cas de l'insertion du n-uplet t2=[93821,Perrin,4500,Graphiques ,C2]. Le département "Graphiques" ne se trouve pas sur la relation DEPT, il est alors inséré; mais, immédiatement un effet de bord se produit: le n-uplet [83053,Abel,Graphiques,C2] apparait dans la relation globale EMPLOYE. Ceci peut ne pas être trés gênant étant donné que la relation EMPLOYE est une relation abstraite sans occurrence directe; cependant l'insertion viole la condition 1 du paragraphe 2.1.2 maintenant la transparence (pour l'usager) de la définition de cette relation sur d'autres.

Une solution pour éviter ce problème est de considérer l'attribut Ndept dupliqué dans la relation EMP, c'est à dire: [Ndept]EMP=[Ndept]DEPT. Cette condition permet d'assurer que les dépendances fonctionnelles Nemp→ Ndept et Ndept→ Etage sont satisfaites. Ceci est facilement contrôlé dans une approche descendante; la figure 3.11 représente l'approche ascendante pour laquelle ce contrôle doit être réalisé avant la définition de la relation EMPLOYE afin d'éviter des incohérences. L'opérateur d'union-composition est utilisable pour déterminer les valeurs indéfinies pouvant apparaître et occasionner ces incohérences.

Avec les opérations de suppression et de modification surgissent également des ambiguïtés. Supposons que le n-

uplet [28719,Toto,2000,Systèmes,B2] de la relation EMPLOYE soit supprimé; alors, si l'opération de suppression est repércutée sur la relation EMP il faut supprimer tous les employés du département éliminé (p.ex: [53730,Dupond, 4000,Systèmes]). De même, si un employé est changé d'étage, alors on doit modifier tous les employés du département correspondant pour pouvoir maintenir la dépendance Ndept- Etage. Ce cas peut être contrôlé en considérant l'opération comme une violation de cette contrainte d'intégrité.

Avec cet exemple nous pouvons nous rendre compte qu'il existe un certain degré de liberté dans la définition de la correspondance des mises à jour parce que la fonction de définition n'est pas suffisante pour exprimer toute la sémantique des relations et partant, la sémantique des mises à jour. Cet exemple est considéré comme un cas non-standard de fragmentation, et il est plutôt rare dans une approche ascendante comme POLYPHEME, où l'on considère de préférence la généralisation des relations locales qui représentent la même information.

3.4.3 Fragmentation par voisinage:

Comme nous l'avons souligné au paragraphe 2.1.4.2, la notion de répartition liée à la notion de fragmentation permet de distinguer ce type de fragmentation (ADI78). Il

peut être considéré comme un cas spécial de fragmentation horizontale où une relation est fragmentée selon le critère de fragmentation d'une autre relation globale. Ceci a comme but de pouvoir placer les fragments correspondants des deux relations globales sur la même machine locale. Nous supposons que le lien entre les deux relations globales se réalise à travers la clé primaire ou une clé secondaire de la relation qui posséde le critère de fragmentation, cela revient à dire que si la relation RGA est fragmentée par voisinage de la relation RGB, alors la clé (ou une clé secondaire) de cette dernière est contenue dans la relation RGA.

Exemple:

La relation globale USINE(Num-us,Ville,Budget) est fragmentée horizontalement en deux relations locales:

Usines de SERL-1 avec Ville='Lyon ^ Grenoble'

Fabriques de SERL-2 avec Ville='Paris ^ Lille'.

La relation globale EMPLOYE(Nemp,Nom,Salaire,Num-us) est dite fragmentée par voisinage de la relation USINE. Cela signifie que EMPLOYE est fragmentée horizontalement en deux fragments EMP1 et EMP2 correpondants aux employés des Usines situées à "Lyon ^ Grenoble" et "Paris ^ Lille" respectivement. Le lien entre les deux relations est obtenu grâce à la clé de la relation USINE; la condition suivante [Num-us]EMPLOYE=[Num-us]USINE doit être valide.

L'obtention de la relation EMPLOYE ne pose pas de problèmes, elle est simplement l'union des fragments EMP1 et EMP2. Par contre avec les opérations de mise à jour, il faut considérer le critère de fragmentation de la relation USINE pour déterminer le fragment (ou les fragments) sur lequel doit être réalisé la mise à jour. Une façon d'obtenir cette localisation est de réaliser la composition du n-uplet à mettre à jour et de la relation USINE pour appliquer ainsi le critère de fragmentation d'USINE.

Si l'attribut qui fait le lien est la clé de la relation qui contrôle la fragmentation (c'est le cas de notre exemple) on l'utilise pour regarder l'existence du n-uplet sur chaque fragment (de la relation USINE); si le n-uplet existe, l'opération de mise à jour est réalisée sur le fragment correspondant de l'autre relation (EMPLOYE). Nous allons voir ceci avec la définition de la relation EMPLOYE.

Relation globale EMPLOYE

a.Attributs:

Nemp	clé entier de 0 à 99999999
Nom	chaîne(16)
Salaire	réel de 0.00 à 99999.99
Num-us	clé de USINE

b.Critère de fragmentation: "par voisinage de USINE"

c.Fragments:

EMP1 de SERL-1 avec prédicat de Usines

EMP2 de SERL-2 avec prédicat de Fabriques

d.Opérations autorisées: 'toutes'

e.Traitement opérations: 'standard'

f.Description des opérations:

1.Obtenir:

OBTENIR(EMPLOYE):=

P-ET(OBTENIR(EMP_i)) ⇒ UNION(EMP_i);

2.Insérer:

INSERER(EMPLOYE,t):=

P-OU(*pour tout fragment d'USINE

CLE($USINE_i$,t_{Num-us}): *Si la clé d'USINE existe

*sur ce fragment, alors pour le frag.

(CLE(EMP_i,t): *correspondant d'EMPLOYE et

(FAUX/ *seulement si la clé n'existe

INSERER(EMP_i)) *pas, l'insertion est possible.

/FAUX)) *Si la clé n'existe pas sur le frag

*d'USINE, alors l'insertion n'est pas

*possible du tout.

⇒ VALIDER; *Si l'expression est Vrai, alors valider*

Nota:

-$USINE_i$,EMP_i représentent les fragments des relations USINE et EMPLOYE.

-t_{Num-us} est la projection du n-uplet à insérer sur la clé de la relation USINE.

3.Suppression:

```
SUPPRIMER(EMPLOYE,t):=
  P-OU(                    *pour tout fragment d'EMPLOYE
   CLE(EMP_i,t):  *si la valeur de la clé existe déjà,
    (SUPPRIMER(EMP_i)))  *la suppression est possible.
=>  VALIDER; *Si P-OU est évalué VRAI, alors valider*
```

4.Modification:

Cette opération doit tenir compte de la migration d'un n-uplet étant donné que la relation EMPLOYE est fragmentée horizontalement selon le critère de fragmentation d'USINE. Il faut alors vérifier que la valeur de l'attribut Num-us existe sur USINE pour réaliser la modification. En vérifiant la clé des deux relations, la migration est facilement contrôlée.

```
MODIFIER(EMPLOYE,t,t'):=
 P-OU(                        *pour tout fragment d'EMPLOYE
  CLE(USINE_i,t'Num-us) *si la clé existe sur USINE_i
   (CLE(EMP_i,t):          * et si t existe sur EMP_i alors
    (MODIFIER(EMP_i,t,t')   *modifier t par t'
    /INSERER(EMP_i,t'))     *sinon, il faut insérer t'
                           *(il y a alors une migration).
  /            *Si t'Num-us n'existe pas sur USINE_i il
   CLE(EMP_i,t):  *faut vérifier la migration eventuelle
    (SUPPRIMER(EMP_i,t)      *de t. Si t existe il faut
    /FAUX))                *le supprimmer et sinon ni t'
```

*ni t n'existent sur ce fragment.

=▶ VALIDER; *Si P-OU est évalué VRAI alors valider *

Nota:

La condition suivante doit être satisfaite:

CLE(EMPLOYE,t)=CLE(EMPLOYE,t')

Noter que si $CLE(USINE,t_{Num-us})=CLE(Usine,t'_{Num-us})$ alors il n'y a pas de migration de n-uplet.

g.Contraintes d'intégrité:

C1:SUR INSERTION ET MODIFICATION

$P\_OU(CLE(USINE,t_{Num-us}))$="VRAI"

C'est à dire, si l'usine existe alors on localise l'opération sur le fragment correspondant de la relation EMPLOYE. Cette contrainte est implicite dans la description des opérations.

C2:SUR MODIFICATION:

CLE(EMPLOYE,t)=CLE(EMPLOYE,t')

C3:SUR INSERTION:

Unicité de la clé. (Contrainte implicite dans la description de l'opération)

Fin définition relation EMPLOYE;

Cette définition est généralisable à toutes les relations qui ont les mêmes caractéristiques de fragmentation que la relation EMPLOYE. Cependant, il faut remarquer que la fragmentation par voisinage peut mener à des

incohérences si l'on ne tient pas compte qu'une modification de la relation qui contrôle la fragmentation (p.ex: la relation USINE) conduit parfois à des modifications sur l'autre relation (p.ex: la relation EMPLOYE). Ces effets de bord sont occasionnés par des opérations de suppression ou de modification.

Ainsi, dans notre exemple, la suppression d'une usine conduit à la suppression de tous les employés qui travaillent dans cette usine; une modification de la relation USINE, peut occasionner la migration du n-uplet vers un autre fragment, et donc tous les employés de cette usine doivent être déplacés vers le fragment analogue. Ces deux effets de bord doivent être contrôlés par des contraintes d'intégrité définies sur les opérations de modification et de suppression de la relation USINE.

3.4.4 Fragmentation combinée:

Une relation globale a ce type de fragmentation quand elle est formée par une combinaison de fragmentations horizontales et verticales. Par exemple, la relation

EMPLOYE(Numéro,Nom,Age,Salaire,Dept)

est formée par trois fragments:

EMP1:=[Numéro,Nom,Age](EMPLOYE:Age $\leq$ 26)

EMP2:=[Numéro,Nom,Age](EMPLOYE:Age $>$ 26)

EMP3:=[Numéro,Salaire,Dept] EMPLOYE

et sa fonction de définition est donnée par l'expression:

EMPLOYE:= (EMP1 U EMP2) * EMP3

où * est l'opérateur de composition sur l'attribut Numéro.

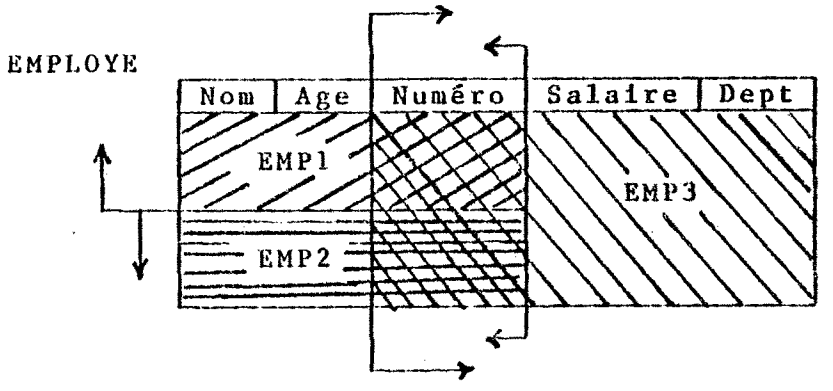


Figure 3.12 Fragmentation combinée

Une solution simple pour définir un traitement standard des opérations de mise à jour est de considérer la relation globale fragmentée en deux fragments dont l'un constitue une autre relation globale pouvant à son tour être fragmentée, et ainsi de suite. Alors, dans notre exemple, la relation EMPLOYE peut être considérée comme une relation fragmentée verticalement en deux fragments: EMPLOYE2 et EMP3. Le fragment EMPLOYE2 est à sont tour une relation globale fragmentée horizontalement en deux relations locales EMP1 et EMP2. Le fragment EMP3 est une relation locale.

Un autre cas de fragmentation combinée correspond à l'inverse du cas précédent: une relation globale est considérée fragmentée horizontalement en deux relations dont l'une est une relation globale fragmentée verticalement.

Cette réduction permet d'appliquer les transformations standard définies pour les cas de fragmentation horizontaux et verticaux.

3.4.5 Autres types de fragmentation:

Dans ce paragraphe nous considérons des relations globales formées par les opérateurs de projection et de restriction appliqués sur une relation locale, ou par des combinaisons de ces opérateurs (FUS77).

Soit RG(C,X) une relation globale définie par une projection de la relation locale RL(C,Y), où C est l'ensemble d'attributs clé et X,Y sont des ensembles d'attributs non-clé tels que X $\subset$ Y. La transformation d'une opération d'insertion doit alors tenir compte du fait que la valeur pour l'ensemble Y-X (ensemble d'attributs de RL qui ne sont pas dans RG) est la valeur "indéfinie" ("?").

Exemple:

Si EMP_NOM(Numéro,Nom) est une relation globale définie sur la relation locale EMPLOYE(Numéro,Nom,Age,Salaire), alors l'opération INSERER(EMP_NOM,[532801,Dupont]) est transformée en INSERER(EMPLOYE,[532801,Dupont,?,?]). Les opérations de suppression et de modification sont aussi transformées (dans ce cas) en opérations équivalentes uniques et donc sans ambiguïtés. Ceci vient du fait qu'il existe une dépendance fonctionnelle simple (monovaluée) entre les attributs Numéro et Nom.

Si la relation RL a une clé composée qui n'est pas complètement conservée dans la projection, alors un problème se présente avec la transformation des opérations de suppression et de modification réalisées sur la relation RG: il peut y avoir plus d'un n-uplet à modifier ou à supprimer. Nous éliminons ce cas ambigu en considérant que la clé de la relation locale doit être conservée dans la relation globale.

La mise à jour sur une relation définie par l'opérateur de restriction ne présente pas de problèmes majeurs: l'opération de mise à jour est erronnée quand elle est en dehors de la restriction. Ceci implique que la valeur de l'attribut (ou des attributs) sur lequel porte la restriction doive être complètement spécifiée.

La mise à jour sur s relations globales définies par une combinaison des opérateurs de projection et de restriction sur une relation locale, peut être réalisée sans ambiguïtés. La restriction est utilisée pour compléter des valeurs ou pour rejeter des opérations en dehors de la définition; elle joue donc le rôle d'une contrainte d'intégrité.

Pour finir, nous considérons qu'une relation globale ne peut pas être définie par l'opérateur de projection appliqué sur une composition de deux relations en éliminant un ou les attributs de la condition que contrôle la composi-

tion; en effet la transformation des opérations de mise à jour est ensuite impossible.

3.5 Conclusions:

Dans ce Chapitre nous avons étudié en détail les problèmes de la mise à jour dans un système de bases de données réparties. Nous avons donné des éléments pour la définition correcte des relations globales de la vue de la coopération. Un formalisme a été présenté et utilisé pour décrire la transformation des opérations de mise à jour. Ce formalisme tient compte d'une certaine cohérence sémantique liée à ces opérations; cette information est utilisable dans la phase de validation des mises à jour.

Nous avons étudié les différents types de fragmentation d'une relation globale et nous avons montré comment, dans certains cas, on peut définir un traitement standard des opérations de mise à jour. Ces cas introduisent des limitations quelque fois très sévères dans la formation de relations globales; à notre avis, cependant, cet ensemble de cas peut être considéré comme suffisant pour réaliser une coopération cohérente des différents schémas locaux.

Toutefois, notre approche est générale et le formalisme est utilisable pour définir explicitement la transformation des opérations.

Ce problème de la mise à jour dans des relations abstraites est aussi étudié dans: (CHA75,STO75,OSM79,DAB78b, DAY79,BAS78,CLE78,FUS78). Cependant, le problème de la fragmentation dans un système de bases de données répartie n'a semble-t-il été bien analysé d'autre part que par (DAB78b,DAY79).

CHAPITRE 4

"Le dernier acte est sanglant, quelque belle que soit la comédie en tout le reste: on jette enfin de la terre sur la tête, et en voilà pour jamais"

Pensées

Blaise Pascal

CHAPITRE 4

LA MAQUETTE DE POLYPHEME

4.1 Introduction 4.1

4.2 LAMB: Logiciel d'accés et manipulation de blocs 4.2

4.3 Architecture de la maquette POLYPHEME 4.7

4.3.1 Le système relationnel URANUS et la décomposition de requêtes 4.9

4.3.2 Le moniteur d'exécution répartie: MER 4.13

4.3.3 Le contrôleur d'exécution répartie 4.17

4.3.4 La base de données répartie (BDR) 4.24

4.3.5 Scénario d'exécution de la maquette. Exemple 4.26

4.4 Extensions à la maquette 4.33

4.5 Conclusions 4.35

4.1 Introduction

Le projet POLYPHEME a eu pour objectif la conception et la réalisation d'un système de coopération de bases de données hétérogènes réparties sur un réseau d'ordinateurs. Nous avons utilisé une méthodologie de conception ascendante et la maquette réalisée- même si elle ne contient pas de mécanismes pour le contrôle de la concurrence, la reprise après panne, ou l'intégrité et la confidentialité- doit être considérée comme un résumé du travail effectué dans le projet POLYPHEME.

Dans le chapitre 1, nous avons analysé de façon très sommaire chacun des problèmes qui se présentent lors d'une coopération de bases de données hétérogènes. Comme nous l'avons déjà souligné (cf §1.1.1) l'homogénéisation des SGBD a été atteinte en les assimilant à des machines relationnelles logiques (ACE78). Ceci peut être obtenu de deux manières différentes. Premièrement, plusieurs programmes d'application locaux sont rajoutés à la base de données pour construire ainsi un interface relationnel; chaque relation est manipulée au travers de ces programmes qui tirent avantage de l'organisation et des chemins d'accés aux données. Chaque opérateur relationnel correspond alors à un programme spécifique activé sur demande. Cette solution est décrite dans (ADI78,CAL78,EUZ79,PIS78,CAS77), et nous en avons développé une partie: le Logiciel d'Accés et Ma-

nipulation de Blocs (LAMB) qui fournit des fonctions de base de stockage et d'accés à des données relationnelles (cf. §4.2).

La deuxième possibilité, qui a été choisie dans la maquette, utilise des mécanismes développés pour le projet URANUS (NGU77,NGU78). Chaque base de données coopérante (une base SOCRATE) est traduite de la forme réseau vers une forme relationnelle et c'est seulement cette copie qui est manipulée. Avec URANUS, on dispose d'un langage basé sur l'algèbre relationnelle, et les requêtes sont exprimées par la combinaison des opérateurs (JOIN, SELECT, PROJECT,...) et des noms de relations, de manière non procédurale.

Nous allons décrire dans les paragraphes suivants l'architecture de la maquette qui intègre cette deuxième possibilité et qui a été expérimentée depuis Juin 1979 sur le réseau CYCLADES (AND79,AAD79,ADI80a,ADI80b,DEA80,DAN80) Une démonstration de cette maquette a été enregistrée sur une bande vidéo.

4.2 LAMB: Logiciel d'accés et manipulation de blocs

LAMB (AND77a,AND77b,AND78) est un système qui permet la création, la manipulation et l'accés à des tableaux d'information. Un tableau est un ensemble de n-uplets, et chaque n-uplet est composé de champs ayant une certaine longueur et contenant une valeur.

Chaque tableau correspond à un type qui définit des caractéristiques structurelles: nombre de colonnes (degré), longueur de chaque champ, champs clé, etc.

Les tableaux sont identifiés par un identificateur de tableau (idt) et par un nom externe donné par l'utilisateur lors de sa création. Un n-uplet est repéré par un identificateur de n-uplet (idu). L'utilisateur se sert de ces identificateurs au niveau de l'interface d'utilisation de LAMB.

Le tableau est l'unité logique que manipule l'utilisateur. Physiquement, l'unité de travail est un bloc; un tableau peut être réparti sur plusieurs blocs, mais ceci est transparent à l'utilisateur.

L'interface d'utilisation de LAMB est constitué par un ensemble de primitives appelables d'un programme PL/1 ainsi que par un ensemble de règles concernant les passages de paramètres et des codes d'erreur.

On trouve dans LAMB un ensemble d'opérations sur chaque objet. Un objet est soit un type, soit un tableau, soit un n-uplet. Les opérations suivantes assurent la gestion des objets:

1. Sur les types:
   - définir un type de tableau (DTYP)
   - supprimer un type (et tous les tableaux de ce type) (TTYP)

2.Sur les tableaux:

-définir un tableau d'un type donné (DTAB). La place est réservée pour le tableau et un idt lui est affecté.

-supprimer l'occurrence d'un tableau (TTAB).

-retrouver l'identificateur d'un tableau (LIDT)

-connaître les caractéristiques d'un tableau (LCTRL)

3.Sur les n-uplets:

-créer un n-uplet dans un tableau (CNPL)

-supprimer un n-uplet (TNPL)

-lire les valeurs d'un n-uplet (LNPL)

-obtenir le n-uplet suivant (SNPL)

-rechercher un n-uplet par la valeur du champ clé (RCNPL)

-rechercher un n-uplet par sa position (RDNPL)

-modifier les valeurs d'un n-uplet (MNPL);

Le contrôle de la création, suppression, et modification des tableaux est assuré par trois tableaux gérés et accédés exclusivement par LAMB et donc transparents à l'utilisateur:

-le tableau maître (catalogue de tous le tableaux existants)

-le tableau des types (contenant tous les types de tableau possibles)

-le tableau des caractéristiques des n-uplets d'un type de tableau.

LAMB utilise un espace virtuel statique (type SOCRATE) pour mettre en oeuvre les tableaux. La gestion physique (au niveau fichier) des informations utilise les primitives d'accés "espace virtuel" d'URANUS (NGU77). Le lecteur trouvera une description plus complète de la structure de LAMB dans (AND78).

LAMB a été conçu pour effectuer dans la machine globale: (ADI78)

-le stockage et l'accés aux catalogues des vues locales et de la vue globale,

-le stockage et l'accés aux graphes provenant de la décomposition d'une requête,

-le stockage et l'accés aux informations provenant des bases locales au cours de l'exécution d'une requête.

Une première version de LAMB est opérationnelle depuis Juin 1978, et un ensemble d'opérateurs de l'algèbre relationnelle a été rajouté (PAI78). Bien que, l'architecture de POLYPHEME (dans laquelle LAMB aurait joué un rôle très important) n'aie pas été mise en oeuvre dans sa totalité, nous avons pu constater l'intérêt tel que LAMB pour la construction modulaire de systèmes relationnels. En fait, LAMB peut être considéré comme une mémoire relationnelle (du type XRM du Système R ,(AST76)) avec des primitives de très bas niveau (utilisation des idt, et idu) utilisables pour développer des langages relationnels du type algébri-

que ou prédicatif, ou un langage qui considère les reltions comme des types de données abstraites, ou encore pour définir directement des transactions, etc. Cette idée a été reprise dans le projet MICROBE (MIC80), où F.Fernandez développe une mémoire relationnelle beaucoup plus perfectionnée (FER80) qui servira pour l'expérimentation et le développement des différents langages relationnels. L'expérience que nous avons tirée de POLYPHEME, nous permet de confirmer la nécessité d'un tel outil pour le développement de futures applications. On peut même envisager de réaliser LAMB en materiel spécialisé (micro-processeurs etc).

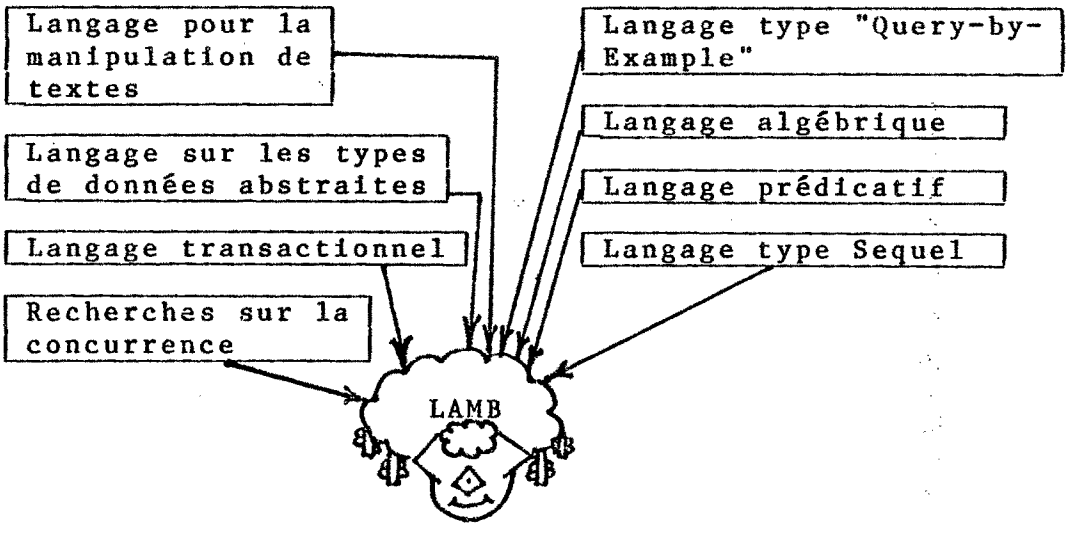


Figure 4.1 LAMB un outil de recherche pour la construction de systèmes relationnels modulaires.

4.3 Architecture de la maquette POLYPHEME

La maquette POLYPHEME, qui a été mise en oeuvre sur le réseau CYCLADES (AAL80a,AAL80b), est dessinée comme un réseau de machines relationnelles logiques (ACE78). Dans le Chapitre 1, nous avons discuté cette architecture logique composée de machines locales construites sur chaque base locale pour obtenir un comportement homogène, et de machines Globales qui permettent aux usagers de manipuler la base de données réparties. L'usager travaille en fait comme s'il était sur un SGBD centralisé; la localisation des données lui est transparente. Une base locale peut être intégrée dans la Machine Globale elle même; c'est à dire qu'une "machine locale" se trouve dans la machine globale.

Un usager de POLYPHEME soumet des requêtes simples (d'obtention ou de mise à jour) ou des séquences de requêtes à une machine globale. Chaque requête est alors décomposée en sous-requêtes qui sont envoyées aux machines locales correspondantes pour leur évaluation (cf. §4.3.3, §4.3.5).

La figure 4.2 montre l'architecture de la maquette. Chaque machine (globale ou locale) se compose de trois éléments:

1) Un système relationnel, URANUS (NGU77), qui stocke et obtient les informations locales.

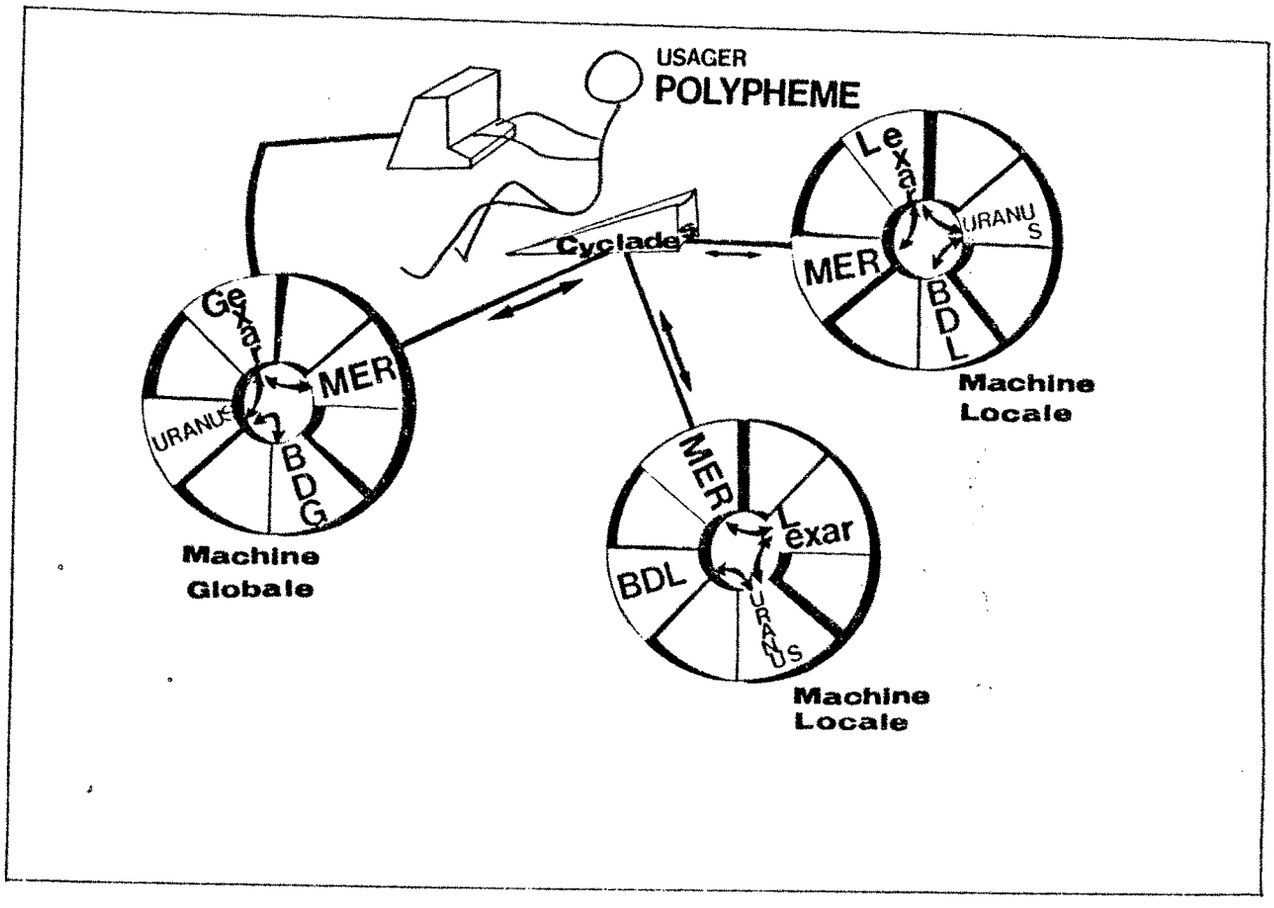


Figure 4.2 Eléments de la maquette POLYPHEME

2) Un moniteur d'exécution répartie, MER (DEA80), qui assure la communication entre les machines distantes ainsi que l'activation et la synchronisation de procédures PL/1 à distance;

3) Un contrôleur d'exécution répartie construit en utilisant les services de MER, et qui permet de synchroniser l'exécution répartie des requêtes (AAD79). Il est composé d'une partie (appelée GEXAR) sur chaque machine globale et d'une autre partie (appelée LEXAR) sur chaque machine locale. Il permet l'intégration d'URANUS et de MER.

Nous allons voir en détail chacun de ces éléments.

4.3.1 Le système relationnel URANUS et la décomposition de requêtes

Le système URANUS développé par Nguyen Gia Toan (NGU77) a été adapté pour considérer la distribution des relations, et il a été découpé en deux parties pouvant être activées indépendamment:

a) La Décomposition: qui analyse et transforme une requête en arborescence. L'arbre généré est ensuite découpé en sous-arbres qui peuvent être interprétés sur des machines locales.

b) L'Interprétation: qui interprète un arbre.

URANUS présente les caractéristiques suivantes:

- Définition et manipulation des données au travers d'un langage algébrique non-procédural.
- Gestion de "transactions": une "transaction" est considérée comme une séquence de requêtes qui peuvent avoir des paramètres formels remplacés à l'exécution par des valeurs réelles. La notion de transaction ne correspond pas à la définition que nous avons donné dans §2.3, parce que URANUS est mono-usager et ne dispose pas d'un mécanisme pour le contrôle de la concurrence et de la reprise.

- Génération d'un arbre.
- Localisation et décomposition d'un arbre en sous-requêtes.
- Interprétation d'arbres.

Deux versions de l'algorithme de décomposition on été dévelopées pour URANUS. La première considère une localisation statique des différents noeuds d'un arbre; chaque feuille de l'arbre (une relation) est localisée en utilisant les informations des catalogues. Ensuite, l'algorithme remonte l'arbre pour localiser les différents noeuds opérateurs. L'idée est de caractériser des sous-arbres maximaux qui peuvent être interprétés sur chaque machine locale. Ceci est réalisé en suivant les régles ci dessous:

i) Tout opérateur unaire (PROJECT, SELECT...) peut être interprété sur la même machine que son argument.

ii) Tout opérateur binaire (JOIN,UNION,...) peut être localisé si les deux arguments ont été localisés sur la même machine.

iii) Tout opérateur binaire, où les arguments on été localisés sur des machines différentes, est localisé sur la machine globale.

EXEMPLE:

On trouve dans la BDR, les relations X, Y, Z qui sont sur les sites A, B, C respectivement. Sur chaque site,

s'execute une machine locale. L'usager s'adresse à une machine globale localisée sur le site D, et il soumet la requête:

```
JOIN(JOIN(PROJECT(SELECT(X,critère1),Attributs_X),
         PROJECT(SELECT(Y,critère2),Attributs_Y)),
     SELECT(Z,critère3)) ;
```

Cette requête est analysée et transformée dans un arbre binaire (NGU77,CHA75). Cet arbre est alors décomposé en sous-arbres qui doivent être interprétés par une machine locale spécifique. La figure suivante montre le processus: (application des trois règles)

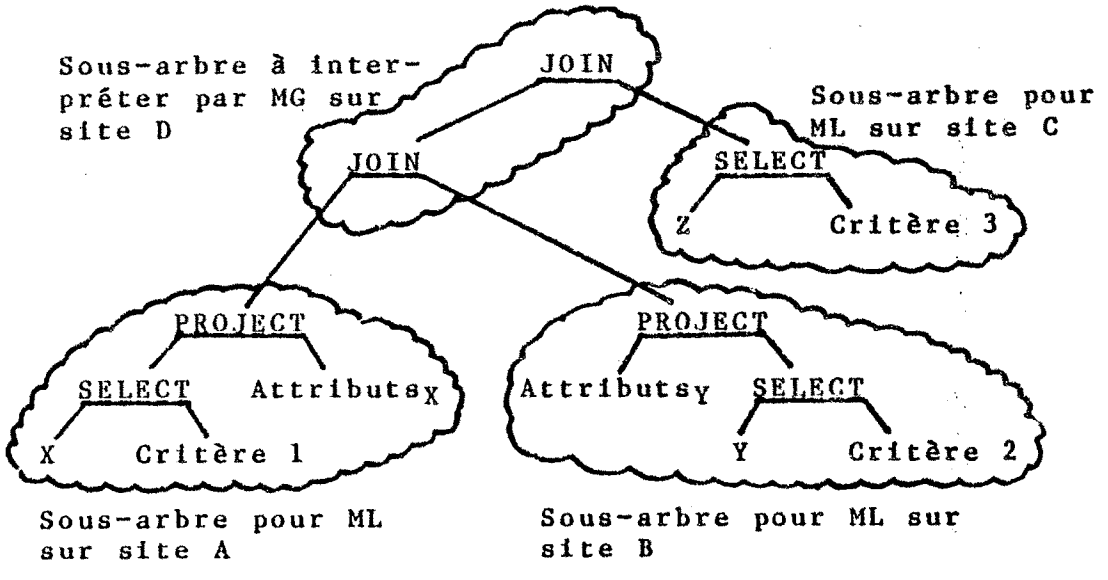


Figure 4.3 Décomposition et Localisation

La deuxième version, la décomposition dynamique (NGU79, NGU80,FER79), remplace la règle iii) de la première version par un processus adaptatif qui sélectionne une machi-

ne pour l'interprétation de l'opérateur en fonction des cardinalités des relations (qui peuvent être des résultats partiels de l'évaluation de la requête) données comme argumments. Une description complète de ce processus est donnée dans (FER79 ou NGU80).

Nous avons mis en oeuvre sur la maquette la première version de l'algorithme de décomposition. Il faut noter que URANUS ne dispose ni d'un mécanisme pour la définition de relations abstraites (ce qui simplifie en quelque sorte la définition de la BDR, cf §4.3.4) ni de mécanismes d'optimisation pour la réduction des opérateurs,etc (cf. §3.3, CAL78, CHA75). L'usager se voit alors obligé de réaliser cette optimisation avant de soumettre une requête à la maquette.

Les mécanismes devéloppés pour URANUS (NGU77,NGU78) permettent la traduction d'une base de données SOCRATE (un schéma réseau) en une base relationnelle. L'architecture du niveau local de la maquette correspond alors au cas 1 de la figure 2.6. Le problème de la correspondance entre le SERL (schéma externe relationnel local) et le SCL (schéma conceptuel local) de la base SOCRATE est résolu d'une façon très simple: une copie de la base est réalisée à l'initialisation du système. Cette copie, qui correspond à l'occurrence du SERL, est manipulée directement pendant toute la session de POLYPHEME. A la fin de la session, le

processus inverse a lieu: la base URANUS est retraduite vers la base SOCRATE, ce qui correspond alors, au cas 3 de la figure 2.6 de l'architecture de POLYPHEME.

4.3.2 Le moniteur d'exécution répartie: MER (ADE78,DEA80)

Ce module permet la communication entre les machines globales et les machines locales. Il a été développé pour offrir un interface PL/1 à la programmation d'applications réparties, p.ex: un SGBDR.

Un moniteur d'exécution répartie, MER, est implanté sur chaque machine du système POLYPHEME. Un MER offre des primitives pour l'activation asynchrone de procédures à distance; l'activation d'une procédure est considérée comme l'initilisation de l'exécution de la procédure et non comme un appel procédural avec blocage de l'appelant. Une application répartie est alors un ensemble de programmes locaux, chacun sur une machine; un programme local est un ensemble de procédures PL/1 qui s'exécutent sous le contrôle d'un moniteur (MER). La communication entre les machines est une communication entre des programmes locaux à chaque machine; cette communication est réalisée au travers de l'activation à distance de procédures.

Pour permettre un contrôle ordonné des activations de procédures, il est possible d'établir des connexions logiques (appelées interactions) entre deux MER distants. Tou-

te activation distante doit alors être réalisée sur une connexion logique.

MER offre un interface constitué par un ensemble de primitives appelables d'un programme PL/1. Les quatre primitives de base sont:

a) ouvrir une connexion logique (OITR),

b) fermer une connexion logique (CITR),

c) activer une procédure sur un site distant au travers d'une connexion logique déjà définie (INIT),

d) demander la surveillance d'une connexion logique (SURVEY).

Les primitives OITR, INIT, et SURVEY sont asynchrones, et le programme utilisateur est informé de sa terminaison par l'exécution d'une procédure d'acquittement dont le nom a été donné en paramètre à l'invocation de la primitive. Il y a alors trois types d'acquittement:

a) acquittement de l'ouverture d'une connexion logique;

b) acquittement de la procédure utilisateur de surveillance (le nom a été donné en paramètre de la primitive SURVEY) quand la connexion logique a été rompue; la connexion logique est alors automatiquement fermée;

c) acquittement de l'activation d'une procédure. Cet acquittement arrive quand l'exécution de la procédu-

re activée est finie.

Une procédure est alors considérée comme une unité atomique, c'est à dire qui s'exécute jusqu'à sa fin (une instruction RETURN ou END).

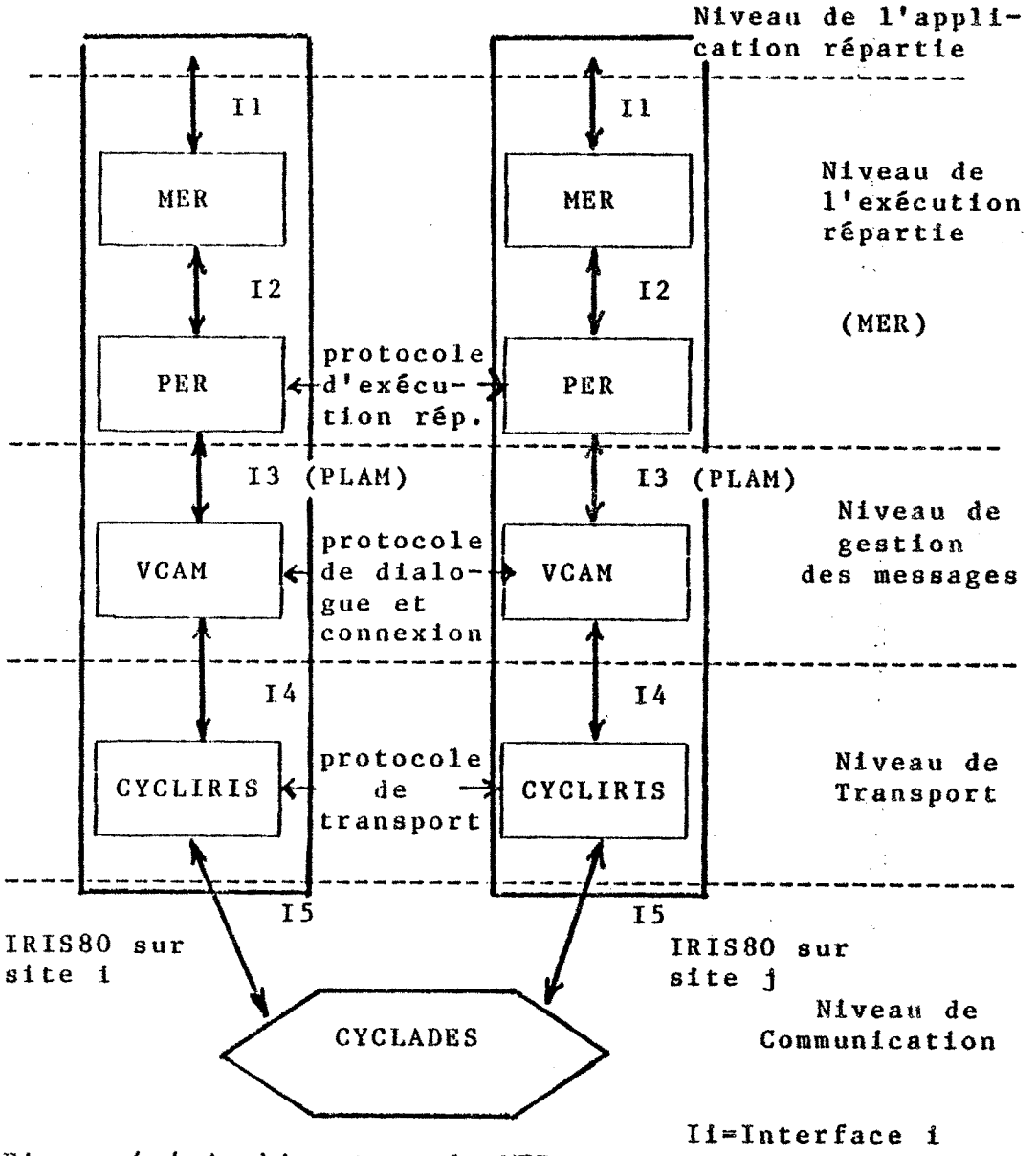


Figure 4.4 Architecture de MER

MER a été construit en suivant les recommandations de l'ISO (ISO79) pour l'architecture de systèmes ouverts. Cette architecture est basée sur plusieurs niveaux; l'interface entre chaque niveau est défini par un protocole qui établit les règles de dialogue entre les entités des différents niveaux.

La figure 4.4 (qui a été reprise de DEA80) montre cette architecture où la coopération entre deux moniteurs est définie par le protocole d'exécution répartie (PER). Ce protocole est construit sur la méthode standard de gestion de messages (VCAM) du système d'exploitation SIRIS 8. Un interface (I3, de la fig. 4.4) a été développé pour avoir accès à la VCAM depuis un programme PL/1 (l'interface PLAM). Cette méthode d'accès offre des primitives pour établir des "connexions logiques" (groupes de messages) entre deux utilisateurs de la VCAM; chaque utilisateur est connu du système par une "boîte à lettres" où les messages sont déposés. La VCAM permet aussi d'avoir un accès à la station de transport (CYCLIRIS) et donc à des utilisateurs distants.

Quand une primitive du MER est invoquée, le moniteur transforme cette invocation en une séquence de messages qui sont envoyés par CYCLADES au MER correspondant, qui a son tour les transforme en activations de procédures. Le moniteur assure le séquencement et le contrôle du flux des

messages sur la connexion logique. Une description du contrôle de l'activation asynchrone des procédures peut être trouvée dans (DEA80).

4.3.3 Le contrôleur d'exécution répartie: (AAD79)

Le contrôleur d'exécution répartie, qui correspond au niveau de l'application répartie de la figure 4.4, est construit directement sur l'interface d'utilisation de MER (I1 de la fig. 4.4, cf §4.3.2). Ce contrôleur est le noyau de la maquette POLYPHEME; il est composé de deux parties: le contrôleur global (ou GEXAR), et le contrôleur local (ou LEXAR) qui sont les noyaux des machines globale et locale respectivement.

La communication entre contrôleur global et local est définie par un protocole spécial: le protocole d'activation de procédures (PAP). les contrôleurs sont composés d'un ensemble de procédures qui utilisent les services de MER. Le protocole définit: la synchronisation entre les procédures des différents contrôleurs, les procédures qui doivent être activées, les conventions pour le passage des paramètres (valeurs et types), l'utilisation des connexions logiques, la forme de transmission des sous-arbres, la forme de transmission des n-uplets et le traitement des erreurs. Ce protocole est décrit dans (AAD79).

Le contrôleur global (GEXAR) met en oeuvre la partie du protocole PAP qui régule les échanges d'une machine globa-

le vers une machine locale. Ce contrôleur réalise les fonctions suivantes:

- gérer le dialogue avec l'usager de POLYPHEME,
- activer URANUS pour l'obtention d'un arbre localisé à partir de la requête de l'usager,
- déterminer les sous-arbres à interpréter sur chaque machine (globale ou locale),
- extraire et envoyer chaque sous-arbre à la machine locale correspondante. Il faut remarquer que c'est un algorithme (le sous-arbre peut être considéré comme tel) qui est transmis sur le réseau. Les sous-arbres peuvent être envoyés sans ordonnancement particulier et être interprétés en parallèle par les contrôleurs locaux;
- recevoir et stocker les n-uplets correspondants à l'évaluation de chaque sous-arbre. Quand tous les sous-arbres ont été évalués, le contrôleur appelle URANUS pour l'interprétation du sous-arbre correspondant à la machine globale (s'il y en a), et la réponse finale est donnée à l'usager;
- contrôler les erreurs d'une machine locale. Ces erreurs peuvent être provoquées par des irrégularités des sous-arbres, par une mauvaise évaluation, par la rupture de la communication (soit par panne de la ligne soit par panne de l'ordinateur, soit par une erreur fatale ("abort") de la machine locale, etc). Dans

le dernier cas la machine locale concernée abandonne le système et l'usager en est informé; l'exécution des requêtes peut continuer pour toute requête qui n'utilise pas cette machine. Une machine locale peut aussi être reconnecté dynamiquement au système.

- sauvegarder la partie locale de la BDR qui se trouve sur la machine globale,
- arrêter le système (arrêt des machines locales) à la fin de la session de l'usager de POLYPHEME.

Le contrôleur local met (LEXAR) en oeuvre la partie du protocole qui gére les échanges entre une machine locale et une machine globale. Ses fonctions sont:

- recevoir des sous-arbres et les mettre sous une forme interprétable par URANUS,
- activer l'interpréteur d'URANUS,
- prendre la relation résultat et envoyer les n-uplets correspondants vers la machine (le contrôleur) qui les attend,
- sauvegarder la base locale et arrêter la machine locale en cas de panne (rupture de la connexion logique ou "abort" de la machine locale).

Une machine locale doit être considérée comme une sorte de serveur. Au début d'une session POLYPHEME (rappelons que la maquette est mono-usager), le contrôleur établit des connexions logiques (interactions) de contrôle avec chaque

machine locale; l'usager est informé des connexions qui n'ont pas été possibles. Ces connexions servent à contrôler l'abandon des machines du système (par rupture de la connexion), et elles permettent la reconnexion dynamique des machines locales. Le contrôleur global établit, aussi, une connexion logique pour chaque sous-arbre à envoyer à une machine locale; la durée de vie de ces connexions est très brève, étant donné qu'elles sont fermées à la fin de l'évaluation du sous-arbre. Les connexions de contrôle sont maintenues pendant toute la session; elles permettent d'arrêter proprement les machines locales à la fin de la session.

Le contrôleur global gére le dialogue avec l'usager de POLYPHEME; un interface, qui consiste en un ensemble de commandes, a été défini. Ces commandes sont:

-exécuter une requête (REQ),
-exécuter une séquence de requêtes ("transaction")(TRA)
-créer une ou plusieurs transactions (CTR),
-reconnecter une machine locale (CML),
-arrêter la session (FIN).

L'usager est obligé de taper une de ces commandes après l'exécution d'une requête ou d'une transaction. Une requête est exprimée en langage relationnel algébrique d'URANUS (VOIR NGU77 ou AND79).

L'usager peut aussi activer des commandes pendant l'exécution d'une requête ou d'une transaction pour obtenir certaines informations. Cette entrée asynchrone de commandes est contrôlée par MER, qui active une procédure du contrôleur global pour son évaluation. Ces commandes sont:

-interrompre l'exécution d'une requête ou d'une transaction (INTR),
-connaître l'état des sous-arbres en cours d'exécution (ETSS),
-lister les descriptions des relations de la BDR (LSTR)
-arrêter brutalement la session (AFIN),
-connaître les machines locales connectées à POLYPHEME (SITC),
-connaître les machines locales déconnectées de POLYPHEME (SITD).

Une description plus détaillée du contrôleur réparti peut être trouvée dans (AAD79); le lecteur intéressé peut se reporter à (AND79) pour obtenir plus de détails techniques sur le mode d'opération de la maquette. Au paragraphe 4.3.5, nous montrons un scénario d'exécution d'une requête.

La figure 4.5 montre l'architecture réelle de la maquette POLYPHEME. Dans cette architecture toutes les combinaisons de machines locales et globales possibles sur les ordinateurs du réseau. La phase de développement de la maquette a été réalisée sur un seul ordinateur, où une machine globale et deux machines locales ont été testées avant de réaliser des expérimentations sur le réseau. Nous avons aussi expérimenté POLYPHEME sur le réseau CYCLADES avec une machine globale située à Paris (ordinateur Iris-80 de l'IRIA) et deux machines locales localisées sur le même ordinateur (un autre Iris-80) à Grenoble, (DAN80). Une démonstration publique a été effectuée lors du Congrès International sur les Bases de Données Réparties au mois de Mars 1980 à Paris.

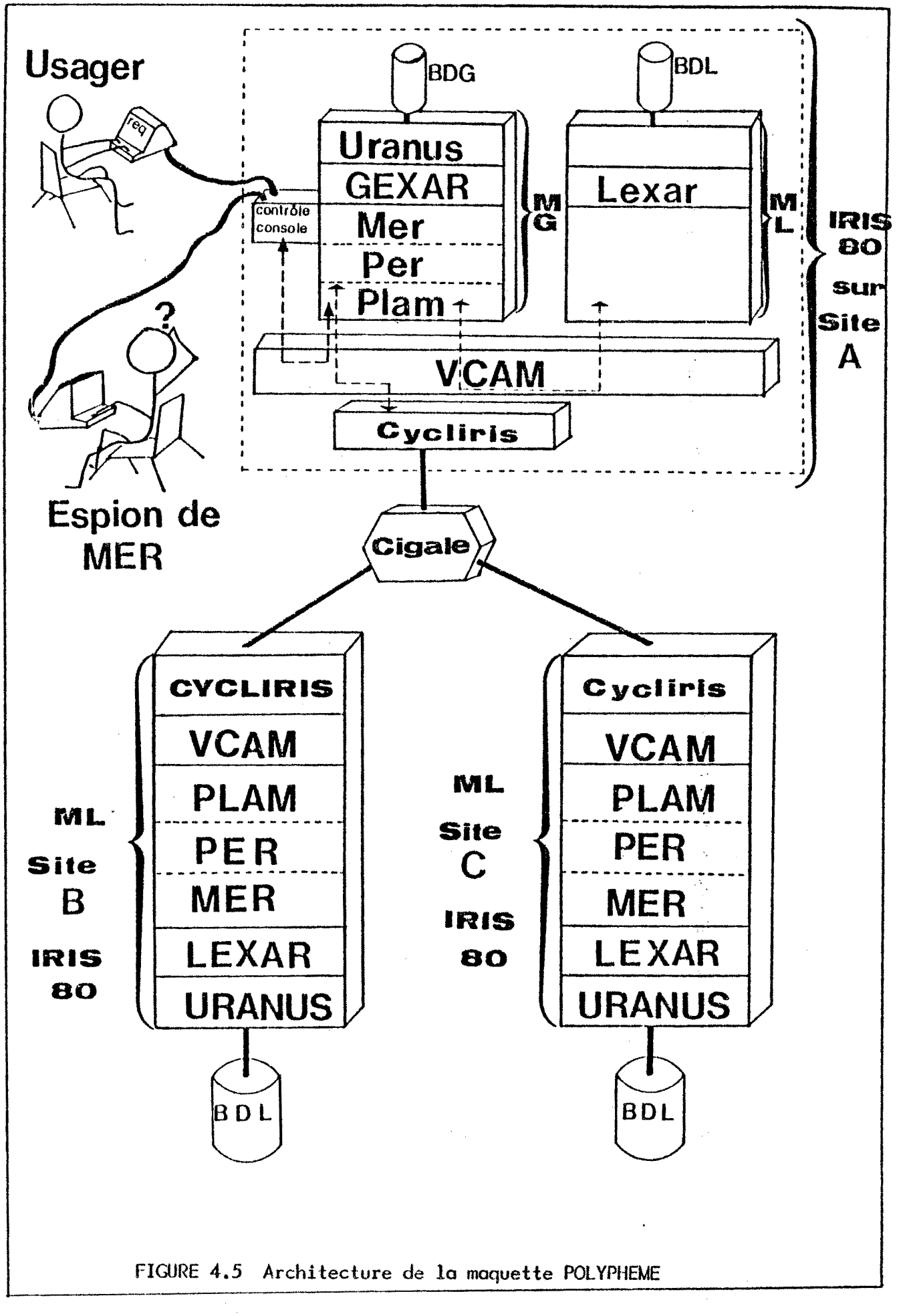


FIGURE 4.5 Architecture de la maquette POLYPHEME

4.3.4 La base de données répartie (BDR):

Nous avons centré la maquette sur le problème général d'analyse et d'exécution de requêtes portant sur des données relationnelles réparties.

L'unité de répartition est la relation, et une relation se trouve sur une seule machine. La notion de relation abstraite n'a pas été mise en oeuvre dans la maquette; donc, le schéma conceptuel de la base de données répartie (BDR) se compose de l'ensemble des relations de toutes les machines locales. Néanmoins, il n'est pas interdit d'avoir deux relations R1 et R2 stockées sur des machines différentes et telles que R1 et R2 constituent deux fragments d'une même relation "virtuelle" (abstraite) R, soit en fragmentation horizontale, soit en fragmentation verticale; on peut aussi considérer des relations globales qui sont des projections ou restrictions d'une relation locale. Cependant, c'est l'usager qui doit contrôler explicitement cette fragmentation; c'est à dire, qu'il doit inclure la fonction de définition dans toute requête adressée à ces relations.

La maquette ne supporte donc qu'un seul niveau de répartition des données. La description des relations qui font partie de la coopération est réalisée en spécifiant pour chacune le "site" sur lequel elle se trouve. Ce nom de "site" peut correspondre au nom d'une machine locale.

Exemple: définition d'une relation globale

```
$DEF
    Employe REL 500 SUR Toulouse
    DEBUT
      Numéro DE 0 A 999999 CLE
      Nom MOT 10
      Salaire DE 0 A 99999
    FIN
$OFF
```

Une fois la BDR décrite (sur la machine globale), l'usager n'a pas besoin de connaître la localisation des relations. La maquette se charge automatiquement de la décomposition et de la localisation des requêtes; c'est à dire que POLYPHEME assure la transparence de la localisation des relations pour l'usager.

Cette définition des relations, ainsi que leur manipulation est réalisée dans POLYPHEME au travers d'un langage relationnel algébrique développé pour le système URANUS. Les requêtes s'expriment par combinaisons des opérateurs relationnels (JOIN,SELECT,PROJECT,etc). Chaque opération de mise à jour (INSERT,DELETE,MODIFY) ne prend en compte qu'une seule relation à la fois (ce sont des opérations "ponctuelles" adressées à une seule machine). Ceci permet d'avoir plusieurs machines globales et locales simultanément en opération avec (évidemment!) des conventions de

mise à jour (rappelons que POLYPHEME ne possède pas de mécanisme pour le contrôle de la concurrence).

L'usager doit adresser à une machine globale toutes ses requêtes. La machine globale décompose ces requêtes en sous-requêtes qui seront (ou sont) interprétées sur les machines locales distantes. Cette interprétation est réalisée en parallèle et de manière asynchrone.

4.3.5 Scénario d'exécution de la maquette. Exemple

Nous allons prendre comme exemple celui de la démonstration de la maquette (DAN80,POL79). C'est un exemple très simple sur la gestion de wagons sur un réseau ferroviaire; il considère deux bases de données indépendantes et que l'on a homogénéisées pour les faire coopérer. La première base, située à Rennes, s'occupe du matériel roulant et de sa cargaison; on trouve donc, deux relations décrivant les wagons et les marchandises transportées:

WAGONS(Nw,Voie,Charge,Gare-attache,Marchandise,Date-charg)
MARCH(Marchandise,Descr,Type,Client,Date-soumission)

L'autre base de données décrit le réseau ferroviaire, c'est à dire les informations relatives aux gares de triage et aux voies reliant les différentes gares:

VOIES(Voie,Gare-origine,Destination,Type,Etat)
GARES(Gare,Secteur)

Cette base est implantée à Toulouse.

Ces deux bases de données se trouvent chacune sur une machine locale qui opère sur un ordinateur situé respectivement à Rennes et à Toulouse.

La coopération de ces bases de données prend en compte le fait que pour être acheminés, les wagons sont regroupés en trains et que ce sont les trains qui circulent sur les voies. Cette coopération est réalisée sur une machine globale située à Grenoble, et sur laquelle deux nouvelles relations ont été créées: WAGON_TRAIN(NW,NT) et TRAIN(NT, Voie).

Le schéma conceptuel global ou vue de la coopération (cf. §1.1.2) correspond alors aux relations: WAGONS,MARCH, VOIES, GARES, TRAINS, et WAGON-TRAIN. Cependant nous nous sommes limités à une partie de ce schéma (une vue) que nous allons définir sur la machine globale. Cette vue globale est formée par les relations: TRAINS, WAGON-TRAIN, et par une relation qui est une projection de la relation WAGONS (voir fig. 4.6).

```
TRAINS(NT,Voie) sur Grenoble
WAGON-TRAIN(NW,NT) sur Grenoble
VOIES(Voie,Gare-origine,Dest,Type,Etat) sur Toulouse
WAGONS(NW,Voie,Charge,Gare-attache)  sur Rennes

Note: La localisation est transparente pour l'usager
```

Figure 4.6 La vue Globale

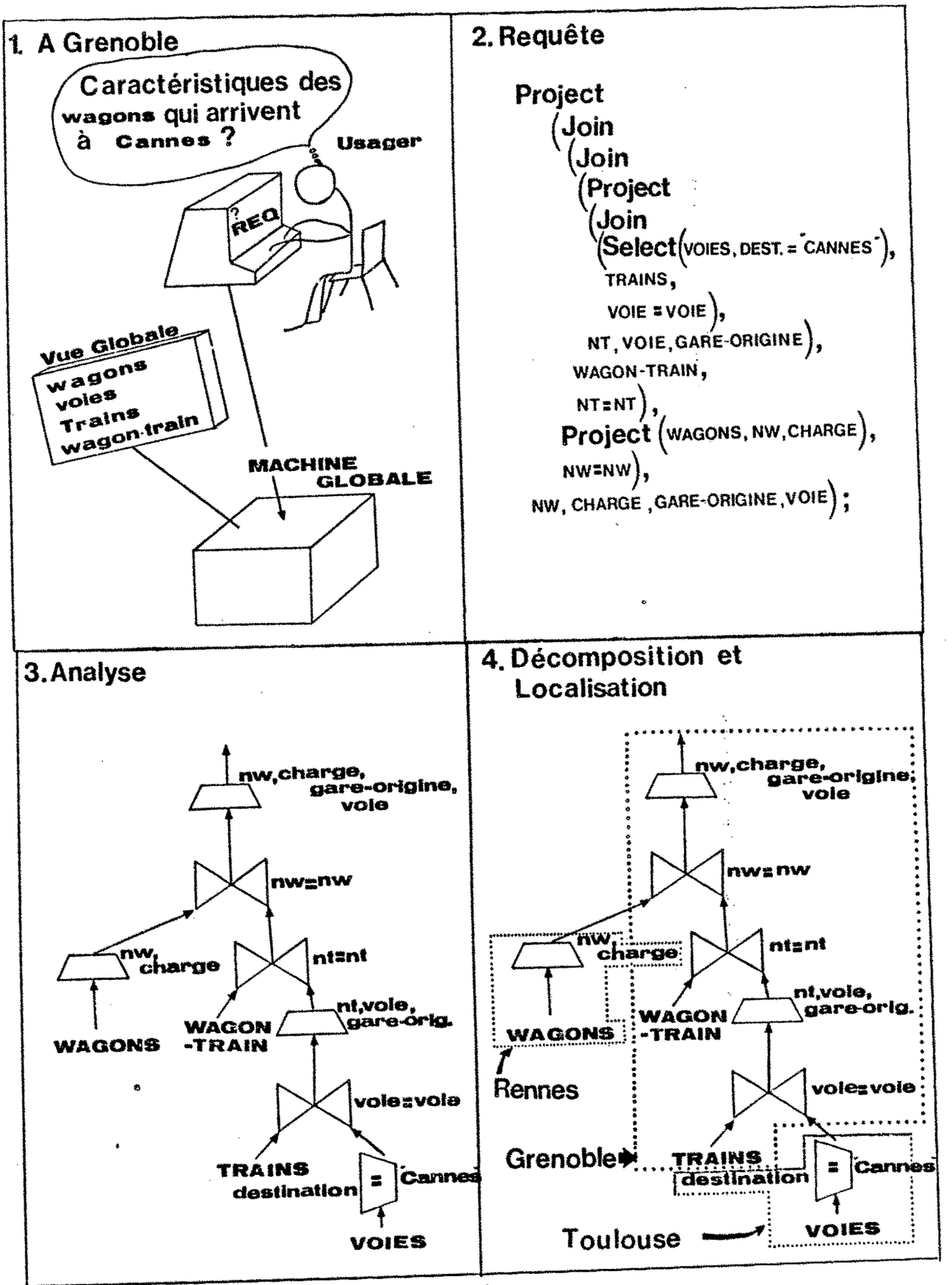


Figure 4.7 Histoire d'une requête

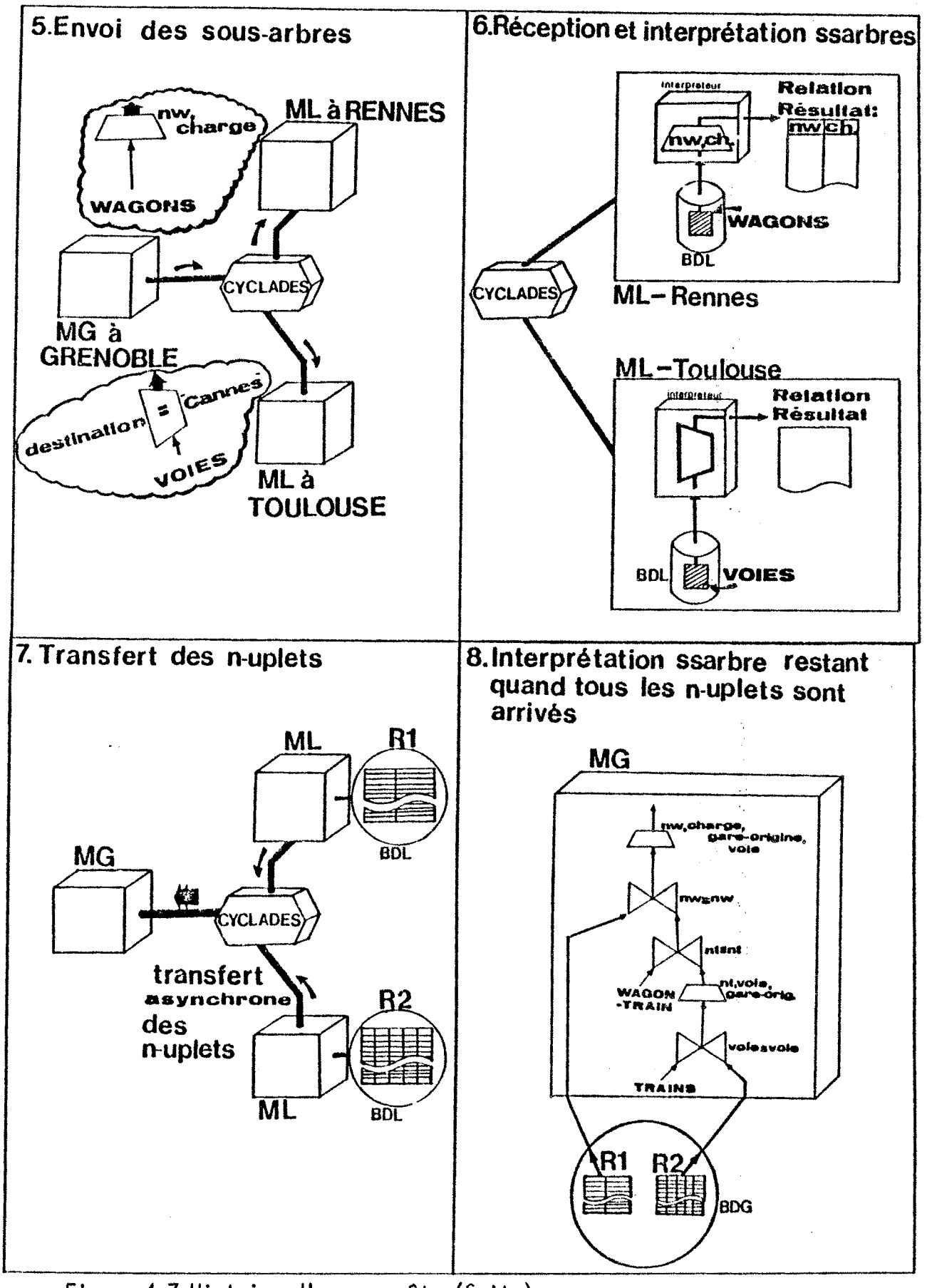


Figure 4.7 Histoire d'une requête (Suite)

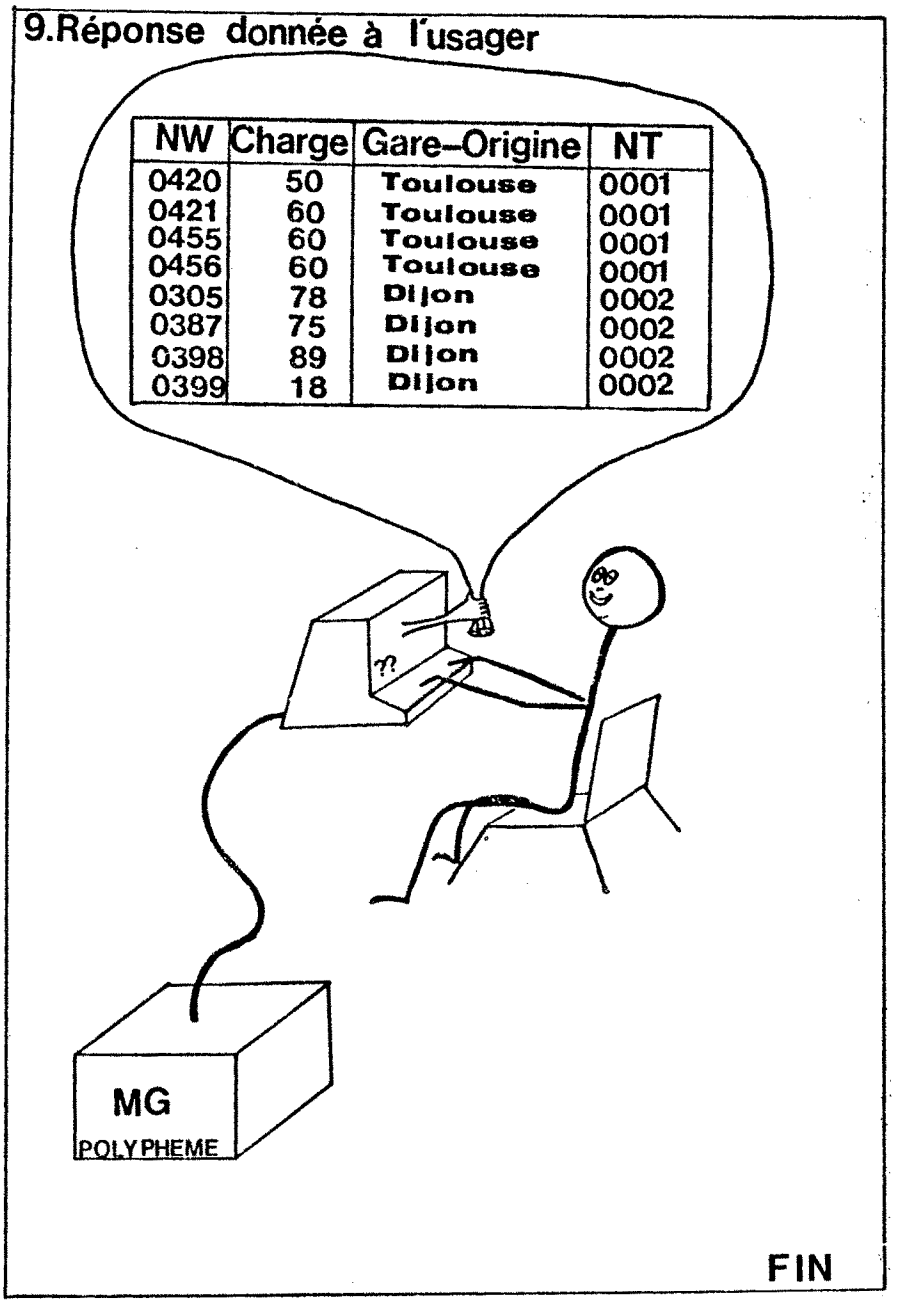


figure 4.7 Histoire d'une requête (Suite)

Pour illustrer le fonctionnement de la maquette, nous allons décrire le processus d'exécution d'une requête:

A. SUR LA MACHINE GLOBALE

1. Requête de l'usager: l'usager ne voit que la vue globale et une machine capable d'interpréter ses requêtes sur cette vue. La localisation des relations lui est transparente.

Un usager désire connaître les caractéristiques des wagons qui arrivent à la station de Cannes, c'est à dire le numéro de wagon (NW), sa charge, sa gare d'origine, la voie sur laquelle il circule. Cette requête est exprimée avec le langage relationnel d'URANUS. La figure 4.7, cadres 1 et 2, illustre la requête.

2.Analyse, décomposition et localisation: la requête est alors analysée par la machine globale. L'arbre binaire (cadre 3, fig 4.7) résultant de l'analyse, est ensuite localisé et décomposé en sous-arbres (cadre 4,fig. 4.7).

Un algorithme de localisation statique (cf. 4.3.1) est appliqué donnant ainsi trois sous-arbres qui peuvent être interprétés chacun sur une seule machine; un sous-arbre sera interprété sur la machine locale de Rennes, le deuxième sur la machine locale de Toulouse, et le troisième sur la machine globale à Grenoble mais seulement après l'interprétation des deux premiers sous-arbres.

3.Transport: Les sous-arbres sont envoyés aux machines locales correspondantes (cadre 5, Fig.4.7).

B. SUR LES MACHINES LOCALES

4.Réception d'un sous-arbre: chaque machine locale reçoit les sous-arbres sous un format très particulier (cf. protocole PAP, §4.3.3) qui doit être transposé sous une forme interprétable par URANUS.

5.Interprétation du sous-arbre: le sous-arbre est interprété par URANUS. Le résultat est une relation temporaire dont les n-uplets sont transférés vers la machine globale (cadre 6, fig.4.7).

6.Transfert de la relation résultat: les n-uplets de la relation résultat sont tranférés vers la machine globale en utilisant le PAP local-global (cadre 7, fig 4.7).

C.SUR LA MACHINE GLOBALE

7.Rassemblement des relations: les n-uplets des relations résultant de l'interprétation des sous-arbres sont rassemblés, la machine globale recevant de manière asynchrone les n-uplets envoyés par les machines locales.

8.Interprétation du sous-arbre restant:(s'il existe) une fois que tous les n-uplets des relations résultant des sous-arbres ont été reçus, la machine globale peut interpréter le reste de l'arbre (cadre 8,fig. 4.7). La réponse est donnée à l'usager (cadre 9, fig 4.7) sous la forme d'un tableau.

L'usager peut alors effectuer une autre requête ou transaction.

4.4 Extensions de la maquette:

La maquette de POLYPHEME que nous venons de décrire n'est qu'un premier niveau de mise en oeuvre d'un SGBDR:

- les mises à jour sont "ponctuelles",
- il n'existe pas de contrôle d'accès concurrent aux données,
- il n'y a pas de reprise après panne,
- il n'existe pas de possibilité de définir des relations "abstraites",
- le code de la maquette n'est pas un code optimisé et il est donc limité par le temps d'unité centrale de l'ordinateur.

Cependant, il existe les notions suivantes:

-décomposition de requêtes,
-transport d'algorithmes,
-exécution répartie d'algorithmes,
-mise à jour à distance,
-mode "transactionnel",
-détection de panne de réseau/machine,
-fonctionnement en mode dégradé,
-reconnexion dynamique de machines.

Une première extension est la mise en oeuvre d'un mécanisme pour la définition et la manipulation de relations "abstraites" globales comme compositions de relations locales en accord avec les critères de fragmentation que nous avons décrits au Chapitre 3. La mise en oeuvre du formalisme proposé dans ce chapitre 3, peut être réalisée facilement en utilisant des algorithmes répartis (cf. §4.3.2). Dans ce sens, MER se caractérise par sa flexibilité et son extensibilité. Nous avons envisagé de mettre en oeuvre le formalisme des mises à jour sur LAMB, mais la maquette et sa démonstration ont eu la priorité de réalisation.

Une deuxième extension est d'adapter la maquette à un environnement multi-usager. Ceci implique des mécanismes de contrôle de la concurrence et de la validation des mises à jour. Ce n'est pas un problème trivial et plusieurs études ont été réalisées sur ce sujet.

A partir de cette deuxième extension, une troisième est envisageable: un mécanisme de reprise après panne.

Ces deux extensions sont un des buts du projet SCOT (SCO80) développé au sein du Centre Scientifique de la CII-Honeywell Bull à Grenoble, qui a décidé d'utiliser l'expérience de POLYPHEME pour la réalisation d'un système transactionnel réparti. Le projet MICROBE (MIC80) qui est aussi un prolongement de POLYPHEME sur un réseau local de

micro-ordinateurs, tiendra compte des mécanismes pour le contrôle de la concurrence; la reprise, dans ce cas, est plus simple à considérer étant donné que le réseau local est un réseau à diffusion où tout le monde reçoit les messages émis.

4.5 Conclusions

Dans ce chapitre nous avons décrit la maquette POLYPHEME, pour laquelle nous avons réalisé le contrôleur réparti (cf.§4.3.3). Le paragraphe 4.4 résume les caractéristiques principales de cette maquette.

Nous avons donné un exemple complet de l'exécution d'une requête et nous avons décrit le Logiciel d'Accès et Manipulation de Blocs (LAMB) dont la réalisation nous a appris l'importance du développement de systèmes relationnels extensibles et modulaires.

La maquette, même si elle n'est qu'un premier niveau dans la mise en oeuvre d'un SGBDR, doit être considérée comme un résumé du travail réalisé dans le projet POLYPHEME.

CHAPITRE 5

CONCLUSIONS GENERALES

EXPERIENCES ACQUISES DE POLYPHEME

"Del dicho al hecho,
hay mucho trecho"
(Proverbe populaire)

"Savoir où l'on veut aller,
c'est très bien; mais il
faut encore montrer qu'on
y va."

Emile Zola

Trois questions se posent:

Q1: Peut-on construire un système général de coopération de bases de données réparties sur un réseau d'ordinateurs?

Q2: Si oui, quelles sont les caractéristiques de ce système?

Q3: Est-ce que ces caractéristiques permettent de satisfaire tous les besoins d'une application de type bases de données réparties?

C'est pour répondre à ces questions que l'équipe POLYPHEME a mené depuis 1976 plusieurs travaux. Ce document, en particulier, a tenté d'éclaircir plusieurs aspects concernant l'architecture générale de POLYPHEME, et les problèmes liés à la conception d'une vue de coopération, à l'intégrité et au traitement des opérations de mise à jour.

La réponse à la première question est affirmative, et elle est confirmée par la réalisation d'une maquette dont nous avons présenté l'architecture au Chapitre 4. Oui, on peut construire des SGBDR utilisant une approche de conception ascendante. Cependant, nous ne prétendons pas avoir abordé tous les aspects concernant la réalisation de tels systèmes, en particulier les problèmes de cohérence, de validation et de reprise qu'il est indispensable de résoudre mais dont le contrôle est particulièrement délicat

étant donné la dynamique du système (pannes, ré-insertion et abandon des machines, etc). En ce sens, nous considérons les projets SCOT (SCO80) et MICROBE (MIC80) comme une continuation de POLYPHEME.

L'introduction de ce travail a montré les différentes hypothèses et solutions (en réponse à la question Q2) que nous avons prises pour la construction du SGBDR-POLYPHEME. D'abord, rappelons que notre but principal était la coopération de bases de données hétérogènes réparties sur un réseau d'ordinateurs. Un modèle commun a été choisi pour homogénéiser les bases de données, et pour décrire une vue commune au travers de laquelle la coopération est réalisée. Des mécanismes pour le traitement des requêtes adressées à cette vue ont été définis (CAL78,NGU79) et mis en oeuvre (FER79,AAL80a,AAL80b). Ce traitement tient compte d'une part du débit des données sur le réseau qui est beaucoup plus lent que celui entre mémoire secondaire et mémoire centrale, et d'autre part de l'opportunité d'exécution parallèle par la présence de plusieurs ordinateurs. Un mécanisme pour l'exécution et la synchronisation de programmes répartis (p.ex: un SGBDR) a été développé (ADE78,DEA80) ce qui permet de montrer que la mise en oeuvre d'applications réparties n'implique pas d'attendre que des systèmes d'exploitation répartis soient réalisés.

La maquette nous a montré comment toutes ces recherches peuvent être intégrées, même si cette intégration n'a été

réalisée que d'une manière partielle (la réalisation était également liée aux problèmes matériels et logiciels du Centre de Calcul de Grenoble). Nous avons effectué une démonstration publique de cette maquette lors du Congrès International sur les Bases de Données Réparties à Paris au mois de Mars 1980 (DEA80). Aujourd'hui, c'est le projet SIRIUS de l'IRIA qui en assure l'expérimentation.

L'approche ascendante que nous avons prise pour la conception de la vue commune est difficile à mettre en pratique à cause des différents risques de création d'incohérences entre les données. Il faut alors considérer tous les liens sémantiques des données, ce qui n'est pas toujours possible car cela dépend de la capacité de description du modèle. Un modèle conceptuel-MOGADOR, (ADI78)- a été développé pour suppléer à ces besoins. Cependant, ce modèle est assez complexe à mettre en oeuvre et ses caractéristiques ne permettent pas de le proposer à un usager non-informaticien; nous avons donc choisi- pour sa simplicité et sa facilité de réalisation- le modèle relationnel comme modèle commun d'homogénéisation des bases de données. Toutefois, nous avons proposé l'utilisation de MOGADOR au niveau de la conception de la BDR et des techniques pour la transformation des schémas hétérogènes des bases de données en schémas relationnels. Cependant, comme le modèle relationnel ne permet pas d'exprimer tous les liens sémantiques des données, il faut imposer des restrictions

(parfois très sévères) dans la description des relations de la vue commune.

Aux Chapitre 2 et 3, nous avons analysé le problème de la correspondance des opérations réalisées sur la vue commune en opérations sur les schémas relationnels qui constituent cette vue. Une conclusion importante de cette étude, est qu'il est possible de définir un traitement standard des opérations de mise à jour réalisées sur les relations de la vue commune; ce traitement dépend directement du critère de fragmentation de la relation concernée. Le formalisme donné permet d'exprimer facilement cette correspondance et de définir une certaine cohérence sémantique associée à la fragmentation.

Il n'est pas très facile de répondre à la question Q3. Il faut tout d'abord analyser la notion de localisation associée à la fragmentation des relations globales (Chapitre 3). Rappelons que les fragments (relations locales) appartiennent chacun à un schéma particulier; ce schéma est localisé sur une machine qui opère sur un des ordinateurs du réseau. Une relation globale est formée par un ou plusieurs fragments, et cette fragmentation est inconnue de l'usager; donc, POLYPHEME assure une transparence de la localisation. Cependant, la conception d'une BDR n'est pas évidente et l'administrateur se voit parfois dans l'obligation de créer de nouvelles relations contenant une no-

tion de localisation applicable par transitivité (voisinage) aux autres relations globales.

Exemple: Trois usines d'une entreprise ont des bases de données qui sont exploitées de manière indépendante. La direction de l'entreprise décide d'unifier l'exploitation des BD en utilisant les principes de POLYPHEME. Supposons que les BD gérent des employés (trois relations indépendantes) qui sont alors généralisées dans une seule relation globale EMPLOYE. Comment savoir qu'un employé travaille dans telle ou telle usine? (C'est à dire, quel est le critère d'identification des fragments?). Une solution consiste à modifier le SERL (schéma externe relationnel local) pour inclure la relation USINE(Numèro-usine, Ville) et à rajouter l'attribut Numèro-usine à la relation EMPLOYE. On peut alors former une relation globale USINE fragmentée horizontalement par l'attribut Ville et la relation globale EMPLOYE fragmentée par voisinage de la relation globale USINE.

Nous pouvons nous rendre compte que ces choix quelque peu arbitraires dépendent intrinsequement de la manière dont la coopération est réalisée. Il faut remarquer que dans ce sens la solution de SDD-1 (ROT80) et INGRES (EPS78) avec l'identificateur de n-uplet (No-nuplet.site) est beaucoup plus précise, mais toute relation globale doit avoir cet identificateur comme attribut additionnel.

Au cours du projet quatre applications ont été étudiées. La première était fournie par le CNET (catalogues et inventaires du système E10) (CL077); la deuxième était constituée par le système de courrier électronique AGORA (BOG78); la troisième était un exemple très élémentaire sur la gestion de la scolarité (professeurs, étudiants et enseignements) réalisé pour les tests initials de la maquette; la quatrième application est l'exemple de la gestion de wagons sur un réseau ferroviaire développé au Chapitre 4 (cf§ 4.3.5 et DAN80,POL79) et qui a été mis en oeuvre pour les demonstrations de la maquette. Une cinquième application a été définie à l'IRIA (VIL80); elle illustre les possibilités que POLYPHEME offrirait à une banque.

Nous ne pouvons pas dire que POLYPHEME permet de satisfaire à tous les besoins d'une application. Un des points très contestés est la localisation (comme nous venons de le voir) qui implique l'adaptation de schémas centralisés à un schéma réparti. Le deuxième point criticable concerne l'aspect dynamique de l'application: il est très difficile (voire impossible) de modéliser complètement le comportement de l'application avec le modèle relationnel. Au chapitre 3, nous avons vu comment il faut recourir à une structure basée sur les types de données abstraits pour pouvoir définir le comportement des opérations de mise à jour et des contraintes d'intégrité. MOGADOR s'approche

un peu de la modélisation d'un comportement (les fonctions ...), mais il faudrait encore travailler sur la présentation du modèle aux usagers non-informaticiens. Dans ce sens, le projet SCOT (SCO80) a repris cette expérience et étudie les possibilités de définition d'un modèle transactionnel.

Les perspectives de recherches futures sont considérables. POLYPHEME n'a analysé qu'une partie des techniques de coopération de bases de données réparties et deux enseignements très importants peuvent être tirés de cette étude:

i) il faut un modèle de coopération permettant d'exprimer le comportement des applications;

ii) il faut un modèle pour l'exécution répartie.

Nous avons déjà discuté suffisamment le point i). Le deuxième point concerne directement le moniteur d'exécution répartie (MER)- décrit dans (DEA80) et au Chapitre 4 de ce travail- réalisé pour POLYPHEME; ce moniteur offre des services pour la programmation en PL/1 d'applications réparties et il a été construit initialement pour répondre aux nécessités primordiales de POLYPHEME: l'exécution distante de requêtes et la communication entre les différentes machines. Cependant, l'utilisateur de ce système est obligé de "descendre" à un niveau tel qu'il doit établir

et gérer des connexions logiques pour la communication entre les différentes parties du programme réparti. MER ne fournit pas à l'usager de mécanismes de contrôle d'accès concurrent; la reprise doit être contrôlée entièrement par l'utilisateur. Le système SER (SET80) utilisé dans SIRIUS DELTA (SIR80) et MICROBE (MIC80) introduit des améliorations notables qui offrent une meilleure transparence des notions de contrôle associées au réseau. Le projet SORTILEGE (SOR79,GUI79) est aussi un pas vers le développement de langages pour l'écriture de programmes répartis, mais les notions de reprise ont été quelque peu oubliées.

Nous pensons que le système d'exécution répartie pour un SGBDR doit offrir des outils pour la synchronisation (parallélisme et séquencement) de l'exécution des différentes opérations, la reprise après panne, et le contrôle de la concurrence. SIRIUS-DELTA (SIR80) et SCOT (SCO80) ont fait un bon pas dans cette direction.

Pour conclure il faut dire que la recherche dans les systèmes de coopération de bases de données réparties est loin d'être close. POLYPHEME nous a permis d'ouvrir la voie vers d'autres projets pour lesquels il reste encore beaucoup à faire.

BIBLIOGRAPHIE

"Morte la bête, Mort le venin"

(Proverbe populaire)

ADD79 J.M.ANDRADE,M.ADIBA

"Description de la maquette et de l'Interface Réparti"

Note technique 39, Projet Polyphème, Decembre 1979

AAL80a M.ADIBA,J.M.ANDRADE,P.DECITRE,F.FERNANDEZ,NGUYEN GIA TOAN

"Polyphème, an experience in distributed data base design and implementation",Congrès International sur les Bases de Données Réparties, Paris, Mars 1980.

AAL80b M.ADIBA,J.M.ANDRADE,F.FERNANDEZ,NGUYEN GIA TOAN

"An overview of the POLYPHEME distributed database management system", IFIP Conf., Melbourne, October 1980.

ABR74 J.R.ABRIAL

"Data Semantics", IFIP TC2 Working Conference, Cargèse, Avril 1974.

ABU79 A.V.AHO,C.BEERI,J.D.ULLMAN

"The theory of joins in relational databases", ACM transactions on Database Systems, Vol 4, No 3, Septembre 1979, pp297-314.

ACA78 M.ADIBA,Y.Y.CALECA

"Langage de description et de manipulation de vues relationnelles. Structure interne de représentation des catalogues et requêtes.", Note technique 25, Projet POLYPHEME, Juin 1978.

ACE78 M.ADIBA,J.Y.CALECA,C.EUZET

"A distributed database system using logical relational machines", IV VLDB Conf., Berlin, Septembre 1978.

ADA79 M.ADIBA,J.M.ANDRADE

"Expressing update consistency in distributed databases", RR 193, Laboratoire IMAG, Grenoble, Decembre 1979.

ADD77 M.ADIBA,C.DELOBEL

"The cooperation problem between different data base management systems", IFIP TC2 Working Conf., Nice, Janvier 1977.

ADD78 K.C.TOTH,S.A.MAHMOUD,J.S.RIORDON,O.SHERIF

"The ADD System: An Architecture for Distributed Databases", IV VLDB Conf., Berlin, Septembre 1978.

ADE78 E.ANDRE,P.DECITRE

"On providing distributed application programmers with control over synchronization", Proc. Computer Network Protocol Symposium, Liège, Février 1978.

ADI77 M.ADIBA

"Projet POLYPHEME: MOGADOR un modèle général de données réparties", RR 81, Laboratoire IMAG, Grenoble, Juillet 1977.

ADI78 M.ADIBA

"Un modèle relationnel et une architecture pour les

systèmes de Bases de Données Réparties. Application au Projet POLYPHEME", Thèse d'Etat, Université de Grenoble, Septembre 1978.

ADL76 M.ADIBA,C.DELOBEL,M.LEONARD
"A unified approach for modelling data in logical data base design", IFIP TC2, Freudenstadt, Janvier 1976.

ANB80 E.ANDRE,G.BOGO
"ADA, Abstract data types and distributed database transactions", RR SCOT, Centre Scientifique CII Honeywell Bull, Grenoble, Mai 1980.

AND77 J.M.ANDRADE
"Simulation du comportement de différentes bases de données réparties sur un réseau d'ordinateurs", Projet DEA, INPG, Grenoble, Juin 1977.

AND77a J.M.ANDRADE
"Spécification LAMB, Vol 0", Note technique 11, Projet POLYPHEME, Octobre 1977.

AND77b J.M.ANDRADE,NGUYEN GIA TOAN,A.STIERS
"Spécifications LAMB, Vol 1", Note Technique 15, Projet POLYPHEME, Décembre 1977.

AND78 J.M.ANDRADE
"LAMB: Spécifications générales de réalisation, Vol 2" Note technique 26, Projet POLYPHEME, Juin 1978.

AND79 J.M.ANDRADE

"Manuel d'opération de la maquette POLYPHEME", Note Technique 38, Projet POLYPHEME, Decembre 1979.

ANS75 ANSI(X3)SPARC

"Study group on database mono processor systems: interim report", Bulletin of ACM SIGMOD, Vol 7, No 2, Juin 1976.

AST76 M.ASTRAHAN et al.

"System R: Relational Approach to Database Management" ACM Transaction on Database Systems, Vol 1, No 2, Juin 1976.

BAD79 D.Z.BADAL

"Semantic integrity, consistency and concurrency in distributed database systems", Phd dissertation, Computer Science Dept., UCLA, Mars 1979.

BAN79 J.P.BANATRE,M.BANATRE

"Language features for description of cooperating processes", Working Note, IRISA, Rennes 1979.

BAP79 D.Z.BADAL,G.POPEK

"Cost and Performance Analysis of Semantic Integrity Validation Methods", Proc. of ACM SIGMOD 79, Boston, Mai 1979.

BAS78 F.BANCILHON,N.SPYRATOS

"Update semantics of relational views", RR 329, IRIA,

Septembre 1978.

BBG78 C.BEERI,P.BERNSTEIN,N.GOODMAN
"A Sophisticate's Introduction to Database Normalization Theory", Proc IV VLDB, Berlin, Septembre 1978.

BCL79 O.P.BUNEMAN,E.K.CLEMONS
"Efficiently monitoring Relational Databases", ACM Transactions on Database Systems, Vol 4, No 3, Septembre 1979.

BER77 P.A.BERNSTEIN,D.W.SHIPMAN,J.B.ROTHNIE,N.GOODMAN
"The concurrency control mechanism of SDD1: A System for Distributed Databases", Tech. Rep. CCA-77-09, Computer Corporation of America, Cambridge, Mass., Decembre 1977.

BOG78 G.BOGO
"AGORA: un système de courrier électronique sur le système de bases de données réparties POLYPHEME", Rapport DEA Génie Informatique, INPG, Grenoble, Sept.1980

CAL78 J.Y.CALECA
"Projet POLYPHEME: l'expression et la décomposition de transactions dans un système de bases de données réparties", Thèse de Docteur 3ème Cycle, Université de Grenoble, Septembre 1978.

CAS77 J.Y.CALECA,A.STIERS
"Interface relationnel POLYPHEME-SGBD SOCRATE", Note

Technique 1, Projet POLYPHEME, Janvier 1977.

CHA75 D.CHAMBERLIN,J.N.GRAY,I.L.TRAIGER
"Views, authorization, and locking in a relational database system", Proc. AFIPS National Computer Conf., Anaheim, CA, Mai 1975.

CHS75 P.Y.CHANG,J.M.SMITH
"Optimizing the performance of a relational algebra database interface", Comm. ACM, Vol 18, No 10, Octobre 1975.

CHU73 W.W.CHU
"Optimal file allocation in a Computer Network", Computer Communication Network, (eds)N.Abramson&F.Kuo, Prentice Hall, Englewood-Cliffs, NJ, 1972.

CLE78 E.CLEMONS
"An External Schema facility to support database update", Conf. on Databases: Improving Usability and Responsiveness, Haifa, Academic Press, Août 1978.

CLO77 J.CLOAREC
"Présentation de l'application catalogue et inventaire des equipements utilisés dans le système E10", CNET fiche technique FT/RCI/EGN/6, Lannion, Novembre 1977.

COD70 E.F.CODD
"A relational model of data for large shared databanks", Comm ACM, Vol 13, No 6, Juin 1970, pp 377-387.

COD72 E.F.CODD

"Further normalization of the database relational model", in Database System, Courant Comp. Sci. Sym. 6 (R.Rustin,ed), Prentice-Hall, Englewood-Cliffs, NJ, 1972, pp 33-64.

COD74 E.F.CODD

"Recent investigations in a relational database system", Information Processing 74, North-Holland Pub.Co. Amsterdam, 1974, pp 1017-1021.

COD75 E.F.CODD

"Understanding relations (Installment #7), FDT Bulletin of ACM-SIGMOD, Vol 7, No 3-4, 1975, pp 23-28.

DAB78a U.DAYAL,P.A.BERNSTEIN

"On the updatability of relational views", Proc. VLDB, Berlin, Septembre 1978.

DAB78b U.DAYAL,P.A.BERNSTEIN

"The fragmentation problem: lossless decomposition of relations into files", Tech. Rep. CCA-78-13, Computer Corporation of America, Cambridge, Mass., Novembre 1978.

DAT77 C.J.DATE

"An introduction to Database Systems", (2th edition), Addison-Wesley Pub.Co., Reading, Mass., 1977.

DAY79 U.DAYAL

"Schema-mapping problems in Database Systems", Ph.D.

Thesis, Harvard University, Cambridge, Mass., Août 1979.

DEA80 P.DECITRE,E.ANDRE

"POLYPHEME Project: The DEM distributed execution monitor", Proc. Inter. Symp. on Distributed Databases, in Distributed Databases (C.Delobel,W.Litwin, editors) North-Holland Pub.Co., Mars 1980.

DAN80 P.DECITRE,J.M.ANDRADE

"Technical outline of the POLYPHEME demonstration prototype", Proc. Inter. Symp. on Distributed Databases, (C.Delobel,W.Litwin, editors), North-Holland Pub. Co., Mars 1980.

DEL79 C.DELOBEL

"Theoretical aspects of modeling in relational Databases", RR 177, Laboratoire IMAG, Juin 1979.

DOF79 C.S.DOS SANTOS,A.L.FURTADO

"Synthesis of update transactions", T.R. DB107901, Dept. Informatica, Pontificia Universidade Catolica do Rio de Janeiro, Octobre 1979.

ESC75 K.P.ESWARAN,D.D.CHAMBERLIN

"Functional specification of a subsystem for database integrity", Proc. I-VLDB Conf., Framingham, Mass., 1975.

ESP78 R.EPSTEIN,M.STONEBRAKER,E.WONG

"Distributed query processing in a relational database system", 3th Berkeley Conf. on Distributed Data Management and Computer Networks, Août 1978.

ESW76 K.P.ESWARAN,J.N.GRAY,R.A.LORIE,I.L.TRAIGER

"The notion of consistency and predicate locks in a database system", Comm. ACM, Vol 19, No 11, Novembre 1976.

EUZ79 C.EUZET

"Interprétation de graphes pour l'enchainement des requêtes dans un système de gestion de bases de données réparties. Projet POLYPHEME", Thèse de Docteur de 3ème Cycle, INPG, Grenoble, Septembre 1979.

EUZ80 C.EUZET

"Etude de la confidentialité dans le cadre de la coopération d bases de données", Rapport DRET, Institut National Polytechnique de Grenoble, Janvier 1980.

FER79 F.FERNANDEZ

"Etude d'algorithmes d'optimisation de requêtes dans les SGBD répartis. Mise en oeuvre de l'optimiseur DOPAGE", Rapport DEA, USMG, Grenoble, Septembre 1979.

FER80 F.FERNANDEZ

"Micro Memoire Relationnelle (MIMER)", Note Technique NTM01, Projet MICROBE, IMAG, Mars 1980.

FLO74 J.J.FLORENTIN

"Consistency Auditing of Data Bases", The Computer Journal, 17, 1, Fevrier 1974.

FUR78 A.L.FURTADO

"A view construct for the specification of external schemas", Tech. Rep. Dept. de Informatica, Pontificia Universidade Catolica, Rio de Janeiro, Fevrier 1978.

FUS77 A.L.FURTADO,K.C.SEVCIK

"Permitting updates through views of databases", Tech. Rep. Dept. de Informatica, Pontificia Universidade Catolica, Rio de Janeiro, Fevrie 1978.

GRA78 J.N.GRAY

"Notes on database operating systems", IBM Research Lab., RJ2188 (30001) 2/23/78, San Jose, Feb. 1978.

GRA79 J.N.GRAY

"A discussion of distributed systems", IBM Research Lab., RJ2699(34594) 9/13/79, San Jose, Sept. 1979.

GUI79 J.M.GUILLOT

"Propositions pour un langage d'écriture de programmes répartis. Expression du contrôle et de la communication entre processus distribués", Thèse de Doctorat 3ème Cycle, Université de Grenoble, Décembre 1979.

HAM75 M.M.HAMMER,D.J.MCLEOD

"Semantic integrity in a relation data base system",

Proc. I-VLDB Conf., Framingham, Mass., Sept. 1975.

HAS78 M.M.HAMMER,S.SARIN
"Efficient monitoring of database assertions", Proc. of ACM SIGMOD 78 Int. Conf. on Management of Data, Juin 1978.

ISO79 ISO. TC97.SC16.N117
"Reference model of Open System Architecture", Version-3.

KAN79 R.K.KANODIA,D.P.REED
"Synchronisation with eventcount and sequencers", Comm. ACM, Vol 22, No 2, Fevrier 1979.

LAM76 L.LAMPORT
"Time, clocks and the ordering of events in a distributed system", Massachussetts Computer Associates Report CA-7603-2911, Mars 1976.

LEO76 M.LEONARD
"Aides algorithmiques à la conception de bases de données", Thèse Docteur Ingenieur, Université de Grenoble, Juin 1976.

LEL78 M.LEONARD,B.T.LUONG
"Approche de la conception des systèmes d'informations intégrant les données et les traitements", INFORSID, Cergy-Pontoise, Octobre 1978.

LIN79 B.G.LINDSAY et al.
"Notes on Distributed Databases", IBM Research Report RJ2571, San Jose, Juin 1979.

MIC80 F.FERNANDEZ,L.FERRAT,NGUYEN GIA TOAN
"MICROBE: Un système de base de données relationnelle répartie sur une réseau d'ordinateurs", Congrès AFCET, Nancy, Octobre 1980.

NGU77 NGUYEN GIA TOAN
"URANUS, une approche relationnelle à la coopération de bases de données", Thèse Docteur 3ème Cycle, Université de Grenoble, Decembre 1977.

NGU78 NGUYEN GIA TOAN
"L'adaptabilité des bases de données par les moyens relationnels", Congrès AFCET Informatique 78, Gif S/Yvette, Novembre 1978.

NGU79 NGUYEN GIA TOAN
"A unified method for query de composition and shared information updating in distributed systems", 1st Int. Conf. on Distributed Computing Systems, Hunstville, Alabama, Octobre 1979.

NGU80 NGUYEN GIA TOAN
"Decentralized dynamic query decomposition for distributed database systems", ACM Pacific 80 Conf., San Francisco, Novembre 1980.

OSM79 I.M.OSMAN

"Updating defined relations", Proc. National Computer Conf., New York, Juin 1979.

PAD79 IN-SUP PAIK, C.DELOBEL

"A strategy for optimizing the distributed query processing", lst Int. Conf. on Distributed Computing Systems, Hunstville, Alabama, Octobre 1979.

PAI78 IN-SUP PAIK

"Définition et réalisation d'un ensemble d'opérateurs relationnels pour une machine POLYPHEME", Rapport DEA, Université de Grenoble, septembre 1978.

PAP77 G.PELAGATTI, P.PAOLINI, G.BRACCHI

"Mappings in database systems", Information Processing 77, North-Holland Pub. Co., Amsterdam, 1977.

PAP78 P.PAOLINI, G.PELAGATTI

"Formal definition of mappings in a database", Proc. ACM SIGMOD Int. Conf. on Management of Data, Toronto, Août 1978.

PAR78 D.S.PARKER

"Some notes on distributed database optimization", Note Technique 34, Projet POLYPHEME, Decembre 1978.

PIS78 E.PICHAT, A.STIERS

"Spécifications générales de la maquette POLYPHEME.

Séminaire Prélenfrey", Note Technique 18, Projet POLYPHEME, Janvier 1978.

POL79 EQUIPE POLYPHEME
"Rapport final du projet POLYPHEME", CII-Honeywell Bull et laboratoire IMAG, Grenoble, Octobre 1979.

PPB78 G.PELAGATTI,P.PAOLINI,G.BRACCHI
"Mapping external views to a common data model", Information Systems, Vol 3, No 2, 1978.

RIC78 H.RICHY
"Confidentialité, bases de données et réseaux d'ordinateurs", Thése de Docteur-Ingenieur, INP Grenoble, Fevrier 1978.

RID73 J.RISSANEN,C.DELOBEL
"Decomposition of files. A basis for data storage and retrieval", IBM Research Report RJ1220, Jan Jose, Ca., Mai 1973.

RIS77 J.RISSANEN
"Independent components of relations", ACM Transactions on Database Systems, Vol 2, No 4, Decembre 1977.

ROS78 D.J.ROSENCRANTZ,R.E.STEARNS,P.W.LEWIS
"System level concurrency control for distributed database systems", ACM Transactions on Database Systems, Vol 3, No 2, Juin 1978, pp 178-198.

ROT80 J.B.ROTHNIE et al

"Introduction to a system for distributed databases (SDD-1)", ACM transactions on Database Systems, Vol 5, No 1, Mars 1980.

SCO80 EQUIPE SCOT

"SCOT: A system for the consistent cooperation of transactions", Rapports de recherche, Centre Scientifique CII-Honeywell Bull de Grenoble, Mai 1980.

SEF78 K.C.SEVCIK,A.L.FURTADO

"Complete and compatible sets of update operations", Int. Conf. on Management of Data (ICMOD), Milan, Juin 1978.

SET80 G.SERGEANT,L.TREILLE

"SER: a system for distributed execution based on decentralized control techniques", 5th Int. Conf. on Computer Communication, Atlanta, Octobre 1980.

SIR80 J.LE BIHAN,C.ESCULIER,G.LE LANN,L.TREILLE

"SIRIUS-DELTA: un prototype de système de gestion de bases de données réparties", Proc. Int. Symp. on Distributed Data Bases (C.Delobel,W.Litwin,editors), North-Holland Pub. Co., Mars 1980.

SMI77 J.M.SMITH,D.C.P.SMITH

"Database Abstractions: Aggregation and Generalization", ACM Trans. on Database Systems, Vol 1, No 2, Juin 1977, pp 105-133.

SOR79 J.L.CHEVAL et al.

"Un projet de système distribué: SORTILEGE", RR 150, Laboratoire IMAG, Janvier 1979.

STO75 M.R.STONEBRAKER

"Implementation of integrity constraints ans views by query modification", Proc. ACM SIGMOD Int. Conf. on Management of Data, San Jose, 1975, pp 65-78.

STO76 M.R.STONEBRAKER et al.

"The design and implementation of INGRES", ACM Trans. on Database Systems, Vol 1, No 3, Septembre 1976.

STO78 M.R.STONEBRAKER

"Concurrency control and consistency of multiple copies of data in distributed INGRES", 3th Berkeley Workshop on Distributed Data Management and Computer Network, Berkeley, Août 1978.

THO77 R.H.THOMAS

"A majority consensus approach to concurrency control for multiple data bases", Bolt Beranek and Newman Inc., Report 3733, Decembre 1977.

TOD77 S.TODD

"Automatic constraint maintenance and updating defined relations", Information Processing 77, North-Holland Pub. Co., Amsterdam, 1977, pp 145-148.

TRA78 I.L.TRAIGER,J.N.GRAY,C.A.GALTIERI,B.G.LINDSAY
"Transactions and consistency in distributed database systems", IBM research Report RJ2555, San Jose, Ca., Juin 1978.

VAS79 Y.VASSILIOU
"Null values in database management: A denotational semantics approach", Proc. ACM SIGMOD Int. Conf. on management of Data, Boston, Juin 1979, pp 162-169.

VER78 J.S.M.VERHOFSTAD
"Recovery techniques for Database Systems", ACM Computing Surveys, Vol 10, No 2, Juin 1978.

VIL80 M.VILON
"POLYPHEME: Application bancaire", Note interne Projet SIRIUS, 1980.

WIL79 P.WILMS
"Etudes d'algorithmes de cohérence d'informations dupliquées et réparties. Formalisation à l'aide des réseaux de Nutt", RR 160 et 160 bis, Laboratoire IMAG, Fevrier 1979.

WIL80 P.WILMS
"Qualitative an quantitative comparison of update algorithms in distributed databases", Proc. Int. Symp. on Distributed Databases,(C.Delobel,W.Litwin,editors), North-Holland Pub. Co., Mars 1980.

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Année universitaire 1979-1980

Président : M. Philippe TRAYNARD

Vice-Présidents : M. Georges LESPINARD
M. René PAUTHENET

PROFESSEURS DES UNIVERSITES

MM.	ANCEAU François	Informatique fondamentale et appliquée
	BENOIT Jean	Radioélectricité
	BESSON Jean	Chimie Minérale
	BLIMAN Samuel	Electronique
	BLOCH Daniel	Physique du Solide - Cristallographie
	BOIS Philippe	Mécanique
	BONNETAIN Lucien	Génie Chimique
	BONNIER Etienne	Métallurgie
	BOUVARD Maurice	Génie Mécanique
	BRISSONNEAU Pierre	Physique des Matériaux
	BUYLE-BODIN Maurice	Electronique
	CHARTIER Germain	Electronique
	CHERADAME Hervé	Chimie Physique Macromoléculaires
Mme	CHERUY Arlette	Automatique
MM.	CHIAVERINA Jean	Biologie, Biochimie, Agronomie
	COHEN Joseph	Electronique
	COUMES André	Electronique
	DURAND Francis	Métallurgie
	DURAND Jean-Louis	Physique Nucléaire et Corpusculaire
	FELICI Noël	Electrotechnique
	FOULARD Claude	Automatique
	GUYOT Pierre	Métallurgie Physique
	IVANES Marcel	Electrotechnique
	JOUBERT Jean-Claude	Physique du Solide - Cristallographie
	LACOUME Jean-Louis	Géographie - Traitement du Signal
	LANCIA Roland	Electronique - Automatique
	LESIEUR Marcel	Mécanique
	LESPINARD Georges	Mécanique
	LONGEQUEUE Jean-Pierre	Physique Nucléaire Corpusculaire
	MOREAU René	Mécanique
	MORET Roger	Physique Nucléaire Corpusculaire
	PARIAUD Jean-Charles	Chimie - Physique
	PAUTHENET René	Physique du Solide - Cristallographie
	PERRET René	Automatique

.../...

MM.	PERRET Robert	Electrotechnique
	PIAU Jean-Michel	Mécanique
	PIERRARD Jean-Marie	Mécanique
	POLOUJADOFF Michel	Electrotechnique
	POUPOT Christian	Electronique - Automatique
	RAMEAU Jean-Jacques	Chimie
	ROBERT André	Chimie Appliquée et des matériaux
	ROBERT François	Analyse numérique
	SABONNADIERE Jean-Claude	Electrotechnique
Mme	SAUCIER Gabrielle	Informatique fondamentale et appliquée
M.	SOHM Jean-Claude	Chimie - Physique
Mme	SCHLENKER Claire	Physique du Solide - Cristallographie
MM.	TRAYNARD Philippe	Chimie - Physique
	VEILLON Gérard	Informatique fondamentale et appliquée
	ZADWORNY François	Electronique

CHERCHEURS DU C.N.R.S. (Directeur et Maître de Recherche)

M.	FRUCHART Robert	Directeur de Recherche
MM.	ANSARA Ibrahim	Maître de Recherche
	BRONOEL Guy	Maître de Recherche
	CARRE René	Maître de Recherche
	DAVID René	Maître de Recherche
	DRIOLE Jean	Maître de Recherche
	KAMARINOS Georges	Maître de Recherche
	KLEITZ Michel	Maître de Recherche
	LANDAU Ioan-Doré	Maître de Recherche
	MERMET Jean	Maître de Recherche
	MUNIER Jacques	Maître de Recherche

Personnalités habilitées à diriger des travaux de recherche (décision du Conseil Scientifique)

E.N.S.E.E.G.

MM.	ALLIBERT Michel	
	BERNARD Claude	
	CAILLET Marcel	
Mme	CHATILLON Catherine	
MM.	COULON Michel	
	HAMMOU Abdelkader	
	JOUD Jean-Charles	
	RAVAINE Denis	
	SAINFORT	C.E.N.G.

.../...

MM. SARRAZIN Pierre
SOUQUET Jean-Louis
TOUZAIN Philippe
URBAIN Georges — Laboratoire des Ultra-Réfractaires ODEILLO

E.N.S.M.E.E.

MM. BISCONDI Michel
BOOS Jean-Yves
GUILHOT Bernard
KOBILANSKI André
LALAUZE René
LANCELOT Françis
LE COZE Jean
LESBATS Pierre
SOUSTELLE Michel
THEVENOT François
THOMAS Gérard
TRAN MINH Canh
DRIVER Julian
RIEU Jean

E.N.S.E.R.G.

MM. BOREL Joseph
CHEHIKIAN Alain
VIKTOROVITCH Pierre

E.N.S.I.E.G.

MM. BORNARD Guy
DESCHIZEAUX Pierre
GLANGEAUD François
JAUSSAUD Pierre
Mme JOURDAIN Geneviève
MM. LEJEUNE Gérard
PERARD Jacques

E.N.S.H.G.

M. DELHAYE Jean-Marc

E.N.S.I.M.A.G.

MM. COURTIN Jacques
LATOMBE Jean-Claude
LUCAS Michel
VERDILLON André

dernière page de la thèse

AUTORISATION DE SOUTENANCE

Vu les dispositions de l'article 3 de l'arrêté du 16 Avril 1974,

Vu le rapport de présentation de MM

C. DELOBEL Professeur sans chaire à l'U.S.M.-G

P. DECITRE Ingénieur à la CII Honeywell Bull

Monsieur Juan Manuel ANDRADE

est autorisé à présenter une thèse en soutenance pour l'obtention du diplôme de DOCTEUR-INGENIEUR, spécialité "Génie Informatique".

Fait à Grenoble, le 30 Septembre 1980

Le Président de l'I.N.P.-G

Ph. TRAYNARD.