50376
1982
59

N° d'ordre : 944

50376
1982
59

THÈSE

présentée à

L'UNIVERSITÉ DES SCIENCES ET TECHNIQUES DE LILLE

pour obtenir le titre de

DOCTEUR DE 3ème CYCLE

(INFORMATIQUE)

par

Consuelo FRANKY-TORO

CONTRIBUTION AU CONTROLE DE COHERENCE DES SYSTEMES TRANSACTIONNELS REPARTIS EN VUE DE FAVORISER L'UTILISATEUR INTERACTIF.

Thèse soutenue le 12 janvier 1982 devant la Commission d'Examen

Membres du Jury	V.CORDONNIER	Président
	C.CARREZ	Rapporteur
	M.DAUCHET	Examinateurs
	G.LE LANN	

PROFESSEURS 1ère CLASSE

M.	BACCHUS Pierre	Mathématiques
M.	BEAUFILS Jean-Pierre (dét.)	Chimie
M.	BIAYS Pierre	G.A.S.
M.	BILLARD Jean	Physique
M.	BONNOT Ernest	Biologie
M.	BOUGHON Pierre	Mathématiques
M.	BOURIQUET Robert	Biologie
M.	CELET Paul	Sciences de la Terre
M.	COEURE Gérard	Mathématiques
M.	CONSTANT Eugène	I.E.E.A.
M.	CORDONNIER Vincent	I.E.E.A.
M.	DEBOURSE Jean-Pierre	S.E.S.
M.	DELATTRE Charles	Sciences de la Terre
M.	DURCHON Maurice	Biologie
M.	ESCAIG Bertrand	Physique
M.	FAURE Robert	Mathématiques
M.	FOURET René	Physique
M.	GABILLARD Robert	I.E.E.A.
M.	GRANELLE Jean-Jacques	S.E.S.
M.	GRUSON Laurent	Mathématiques
M.	GUILLAUME Jean	Biologie
M.	HECTOR Joseph	Mathématiques
M.	HEUBEL Joseph	Chimie
M.	LABLACHE COMBIER Alain	Chimie
M.	LACOSTE Louis	Biologie
M.	LANSRAUX Guy	Physique
M.	LAVEINE Jean-Pierre	Sciences de la Terre
M.	LEBRUN André	C.U.E.E.P.
M.	LEHMANN Daniel	Mathématiques
Mme	LENOBLE Jacqueline	Physique
M.	LHOMME Jean	Chimie
M.	LOMBARD Jacques	S.E.S.
M.	LOUCHEUX Claude	Chimie
M.	LUCQUIN Michel	Chimie

M.	MAILLET Pierre	S.E.S.
M.	MONTREUIL Jean	Biologie
M.	PAQUET Jacques	Sciences de la Terre
M.	PARREAU Michel	Mathématiques
M.	PROUVOST Jean	Sciences de la Terre
M.	SALMER Georges	I.E.E.A.
Mme	SCHWARTZ Marie-Hélène	Mathématiques
M.	SEGUIER Guy	I.E.E.A.
M.	STANKIEWICZ François	Sciences Economiques
M.	TILLIEU Jacques	Physique
M.	TRIDOT Gabriel	Chimie
M.	VIDAL Pierre	I.E.E.A.
M.	VIVIER Emile	Biologie
M.	WERTHEIMER Raymond	Physique
M.	ZEYTOUNIAN Radyadour	Mathématiques

PROFESSEURS 2ème CLASSE

M.	AL FAKIR Sabah	Mathématiques
M.	ANTOINE Philippe	Mathématiques
M.	BART André	Biologie
Mme	BATTIAU Yvonne	Géographie
M.	BEGUIN Paul	Mathématiques
M.	BELLET Jean	Physique
M.	BKOUCHE Rudolphe	Mathématiques
M.	BOBE Bernard	S.E.S.
M.	BODART Marcel	Biologie
M.	BOILLY Bénoni	Biologie
M.	BONNELLE Jean-Pierre	Chimie
M.	BOSQ Denis	Mathématiques
M.	BREZINSKI Claude	I.E.E.A.
M.	BRUYELLE Pierre (Chargé d'enseignement)	Géographie
M.	CAPURON Alfred	Biologie
M.	CARREZ Christian	I.E.E.A.
M.	CHAMLEY Hervé	E.U.D.I.L.

M.	CHAPOTON Alain	C.U.E.E.P.
M.	COQUERY Jean-Marie	Biologie
Mme	CORSIN Paule	Sciences de la Terre
M.	CORTOIS Jean	Physique
M.	COUTURIER Daniel	Chimie
Mle	DACHARRY Monique	Géographie
M.	DEBRABANT Pierre	E.U.D.I.L.
M.	DEGAUQUE Pierre	I.E.E.A.
M.	DELORME Pierre	Biologie
M.	DEMUNTER Paul	C.U.E.E.P.
M.	DE PARIS Jean-Claude	Mathématiques
M.	DEVRAINNE Pierre	Chimie
M.	DHAINAUT André	Biologie
M.	DORMARD Serge	S.E.S.
M.	DOUKHAN Jean-Claude	E.U.D.I.L.
M.	DUBOIS Henri	Physique
M.	DUBRULLE Alain	Physique
M.	DUEE Gérard	Sciences de la Terre
M.	DYMENT Arthur	Mathématiques
M.	FLAMME Jean-Marie	E.U.D.I.L.
M.	FONTAINE Hubert	Physique
M.	GERVAIS Michel	S.E.S.
M.	GOBLOT Rémi	Mathématiques
M.	GOSSELIN Gabriel	S.E.S.
M.	GOUDMAND Pierre	Chimie
M.	GREVET Patrice	S.E.S.
M.	GUILBAULT Pierre	Biologie
M.	HANGAN Théodore	Mathématiques
M.	HERMAN Maurice	Physique
M.	JACOB Gérard	I.E.E.A.
M.	JACOB Pierre	Mathématiques
M.	JOURNEL Gérard	E.U.D.I.L.
M.	KREMBEL Jean	Biologie
M.	LAURENT François	I.E.E.A.
Mle	LEGRAND Denise	Mathématiques
Mle	LEGRAND Solange	Mathématiques (Calais)
Mme	LEHMANN Josiane	Mathématiques

M. LEMAIRE Jean	Physique
M. LENTACKER Firmin	G.A.S.
M. LEVASSEUR Michel	I.P.A.
M. LHENAFF René	G.A.S.
M. LOCQUENEUX Robert	Physique
M. LOSFELD Joseph	I.E.E.A.
M. LOUAGE Francis	E.U.D.I.L.
M. MACKE Bruno	Physique
M. MAIZIERES Christian	I.E.E.A.
Mle MARQUET Simone	Mathématiques
M. MESSELYN Jean	Physique
M. MICEON Michel	E.U.D.I.L.
M. MIGNOT Fulbert	Mathématiques
M. MONTEL Marc	Physique
Mme NGUYEN VAN CHI Régine	G.A.S.
M. PARSY Fernand	Mathématiques
Mle PAUPARDIN Colette	Biologie
M. PERROT Pierre	Chimie
M. PERTUZON Emile	Biologie
M. PONSOLLE Louis	Chimie
M. PORCHET Maurice	Biologie
M. POVY Lucien	E.U.D.I.L.
M. RACZY Ladislas	I.E.E.A.
M. RICHARD Alain	Biologie
M. RIETSCH François	E.U.D.I.L.
M. ROGALSKI Marc	M.P.A.
M. ROUSSEAU Jean-Paul	Biologie
M. ROY Jean-Claude	Biologie
M. SALAMA Pierre	S.E.S.
Mme SCHWARZBACH Yvette (CCP)	M.P.A.
M. SCHAMPS Joël	Physique
M. SIMON Michel	S.E.S.
M. SLIWA Henri	Chimie
M. SOMME Jean	G.A.S.
Mle SPIK Geneviève	Biologie
M. STERBOUL François	E.U.D.I.L.
M. TAILLIEZ Roger	Institut Agricole

M.	TOULOTTE Jean-Marc	I.E.E.A.
M.	VANDORPE Bernard	E.U.D.I.L.
M.	WALLART Francis	Chimie
M.	WATERLOT Michel	Sciences de la Terre
Mme	ZINN JUSTIN Nicole	M.P.A.

CHARGES DE COURS

M.	TOP Gérard	S.E.S.
M.	ADAM Michel	S.E.S.

CHARGES DE CONFERENCES

M.	DUVEAU Jacques	S.E.S.
M.	HOFLACK Jacques	I.P..A
M.	LATOUCHE Serge	S.E.S.
M.	MALAUSSENA DE PERNO Jean-Louis	S.E.S.
M.	OPIGEZ Philippe	S.E.S.

AVANT PROPOS

Ce travail de recherche a été réalisé au Laboratoire d'Informatique de l'Université des Sciences et Techniques de Lille I.

Je remercie Monsieur le Professeur Vincent CORDONNIER pour m'avoir accueilli dans son Laboratoire et pour me faire l'honneur de présider le Jury de cette thèse.

Je remercie également Monsieur Christian CARREZ, Professeur à l'Université de Lille I, qui m'a dirigé tout au long de ce travail et dont les remarques et conseils m'ont été précieux.

Mes remerciements vont aussi à Monsieur Max DAUCHET, Professeur à l'Université de Lille I, pour avoir eu l'amabilité de participer au Jury.

Je remercie Monsieur Gérard LE LANN, Directeur de Recherches à l'INRIA, qui a accepté d'examiner ce manuscrit et d'autres documents préliminaires et qui me fait l'honneur de participer au Jury. Les documents que j'ai pu me procurer au sein du groupe CORE que Monsieur LE LANN dirige (dans le cadre du projet SIRIUS) m'ont été précieux pour l'orientation et l'élaboration de ce travail.

Je tiens à remercier également Mademoiselle Bénédicte FIEVET et Madame Françoise TAILLY pour la gentillesse et le soin qu'elles ont apportés à la mise en pages de cette thèse, ainsi que Monsieur et Madame DEBOCK qui ont assuré le tirage de ce document avec beaucoup de diligence.

Enfin, il m'est agréable de remercier ici Monsieur BARFETY, Directeur du CROUS de Lille, Madame CROSS et toutes les personnes du Service d'Accueil d'étudiants étrangers qui ont tout fait pour faciliter et rendre agréable mon séjour en FRANCE. Avec eux, je voudrais remercier aussi tous les amis que j'ai rencontré dans le Nord : qu'ils sachent que grâce à eux ces années en FRANCE ont été pour moi une expérience très enrichissante et positive dont je garderai toujours un chaleureux souvenir.

A Victor Manuel

AVERTISSEMENT AU LECTEUR

Le chapitre I contient des concepts largement connus dans le domaine des systèmes informatiques répartis, lesquels constituent le cadre de référence de ce travail. Le chapitre III décrit les approches conventionnelles utilisées pour faire face aux problèmes de contrôle de cohérence dans les systèmes répartis. Pour le lecteur spécialiste du sujet ces 2 chapitres I et III paraîtront un peu longs et sans apports originaux: dans ce cas nous proposons de lire très rapidement ces chapitres, de lire avec un peu plus d'attention le chapitre II et d'aller directement à la partie originale et fondamentale de la thèse: les chapitres IV et V.

Pour le lecteur non spécialiste du sujet des systèmes répartis nous conseillons de lire avec attention tous les chapitres, même les premiers (lesquels définissent les concepts utilisés par la suite).

TABLE DES MATIERES

INTRODUCTION

Durant ces dernières années on a assisté à un fort développement des systèmes informatiques nommés "répartis", caractérisés par la répartition des fonctions de stockage et de traitement des données. Ce phénomène s'explique principalement par deux raisons [CHAMP 80, Chapitre 1, ; CORNA 81, Chapitre 1] :

a) Nécessité d'une meilleure adaptation des systèmes informatiques aux structures des organisations et aux besoins de l'utilisateur ; d'une part, un changement d'objectif est intervenu dans l'évolution des systèmes informatiques : au lieu de chercher une meilleure puissance de calcul, on préfère améliorer la disponibilité de l'information à l'utilisateur ce qui devrait augmenter l'efficacité de l'organisation ; d'autre part, avec le développement des organisations, les utilisateurs se sont dispersés et éloignés du système central. Ces deux situations constituent une motivation pour la répartition du stockage et du traitement des données afin de les rapprocher du site géographique de l'utilisateur qui est l'origine ou le consommateur de ces données.

b) Evolution de la technologie : d'une part, les progrès des composants intégrés ont entraîné une baisse du coût et une meilleure performance des processeurs et des mémoires ; d'autre part, le développement des moyens de communication ont facilité l'échange de l'information entre équipements informatiques éloignés. En fait, les coûts de stockage et de traitement de l'information ont diminué deux fois plus vite que les coûts de communication [CHAMP 77], ce qui rend plus intéressante la multiplication des capacités de traitement et de stockage, que celle des communications.

Des applications réalisées sur les systèmes informatiques répartis, on attend principalement les bénéfices suivants [ENSLO 78] :

- une meilleure fiabilité et résistance du système aux défaillances,
- modularité facilitant la mise au point, la reconfiguration, et l'extension du système,
- décentralisation des données et des décisions,
- réduction des coûts de communications.

En contrepartie, dans les systèmes répartis on est confronté à des problèmes plus complexes que dans les systèmes centralisés, tels la sécurité et le contrôle d'accès aux données dispersées, la synchronisation des mises à jour concurrentes, la compatibilité entre des environnements indépendants et la complexité dans la conception, mise en oeuvre et opération du système. En fait, et d'après Le Lann [LELAN 79 C], le problème fondamental introduit par les systèmes à contrôle réparti provient de l'existence de plusieurs contrôleurs qui peuvent observer des vues différentes et incohérentes de l'état global du système : les décisions prises par ces contrôleurs peuvent alors entrer en conflit et détruire la cohérence des données du système.

Il est donc nécessaire d'établir pour les systèmes répartis des nouvelles techniques de contrôle qui ne soient pas de simples extensions des techniques utilisées dans les systèmes centralisés (en fait, ces techniques centralisées doivent constituer des cas particuliers des techniques plus générales nécessitées par les systèmes répartis).

- * - * - * -

Le travail présent essaie de caractériser et de modéliser le contrôle de cohérence dans les systèmes informatiques répartis, sujet dont l'importance est justifiée par les considérations antérieures et qui constitue un des principaux axes de recherche dans le domaine des systèmes ré-

partis. Un tel contrôle de cohérence signifie en dernière instance le maintien de l'intégrité des données réparties en environnement partagé et non fiable.

Plus précisément, nous voulons établir des techniques de contrôle réparti de cohérence qui favorisent l'utilisateur interactif du système : nous allons considérer fondamentalement les facilités et les aspects "utilisateur" dans le cadre de transactions interactives qui exigent une réponse rapide. Nous considérons comme très important de donner à l'utilisateur, diverses options de contrôle lors de concurrence de transactions et de défaillances du système afin qu'il ne devienne pas un "esclave" des décisions du système ; de telles options doivent comprendre notamment : la possibilité de retour à un état cohérent antérieur, des alternatives en cas de défaillances du système, un verrouillage flexible des ressources, etc. Nous estimons que de tels aspects "utilisateur" sont quelquefois négligés dans les recherches sur les systèmes répartis, et que les mécanismes existants de contrôle réparti de cohérence, malgré leur efficacité, sont rigides envers l'utilisateur.

- * - * - * -

Les chapitres de ce travail comprennent successivement les sujets suivants :

- Dans le Chapitre I nous caractérisons les systèmes informatiques répartis et leurs problèmes de contrôle de cohérence.

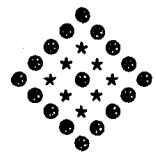
- Dans le Chapitre II nous caractérisons l'utilisateur interactif des systèmes répartis en relation avec les facilités et les options que nous voulons lui donner.

- Dans le Chapitre III nous présentons un premier modèle qui permet de résoudre les problèmes de contrôle de cohérence posés par l'accès à des fichiers répartis de données par des transactions

concurrentes, lesquelles peuvent être affectées par des défaillances du système. Ce modèle est basé sur la réalisation de transactions atomiques à l'aide de listes d'intentions.

- Dans le Chapitre IV nous modifions le premier modèle afin de favoriser la concurrence entre transactions et d'éviter à l'utilisateur les attentes trop longues des ressources. On utilise dans ce modèle un mécanisme flexible de verrouillage qui permet de choisir la taille de l'objet à verrouiller. De plus, l'utilisation de transactions "imbriquées" va permettre la libération graduelle de ressources pendant une transaction, et va éviter à l'utilisateur la perte totale de ses opérations en cas de défaillances du système.

- Dans le Chapitre V nous généralisons le modèle de contrôle de cohérence à un système réparti d'objets abstraits avec des verrous généralisés. Ceci permet d'étendre le domaine d'application des résultats obtenus à diverses catégories de systèmes informatiques répartis, notamment celui des Bases de Données Réparties.



CHAPITRE I

CONCEPTS RELATIFS AUX SYSTEMES REPARTIS ET A LEURS PROBLEMES DE CONTROLE DE COHERENCE

Il existe une prolifération de recherches et une littérature très abondante dans le domaine des systèmes répartis, ce qui a donné lieu à une grande variété de notions et de définitions qui ne sont pas les mêmes pour tous les auteurs. Ainsi, il n'existe pas un accord général sur la définition du concept de "Système Informatique Réparti".

Ce chapitre veut être une synthèse des principales approches qui existent dans la littérature concernant les systèmes répartis, afin de dégager les concepts relevant du Contrôle de Cohérence et d'établir un cadre de référence pour les développements qui seront faits dans les chapitres suivants.

Dans la première partie de ce chapitre, nous voulons dégager une caractérisation globale des systèmes informatiques répartis. Ensuite, dans la deuxième partie, nous voulons délimiter les problèmes de contrôle de cohérence auxquels on doit faire face dans ces systèmes, notamment le maintien de la cohérence des informations lors de concurrence de transactions, et lors de défaillances du système ; pour cela il est nécessaire d'établir clairement les concepts de "Cohérence" et de "Transaction". Dans le reste du chapitre (parties 3, 4 et 5), nous présentons les stratégies générales pour résoudre de tels problèmes de contrôle de cohérence.

1 - CARACTERISATION DES SYSTEMES REPARTIS

1.1 - Conditions d'existence d'un système réparti

Un Système Informatique Réparti peut être défini comme un ensemble de sites géographiques distincts munis de capacités de stockage et de traitement de données, et qui communiquent entre eux au moyen d'un réseau de communication afin de réaliser le traitement d'applications de façon coopérative.

De cette définition, nous pouvons déduire les conditions que doit remplir un système informatique pour être nommé "réparti" [ENSLO 78 ; CHAMP 80, Chapitre 2] :

- Il doit avoir une multiplicité de ressources-composants (physiques et logiques) répartis géographiquement sur plusieurs sites.

- Il doit exister la nécessité d'intéraction entre les sites pour partager les ressources.

- L'interaction entre les sites doit être réalisée par l'échange de messages au moyen d'un réseau de communication.

- Le contrôle du système doit être à la fois réparti et unifié : chaque site doté de capacités propres de traitement et de stockage de données, doit opérer selon un ensemble de règles bien définies pour assurer une autonomie coopérative entre les sites.

Il existe une grande diversité d'organisations possibles pour un système informatique réparti, selon les options prises en matière de répartition du matériel (composants physiques), de répartition du traitement et de répartition des données. Ces différentes options vont être examinées dans les sections suivantes.

1.2 - Le transport de l'information

Pour faciliter la conception et la réalisation d'un système informatique réparti, il est utile de séparer les fonctions de transport et de traitement de l'information [MACCH 79] : le système est alors représenté comme un ensemble de sites de traitement communiquant au travers d'un Sous-système de Transport chargé d'acheminer l'information sans la traiter. Un tel Sous-système de Transport est constitué par deux types de composants (voir Fig. 1) :

- Un "Réseau de Transmission de données" ou ensemble de composants physiques spécialisés dans le transport de l'information.

- Des "Stations de Transport" ou composants logiques situés à l'intérieur des sites, chargés de faire l'interface entre le réseau de transmission et le Sous-système de traitement.

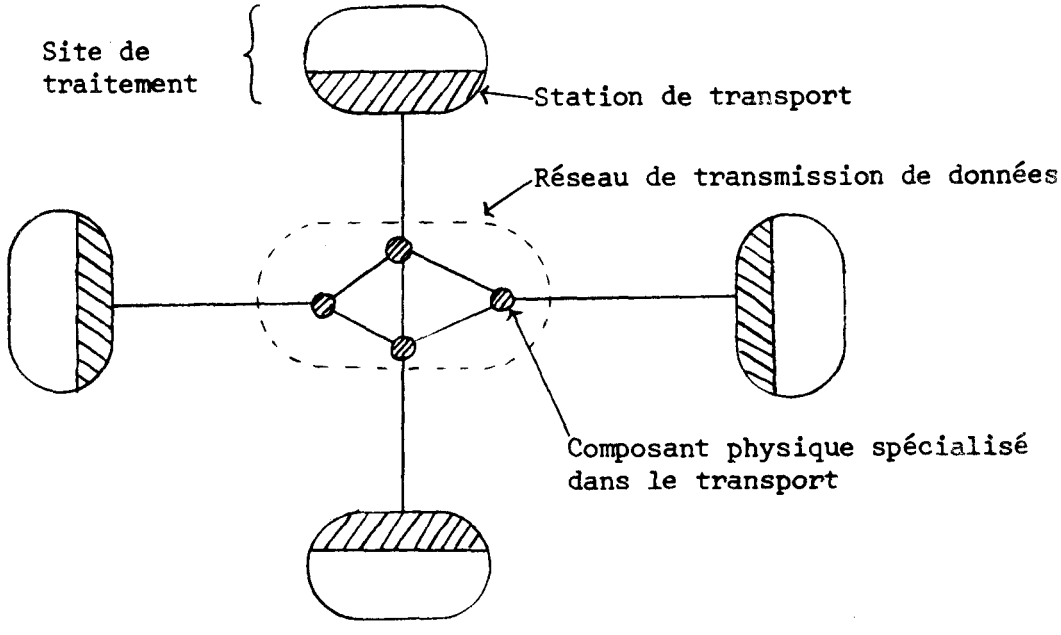


Fig. 1 : Structure du Sous-système de Transport.

1.2.1 - *Hypothèses sur le Sous-système de Transport*

Le Sous-système de Transport doit offrir aux sites de traitement un service *indépendant* des moyens et réseaux de télécommunication qu'il

utilise (tels les réseaux ARPANET, TRANSPAC, CYCLADES, etc...) ; en fait, ce sont les Stations de Transport qui doivent assurer aux sites de traitement la transparence aux différences présentées par les réseaux.

Du point de vue du Sous-système de Traitement, on demande au Sous-système de Transport la vérification de certaines hypothèses de base sur la transmission de messages, hypothèses qui facilitent la réalisation d'algorithmes de contrôle réparti des applications. Des hypothèses souvent adoptées dans la littérature sont les suivantes [WILLS 79, Chapitre 1 ; CORNA 81, Chapitre 8 ; SCOT 81 B ; MACCH 79, Chapitre 6] :

- Le délai de transmission d'un message d'un site à un autre est variable mais borné.

- Entre deux sites du système l'ordre de réception de messages est identique à l'ordre d'émission (mais on ne peut rien supposer de l'ordre de réception de messages envoyés à un site à partir de plusieurs autres sites).

- Il n'y a pas de perte ni duplication de messages : cette hypothèse peut être obtenue par le Sous-système de Transport en utilisant une méthode de numérotation séquentielle des messages et en exigeant un accusé de réception pour chaque message envoyé ; ces mécanismes permettent de détecter la perte ou la duplication d'un message et de corriger l'erreur en retransmettant le message manquant ou en ignorant le message en double.

- Il n'y a pas d'erreur de transmission qui altère le contenu des messages : cette hypothèse peut être obtenue en utilisant une information de redondance qui accompagne le message, et qui permet de détecter et de corriger l'erreur par retransmission du message.

- La panne d'un site (qui se traduit par son isolement dans le réseau) est détectée et signalée aux autres sites qui tentent

de communiquer avec le site défaillant : le Sous-système de Transport peut considérer un site inaccessible s'il y a manque d'acquittement à un message après plusieurs tentatives et dans un délai fixe.

1.2.2 - *Asynchronisme et observation d'évènements*

Dans un système réparti la nature de la transmission est telle que le temps moyen de transfert d'une information entre deux sites est variable et grand par rapport au temps qui peut séparer l'occurrence de deux évènements sur un même site. Il en résulte une caractéristique intrinsèque des systèmes répartis [CORNA 81, Chapitre 1] : à un instant donné, un site ne peut connaître que de manière partielle l'état d'un autre site à cet instant ; aussi, il est impossible pour un site de connaître l'état global de tout le système.

Cette caractéristique constitue la complexité fondamentale dans le contrôle des systèmes répartis : en effet, la connaissance partielle et différente de **l'état global du système que possède chaque site**, peut conduire à des décisions contradictoires prises par les différents sites.

1.2.3 - *Configuration de base d'interconnexion entre sites*

L'interconnexion matérielle entre les sites d'un système réparti au moyen d'un réseau de communication, peut prendre différentes formes d'après l'organisation désirée pour le flux de messages. Champine [CHAMP 80, Chapitre 7] différencie six configurations possibles d'interconnexion entre sites :

- *En étoile* (voir Fig. 2) : dans cette configuration il existe un site central auquel sont connectés tous les autres sites ; le contrôle du flux de messages est à la charge du site central.

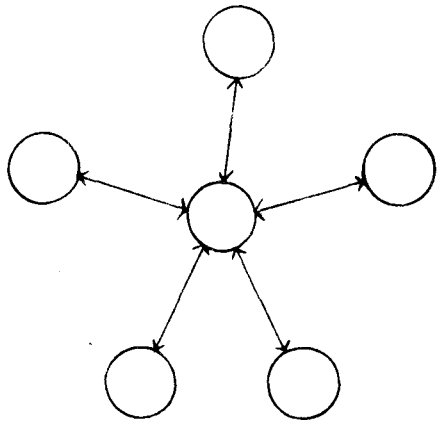


Fig. 2 : Configuration "en étoile".

- *En arbre* (voir Fig. 3) : cette configuration est une extension de la configuration "en étoile" où la responsabilité dans le contrôle du flux de messages est plus importante dans le site racine et décroît à chaque niveau qui s'éloigne de la racine.

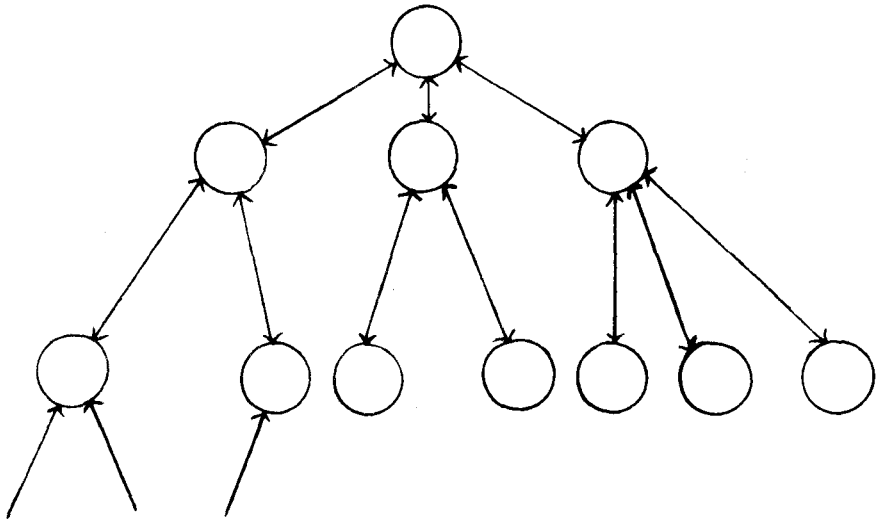


Fig. 3 : Configuration "en arbre".

- *En anneau* (voir Fig. 4) : dans cette configuration les sites sont connectés dans une chaîne ; ainsi chaque site est connecté directement à deux sites adjacents, et possède le même niveau de responsabilité dans le contrôle du flux de messages.

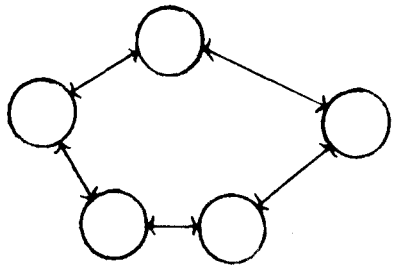


Fig. 4 : Configuration "en anneau".

- *Linéaire* (voir Fig. 5) : cette configuration correspond à une configuration en anneau ouverte.

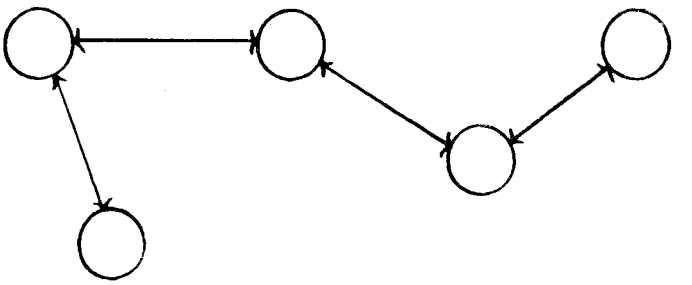


Fig. 5 : Configuration "linéaire".

- *Totalement connectée* (voir Fig. 6) : dans cette configuration les sites sont totalement connectés entre eux, et chacun est également responsable dans le contrôle du flux de messages. Cette configuration peut être considérée comme une extension de la configuration en anneau puisque chaque chemin fermé dans le réseau constitue une structure en anneau.

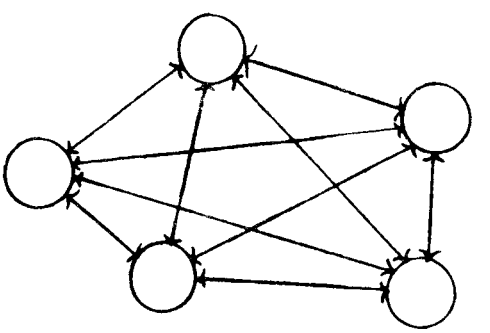


Fig. 6 : Configuration "totalement connectée".

- *Le réseau général* (voir Fig. 7) : c'est la configuration généralisée d'interconnexion construite à partir des autres configurations.

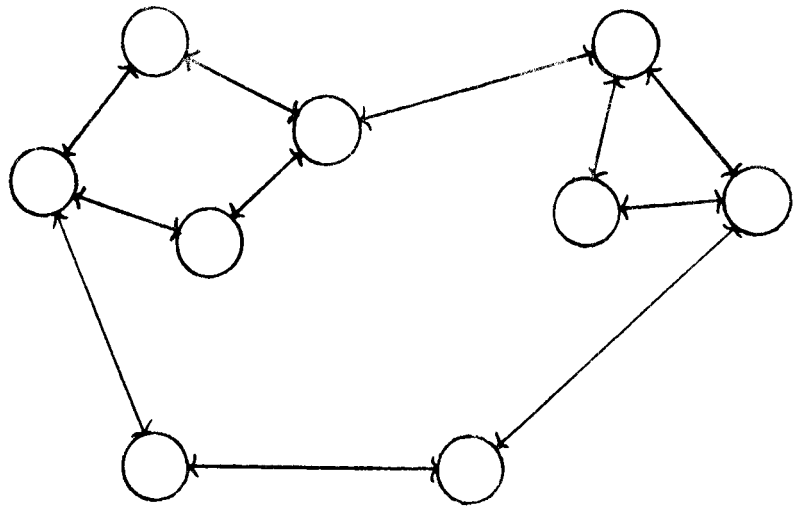


Fig. 7 : Configuration de "réseau général".

Les différentes configurations d'interconnexion présentent des avantages et des inconvénients variés quant à la performance, la fiabilité et le coût dans la transmission de messages. Ainsi, les structures centralisées (étoile, arbre) offrent une meilleure performance que les structures décentralisées (anneau, linéaire) à cause de l'absence d'interférence (et de partage) entre les lignes de connexion, mais au même temps elles entraînent un coût plus élevé. La configuration totalement connectée offre une fiabilité supérieure aux autres configurations à cause de la redondance de chemins, mais son coût est trop élevé (puisque le nombre de lignes s'accroît proportionnellement au carré du nombre de sites).

1.3 - Options pour la répartition du traitement

Conceptuellement le traitement réparti est naturellement dirigé par des messages [ZIMME 81] : on peut représenter l'exécution d'une activité répartie (i.e. réalisée de façon coopérative par les sites d'un système réparti) par un graphe constitué par des étapes de traitement connectées par des messages (voir Fig. 8), de telle façon que chaque étape de traitement soit déclenchée par la réception d'un message et qu'elle se termine par l'envoi d'un ou plusieurs messages.

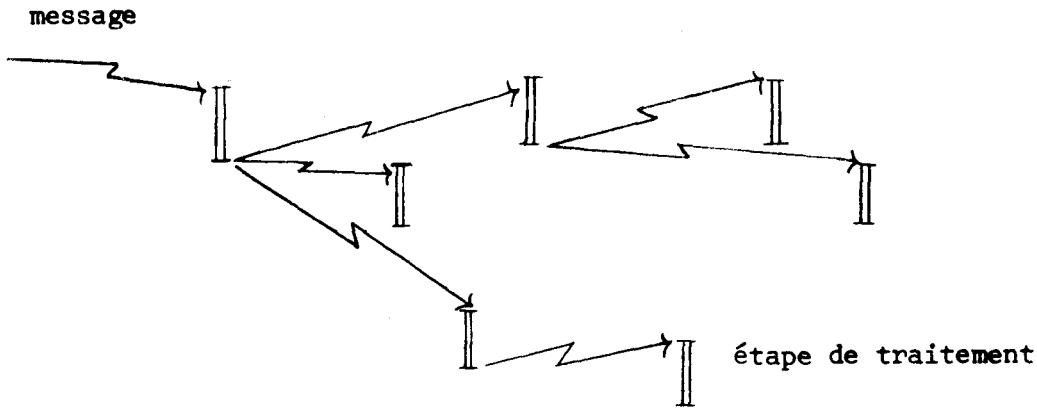


Fig. 8 : Graphe de l'exécution d'une activité répartie.

Dans un système réparti, la façon de gérer le contrôle de l'exécution des transactions soumises par les utilisateurs conduit à définir plusieurs options de répartition du traitement ou structures fonctionnelles. Nous considérons ici la transaction comme l'unité d'activité répartie, mais ce concept sera précisé dans la partie 2 de ce chapitre.

La structure fonctionnelle d'un système réparti n'est pas forcément déterminée par la structure matérielle d'interconnexion entre les sites : on peut avoir un système réparti avec une configuration d'interconnexion décentralisée mais fonctionnellement centralisé.

On peut distinguer principalement quatre schémas fonctionnels possibles pour un système réparti, d'après Wills [WILLS 79, Chapitre 5] :

- *Schéma de Contrôle Centralisé* : le contrôle de l'exécution des transactions est à la charge d'un seul site dit "central" ; toutes les transactions initialisées sur les autres sites lui sont communiquées et mises dans une file d'attente unique. Le site central réalise les transactions séquentiellement dans l'ordre d'arrivée et pour cela il envoie les messages nécessaires aux autres sites. Dans cette structure il existe deux possibilités : le contrôle centralisé *statique* où la fonction de contrôle est assignée de manière fixe à un site particulier, et le contrôle centralisé *dynamique* où chaque site doit obtenir le droit de contrôle avant de lancer l'exécution d'une transaction.

- *Schéma de Contrôle Local* : Chaque site exécute les transactions de manière indépendante : un site qui désire lancer l'exécution d'une transaction envoie les messages nécessaires aux autres sites ; au niveau de chaque site, un mécanisme de files d'attente doit permettre l'ordonnancement des messages reçus provenants des différentes transactions, afin d'exécuter ces dernières de manière séquentielle.

- *Schéma de Contrôle Partiellement Réparti* : à chaque transaction est associé un site "responsable" qui a la charge de dialoguer avec les autres sites pour décider sur l'acceptation ou le rejet de la nouvelle transaction. Dans ce schéma il y a deux possibilités : une fois qu'une nouvelle transaction a été acceptée, elle peut être exécutée avec ou sans possibilité de parallélisme avec d'autres transactions.

- *Schéma de Contrôle Totalement Réparti* : à chaque transaction est associé un processus unique qui visite tous les sites pour obtenir un accord sur l'acceptation ou le rejet de la transaction. Une fois la transaction acceptée, son processus commence une deuxième étape pour exécuter la transaction même, et pour cela il y a aussi deux possibilités : avec ou sans parallélisme avec d'autres transactions.

1.4 - Options pour la répartition des données

1.4.1 - Partition ou Duplication des données

On considère trois alternatives de base pour répartir les données dans un système informatique réparti [ROTHN 77] :

- *La Partition sans redondance* : l'ensemble total des données du système est divisé en des ensembles disjoints et chaque ensemble est assigné à un site distinct ; l'objectif usuel est d'assigner chaque ensemble de données au site qui lui accède plus fréquemment afin de minimiser le temps de répon-

se et le nombre de transferts de communication causés par les accès aux données non locales à un site.

- *La Duplication totale* : l'ensemble total des données est dupliqué sur chaque site ; on obtient ainsi une meilleure fiabilité puisque grâce à la redondance, l'indisponibilité des données d'un site (suite à une panne) n'est pas si grave que dans l'alternative de Partition. Mais en même temps, les mises à jour des données deviennent plus complexes et coûteuses puisque chaque modification doit être répercutée sur l'ensemble des copies afin de maintenir la même version entre ces copies ; ainsi, la duplication n'est avantageuse que si le nombre de mises à jour est très faible par rapport au nombre de consultations. En plus, un grand volume de l'ensemble des données peut rendre prohibitif sa duplication totale.

- *La Duplication partielle* : pour chaque donnée il existe une ou plusieurs copies assignées à différents sites. Cette alternative offre une meilleure flexibilité dans le choix du degré de redondance désiré pour le système : les données plus fréquemment accédées peuvent être dupliquées sur plusieurs sites à stockage rapide tandis que les autres peuvent résider à une seule copie sur stockage lent mais moins cher.

Dans un système réparti, la sélection d'une alternative de répartition des données doit dépendre des caractéristiques de l'application à mettre en place telles : le volume des données manipulées, la fréquence des mises à jour, la fréquence des accès aux données non locales (i.e. taux d'exception), la disponibilité et fiabilité désirées, etc. Pour les trois premiers critères antérieurs, Champine [CHAMP 80, Chapitre 3] considère l'arbre de décisions suivant :

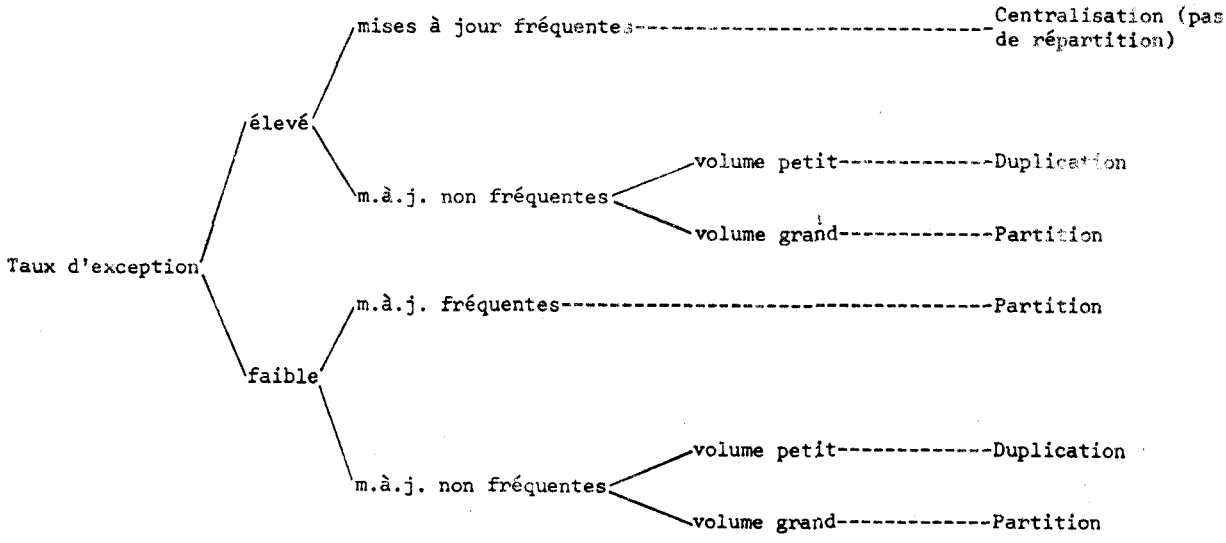


Fig. 9 : Sélection d'une alternative de répartition des données.

Les répertoires d'un système réparti (contenant l'information qui associe aux données les sites où elles résident) peuvent être répartis à leur tour et pas nécessairement de la même façon que les données. Ces répertoires présentent généralement les caractéristiques de : taux d'accès élevé, mises à jour peu fréquentes, une surcharge faible en stockage et haute disponibilité et fiabilité nécessitées ; il semble alors que la duplication totale des répertoires soit l'approche la plus appropriée [WILLS 79, Chapitre 1 ; CHAMP 80, Chapitre 3]. Il faut noter que, dans le cas de données totalement dupliquées, aucun répertoire n'est nécessaire parce que tous les accès dans un site deviennent locaux.

Le problème de répartition optimale des données dans un système réparti devient un problème complexe de programmation non-linéaire lorsqu'on cherche à optimiser de multiples facteurs comme le temps de réponse, le coût de communication, le coût de stockage, la charge de traitement, etc..., et sous des contraintes fixées par les impératifs organisationnels de l'entreprise (voir par exemple [MORGA 77]).

1.4.2 - Fichiers répartis ou Bases de Données réparties

Dans la littérature sur les Systèmes Répartis, deux alternatives sont couramment présentées pour la structuration des données d'un système informatique réparti ; en fichiers ou en Bases de Données. Conceptuellement un système de fichiers peut être considéré comme un sous-ensemble d'un système de Bases de Données ; en effet, la différence fondamentale entre les deux réside dans le degré de complexité structurelle gérée par le système logiciel [CHAMP 80, Chapitre 3] : un fichier est composé de quelques types de données qui sont accessibles par une seule méthode et dont les relations entre données sont gérées par le programme d'application ; par contre, une Base de Données est composée de nombreux types de données qui sont accessibles par de multiples méthodes et dont les interrelations (plus complexes) sont gérées par un Système de Gestion de données.

L'évolution des systèmes de fichiers vers les systèmes de Bases de Données répond à un objectif de recherche d'indépendance entre les programmes d'application et les structures des données et du stockage. Initialement, les Bases de Données permettent de combiner plusieurs fichiers dans une seule Base de Données logique et physique qui facilite le partage et le maintien de l'intégrité.

L'extension des Bases de Données vers les Bases de Données Réparties constitue une suite de cette évolution : les aspects physiques de la Base de Données sont décomposés tout en conservant unifiés les aspects logiques, afin d'obtenir les avantages de la répartition : meilleure disponibilité et fiabilité, réduction du coût, meilleur temps de réponse, possibilité de combiner des systèmes hétérogènes, etc.

Toutefois la réalisation de Systèmes de gestion de Bases de Données réparties pose des problèmes importants de traduction, surtout lorsqu'il s'agit de Bases de Données hétérogènes [LEBIH 76 ; JOUVE 80] : dans ce cas, chaque site possède une Base de Données décrite par un modèle local ; mais les différents modèles locaux sont décrits de manière différente (en langages différents) et pour exprimer les accès aux données non locales, il est nécessaire de définir un modèle global qui décrit toutes les données manipulées par les différentes Bases de Données Locales, leurs relations et leurs localisations.

2 - LES PROBLEMES DE MAINTIEN DE LA COHERENCE DANS LES SYSTEMES REPARTIS

2.1 - Le concept de cohérence

Dans tout système informatique, les données modifiables ont une "signification" qui dépend de l'instant de leur utilisation ; de façon intuitive, on peut affirmer que les données d'un système sont dans un état "cohérent" s'ils possèdent une signification valable pour les processus qui les utilisent et pour le monde externe [LELAN 79 C].

Plus précisément, pour tout système informatique il existe un ensemble de relations appelées *"contraintes d'intégrité"* lesquelles expriment les spécifications auxquelles les données du système doivent satisfaire en permanence ; l'état du système est *cohérent* à un instant donné, si (et seulement si) l'ensemble des données satisfait aux contraintes d'intégrité à cet instant [ESWAR 76].

La notion globale de cohérence peut être décomposée en plusieurs classes de cohérence qu'on a intérêt à préserver dans le système. Cette décomposition est la suivante :

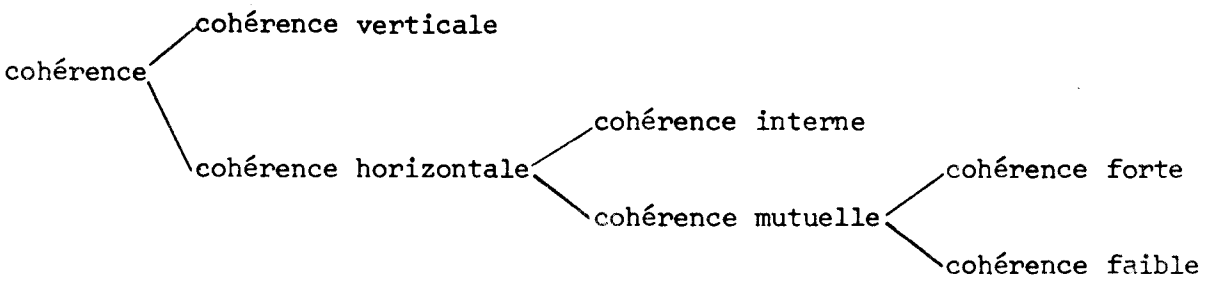


Fig. 10 : Classes de cohérence.

La *cohérence verticale* est relative au maintien des valeurs autorisées pour chaque donnée qui est soumise à une séquence d'opérations, tandis que la *cohérence horizontale* est relative au maintien des relations entre données soumises à des opérations en parallèle [LANCI 78, Chapitre 6].

Dans la cohérence horizontale, on distingue la *cohérence interne* qui est relative à la préservation des relations d'intégrité entre différentes données, et la *cohérence mutuelle* (concept lié uniquement aux systèmes avec duplication de données) qui est relative à la conservation de l'unicité entre les différentes copies d'une donnée. En fait, la cohérence mutuelle peut être considérée comme un cas particulier de la cohérence interne où la relation à maintenir entre les données (qui sont des copies d'une donnée) est la relation d'identité [WILLS 79, Chapitre 1].

Pour la cohérence mutuelle, on considère deux degrés possibles de la cohérence qu'on veut maintenir entre les copies des données [CHAMP 80, Chapitre 10] :

- La *cohérence forte* exige que toutes les copies d'une donnée présentent la même version en permanence ; cette condition exige que chaque mise à jour sur une donnée soit répercutée sur toutes ses copies ; en conséquence, le délai dans le temps de réponse peut être considérable.

- La *cohérence faible* exige que toutes les copies d'une donnée convergent dans le temps à la même version ; mais, à chaque instant, quelques copies peuvent être en retard par rapport aux autres dans le nombre des mises à jour traitées ; cette condition moins exigente que l'antérieure, permet de réduire le délai dans le temps de réponse (on obtient ainsi une meilleure "promptitude").

2.2 - Le concept de transaction

Dans un système informatique où les données sont soumises aux actions de multiples processus, l'état qui résulte après chaque opération particulière n'est pas toujours cohérent ; alors, pour délimiter les états cohérents du système (qui devraient constituer les seuls points observables au monde externe) on utilise la notion de *transaction* : une transaction est l'unité de traitement déclenchée par l'utilisateur, constituée d'une séquence d'actions telle que son exécution complète transforme le système d'un état cohérent à un nouvel état cohérent (toutefois la cohérence n'est pas assurée pendant une telle transformation) [ESWAR 76].

Dans un système informatique réparti, chaque *transaction répartie* (i.e. contenant des actions qui concernent les données de plusieurs sites) entraîne une collection de processus nommés *"incarnations de la transaction"* qui servent la transaction dans les différents sites remplissant deux fonctions [MENAS 79] :

- acquérir, utiliser et libérer les ressources locales ;

- échanger des messages avec les autres incarnations afin d'obtenir la coopération nécessaire à la réalisation de la transaction.

Il existe deux schémas généraux pour l'exécution d'une transaction répartie [CHAMP 80, Chapitre 3] :

- *schéma de données statiques* : les opérations qui font partie de la transaction sont envoyées aux sites où les données résident et la réponse est retournée au site initiateur de la transaction.

- *schéma de transaction statique* : les données concernées par la transaction sont envoyées au site initiateur de la transaction qui devra réaliser les opérations sur ces données. (Ce schéma pose le problème de traduction des modèles de données lorsqu'il s'agit d'un système de sites hétérogènes).

Dans la pratique, lorsque chaque opération de la transaction concerne plusieurs données qui ne résident pas forcément toutes dans le même site, il est nécessaire d'utiliser une combinaison des deux schémas précédants ; l'exécution de la transaction entraîne alors un ensemble d'actions de deux types [ROTHN 77] :

- des traitements locaux réalisés (éventuellement en parallèle) sur plusieurs sites,

- des transferts de données d'un site à un autre (éventuellement en parallèle).

Plusieurs stratégies étant possibles pour la définition de ces actions, la sélection d'une alternative devrait consister à chercher l'optimisation de facteurs comme : le coût des communications, le délai de transmission, etc.

2.3 - Menaces au maintien de la cohérence lors de concurrence et de défaillances

Le partage de données en environnement réparti et non totalement fiable est à l'origine de problèmes d'incohérence. En effet, en supposant que l'exécution de chaque transaction isolée préserve les contraintes d'intégrité du système, le maintien de la cohérence est menacé lorsqu'il y a *concurrence* de transactions qui utilisent les mêmes données, et lorsque se produisent des *défaillances* du système. Pour illustrer ceci, considérons l'exemple simple d'une application bancaire qui s'occupe uniquement du virement de sommes d'argent entre comptes bancaires ; on peut définir pour cette application deux contraintes d'intégrité qui doivent être satisfaites en permanence :

- C1 : Le solde de tout compte doit être positif ou nul.
- C2 : La somme de tous les comptes demeure constante.

La transaction T suivante (qui préserve les deux contraintes d'intégrité lorsqu'elle est exécutée seule dans le système) contient les opérations nécessaires pour réaliser un virement d'une somme Q, d'un compte A vers un autre compte B :

```
Début-transaction T(Q, A, B) ;
    a1 : Lire A ;
    a2 : Si A ≥ Q
         alors a3 : A : = A - Q ;
              a4 : B : = B + Q ;
         sinon a5 : Virement refusé
         Fin-si
Fin-transaction
```

Pour cette transaction, les actions a3 et a4 peuvent être exécutées en parallèle si les comptes A et B résident dans deux sites différents (nommés SA et SB). Supposons qu'un utilisateur au site SB déclenche une transaction de type T pour effectuer un virement d'une somme Q_1 du compte A vers B : la lecture de A donne une valeur supérieure à Q_1, ainsi l'action a3 de mise à jour de A est lancée mais au moment même d'effectuer cette mise à jour le compte A peut se trouver déjà modifié par une autre transaction de virement déclenchée par un utilisateur sur le site SA. Cette concurrence de transactions peut donner comme résultat un solde négatif pour le compte A entraînant l'incohérence du système, puisque la contrainte C1 n'est pas satisfaite.

Dans une autre situation, supposons qu'un utilisateur sur le site SA déclenche une transaction de virement, mais que le site SB où le compte B réside est isolé du réseau de communication à la suite d'une défaillance. Alors le compte A sera mis à jour mais pas le compte B, ceci entraînant l'incohérence du système puisque la contrainte C2 n'est pas satisfaite.

L'exemple antérieur illustre bien les deux menaces fondamentales au maintien de la cohérence dans un système informatique réparti :

- La concurrence de transactions rend possible l'accès d'une transaction aux données utilisées par une autre transaction en cours. Ces données peuvent se trouver dans un état temporairement incohérent conduisant, lors de la première transaction, à des résultats non souhaités et incohérents.

- Les défaillances du système peuvent interrompre une transaction en cours laissant les données dans un état incohérent.

Pour le cas de données dupliquées, il faut s'assurer en plus que toute mise à jour soit répercutée sur toutes les copies malgré les défaillances, et que la séquence des mises à jour soit la même pour toutes les copies malgré la concurrence de transactions [LELAN 78].

2.4 - Propriétés demandées à un protocole de contrôle de cohérence

La réalisation d'applications réparties exige une coopération entre les sites du système basée sur les échanges de messages. Cette coopération nécessite un ensemble de conventions communes, nommé "protocole de contrôle", pour gérer la compétition pour l'accès à des données partagées, et dont le but final est la préservation de la cohérence dans le système.

Les principales propriétés demandées à un tel protocole sont les suivantes [MENAS 78 ; LELAN 79 C] :

- *Convergence et impartialité* : le protocole doit régler les conflits entre processus pour l'accès aux données partagées, en évitant les attentes infinies (et les interblocages) et sans favoriser aucun processus.

- *Résilience* : le protocole doit assurer la survivance du système aux défaillances des sites ou du réseau de communication ; les sites isolés, une fois redevenus fonctionnels, doivent être restaurés correctement.

- *Performance* : l'opération du protocole ne doit pas dégrader la performance du système ; la surcharge entraînée par le protocole en trafic de messages additionnels, en traitement et en stockage d'information de contrôle, doit être acceptable par rapport au temps de réponse, au coût des communications et au débit d'opérations.

- *Déterminisme* : le protocole doit effectivement préserver la cohérence du système : l'intégrité de l'ensemble des données ne doit jamais être menacée.

Dans la suite de ce chapitre nous allons présenter les stratégies générales pour établir le protocole de contrôle de cohérence d'un système réparti. Ces stratégies sont divisées en trois aspects : les stratégies pour le contrôle de concurrence (partie 3), les solutions aux interblocages (partie 4) et les techniques de récupération aux défaillances (partie 5).

III - CONTROLE DE COHERENCE LORS DE CONCURRENCE

Dans un système informatique réparti, trois degrés possibles de concurrence entre transactions peuvent être considérés [ANDRA 80] :

- *Ni concurrence, ni parallélisme* : le système ne permet qu'une seule transaction exécutée à la fois, et à tout moment il n'y a qu'une seule opération en cours de réalisation.

- *Pseudo-concurrence* (parallélisme interne) : le système ne permet qu'une transaction à la fois, mais la répartition des données permet le parallélisme interne entre opérations d'une transaction concernant des sites différents.

- *Concurrence réelle* (parallélisme interne et externe) : le système permet l'exécution simultanée de plusieurs transactions et aussi la réalisation en parallèle des opérations de chaque transaction.

C'est seulement pour ce dernier degré de concurrence réelle qu'on profite au maximum des propriétés de parallélisme des systèmes répartis ; mais c'est également dans ce cas qu'un contrôle de concurrence et une gestion de conflits entre transactions sont encore plus nécessaires pour préserver la cohérence dans le système. L'objectif d'un tel contrôle de concurrence est alors de permettre l'exécution simultanée de plusieurs transactions en garantissant à chacune un fonctionnement équivalent à celui qu'elle aurait eu si elle avait été seule dans le système.

3.1 - Identification de transactions

L'allocation de ressources dans un système réparti nécessite un mécanisme d'identification de transactions qui assigne à chaque transaction un identificateur unique et permanent dans le système. En plus, ce mécanisme doit permettre d'établir un ordre total entre transactions afin de résoudre les conflits d'accès aux données partagées.

Si le système ne permet pas un degré de concurrence réelle entre transactions, le mécanisme d'identification de transactions n'est pas tellement nécessaire puisque des conflits d'accès ne peuvent pas se produire.

Pour un système réparti avec une concurrence réelle entre transactions, un ordre total peut être établi en associant une date à toute transaction lors de son initialisation et dans ce cas trois méthodes générales d'identification sont considérées [LELAN 79 A ; LELAN 79 C] :

- *Utilisation d'horloges physiques* : chaque transaction initialisée sur un site reçoit une estampille formée avec la valeur de l'horloge locale et complétée avec l'identité (ou priorité) du site afin de pouvoir établir un ordre total entre toutes les estampilles du système. L'estampille d'une transaction accompagne tous les messages entraînés par la transaction. Cette méthode nécessite néanmoins de pouvoir réajuster régulièrement les horloges afin de les synchroniser.

- *Utilisation de compteurs* (ou horloges logiques) : cette méthode basée sur les principes de Lamport [LAMPO 78] consiste à utiliser dans chaque site un compteur local d'événements qui prend des valeurs entières croissantes. Chaque message à envoyer est marqué avec la valeur courante du compteur local, ce qui combiné avec la priorité du site permet d'établir un ordre total. La synchronisation des compteurs a lieu lors de la réception des messages : un site avance son compteur local pour marquer une valeur plus grande que l'estampille du message arrivé.

- *Utilisation de séquenceurs* : la notion de séquenceur introduite par Reed [REED 79] permet d'ordonner totalement les événements d'un système tout en évitant les trous dans la numérotation (dans les méthodes antérieures, des trous sont créés lors des réajustements des horloges ou des compteurs) et en n'utilisant pas les identités des sites pour ne pas rendre un site prioritaire par rapport à un autre. Un séquenceur est accédé par une opération unique - "ticket" - réalisée en exclusion mutuelle et qui délivre sa valeur courante et l'incrémente.

Une mise en oeuvre des séquenceurs pour l'identification des transactions dans un système réparti est la méthode du *"Séquenceur Circulant"* de Le Lann [LELAN 79 A] : les identités uniques des sites permettent de considérer un anneau virtuel reliant tous les sites ; un message spécial appelé "Jeton de Contrôle" contenant un séquenceur circule en permanence sur cet anneau, et permet au site qui le détient de prélever des tickets pour les attribuer aux transactions initialisées sur le site depuis le dernier passage du jeton et qui sont en attente de référence. Les transactions reçoivent de cette façon des identificateurs uniques et croissants qui accompagneront tous les messages entraînés par leur exécution.

Cette méthode requiert néanmoins des techniques de reconfiguration de l'anneau et de récupération du jeton lors de défaillances : pour cela chaque site vérifie régulièrement si son successeur dans l'anneau est "vivant" en lui envoyant des messages qui doivent être acquittés ; dans le cas négatif, le site déclenche une reconfiguration en envoyant des messages au successeur potentiel de son successeur précédant ; un numéro de cycle qui accompagne le jeton et qui est incrémenté à chaque révolution complète sur l'anneau, sert a savoir si le jeton est perdu lors de la défaillance d'un site et à le régénérer correctement.

3.2 - Mécanismes de verrouillage pour l'ordonnancement cohérent de transactions

La concurrence et le partage de données dans un système réparti entraînent sur un même site des séquences d'actions provenant de différentes transactions : pour préserver la cohérence des données, un protocole de verrouillage est nécessaire afin d'éviter qu'une donnée en cours d'utilisation par une transaction et temporairement incohérente, soit accessible à une autre transaction concurrente. Un tel protocole assurera alors un "ordonnancement cohérent" des transactions [ESWAR 76].

Le principe pour obtenir un ordonnancement cohérent de transactions est de retarder une action d'une transaction à l'aide de verrous, jusqu'au moment où son exécution ne détruit pas la cohérence des données.

3.2.1 - *Le Concept de verrou*

Les verrous sont utilisés pour donner à chaque transaction une vue cohérente des données : une transaction doit verrouiller les données à consulter pour garantir qu'elles sont en état cohérent et doit verrouiller les données à modifier pour les marquer comme temporairement incohérentes aux yeux des autres transactions concurrentes [GRAY 75].

Une transaction qui exécute une opération de verrouillage sur une donnée tente d'obtenir le droit d'utiliser cette donnée selon un certain mode d'accès. De manière générale on distingue deux modes principaux pour les verrous :

- *verrou en mode partagé* : pour obtenir le droit d'utiliser une donnée en lecture seule (ce qui permet son utilisation partagée par de multiples lecteurs),

- *verrou en mode exclusif* : pour obtenir le droit d'utiliser une donnée en lecture et en écriture (ce qui implique son utilisation exclusive par une seule transaction à la fois).

Deux demandes de verrous sur une même donnée faites par deux transactions différentes sont *compatibles* si elles peuvent être octroyées simultanément. Pour les modes partagé et exclusif, seules les demandes de verrous en mode partagé sont compatibles entre elles. Lorsqu'une transaction a obtenu un verrou sur une donnée, elle est la seule à pouvoir la déverrouiller.

Selon la durée pendant laquelle un verrou est maintenu par une transaction, on peut classer les verrous en deux catégories [GRAY 75] :

- verrou *court* : s'il est maintenu pour une seule action,
- verrou *long* : s'il est maintenu jusqu'à la fin de la transaction.

3.2.2 - *Ordonnancement cohérent de transactions*

L'exécution d'un ensemble de transactions concurrentes peut être représentée par un *ordonnancement* défini comme une suite d'actions appartenant aux diverses transactions et représentant l'ordre d'exécution de toutes les actions de ces transactions (incluant les actions de verrouillage et de déverrouillage) [ESWAR 76].

Un ordonnancement de transactions est *séquentiel* si dans la suite, toutes les actions de chaque transaction sont terminées avant le début de la suivante.

Un ordonnancement de transactions est *légal* si les verrous fonctionnent selon leurs spécifications [ESWAR 76] :

- Une donnée verrouillée en mode partagé par une transaction, peut être verrouillée par d'autres transactions en mode partagé mais non en mode exclusif.

- Une donnée verrouillée en mode exclusif par une transaction ne peut subir aucun nouveau verrouillage.

Il serait intéressant de pouvoir établir la caractérisation générale des ordonnancements *cohérents* de transactions concurrentes, c'est-à-dire les ordonnancements qui préservent la cohérence du système. Mais actuellement, une telle caractérisation générale n'est pas connue. Néanmoins, deux classes particulières d'ordonnancements cohérents ont pu être caractérisées [ESWAR 76 ; CORNA 81, Chapitre 10] :

- Tout ordonnancement séquentiel est cohérent : en effet, par définition chaque transaction isolée préserve la cohérence du système et en conséquence une séquence de transactions la préserve aussi. Un ordonnancement non séquentiel, mais équivalent à un ordonnancement séquentiel sera aussi cohérent ; cette *équivalence entre ordonnancements* signifie que :

* pour chaque donnée, les modifications sont les mêmes pour les deux ordonnancements et réalisées dans le même ordre ;

* les consultations effectuées entre deux mises à jour consécutives sont les mêmes pour les deux ordonnancements (mais elles peuvent être réalisées en ordre différent).

- Tout ordonnancement légal d'un ensemble de transactions qui présentent les deux propriétés : être "Bien Formées" et à "Deux Phases", est cohérent : ce résultat est démontré par Eswaran et Gray [ESWAR 76] par l'équivalence d'un tel ordonnancement a un ordonnancement séquentiel. Les deux propriétés demandées aux transactions sont définies ainsi : Une transaction est *Bien Formée* si elle utilise les verrous selon leurs spécifications, c'est-à-dire :

  * avant d'effectuer une opération sur une donnée, la donnée doit se trouver verrouillée par la transaction, dans un mode compatible avec l'opération ;

  * chaque donnée utilisée est verrouillée une seule fois par la transaction ;

  * toutes les données verrouillées sont déverrouillées par la la transaction avant sa fin.

Une transaction est à *Deux Phèses* si la suite de ses opérations de verrouillage peut être décomposée en deux phases successives : une phase "croissante" où les verrous sont placés, puis une phase "décroissante" où ils sont retirés. Cette propriété évite qu'une transaction puisse utiliser un ensemble de données où une partie a été mise à jour par une autre transaction concurrente (qui n'est pas à Deux Phases) : mais l'autre partie non (i.e. un ensemble incohérent).

Les résultats antérieurs établissent les bases d'un protocole de verrouillage tel que l'exécution de transactions concurrentes préserve la cohérence des données.

3.2.3 - *Choix de la granularité*

La *granularité* est la taille de l'entité contenant des données qui peut être verrouillée par les transactions. La sélection de la granularité représente un compromis à faire entre la concurrence et la surcharge : d'une part une granularité fine favorise un haut degré de concurrence entre transactions, mais d'autre part la surcharge en performance, en stockage et en gestion de verrous, croît rapidement avec la finesse de granularité (il y a un plus grand nombre de verrous à établir).

En fait, pour satisfaire un tel compromis il est souhaitable d'avoir des granularités de différentes tailles dans le même système [GRAY 75].

3.3 - Modalités pour les protocoles de contrôle de concurrence

Les protocoles de contrôle de concurrence de transactions pour les systèmes répartis peuvent être classés en quatre groupes selon le schéma utilisé de répartition du traitement (schémas présentés dans le paragraphe 1.3 de ce chapitre). Pour chacun de ces groupes nous allons présenter brièvement les caractéristiques de quelques protocoles très connus dans la littérature des systèmes répartis (une analyse détaillée de ces protocoles peut être trouvée dans la Thèse de Wills [WILLS 79]).

3.3.1 - *Protocoles à contrôle centralisé*

Ces protocoles se caractérisent par l'existence d'un site central, point de contrôle unique pour l'ordonnancement de toutes les transactions produites dans le système. Ils présentent l'avantage de faciliter le traitement de conflits d'accès entre transactions concurrentes, mais en même temps ils sont très vulnérables aux défaillances sur le site central, lequel contient des informations de contrôle nécessaires à l'opération du système.

Quelques protocoles à contrôle centralisé *statique*, où le rôle du site central est assigné de manière fixe à un site particulier, sont les suivants :

- Protocole d'"Alternance entre sessions de travail et de mise en cohérence", défini par *Meyer et Dargent* [MEYER 78] : le système considéré possède une organisation étoilée où tous les sites sont connectés à un site central. Les utilisateurs réalisent des transactions dans des sessions de "travail" qui alternent avec des sessions de "mise en cohérence". Dans les sessions de travail, une transaction accède et modifie uniquement les données du site local (on suppose un système avec duplication des données) ; dans les sessions de mise en cohérence, de nouvelles transactions ne sont plus acceptées et les modifications réalisées par chaque transaction déjà terminée sont répercutées par le site central sur les copies de données dans les autres sites (pour un ordre séquentiel des transactions). Les caractéristiques fondamentales du protocole sont : contrôle centralisé statique, duplication totale des données, maintien de la cohérence faible, pas de parallélisme entre transactions (et non disponibilité du système aux utilisateurs pendant les sessions de mise en cohérence).

- Protocole de "Copie primaire unique", défini par *Alsberg et al.* [ALSBE 76] : le site central possède une copie de l'ensemble des données du système (copie primaire) ; toute nouvelle transaction de mise à jour est adressée au site central qui est chargé d'actualiser sa copie et de diffuser les mises à jour aux autres sites ; les transactions sont alors exécutées en ordre séquentiel d'après leur arrivée au site central. Les caractéristiques sont : contrôle centralisé statique, duplication totale, cohérence forte, pas de parallélisme entre transactions.

- Protocole de "Contrôleur central de verrous", défini par *Menasce et al.* [MENAS 78] : le site central est chargé de la gestion globale des verrous pour assurer un ordonnancement cohérent des transactions concurrentes (qui doivent être "Bien Formées" et "à Deux Phases"). Pour cela, toute demande de verrouillage des différentes transactions est adressée au site central qui est le seul à décider sur son acceptation ou rejet ; un verrou octroyé entraîne l'actualisation d'une table globale de verrous et sa diffusion afin que les autres sites

actualisent leurs tables locales de verrous. Les caractéristiques sont : contrôle centralisé statique, duplication ou partition des données, cohérence forte, parallélisme entre transactions au niveau du système et au niveau de chaque site.

Parmi les protocoles à contrôle centralisé dynamique, où la fonction de site central change de site (entraînant le transfert des informations de contrôle au nouveau site central), on peut citer :

- Protocole d'"Exclusion mutuelle par circulation d'un jeton de contrôle", défini par *Le Lann* [LELAN 79 C] : un jeton de contrôle circule sur un anneau virtuel établi entre les sites ; le site possédant le jeton est le seul autorisé pour déclencher l'exécution de transactions (transactions qui étaient en attente d'exécution depuis le dernier passage du jeton sur le site) ; une fois que de telles transactions sont totalement terminées, le site passe le jeton de contrôle au site suivant dans l'anneau. Les caractéristiques sont : contrôle centralisé dynamique, duplication ou partition, cohérence forte, pas de parallélisme entre transactions.

3.3.2 - *Protocoles à contrôle local*

Dans ces protocoles chaque site exécute les transactions de manière indépendante et à sa propre vitesse, basé sur un mécanisme de files d'attente qui permet d'ordonner les requêtes arrivées pour exécuter séquentiellement les transactions. Avant d'exécuter une transaction, un site doit s'assurer que toutes les transactions antérieures sont déjà arrivées et exécutées. Ces protocoles sont adaptés uniquement aux systèmes avec duplication totale des données, pour lesquels l'exécution séquentielle des transactions et dans un ordre identique pour chaque site, assure la cohérence faible des données.

Parmi les protocoles à contrôle local on peut citer les suivants :

- Protocole de "Canal unique à diffusion", défini par *Banino et al.* [BANIN 79] : les sites du système considéré se communiquent par un canal unique qui permet la diffusion d'un seul message à la fois ; cette exclusion mutuelle permet d'être sûr que toutes les requêtes de mises à jour soient reçues dans le même ordre sur chaque site. Au niveau de chaque site il existe deux files de messages : une file principale avec les messages contenant les requêtes de la transaction en cours que le site exécute à sa propre vitesse, et une file secondaire où le site place les messages d'autres transactions arrivés pendant l'exécution de la transaction en cours. Les caractéristiques sont : contrôle local, duplication totale, cohérence faible, parallélisme entre transactions au niveau du système (et application limitée aux systèmes avec un canal unique à diffusion).

- Protocole de "Multiples files d'attente à chaque site", défini par *Herman et Verjus* [HERMA 79] : les sites déclenchent les transactions en diffusant tous les messages correspondants lesquels portent des estampilles générées par compteurs ; une seule estampille accompagne tous les messages d'une même transaction. Au niveau de chaque site il existe n files d'attente pour placer les messages provenants des n sites du système ; à chaque fois, le site exécute la transaction qui possède l'estampille la plus ancienne entre les transactions qui se trouvent en tête des files ; mais le site doit s'assurer que toutes les transactions plus anciennes que celle choisie ont été déjà exécutées, et pour cela, lorsqu'une file est vide, il doit consulter le site correspondant. Les caractéristiques sont : contrôle local, duplication totale, cohérence faible, parallélisme entre transactions au niveau du système.

3.3.3 - *Protocoles à contrôle partiellement réparti*

Ces protocoles se caractérisent par l'existence d'un site "responsable" associé à chaque transaction qui a la charge de décider sur l'acceptation ou le rejet d'une nouvelle transaction après une consultation des sites concernés. Ces protocoles nécessitent un grand nombre de

messages pour accomplir les étapes de chaque transaction : fréquemment une étape de synchronisation, où la consultation des sites et l'allocation des données ont lieu, et une étape de validation, où la transaction une fois acceptée est exécutée sur les sites. Par contre, ces protocoles offrent une fiabilité supérieure par rapport aux protocoles à contrôle centralisé.

Entre les protocoles à contrôle partiellement réparti on trouve les suivants :

- Protocole d'"Un seul Administrateur à la fois", défini par *Seguin et al.* [SEGUI 79] : à tout instant il n'y a qu'un seul site responsable dans le système, appelé "l'Administrateur" ; tout site désirant lancer une transaction, doit obtenir le statut d'Administrateur en le demandant au site Administrateur courant. On considère un système avec duplication totale des données ; toute requête de mise à jour d'une transaction entraîne une étape de synchronisation où le site Administrateur doit obtenir un vote affirmatif de la majorité des sites au sujet de l'acceptation de la transaction (en fait le but de cette étape est de vérifier qu'une majorité des sites sont vivants), et une étape de validation où la mise à jour est diffusée sur toutes les copies. La réinsertion d'un site isolé du système est réalisée facilement en prenant une copie à jour des données d'un autre site. Les caractéristiques sont : contrôle partiellement réparti, duplication totale, cohérence forte, pas de parallélisme entre transactions.

- Protocole de "Consensus majoritaire par examen de dates", défini par *Thomas* [THOMA 78] : une estampille générée par une horloge physique locale, marque chaque donnée pour indiquer la date de la dernière mise à jour. On considère une duplication totale des données. Lorsque le site responsable d'une transaction (celui où la transaction a été initialisée) désire réaliser une mise à jour des données, il envoie à chaque site une requête comportant : la liste des données à modifier et

leurs nouvelles valeurs, et la liste des données de base (à partir desquelles les modifications sont déterminées) et leurs dates associées au moment de la lecture ; un site accepte une telle requête seulement si les dates des données de base ne sont pas périmées par rapport aux dates des copies locales de ces mêmes données. S'il y a un consensus majoritaire sur l'acceptation de la requête, le site responsable diffuse l'ordre de mise à jour à tous les sites. Les caractéristiques sont : contrôle partiellement réparti, duplication totale, cohérence faible, parallélisme entre transactions au niveau du système et au niveau de chaque site (et pas d'utilisation de verrous).

- Protocole de "Copie primaire pour chaque donnée", utilisé dans le système INGRES et défini par *Stonebraker* [STONE 79] : les données sont partiellement dupliquées et pour chaque donnée, il existe une copie "primaire" ; chaque requête d'une transaction est adressée au site contenant la copie primaire de la donnée concernée (site primaire). Une transaction se déroule en deux étapes : une étape de synchronisation où toutes ses requêtes sont adressées aux sites primaires afin qu'ils réalisent les verrouillages nécessaires, et une étape de validation où, après avoir reçu l'acquittement de tous, le site responsable envoie l'ordre de validation aux sites primaires afin qu'ils diffusent les mises à jour à toutes les copies. Les caractéristiques sont : contrôle partiellement réparti, duplication partielle, cohérence faible, parallélisme entre transactions au niveau du système et de chaque site.

- Protocole de "Producteur-Consommateurs", utilisé dans le système SCORE et défini par *Le Lann et al.* [LELAN 79 B] (dans le cadre du projet SIRIUS [LEBIH 80]) : chaque transaction se déroule en trois étapes : dans l'étape de "Synchronisation" le site responsable de la transaction, appelé "le Producteur", envoie les requêtes aux sites concernés, appelés les "Consommateurs" ; les Consommateurs doivent établir les verrous sur les données locales (afin d'assurer l'ordonnancement cohérent de transactions "Bien Formées" et "à Deux Phases"), et doivent élaborer des "Listes d'Intentions" contenant les modifications à faire sur ces don-

nées ; dans l'étape de "Prévalidation", le Producteur vérifie que tous les Consommateurs sont "vivants", et si c'est le cas, dans l'étape de "Validation" le Producteur diffuse aux Consommateurs l'ordre de rendre effectives les modifications sur les données. Les caractéristiques sont : contrôle partiellement réparti, partition, duplication totale ou duplication partielle des données, cohérence forte, parallélisme entre transactions au niveau du système et au niveau de chaque site.

- Protocole par "Classes de transactions", utilisé dans le système SDD-1 et défini par *Bernstein et al.* [BERNS 78] : quatre classes de transactions sont définies, d'après la nature des interférences qu'elles peuvent présenter, et, pour chaque classe, un protocole différent de verrouillage est appliqué. Ces protocoles spécifient les messages à échanger entre le site responsable d'une transaction et les autres sites afin de préserver la cohérence des données ; ils se caractérisent par un coût croissant (en nombre de messages nécessaires) et un degré de concurrence décroissant (ainsi le premier protocole, applicable aux transactions qui consultent uniquement les données locales, évite les messages et les verrous non nécessaires et permet un parallélisme maximum entre transactions). Les caractéristiques sont : contrôle partiellement réparti, duplication totale, cohérence faible, parallélisme entre transactions.

3.3.4 - *Protocoles à contrôle totalement réparti*

Dans ces protocoles la responsabilité pour accepter ou rejeter une transaction n'appartient pas à un site particulier, mais est totalement répartie entre les sites du système. Parmi ces protocoles on peut citer le suivant :

- Protocole de "Requête séquentielle sur un anneau virtuel", défini par *Ellis* [ELLIS 77] : l'exécution de toute transaction entraîne deux révolutions d'un message de synchronisation sur un anneau virtuel défini entre les sites : la première révolution est pour verrouiller toutes les données concernées par la transaction, et la deuxième révolution est pour effectuer les

mises à jour sur ces données. Les caractéristiques sont : contrôle totalement réparti, duplication totale, cohérence forte, pas de parallélisme entre transactions (et un temps de réponse qui peut être élevé à cause de la visite séquentielle des sites).

4 - TRAITEMENT D'INTERBLOCAGES

Dans les systèmes répartis, la possibilité d'interblocage entre transactions concurrentes est un problème dont la solution appartient aussi aux mécanismes de contrôle de concurrence ; ce problème d'interblocage, bien connu dans les systèmes centralisés, est devenu plus complexe dans les environnements répartis, méritant ainsi une attention particulière.

4.1 - Concepts d'interblocage et de famine

L'allocation des ressources-données requises par une transaction répartie, peut être réalisée de trois manières différentes [WILLS 79, Chapitre 3] :

- *Allocation statique* : les ressources sont allouées au moment de l'initialisation de la transaction de façon non-préemptible. (i.e. sans possibilité d'être retirées pendant la transaction). Il faut alors connaître toutes les ressources nécessitées par la transaction dès son début.

- *Allocation pseudo-dynamique* : la transaction entraîne deux étapes : pendant la première les ressources sont allouées progressivement mais elles restent préemptibles ; dans la deuxième étape les ressources sont bloquées pour devenir non-préemptibles et les mises à jour de la transaction sont effectuées.

- *Allocation dynamique* : un processus associé à la transaction visite l'ensemble des sites pour acquérir sur chacun d'eux les ressources nécessaires et de façon non-préemptible. La transaction ne peut pas modifier ces ressources avant de les avoir toutes acquises.

Quelle que soit la manière choisie, l'allocation de ressources doit être réalisée en évitant les problèmes liés de Famine et d'Interblocage [CORNA 81, Chapitre 9] :

- *La Famine* signifie l'attente indéfinie d'une transaction pour une ou plusieurs ressources qui ne lui sont pas allouées.

- *L'Interblocage* correspond à la situation de blocage mutuel qui se produit entre deux ou plusieurs transactions si chacune attend une ressource déjà allouée à une autre et, en même temps, possède une ressource demandée par une autre.

La figure 11 illustre une situation d'Interblocage entre trois transactions :

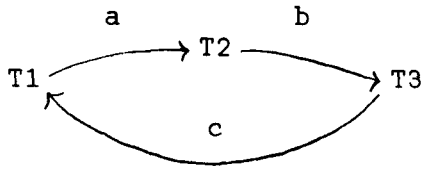


Fig. 11 : Situation d'Interblocage.

La transaction T1 possède la ressource c et attend a, la transaction T2 possède a et attend b et la transaction T3 possède b et attend c : aucune des trois transactions ne peut poursuivre son traitement. Dans un système réparti, les ressources concernant une situation d'interblocage (a, b et c pour l'exemple) peuvent se trouver sur différents sites ou toutes sur un même site, la détection de la situation étant plus difficile dans le premier cas.

4.2 - Conditions pour l'existence d'un interblocage

Une situation d'interblocage est matérialisée par une chaîne circulaire de transactions où chaque transaction possède une ou plusieurs ressources-données qui sont demandées par la prochaine transaction dans la chaîne. Quatre conditions sont nécessaires pour qu'une telle situation d'interblocage arrive [CHAMP 80, Chapitre 11] :

- C1 : Existence de transactions concurrentes,
- C2 : Allocation non statique de ressources,
- C3 : Accès aux ressources de façon mutuellement exclusive,
- C4 : Allocation non-préemptible des ressources (le système ne peut retirer arbitrairement une ressource déjà assignée à une transaction).

Les situations d'interblocage dans un système peuvent être évitées en excluant en permanence une des quatre conditions antérieures, mais dans un système réparti cela n'est pas souhaitable : en fait les conditions C1 et C2 sont nécessaires pour l'efficacité du système et les conditions C3 et C4 sont nécessaires pour la préservation de la cohérence des ressources-données.

4.3 - Stratégies générales pour résoudre l'interblocage

Les systèmes centralisés utilisent trois stratégies générales pour résoudre le problème de l'Interblocage ; celles-ci peuvent aussi être en principe appliquées aux systèmes répartis. Ces stratégies sont les suivantes [CHAMP 80, Chapitre 11] :

- *L'Evitement* : le comportement des transactions en cours est examiné de façon permanente pour déterminer les séquences d'actions qui permettent d'aboutir à une fin normale de toutes les transactions ; la décision sur l'allocation des ressources est faite d'après ces séquences déterminées. Cette stratégie exige une connaissance anticipée de l'utilisation des ressources par les transactions et de l'état de toutes les ressources du système à chaque instant ; elle est alors très peu compatible avec les systèmes répartis où l'obtention de telles informations entraînerait un coût excessivement élevé en nombre de messages.

- *La Détection et la Guérison* : toute situation d'Interblocage est détectée après son occurrence et elle est résolue en annulant

une ou plusieurs transactions. La détection est faite par localisation de cycles dans un graphe "d'attentes" qui représente les relations entre transactions dérivées des attentes de ressources.

- *La Prévention* : les situations d'Interblocage sont évitées en empêchant que l'une au moins des quatres conditions nécessaires à son occurrence puisse se présenter.

L'application des stratégies de Détection et de Prévention aux systèmes répartis conduit à diverses techniques qui seront exposées dans la suite.

4.3.1 - Techniques de Détection

Au lieu de représenter les relations d'attente entre transactions, un graphe d'attentes pour un système réparti doit représenter les relations d'attente entre "incarnations" de ces transactions (i.e. entre les processus chargés de servir les transactions dans les différents sites). Un tel graphe est construit alors de la manière suivante [MENAS 79] :

- Chaque noeud représente une incarnation d'une transaction nommée par le couple (nom de la transaction, nom du site).

- Chaque arc exprime l'attente d'une incarnation, soit d'une ressource possédée par une autre transaction dans le même site, soit d'un message qu'une incarnation de la même transaction doit lui envoyer à partir d'un autre site.

La figure 12 présente un graphe d'attentes pour trois transactions (T1, T2 et T3) actives dans un système réparti qui est composé de deux sites (S1, S2) :

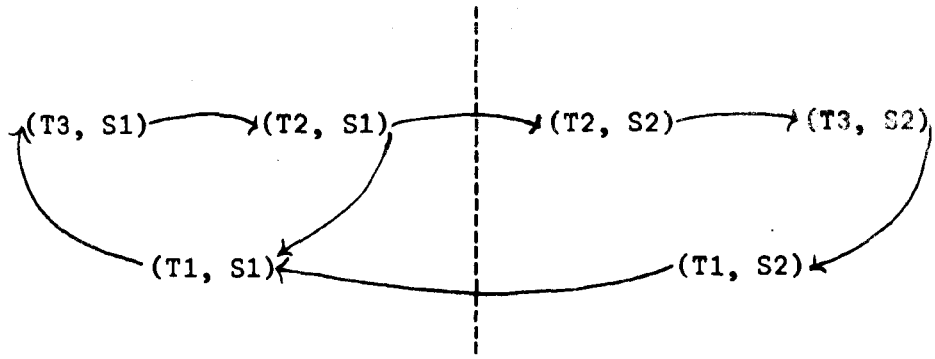


Fig. 12 : Graphe d'attentes pour des transactions réparties.

La détection d'une situation d'Interblocage est faite en localisant les cycles dans le graphe : ainsi par exemple, le graphe de la figure 12 présente un interblocage local au site S1 et un interblocage global concernant les deux sites S1 et S2. Un graphe local, représentant les relations d'attentes entre incarnations dans un site particulier, est suffisant pour détecter les interblocages locaux, mais le graphe global est nécessaire pour détecter les interblocages globaux.

On peut envisager deux manières de gérer le graphe global d'attentes pour détecter les interblocages :

- *de manière centralisée,* comme dans l'approche de Stonebraker en [STONE 79] : un site unique particulier maintient le graphe global d'attentes et le met à jour, à partir des informations des autres sites sur les attentes des transactions pour des ressources locales et sur la libération de ressources. Les autres sites peuvent détecter les interblocages locaux mais seul le site particulier peut détecter les interblocages globaux.

- *de manière répartie,* comme dans l'approche de Menasce en [MENAS 79] : le graphe global est réparti sur les sites et la détection d'un interblocage global peut être réalisée par n'importe quel site. Pour cela, chaque site maintient le graphe local d'attentes, la liste d'incarnations externes bloquées par les incarnations locales (i.e. qui attendent des messages) et la liste d'incarnations externes qui bloquent les incarnations locales. Avec l'aide des autres sites, tout site peut mettre à jour ces informations et générer le graphe global pour détecter les interblocages globaux.

Après la détection d'un interblocage, la guérison entraîne la sélection d'une ou plusieurs transactions pour les annuler afin d'éliminer le cycle dans le graphe d'attentes ; il y a plusieurs critères possibles pour cette sélection : la transaction qui possède moins de ressources, la transaction la plus récente, ou celle qui entraîne le moindre coût d'annulation, etc.

4.3.2 - *Techniques de Prévention*

Divers mécanismes sont possibles pour empêcher la vérification d'une condition nécessaire à l'occurrence des interblocages ; par exemple :

- L'exécution séquentielle des transactions au niveau du système par un contrôle centralisé, ou au niveau de chaque site par un contrôle local, éliminent la condition C1 sur l'existence de transactions concurrentes.

- La réclamation de toutes les ressources nécessitées par une transaction lors de son initialisation, élimine la condition C2 sur l'allocation non statique.

Mais l'élimination des conditions C1 ou C2 entraîne une baisse de l'efficacité du système. Par ailleurs, la condition C3 sur l'accès exclusif aux ressources est nécessaire pour la préservation de la cohérence. Il semble alors préférable de relâcher la condition C4 sur l'allocation non-préemptible, en supposant que le système soit capable de remettre dans un état cohérent les ressources retirées à des transactions en cours. Dans cette optique Rosenkrantz [ROSEN 78] propose deux techniques de Prévention, qui requièrent une identification des transactions assurant un ordre total :

- La technique du "WAIT-DIE" (Attendre ou Mourir) : lorsqu'une transaction T1 détecte un conflit avec une autre transaction T2 (à cause d'une ressource possédée par T2) il y a deux alternatives : si T1 est plus ancienne que T2, alors T1 doit attendre la fin normale ou anormale de T2 ; dans le cas contraire T1 doit être annulée et réinitialisée (et ses ressources retirées).

- La technique du "WOUND-WAIT" (Blesser ou Attendre) : lorsque T1 détecte un conflit avec une autre transaction T2 et T1 est plus ancienne que T2, alors T1 "blesse" T2, sinon T1 doit attendre la fin normale ou anormale de T2. Une blessure de T1 à T2 provoquera l'annulation de T2, uniquement si cette dernière transaction se met en attente de ressource, ou si elle l'est déjà.

Ces techniques préviennent les interblocages mais aussi la famine : au bout d'un temps fini toute transaction devient la plus prioritaire (la plus ancienne) et elle est sûre de ne plus être annulée. Le désavantage que présentent ces techniques est le risque de provoquer des annulations ("Rollbacks") non nécessaires ; mais comme le fait remarquer Le Lann [LELAN 81], dans les systèmes répartis où les transactions n'entraînent pas de grandes quantités de traitement, le coût impliqué par les annulations de transactions est acceptable, face au coût impliqué par les échanges d'informations nécessitées par un graphe global d'attentes dans les techniques de détection.

5 - CONTROLE DE COHERENCE LORS DE DEFAILLANCES

Les systèmes répartis offrent en principe une haute disponibilité des données parce que celle-ci n'est pas limitée par la fiabilité d'un seul composant mais par celle d'une combinaison de composants. En fait, pour obtenir cet objectif de haute disponibilité, un système réparti doit être *"résilient"* , concept introduit par Alsberg [ALSBE 76] qui signifie que le système peut continuer à opérer malgré les défaillances sur un ou plusieurs composants (éventuellement en mode dégradé) ; un système réparti présente un *degré de résilience* "N" s'il faut N sites simultanément en panne pour que l'opération du système ne soit plus assurée.

Le problème central des défaillances est la préservation de la cohérence du système, laquelle est menacée lorsqu'une transaction en cours est interrompue laissant des données en état incohérent ; les mécanismes de récupération doivent alors restaurer le système dans un état cohérent. La difficulté de réalisation d'une telle récupération dans les systèmes répartis, provient de l'existence de divers contrôleurs qui peuvent prendre des décisions indépendantes et contradictoires conduisant à d'autres situations d'incohérence.

5.1 - Types de défaillances

On peut distinguer deux classes d'incidents dans un système réparti [ANDRA 80] :

- *Les erreurs* qui affectent une ou plusieurs transactions actives mais qui laissent le système en activité et n'entraînent pas de perte d'informations (par exemple la détection d'un interblocage, une erreur de l'utilisateur au terminal, une erreur d'Entrée/Sortie, etc...). Le système traite ces erreurs de manière en-ligne dans le cadre de son déroulement normal.

- *Les défaillances* générales qui interrompent l'opération du système et peuvent entraîner la perte d'informations ; un mécanisme de récupération est alors nécessaire pour réinitialiser le système dans un état cohérent.

Trois types de défaillances peuvent être considérés [WILLS 79, Chapitre 1] :

- *Défaillance d'un site* : lorsqu'un site est affecté par une panne qui rend impossible la communication des autres sites avec lui.

- *Défaillance de ligne logique* de communication : lorsqu'une panne affectant spécifiquement le réseau de communication provoque la rupture de la communication entre deux ou plusieurs sites (et éventuellement la partition du réseau en deux sous-réseaux indépendants qui peuvent évoluer de façon différente et contradictoire).

- *Défaillances multiples* : lorsqu'une combinaison de défaillances des deux types antérieurs se produit simultanément.

5.2 - Mécanismes de récupération aux défaillances

La récupération à une défaillance concerne, d'une part les sites qui sont restés en activité, lesquels doivent être capables de poursuivre leurs traitements (même de façon dégradée) sans avoir à attendre la récupération des sites isolés ; d'autre part, la récupération concerne ces sites isolés qui doivent se remettre en état cohérent lors de leur reconnexion.

En plus d'assurer la préservation de la cohérence dans le système, les mécanismes de récupération doivent être efficaces : la récupération d'un site ne doit pas entraîner des délais intolérables.

5.2.1 - *Duplication versus Partition*

En général, les protocoles de contrôle de concurrence des systèmes répartis sont plus complexes et coûteux pour les données dupliquées que pour les données partitionnées à cause de la nécessité d'assurer la cohérence mutuelle entre copies ; en contrepartie les défaillances sont moins graves pour les données dupliquées puisque la partie du système non affectée et les transactions en cours peuvent continuer à opérer norma-

lement, en assurant ainsi une disponibilité supérieure à celle offerte par les données partitionnées : en effet, dans ce dernier cas la défaillance d'un site peut entraîner l'interruption de toutes les transactions en cours qui utilisent des données de ce site. Pour assurer une disponibilité acceptable dans un système de données partitionnées, il faut prendre une décision sur les transactions interrompues (par exemple : les annuler ou les compléter) sans devoir attendre la réinsertion du site en panne, laquelle peut prendre un temps indéfini : les mécanismes de récupération deviennent alors plus complexes pour les données partitionnées que pour les données dupliquées.

5.2.2 - *Récupération dans les systèmes avec Duplication des données*

Dans ces systèmes, le principe de récupération pour un site isolé suite à une défaillance consiste à remettre à jour sa copie, soit en appliquant les dernières mises à jour qu'il n'a pas pris en compte suite à sa défaillance et qui sont contenues dans un journal d'un autre site, soit en copiant toutes les données d'un autre site. Les mécanismes de récupération requièrent alors que chaque site garde un journal avec l'historique des mises à jour effectuées sur les données locales, mais cela introduit le problème du maintien effectif et non coûteux de ces journaux : ils doivent contenir le nombre d'informations nécessaires pour permettre la remise à niveau de la copie d'un site, mais pour éviter un stockage trop coûteux, il doit y avoir une élimination périodique des informations sur les mises à jour totalement réalisées dans tous les sites. Pour résoudre un tel problème, trois méthodes sont considérées par Wills [WILLS 79, Chapitre 3] :

- Chaque site conserve la valeur la plus récente de chaque donnée accompagnée de la date de la mise à jour associée (pas de journal).

- Chaque site conserve un journal avec l'historique des modifications réalisées par les transactions, et périodiquement effectue un sondage auprès des autres sites pour savoir quelles transactions ont été totalement réalisées : les informations concernant ces transactions cessent d'être nécessaires et peuvent ainsi être enlevées du journal.

- Chaque modification enregistrée dans le journal d'un site est associée à une liste des sites qui doivent envoyer un acquittement au premier site, après avoir réalisé eux-mêmes la modification. Lorsque le site reçoit tous les acquittements correspondant à une modification, il est autorisé à l'enlever du journal.

Les journaux d'un système pourraient résider seulement sur un sous-ensemble de sites considérés plus fiables, pour réduire encore les coûts de stockage.

La partition d'un réseau en deux sous-réseaux indépendants présente le risque d'une évolution différente des deux ensembles des copies. Une solution à ce problème, fréquemment adoptée, est de permettre la continuation des opérations uniquement à la partition qui possède une majorité des copies.

5.2.3 - *Récupération dans les systèmes avec Partition des données*

Dans ces systèmes une défaillance peut entraîner l'interruption d'une ou plusieurs transactions en cours, et pour chacune il faut prendre une décision sur l'action nécessaire pour restaurer la cohérence dans le système. On utilise alors une technique de récupération de "Validation" (Commitment), basée sur les principes énoncés par Gray et al. [GRAY 75] pour les transactions en environnement centralisé :

- une écriture d'une transaction est dite "*validée*" lorsque la transaction renonce au droit d'annuler l'écriture et met la nouvelle valeur à la disposition des autres transactions ;

- le *Point de Validation* d'une transaction correspond à la première validation d'écriture et constitue un point de non-retour : avant ce point la transaction peut être annulée sans affecter les autres transactions, mais à partir de ce point, la transaction doit nécessairement être menée à son terme.

La *technique de récupération de Validation*, dérivée des principes antérieurs, consiste alors à annuler tous les effets des opérations d'une transaction qui a été interrompue avant son point de validation (point commit), ou à compléter et rendre permanents ces effets si la transaction a été interrompue après son Point de Validation. Pour pouvoir annuler ou compléter une transaction, une sorte de journal contenant l'historique de la transaction, s'avère nécessaire.

L'application de la technique de Validation à un environnement réparti présente le problème de la diffusion du Point de Validation [ANDRA 80] : pour une transaction répartie, le Point de Validation est celui d'une incarnation particulière chargée de le propager vers les autres incarnations participant dans la même transaction (diffusion d'un message de Validation) ; mais cette propagation peut être empêchée par une défaillance, et tant qu'une incarnation ne reçoit pas de message de validation, elle ne peut pas décider d'annuler ou de compléter la transaction localement.

Pour réduire la zone de doute de chaque incarnation dans le schéma antérieur de validation *en une étape*, un schéma *en deux étapes* est proposé (dans les approches de [SCOT 81 B ; LELAN 79 B ; BERNS 78 ; ROSEN 78 ; LAMPS 77], et d'autres) : avant de propager le Point de Validation, l'incarnation responsable diffuse un message de Prévalidation qui doit être acquitté par toutes les autres incarnations et qui a pour but principal la vérification du fait que ces incarnations sont vivantes. Si l'incarnation responsable reçoit tous les acquittements, elle initie la diffusion du message de Validation. La zone de doute de chaque incarnation se réduit maintenant à la zone comprise entre la réception du message de Prévalidation et celle du message de Validation. Si l'incarnation est interrompue par une défaillance :

- avant d'entrer dans sa zone de doute : alors elle décidera d'annuler la transaction localement, lors de sa récupération ;

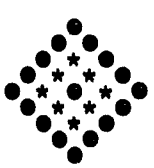
- après être sortie de sa zone de doute : alors elle déclarera de compléter la transaction,

- dans sa zone de doute : alors elle ne peut pas décider et doit consulter les autres incarnations (pour prendre une décision d'un commun accord).

La responsabilité de propager le Point de Validation d'une transaction répartie, est généralement assignée à son incarnation dans le site où la transaction est soumise par l'utilisateur et initialisée ; cela assure que l'utilisateur sera toujours informé du résultat de la transaction malgré les défaillances : en effet, cette incarnation, qui est la seule libre de doute, peut assurer sans délai la fin de la transaction (soit son achèvement ou son annulation lors de défaillances).

Par rapport au schéma de Validation en une étape, le schéma en deux étapes permet une meilleure fiabilité et une meilleure disponibilité de ressources, mais il entraîne une surcharge et un coût supérieurs à cause de l'échange additionnel de messages, et dans ce cas, certaines transactions se voient pénalisées (celles qui supposent un schéma simple d'exécution). La solution idéale est un protocole mixte comme celui proposé par le projet SCOT [SCOT 81 B] : le concepteur de l'application choisit le schéma le plus approprié pour chaque incarnation d'une transaction répartie.

La partition du réseau en deux sous-réseaux indépendants ne représente pas un problème différent lorsque les données sont partitionnées : la récupération est faite avec les mêmes schémas de validation et il n'y a pas de risque d'une évolution incohérente entre les deux sous-ensembles du système : seules les transactions possédant toutes les ressources nécessaires pourront être menées à terme.



CHAPITRE II

CONCEPTS RELATIFS A L'UTILISATION INTERACTIVE DES SYSTEMES REPARTIS

Après avoir caractérisé les systèmes informatiques répartis dans le chapitre I, nous voulons à présent nous situer dans le contexte spécifique des applications *interactives* conçues pour ces systèmes ; ces applications se caractérisent par l'existence d'un dialogue entre l'utilisateur et le système.

Des motifs semblables à ceux qui ont motivé l'apparition des systèmes informatiques répartis, expliquent le développement des applications interactives :

- D'une part, l'évolution de la technologie a permis le rapprochement du système de l'utilisateur au moyen d'équipements terminaux dont l'utilisation a été facilitée et perfectionnée.

- D'autre part, la recherche d'une meilleure adéquation aux exigences de l'utilisateur et d'une communication plus facile entre lui et le système constituent l'objectif primordial dans la conception des applications actuelles ; l'efficacité du calcul devient alors un objectif moins important [MARTI 73, Chapitre 1].

Seul le mode interactif peut satisfaire des besoins de l'utilisateur telles que [CHAMP 80, Chapitre 1 ; JAMES 80] : un accès à l'information plus rapide ; un mode de communication avec le système qui soit facile, tolérant aux erreurs comises par l'utilisateur et adaptable aux différentes classes d'utilisateurs selon leur niveau d'expérience ; et de possibilités de contrôle sur le déroulement de l'application de la part de l'utilisateur.

Pour répondre aux exigences antérieures de l'utilisateur, les premières applications interactives ont essayé d'adapter les techniques et les langages utilisés dans les applications par lot traditionnelles ; mais comme il est remarqué par Lafuente et Gries [LAFUE 78], cette adaptation se révèle inadéquate car les applications interactives répondent

à des nécessités bien différentes de celles qui ont inspiré les applications par lot. En fait, très peu de recherches ont essayé d'isoler les facteurs qui peuvent rendre plus agréable et plus humaine l'interface de l'utilisateur avec le système ; ce sont ces facteurs qui devraient guider la conception des applications interactives [STERL 74].

Dans ce chapitre nous essayons d'abord de caractériser la nature des applications interactives (partie 1) ; ensuite, nous voulons délimiter les exigences et les facilités demandées par l'utilisateur et le concepteur d'applications interactives (parties 2 et 3). La mise en place de ces facilités dans un système réparti introduit une complexité additionnelle dans le contrôle de cohérence et des problèmes spécifiques qui seront traités dans la dernière partie du chapitre (partie 4).

1 - NATURE DES APPLICATIONS INTERACTIVES

Les *applications interactives* où l'utilisateur établit un dialogue avec le système pour soumettre des transactions au moyen d'un terminal, sont devenues très populaires dans les secteurs de l'industrie, de la banque, de la gestion de places d'avion, etc. Ce type d'applications est apparu comme une alternative différente face aux *applications par lot* traditionnelles où il y a un regroupement de traitements en trains de travaux pour être exécutés de façon automatique et en différé. Ainsi, il est naturel de caractériser les applications interactives par opposition aux applications par lot, de la manière suivante [FRANK 79] :

- *Structure des applications* : Une application par lot peut être considérée comme une suite ininterrompue de traitements ; par contre dans les applications interactives, chaque transaction est conçue comme une séquence d'étapes de dialogue accompagnées d'étapes de traitement : dans les étapes de dialogue l'utilisateur fait ses requêtes de traitement et fournit l'information nécessaire ; le système vérifie la validité de cette information, réalise les traitements demandés et fournit les réponses en passant à une nouvelle étape de dialogue avec l'utilisateur.

- *Unité de traitement* : Dans les applications par lot, un traitement complet est réalisé pour chaque lot ou train de programmes contenant des transactions (non-interactives) de divers utilisateurs ; par contre, pour les applications interactives l'unité de traitement est la transaction soumise par l'utilisateur.

- *Critère de performance* : Dans les applications par lot, il est important d'obtenir un bon taux d'utilisation de l'installation, lequel se traduit par un débit élevé de programmes exécutés par unité de temps. Dans les applications interactives, il est plus important d'obtenir un bon temps de réponse à chaque requête de l'utilisateur.

- *Niveau de Concurrence* : A l'origine, les applications interactives ont été conçues pour une exécution purement séquentielle de programmes ; par contre, les applications interactives sont créées précisément pour supporter un environnement de transactions concurrentes : pour cela, elles doivent utiliser des mécanismes de contrôle pour l'accès aux ressources partagées et aussi pour limiter la durée de blocage de ces ressources.

- *Traitement d'Erreurs* : Dans les applications par lot, une erreur dans les données fournies par l'utilisateur entraîne l'abandon de son programme ; l'utilisateur doit alors la corriger et attendre une nouvelle exécution de son programme. Par contre, dans les applications interactives, la présence en-ligne de l'utilisateur lui permet de corriger ses erreurs au fur et à mesure qu'elles sont détectées par le système, sans entraîner un abandon de la transaction.

- *Iteraction utilisateur-système* : Par sa nature, les applications par lot ne permettent aucune intervention de l'utilisateur pendant le déroulement de son programme et tous les mécanismes de contrôle restent à la charge du système. Par opposition, les applications interactives doivent faciliter l'interaction de l'utilisateur avec le système en lui offrant :

  * un dialogue simple qui puisse être utilisé même par un utilisateur occasionnel, sans expérience informatique, et qui profite des ressources du terminal pour faciliter son travail ;

  * des possibilités de contrôle de la part de l'utilisateur sur le déroulement de sa transaction (notamment l'utilisateur devrait pouvoir interrompre ou annuler sa transaction ou revenir sur un contexte antérieur de la même transaction).

2 - EXIGENCES DE L'UTILISATEUR INTERACTIF (INTERFACE SYSTEME-UTILISATEUR)

D'après la caractérisation des applications interactives que nous venons de faire, il est évident que les mécanismes dérivés de ceux utilisés dans les applications par lot ne sont pas adéquats pour établir une réelle interaction entre le système et l'utilisateur. Par la suite, nous voulons délimiter les mécanismes possibles qui répondent effectivement aux exigences de l'utilisateur interactif, et ceci dans le contexte d'un système informatique réparti.

2.1 - Promptitude de réponse

En raison de sa présence en-ligne, l'utilisateur d'applications interactives exige du système une réponse rapide à ses requêtes. Les systèmes avec répartition de données apportent une première solution à cette exigence en plaçant les données plus près de l'utilisateur qui en a besoin. Le temps de réponse à une requête est défini alors comme le temps s'écoulant entre l'émission d'un message contenant la requête, et la réception de la réponse correspondante après cheminement dans le réseau de communication de la requête, traitement dans les sites concernés et cheminement de la réponse [MACCH 79, Chapitre 11] ; il s'en suit que pour obtenir un bon temps de réponse, la répartition des données doit être faite de manière a favoriser les requête locales.

A part la répartition des données, la répartition du traitement constitue un deuxième facteur pour obtenir un bon temps de réponse aux requêtes de l'utilisateur. Mais en fait, la performance réelle obtenue (en termes de temps de réponse) dépend des relations entre le degré de concurrence des transactions et la surcharge dans le système occasionnée par cette concurrence [BOUCH 81] : en effet, le gain dans le temps de réponse obtenu par le parallélisme interne et externe entre transactions, peut être contrecarré par la surcharge entraînée dans le volume de communications, dans la gestion de conflits d'accès aux données, dans les délais de synchronisation, etc.

2.2 - Facilités de dialogue

Le type d'utilisateur des systèmes informatiques est en train de changer rapidement [JAMES 80] : bientôt les utilisateurs sans expérience informatique constitueront la principale proportion de tous les utilisateurs. Ces utilisateurs manifestent une grande insatisfaction avec beaucoup de systèmes actuels, à cause de leur manque de facilité d'utilisation et demandent un dialogue simple pour se servir du système et obtenir l'information souhaitée ; ce dialogue devrait être conçu dans les termes d'une communication humaine et non plus comme un dialogue homme-machine [MARTI 73].

Afin d'établir une interface système-utilisateur agréable, le dialogue utilisé devrait présenter les propriétés suivantes [JAMES 80 ; HAYES 81 ; STERL 74] :

- *Facilité d'utilisation* : Le dialogue doit être facile à comprendre et à apprendre. Le système devrait instruire l'utilisateur de la façon d'accéder à ses services.

- *Adaptabilité* : Le système doit reconnaître les différents types d'utilisateurs et adapter le dialogue à l'expérience de chacun.

- *Mémorisation* : Le système doit garder une trace du dialogue pour permettre à l'utilisateur un retour possible à un contexte antérieur, ou une interruption temporaire de sa transaction.

- *Auto-suffisance* : Le système doit reconnaître les conditions exceptionnelles et réaliser les actions spéciales nécessaires sans requérir aucune aide de l'utilisateur.

- *Tolérance* : Le système doit reconnaître l'information fournie par l'utilisateur lorsqu'il fait des fautes lingüistiques ou d'autres erreurs ; le système doit corriger ces erreurs si cela est possible, ou demander à l'utilisateur de choisir entre alternatives d'interprétation, ou laisser à l'utilisateur la possibilité de les corriger lui-même.

- *Absence d'ambigüité* : Lorsque l'utilisateur fournit une information, le système doit lui permettre de savoir s'il est compris ou non, quelles sont les hypothèses qui sont faites, et lui permettre de corriger l'information ou de changer de telles hypothèses.

2.3 - Vérification de données en temps réel

Dans un système réparti toute erreur dans les données fournies par l'utilisateur doit être détectée de façon immédiate, afin d'éviter sa propagation dans l'ensemble des données ce qui menacerait la cohérence du système. L'utilisateur interactif exige en plus de cette détection immédiate de la part du système, l'opportunité de corriger ses données erronées en temps réel sans qu'il y ait besoin d'annuler sa transaction.

Avec l'évolution technologique des équipements terminaux, la vérification, la détection et le traitement d'erreurs dans les données fournies par l'utilisateur, deviennent des tâches assignées au terminal même : ainsi, un terminal dit "intelligent" (capable d'effectuer une partie du traitement des données) peut assister l'utilisateur dans la saisie des données, en lui informant au préalable le format des données à fournir et en vérifiant ensuite la validité de ces données [MACCH 79, Chapitre 9].

Les données que l'utilisateur peut fournir pendant une transaction appartiennent à trois types, d'après Carrez [CARRE 80] :

- *données de Définition* : qui précisent le type d'une opération à effectuer (par exemple, l'opération "Débiter" sur un compte bancaire) ;

- *données d'Accès* : qui définissent les données concernées par une opération (par exemple, le code du compte à débiter) ;

- *données de Traitement* : qui définissent les modification à apporter sur ces données (par exemple, le montant à débiter).

Pour chacun de ces types de données à saisir, la vérification de leur validité doit être réalisée dans les trois niveaux suivants [CARRE 80] :

- *Vérification Syntaxique* : le système doit vérifier que chaque donnée fournie par l'utilisateur appartient au format et type attendus ; en cas d'erreur, le système doit en informer l'utilisateur de façon immédiate afin de lui permettre la correction (par exemple, il faut vérifier que le montant à débiter soit une valeur numérique).

- *Vérification de Sémantique Intrinsèque* : le système doit vérifier les spécifications (ou contraintes d'intégrité) qu'une donnée saisie doit respecter et qui ne peuvent pas être simplement déduites de son type (par exemple, il faut vérifier qu'un code de compte appartient à un compte qui existe réellement). En cas de non respect de ces spécifications, l'utilisateur doit avoir aussi l'opportunité de corriger immédiatement la donnée.

- *Vérification de Sémantique d'Ensemble* : le système doit vérifier pour l'ensemble des données saisies le respect des relations d'intégrité qui doivent être maintenues entre toutes les données du système (par exemple, une série de montants pour créditer et/ou débiter un ou plusieurs comptes bancaires, ne doit pas entraîner le solde négatif d'aucun compte). En cas de non respect de ces relations, l'ensemble des données saisies doit être rejeté par le système et l'utilisateur doit pouvoir le fournir à nouveau.

2.4 - Contrôle de l'utilisateur

Avec les applications interactives réalisées dans un environnement réparti, le contrôle revient à l'utilisateur qui devient responsable de ses opérations et est par conséquence plus motivé. Pour pouvoir exercer ce contrôle, le système doit offrir à l'utilisateur des possibilités de décision et diverses options à choisir lors de la saisie de données, lors des erreurs qu'il commet et lors des défaillances du système ; aussi, l'utilisateur

doit pouvoir définir des spécifications de protection sur ses données. Nous examinons par la suite ces options de contrôle de l'utilisateur et les mécanismes possibles pour les mettre en place dans le système.

2.4.1 - Options de contrôle lors d'incidents

Les erreurs commises par les utilisateurs, les situations d'interblocage, les défaillances dans un site et dans le réseau de communication, sont tous des incidents ou des événements imprévus qui peuvent interrompre une ou plusieurs transactions en cours dans le système. Le système doit détecter de tels incidents et lancer automatiquement une action de récupération afin d'assurer la continuité du traitement et de préserver la cohérence des données partagées, selon les mécanismes présentés dans le chapitre I.

L'utilisateur interactif devrait être informé des décisions du système lors d'incidents, surtout lorsque ces décisions l'affectent directement. En plus, l'utilisateur devrait avoir la possibilité de participer à ces décisions, en utilisant pendant sa transaction les options suivantes :

- accepter les corrections de données faites par le système, ou les fournir à nouveau ;

- interrompre ou annuler la transaction ;

- demander au système un historique des actions déjà réalisées dans la transaction ;

- demander la sauvegarde de l'état de la transaction aux instants qu'il croit convenable ;

- revenir sur un état ou contexte antérieur de la transaction.

Aussi, lors de défaillances du système il est souhaitable que l'utilisateur ne soit pas soumis à l'annulation systématique de sa transaction en cours ; au contraire, dans ce cas l'utilisateur devrait être sûr qu'une portion de ses opérations sont effectivement réalisées et choisir parmi diverses alternatives de récupération.

2.4.2 - *Options de protection des données*

D'après Lanciaux [LANCI 78, Chapitre 1] le terme de *Protection* désigne les mécanismes qui contrôlent l'accès aux données du système afin de :

- maintenir l'intégrité des données en permettant uniquement les opérations définies sur elles,

- assurer la sélectivité d'accès et la variété des droits d'accès à une donnée d'après l'utilisateur.

En particulier l'utilisateur doit pouvoir se protéger de la modification, l'utilisation ou la destruction non autorisées de ses données. Il doit pouvoir exprimer des contraintes d'accès à ses données dans un langage approprié : par exemple dans le langage SQL (ancien SEQUEL 2) du système SYSTEM R [CHAMB 76], l'utilisateur dispose d'une commande (Grant) pour octroyer des privilèges sélectifs d'accès à ses données à d'autres utilisateurs ; ainsi dans la commande suivante :

"Grant Read, Update, Delete on Emp to Smith"

l'utilisateur spécifie qu'un autre utilisateur "Smith" est autorisé à lire, mettre à jour ou éliminer une relation de la Base de Données nommée "Emp" (d'Employés). Dans cet exemple, l'utilisateur peut révoquer ultérieurement tout ou partie des privilèges octroyés au moyen d'une commande de la forme :

"Revoke Read on Emp from Smith".

Dans les système répartis les mécanismes de protection deviennent plus complexes du fait qu'il y a une meilleure disponibilité des données aussi bien pour les accès autorisés que pour les accès non autorisés.

2.4.3 - *Mécanismes pour mettre en place les options utilisateur*

Pour permettre à l'utilisateur le retour de sa transaction à un état antérieur lors d'incidents, le système doit maintenir des journaux

du déroulement de chaque transaction permettant la sauvegarde des états susceptibles de retour. Ainsi par exemple dans le langage SQL [CHAMB 76] l'utilisateur spécifie les points de sa transaction où il veut que le système réalise une sauvegarde d'état avec la commande "Save <identificateur du point>" ; après, une commande "Restore <identificateur du point>" entraîne la remise du journal dans l'état souhaité.

Il existe un problème de stockage de ces journaux dû à la prolifération possible d'états sauvegardés. Une solution possible à ce problème est d'enlever du journal d'une transaction les états qui deviennent non nécessaires lorsque les données concernées ne sont plus utilisées. Par exemple, dans la méthode proposée par Carrez [CARRE 80], l'utilisateur structure sa transaction en blocs emboîtés spécifiant ainsi les différentes portées d'utilisation des données. Les états de la transaction définis par le début de chaque bloc sont sauvegardés et gérés en pile : ainsi, au début d'un bloc le système garde dans la pile des copies des données déclarées dans les blocs englobants et pendant le déroulement du bloc les mises à jour sont réalisées sur ces copies et sur des copies des données "locales" déclarées dans le bloc ; l'annulation du bloc lors d'incident (erreur ou défaillance) est faite simplement par la destruction des copies. Si le bloc se termine normalement les copies remplaceront les versions que les données possédaient avant le bloc et les copies des données locales seront détruites après modification effective de ces données.

La mise en place des options de protection peut être réalisée par des listes de droits d'accès contenues dans les descripteurs des données et plus généralement par des mécanismes à capacités ; les aspects de Protection des données sont en dehors de la portée de ce travail et ne feront pas l'objet des développements ultérieurs (voir à ce propos les travaux de Lanciaux [LANCI 78] et Delattre [DELAT 79]).

3 - EXIGENCES DU CONCEPTEUR D'APPLICATIONS INTERACTIVES

Le concepteur d'une application interactive est celui qui définit les caractéristiques du dialogue que l'utilisateur va soutenir avec le système : le concepteur défint ainsi l'interface système-utilisateur. Pour cela, le système doit offrir au concepteur un langage de programmation approprié.

D'après Winograd [WINOG 79], l'évolution dans les langages de programmation marque une tendance vers la *description de propriétés* des processus et des objets qu'ils manipulent, au lieu de la spécification détaillée des instructions à exécuter pour réaliser de tels processus. Dans cette voie le concepteur d'une application interactive va se servir du langage pour décrire le comportement dans le temps de l'interface système-utilisateur sans spécifier en détail le mécanisme qui le génère.

3.1 - Description de l'interface système-utilisateur

Dans les applications interactives, une transaction peut être considérée comme une séquence de "cadres", qui représentent des scènes de dialogue, avec des étapes de traitement intermédiaires ; ainsi une transaction pourrait être représentée par un graphe de cadres interconnectés par des étapes de traitement selon les choix faits par l'utilisateur [FAULL 80].

Le langage de programmation utilisé par le concepteur doit lui permettre de décrire la composition de chaque cadre de manière facile et indépendante des caractéristiques du terminal ; il doit pouvoir aussi définir la vérification des données de manière non-procédurale, c'est-à-dire sans nécessité de spécifier des procédures de vérification et traitement d'erreurs. Un exemple d'un langage qui vérifie ces propriétés est celui proposé par Lafuente et Gries [LAFUE 78] : chaque cadre est composé d'items attachés à des positions sur l'écran du terminal ; chaque item est constitué d'un ensemble d'informations associé à la saisie d'une donnée et appelé "structure de saisie". Au travers de la déclaration d'un item le concepteur spécifie :

- le nom de la variable associée à la donnée,

- le message accompagnateur qui sert à informer à l'utilisateur la semantique associée à la saisie de la donnée,

- le type et format de la donnée : ils définissent la vérification syntaxique que le système doit réaliser lors de la saisie de la donnée,

- les contraintes d'intégrité que le système doit vérifier aussi lors de la saisie de la donnée (vérification de semantique intrinsèque),

- etc...

3.2 - Spécification des assertions de cohérence

Dans le langage de gestion de cadres cité ci-dessus, le concepteur peut définir pour chaque cadre les contraintes d'intégrité ou "assertions de cohérence" que le système doit vérifier, une fois que toutes les données des items du cadre sont saisies : ceci définit la vérification de semantique d'ensemble pour ces données.

Dans le langage SQL de manipulation de Bases de Données [CHAMB 76] le concepteur définit les assertions de cohérence qui doivent être respectées en permanence au moyen de la commande : "Assert R1 on <relation> : <condition>". La vérification de ces assertions est faite par le système à la fin de toute transaction réalisée par un utilisateur : si les assertions ne sont pas satisfaites toute la transaction est annulée et retournée à son état initial. Le concepteur peut éliminer une assertion de cohérence au moyen d'une autre commande : "Drop assert R1".

Une proposition pour permettre la définition et modification dynamiques d'assertions de cohérence est faite par Sales [SALES 80] : A chaque assertion sont associés trois paramètres qui la définissent :

- son type : statique ou dynamique,

- sa durée de vie,

- ses conditions d'application : vérification de l'assertion chaque fois que les données concernées sont modifiées, ou simplement à la fin d'une transaction.

A chaque assertion peut être associé un ensemble de fonctions de manipulation : lecture, suppression, restriction ou élargissement de son domaine d'application, modification de ses attributs, transmission de ces fonctions à certains types d'utilisateurs. Ces assertions sont énoncées pour la première fois lors de la définition des données (des relations dans une Base de Données relationnelle) et peuvent être modifiées dynamiquement par la suite.

La mise en place des assertions de cohérence définies par le consepteur d'une application interactive entraîne leur évaluation et vérification fréquentes ce qui ralentit les opérations et affecte la performance du système. Pour les applications simples destinées à des utilisateurs sans expérience informatique, il semble préférable que le concepteur définisse toutes les transactions disponibles sans avoir à spécifier d'assertions de cohérence, chaque transaction exprimant des manipulations de données qui préservent leur intégrité ; l'utilisateur se limite alors à invoquer de telles transactions.

4 - LE PROBLEME DE L'EFFET DOMINO

La mise en place dans un environnement réparti des différentes options de contrôle offertes à l'utilisateur et au concepteur d'applications interactives, pose des problèmes de maintien de la cohérence des données du système. En particulier, la restauration du système dans un état cohérent antérieur lors de défaillances ou lors d'erreurs des utilisateurs devient un problème complexe à cause de l'interaction entre transactions concurrentes [KIM 79] : en effet, la demande d'un utilisateur de retour à un point antérieur P de sa transaction, entraîne l'annulation de l'information échangée entre cette transaction et les autres après le point P, ainsi que l'annulation de tous les accès réalisés après P aux données modifiées par la transaction en question ; en conséquence les autres transactions doivent aussi être restaurées dans un état antérieur, bien que ces retours ne soient ni provoqués ni souhaités par les autres utilisateurs. Dans le pire des cas une avalanche de propagations de retours peut survenir en entraînant l'annulation des traitements de toutes les transactions concurrentes : c'est *l'effet Domino*.

Pour résoudre le problème de l'effet Domino plusieurs approches sont possibles. Parmi les approches qui n'exigent pas de verrouillage des données pendant la réalisation d'une transaction Kim [KIM 79] distingue deux classes :

- *Approche dépendante du concepteur* : le concepteur de l'application structure les transactions qui peuvent être exécutées en parallèle de telle façon que les points de sauvegarde d'état puissent être établis de manière coordonnée en temps d'exécution. Pour cela on utilise le concept de *"conversation"* (concept introduit par Randell [RANDE 75]) : on considère que les utilisateurs réalisant des transactions sur un ensemble de données partagées font partie d'une conversation dont la définition est faite par tous. Une conversation entre un ensemble de transactions est délimitée par

* une ligne de récupération : constituée par l'ensemble de Points de Sauvegarde établis par les transactions avant d'effectuer des interactions entre elles,

* une ligne de test : constituée par un ensemble de tests de validation que toutes les transactions doivent réaliser à la fin de la conversation ; si au moins une transaction ne réussit pas ces tests toutes les transactions doivent être retournées à la ligne de récupération.

Les conversations peuvent être imbriquées pour permettre différents points de sauvegarde dans chaque transaction, mais en général cette approche entraîne une tâche complexe pour le concepteur puisqu'il doit déclarer toutes les interactions entre transactions.

- *Approche transparente au concepteur* : le concepteur structure chaque transaction de façon indépendante aux autres et c'est le système qui établit automatiquement les points de sauvegarde des transactions concurrentes. Pour éviter l'effet Domino, le système doit établir les points de sauvegarde en fonction de la direction du flux d'informations entre transactions et avec une fréquence suffisante [RUSSE 77] (par exemple, avant chaque opération d'envoi de messages) ; cela entraîne une surcharge dans les opérations du système.

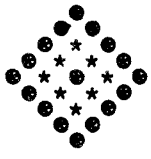
Parmi les approches pour résoudre le problème de l'effet Domino qui exigent un verrouillage des données pendant la réalisation d'une transaction, on peut considérer les classes suivantes :

- *Verrouillage long* : toute donnée accédée par une transaction est verouillée par le système pendant toute la durée de la transaction. Le système garantit ainsi que le retour d'une transaction à un point de sauvegarde puisse être réalisé par l'élimination des modifications et des verrous établis après ce point sans affecter pour autant les autres transaction. Cette méthode (utilisée par exemple, dans le système SYSTEM R [BLASG 81])

peut entraîner une longue attente d'une transaction pour les ressources bloquées par une autre, lesquelles ne sont libérées qu'à la fin de cette dernière transaction.

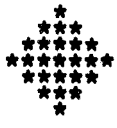
- *Verrouillage court* : pour une transaction structurée en blocs définissant la portée d'utilisation des données, chaque donnée peut être libérée à la fin du bloc qui l'utilise ; ceci permet la libération graduelle de ressources pendant une transaction et empêche les attentes trop longues aux autres transactions. La modification sur chaque donnée ne doit être effectuée qu'à la fin du bloc qui la déclare : ainsi l'utilisateur peut demander des retours dans le bloc plus interne qu'il est en train de réaliser ; mais une fois un bloc terminé l'utilisateur ne peut plus demander un retour à ce bloc : cette restriction est nécessaire pour éviter l'effet Domino.

En conclusion, la sélection d'une approche pour résoudre le problème de l'effet Domino, représente un compromis à faire entre les critères suivants : la flexibilité des retours permis dans une transaction, la durée du blocage des ressources utilisées par une transaction, la simplicité dans la programmation, la surcharge entraînée dans le système, etc...



CHAPITRE III

MODELE FONDAMENTAL DE CONTROLE DE COHERENCE DANS UN SYSTEME REPARTI DE FICHIERS : TRANSACTIONS ATOMIQUES



Dans le chapitre I nous avons délimité les problèmes de cohérence des systèmes informatiques répartis, et dans le chapitre II nous avons identifié les exigences de l'utilisateur interactif en environnement réparti. Dans la partie qui reste de ce travail, nous essayerons de répondre à de telles exigences tout en préservant la cohérence des données du système réparti considéré.

La conception d'un modèle général de contrôle de cohérence pour un système réparti, répondant aux diverses exigences de l'utilisateur interactif et aux différentes options de répartition, s'avère une tâche très complexe. Nous préférons commencer par présenter dans ce chapitre un modèle assez simple pour un système réparti de fichiers qui comprend un ensemble réduit d'options, mais qui permettra facilement d'être étendu et généralisé dans les chapitres suivants.

Le modèle de contrôle de cohérence que nous allons présenter, utilise des éléments présents dans diverses approches de la littérature des systèmes répartis, notamment les concepts de Transaction Atomique, Liste d'Intentions, Producteur et Consommateurs d'une transaction, Validation en deux étapes. Notre but est de rassembler ces divers éléments dans un seul modèle qui permette de caractériser totalement le contrôle de cohérence pour un système réparti de fichiers qui dessert des transactions interactives : nous allons donc spécifier les différents processus mis en jeu par les transactions, les processus "Moniteur" chargés de la préservation de la cohérence lors de concurrence de transactions et lors de défaillances du système, les structures de données et d'informations de contrôle nécessaires, etc. Notre souci initial est de donner à l'utilisateur un temps de réponse acceptable, et pour cela nous chercherons un niveau élevé de parallélisme entre transactions ; d'autres exigences et options de contrôle de l'utilisateur interactif seront considérées dans les chapitres suivants où une extention du modèle sera faite.

1 - HYPOTHESES ET CARACTERISATION INITIALE DU SYSTEME

1.1 - Le sous-système de transport

Nous considérons à présent un système réparti généralisé, constitué par un ensemble de sites indépendants reliés par un réseau de communication. Par rapport au sous-système de transport qui gère la communication entre les sites, on suppose que cette communication est réalisée par des échanges de messages qui suivent un protocole, pour lequel on fait quelques hypothèses courantes :

- Le délai de transmission d'un message d'un site à un autre est variable mais borné.

- Entre deux sites du système, l'ordre de réception de messages est identique à l'ordre d'émission.

- Il n'y a pas d'erreur de transmission qui altère le contenu des messages.

- Pour chaque message-demande envoyé par un site à un autre, le protocole exige un message-réponse contenant l'identification du message-demande et l'information demandée ou un acquittement. La non-réception d'un message-réponse dans un délai maximum de temps, sera interprétée par le site émetteur comme une situation d'isolement du site récepteur.

1.2 - La répartition des données : partition en fichiers

La duplication des données assure une meilleure disponibilité et fiabilité, mais elle entraîne des problèmes propres, déjà présentés dans le chapitre I, notamment la nécessité de maintenir la cohérence mutuelle entre les copies des données, la complexité dans le contrôle de concurrence et le stockage volumineux des données. Avec le souci de consi-

dérer une approche simple de base qui puisse conduire à des solutions plus générales, nous préférons adopter l'option de Partition des données.

Nous allons considérer alors un système où l'ensemble total des données est divisé en sous-ensembles disjoints assignés aux différents sites. Dans chaque site, les données qui y sont stockées composent un ou plusieurs objets de type "*fichier*" sur lesquels l'utilisateur peut réaliser un ensemble d'opérations bien délimité ; chaque site gère un ou plusieurs fichiers mais chaque fichier réside sur un seul site.

Par rapport à la configuration des fichiers, on suppose de façon générale que chaque fichier est un ensemble ordonné de "*segments*", où le segment constitue l'unité logique et indivisible de transfert d'information. Une *carte de segments* est associée à chaque fichier contenant la liste de descripteurs des segments du fichier ; un descripteur d'un segment contient au minimum son adresse physique et éventuellement des droits d'accès et d'autres informations. Ces hypothèses sur la configuration des fichiers sont adoptées dans plusieurs approches de systèmes répartis de fichiers ([LAMPS 77], [ISRAE 78], [PAXTO 79], [MUXER 79]) et elles ne sont pas très contraignantes puisqu'on peut supposer encore une structure quelconque pour les fichiers du système : séquentielle, indexée, etc...

Un exemple de configuration de fichier répondant aux hypothèses antérieures est la suivante :

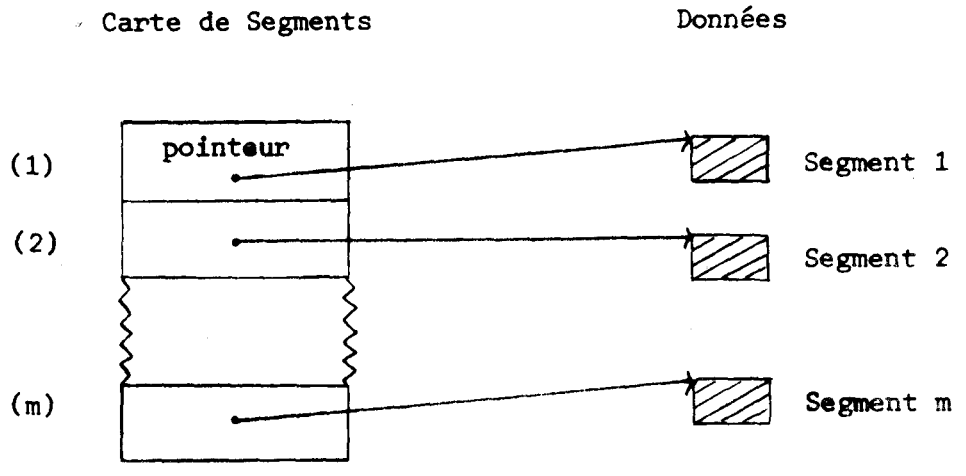


Fig. 1 : Configuration d'un fichier de données.

1.3 - Les transactions

Les utilisateurs interagissent avec le système par l'intermédiaire de transactions qu'ils soumettent aux sites, à l'aide de terminaux et dans le cadre de sessions interactives.

Une *transaction* est constituée d'une séquence de *commandes* contenant les opérations que l'utilisateur veut réaliser sur les fichiers du système ; les commandes spéciales DEBUT-TR et FIN-TR servent à délimiter une transaction.

Nous ne considérons pas le problème de vérification des contraintes d'intégrité du système pour chaque transaction : nous supposons donc que chaque transaction soumise par l'utilisateur est une unité de cohérence dans le sens défini par Eswaran et Gray [ESWAR 76], c'est-à-dire qu'elle transforme le système d'un état cohérent en un nouvel état cohérent. Nous nous intéressons plutôt aux problèmes de la préservation de la cohérence des données lors de concurrence de transactions et lors de défaillances du système.

Nous considérons initialement trois opérations possibles sur un fichier, ce qui donne lieu aux commandes suivantes :

- La commande OUVRIR (f, v) demande au système le droit d'accéder au fichier f en mode v (deux modes possibles sont considérés initialement : le mode "S" partagé et le mode "X" exclusif). L'utilisateur ignore la localisation du fichier f, lequel peut résider sur n'importe quel site du système.

- La commande LIRE (f, i, b) demande la lecture du segment i du fichier f et le placement des données lues dans la zone-tampon b, locale au site de l'utilisateur.

- La commande ECRIRE (f, i, b) demande l'écriture des données contenues dans la zone-tampon locale b sur le segment i du fichier f.

En plus nous considérons la commande spéciale ANNULER qui permet à l'utilisateur de demander au système l'annulation des opérations réalisées dans sa transaction.

Le style de transactions considéré pourrait paraître trop simple, mais en fait, on peut présumer qu'il s'agit d'un schéma de transactions statiques dans lequel l'utilisateur utilise les commandes pour faire venir sur son site les données d'autres sites (et inversement pour les renvoyer), et qu'il dispose d'autres opérations locales au niveau de son terminal pour modifier ces données. En considérant comme commandes des transactions uniquement les opérations d'accès aux données partagées du système, nous espérons mieux isoler les problèmes de cohérence posés par ce partage.

1.4 - La répartition du traitement : producteur et consommateurs d'une transaction

La réalisation d'une transaction entraîne un traitement réparti pris en charge par les sites concernés par les commandes de la transaction. Dans le chapitre I (partie 1) nous avons présenté quatre options de répartition du traitement qui donnent lieu à quatre schémas fonctionnels possibles pour un système (définis par Wills en [WILLS 79, Chapitre 5]) ; ce sont : le schéma de Contrôle Centralisé, le schéma de Contrôle Local, le schéma de Contrôle Partiellement Réparti et le schéma de Contrôle Totalement Réparti.

Les deux premiers schémas ne permettent qu'une exécution séquentielle des transactions, ce qui peut entraîner une longue période d'attente avant qu'une transaction puisse être exécutée. Les deux derniers schémas permettent l'exécution en parallèle de transactions qui ne sont pas en conflit, mais si le nombre de sites est élevé, le schéma de Contrôle Totalement Réparti peut aussi entraîner un long temps de réponse à cause de la visite séquentielle des sites.

Nous adoptons alors le schéma de Contrôle Partiellement Réparti qui semble le plus approprié pour atteindre l'objectif de donner un bon

temps de réponse à l'utilisateur. Ce schéma entraîne pour chaque transaction, un dialogue entre son site "responsable" et les autres sites concernés, pour décider de l'acceptation ou du rejet de la transaction. Avec ce schéma on peut envisager un degré maximum de concurrence qui comprend le parallélisme interne et externe entre transactions.

Plus spécifiquement, nous adoptons la terminologie du modèle "Producteur-Consommateurs", utilisé dans le projet SIRIUS [LELAN 81] : chaque transaction implique deux types de processus : le processus *Producteur*, associé au site où la transaction est soumise par l'utilisateur et les processus *Consommateurs*, associés aux sites possédant les fichiers concernés par la transaction. Le Producteur, chargé de la coordination de la transaction, doit envoyer aux Consommateurs les messages-demande nécessaires pour accomplir les commandes de la transaction. Les Consommateurs doivent réaliser les accès aux fichiers et retourner au Producteur les messages-réponse correspondants. Il existe alors une structure logique de communication de type "étoile" entre le Producteur et les Consommateurs d'une transaction.

En toute généralité, nous considérons qu'il est possible qu'un site soit simultanément Producteur et Consommateur d'une transaction, où même plus, Producteur et/ou Consommateur de *diverses* transactions concurrentes présentes dans le système (plusieurs processus de type Producteur ou Consommateur pouvant alors exister dans un seul site). Par exemple, considérons les deux transactions suivantes :

Transaction T1 :

```
- DEBUT-TR
    OUVRIR (F, X)
    OUVRIR (G, X)
    LIRE (F, i, a)
    LIRE (G, i, b)
    ECRIRE (F, i, b)
    ECRIRE (G, i, a)
- FIN-TR
```

Transaction T2 :

```
- DEBUT-TR
    OUVRIR (H, S)
    LIRE (H, j, c)
    OUVRIR (G, X)
    ECRIRE (G, i, c)
- FIN-TR
```

Supposons que T1 et T2 sont soumises aux sites S1 et S2 respectivement et que les fichiers référencés par ces transactions (F, G et H) sont localisés de la manière suivante :

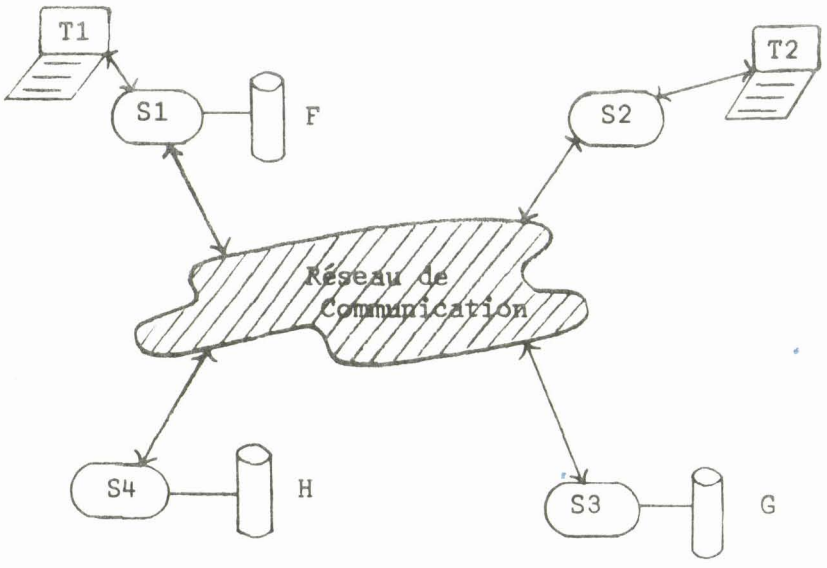


Fig. 2 : Localisation des fichiers référencés par les transactions T1 et T2.

On remarque alors que :

- le site S1 est Producteur et Consommateur de la transaction T1,
- le site S2 est Producteur de T2,
- le site S3 est Consommateur de T1 et Consommateur de T2,
- le site S4 est Consommateur de T2.

2 - CONCEPTS DE TRANSACTION ATOMIQUE ET DE LISTE D'INTENTIONS

Une solution pour préserver la cohérence des données, dans un environnement de transactions concurrentes sujettes à des défaillances est de réaliser chaque transaction de façon "*atomique*" [LAMPS 77] : cela signifie exécuter comme un acte indivisible toutes les modifications aux données apportées par la transaction, de telle manière qu'on ne puisse obtenir que les deux possibilités suivantes en cas de défaillance : ou tous les effets de la transaction ont été réalisés ou aucun.

Pour obtenir cet effet atomique, chaque transaction doit être réalisée en utilisant une "*Liste d'Intentions*", concept introduit par Lampson et Sturgis [LAMPS 77] : l'exécution, d'une transaction consistera alors à verrouiller les fichiers référencés et à élaborer une Liste d'Intentions contenant les modifications à faire sur ces fichiers ; si la transaction se termine normalement, ces intentions seront rendues effectives et les fichiers seront libérés après modification ; si la transaction se termine anormalement, les intentions seront ignorées et les fichiers seront libérés intacts.

Grâce à ce mécanisme, l'exécution de transactions concurrentes donne le même effet que celui produit par une exécution purement séquentielle des transactions, ce qui assure la préservation de la cohérence des données [ISRAE 78]. (En fait, on va utiliser ce mécanisme de Liste d'Intentions pour obtenir des ordonnancements légaux de transactions Bien-Formées et à Deux-Phases, c'est-à-dire des ordonnancements cohérents, d'après les résultats de Eswaran et Gray [ESWA 76]).

Dans notre système réparti de fichiers, la Liste d'Intentions d'une transaction sera réalisée de manière répartie par ses Consommateurs : chaque Consommateur construira une Liste d'Intentions locale contenant les modifications à effectuer sur les fichiers locaux.

3 - ETAPES ET ETATS D'UNE TRANSACTION ATOMIQUE

A la fin normale d'une transaction, son Producteur est chargé de diffuser aux Consommateurs l'ordre de rendre effectives les intentions de modification des fichiers. Mais cette diffusion peut être empêchée totalement ou partiellement, à cause d'une défaillance des lignes de communication ou d'un site Consommateur, menaçant ainsi l'atomicité de la transaction (et en conséquence la cohérence du système) : en effet, quelques Consommateurs pourraient réaliser leurs intentions tandis que d'autres non.

Pour préserver l'atomicité des transactions lors de défaillances, nous adoptons le schéma de Validation en deux étapes, déjà présenté dans le chapitre I (partie 5.2.3). L'application de ce schéma à notre système entraîne la réalisation de chaque transaction en trois étapes :

- Dans l'*Etape I de Synchronisation* les processus Producteur et Consommateurs sont créés, les fichiers référencés par la transaction sont verrouillés et le traitement des commandes a lieu. Un tel traitement se traduit en l'élaboration de Listes d'Intentions locales par les Consommateurs, mais les fichiers mêmes ne sont pas modifiés dans cette étape.

- Dans l'*Etape II de Prévalidation* déclenchée par la commande FIN-TR, le Producteur diffuse aux Consommateurs un message qui sert deux propos : vérifier que chaque Consommateur est "vivant" (c'est-à-dire qu'il ne se trouve pas isolé à la suite d'une défaillance), et l'informer de la liste totale des Consommateurs impliqués par la transaction (information que lui permettra éventuellement de consulter les autres Consommateurs lors d'une défaillance). Lorsque le Producteur reçoit les acquittements correspondants de tous les Consommateurs, il marque le Point de Validation de la transaction (point de non-retour) et déclenche l'étape III.

- Dans l'*Etape III de Validation*, le Producteur diffuse aux Consommateurs l'ordre de rendre effectives les intentions sur les fichiers et de libérer ces fichiers.

Une transaction peut se trouver en différents états qui sont en rapport avec ses étapes, le Producteur et les Consommateurs possédant chacun sa propre version de l'état courant de la transaction ; ces versions permettront de prendre la décision d'annuler ou de compléter la transaction lors de défaillances.

Pour le *Producteur*, l'état de la transaction (*étatr*) peut prendre trois valeurs avec les significations suivantes :

- INITIAL : la transaction peut être annulée sans effets ;

- VALIDATION : toutes les intentions consignées par les Consommateurs doivent être rendues effectives ;

- COMPLET : toutes les intentions ont été faites.

Pour chaque *Consommateur*, l'état de la transaction (*étatr*) peut prendre quatre valeurs avec les significations suivantes :

- INITIAL : la transaction peut être annulée localement sans effets ;

- PREVALIDATION : le Consommateur n'est pas sûr que les intentions locales doivent être rendues effectives ou non ;

- VALIDATION : toutes les intentions locales doivent être rendues effectives ;

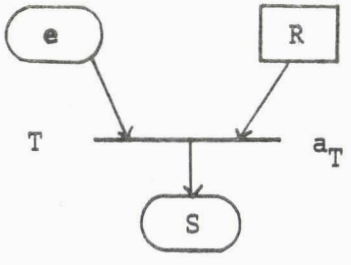
- COMPLET : toutes les intentions locales ont été faites.

4 - REALISATION D'UNE TRANSACTION ATOMIQUE

4.1 - Formalisme employé : les Réseaux de Nutt

Nous allons décrire les algorithmes des processus Producteur et Consommateur d'une transaction en utilisant le formalisme graphique des Réseaux d'Evaluation de Nutt [NOE 73]. Ce formalisme constitue un moyen clair, efficace et très approprié pour exprimer les processus dans les systèmes répartis où le traitement est dirigé par des messages (il est utilisé par exemple dans [ELLIS 77 ; WILLS 79 ; ROLIN 80 ; SCOT 81 A]) ; en fait, les Réseaux de Nutt, dérivés des Réseaux de Pétri, permettent de noter graphiquement les émissions et les réceptions de messages contenant des requêtes, les transitions d'état et les actions déclenchées dans un processus par l'arrivée d'une requête, ce qui permet leur traduction rapide dans un langage de programmation. Les Réseaux de Nutt peuvent être utilisés dans toutes les étapes de la vie d'un système : pour la spécification et mise au point du système, pour réaliser des preuves formelles sur ses spécifications, pour définir et interpréter des mesures de performance, etc. [ROLIN 80].

Le schéma de base d'un Réseau de Nutt est le suivant [WILLS 79, Annexe 1] :



e : état d'entrée du processus
R : requête arrivée
T : transition associée à l'action a_T
S : état de sortie du processus

Fig. 3 : Schéma de base des Réseaux d'Evaluation de Nutt.

La transition T est déclenchée uniquement si le processus se trouve dans l'état e et si la requête R arrive, et dans ce cas, le processus réalise l'action a_T et passe à l'état S pour attendre une nouvelle requête.

Nous utiliserons la notation suivante dans les graphes :

- Etat d'un processus : 

- Requête arrivée :

 message-demande (DEM) ou message-réponse (REP) venant d'un autre site ;

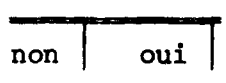
 commande venant du terminal.

- Transitions :

——— *a* Transition associée à la réalisation d'une *action locale a* qui peut être de la forme :
* assigner une valeur à une variable (Ex. : V : = 1) ;
* une action sur une liste L : INS_L [élém], CON_L [élém], EXT_L [élém] (insérer, consulter, ou extraire un élément de L) ;
* une opération sur des données d'un fichier local : lire, écrire, etc ;
* afficher un message sur le terminal de l'utilisateur.

· · · ◁—— DEM_L [texte] Transition associée à la *diffusion* d'un message-demande à un ensemble de sites L.

· ◁—— *m* Transition associée à la *transmission* d'un message *m* à un autre site. Le message *m* peut être de deux classes :
* DEM_{S_K} [texte] message-demande envoyé au site S_K
* REP_{S_K} [texte] message-réponse envoyé au site S_K



Transition associée à l'*évaluation* d'une condition booléenne C.

-*-*-*-*-*-*-*-
--*-*-*-*-*
-*-*-*-*-*-*-*-

Les graphes de Réseaux de Nutt des figures 4 et 5 décrivent la réalisation d'une transaction atomique à l'aide de Listes d'Intentions. Par la suite, nous donnons les explications concernant ces graphes.

4.2 - Etape de synchronisation

Dans l'étape de Synchronisation d'une transaction t, (soumise à un site S_j) a lieu le traitement de ses commandes. Les actions déclenchées par les commandes DEBUT-TR, OUVRIR, ECRIRE et LIRE sont représentées dans la figure 4 et peuvent être résumées de la manière suivante [FRANK 81] :

4.2.1 - *Commande* DEBUT-TR.

La nouvelle transaction reçoit un identificateur t (par un mécanisme qui sera expliqué postérieurement) et son processus Producteur est créé et activé dans le site S_j.

P_t initialise un *Descripteur* de la transaction (DES) avec l'information suivante :

* l'identificateur t de la nouvelle transaction (*tr*)
* l'état de la transaction (*etatr*) marqué à INITIAL
* l'identificateur S_j du site Producteur (*site-p*)

Ensuite P_t se met en connexion avec le terminal de l'utilisateur pour recevoir les autres commandes de la transaction.

4.2.2 - *Commande* OUVRIR (f, v).

Il doit exister un répertoire ou *Table de fichiers* (TF) totalement dupliquée sur chaque site, indiquant la localisation de chaque fichier du système.

P_t consulte dans TF le site de résidence du fichier f, disons S_K.
Si S_K ne figure pas encore dans sa *Liste de Consommateurs* LC (initialement vide), alors P_t insère S_K dans LC et envoie au site S_K le message-demande "DEBUT-TR (t, S_j, S_K)" pour provoquer l'activation d'un processus Consommateur, après quoi P_t finit par envoyer au Consommateur du site S_K le message-demande "OUVRIR (f, v)".
Si S_K figurait déjà dans sa Liste de Consommateurs LC, alors P_t envoie au Consommateur du site S_K simplement le message-demande "OUVRIR (f, v)".

Lorsqu'un site S_K reçoit un message-demande "DEBUT-TR (t, S_j, S_K)", alors un processus Consommateur C_t pour la transaction t est créé et activé :

C_t initialise un *Descripteur* local de la transaction (DES) avec l'information suivante :
* l'identificateur t de la transaction (*tr*)
* l'état de la transaction (*etatr*) marqué à INITIAL
* l'identificateur S_j du site Producteur (*site-p*)
* l'identificateur S_K du site Consommateur (*site-c*)

Ensuite C_t envoie au Producteur le message-réponse exigé par le protocole de communication (un simple acquittement dans ce cas).

Lorsque le Consommateur C_t du site S_K reçoit le message-demande "OUVRIR (f, v)", les actions suivantes sont réalisées :

C_t verrouille le fichier f en mode v et insère l'identificateur f dans sa *Liste d'Intentions* (LI). Cette liste va contenir pour chaque fichier consigné f, une *liste des modifications* à effectuer sur ce fichier (LM_f). Finalement, C_t envoie un acquittement au Producteur.

4.2.3 - *Commande* ECRIRE (f, i, b).

Nous supposons ici que lorsque l'utilisateur utilise cette commande, le fichier f a été ouvert auparavant en mode X, et que la zone-tampon locale b contient des données que l'utilisateur veut écrire dans le segment i du fichier f.

P_t envoie au Consommateur du site S_K, où le fichier f réside, le message-demande "ECRIRE (f, i, contenu (b))" avec les données contenues dans la zone-tampon b.

Les fichiers ne sont pas modifiés durant la transaction : toute nouvelle écriture a lieu sur un segment physique libre dont l'adresse est enregistrée dans la Liste d'Intentions locale. Ainsi, à la fin de la transaction il suffira d'actualiser la Carte de Segments de chaque fichier concerné avec les adresses nouvelles de ses segments. Dans ce but, lorsque le Consommateur C_t du site S_K reçoit le message-demande "ECRIRE (f, i, (données))", les actions suivantes sont réalisées :

C_t consulte sa Liste d'Intentions (LI) : si le segment i du fichier f n'a pas fait l'objet d'une écriture antérieure, alors C_t écrit les données sur un segment physique libre, d'adresse *ad-n* et enregistre dans la liste de modifications de f (LM_f) le couple d'informations <i, *ad-n*>. S'il s'agit d'une nouvelle écriture sur le même segment logique i du fichier f, alors C_t réutilisera le même segment physique libre de la première écriture. Finalement, C_t envoie un acquittement au Producteur.

4.2.4 - Commande LIRE (f, i, b).

On suppose que le fichier f a été ouvert auparavant en mode S ou X.

P_t envoie au Consommateur du site S_K (où le fichier f réside) le message-demande "LIRE (f, i, b)" et se met en attente du message-réponse correspondant, lequel doit retourner les données demandées.

C_t (du site S_K) après réception du message-demande "LIRE (f, i, b)", consulte sa Liste d'Intentions (LI) : Si le segment i du fichier f n'a pas fait l'objet d'une écriture antérieure, alors les données sont lues à l'adresse "ancienne" du segment i (*ad-a*) indiquée dans la Carte de Segments de f ($CARTE_f$) ; dans le cas contraire, les données sont lues à la nouvelle adresse du segment i (*ad-n*) indiquée dans la liste de modifications de f (LM_f). Finalement, C_t envoie au Producteur le message-réponse "ACK-LIRE (b, (données))" contenant les données lues.

P_t après avoir reçu ce message-réponse, place les données dans la zone-tampon locale b.

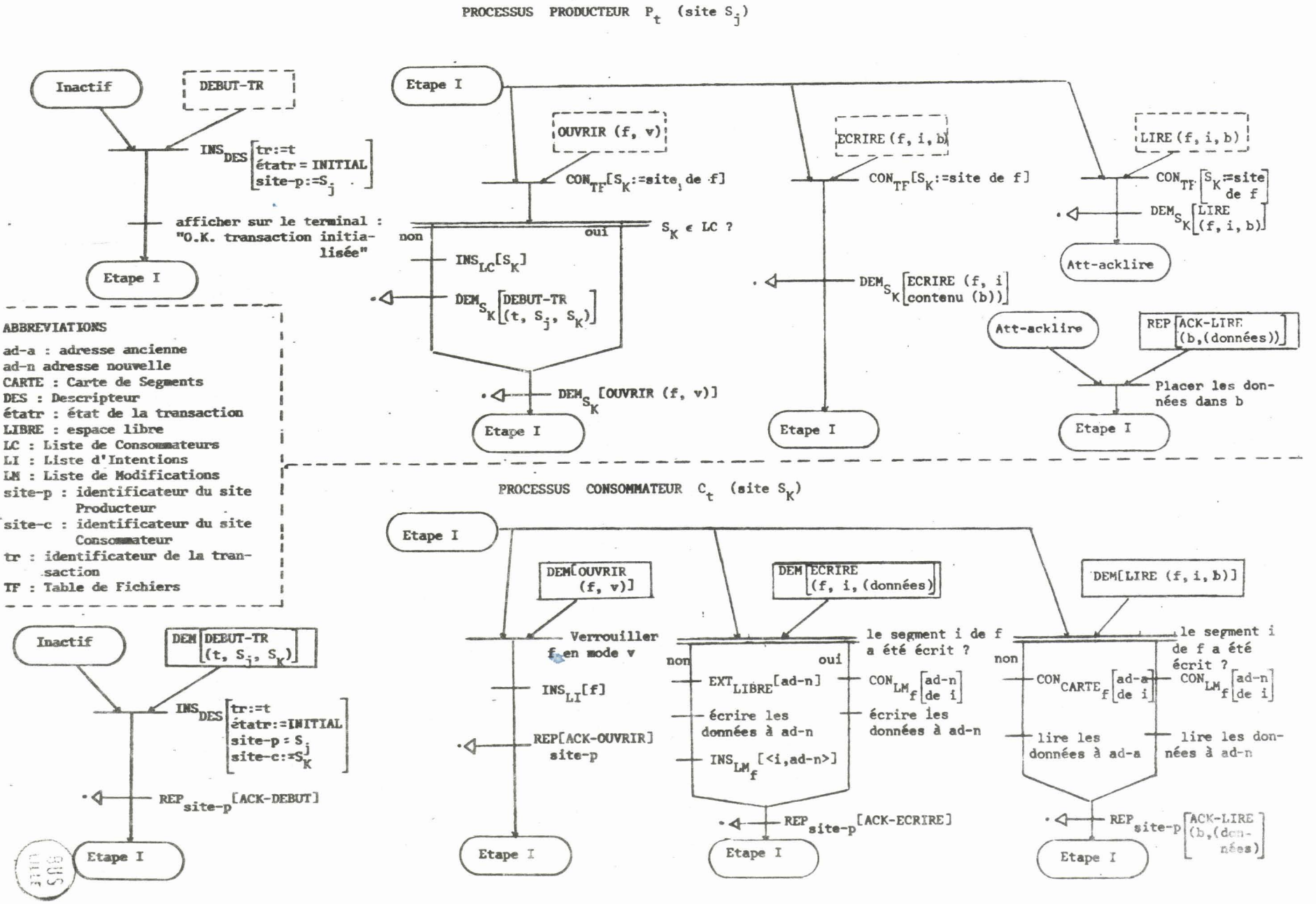


Fig. 4 : Etape de SYNCHRONISATION d'une transaction atomique.

4.3 - Etape de prévalidation

La figure 5 représente les actions déclenchées par la commande FIN-TR correspondant à l'étape de Prévalidation de la transaction. Ces actions peuvent être résumées ainsi :

P_t diffuse à tous les Consommateurs un message-demande "FIN-TR (contenu(LC))" contenant les identificateurs de tous les sites Consommateurs de la transaction.

Chaque C_t après réception du message-demande, place les identificateurs dans sa propre Liste de Consommateurs (LC). En plus, C_t marque son *état* à PREVALIDATION et envoie un acquittement au Producteur.

P_t attend les acquittement de *tous* les Consommateurs pour marquer son *état* à VALIDATION, après quoi, il diffuse à tous les Consommateurs un message-demande "VALIDER" qui constitue l'ordre de rendre effectives toutes les intentions de modification sur les fichiers. Ce moment constitue un point de non-retour de la transaction : à partir de là, la transaction pourra être terminée correctement malgré les défaillances qui peuvent arriver, et cela grâce aux mécanismes de récupération qui seront définis postérieurement (partie 5 de ce chapitre). En conséquence, P_t peut notifier à l'utilisateur que la transaction est terminée.

4.4 - Etape de validation

La figure 5 représente aussi les actions déclenchées par la diffusion du message-demande "VALIDER" du Producteur à tous les Consommateurs.

Chaque C_t après réception du message-demande, marque son *état* à VALIDATION et réalise les intentions locales : pour chaque fichier f figurant dans la Liste d'Intentions (LI), sa Carte de Segments ($CARTE_f$) est mise à jour avec les adresses nouvelles des segments qui ont fait l'objet d'écritures, adresses qui sont indiquées dans la liste de modifications de f (LM_f) ; les adresses

anciennes de ces segments sont délivrées à l'espace libre. Une fois que tous les fichiers sont mis à jour et libérés (déverrouillés), C_t marque son *état* à COMPLET, envoie un acquittement au Producteur et lui-même devient inactif.

P_t attend les acquittements de tous les Consommateurs pour marquer son *état* à COMPLET, après quoi la transaction est totalement terminée et le processus Producteur P_t devient inactif.

BIU LILLE

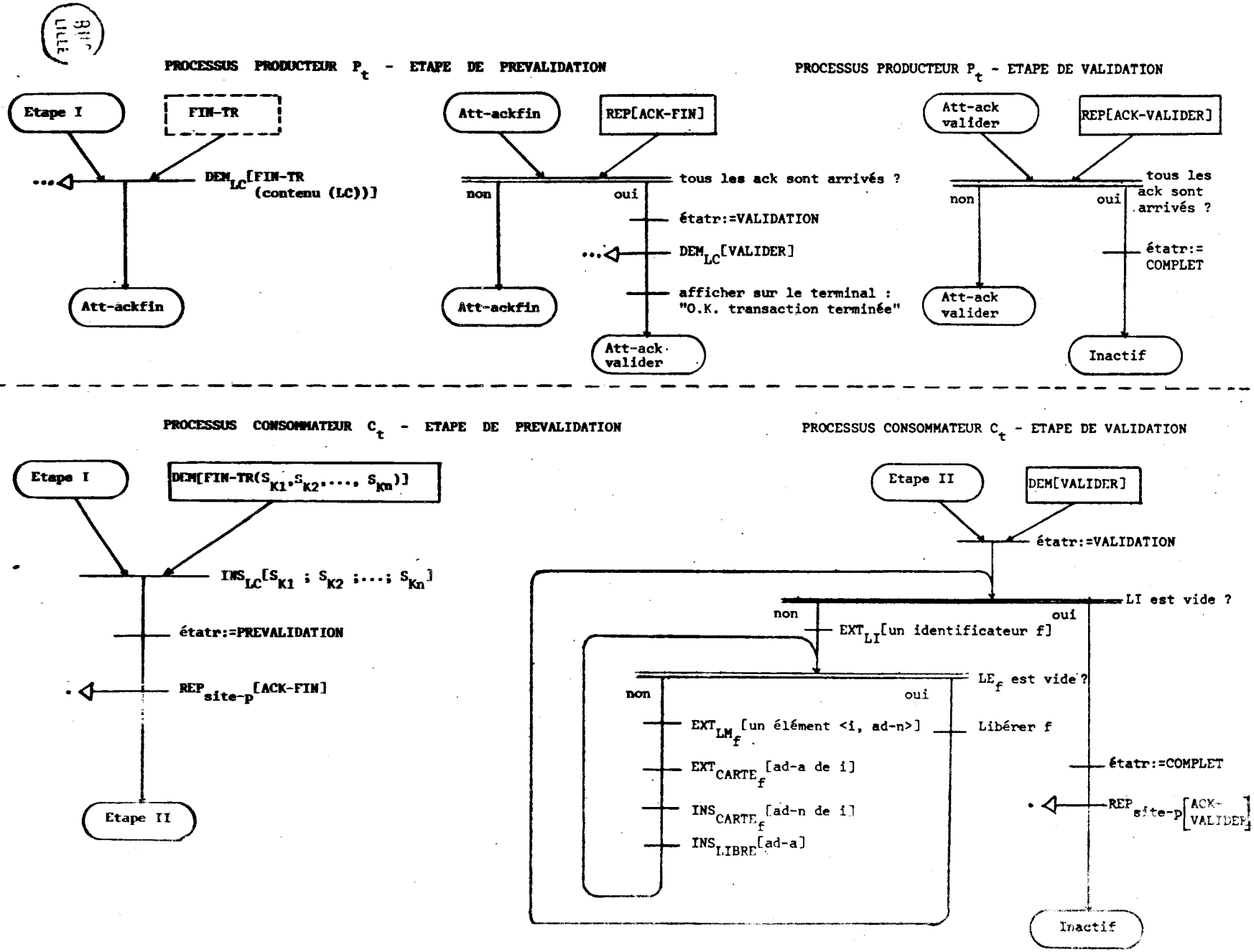


Fig. 5 : Etape de PREVALIDATION et de VALIDATION d'une transaction atomique.

5 - TRAITEMENT DES SITUATIONS D'EXCEPTION

Les situations d'exeption qui peuvent arriver pendant le déroulement d'une transaction sont fondamentalement les demandes d'annulation faites par l'utilisateur et les défaillances du système. Pour les premières, le système doit remettre la transaction dans son état initial après avoir annulé tous les effets des opérations réalisées : les fichiers concernés doivent se retrouver sans modification apportée par la transaction annulée.

Par rapport aux défaillances, nous considérons que du point de vue d'une transaction, toute défaillance dans le système (i.e. défaillance d'un site, défaillance de ligne logique avec une éventuelle partition du réseau de communication ou défaillances multiples) est interprétée comme une situation d'*isolement* du site Producteur ou d'un site Consommateur (ou de plusieurs) à une étape quelconque de la transaction. Pour faire face à cette situation nous appliquerons la technique de récupération de Validation (plus précisément le schéma de Validation en deux étapes) présentée dans le chapitre I (partie 5.2.3) : cette technique implique de compléter la transaction si elle était arrivée à son point de Validation (celui où le Producteur marque l'état de la transaction à VALIDATION) avant la défaillance, ou d'annuler la transaction dans le cas contraire.

Nous décrivons par la suite les actions que le Producteur et les Consommateurs d'une transaction doivent réaliser pour faire face aux situations d'exception.

5.1 - Demande d'annulation faite par l'utilisateur

L'utilisateur dispose de la commande "ANNULER" pour demander au système l'annulation de toutes les opérations réalisées dans sa transaction ; une fois que le système accomplit cette commande, l'utilisateur peut demander de nouvelles commandes pour remplacer celles qui ont été annulées (et même il peut demander une nouvelle annulation). De toutes

façons, une fois que l'utilisateur demande la fin de la transaction (commande FIN-TR) il ne peut plus demander une annulation de sa transaction (cette restriction va éviter le risque de l'effet Domino).

Pour une transaction t, la commande ANNULER déclenche les actions suivantes dans les processus Producteur (P_t) et Consommateurs (C_t) :
P_t diffuse à tous les Consommateurs le message-demande "ANNULER".

Chaque C_t après réception du message-demande, annule les intentions locales : pour chaque fichier f figurant dans la Liste d'Intentions (LI), les segments physiques pris pour réaliser les écritures sont retournés à l'espace libre (les adresses de ces segments sont indiquées dans la liste de modifications de f : LM_f). C_t applique ce traitement aux fichiers en question et les libère : en fait, les fichiers n'ont jamais été modifiés par la transaction puisque le point de Validation n'a pas encore été atteint. Finalement, C_t marque son *étatr* à INITIAL et envoie un acquittement au Producteur.

P_t attend les acquittements de tous les Consommateurs pour marquer son *étatr* à INITIAL et notifier à l'utilisateur que la transaction é été annulée (remise à l'état initial). P_t est prêt alors pour recevoir de nouvelles commandes.

5.2 - Isolement du producteur

Cette situation n'est pas détectée par les Consommateurs, qui, en conséquence, n'entreprennent aucune action. C'est le Producteur qui doit déclencher la récupération de la transaction une fois redevenu actif ; pour cela, il examine sa version de l'état de la transaction (*étatr*) pour prendre la décision adéquate, ainsi :

Cas *étatr* :

INITIAL : Annuler la transaction : diffuser à tous les Consommateurs le message-demande "ANNULER".

VALIDATION : Compléter la transaction : diffuser à tous les Consommateurs le message-demande "VALIDER".

COMPLET : Devenir inactif (la transaction avait été terminée avant la défaillance).

Fin-cas.

5.3 - Isolement d'un consommateur

Le manque d'un message-réponse (ou acquittement) de la part d'un Consommateur isolé, dans un délai maximum de temps est signalé au Producteur de la transaction. Le Producteur déclenche alors la récupération de la transaction en réalisant le même procédé qui a été présenté pour le cas d'isolement du Producteur.

De sa part, le Consommateur isolé réalise un procédé de réinsertion, une fois qu'il redevient actif ; pour cela, il examine sa version de l'état de la transaction (*étatr*) pour prendre la décision adéquate, ainsi :

- Cas *étatr* :

  INITIAL : Annuler localement la transaction : réaliser les actions qui correspondent à la réception d'un message-demande "ANNULER".

  VALIDATION : Compléter localement la transaction : réaliser les actions qui correspondent à la réception d'un message-demande "VALIDER".

  COMPLET : Envoyer un acquittement au Producteur ("ACK-VALIDER") et devenir inactif.

  PREVALIDATION : Consulter le Producteur ou les autres Consommateurs pour décider :

  - Si le Producteur n'est pas isolé
    - alors - Si pour le Producteur *étatr* = VALIDATION
      - alors Compléter localement la transaction
      - sinon Annuler localement la transaction
      - Fin-si
    - sinon - Si au moins pour un autre Consommateur *étatr* = VALIDATION ou COMPLET
      - alors Compléter localement la transaction
      - sinon - Si au moins pour un autre Consommateur *étatr* = INITIAL
        - alors Annuler localement la transaction
        - sinon (c'est une situation indécidable) Attendre la récupération du Producteur
        - Fin-si
      - Fin-si
  - Fin-si

- Fin-cas

La Liste de Consommateurs (LC) que chaque Consommateur reçoit avec le message-demande "FIN-TR" permet une résolution de la transaction (et une libération de ressources) plus rapide lors de défaillances. En effet, un Consommateur qui souhaite se réinsérer et qui n'est pas sûr si le point de Validation de la transaction a été atteint ou non ($\acute{e}tat_{TR}$ = PREVALIDATION), doit alors consulter le Producteur ; mais si le Producteur se trouve isolé, le Consommateur en question n'est pas obligé d'attendre indéfiniment puisqu'il peut consulter les autres Consommateurs. Il y a seulement une situation indécidable où le Consommateur en question est obligé d'attendre indéfiniment la récupération du Producteur : celle où tous les Consommateurs consultés indiquent aussi $\acute{e}tat_{TR}$ = PREVALIDATION.

6 - CONTROLE DE CONCURRENCE

Jusqu'à maintenant nous avons décrit les processus nécessaires à l'exécution d'une transaction atomique sans mentionner sa concurrence possible avec d'autres transactions. Lorsqu'on considère plusieurs transactions simultanées, il s'avère nécessaire de contrôler la concurrence pour préserver la cohérence des données du système et empêcher les problèmes qui ont été présentés dans le chapitre I (partie 2). Ainsi, nous considérons à présent l'existence en chaque site du système d'un processus permanent "*Moniteur*", lequel est chargé d'activer et de contrôler les processus locaux, Producteur et Consommateurs appartenant à diverses transactions présentes dans le systèmes.

La configuration d'un site du système en termes de processus assurant la réalisation de transactions atomiques concurrentes, devient alors la suivante :

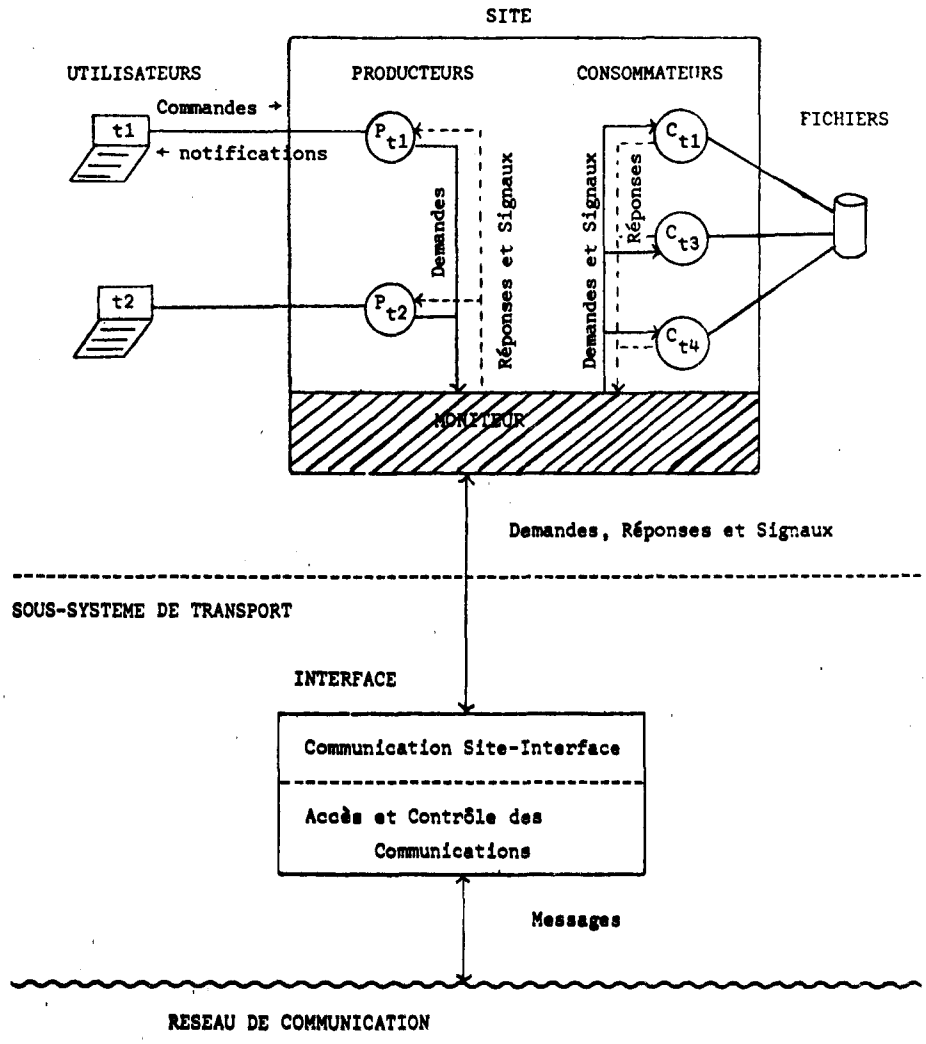


Fig. 6 : Configuration d'un site du système.

Dans le déroulement normal d'une transaction t, lorsque l'utilisateur soumet une commande, le Producteur P_t génère et délivre la demande correspondante au Moniteur du site, qui à son tour délivre la demande au Sous-système de transport, ou plus spécifiquement au module chargé de faire l'Interface entre le site et le réseau de communication. Cette Interface génère et transmet le message-demande correspondant, et en général, elle est responsable du contrôle de la transmission et de la réception de messages d'après les règles du protocole de communication.

Lorsque le module Interface d'un autre site reçoit le message-demande en question, il délivre la demande au Moniteur qui à son tour, après certaines fonctions de contrôle, délivre la demande au Consommateur concerné : C_t. Ce flux d'informations déclenché par la commande de l'utilisateur est représenté dans la figure 7 :

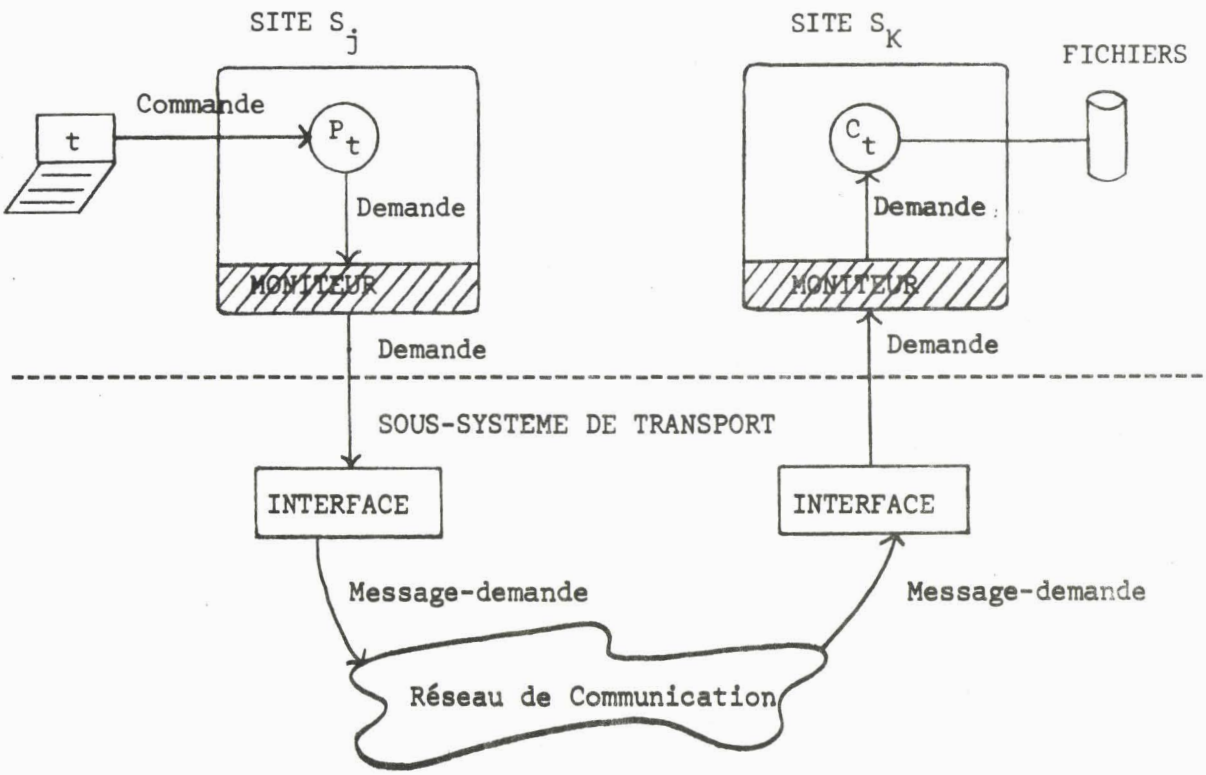


Fig. 7 : Flux d'information déclenchée par une commande d'une transaction t.

Une fois que le Consommateur C_t réalise les opérations demandées, il déclenche le flux d'informations inverse pour retourner la réponse au Producteur P_t. (Dans le cas où C_t se trouve dans le même site que P_t, il n'y a ni génération ni transmission de messages : l'Interface retourne directement au Moniteur la demande ou réponse en question).

Le Moniteur d'un site réalise trois fonctions principales que nous allons traiter par la suite : l'identification de transactions, l'ordonnancement de transactions concurrentes et la prévention d'interblocages.

6.1 - Identification de transactions

Le Moniteur d'un site doit assigner à chaque transaction soumise par les utilisateurs locaux, un identificateur *unique* dans le système. Le mécanisme d'identification utilisé doit en plus, permettre d'établir un *ordre total* entre les transactions présentes dans le système afin de résoudre les conflits d'accès aux données partagées.

Dans le chapitre I (partie 3.1), on a présenté trois méthodes générales d'identification de transactions basées respectivement sur l'utilisation d'horloges physiques, l'utilisation de compteurs et l'utilisation de séquenceurs. La méthode de séquenceurs présente l'avantage, par rapport aux autres, d'éviter les trous dans la numérotation et d'éviter la priorité d'un site en relation à un autre. En raison de cela, nous adoptons pour notre système cette méthode et plus particulièrement la méthode du "Séquenceur Circulant" de Le Lann [LELAN 79 A], laquelle consiste à faire circuler en permanence sur tous les sites un message spécial appelé "Jeton de Contrôle", contenant un séquenceur qui délivre des "tickets". Ainsi, lorsque le Moniteur d'un site reçoit le Jeton de Contrôle, il est autorisé à prélever des tickets pour les attribuer aux transactions soumises par les utilisateurs du site (via la commande DEBUT-TR) depuis le dernier passage du Jeton ; ensuite, le Moniteur envoie le Jeton au site suivant, ce qui suppose la définition d'un anneau virtual reliant tous les sites du système. Prélever un ticket consiste à assigner sa valeur à une des nouvelles transactions et à actua-

liser le ticket en l'incrémentant. Le Moniteur d'un site doit posséder une file de transactions en attente d'identification (on stockera l'identification de l'utilisateur qui veut initier une nouvelle transaction).

L'identification de transactions par la méthode du "Séquenceur circulant" permet d'identifier les transactions de manière unique, croissante et en ordre d'initialisation ; l'identificateur d'une transaction accompagnera tous les messages qui la concernent (on doit supposer néanmoins que le système utilise des techniques de reconfiguration de l'anneau et de récupération du Jeton lors de défaillances).

6.2 - Ordonnancement de transactions concurrentes

Plusieurs messages-demandes appartenant à diverses transactions concurrentes peuvent arriver à un même site. Afin de préserver la cohérence des données, le Moniteur du site doit ordonner ces demandes avant de les délivrer aux Consommateurs concernés, chargés de réaliser les accès aux données locales. Le principe d'ordonnancement que nous utiliserons est de permettre l'exécution simultanée des demandes qui ne présentent aucun conflit d'accès et de mettre les autres en attente de la disparition du conflit.

Plus précisément, lorsque le Sous-système de transport délivre une demande au Moniteur d'un site, le Moniteur décide si la demande présente ou non un conflit d'accès d'après les critères suivants :

- *Conflit de sérialisation interne* : étant donné que les demandes d'une même transaction doivent être exécutées séquentiellement au niveau d'un site, une demande de transaction t présente un conflit s'il existe une autre demande de t en exécution ou en attente.

- *Conflit de concurrence* : une demande OUVRIR d'une transaction t présente un conflit si elle requiert de verrouiller un fichier f en mode v, et f est déjà verrouillé par une au-

tre transaction en mode incompatible avec v. Les modes incompatibles pour v = S (partagé) ou X (exclusif) sont indiqués dans le tableau suivant :

	S (partagé)	X (exclusif)
S		*
X	*	*

* indique incompatibilité

Fig. 8 : Tableau d'incompatibilités pour les verrous S et X.

Si la demande ne présente aucun conflit, alors elle est délivrée au Consommateur concerné par le Moniteur. Dans le cas contraire, elle sera mise dans une file d'attente pour la transaction correspondante où les demandes sont ordonnées par ordre d'arrivée.

Chaque fois qu'un Consommateur termine l'exécution d'une demande, le Moniteur doit examiner les files d'attentes des transactions pour délivrer les demandes pour lesquelles le conflit est disparu ; dans cet examen le Moniteur donne la priorité aux transactions les plus anciennes.

En termes de protection, il est convenable d'assigner toute la gestion de verrous au Moniteur : dans cette voie, en plus des files d'attente des transactions, le Moniteur doit gérer une table de verrous pour chaque transaction active dans le site ; dans ces tables le Moniteur consigne les verrous octroyés aux transactions avant de délivrer les demandes OUVRIR aux Consommateurs concernés. A la fin d'une transaction, le Moniteur libérera tous les verrous possédés par la transaction.

6.3 - Prévention d'interblocages

Les mécanismes d'ordonnancement de transactions qui viennent d'être présentés ne suffisent pas pour éviter une situation typique d'interblocage, dans laquelle une transaction t1 se trouve bloquée en

attente d'un fichier f1 qui a été verrouillé par une autre transaction t2, et en même temps, t2 se trouve bloquée en attente d'un fichier f2 qui a été verrouillé par t1 (les fichiers f1 et f2 peuvent résider dans un même site ou dans différents sites du système). Aucune des deux transactions ne peut poursuivre son traitement, et ceci durant un temps indéfini, sauf si les Moniteurs des sites concernés interviennent pour détecter et résoudre la situation, ou alternativement, pour prévenir qu'une telle situation arrive.

Pour traiter les situations d'interblocage, on peut alors utiliser les techniques de Détection ou de Prévention qui ont été présentées dans le chapitre I (partie 4.3). Nous adoptons ici la technique de Prévention de "Wound-Wait" (Blesser ou Attendre) de Rosenkrantz [ROSEN 78] ; cette technique peut provoquer des annulations non nécessaires de transactions, mais elle est beaucoup plus simple et moins coûteuse que les techniques de Détection, lesquelles requièrent la gestion d'un graphe global d'attentes.

L'application de la technique de "Wound-Wait" implique de donner la priorité à la transaction la plus ancienne, en cas de conflit de concurrence entre deux transactions. Ainsi, si une nouvelle demande d'une transaction t1 présente un conflit de concurrence avec une autre transaction t2 (i.e. le verrou demandé par t1 est incompatible avec celui possédé par t2), le Moniteur du site du conflit décidera de :

- Annuler t2 si t1 est plus ancienne que t2 (d'après l'examen des identificateurs) et si t2 n'a pas encore atteint son point de Validation. Dans ce cas, la demande de t1 est délivrée au Consommateur concerné.

- Dans le cas contraire, placer la demande de t1 dans la file d'attente correspondante.

BUS LILLE

Une autre technique possible de Prévention est celle du "Time-out" utilisée par Lampson et Sturgis [LAMPS 77], laquelle consiste à annuler une transaction si, après un délai maximum de temps, elle maintient verrouillé un fichier qui est requis par une autre transaction. Cette technique peut provoquer aussi des annulations non nécessaires (c'est-à-dire sans qu'il y ait réellement une situation d'interblocage), mais en plus, elle peut entraîner des problèmes de Famine : une transaction très longue peut subir un nombre indéfini d'annulations. La technique du "Wound-Wait" semble alors préférable puisque ce problème de Famine n'existe pas : en effet, au bout d'un temps fini toute transaction devient la plus prio-titaire (la plus ancienne) et elle est sûre de ne plus subir d'annulations.

7 - RECAPITULATION DU MODELE DE TRANSACTIONS ATOMIQUES

7.1 - Résumé de caractéristiques

Le Modèle de Transactions Atomiques que nous venons de présenter peut être caractérisé en termes des critères qualitatifs d'évaluation définis par Wills [WILLS 79, Chapitre 3] de la manière suivante :

- Type de Répartition des données : par partition (et composition de fichiers).

- Réalisation d'une transaction : de façon atomique à l'aide de Listes d'Intentions.

- Récupération aux défaillances : par achèvement ou annulation de chaque transaction interrompue, selon que son point de Validation a été atteint ou non.

- Allocation de ressources-données : de manière dynamique.

- Contrôle de concurrence : partiellement réparti, avec un processus Producteur par transaction chargé de la coordination de ses processus Consommateurs.

- Outils de Synchronisation : les tickets, générés par un Séquenceur Circulant, pour l'identification de transactions, et les files d'attente et les tables de verrous pour l'ordonnancement de transactions.

- Résolution de conflits : par attente.

- Prévention des situations d'Interblocage et de Famine : par annulation d'une transaction lorsqu'elle possède des données verrouillées qu'une autre transaction, plus ancienne, vient de demander.

- Granularité des verrous : le fichier de données.

- Modes des verrous : S (partagé) et X (exclusif).

- Durée des verrous : toute une transaction.

- Parallélisme : inter-transactionnel (divers transactions sans conflit peuvent s'exécuter simultanément), et éventuellement intra-transactionnel (divers commandes d'une même transaction peuvent s'exécuter simultanément si elles concernent différents sites).

7.2 - Préservation de la cohérence

Le Modèle de Transactions Atomiques est avant tout un modèle de contrôle de cohérence : il doit assurer au minimum la préservation de la cohérence des données du système.

Il faut montrer tout d'abord que l'exécution de toute transaction entraîne la réalisation de tous ses effets ou d'aucun ; en effet selon la définition de transaction que nous avons adoptée, chaque transaction indépendante est une séquence d'opérations qui, exécutée totalement, fait passer le système d'un état cohérent à un autre état cohérent. Il faut alors *préserver l'atomicité* de chaque transaction lors de défaillances ; dans le Modèle présenté cela est possible grâce aux Listes d'Intentions :

- Si une défaillance affecte une transaction avant d'atteindre son Point de Validation, les mécanismes de récupération annuleront tous ses effets par la simple élimination des Listes d'Intentions : en fait aucun fichier de données n'a été modifié par la transaction jusqu'à ce moment.

- Si une défaillance affecte une transaction après qu'elle ait atteint son Point de Validation, les mécanismes de récupération compléteront tous ses effets grâce aux Listes d'Inten-

tions qui contiennent à ce moment toute l'information nécessaire pour réaliser les modifications de fichiers prévues par la transaction.

L'atomicité de chaque transaction *indépendante* est ainsi préservée lors de défaillances (et en conséquence la cohérence que la transaction assure par elle-même). Il faut montrer maintenant que la *concurrence* de transactions ne menace pas la cohérence des données du système ; en fait, les mécanismes d'ordonnancement (et plus spécifiquement la gestion de verrous à la charge du Moniteur de chaque site) obligent à chaque transaction à être *Bien-Formée* et à *Deux-Phases* (concepts définis dans le chapitre I, partie 3.2) :

- Chaque transaction est Bien-Formée parce que chaque fichier qu'elle utilise est verrouillé une seule fois (lors de la commande OPEN) et libéré à la fin de la transaction.

- Chaque transaction est à Deux-Phases parce que tous les verrous qu'elle entraîne sont établis pendant son étape de Synchronisation, et ils sont tous retirés pendant son étape de Validation.

En plus, le traitement de conflits d'accès réalisé par le Moniteur de chaque site assure, que l'ordonnancement des transactions soit *légal* : un fichier verrouillé par une transaction en mode S partagé peut être accédé par d'autres transactions, mais un fichier verrouillé en mode X exclusif ne peut être accéder par aucune autre transaction. Il résulte alors des ordonnancement légaux des transactions Bien-Formées et à Deux-Phases, ce qui assure la préservation de la cohérence des données du système lors de concurrence de transactions, d'après les résultats de Eswaran et Gray [ESWAR 76]. Dans ces ordonnancements on ne considère pas les transactions annulées par les mécanismes de récupération aux défaillances ou par prévention des interblocages : en effet, ces transactions par leur propriété atomique ne changent pas les données et en conséquence elles n'ont aucun effet sur la cohérence (pour une transaction annulée plusieurs fois, ce qui compte est la transaction finale non-annulée qui est Bien-Formée et à Deux-Phases).

Finalement, le Modèle de Transactions Atomiques évite le problème de l'*effet Domino* (présenté dans le chapitre II, partie 4) : Toutes les annulations d'une transaction provoquées par le système lors de défaillances, ou par l'utilisateur lors de la commande ANNULER, ne peuvent être déclenchées qu'avant la fin de la transaction ; comme la transaction n'a pas atteint son Point de Validation à ce moment, les fichiers qu'elle a accédé restent sans modification et en conséquence sa récupération (i.e. son retour à l'état initial) n'affecte pas les autres transactions présentes dans le système. Ainsi, une avalanche de retours de transactions ne peut pas se produire.

7.3 - Discussion

En plus de préserver la cohérence des données du système et de permettre un certain degré de parallélisme entre transactions concurrentes, le Modèle de Transactions Atomiques présente quelques avantages par rapport à d'autres approches qui utilisent aussi des Listes d'Intentions (comme celles de Lampson et Sturgis [LAMPS 77], Israel et Mitchell [ISRAE 78], et Paxton [PAXTO 79]) :

- Le modèle ne requiert aucune caractéristique spéciale sur le matériel de stockage : le stockage "*stable*" requis par quelques approches [LAMPS 77 ; ISRAE 78] pour garder les Listes d'Intentions, n'est pas nécessaire du fait que les Listes d'Intentions sont formées au fur et à mesure qu'une transaction se déroule et non à la fin (le stockage stable est celui qui vérifie une propriété atomique lorsqu'une défaillance survient pendant une écriture en cours : ou bien le segment correspondant a été correctement modifié, ou bien il reste sans modification).

- Les Listes d'Intentions entraînent très peu de surcharge en stockage, puisqu'elles consignent uniquement les nouvelles adresses des segments modifiés. Dans l'approche de Paxton [PAXTO 79] les Listes d'Intentions contiennent des copies

des Cartes de segments des fichiers accédés, ce qui entraîne un stockage important lorsque les fichiers sont grands et que les Cartes contiennent diverses informations en plus des adresses (par exemple les listes de droits d'accès).

- La séparation entre les fonctions de coordination et d'accès aux fichiers simplifie les algorithmes correspondants à la réalisation d'une transaction. En plus, l'assignation de la fonction de coordination d'une transaction à un processus Producteur localisé sur le même site que l'utilisateur permet à ce dernier d'être libre de doute sur le sort de sa transaction lors de défaillances. Dans les approches mentionnées [LAMPS 77 ; ISRAE 78 ; PAXTO 79] la fonction de coordination est assignée au premier Consommateur référencé ce qui entraîne un double trafic de messages entre le site de l'utilisateur et le site du Consommateur-coordinateur, et entre ce dernier et les sites des autres Consommateurs.

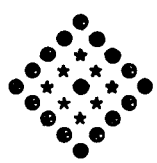
Malgré les avantages précédents, le Modèle de Transactions Atomiques pénalise l'utilisateur interactif dans les aspects suivants :

- *Monopolisation longue des données* : Les verrous qu'une transaction établit sur les fichiers sont maintenus jusqu'à sa fin, empêchant l'utilisation de ces fichiers par d'autres transactions concurrentes (sauf s'il s'agit de verrous en mode partagé). Cette monopolisation de données peut entraîner un degré de concurrence faible entre transactions, et de longues attentes pour accéder aux données, ce qui affecterait le temps de réponse à l'utilisateur.

- *Perte totale des opérations lors de défaillances* : L'annulation d'une transaction dans le cas de récupération à une défaillance (et aussi de prévention d'interblocages) signifie pour l'utilisateur la perte de toutes les opérations réalisées, puisque aucune n'est rendue effective avant la fin de la transaction.

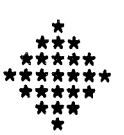
- *Impossibilité de demander l'annulation partielle d'une transaction* : L'utilisateur est soumis aux annulations non désirées de sa transaction, lors de défaillances et par prévention d'interblocages et ne possède en contrepartie qu'une seule option de contrôle : cette option est la demande d'annulation de toutes les opérations réalisées ; mais il n'a pas la possibilité de demander le retour à un état antérieur de sa transaction par annulation de seulement une partie de ces opérations.

Dans le chapitre suivant, nous étendrons le Modèle de Transactions Atomiques afin de résoudre les problèmes cités.



CHAPITRE IV

MODELE DE CONTROLE DE COHERENCE ADAPTE A L'UTILISATEUR INTERACTIF D'UN SYSTEME REPARTI DE FICHIERS : TRANSACTIONS EMBOITEES A VERROUILLAGE SELECTIF



Dans ce chapitre nous allons étendre le Modèle de Transactions Atomiques présenté dans le chapitre III, afin de mieux répondre aux exigences de l'utilisateur interactif en environnement réparti, principalement dans les trois aspects suivants :

- Diminution du temps de réponse, par un verrouillage plus sélectif qui permette un degré plus élevé de concurrence entre transactions et qui réduise la monopolisation exclusive des données par une seule transaction.

- Possibilité de demander l'annulation partielle d'une transaction, ce qui implique la remise de la transaction dans l'état antérieur que l'utilisateur souhaite et non uniquement dans l'état initial.

- Libération graduelle de ressources-données pendant la transaction, ce qui implique la réalisation effective des opérations pendant la transaction et non uniquement à sa fin ; l'utilisateur serait ainsi épargné de perdre toutes ses opérations lors de défaillances et les autres transactions pourraient accéder plus tôt aux ressources libérées, ce qui diminuerait aussi les temps de réponse.

Pour obtenir une diminution du temps de réponse, nous allons utiliser d'abord les "Verrous Hiérarchiques" de Gray et al. [GRAY 75] qui impliquent cinq modes de verrous appliqués à des granules de différentes tailles et qui permettent diverses degrés de concurrence.

Pour permettre les annulations partielles et la libération graduelle des données pendant chaque transaction, nous allons utiliser une nouvelle structure de transaction qu'on peut appeler "les Transactions Emboîtées" ; les transactions ne seront plus atomiques, ce qui rend

plus complexe la préservation de la cohérence des données du système. En fait, les *transactions emboîtées* avec lesquelles nous travaillerons ne devraient plus être nommées "transactions" dans le sens défini par Eswaran et al. [ESWAR 76], puisqu'elles ne seront plus des unités de cohérence : au lieu de cela, une transaction emboîtée contiendra plusieurs transactions internes qui vont délimiter *plusieurs unités de cohérence.*

(Il semblerait plus adéquat d'utiliser le mot "cohorte" pour nommer une transaction emboîtée totale - du terme anglais "cohort" utilisé par Gray - en réservant le mot "transaction" pour chacune des unités de cohérence qui la composent ; pourtant, nous utiliserons les mots "transaction emboîtée" pour nommer la transaction totale, ce qui nous semble plus natural).

Chaque transaction emboîtée peut être vue comme une séquence t_1; t_2;...; t_m de transactions atomiques, Bien-Formées et à Deux-Phases. Entre deux transactions successives t_i et t_{i+1}, composantes de la transaction emboîtée t, il existera une indépendance totale par rapport à l'utilisation des ressources qu'elles déclarent localement : ces ressources sont verrouillées au fur et à mesure où elles sont déclarées et sont libérées à la fin de la transaction composante qui les utilise (i.e. c'est le schéma conventionnel de transaction atomique). Par contre, les ressources déclarées au niveau global seront maintenues verrouillées d'une transaction composante, t_i, à la suivante, t_{i+1}, pour être libérées seulement à la fin de toute la transaction emboîtée : cela évitera à l'utilisateur des attentes non nécessaires pour le verrouillage de ces ressources.

Pour prévenir les interblocages et l'effet Domino on permettra à l'utilisateur les retours en arrière à l'intérieur de chaque transaction composante t_i (en utilisant le mécanisme de Listes d'Intentions), mais une fois finie t_i, elle sera considérée définitive et irréversible : tous les ressources utilisées (locales et globales) seront jugées en état cohérent et l'utilisateur ne pourra plus demander un retour en arrière de ce point (cela suppose une programmation judicieuse de la part de l'utilisateur mais dont il peut tirer profit).

La mise en oeuvre du modèle résultante de "Transactions Emboîtées à Verrouillage Sélectif" sera spécifiée dans la partie 3 de ce chapitre, en termes des structures d'information de contrôle nécessaires et des algorithmes des processus Producteur, Consommateur et Moniteur pour le traitement normal des transactions, et pour le traitement des situations d'exception.

1 - CONCEPT DE VERROUILLAGE SELECTIF

Pour améliorer le degré de concurrence qu'on peut obtenir avec le Modèle de Transactions Atomiques, nous voulons à présent utiliser une granularité de verrous plus fine que celle du fichier ; ainsi, on peut appliquer les verrous au niveau du segment de fichier pour permettre l'accès simultané à un même fichier par plusieurs transactions qui utilisent différents segments. Mais cette granularité plus fine entraîne aussi une surcharge plus grande en gestion de verrous qui peut répercuter négativement sur le temps de réponse, surtout pour les transactions qui accèdent beaucoup d'unités d'informations.

La sélection de la granularité des verrous constitue alors un compromis à faire entre concurrence et surcharge. Les *"Verrous Hiérarchiques"* définis par Gray et al [GRAY 75] constituent une solution à ce compromis : dans cette approche, qui comprend des granularités de différentes tailles, les ressources susceptibles d'être verrouillées sont organisées dans une hiérarchie ; l'obtention d'un verrou "S" partagé, ou "X" exclusif sur une ressource R doit permettre l'accès à R et implicitement à tous ses descendants (en fait, à tous ses composants), mais pour cela on doit verrouiller auparavant tous les ascendants de R en mode "Intention" (Intention Partagée "IS", ou Intention Exclusive "IX") afin de prévenir que d'autres transactions n'appliquent des verrous incompatibles sur ces ascendants.

Dans notre système la hiérarchie des ressources-données comprend uniquement deux niveaux : le fichier et le segment. L'application des Verrous Hiérarchiques à ces deux niveaux donne lieu à une spécification nouvelle de modes de verrous et d'opérations qui sera présentée par la suite.

1.1 - Verrous au niveau de fichier et au niveau de segment [FRANK 81]

Au niveau de fichier, nous allons considérer maintenant cinq modes possibles de verrous que l'utilisateur pourra spécifier dans la commande OUVRIR : les deux modes traditionnels S et X et trois nouveaux "intention", avec les significations suivantes :

- *Le verrou* S *"partagé"* établit un verrou S implicite sur chaque segment du fichier.

- *Le verrou* X *"exclusif"* établit un verrou S implicite sur chaque segment du fichier.

- *Le verrou* IS *"intention-partagée"* n'établit aucun verrou implicite sur les segments, mais permet à l'utilisateur de demander un verrou S explicite sur chaque segment qu'il veut lire (en conséquence, chaque lecture entraîne une attente possible pour obtenir le verrou explicite).

- *Le verrou* IX *"intention-exclusive"* permet à l'utilisateur de demander un verrou S explicite sur chaque segment qu'il veut lire ou un verrou X explicite sur chaque segment qu'il veut modifier (lecture ou écriture avec attente possible).

- *Le verrou* SIX *"partagé-intention-exclusive"* (combination des verrous S et IX) établit un verrou S implicite sur chaque segment du fichier et, en plus, permet à l'utilisateur de demander un verrou X explicite sur chaque segment qu'il veut modifier (lecture sans attente et écriture avec attente possible).

Les incompatibilités entre ces cinq modes de verrous (i.e. les verrous qui ne peuvent pas être octroyés simultanément) sont indiquées dans le Tableau de la figure 1 [GRAY 75] :

	intention-partagée	intention-exclusive	partagé	partagé-intention-exclusive	exclusif
	IS	IX	S	SIX	X
IS					*
IX			*	*	*
S		*		*	*
SIX		*	*	*	*
X	*	*	*	*	*

* indique incompatibilité

Fig. 1 : Tableau d'Incompatibilités entre les verrous IS, IX, S, SIX, X.

D'une part, les verrous "intention" IS et IX favorisent la concurrence entre transactions puisque la monopolisation des données ne se fait plus au niveau du fichier mais au niveau du segment. D'autre part, les verrous S et X appliqués sur un fichier évitent la surcharge en gestion de verrous lorsque l'utilisateur souhaite lire ou modifier presque tous les segments du fichier. Le verrou SIX, qui combine les propriétés des verrous S et IX, devient utile lorsque l'utilisateur veut lire presque tous les segments d'un fichier et n'en modifier que quelques-uns.

Les verrous hiérarchiques peuvent être ordonnés partiellement d'après les privilèges qu'ils impliquent (voir figure 2) ; les verrous avec moins de privilèges (i.e. IS) laissent plus de possibilités de concurrence.

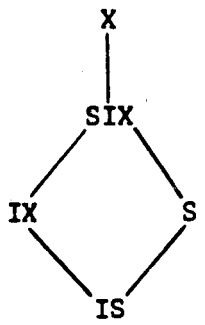


Fig. 2 : Ordre partiel des verrous hiérarchiques selon leurs privilèges.

-*-*-*-*-*-*-*-*-
-*-*-*-*-*-*-*-
-*-*-*-*-*-*-*-*-

Au niveau de segment, l'utilisateur peut demander comme avant les opérations LIRE et ECRIRE. L'opération LIRE requiert au préalable un verrou S, implicite ou explicite, sur le segment (ou un verrou supérieur à S), et l'opération ECRIRE requiert au préalable un verrou X, implicite ou explicite, sur le segment.

Si au moment d'une opération LIRE ou ECRIRE, la transaction possède un verrou intention pour le fichier mais aucun verrou pour le segment concerné, le système établira le verrou explicite nécessaire (S ou X), en supposant qu'il n'y ait pas de conflit de concurrence. Il nous semble utile de définir une troisième opération "LIEC" (Lire avec Intention d'Ecrire) qui réalise la lecture d'un segment mais qui requiert un verrou X en vue d'une modification postérieure du même segment ; en effet, un verrou S sur le segment empêcherait sa modification postérieure.

La table de la figure 3 résume les opérations permises à une transaction qui a obtenu un verrou v sur un fichier f (à travers la commande OUVRIR (f, v) pour v = IS, IS, SIX, S ou X) et les opérations permises aux autres transactions concurrentes, par rapport au fichier f :

Opérations sur f / Verrou sur f	Opérations permises à la transaction	Type d'OUVRIR permis aux autres	Application
IS	LIRE (f, i, b) avec attente	OUVRIR (f, {IS, IX, S, SIX})	Consultation de quelques segments du fichier
IX	{LIRE, LIEC, ECRIRE} (f, i, b) avec attente	OUVRIR (f, {IS, IX})	Modification de quelques segments du fichier
S	LIRE (f, i, b) sans attendre	OUVRIR (f, {IS, S})	Consultation de presque tout le fichier
SIX	LIRE (f, i, b) sans attendre {LIEC, ECRIRE} (f, i, b) avec attente	OUVRIR (f, IS)	Consultation de presque tout le fichier, et modification de quelques segments
X	{LIRE, LIEC, ECRIRE} (f, i, b) sans attente	(aucun)	Modification de presque tout le fichier

Fig. 3 : Opérations permises par les verrous hiérarchiques.

1.2 - Contrôle de concurrence pour un verrouillage sélectif

L'incorporation des verrous hiérarchiques au Modèle de Transactions Atomiques afin d'obtenir un verrouillage sélectif, nécessite seulement la modification des mécanismes d'ordonnancement de transactions concurrentes, énoncés dans le chapitre III (partie 6.2).

Pour cela, lorsqu'une demande appartenant à une transaction t arrive à un site, le Moniteur du site décidera si la demande présente ou non un conflit d'accès d'après les nouveaux critères suivants :

- *Conflit de sérialisation interne* : comme avant, la demande qui arrive présente un conflit s'il existe une autre demande de la même transaction t en exécution ou en attente.

- *Conflit de concurrence au niveau de fichier* : une demande OUVRIR présente un conflit si le fichier concerné se trouve déjà verrouillé par une autre transaction en mode incompatible avec le verrou v requis (pour v = IS, IX, S, SIX ou X).

- *Conflit de concurrence au niveau de segment* : une demande LIRE, LIEC ou ECRIRE présente un conflit si le segment de fichier concerné se trouve déjà verrouillé (implicitement ou explicitement) par une autre transaction en mode incompatible avec le verrou v requis (v = S pour LIRE et v = X pour LIEC et ECRIRE).

Si la demande arrivée ne présente aucun conflit, alors le Moniteur la délivre au Consommateur de la transaction t, mais dans le cas contraire, elle est mise dans la file d'attente de t (comme avant, il peut avoir des annulations de transactions pour prévenir les interblocages).

8 - CONCEPT DE TRANSACTIONS EMBOITEES

Le verrouillage sélectif, qui résulte de l'utilisation des Verrous Hiérarchiques, améliore le degré réel de concurrence entre transactions et entraîne ainsi une réduction du temps moyen de réponse à une transaction, mais il n'est pas suffisant pour satisfaire les autres exigences de l'utilisateur interactif telles la possibilité d'annulations partielles de sa transaction et la libération graduelle de données pendant sa transaction.

Pour satisfaire ces exigences, nous allons considérer maintenant des *transactions emboîtées* qui peuvent contenir plusieurs niveaux de transactions internes. Comme exemple, considérons la transaction emboîtée suivante contenant un seul niveau de transaction interne :

```
P1 : - DEBUT-TR
     |    OUVRIR (f1, IX)
     |    mises à jour sur f1
P2 : |  - DEBUT-INT
     |  |    OUVRIR (f2, X)
     |  |    mises à jour sur f1, f2
P3 : |  - FIN-INT
     |    mises à jour sur f1
P4 : - FIN-TR
```

Chaque transaction interne sert d'abord à délimiter la *durée d'utilisation* des fichiers qui sont "ouverts" dans elle (dans l'exemple le fichier f_2 est utilisé seulement dans la transaction interne P2-P3, tandis que le fichier f_1 est utilisé dans toute la transaction englobante P1-P4). En fait, on utilise ici pour les transactions, une structure en blocs de type Algol. Chaque début de transaction interne va marquer, en plus, un *point de sauvegarde* défini par l'utilisateur.

Nous allons analyser maintenant comment résoudre les problèmes d'annulations partielles et de libération graduelle des données, avec cette nouvelle structure de transaction.

2.1 - Annulations partielles pendant une transaction

L'annulation d'une transaction, déclenchée par une demande de l'utilisateur ou par les mécanismes de récupération aux défaillances, doit remettre la transaction dans l'état qu'elle avait au début de la transaction interne en cours et non pas au début de toute la transaction. Pour permettre cette annulation partielle, les points de sauvegarde associés aux transactions internes doivent être stockés et gérés de forme naturelle en pile.

Dans cette voie, la réalisation d'une transaction emboîtée entraîne la gestion en pile de la Liste d'Intentions de chacun de ses Consommateurs : le début d'une transaction interne se traduira en l'initialisation d'une nouvelle zone dans la "Pile d'Intentions" de chaque Consommateur ; dans cette zone le Consommateur devra enregistrer les intentions de modification des fichiers locaux, apportées par la transaction interne. Dans le cas d'annulation il suffira d'éliminer les intentions de la dernière zone de la pile de chaque Consommateur, pour retrouver l'état du début de la transaction interne en cours (qui constitue le dernier point de sauvegarde défini par l'utilisateur).

La fin d'une transaction interne doit entraîner la suppression du point de sauvegarde associé à cette transaction. Pour cela, on peut supprimer la dernière zone de la pile de chaque Consommateur en intégrant en même temps, les intentions apportées par cette transaction interne à celles de la transaction immédiatement englobante. Dans ce cas, on reporte la réalisation effective de toutes les intentions à la fin de la transaction la plus externe pour donner à l'utilisateur la possibilité de demander le retour de la transaction au début de n'importe quelle transaction englobante, même au début de toute la transaction (pour cela, l'utilisateur devra d'abord finir les transactions internes pour revenir à la transaction englobante qui l'intéresse, et ensuite demander l'annulation).

2.2 - Libération graduelle de données pendant une transaction

Les transactions internes d'une transaction délimitent la durée d'utilisation des fichiers accédés ; ainsi, les fichiers ouverts dans une transaction interne devraient être libérés à la fin de celle-ci puisqu'ils ne seront plus utilisés et dans ce cas, la libération de ces fichiers doit être précédée de la réalisation effective des intentions de modification qui les concernent.

Cette libération graduelle de données pendant une transaction favorise la concurrence réelle entre transactions, en réduisant la monopolisation de données et assure à l'utilisateur la réalisation effective d'une partie de ses opérations lors de défaillances. Mais par ailleurs, cette libération graduelle va à l'encontre de l'approche d'annulations partielles que nous venons de proposer : en effet, si des données sont libérées à un point P d'une transaction, l'utilisateur ne pourra plus demander par la suite une annulation entraînant un retour de la transaction à un point de sauvegarde antérieur à P, puisque ce retour pourrait déclencher une avalanche de retours dans les autres transactions concurrentes qui ont pu accéder aux données libérées à P (c'est l'effet Domino présenté dans le chapitre II, partie 4).

Alors, il faut faire un compromis pour permettre simultanément les annulations partielles et la libération graduelle de données dans une transaction emboîtée. Dans cette voie nous allons considérer maintenant deux types de commandes dont l'utilisateur dispose pour finir une transaction interne :

- FIN 1 : implique la suppression du point de sauvegarde associé à la transaction interne par l'intégration de ses intentions à celles de la transaction immédiatement englobante.

- FIN 2 : définit un point cohérent de toute la transaction et implique la suppression du point de sauvegarde associé à la transaction interne par la réalisation effective des intentions

sur les fichiers ouverts dans cette transaction interne (i.e. fichiers "locaux"). L'utilisateur renonce ainsi à l'option de demander le retour à un point de sauvegarde antérieur à ce point cohérent et en conséquence, les intentions sur les fichiers ouverts dans les transactions englobantes (i.e. fichiers "globaux") peuvent aussi être réalisées. La commande FIN 2 implique, en plus, la libération de *toutes* les ressources utilisées par la transaction interne, c'est-à-dire : a) la totalité des fichiers "locaux", b) ceux segments des fichiers "globaux" qui ont été utilisés proprement par la transaction interne. Ces fichiers "globaux" ne sont pas libérés totalement puisqu'ils continuent à être utilisés dans la transaction.

L'annulation partielle d'une transaction, demandée par l'utilisateur ou déclenchée par la récupération à une défaillance, entraînera son retour à l'état défini par :

- le début de la transaction interne en cours, si pendant cette transaction interne il n'y a pas eu de points cohérents (établis par des commandes de type FIN 2) ;

- le dernier point cohérent établi dans la transaction interne en cours, dans le cas contraire.

Pour terminer une transaction interne, l'utilisateur devrait choisir la commande FIN 2 s'il considère qu'à ce point tous les fichiers manipulés par sa transaction sont en état cohérent et qu'il n'aura plus besoin de demander des retours en arrière de ce point ; dans le cas contraire, il doit choisir la commande FIN 1 (avec ce critère, la fin de la transaction totale correspond à une commande FIN 2).

Il s'en suit qu'une transaction emboîtée n'est plus nécessairement une seule unité de cohérence, mais qu'elle comprend plusieurs unités de cohérence délimitées par les commandes FIN 2. Pour l'utilisateur

ces unités de cohérence sont totalement indépendantes, de telle façon que théoriquement le système pourrait libérer toutes les ressources utilisées par la transaction à la fin de chaque unité ; alors, c'est seulement pour épargner à l'utilisateur des attentes inutiles que le système garde réservés à la fin d'une unité de cohérence, les segments de fichiers "globaux" qui seront utilisés dans l'unité suivante.

2.3. - Construction de transactions emboîtées à partir de transactions atomiques

Une transaction emboîtée est *Bien-Formée* parce que tout segment de fichier accédé est verrouillé auparavant (soit par un verrou implicite lors de la commande OUVRIR, soit par un verrou explicite lors de la première commande LIRE, LIEC ou ECRIRE le concernant) et il est libéré avant la fin de la transaction. Par contre, les transactions emboîtées ne sont plus à *Deux-Phases* : en effet, la libération graduelle de données est réalisée parallèlement au verrouillage progressif de nouvelles données. Dans ce cas se pose le problème de la préservation de cohérence des données du système avec ce type de transactions.

Nous voulons illustrer avec un exemple la manière comme on peut construire, sans ménacer la cohérence des données, une transaction emboîtée à partir d'une séquence de transactions atomiques, Bien-Formées et à Deux-Phases.

Supposons d'abord que l'utilisateur veut réaliser 2 transactions successives t_a et t_b de la forme suivante:

transaction t_a:

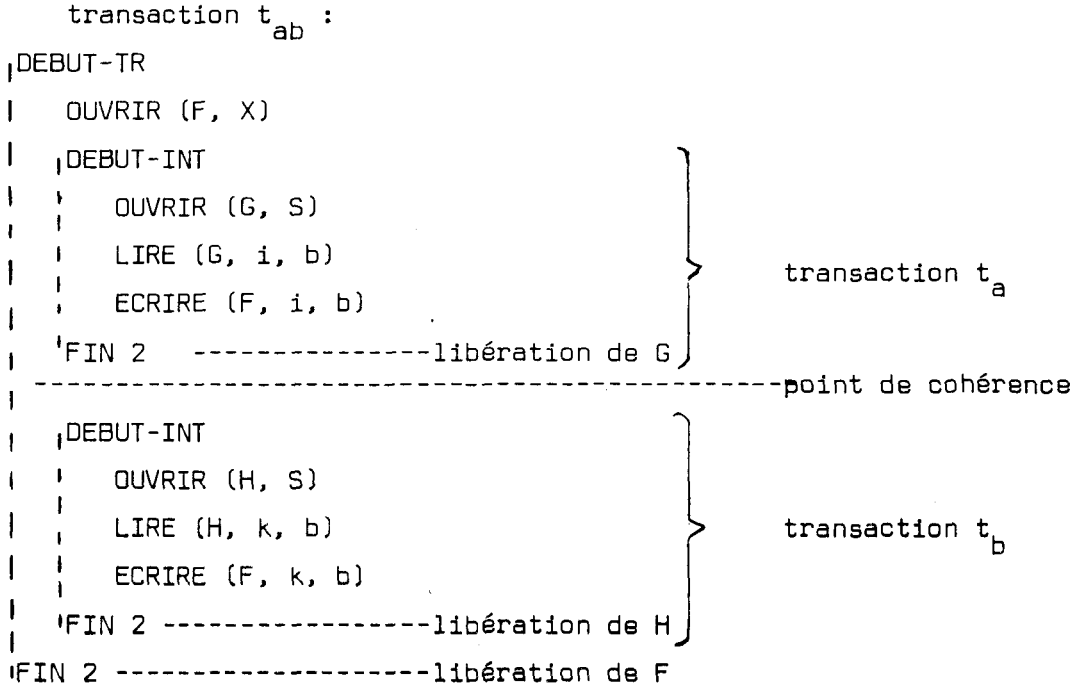
```
DEBUT-TR
   OUVRIR (F, X)
   OUVRIR (G, S)
   LIRE (G, i, b)
   ECRIRE (F, i, b)
FIN 2
```

transaction t_b:

```
DEBUT-TR
   OUVRIR (F, X)
   OUVRIR (H, S)
   LIRE (H, k, b)
   ECRIRE (F, k, b)
FIN 2
```

Chacune de ces transactions, t_a et t_b, est Bien-Formée, à Deux-Phases (i.e. la libération des fichiers est faite avec la commande FIN 2), et réalisée

par le système de façon atomique (en utilisant des listes d'intentions): ces propriétés assurent la préservation de la cohérence des données (d'après les résultats présentés dans le chapitre III - partie 3.2.2). Mais pour l'utilisateur il existe un inconvénient: une fois libéré le fichier F, à la fin de la transaction t_a, on doit attendre à nouveau l'obtention d'un verrou sur F au début de la transaction t_b. Une transaction emboîtée t_{ab}, regroupant t_a et t_b, va permettre d'enlever cet inconvénient *sans menacer la cohérence des données*: simplement le fichier F, considéré maintenant global, va être maintenu verrouillé entre la fin de t_a et le début de t_b:



Pour un exemple plus complet, considérons maintenant 2 transactions T1 et T2 où chacune est, comme avant, Bien-Formée et à Deux-Phases; en plus, dans chacune, T1 et T2, on va utiliser une structure de transactions internes, avec les commandes DEBUT-INT et FIN 1, pour délimiter des points de sauvegarde. Les transactions T1 et T2 sont alors de la forme suivante:

Transaction T1 :

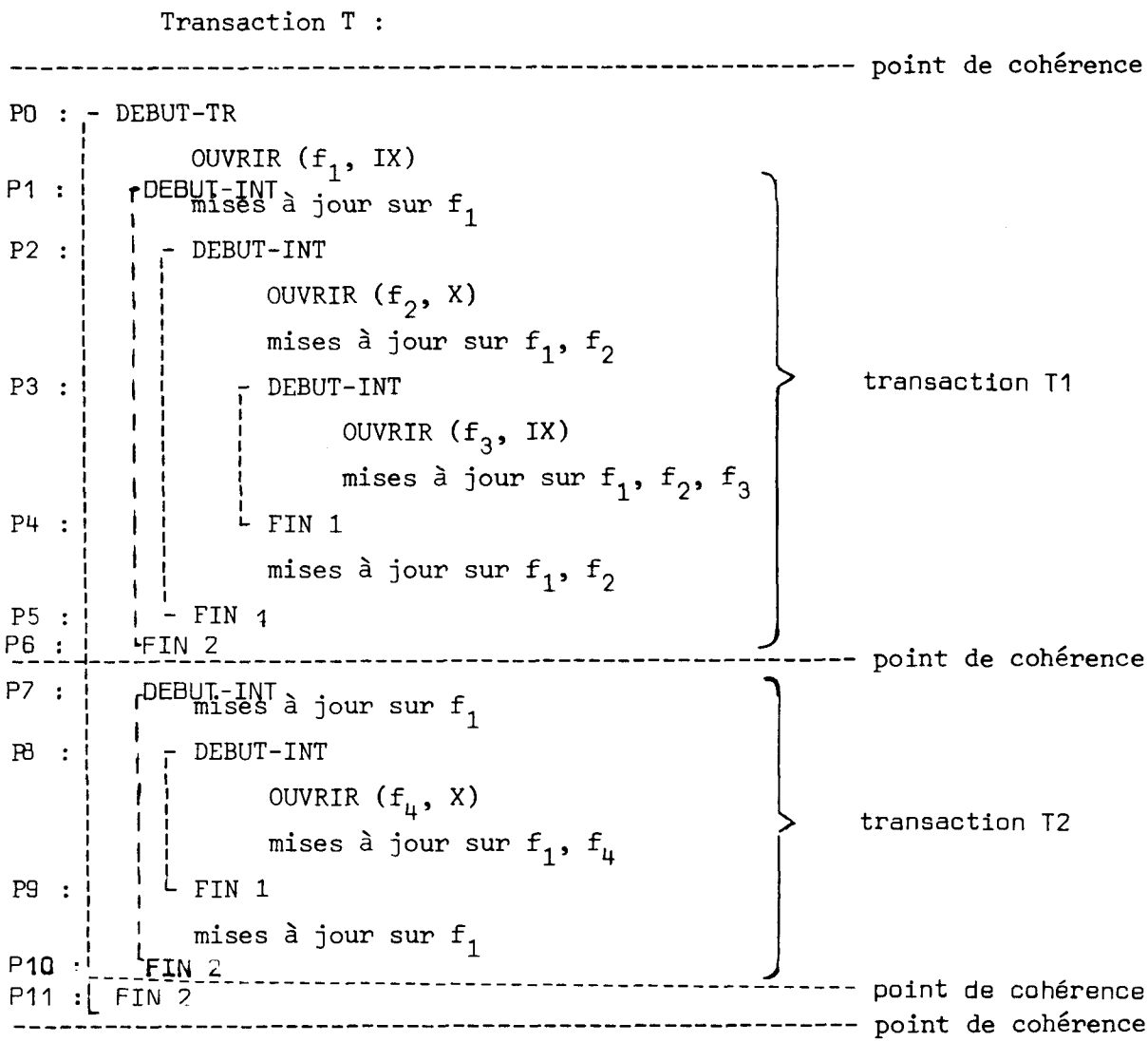
```
P1 : - DEBUT-TR
       OUVRIR (f1, IX)
       mises à jour sur f1
P2 :   - DEBUT-INT
         OUVRIR (f2, X)
         mises à jour sur f1, f2
P3 :     - DEBUT-INT
           OUVRIR (f3, IX)
           mises à jour sur f1, f2, f3
P4 :     - FIN 1
           mises à jour sur f1, f2
P5 :   - FIN 1
P6 : - FIN 2
```

Transaction T2 :

```
P7: - DEBUT-TR
      OUVRIR (f1, IX)
      mises à jour sur f1
P8 :  - DEBUT-INT
        OUVRIR (f4, X)
        mises à jour sur f1, f4
P9 :  - FIN 1
        mises à jour sur f1
P10: - FIN 2
```

Dans chacune de ces transactions, T1 et T2, l'utilisateur peut demander des annulations partielles pour revenir au point de sauvegarde le plus proche, défini par la dernière commande DEBUT-INT; ces retours sont possibles puisque les intentions de modification et la libération des fichiers ne sont effectuées qu'au moment de la commande FIN 2: chaque transaction, T1 et T2, est alors réalisée par le système de façon *atomique*, et préserve la cohérence des données.

Comme avant, on va utiliser la notion de transaction emboîtée comme un mécanisme permettant de regrouper les 2 transactions T1 et T2 tout en conservant verrouillé le fichier global f_1 pendant toute la durée des 2 transactions, ce qui est souhaité par l'utilisateur lorsqu'il doit réaliser successivement T1 et T2 (ce mécanisme ne représente aucune ménace à la préservation de la cohérence des données étant donné que chacune des transactions composantes T1 et T2 est atomique, Bien-Formée et à Deux-Phases). La transaction T qui résulte en regroupant T1 et T2 est la suivante:



Dans cette transaction emboîtée T, les commandes FIN2 signalent les points où il y a libération de ressources; ces ressources doivent se trouver en état cohérent (ce qui était vrai pour les transactions originales T1 et T2):nous supposons alors que cette responsabilité est laissée à la charge de l'utilisateur. Les commandes FIN 2 signalent alors les points de cohérence de la transaction et délimitent les "unités de cohérence" composant la transaction emboîtée T: pour l'exemple, ce sont les unités P0-P6, P7-P10, et P10-P11.

A l'intérieur de chaque unité de cohérence, l'utilisateur peut revenir en arrière au point de sauvegarde en cours (avec une commande ANNULER): ainsi par exemple pour l'unité de cohérence P0-P6:

- entre P0-P1, on peut revenir à P0;
- entre P1-P2, on peut revenir à P1;
- entre P3-P4, on peut revenir à P2;
- entre P4-P5, on peut revenir à P2 (le point de sauvegarde P3 n'existe plus);
- entre P5-P6, on peut revenir à P1 (les points de sauvegarde P2 et P3 n'existent plus).

A la fin d'une unité de cohérence tous les fichiers sont mis à jour avec les intentions de modification accumulées jusque là dans les Listes d'Intentions, et toutes les ressources utilisées localement sont libérées. Ainsi par exemple, à la fin de l'unité de cohérence P0-P6 il y aura la modification effective des fichiers f_1, f_2 et f_3 , et la libération des fichiers f_2, f_3 , ainsi que des segments de f_1 qui ont été utilisés entre P1-P6 (en général ces segments de f_1 seront différents de ceux utilisés dans les autres unités de cohérence, mais si ce n'est pas le cas, l'utilisateur doit être conscient que ces segments libérés peuvent être modifiés par d'autres utilisateurs). A P6 le fichier global f_1 est en état cohérent mais il n'est pas relaché puisqu'il continue à être utilisé dans la suite de la transaction: il sera libéré uniquement à P11.

Une transaction emboîtée constitue donc un mécanisme qui permet de regrouper plusieurs transactions atomiques successives tout en conservant réservées les ressources globales necéssaires pour toutes, ce qui est intéressant du point de vue d'efficacité: il y a pour l'utilisateur une diminution du temps d'attente et moins besoin de faire de relectures. Pour les autres utilisateurs il peut avoir un allongement du temps d'attente, mais ceci peut dans certains cas être voulu: considérons par exemple le cas de 2 types de traitements A et B qui peuvent être exécutés sur un fichier donné:

- A: traitement global sur tout le fichier demandant un accès exclusif;
- B: une suite de traitements individuels de mises à jour de segments.

Le traitement B commence à réserver le fichier en partagé, puis effectue des transactions atomiques successives sur différents segments; chaque transaction atomique libère les segments concernés, mais ne libère pas le fichier. Il peut ainsi y avoir des traitements de type B simultanés, mais ceux ci sont en exclusion mutuelle avec le traitement A. On évite ainsi que le traitement A ne puisse s'insérer entre deux transactions atomiques de B.

Notons que la cohérence des données repose sur l'indépendance des transactions successives, et donc sur la responsabilité de l'utilisateur-programmeur. Les notions d'objets et d'opérations sur les objets qui seront presentées dans le chapitre 5 essaieront d'alléger cette responsabilité du programmeur.

3 - MISE EN OEUVRE DES TRANSACTIONS EMBOITEES A VERROUILLAGE SELECTIF

Dans cette partie nous allons décrire les algorithmes des processus Producteur, Consommateur ; et Moniteur, nécessaires pour l'exécution de transactions emboîtées à verrouillage sélectif. Les algorithmes seront décrits à l'aide de Réseaux de Nutt et ils spécifieront les actions concernant le traitement normal des transactions ainsi que le traitement des situations d'exeption.

3.1 - Tables utilisées par les processus Producteur, Consommateur et Moniteur

Tous les processus du système peuvent consulter un répertoire, nommé "Table de Fichiers" (TF), qui indique la localisation de chaque fichier de données. Ce répertoire est dupliqué sur chaque site, et sa configuration peut être représentée de la manière suivante :

3.1.1 - Répertoires Globaux

TF (Table de Fichiers) :

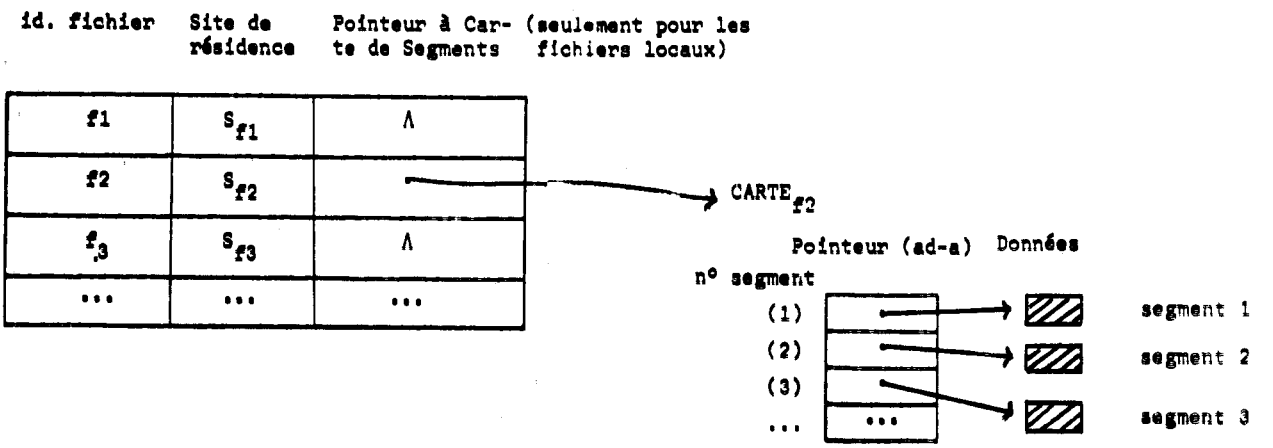


Fig. 4 : Configuration de la Table de Fichiers (TF).

La version de la Table de Fichiers sur un site contient pour chaque fichier local un pointeur vers sa Carte de Segments ; pour les autres fichiers, seulement le site de résidence est indiqué.

3.1.2 - *Tables propres au processus Producteur d'une transaction*

Le Producteur d'une transaction gère un Descripteur de la transaction qui contient les informations suivantes :

DES (Descripteur pour le Producteur) :

- *tr* : identificateur de la transaction,
- *étatr* : état de la transaction
- *site-p* : identificateur du site du Producteur,
- *util* : identificateur de l'utilisateur (i.e. de son terminal),
- *niveau* : niveau d'emboîtement de la transaction interne en cours,
- *NA* : niveau d'emboîtement à annuler (dans le cas d'une annulation),
- *RI* : indicateur de risque d'interblocage :

  RI = { 'vrai' si un risque d'interblocage a été détecté pour la transaction,
  'faux' dans le cas contraire.

Le Producteur gère en plus la Liste des Consommateurs (LC) laquelle contient les identificateurs des sites des Consommateurs de la transaction.

3.1.3 - *Tables propres à un processus Consommateur d'une transaction*

Chaque Consommateur d'une transaction gère une version locale du Descripteur de la transaction contenant les informations suivantes :

DES (Descripteur pour le Consommateur) :

. *tr* : identificateur de la transaction
. *étatr* : état de la transaction,
. *site-p* : identificateur du site Producteur,
. *site-c* : identificateur du site Consommateur,
. *niveau* : niveau d'emboîtement de la transaction interne en cours,
. *NV* : niveau d'emboîtement à valider,
. *NA* : niveau d'emboîtement à annuler,
. *RI* : indicateur de risque d'interblocage.

Chaque Consommateur possède une copie locale de la Liste de Consommateurs (LC) de la transaction et gère en plus la Pile d'Intentions des modifications concernant les fichiers locaux ; la configuration de cette Pile peut être représentée de la manière suivante :

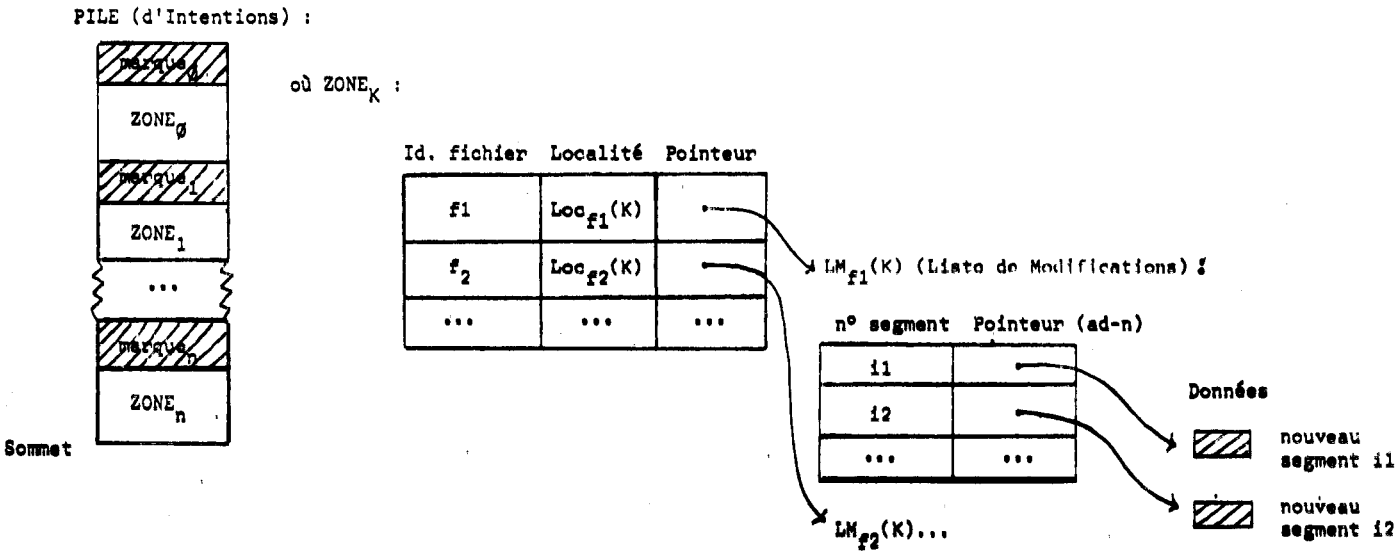


Fig. 5 : Configuration d'une Pile d'Intentions (PILE) d'un Consommateur.

Chaque $ZONE_K$ de la PILE contient les intentions concernant les fichiers locaux référencés dans la transaction interne de niveau K d'emboîtement. Pour un tel fichier f, la $ZONE_K$ indique :

. $Loc_f(K)$: indicateur de localité : égal à 'vrai' si le fichier f a été ouvert dans la $ZONE_K$; égal à 'faux' dans le cas contraire (il a été ouvert dans une zone antérieure).

. $LM_f(K)$: Liste de Modifications du fichier f, apportées par la $ZONE_K$. Comme avant les écritures de la transaction sont réalisées sur des segments nouveaux (pris de l'espace libre) sans affecter les fichiers mêmes ; ainsi, les Listes de Modifications contiennent les adresses nouvelles des segments de fichier qui ont été l'objet d'écritures.

3.1.4 - Tables propres au processus Moniteur d'un site

Le Moniteur de chaque site gère une File de transactions en attente d'Identification (FID) contenant les identificateurs des utilisateurs qui veulent lancer une nouvelle transaction sur le site (le Moniteur attendra le passage du "Jeton de Contrôle" pour assigner des "tickets" à ces nouvelles transactions).

Le Moniteur gère en plus une Table de Producteurs locaux (TPRO) contenant les identificateurs des transactions qui ont été initialisées sur le site (et qui en conséquence possédent un processus Producteur), et une Table de Consommateurs locaux (TCON) dont la configuration peut être la suivante :

TCON (Table de Consommateurs locaux) :

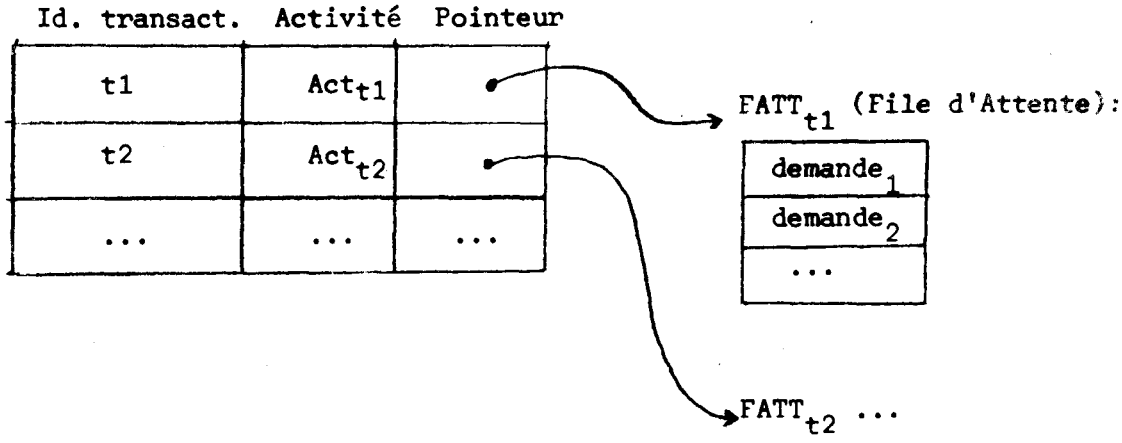


Fig. 6 : Configuration de la Table de Consommateurs locaux (TCON) d'un site.

Pour chaque transaction t qui possède un processus Consommateur dans le site, la Table de Consommateurs locaux (TCON) indique :

. Act_t : indicateur d'activité : égal à "vrai" s'il existe une demande de t en exécution ou en attente sur le site ; égal à "faux" dans le cas contraire.

. $FATT_t$: File d'attente contenant les demandes de t en attente de traitement de la part du Consommateur concerné (C_t).

Pour la gestion de verrous, le Moniteur de chaque site possède une copie du Tableau d'Incompatibilités (TI) entre les verrous hiérarchiques IS, IX, S, SIX, et X (figure 1 de ce chapitre).

$$TI(v_i, v_k) = \begin{cases} \text{vrai si les verrous } v_i \text{ et } v_k \text{ sont incompatibles} \\ \text{faux sinon.} \end{cases}$$

Aussi pour la gestion de verrous, le Moniteur de chaque sité gère une Table de verrous sur les fichiers locaux (TVF). La configuration de cette table peut être la suivante :

TVF (Table de verrous sur fichiers) :

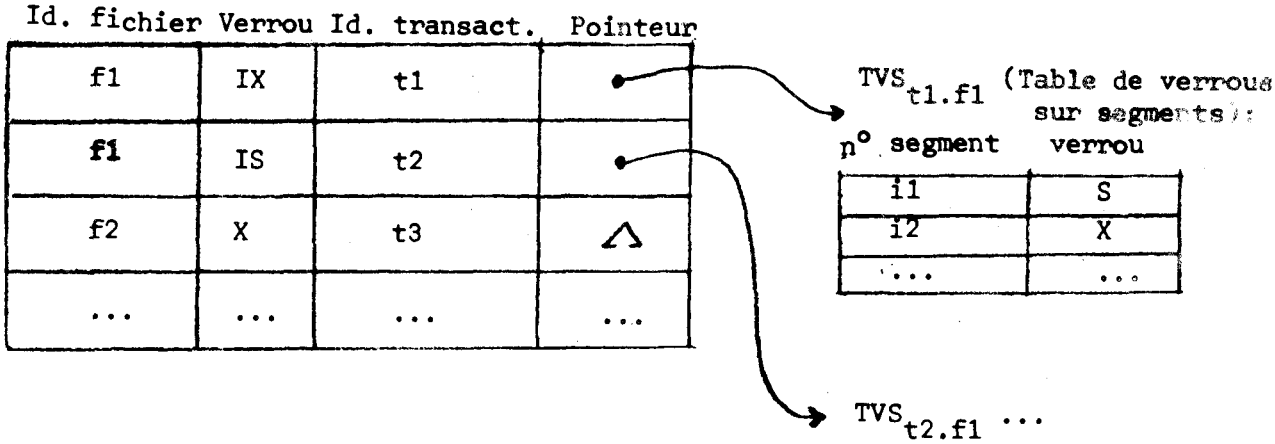


Fig. 7 : Configuration de la Table de verrous sur les fichiers locaux d'un site (TVF).

Pour chaque fichier F verrouillé sur un site, la Table de verrous sur fichiers (TVF) indique :

. le mode v du verrou
. l'identificateur de la transaction t qui possède le verrou
. $TVS_{t.F}$: Table de verrous sur les segments de F établis par t. Cette table existe lorsque le mode v du verrou sur F est IS, IX, et SIX, et indique le verrou spécifique (S ou X) établi sur chaque segment référencé par une opération de t.

Un même fichier peut être verrouillé par 2 ou plusieurs transactions en modes compatibles ; il y aura alors plusieurs entrées pour ce fichier dans TVF.

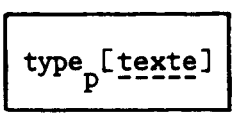
3.2. Notation utilisée dans les réseaux de Nutt.

Dans les réseaux de Nutt, au moyen desquels nous allons décrire les processus Producteur, Consommateur et Moniteur, nous utiliserons la notation suivante (extension de celle utilisée dans le chapitre III (partie 4.1)) :

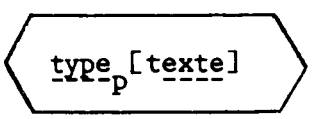
- *Etat d'un processus* : (état)
- *Requêtes* : d'après la configuration d'un site, présentée dans le chapitre III (partie 6), les requêtes que les processus du système

se communiquent entre eux sont de 4 types : C, D, R, et S, selon qu'elles contiennent une commande (C), une demande (D), une réponse (R), ou un signal (S) (les derniers servent à signaler les situations d'exception, et ne font pas partie du protocole demande-réponse). Pour une requête arrivée à un processus, on utilisera 3 sortes de symboles :

commande venant du terminal ;



requête interne, délivrée par un processus local (p dénote le processus destinateur final de la requête, dans le cas d'une chaîne de transmissions) ;

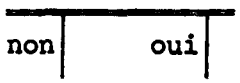


requête externe (venant d'un processus situé sur un autre site, et délivrée par le sous-système de transport).

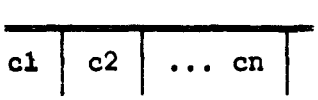
- *Transitions* : les transitions déclenchées par l'arrivée d'une requête à un processus peuvent être associées à la réalisation d'une action locale, à l'évaluation d'une condition, ou à la transmission d'une requête à un autre processus. Ces transitions seront représentées par les symboles suivants :

——— a Transition associée à la réalisation d'une action locale a qui peut être de la forme :

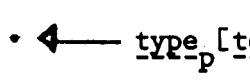
  . assigner une valeur à une variable ;
  . une action sur une liste L : INS_L[élém.], CON_L[élém], EXT_L[élém.] (insérer consulter, ou extraire un élément de L) ;
  . une opération sur des données d'un fichier local : lire, écrire, etc ...
  . afficher un message sur le terminal de l'utilisateur (notification).



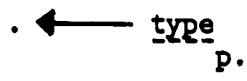
Transition associée à l'évaluation d'une condition booléenne C.



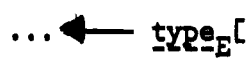
Transition associée à l'évaluation d'une condition c laquelle est une variable qui peut prendre plusieurs valeurs : c1, c2,...,cn.



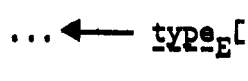
Transition associée à la *transmission interne* d'une requête communiquée à un processus local p ; le type est C, D, R, ou S, selon le type de la requête. S'il s'agit d'une requête qui doit passer par une châine de processus intermédiaires avant d'être délivrée à son processus destinateur, alors p devient de la forme : $p, q.s_q$ où p est le prochain processus de la chaîne, q est le processus destinateur final, et s_q est le site de q.



Transition associée à la *transmission externe* d'une requête délivrée au sous-système de transport pour être envoyée à un processus p situé sur un site s_p (comme avant $p.s_p$ devient $p.s_p$, q.s s'il s'agit d'une chaîne de transmissions).



Transition associée à la *diffusion interne* d'une requête communiquée à un ensemble E de processus locaux (E = {p1 ; p2 ; ... ; pn}).



Transition associée à la *diffusion externe* d'une requête délivrée au sous-système de transport pour être envoyée à un ensemble E de processus situés sur d'autres sites. E est de la forme $\{p1.s_{p1} ; \ldots ; pn.s_{pn}\}$, ou $p.\{s1 ; \ldots ; sn\}$ si tous les processus ont le même nom p.

3.3 Processus producteur.

3.3.1. *Traitement normal*

Pour une transaction t, les figures 8 (A et B) représentent le traitement normal réalisé par son processus Producteur P_t. Ce traitement peut être

résumé de la manière suivante :

- *Initialisation de la transaction* : lorsque la nouvelle transaction a reçu un identificateur t, son processus Producteur P_t est activé par le Moniteur du site (signal "DEBUT-TR"); P_t initialise alors le Descripteur de la Transaction (DES), principalement avec les informations suivantes : tr := t ; etatr := INITIAL ; niveau := ∅ ; ... . Après ceci, P_t se met en connexion avec le terminal de l'utilisateur pour recevoir les commandes de la transaction.

- *Réception de la commande OUVRIR (F,v)* : P_t consulte dans la Table de Fichiers (TF) le site de résidence du fichier F ; si ce site, SF, n'est pas encore dans la liste de Consommateurs (LC), alors P_t passe à son Moniteur (processus M) la demande "DEBUT-TR" destinée au site SF laquelle a pour but d'activer un processus Consommateur C_t sur SF. Dans ce cas, P_t passe en plus au Moniteur la demande "OUVRIR (F,v)" destinée au site SF et se met en attente de réponse à la première demande (en fait la demande est destinée au processus Consommateur C_t du site SF). Lorsque la réponse "ACK-DEBTR (SF)" arrive, P_t insère l'identificateur SF dans la Liste de Consommateurs (LC) et se met en connexion avec l'utilisateur pour la prochaine commande. (En règle générale P_t envoie des demandes et reçoit des réponses par l'intermédiaire de son Moniteur M).

Dans le cas où le site de résidence du Fichier F figure déjà dans sa Liste de Consommateurs, P_t passe à son Moniteur seulement la demande "OUVRIR (F,v)" destinée au site SF et se met en connexion avec l'utilisateur pour la prochaine commande.

- *Réception de la commande ECRIRE (F, i, b)* : P_t passe à son Moniteur la demande "ECRIRE (F, i, contenu (b))" (avec les données contenues dans la zone-tampon b) destinée au site de résidence du fichier F. Après ceci, P_t est prêt à recevoir la prochaine, commande de l'utilisateur.

- *Réception des commandes LIRE (F, i, b) ou LIEC (F, i, b)* : P_t passe à son Moniteur la demande en question, "LIRE (F, i, b)" ou "LIEC (F, i, b)", destinée au site de résidence du fichier F, et se met en attente de la réponse correspondante. Lorsque cette réponse arrive, "ACK-LIRE (b, (données))", P_t place les données reçues dans la zone-tampon b, et notifie à l'utilisateur.

Figure 8 A : Processus Producteur P_t : traitement normal

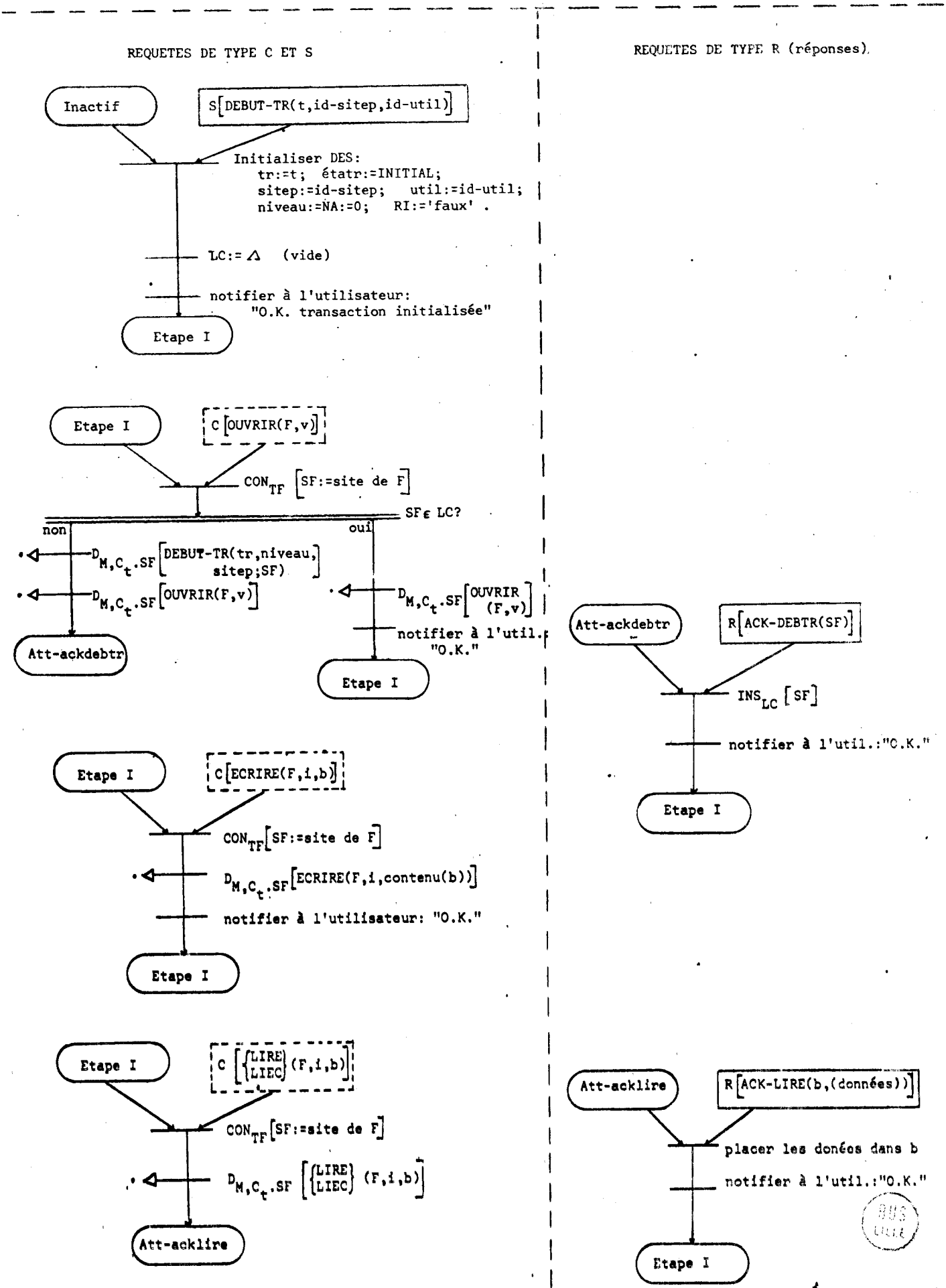
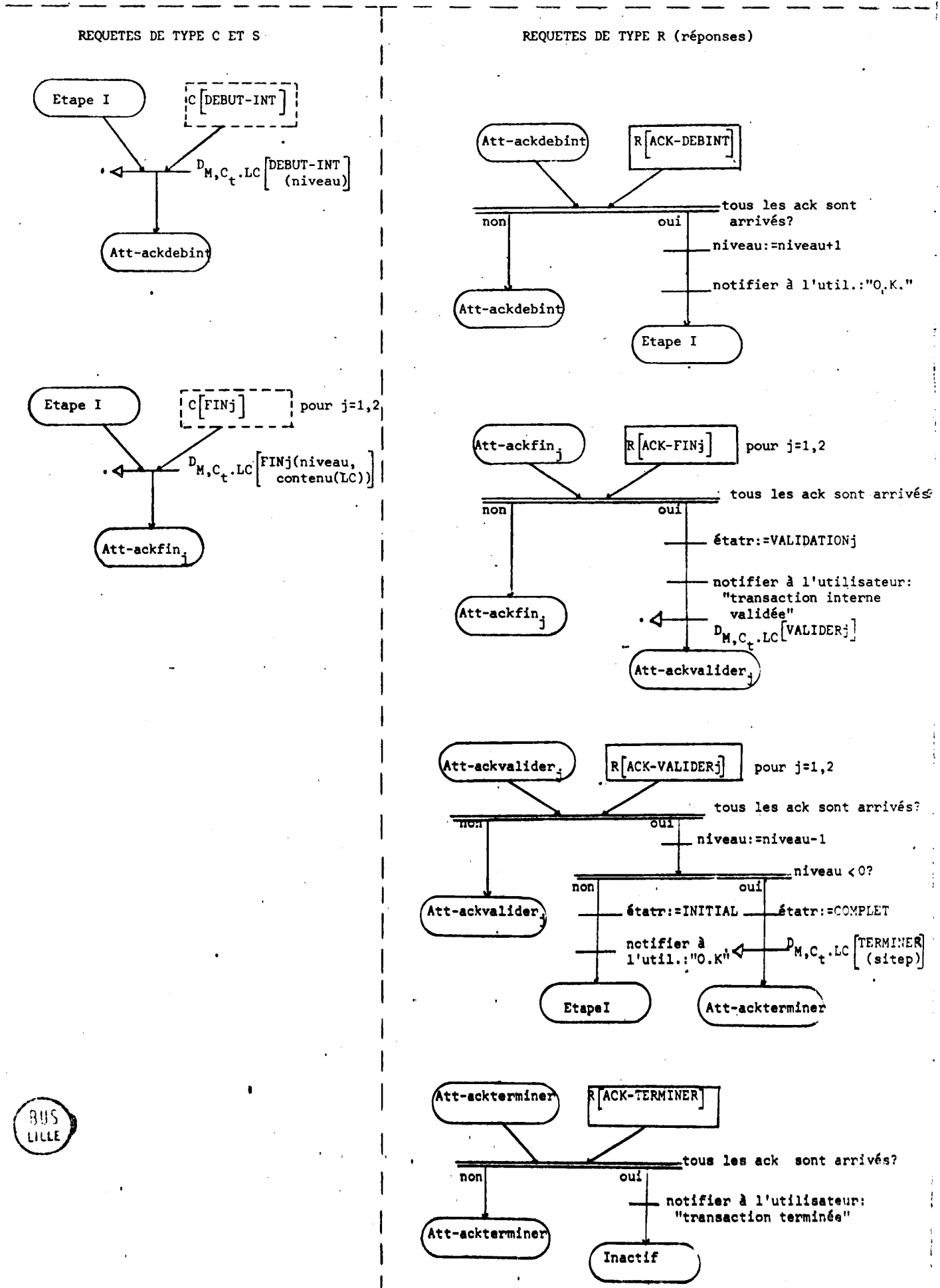


Figure 8 B : Processus Producteur P_t : traitement normal (continuation)



BUS LILLE

- *Réception de la commande DEBUT-INT* : avec cette commande l'utilisateur signale le début d'une transaction interne (point de sauvegarde). P_t passe à son Moniteur la demande "DEBUT-INT (niveau)" destinée à être diffusée à tous les consommateurs de la transaction. Après ceci, P_t attend les réponses ("ACK-DEBINT") de tous les consommateurs pour incrémenter sa variable *niveau* (d'emboîtement) et notifier à l'utilisateur.

- *Réception des commandes FIN1 ou FIN2* : avec cette commande l'utilisateur signale la fin de la transaction interne en cours (et aussi un point de cohérence, s'il utilise FIN2). P_t déclenche une Validation en 2 étapes ainsi : P_t passe à son Moniteur la demande correspondante, "FIN 1 (niveau, contenu (LC))" ou "FIN2 (niveau, contenu (LC))" (avec le contenu de la Liste de Consommateurs), destinée à être diffusée à tous les Consommateurs de la transaction, et après ceci, P_t se met en attente des réponses correspondantes ("ACK-FIN1" ou "ACK-FIN2") de tous les consommateurs.

Lorsque toutes ces réponses arrivent, P_t marque sa variable *étatr* en VALIDATION 1 ou VALIDATION 2, selon le cas ; ceci constitue le point de Validation de la transaction interne en cours, et en conséquence P_t peut notifier à l'utilisateur que cette transaction interne sera achevée totalement (malgré les défaillances). Ensuite, P_t passe à sont Moniteur la demande "VALIDER 1" ou "VALIDER 2", selon le cas, destinée à être diffusée à tous les Consommateurs de la transaction, et se met en attente des réponses correspondantes ("ACK-VALIDER 1" ou "ACK-VALIDER 2") de tous les consommateurs.

Lorsque toutes ces nouvelles réponses sont arrivées, P_t considère la transaction interne comme totalement terminée et décrémente sa variable niveau (d'emboîtement). En plus, P_t met à jour sa variable *étatr* ainsi : si niveau ≥ 0 alors *étatr* devient INITIAL et l'utilisateur peut soumettre de nouvelles commandes ; mais si niveau < 0 alors la transaction globale est totalement terminée et dans ce cas *étatr* devient COMPLET et P_t passe à son Moniteur la demande "TERMINER (sitep)" destinée à être diffusée afin que tous les processus Consommateurs de la transaction deviennent inactifs. Lorsque P_t reçoit les réponses de tous les consommateurs, lui-même devient inactif.

3.3.2 *Traitement des situations d'exception.*

Nous considérons à présent 3 sortes de situations d'exception pour une transaction : demande d'annulation partielle faite par l'utilisateur, détection d'un risque d'interblocage, et détection d'une défaillance du système affectant

la transaction. Pour une transaction t, la figure 9 représente le traitement des situations d'exception réalisé par son processus Producteur P_t ; ce traitement peut être résumé de la manière suivante :

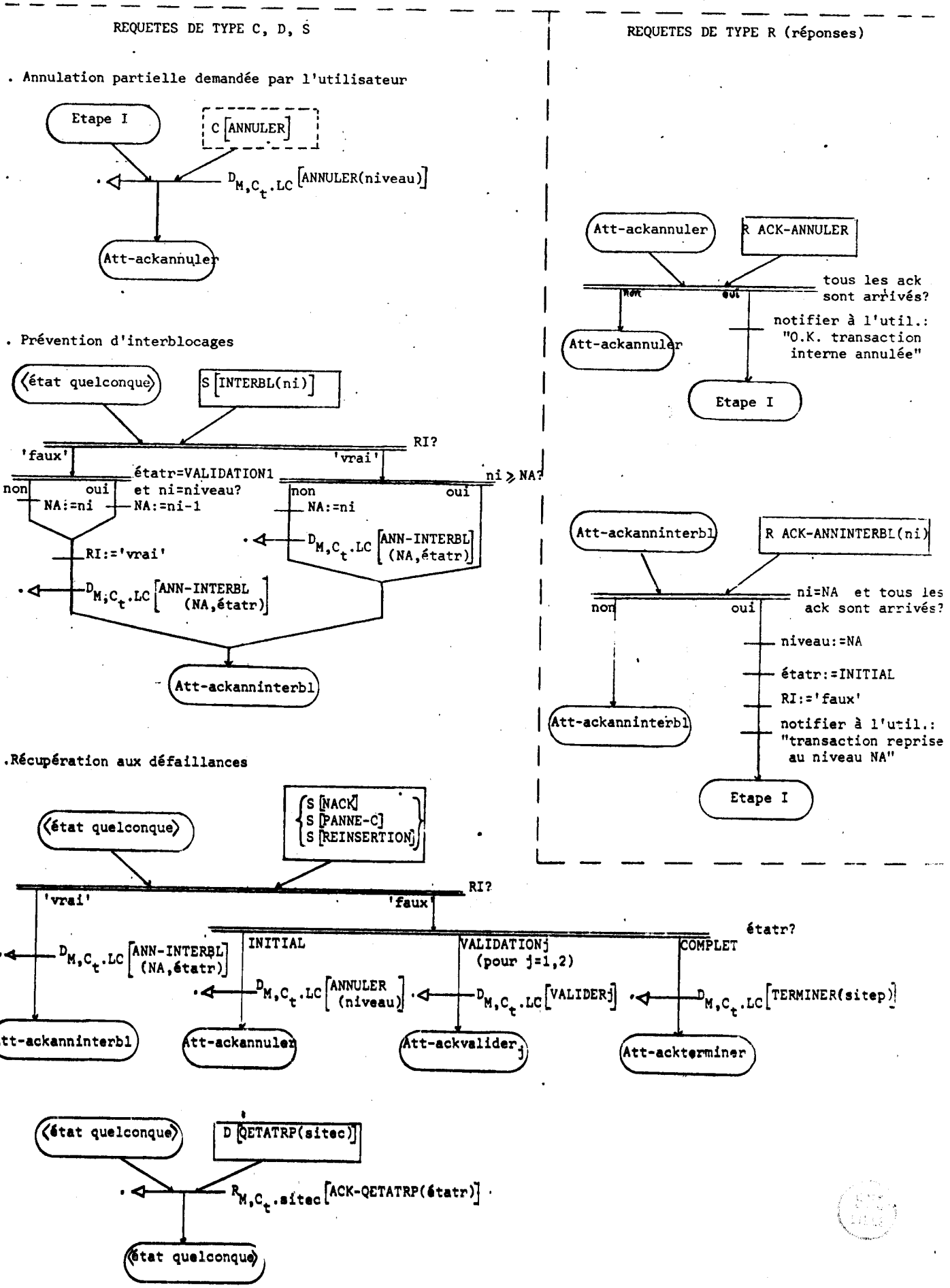
- *Annulation partielle demandée par l'utilisateur* : lorsque P_t reçoit la commande "ANNULER" de l'utilisateur, il déclenche l'annulation de la transaction interne en cours en passant à son moniteur (processus M) la demande "ANNULER (niveau)" destinée à être diffusée à tous les consommateurs de la transaction, après quoi, P_t se met en attente des réponses de tous les consommateurs. Lorsque toutes ces réponses sont arrivées, P_t le notifie à l'utilisateur, qui pourra maintenant demander des nouvelles opérations, pour remplacer celles annulées dans la transaction interne en cours, ou demander la fin de cette transaction interne.

- *Prévention d'Interblocage* : P_t peut recevoir (par l'intermédiaire de son Moniteur) un signal "INTERBL(ni)" indiquant qu'un risque d'interblocage concernant la transaction t a été détecté dans le système, et que sa résolution entraîne l'annulation partielle de t à partir du niveau d'emboîtement ni (cela afin de libérer les fichiers ouverts à partir de ce niveau).

Lorsque P_t reçoit le signal "INTER BL (ni)", il établit son indicateur de risque d'interblocage RI à "vrai" et sa variable "niveau à annuler" NA à ni (dans le cas où *étatr* est VALIDATION 1 et ni = niveau, P_t doit supposer qu'après la validation en cours les fichiers en conflit se trouveront dans le niveau ni-1 ; en conséquence P établit NA à ni-1). Ensuite P_t déclenche une annulation partielle de la transaction en passant à son moniteur la demande "ANN-INTERBL (NA, étatr)" destinée à être diffusée à tous les consommateurs de la transaction, après quoi, P_t se met en attente des réponses. (Si pendant cette attente, P_t reçoit à nouveau un signal "INTERBL(nj)", P_t l'acceptera uniquement s'il entraîne annuler la transaction à partir d'un niveau nj inférieur au précédent (NA) ; dans ce cas, P_t met à jour sa variable NA (NA := nj) et passe à son Moniteur une nouvelle demande "ANN-INTERBL (NA, étatr)").

Lorsque toutes les réponses des consommateurs sont arrivées, indiquant que la transaction a été annulée à partir du niveau d'imbrication NA (réponses "ACK-ANNINTERBL (ni)" pour ni = NA), P_t met à jour sa variable niveau (niveau := NA), marque *étatr* à INITIAL, remet son indicateur de risque d'interblocage RI en "faux" et notifie à l'utilisateur l'annulation partielle indésirée que la transaction a subi. L'utilisateur peut alors demander des nouvelles opérations pour remplacer celles annulées.

Figure 9 : Processus Producteur P_t : traitement des situations d'exception



- *Récupération aux défaillances* : P_t peut recevoir de son Moniteur 3 sortes de signaux en référence aux situations de défaillances :

  . le signal "NACK" indique que le protocole de communication a détecté pour la transaction t, le manque d'un message-réponse après un délai maximum de temps ;

  . le signal "PANNE-C" indique qu'un processus Consommateur de t a subi un isolement temporaire, à cause d'une panne affectant son site ;

  . le signal "REINSERTION" indique la réinsertion au réseau de communication du propre site de P_t, après avoir subi une panne.

A la réception d'un de ces 3 signaux, P_t déclenche la récupération de la transaction ainsi : s'il y a une annulation partielle en cours pour résoudre un risque d'interblocage (indiquée par RI = "vrai"), alors P_t passe à son Moniteur la demande "ANN-INTERBL (NA, étratr)" destinée à être diffusée à tous les Consommateurs de la transaction, et se met en attente des réponses (en effet, cette diffusion peut avoir été empêchée par la défaillance signalée). Dans l'autre cas (RI = "faux"), P_t examine sa variable *étatr* pour choisir l'action de récupération appropriée :

  . dans le cas *étatr* = INITIAL, la transaction interne en cours doit être annulée : P_t passe alors à son Moniteur la demande "ANNULER (niveau)" pour être diffusée à tous les consommateurs ;

  . dans les cas *étatr* = validation 1 ou VALIDATION 2, la transaction interne en cours a été validée avant la défaillance et doit être complétée : P_t passe alors à son Moniteur la demande "VALIDER 1" ou "VALIDER 2", selon le cas, pour être diffusée à tous les Consommateurs ;

  . dans le cas *étatr* = complet, la transaction totale avait fini avant la défaillance : P_t passe alors à son Moniteur la demande "TERMINER (sitep)" pour être diffusée à tous les consommateurs afin qu'ils deviennent inactifs.

Après avoir réalisé l'action de récupération adéquate, P_t se met en attente des réponses correspondantes (déjà mentionnées dans la partie du Traitement normal).

P_t peut recevoir en plus, une demande exceptionnelle "QETATRP (*sitec*) venant d'un processus Consommateur (du site *sitec*) qui veut connaître l'état actuel de la transaction. Dans ce cas, P_t envoie au Consommateur en question la réponse "ACK-Q ETATR P (*etatr*)" contenant sa version propre de l'état de la transaction.

3.4 Processus consommateur.

3.4.1 Traitement normal.

Les figures 10 (A, B, C, D et E) représentent le traitement normal réalisé par chacun des processus Consommateurs C_t d'une transaction t. Ce traitement est constitué par les ensembles d'actions déclenchées dans le processus C_t par la réception des diverses demandes envoyées par le processus Producteur P_t (ces demandes sont délivrées à C_t par l'intermédiaire du Moniteur de son site). On peut résumer le traitement normal d'un processus Consommateur C_t de la manière suivante :

- *Réception de la demande DEBUT-TR (t, ni, id-sitep, id-sitec)* : cette demande déclenche l'activation du processus C_t par son Moniteur. D'abord C_t initialise sa version du Descripteur de la transaction (DES) avec les informations reçues en paramètre. Ensuite C_t initialise sa PILE d'intentions : pour cela il doit introduire une marque pour chaque niveau d'emboîtement de la transaction jusqu'au niveau en cours, indiqué par le paramètre ni ; ces marques vont délimiter les différentes zones d'intentions (initialement vides) correspondant aux diverses transactions internes de la transaction la plus externe. Finalement, C_t envoie au Producteur P_t la réponse "ACK-DEBTR (sitec)" (toujours par l'intermédiaire de son Moniteur : processus M).

- *Réception de la demande OUVRIR (F,v)* : C'est le Moniteur du site de C_t qui établit le verrou demandé v sur le Fichier F, avant de délivrer cette demande au Consommateur C_t ; ce dernier doit alors introduire dans la zone en cours de la PILE d'intentions les informations suivantes sur le Fichier F :

  . l'identificateur du fichier, F ;
  . l'indicateur de localité de F (LOC_F) marqué à "vrai" (parce que le fichier vient d'être ouvert dans cette zone) ;

. la liste de Modifications de F (LM_F) initialisée à vide, pour cette zone.

Finalement C_t envoie au Producteur P_t la réponse correspondante ("ACK-OUVRIR").

- *Réception de la demande ECRIRE (F, i, (données))* : si le fichier F ne figure pas dans la zone en cours de la PILE d'intentions (parce qu'il a été ouvert dans une zone antérieure plus englobante), alors C_t introduit dans la PILE les informations concernant F pour cette zone (son identificateur, son indicateur de localité (LOC_F) marqué à "faux", sa liste de Modifications (LM_F) initialement vide).

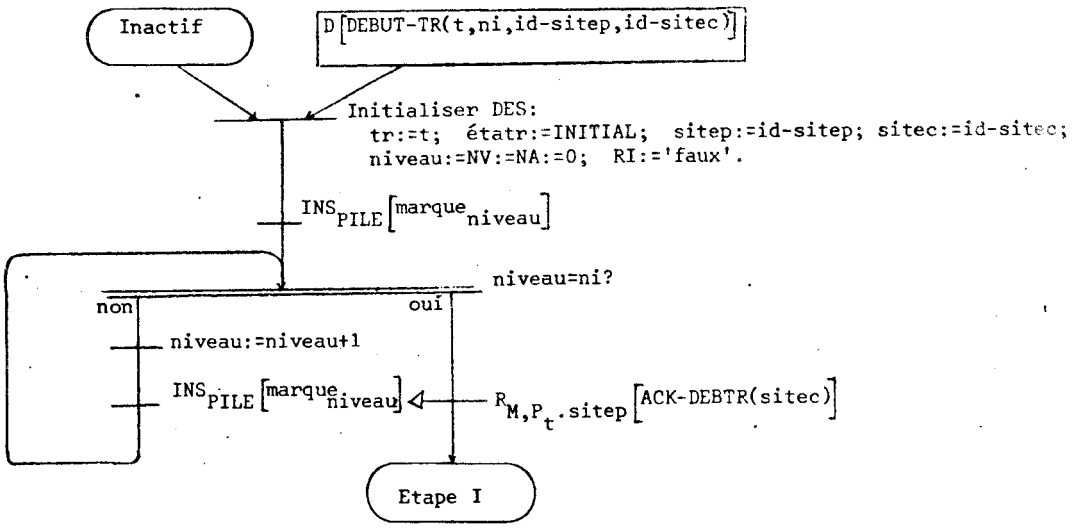
Si c'est la première fois que le segment logique i de F est l'objet d'une écriture dans la zone en cours, alors C_t effectue l'écriture des données sur un segment physique libre, dont son adresse (ad-n) est notée dans la Liste de Modifications de F (LM_F) pour cette zone. Mais s'il s'agit d'une nouvelle écriture dans la zone en cours sur le segment logique i, alors C_t réutilise le même segment physique de l'écriture antérieure.
Pour finir, C_t envoie au Producteur P_t la réponse correspondante ("ACK-ECRIRE").

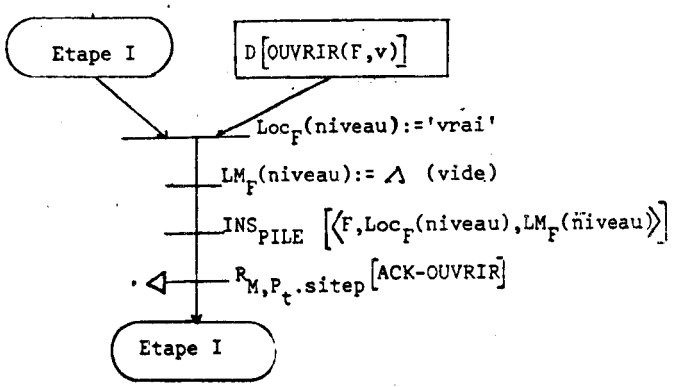
- *Réception des demandes LIRE (F, i, b) ou LIEC (F, i, b)* : si le fichier F ne figure pas encore dans la zone en cours de la PILE d'intentions, alors C_t introduit les informations concernant F pour cette zone.
Si le segment i de F n'a pas été l'objet d'écriture dans aucune zone de la PILE, alors C_t effectue la lecture des données directement du fichier F (à l'adresse de i indiquée dans la Carte de Segments de F). Mais dans le cas contraire, C_t doit lire la version la plus récente du segment i, c'est à dire, à l'adresse indiquée par la liste de Modifications de F (LM_F) de la zone qui a effectué la dernière écriture sur i. Après la lecture, C_t envoie au Producteur P_t la réponse "ACK-LIRE (b, (données))" contenant les données lues.

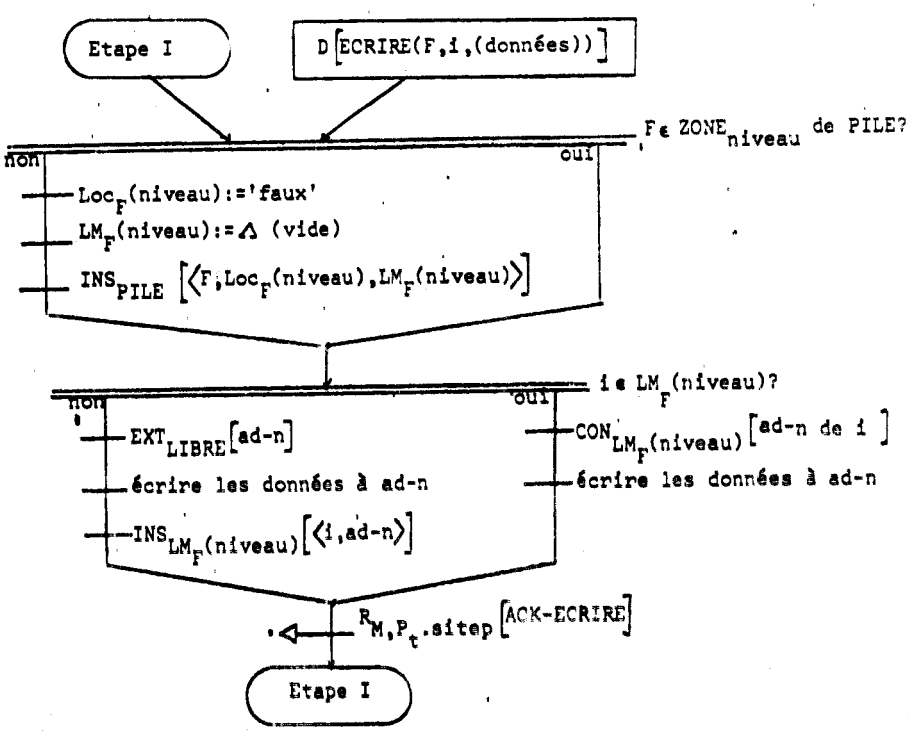
- *Réception de la demande DEBUT-INT (ni)* : dans le cas où le niveau courant de C_t ne coïncide pas avec celui du Producteur P_t (i.e. ni ≠ niveau), alors C_t informe le producteur de cette anomalie (signal "PANNE-C") et se met en attente de la récupération globale de la transaction. Dans l'autre cas, C_t incrémente sa variable niveau (d'emboîtement) et introduit dans sa PILE une marque pour commencer une nouvelle zone correspondante à la nouvelle transaction interne. Après ceci, C_t envoie au Producteur P_t la réponse "ACK-DEBINT".

Figure 10 A : Processus Consommateur C_t : traitement normal.

REQUETES DE TYPE D (demandes)

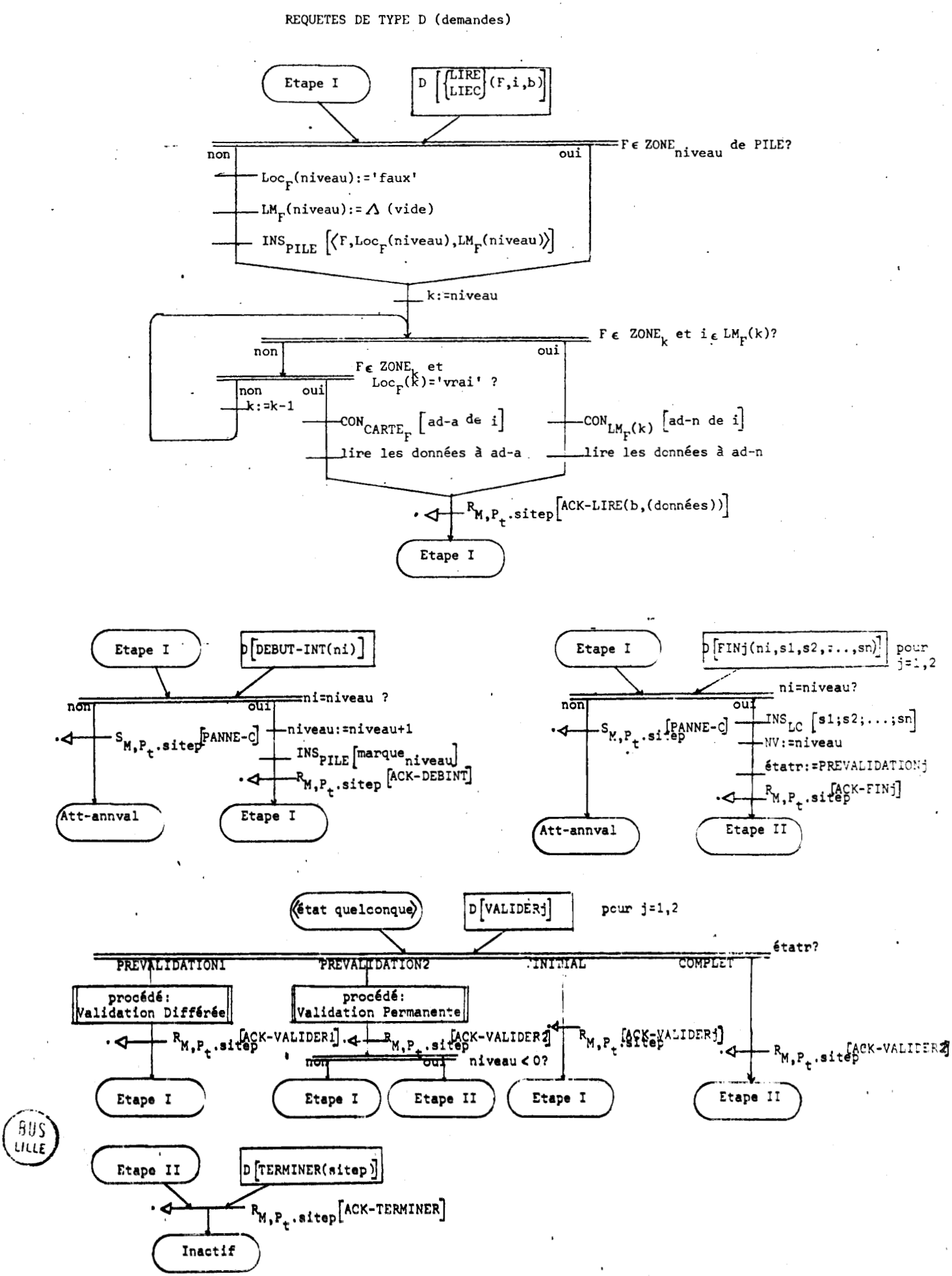






BUS LILLE

Figure 10 B : Processus Consommateur C_t : traitement normal (continuation)

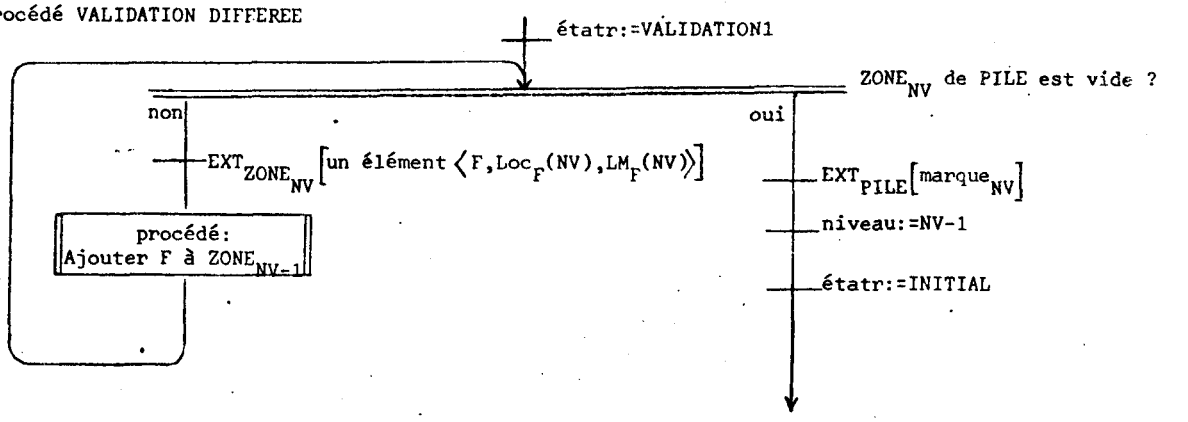


BUS LILLE

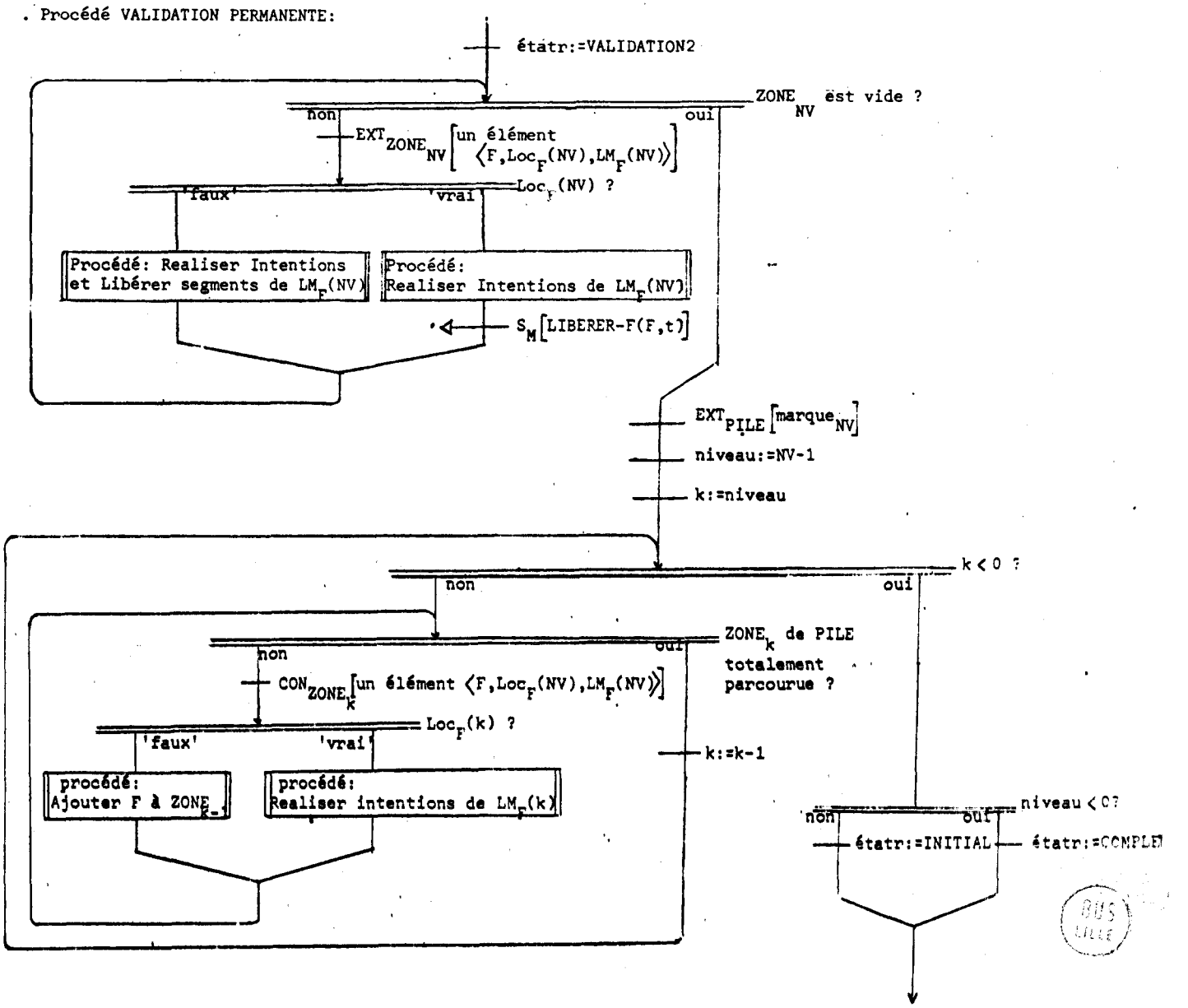
Figure 10 C : Processus Consommateur C_t : traitement normal (continuation)

PROCEDES UTILISES

. Procédé VALIDATION DIFFEREE



. Procédé VALIDATION PERMANENTE:

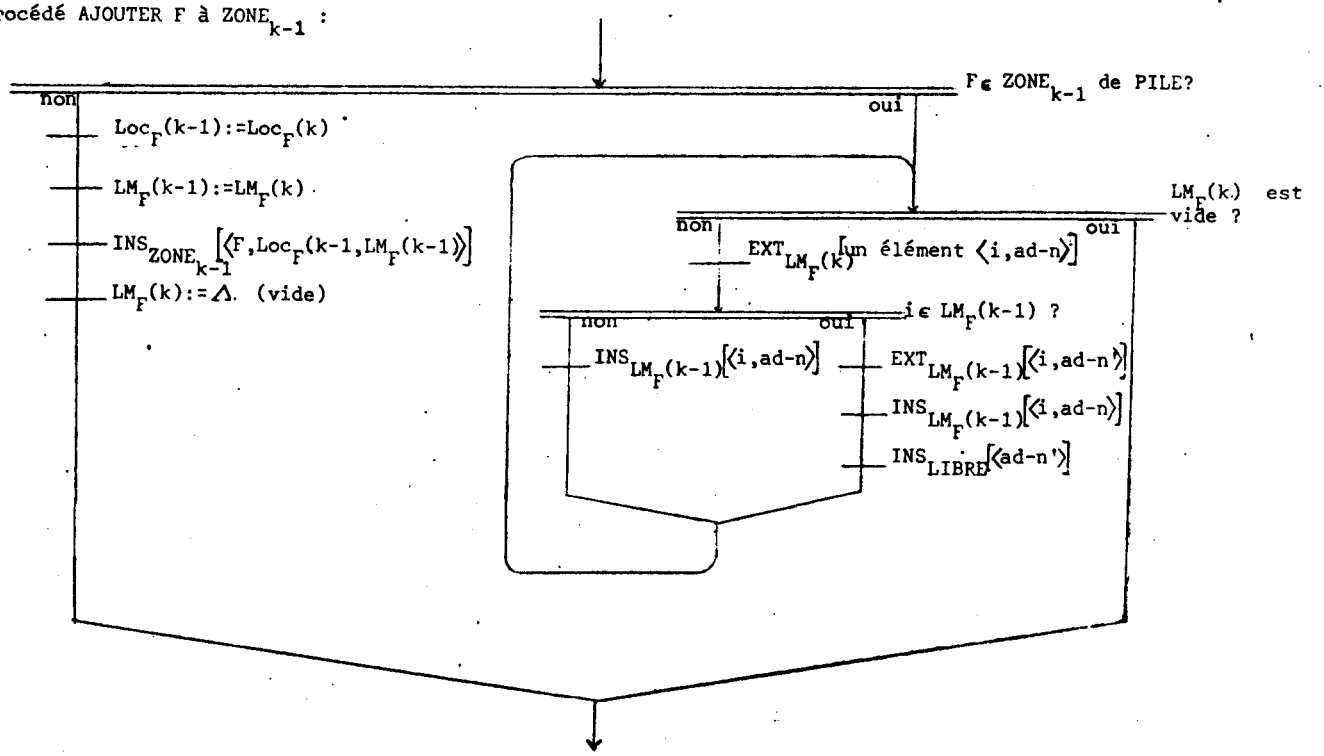


BUS LILLE

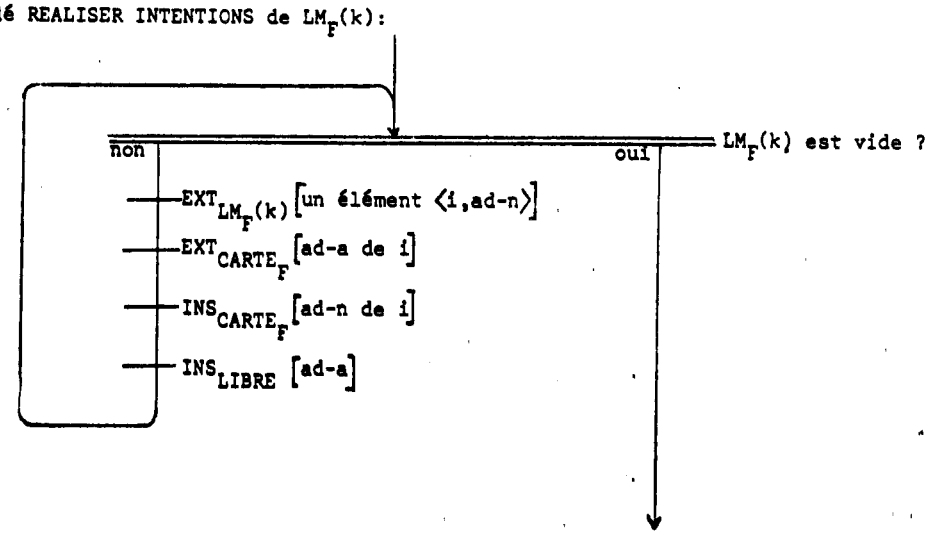
Figure 10 D : Processus Consommateur C_t : traitement normal (continuation)

PROCEDES UTILISES

. Procédé AJOUTER F à $ZONE_{k-1}$:



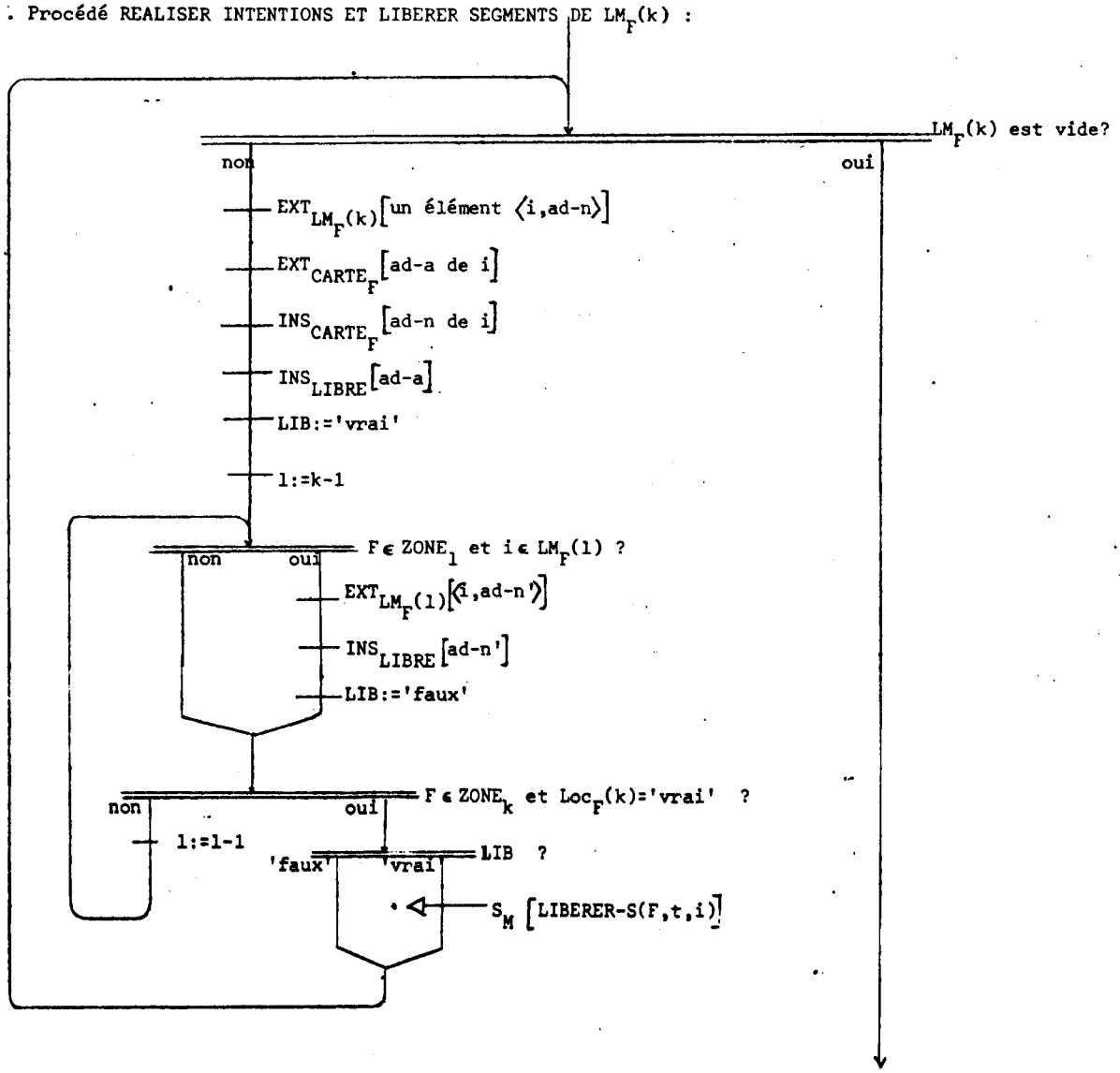
. Procédé REALISER INTENTIONS de $LM_F(k)$:



BIS LILLE

Figure 10 E : Processus Consommateur C_t : traitement normal (continuation)

PROCEDES UTILISES



BUS LILLE

- *Réception des demandes FIN 1 (ni, s1, s2,...,sn) ou FIN 2 (ni, si, s2,...,sn)* : comme avant dans le cas où ni ≠ niveau C_t informe le producteur de cette anomalie (signal "PANNE-C") et se met en attente de la récupération globale de la transaction. Dans l'autre cas C_t actualise sa version de la Liste de Consommateurs de la transaction (LC) avec les identificateurs reçus en paramètre.
En plus, C_t assigne à sa variable NV (niveau a valider) le niveau d'emboîtement de la transaction interne en cours (niveau), et marque sa version de l'état de la transaction, *étatr* à PREVALIDATION 1 ou PREVALIDATION 2, selon le cas. Ensuite, C_t envoie au Producteur P_t la réponse correspondante ("ACK-FIN 1" ou "ACK-FIN 2").

- *Réception des demande VALIDER 1 ou VALIDER 2* : si *étatr* de C_t est PREVALIDATION 1 à la réception de la demande VALIDER 1, C_t réalisera alors une "*Validation Différée*" de la transaction interne en cours ; en effet, il s'agit ici de la suppression du point de sauvegarde de la transaction interne qui vient de finir, en intégrant ses intentions à celles de la transaction englobante, mais sans réaliser effectivement ces intentions.

De même, si *étatr* de C_t est PREVALIDATION 2 à la réception de la demande VALIDER 2, C_t réalisera une "*Validation Permanente*" de la transaction interne en cours ; il s'agit ici d'un point de cohérence de toute la transaction qui entraîne la réalisation de toutes les intentions accumulées dans la PILE d'intentions, la libération de toutes les ressources utilisées dans la transaction interne qui vient de finir et la suppression du point de sauvegarde associé à cette transaction interne. La libération de ressources comporte, d'une part, les fichiers ouverts par la transaction interne et d'autre part, ceux segments d'autres fichiers qui ont été utilisés proprement par la transaction interne.

Après avoir réalisé la validation demandée, C_t envoie au Producteur P_t la réponse correspondante ("ACK-VALIDER 1" ou "ACK-VALIDER 2"), et son *étatr* devient INITIAL (ou COMPLET s'il s'agit de la fin de toute la transaction). Dans le cas où *étatr* est INITIAL ou COMPLET à la réception de la demande VALIDER 1 ou VALIDER 2, C_t envoie simplement la réponse correspondante au Producteur P_t puisque la validation demandée a été déjà réalisée préalablement.

Le procédé pour réaliser une *Validation Différée* est le suivant : d'abord C_t marque *étatr* à VALIDATION 1 ; ensuite, pour chaque fichier f figurant dans la zone à valider de la PILE d'intentions ($ZONE_{NV}$), ses intentions sont ajoutées à la zone immédiatement antérieure ($ZONE_{NV-1}$). Pour ajouter ces intentions de F,

les listes de Modification de F (LM_F) des zones $ZONE_{NV}$ et $ZONE_{NV-1}$ doivent être mélangées de manière appropriée ; c'est à dire pour chaque segment i de F, il doit rester uniquement l'adresse de son écriture la plus récente. Une fois que toutes les intentions de la $ZONE_{NV}$ sont ajoutées à la $ZONE_{NV-1}$, cette dernière zone de la PILE, $ZONE_{NV}$, est éliminée en supprimant sa marque, la variable niveau (d'emboîtement) est mise à jour (niveau := NV-1), et *étatr* est marqué à INITIAL.

Le procédé pour réaliser une *"Validation Permanente"* est le suivant : d'abord C_t marque *étatr* à VALIDATION 2 ; ensuite, chaque fichier f figurant dans la zone à valider de la PILE ($ZONE_{NV}$) est examiné : si F est un fichier qui a été ouvert dans cette zone ($LOC_F(NV)$ = "vrai"), alors les intentions contenues dans la Liste de Modifications sont effectivement réalisées sur le fichier F (i.e. les adresses de sa CARTE de Segments sont mises à jour avec les nouvelles adresses contenues dans LM_F), et en plus, C_t communique un signal à son Moniteur ("LIBERER-F (F,t)") afin de libérer le fichier F. Dans le cas où F est un fichier ouvert dans une autre zone (plus englobante), alors ses intentions de la $ZONE_{NV}$ sont effectivement réalisées (cela implique éliminer les intentions caduquées de zones antérieures qui font référence aux mêmes segments de la $ZONE_{NV}$) ; dans ce cas, seulement les segments de F exclusivement référencés par la $ZONE_{NV}$ sont libérés (au moyen des signaux "LIBERER-S (F,t,i) que C_t passe à son Moniteur)

Après ce traitement, la dernière zone de la PILE, $ZONE_{NV}$, est éliminée en supprimant sa marque, et la variable niveau est mise à jour (niveau := NV-1). Pour les autres zones de PILE, les intentions de chaque fichier sont progressivement ajoutées aux zones antérieures jusqu'à les réunir toutes dans la zone où le fichier a été ouvert ; à ce moment les intentions de modification sont effectivement réalisées sur le fichier ; à la fin de ce traitement la PILE d'intentions devient vide mais les marques des zones et les identificateurs des fichiers y restent (puisqu'ils peuvent continuer à être référencés dans la suite de la transaction).
Finalement, C_t marque sa variable *étatr* à INITIAL (ou à COMPLET s'il s'agit de la fin de toute la transaction).

- *Réception de la demande TERMINER (sitep)* : le processus C_t envoie la réponse correspondante au Producteur P_t ("ACK-TERMINER"), et devient inactif.

3.4.2 *Traitement des situations d'exception.*

Les figures 11 (A et B) représentent le traitement des situations d'exception réalisé par chaque processus Consommateur C_t d'une transaction t ; ce traitement concerne les annulations partielles de la transaction, la prévention d'interblocages et la récupération aux défaillances, et peut être résumé de la manière suivante :

- *Réception de la demande ANNULER (ni)* : C_t réalise alors une annulation des dernières zones de la PILE d'intentions pour revenir au point de sauvegarde associé au début de la transaction interne de niveau ni (c'est la transaction interne en cours pour le Producteur P_t) ; pour réaliser cette annulation, C_t élimine les intentions contenues dans les zones en question, retourne à l'espace libre les segments physiques pris pour les écritures de ces zones, et communique à son Moniteur des signaux "LIBERER-F (f,t)" afin de libérer chaque fichier f ouvert dans ces zones. Après ce procédé la variable niveau devient égale à NA ; C_t marque alors sa variable *étatr* à INITIAL, et envoie au Producteur P_t la réponse "ACK-ANNULER".

- *Réception du signal INTERBLOCAGE (F)* : avec ce signal le Moniteur communique à C_t le risque d'interblocage avec une autre transaction à cause d'un conflit d'accès au fichier local F, lequel se trouve verrouillé par la transaction t. Si le fichier F en conflit va être libéré (i.e. F a été ouvert dans la dernière zone de PILE et *étatr* = PREVALIDATION 2), alors C_t ignore le signal. Dans l'autre cas, C_t doit chercher le niveau K à partir duquel la transaction doit être annulée pour libérer le fichier F ; K étant déterminé, C_t met à jour sa variable NA (NA:=K) ; marque son indicateur de risque d'interblocage RI à "vrai", et communique au Producteur P_t le risque d'interblocage en lui envoyant le signal "INTERBL (NA)". Ensuite, C_t se met en attente de la demande correspondante d'annulation partielle de la transaction ; si pendant cette attente, C_t reçoit de son Moniteur un autre signal "INTERBLOCAGE (f')", alors C_t l'acceptera uniquement s'il entraîne d'annuler la transaction à partir d'un niveau K inférieur au précédent (NA) (dans ce cas NA est mise à jour, et un nouveau signal "INTERBL (NA)" est envoyé au Producteur P_t).

- *Réception de la demande ANN-INTERBL (ni, étatrp)* : cette demande est ignorée par C_t s'il se trouve en attente d'une demande d'annulation partielle de la transaction à partir d'un niveau NA inférieur à ni. Dans l'autre cas, C_t met à jour sa variable NA (NA := ni) et son indicateur RI (RI := "vrai"), et élimine les zones de la PILE d'intentions à partir du niveau NA, en libérant les fichiers ouverts dans ces zones ; avant cette élimination de zones,

C_t réalise une validation de la dernière zone de PILE si son *étatr* est PREVALIDATION j, pour j = 1, 2, et l'*étatr* de P_t, communiqué par le paramètre étatrp, est VALIDATION j.

Finalement, C_t marque son *étatr* à INITIAL, envoie au Producteur P_t la réponse "ACK-ANNINTERBL (NA)", et marque son indicateur RI à "faux".

- *Réception du signal REINSERTION* : ce signal indique à C_t la réinsertion au réseau de communication de son site, après avoir subi une panne. C_t déclenche alors la récupération locale de la transaction ainsi : s'il y a un risque d'interblocage à résoudre (indiqué par RI = "vrai"), alors C_t envoie au Producteur P_t le signal "INTERBL (NA)", et se met en attente de la demande correspondante d'annulation partielle de la transaction. Dans l'autre cas (RI = "faux"), C_t examine sa variable *étatr* pour choisir l'action de récupération appropriée, ainsi :

. dans le cas *étatr* = INITIAL, C_t envoie du Producteur P_t le signal "PANNE-C" lui informant de son isolement temporaire, réalise une Annulation des intentions de la ZONE en cours, et se met en attente de la récupération globale de la transaction (i.e. une demande ANNULER ouVALIDER venant du Producteur) ;

. dans les cas *étatr* = VALIDATION 1 ou VALIDATION 2, C_t réalise, selon le cas, une Validation Différée ou une Validation Permanente de la transaction interne en cours, et envoie la réponse correspondante au Producteur P_t ("ACK-VALIDER 1" ou "ACK-VALIDER 2") ;

. dans le cas *étatr* = COMPLET, C_t envoie au Producteur P_t la réponse "AXK-VALIDER 2" (laquelle a pu manquer à cause de la panne du site) ;

. dans le cas *étatr* = PREVALIDATION 1 ou PREVALIDATION 2, C_t n'est pas sûr que la transaction ait atteint un point de Validation ou non ; alors C_t envoie au Producteur P_t la demande "QETATRP (sitec)" pour lui demander la valeur de sa variable *étatr*, et se met en attente de la réponse correspondante.

- *Réception de la réponse "ACK-QETATRP (étatr-p)* : selon la version de l'état de la transaction pour P_t, communiquée dans le paramètre *étatr-p*, C_t choisit une nouvelle action de récupération ainsi :

. dans le cas *étatr-p* = INITIAL, C_t envoie au Producteur P_t le signal "PANNE-C" lui informant de son isolement temporaire, et réalise une Annulation des intentions de la ZONE en cours ;

. dans les cas *étatr-p* = VALIDATION 1 ou VALIDATION 2, C_t réalise une Validation Différée ou une Validation Permanente, selon le cas, et envoie la réponse correspondante au Producteur P_t.

- *Réception du signal NACK-QETATRP* : ce signal indique le manque de la réponse ACK-QETATRP à cause d'un isolement du Producteur P_t. C_t doit alors consulter les autres Consommateurs de la transaction pour décider de l'action de récupération ; pour cela, C_t passe à son Moniteur la demande "QETATRC (sitec)" destinée à être diffusée à tous les Consommateurs de la transaction, et se met en attente des réponses.

- *Réception de la demande QETATRC (sitec)* : lorsqu'un Consommateur C_t reçoit cette demande provenante du Consommateur du site sitec, C_t envoie à ce deuxième Consommateur la réponse "ACK-QETATRC (étatr, niveau)" contenant ses versions propres sur l'état de la transaction, et sur le niveau d'emboîtement

- *Réception de la réponse ACK-QETATRC (étatrc, ni)* : selon la version de l'état de la transaction pour l'autre consommateur qui envoie la réponse, C_t choisit l'action de récupération à réaliser ainsi :

. dans les cas *étatrc* = VALIDATION 1, VALIDATION 2, ou COMPLET, C_t réalise une Validation Différée ou une Validation Permanente et envoie la réponse correspondante à P_t ;

. dans le cas *étatrc* = INITIAL : si ni < niveau on peut conclure que l'autre Consommateur vient de réaliser une validation de la transaction en cours, et alors C_t réalise à son tour la validation correspondante et envoie la réponse à P_t ; par contre, si ni = niveau, C_t envoie au Producteur P_t le signal "PANNE-C" lui informant de son isolement temporaire, et réalise une Annulation des intentions de la ZONE en cours ;

. dans les cas *étatrc* = PREVALIDATION 1 ou PREVALIDATION 2, C_t continue à attendre d'autres réponses ACK-QETATRC pour décider;

Figure 11 A : Processus Consommateur C_t : traitement des situations d'exception

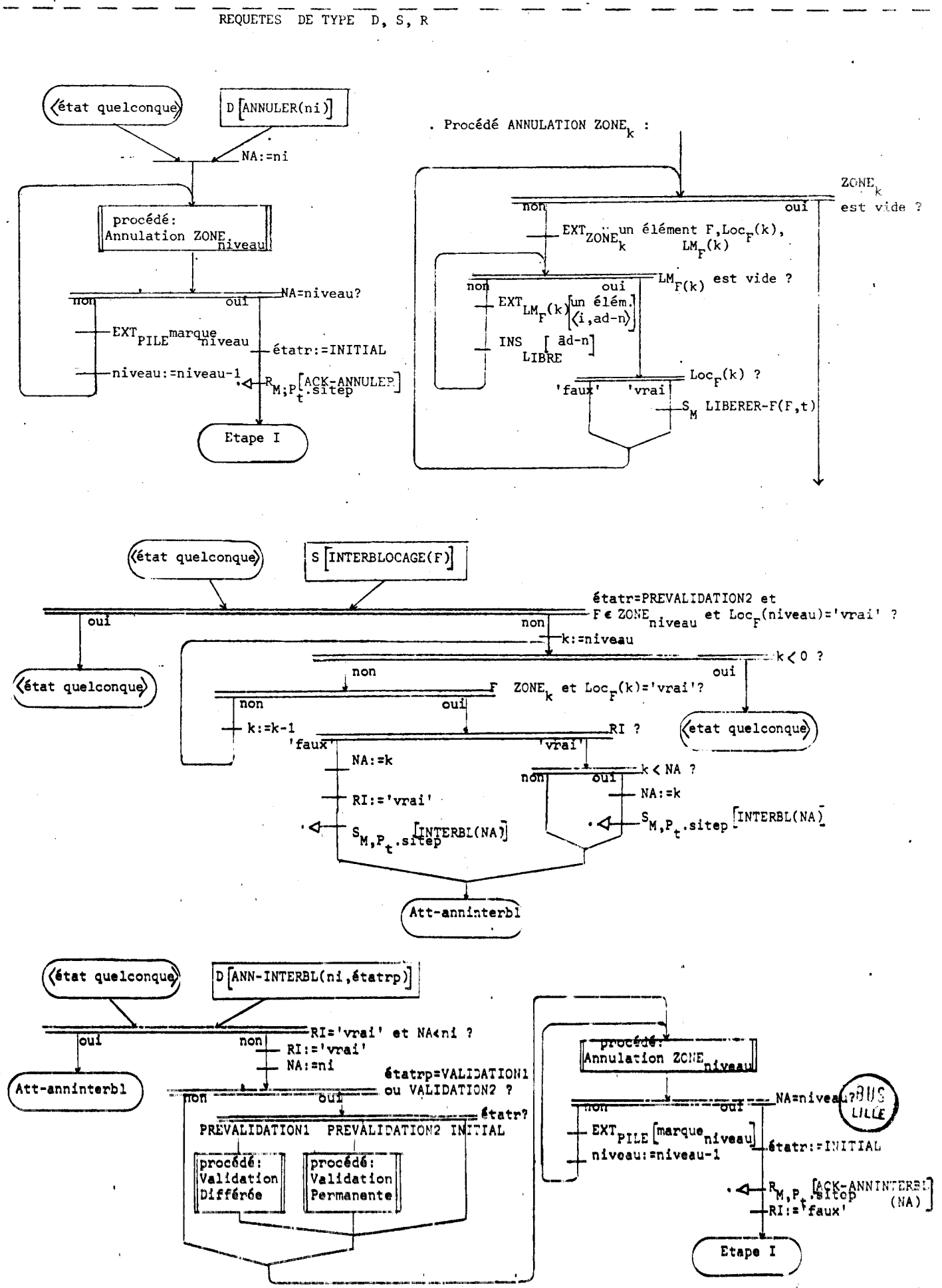
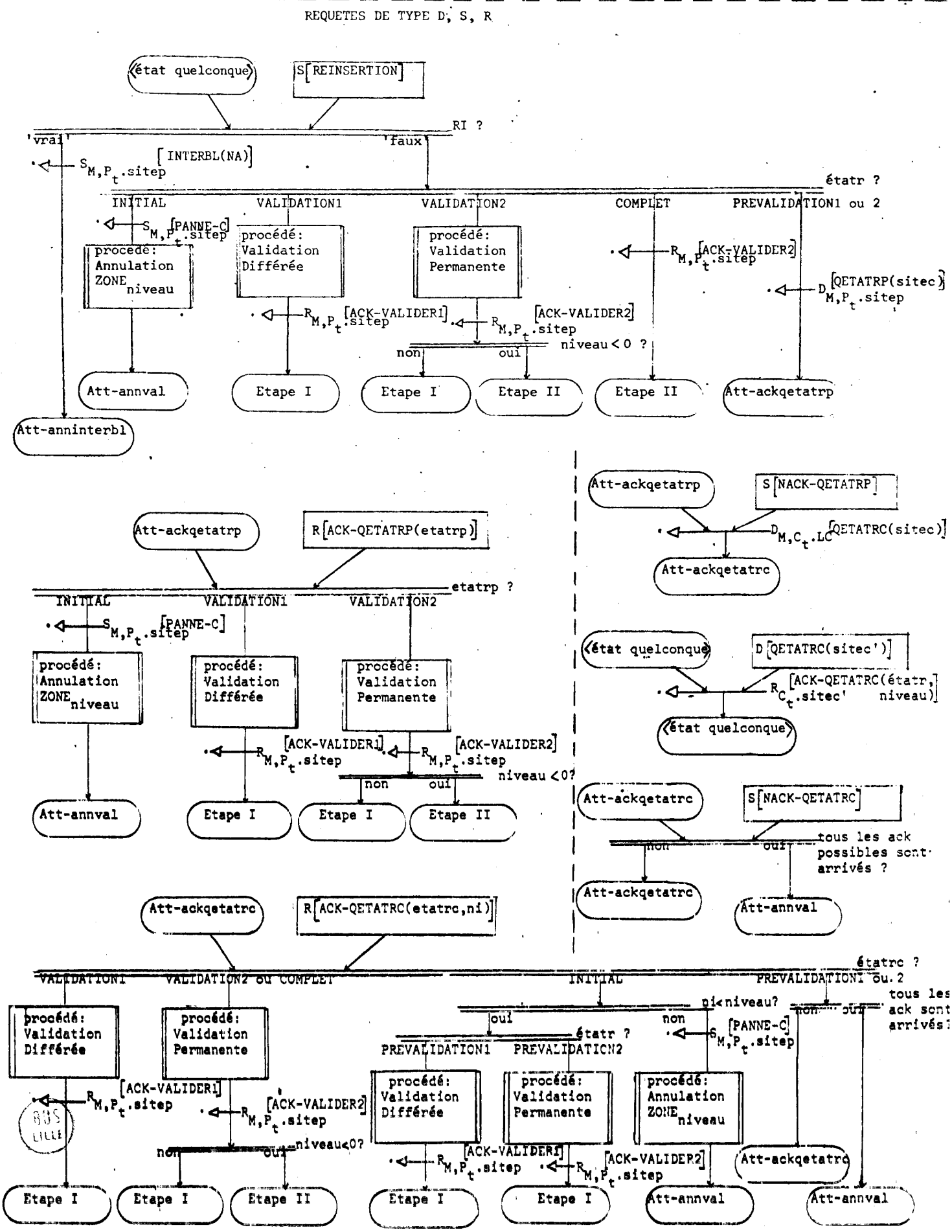


Figure 11 B : Processus Consommateur C_t : traitement des situations d'exception (continuation)



- *Réception du signal* NACK-QETATRC : ce signal indique le manque d'une réponse ACK-QETATRC à cause de l'isolement d'un Consommateur. C_t continue à attendre les réponses des autres Consommateurs pour décider. Si C_t n'a pas pu prendre aucune décision de récupération lorsque toutes les réponses possibles ACK-QETATRC sont arrivées, alors C_t n'a pas d'autre alternative que d'attendre la récupération du Producteur P_t (il se met en attente d'une demande d'annulation ou de validation).

3.5 Processus Moniteur.

Dans son rôle de contrôleur général, le processus Moniteur (M) de chaque site est chargé de traiter toutes les requêtes émises ou destinées aux processus locaux, Producteurs et Consommateurs de diverses transactions, afin d'assurer l'ordonnancement cohérent des transactions dans le système. Ces requêtes traitées par le Moniteur comprennent :

- les demandes et réponses internes, provenantes des processus locaux et destinées à des processus sur d'autres sites ;

- les demandes et réponses externes, provenantes de processus sur d'autres sites et destinées aux processus locaux (délivrées par le sous-système de transport) ;

- les signaux internes et externes, associés à des situations d'exception pour les transactions en cours ;

- exceptionnellement la commande DEBUT-TR d'une nouvelle transaction (toutes les autres commandes d'une transaction sont traitées directement par son Producteur).

3.5.1 *Traitement de la commande* DEBUT-TR.

Avec cette commande, un utilisateur demande l'initialisation d'une nouvelle transaction. Le Moniteur de son site doit alors insérer l'identificateur de l'utilisateur dans sa File de transactions en attente d'identification (FID) :

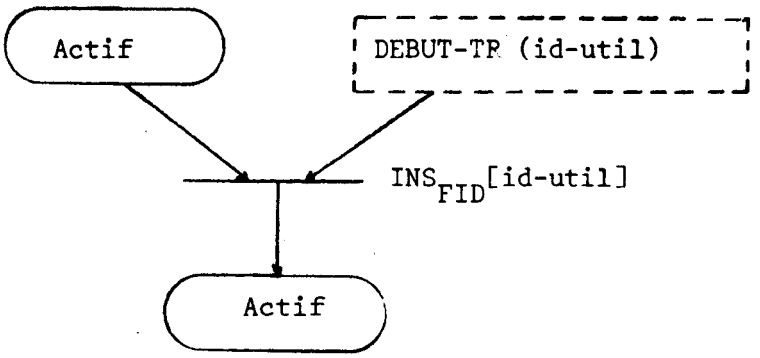


Figure 12 : Processus Moniteur M : traitement de la commande DEBUT-TR

Il faudra attendre le passage du signal "JETON" pour assigner des "tickets" identification à ces nouvelles transactions.

3.5.2 *Traitement de demandes.*

Toute *demande interne* venant des processus locaux d'un site, Producteurs ou Consommateurs, est transmise par leur Moniteur au Moniteur du site destinateur, au moyen du sous-système de Transport. S'il s'agit d'une demande destinée à un ensemble de processus externes, le Moniteur réalise alors une diffusion de la demande vers les sites destinateurs (voir figure 13).

où <dem$_{P1}$> = {DEBUT-TR(paramètres), OUVRIR (par.), ECRIRE (par.), LIRE, LIEC (par.)}

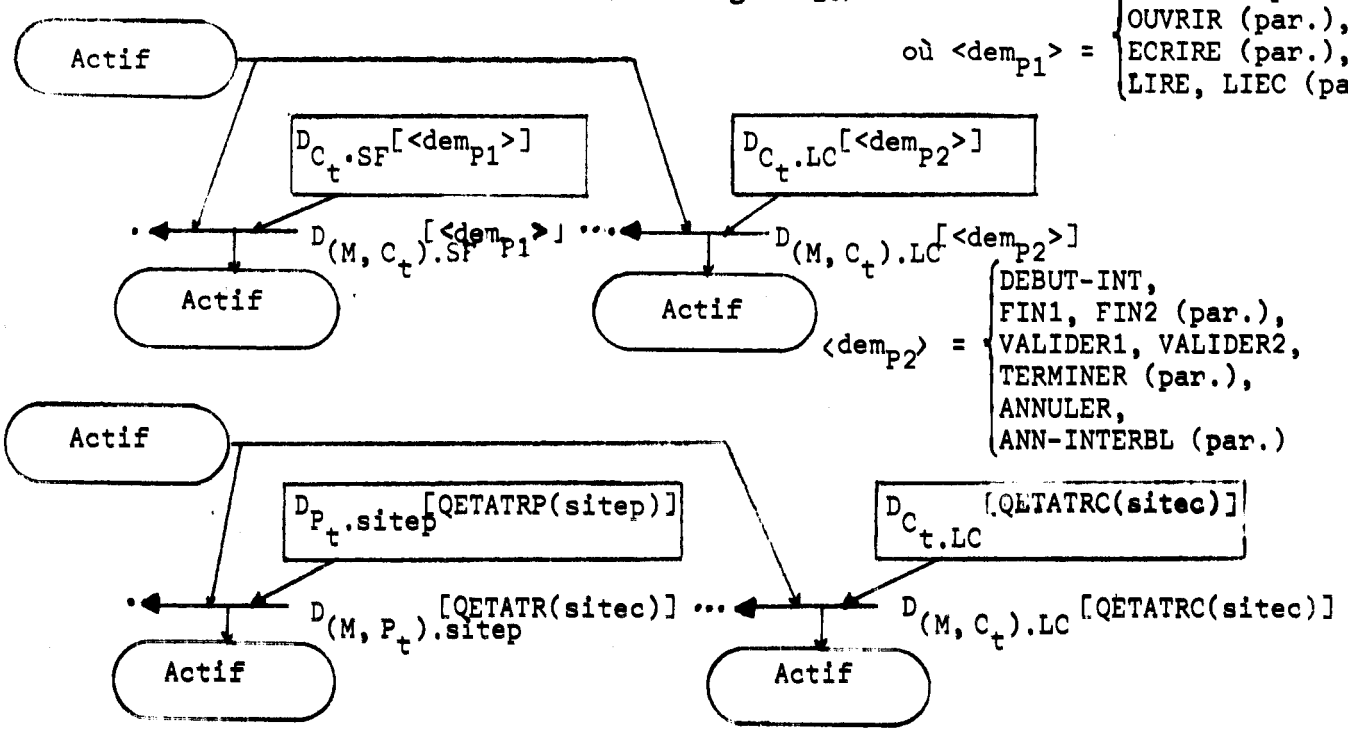


Figure 13 : Processus Moniteur M : traitement de demandes internes.

Le traitement des *demandes externes* délivrées au Moniteur d'un site par le sous-système de transport, est représenté dans les figures 14 (A,B). Ce traitement peut être résumé de la mamière suivante :

- *Réception de la demande externe DEBUT-TR(t, ni, id-sitep, id-sitec) d'une transaction* t : le Moniteur déclenche l'activation d'un processus Consommateur C_t pour la transaction t, et lui délivre la demande. En plus, le Moniteur introduit dans sa table de Consommateurs locaux (TCON) une entrée pour la transaction t (contenant son identificateur t, son indicateur d'activité Act_t égal à "vrai" et sa file d'attente $FATT_t$ initialement vide).

- *Réception de la demande externe OUVRIR (F,v) d'une transaction t* : s'il y a une autre demande de t en exécution ou en attente, ou si la demande présente un conflit de concurrence au niveau du fichier, alors le Moniteur place la demande dans la File d'attente de t ($FATT_t$).
Dans l'autre cas, le Moniteur enregistre dans sa table de verrous sur fichiers (TVF) le verrou v sur F octroyé à la transaction t, délivre la demande au Consommateur concerné C_t et marque l'indicateur Act_t à "vrai".

Le procédé pour évaluer si une demande OUVRIR (F,v) présente un conflit de concurrence au niveau de fichier ou non, est le suivant : on examine toutes les entrées de la Table de Verrous TVF référentes aux verrous établis sur F par diverses transactions ; si un de ces verrous v' est incompatible avec le verrou demandé v, alors il y a un conflit de concurrence ; si en plus, ce verrou v' appartient à une transaction t' plus jeune que t (i.e. son identificateur est plus grand que celui de t), il y a un risque d'interblocage et dans ce cas le Moniteur envoie un signal "INTERBLOCAGE (F)" au Consommateur Ct'.

- *Réception des demandes externes LIRE (F, i, b), LIEC (F, i, b) ou ECRIRE (F, i, (données)) d'une transaction t* : s'il y a une autre demande de t en exécution ou en attente, le Moniteur place la demande dans la File d'attente de t ($FATT_t$). Dans l'autre cas, si la demande n'est pas valable par rapport au verrou que t possède sur le fichier F, alors elle est ignorée (c'est à dire lorsque la demande est LIEC ou ECRIRE et que le verrou sur le fichier est IS ou S ; en fait, il devrait y avoir ici un traitement d'erreur, impliquant un message spécial pour l'utilisateur, que nous ne détaillons pas). Si la demande est valable et ne requiert pas de verrou explicite sur le segment i de F, alors elle est délivrée au Consommateur concerné C_t ; par contre, si la demande requiert un verrou explicite sur le segment i de F (i.e. lorsque F se trouve verrouillé

en un mode intention), le Moniteur doit évaluer s'il y a un conflit de concurrence au niveau du segment : en cas de non-conflit, le verrou explicite requis est enregistré dans la Table de verrous sur F établis par t (TVS $_{F.t}$) et la demande est délivrée au Consommateur C_t ; enfin, en cas de conflit elle est placée dans la File d'attente de t.

Le procédé pour évaluer si la demande en question présente un conflit de concurrence au niveau de segment est le suivant : on examine toutes les entrées de la Table de Verrous TVF référentes aux verrous établis sur F par d'autres transactions différentes de t. Si une de ces transactions, t', possède un verrou vs' sur le segment i qui est incompatible avec le verrou vs requis par t (vs = S si la demande est LIRE, et vs = X si la demande est LIEC ou ECRIRE), alors il y a un conflit de concurrence ; si en plus t' est plus jeune que t, il y a un risque d'interblocage et le Moniteur envoie dans ce cas un signal "INTERBLOCAGE (F)" au Consommateur $C_{t'}$.

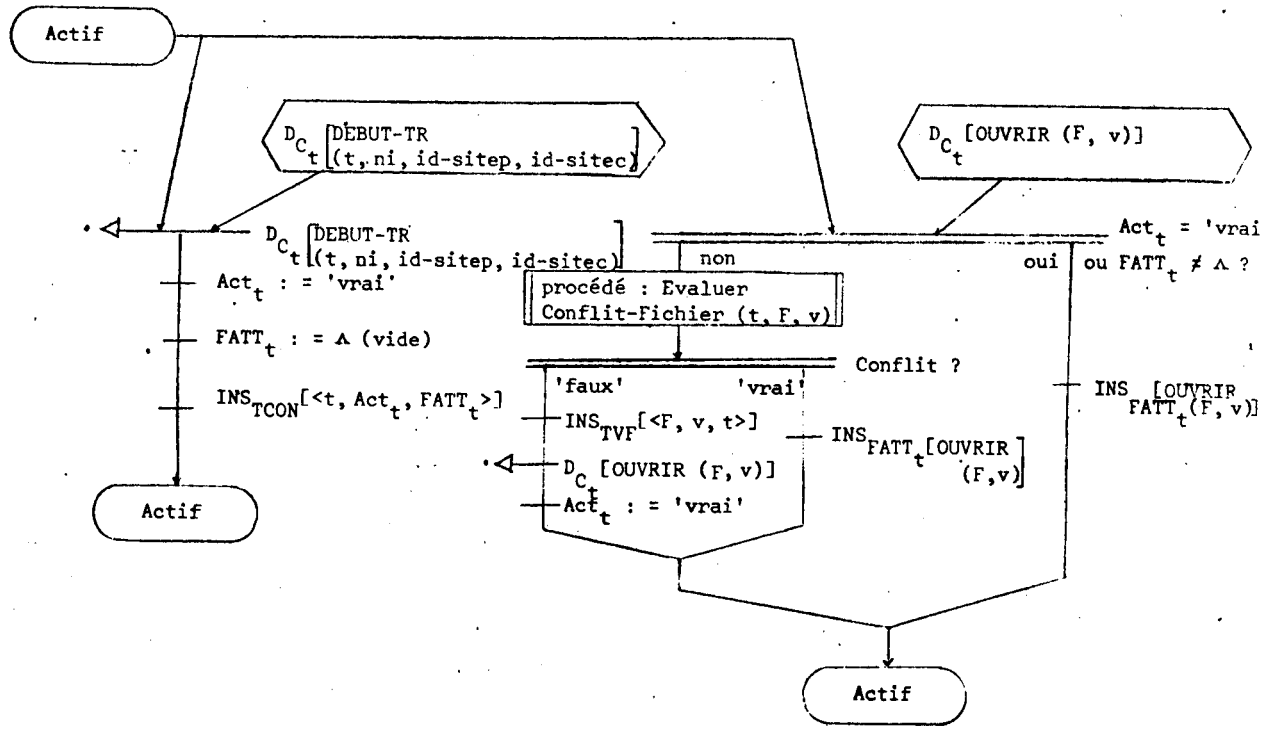
- *Réception des demandes externes DEBUT-INT (ni), FIN 1(ni s1, s2,...,sm), FIN2 (ni, s1, s2,...,sm), VALIDER 1 ou VALIDER 2 d'une transaction t* : s'il y a une suite demande de t en exécution ou en attente, le Moniteur place la demande dans la File d'attente de t ($FATT_t$) ; dans l'autre cas, la demande est délivrée au Consommateur concerné C_t (et l'indicateur Act_t est marqué à "vrai").

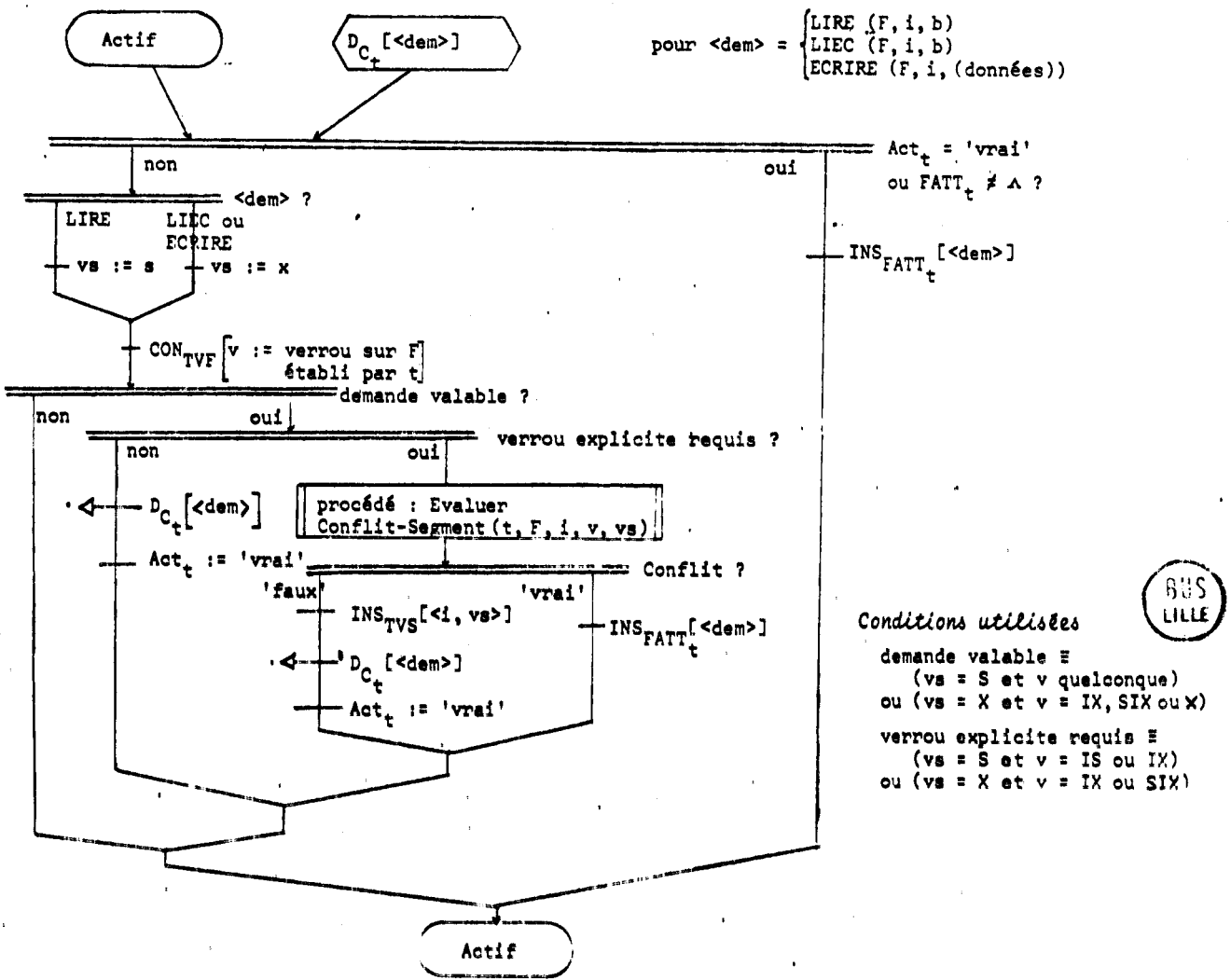
- *Réception de la demande externe TERMINER (sitep) d'une transaction t* : la demande est délivrée au Consommateur concerné C_t si ce processus est encore actif (i.e. la transaction t figure dans la Table de Consommateurs locaux, TCON) ; dans l'autre cas, le Moniteur transmet directement la réponse "ACK-TERMINER" au site Producteur de la transaction t.

- *Réception des demandes externes ANNULER (ni) ou ANN-INTERBL (ni, étatrp) d'une transaction t* : le Moniteur élimine toutes les demandes qui se trouvent dans la File d'attente de t ($FATT_t$), et délivre la demande arrivée au Consommateur concerné C_t.

- *Réception des demandes externes QETATRP (sitec') ou QETATRC (sitec') d'une transaction t* : la demande est délivrée directement au processus concerné : P_t dans le cas de QETATR P, ou C_t dans le cas de QETRATRC.

Figure 14 A : Processus Moniteur M : traitement de demandes externes





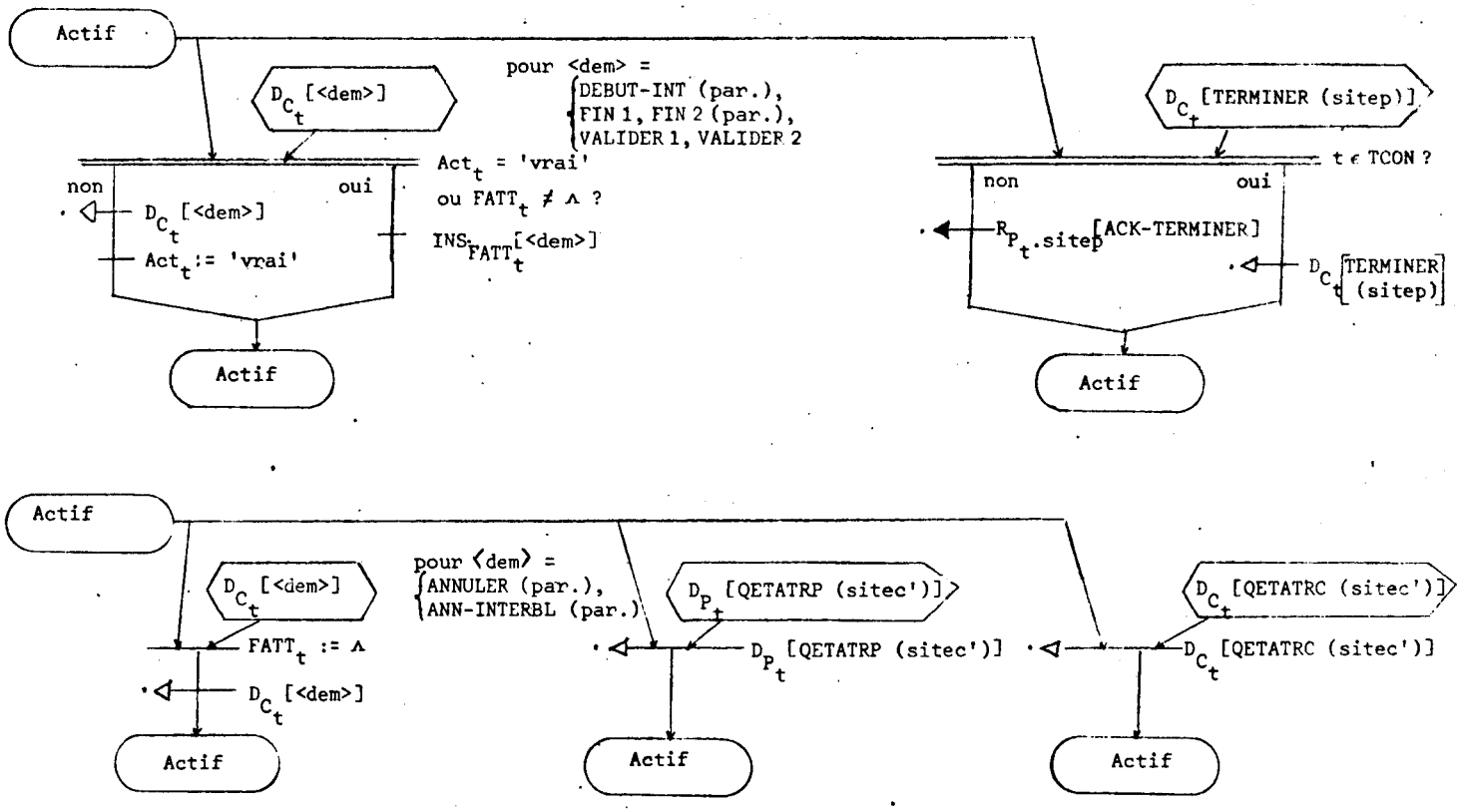
Conditions utilisées

demande valable ≡
(vs = S et v quelconque)
ou (vs = X et v = IX, SIX ou X)

verrou explicite requis ≡
(vs = S et v = IS ou IX)
ou (vs = X et v = IX ou SIX)

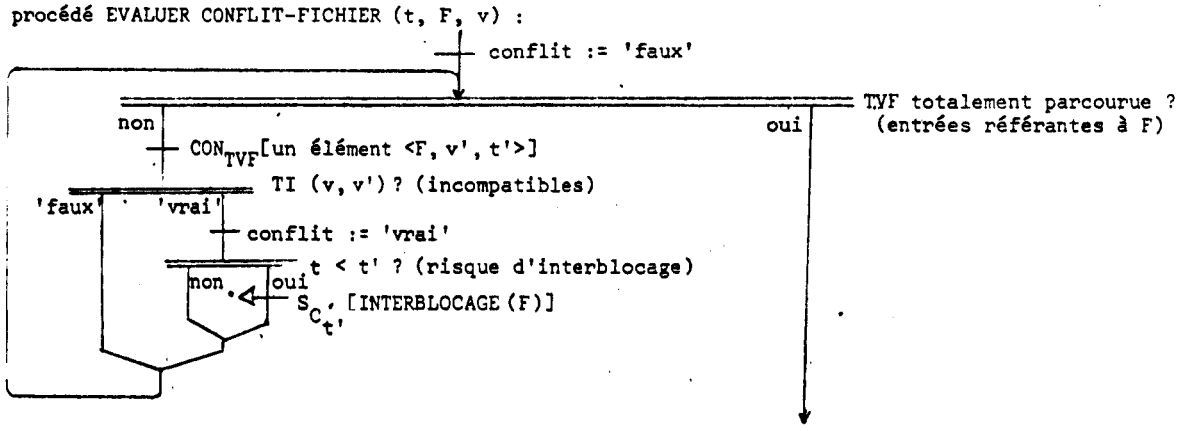
BUS LILLE

Figure 14 B : Processus Moniteur M : traitement de demandes externes (continuation)

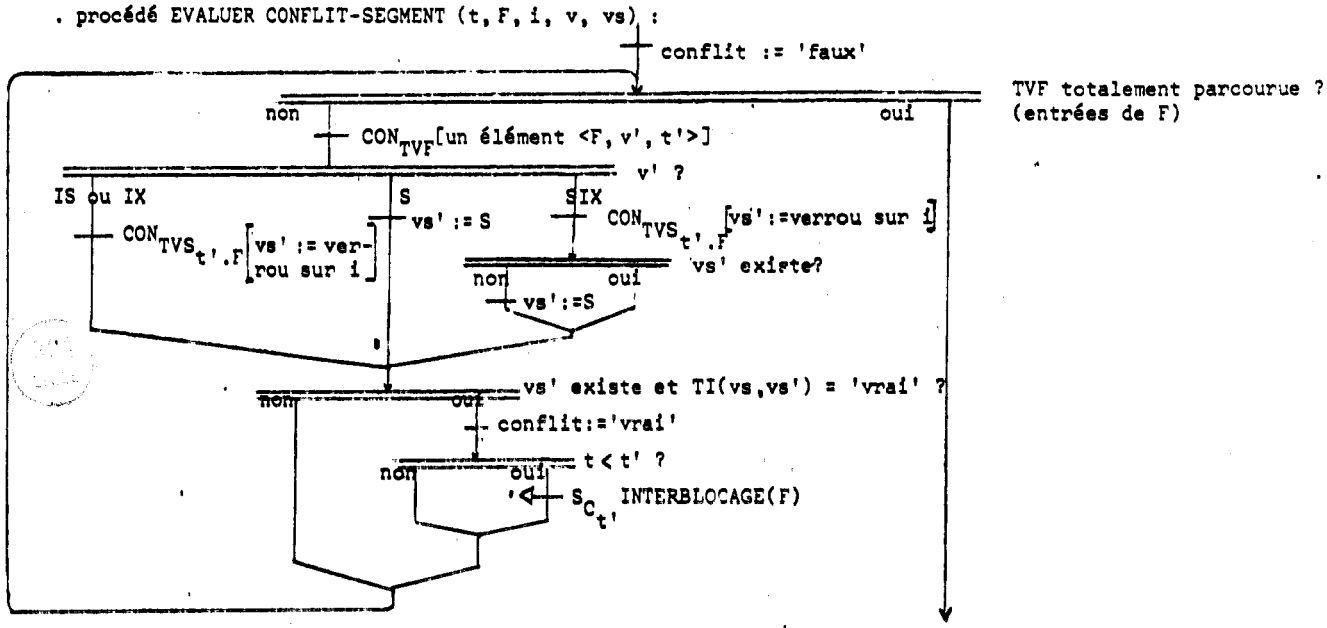


PROCEDES UTILISES :

. procédé EVALUER CONFLIT-FICHIER (t, F, v) :



. procédé EVALUER CONFLIT-SEGMENT (t, F, i, v, vs) :



3.5.3 *Traitement de réponses.*

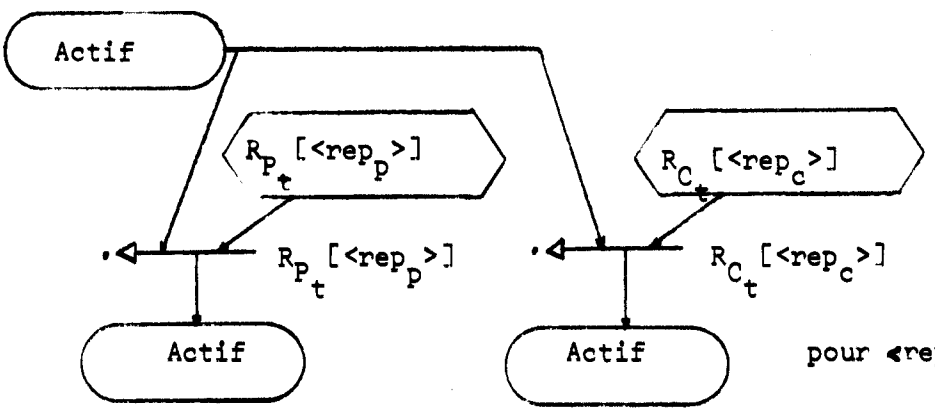
Toute *réponse interne* produite par les processus Consommateurs d'un site S, est transmise par leur Moniteur au Moniteur du site destinateur, au moyen du Sous-système de transport.
En plus, le Moniteur du site S examine la demande en tête de la File d'attente de la transaction qui a produit la réponse : selon la demande, le Moniteur applique le procédé qui correspond parmi ceux que nous venons de spécifier dans la section précédante (i.e. traitement de demandes externes) ; si la demande n'entraîne aucun conflit de concurrence, elle est délivrée au Consommateur concerné et enlevée de la File d'attente.

Dans le cas d'une réponse interne ACK-TERMINER d'une transaction t, le Moniteur transmet la réponse au site du Producteur P_t, et élimine les informations de t dans sa Table de Consommateurs locaux (TCON) : en effet, il s'agit ici de la fin du processus Consommateur C_t.

Dans le cas d'une réponse interne ACK-QETATRP (paramètres) ou ACK-QETRATC (paramètres), le Moniteur transmet simplement cette réponse au site destinateur (i.e. le site d'un processus Consommateur).

Le traitement complet de réponses internes par le Moniteur est représenté dans la figure 16.

Par rapport au traitement de *réponses externes* délivrées au Moniteur d'un site par le sous-système de transport, elles sont simplement délivrées aux processus locaux destinateurs (Producteurs ou Consommateurs) ; ce traitement est représenté dans la Figure 15 suivante :

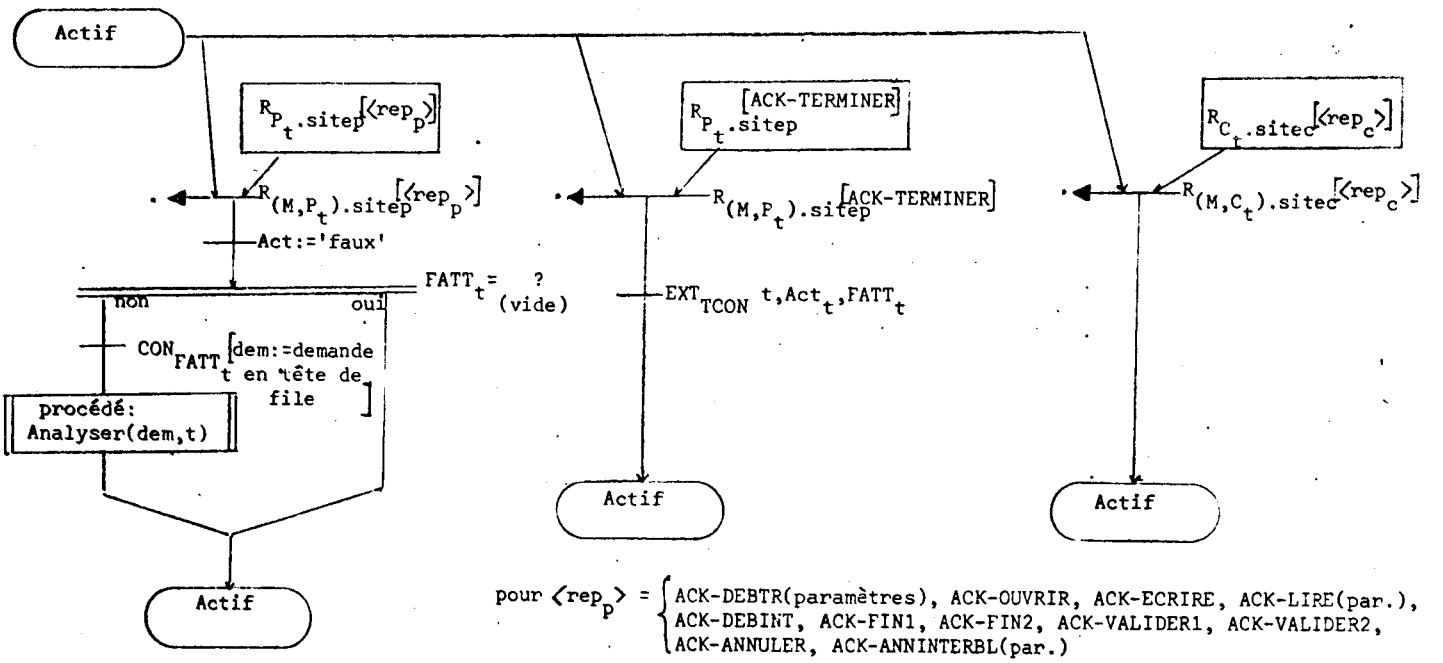


pour <rep$_p$> = { ACK-DEBTR (par.), ACK-OUVRIR, ACK-ECRIRE, ACK-LIRE (par.), ACK-DEBINT, ACK-FIN1, ACK-FIN2, ACK-VALIDER1, ACK-VALIDER2, ACK-TERMINER, ACK-ANNULER, ACK-ANNINTERBL (par.) }

<rep$_c$> = { ACK-QETATRP (par.), ACK-QETATRC (par.) }

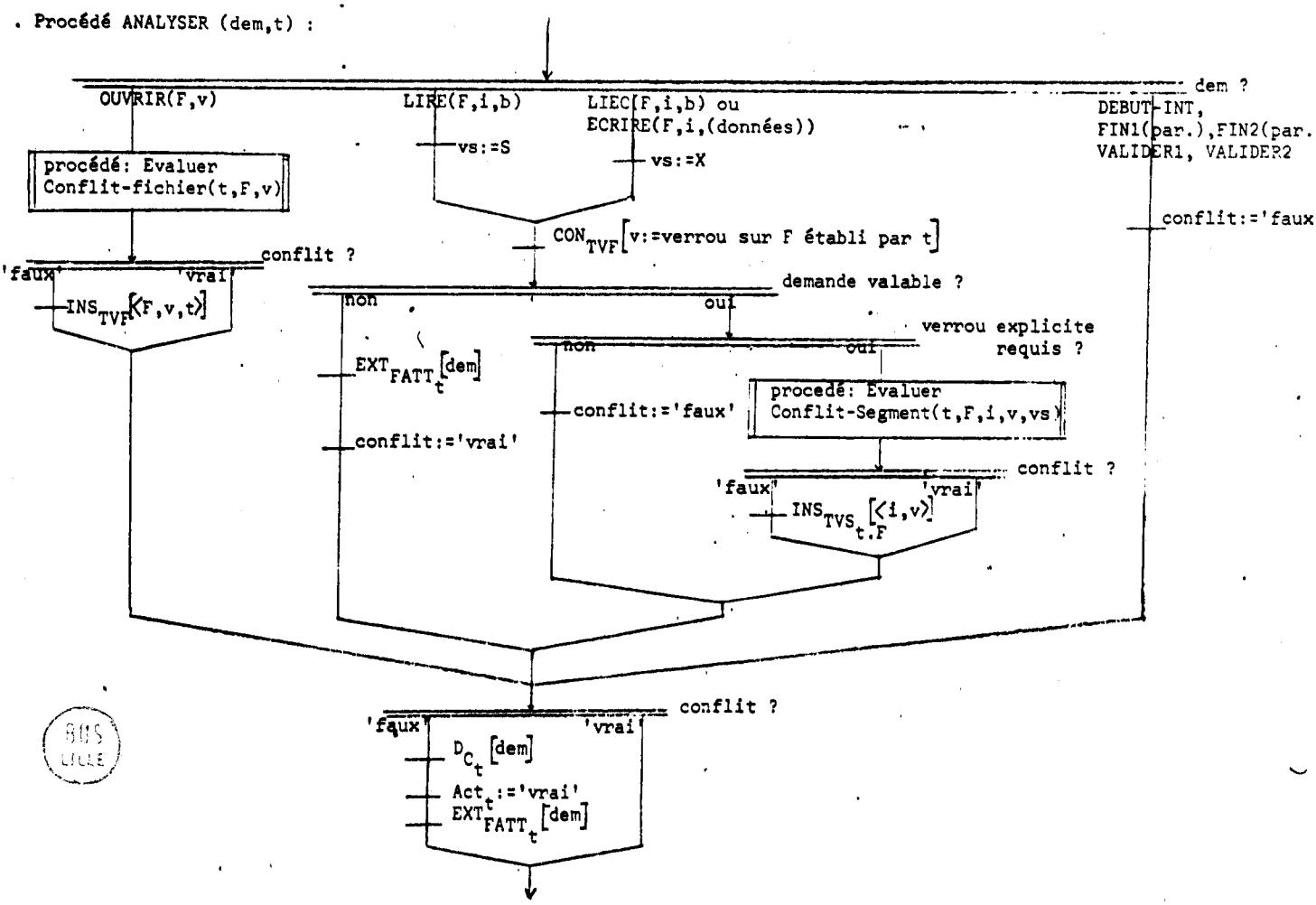
Figure 15 : Processus Moniteur M : traitement de réponses externes.

Figure 16 : Processus Moniteur M : traitement de réponses internes



pour ⟨rep_p⟩ = {ACK-DEBTR(paramètres), ACK-OUVRIR, ACK-ECRIRE, ACK-LIRE(par.), ACK-DEBINT, ACK-FIN1, ACK-FIN2, ACK-VALIDER1, ACK-VALIDER2, ACK-ANNULER, ACK-ANNINTERBL(par.)

pour rep_c = {ACK-QETATRP(par.), ACK-QETATRC(par.)



BUS LILLE

Conditions utilisées:

demande valable ≡ (vs=S et v quelconque) ou (vs=X et v=IX, SIX ou X)

verrou explicite requis ≡ (vs=S et v=IS ou IX) ou (vs=X et v=IX ou SIX)

3.5.4 *Traitement de signaux.*

Les *signaux internes* INTERBL (ni), et PANNE-C produits par un Consommateur local C_t, sont transmis par son moniteur au site destinataire (i.e le site du Producteur P_t).

La réception des signaux internes LIBERER-F (F,t) ou LIBERER-S (F, t, i), produits par le consommateur local C_t, déclenche chez le Moniteur le traitement suivant pour résoudre les attentes de verrous : d'abord : le verrou concerné, établi par la transaction t, est enlevé de la Table de verrous TVF : pour le signal LIBERER-F, c'est le verrou global sur le fichier F pour le signal LIBERER-S, c'est le verrou sur le segment i de F. Ensuite, le Moniteur examine chaque transaction t' (t' ≠ t) figurant dans la Table de Consommateurs locaux (TCON) (par ordre ascendant d'identificateur) : si la demande en tête de la File d'attente de t' ($FATT_{t'}$) est en attente d'un verrou sur le fichier F ou sur un de ses segments, et si ce verrou n'entraîne plus de conflit de concurrence, alors le Moniteur octroie le verrou à t' et délivre la demande au Consommateur concerné $C_{t'}$. Le traitement de signaux internes par le Moniteur est représenté dans la figure 17.

Le traitement de *signaux externes* délivrés au Moniteur d'un site par le Sous-système de Transport est représenté dans la figure 18, et peut être résumé ainsi : les signaux externes INTERBL (ni), PANNE-C, NACK, NACK-QETATRP, ou NACK-QETATRC référents à une transaction t, sont délivrés au processus local destinateur (i.e. P_t pour les 3 premiers, C_t pour les 2 derniers). Le signal REINSERTION signifie la réinsertion du site local après une défaillance : le Moniteur envoie alors le signal à tous les processus locaux, Producteurs ou Consommateurs. Le signal JETON (ticket) déclenche un traitement d'identification des nouvelles transactions : pour chaque identificateur d'utilisateur figurant dans la File de transactions en attente d'identification (FID), le Moniteur assigne la valeur courante du ticket, disons t, comme identificateur de la nouvelle transaction, active le processus Producteur P_t en lui envoyant le signal DEBUT-TR (paramètres), introduit l'identificateur t dans la Table de Producteurs locaux (TPRO), et incrémente le ticket ; à la fin de ce traitement, le Moniteur transmet au Moniteur du site successeur le signal JETON (ticket).

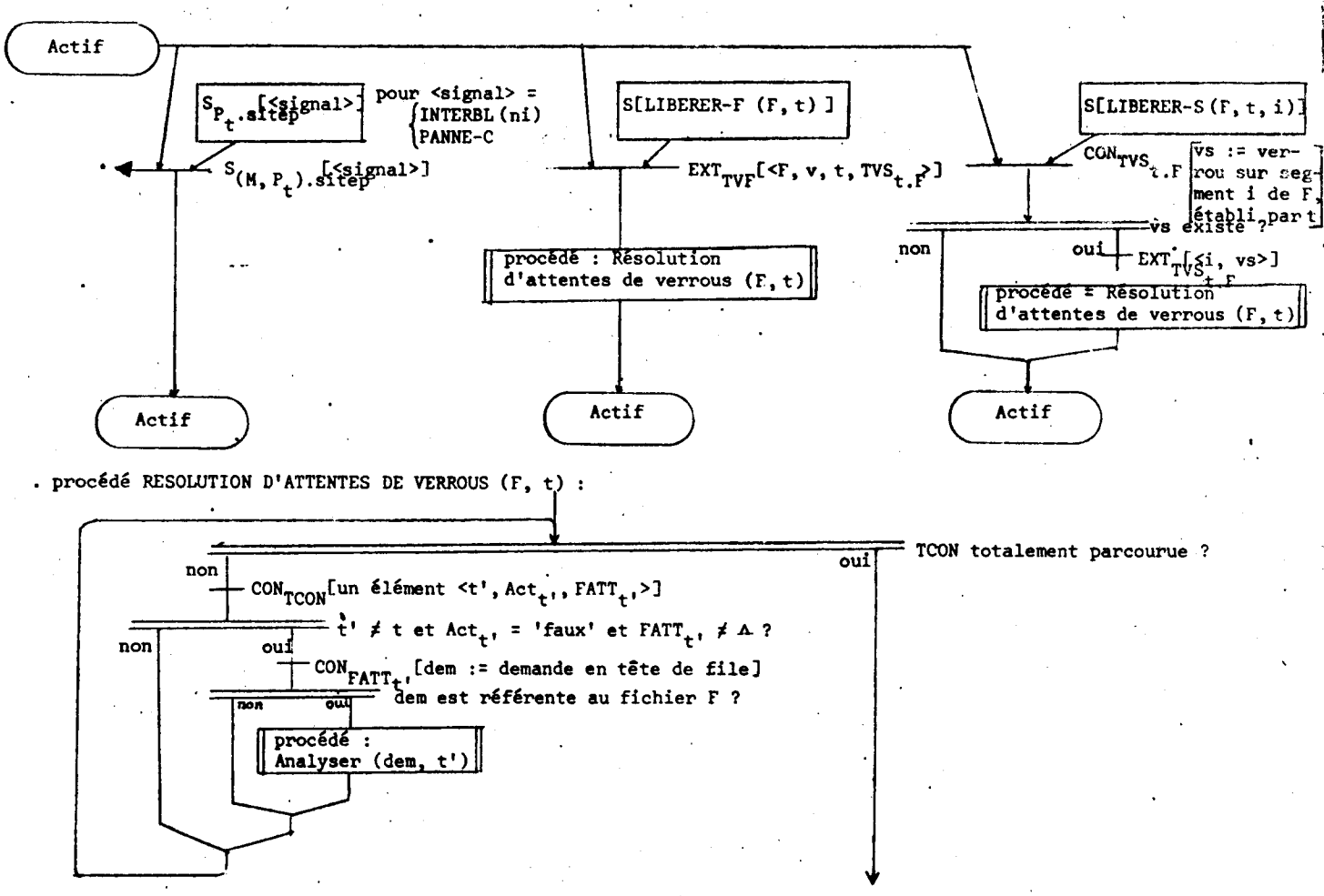


Figure 17 : Processus Moniteur M : traitement de signaux internes

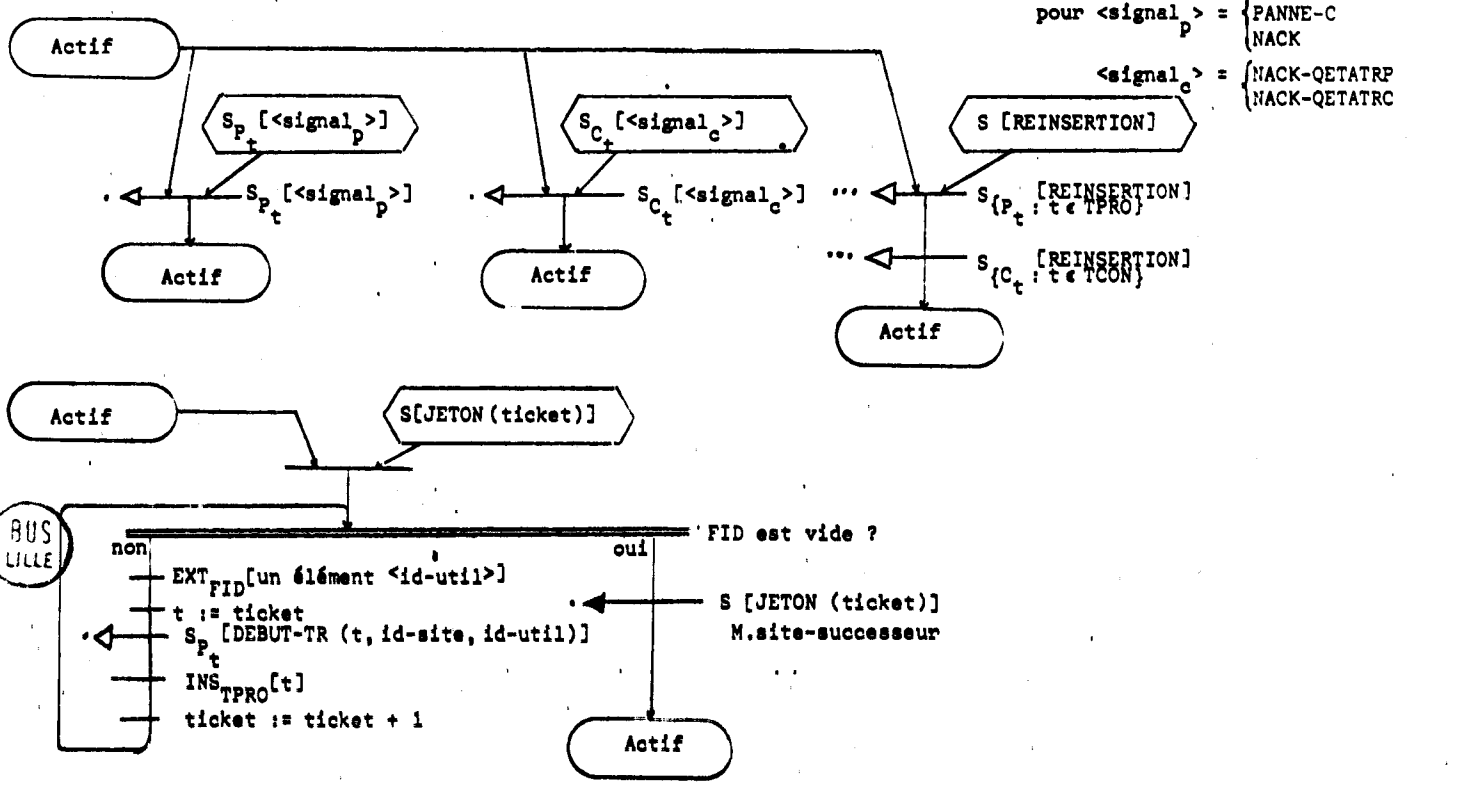


Figure 18 : Processus Moniteur M : traitement de signaux externes

4 - RECAPITULATION DU MODELE DE TRANSACTIONS EMBOITEES A VERROUILLAGE SELECTIF.

Nous venons de présenter dans ce chapitre un Modèle de contrôle de cohérence, pour un système réparti de fichiers, lequel se base sur la structuration des transactions en transactions emboîtées et sur l'utilisation des verrous hiérarchiques. Ce modèle vérifie les propriétés demandées à tout protocole de contrôle de cohérence (énoncées dans le chapitre I, partie 2.4), à savoir :

- *Déterminisme* : le modèle préserve la cohérence des données du système lors de concurrence de transactions et lors de défaillances (d'après les justifications données dans la partie 2.4 de ce chapitre).

- *Convergence et impartialité* : les conflits d'accès aux données sont réglés en évitant les situations d'interblocage et de famine, et sans favoriser aucune transaction (en fait, chaque transaction à son tour, devient la plus prioritaire lorsqu'elle est la plus ancienne).

- *Résilience* : le modèle assure la récupération de toute transaction en cours après une défaillance dans le système.

- *Performance* : le modèle favorise la concurrence réelle entre transactions tout en assurant un temps de réponse acceptable pour l'utilisateur interactif.

Par rapport au Modèle de Transactions atomiques (présenté dans le chapitre III), le Modèle de transactions emboîtées à verrouillage sélectif présente les avantages suivantes :

- *Concurrence réelle élevée et diminution du temps de réponse* : d'une part, l'utilisation des verrous hiérarchiques (aux niveaux de fichier et de segment) favorisent la concurrence réelle entre transactions sans dégrader le temps de réponse donné aux utilisateurs ; d'autre part, la libération graduelle de données pendant la réalisation de chaque transaction évite aux autres les attentes trop longues pour l'accès à ces données. En fait, chaque donnée utilisée dans une transaction est libérée au moment où l'utilisateur considère qu'il n'en aura plus besoin ; cela n'empêche à l'utilisateur de demander à nouveau une donnée libérée préalablement, mais au prix d'une nouvelle attente pour obtenir le verrou.

- *Annulation non totale des transactions lors de défaillances :* pour une transaction, la réalisation de chacune de ces transactions internes utilise un mécanisme de validation en 2 étapes ; ceci permet sa récupération lors de défaillances du système par achèvement ou par annulation de la transaction interne en cours, et non de toute la transaction. L'utilisateur est ainsi épargné de perdre toutes ses opérations lors de défaillances.
De même, la prévention des situations d'interblocage peut entraîner une annulation partielle mais non totale des transactions affectées.

- *Possibilités pour l'utilisateur de demander le retour de sa transaction à un état antérieur :* les points de sauvegarde d'une transaction, associés aux débuts des transactions internes, permettent à l'utilisateur de demander par la suite le retour de la transaction à un de ces points (en fait, pour retourner au début d'une transaction interne quelconque, l'utilisateur doit d'abord finaliser, avec la commande "FIN1", les transactions plus internes qui sont en cours, et ensuite demander l'annulation de la transaction interne qui l'intéresse). Etant donné qu'il y a libération de certaines données à la fin de chaque unité de cohérence de la transaction (unités délimitées par les commandes "FIN2"), les retours permis à l'utilisateur sont limités à l'unité de cohérence en cours afin d'éviter l'effet Domino.

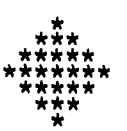
- *Possibilité pour l'utilisateur de défaire ou de modifier les traitements déjà effectués :* les fichiers globaux utilisés par une transaction restent réservés d'une unité de cohérence à la suivante ; cela permet à l'utilisateur, dans l'unité de cohérence en cours, de défaire ou de modifier les traitements qui ont été effectués sur ces fichiers globaux dans l'unité antérieure (i.e. en faisant une sorte de transaction inverse).
De même, l'utilisateur peut modifier les traitements qui ont été effectués sur des fichiers déjà libérés, mais dans ce cas il doit obtenir à nouveau les verrous sur ces fichiers.

En conclusion, le Modèle de transactions emboîtées à verrouillage sélectif favorise l'utilisateur interactif en lui offrant des facilités et des options de contrôle.
Finalement, c'est l'utilisateur qui contrôle le déroulement de sa transaction :

ainsi par exemple, en structurant judicieusement la transaction en transactions internes, l'utilisateur peut préserver les traitements déjà effectués des annulations déclenchées par le système ; de même, un utilisateur peut se protéger des annulations causées par les risques d'interblocage, en réalisant les accès aux fichiers les plus souvent demandés par la communité des utilisateurs, dans les transactions les plus internes.

CHAPITRE V

MODELE GENERALISE DE CONTROLE DE COHERENCE DANS UN SYSTEME REPARTI A UTILISATION INTERACTIVE : OBJETS, OPERATIONS ET VERROUS GENERALISES



Dans ce dernier chapitre, nous voulons généraliser le modèle de contrôle de cohérence présenté dans le chapitre IV pour l'appliquer à un système réparti général, c'est à dire un système réparti où les utilisateurs réalisent diverses opérations sur des objets de différents types, et pas seulement de type fichier.

Cette généralisation va permettre d'étendre le domaine d'application des résultats obtenus dans ce travail. En particulier, une transaction n'entraînera plus le transfert systématique des *données* concernées par ses opérations des sites Consommateurs vers le site Producteur et vice-versa (c'était l'effet des commandes LIRE et ECRIRE du modèle jusqu'à maintenant considéré) ; on peut dire plutôt que dans le modèle généralisé ce seront les *opérations* de la transaction qui seront envoyées aux sites Consommateurs où les données résident (i.e. schéma de données statiques), ce qui est plus approprié pour les applications réelles où les données manipulées peuvent être très volumineuses.

Dans la première partie de ce chapitre nous allons préciser les concepts d'objets, d'opérations et de verrous généralisés qui seront utilisés pour définir un type général de transactions emboîtées. Ensuite, nous allons décrire la mise en oeuvre de transactions emboîtées concernant des objets qui résident sur divers sites d'un système réparti (en fait, cette mise en oeuvre sera une généralisation des processus Producteur, Consommateur et Moniteur présentés dans le chapitre IV). Finalement, pour le Modèle généralisé de contrôle de cohérence qui résulte, nous allons proposer les lignes générales d'une implémentation dans le langage de programmation ADA, en utilisant un exemple d'application spécifique.

1. OBJETS, OPERATIONS ET VERROUS GENERALISES.

1.1 Concepts d'objets et d'opérations généralisés.

Nous considérons à présent un système informatique réparti où les données composent divers *objets* qui sont à la disposition des utilisateurs. Pour manipuler les objets du système, les utilisateurs réalisent des transactions emboîtées interactives où les commandes contiennent des *opérations* à réaliser sur ces objets. Un exemple de transaction emboîtée avec opérations et objets généralisés est le suivant :

```
DEBUT-TR
   OP1 (o1 ..., <d'autres paramètres>)
   OP2 (o2 ..., <d'autres paramètres>)

   DEBUT-INT
      OP1 (o3 ..., <d'autres paramètres>)
      ... d'autres opérations sur o1, o2, o3

      DEBUT-INT
         OP3 (o4 ..., <d'autres paramètres>)
         ... d'autres opérations sur o1, o2, o3, o4

      FIN 1

      OP4 (o3 ..., <d'autres paramètres>)

   FIN 2---------------------------------------- point de cohérence

   DEBUT-INT
         OP3 (o5 ..., <d'autres paramètres>)
         ... d'autres opérations sur o1, o2, o5

   FIN 1
   OP2(o2..., <d'autres paramètres> )
FIN 2
```

Comme dans le chapitre IV, nous supposons ici qu'une transaction emboîtée est composée de plusieurs unités de cohérence, délimitées par les commandes FIN2, où chacune préserve la cohérence des objets manipulés (le problème de vérification des contraintes d'intégrité du système n'est donc pas considéré). Nous supposons aussi qu'il n'y a pas de duplication des objets du système : chaque objet réside sur un seul site du système (i.e. données partitionnées).

En adoptant la notion de *type abstrait* définie par Liskov et Zilles [LISKO 74], nous considérons que les objets du système sont divisés en classes d'équivalence d'après leurs types : un type définit alors une classe d'objets laquelle est caractérisée complètement par les opérations disponibles sur ces objets. Pour l'utilisateur, le seul moyen de manipuler un objet est en utilisant

les opérations de son type, et pour cela, l'utilisateur doit connaître la *spécification* d'un type d'objets laquelle décrit le comportement des objets de ce type en termes des opérations disponibles pour les manipuler ; par contre, l'utilisateur ne doit pas connaître l'*implémentation* d'un type , c'est à dire la représentation exacte des objets et la définition spécifique de chaque opération.

1.1.1 Spécification d'un type d'objets.

Nous allons considérer les objets du système généralement *structurés*, c'est à dire composés d'autres objets. Dans ce cas, la spécification d'un type W d'objets doit déclarer :

- les types des objets composants d'un objet de type W ;
- les opérations disponibles sur les objets de type W.

La forme générale d'une spécification d'un type W d'objets est alors la suivante :

```
type W est
    composants
        id1 : type du 1er composant
        id2 : type du 2ème composant
        ...
        idn : type du nème composant
    fin composants
    opérations
        OP1 (<liste de paramètres>)
        OP2 (<liste de paramètres>)
        ...
        OPK (<liste de paramètres>)
    fin opérations
Fin W
```

Dans la spécification précédente, les n composants d'un objet de type W sont identifiés par des identificateurs id_1, id_2, ..., id_n. Le type d'un composant id_j peut être un type élémentaire ou un type structuré ; dans le dernier cas il y aura une déclaration des types des sous-composants de la forme suivante :

id_j : composants

| id_{j1} : type du 1er sous-composant
| id_{j2} : type du 2ème sous-composant
| ...
| id_{jm} : type du nème sous-composant

Fin composants

L'implémentation des opérations d'un type d'objets doit être connue seulement par les sites du système qui vont contenir des objets de ce type ; un répertoire global dupliqué sur chaque site permettra de référencer tous les objets du système en indiquant pour chacun son nom unique, son type et son site de résidence.

1.1.2 Forme générale d'une opération demandée par l'utilisateur.

Pour demander une opération OP sur un objet σ de type W, l'utilisateur se sert de la commande générale suivante (où on suppose que le premier paramètre est l'objet σ) : OP (< liste de paramètres effectifs>)
où chaque <paramètre effectif> :: <par. effectif-objet du système> /
<par. effectif-valeur>

Un paramètre effectif d'une opération peut être alors de 2 sortes :

- *Un objet du système* : dans ce cas, l'utilisateur spécifie le nom unique de l'objet qui sert de paramètre :
  <par. effectif-objet du système> :: <nom>
  Le premier paramètre d'une opération OP doit être toujours un objet du type W auquel appartient l'opération.

- *une valeur* : dans ce cas, le paramètre est une valeur que l'utilisateur fournit, ou qu'il attend comme réponse :
  <par. effectif-valeur> :: <constante> / <identificateur> = ?

Un exemple d'une opération OP_i sur un objet σ_1 de type W, demandée par l'utilisateur, est le suivant :

$OP_i(\sigma_1, \sigma_2, 1, B= ?)$.

1.1.3 Spécification d'une opération.

Nous voulons à présent donner à l'utilisateur une transparence totale à la gestion de verrous réalisée par le système : l'utilisateur ne spécifiera plus les verrous à établir sur les objets qu'il manipule et cette tâche sera laissée à la charge du système (i.e. la commande OUVRIR ne sera plus considérée). Pour permettre au système d'assurer la gestion de verrous, la spécification d'un type d'objets doit informer pour chaque opération : les types des paramètres et le mode du verrou à établir sur chaque objet concerné par l'opération. La forme générale de la spécification d'une opération OP sur un type W d'objets est alors la suivante :

OP (< liste de paramètres formels>)

où chaque <paramètre formel> :: <par. formel-objet du système> /< par. formel-valeur>

Un paramètre formel se référant à un *objet du système* présente la forme suivante :

<par. formel-objet du système> :: <identificateur>:<classe> <type>
[.<chaîne d'identif. d'un composant>] <verrou>

où <type> spécifie le type de l'objet attendu en paramètre (pour le premier paramètre de OP ce type doit être W) ;

<chaîne d'identif. d'un composant> indique quel (sous) composant de l'objet intervient dans l'opération (optionnel) ;

<classe> spécifie le mode d'utilisation de l'objet dans l'opération : consultation, modification, ou consultation et modification:<classe> :: in/out/in-out ;

<verrou> spécifie le mode du verrou que le système doit établir sur l'objet avant de réaliser l'opération.

Un paramètre formel se référant à une *valeur* présente la forme suivante :

<par. formel-valeur> :: <identificateur>:<classe> <type élémentaire>

Un exemple de spécification d'une opération OP_i définie pour un type W d'objets est le suivant :

OP_i (P1 : in-out $W.id_J$-id_{J1} X, P2 : in $Y.id_{i1}.id_{i2}.id_{i3}$ S, P3 : in entier, P4 : out chaîne).

Cette opération peut être demandée par l'utilisateur au moyen de la commande suivante :

$OP_i(\sigma_1, \sigma_2, 1, B=?)$

où σ_1 est un objet de type W et σ_2 est un objet de type Y.

1.2 Concept de verrous généralisés.

L'utilisation des *verrous hiérarchiques* IS, IX, S, SIX et X (définis par Gray et al. [GRAY 75]) dans le modèle actuel d'objets et d'opérations généralisées présenterait quelques inconvénients :

- Etant donné que la gestion de verrous doit être totalement transparente à l'utilisateur, ce dernier ne peut plus déclarer le mode dans lequel il pense utiliser les objets et les verrous hiérarchiques perdent ainsi une partie de leur intérêt.

- En considérant les objets et leurs composants comme des arborescences la gestion de verrous hiérarchiques peut devenir lourde puisque tout accès à une feuille d'un arbre (i.e. un (sous) composant d'un objet) implique le verrouillage de tous les noeuds dans le chemin qui mène à la racine ;

- Dans certains cas, les verrous hiérarchiques peuvent réduire la concurrence entre transactions au lieu de l'encourager ; considérons par exemple l'arbre suivant représentant un objet σ avec les verrous hiérarchiques établis par une transaction t_1 :

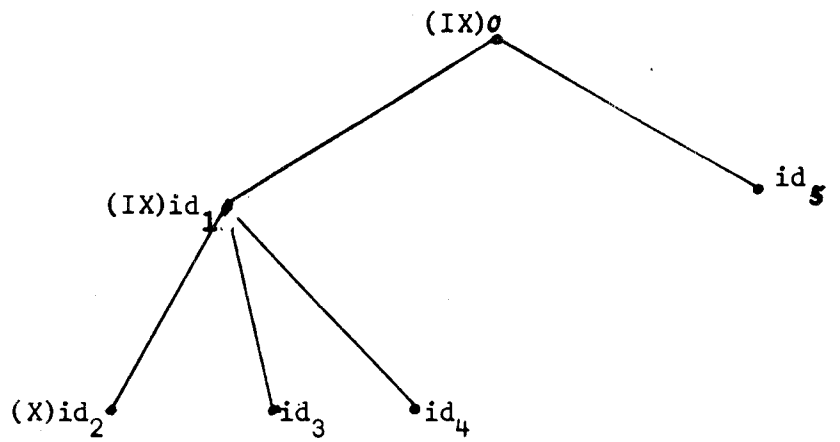


Fig. 1: Verrous Hiérarchiques d'une transaction t_1 sur l'objet o.

Ces verrous permettent à t_1 de modifier la feuille id_2 (c'est à dire le sous-composant $.id_1.id_2$). Si maintenant une autre transaction t_2 demande un verrou SIX sur le noeud id_1 afin de consulter et d'éventuellement modifier les feuilles id_3 et id_4, alors ce nouveau verrou sera refusé à cause de son incompatibilité avec le verrou IX que t_1 possède déjà sur le même noeud id_1 : et pourtant, les opérations de t_2 pourraient être exécutées concurremment avec celles de t_1.

D'après les considérations antérieures, pour favoriser la concurrence il semble préférable d'appliquer les verrous S et X directement au niveau des feuilles de l'arbre d'un objet et ne plus tenir compte des verrous intention (IS, IX, SIX). Malgré cela, la gestion des verrous peut demeurer lourde : chaque nouveau verrou à octroyer à une transaction entraîne de parcourir les arbres des verrous déjà établis par d'autres transactions sur l'objet en question et sur ces composants, afin de détecter les incompatibilités avec le nouveau verrou.

Pour favoriser la concurrence entre transactions et simplifier au même temps la gestion de verrous, nous proposons l'utilisation de *verrous généralisés* : les verrous établis sur les objets du système seront de divers modes propres au système, et pas seulement les modes S et X. La spécification d'une opération indique le verrou à établir sur chaque objet qui est paramètre ; pour chaque type W d'objets du système, il y aura alors m verrous possibles qu'on peut nommer V1, V2, ..., Vm, et un Tableau d'Incompatibilités (m × m) entre ces verrous. Avec ces verrous généralisés on attend les avantages suivants :

- Une simplification de la gestion de verrous : au lieu de considérer les verrous S et X au niveau des (sous) composants d'un objet, on va considérer maintenant divers verrous possibles au niveau de l'objet global ; la possiblité d'utilisation simultanée de différents (sous) composants est alors exprimée par les compatibilités entre les divers verrous possibles sur l'objet et la gestion de verrous ne nécessite plus l'examen des verrous au niveau des (sous) composants mais au niveau de l'objet.

- Une concurrence plus élevée entre transactions : avec les verrous généralisés, la modification d'un composant d'un objet en utilisant une opération OP_1 peut être considérée compatible avec la modification du même composant en utilisant une autre opération OP_2 : les opérations OP_1 et OP_2, demandées par 2 transactions différentes, pourraient donc être exécutées simultanément ; avec les seuls verrous S et X on n'obtiendrait pas une telle concurrence.

Pour réaliser la *Gestion de verrous Généralisés* le système doit connaître 2 tableaux associés à chaque type W d'objets du système :

- *Le Tableau d'Incompatibilités* TI_W exprime les incompatibilités entre les verrous de ce type W, c'est à dire les verrous qui ne peuvent pas être octroyés simultanément pour un même objet à 2 transactions différentes :

$$TI_W(V_i, V_k) = \begin{cases} 1 \text{ si les verrous } V_i \text{ et } V_K \text{ sont incompatibles} \\ 0 \text{ dans l'autre cas } (V_i \text{ et } V_K \text{ sont compatibles}). \end{cases}$$

- *Le Tableu d'Hiérarchies* TH_W exprime les relations d'hiérarchie entre les verrous de ce type W d'après les droits qu'ils impliquent, ainsi :

$$TH_W(V_i, V_K) \text{ pour } V_i \neq V_K = \begin{cases} 1\text{- si le verrou } V_i \text{ est plus "fort" que } V_K \text{ (i.e. } V_K \text{ donne plus de droits sur un objet que } V_i \text{ ; on dit aussi dans ce cas que } V_K \text{ est moins "fort" que } V_i), \\ 0 \text{ dans l'autre cas (: soit } V_K \text{ est plus "fort" que } V_i, \text{ soit il n'existe aucune relation d'hiérarchie entre } V_i \text{ et } V_K). \end{cases}$$

$$TH_W(V_i, V_i) = 0.$$

Les règles générales de la Gestion de Verrous Généralisés permettront d'octroyer un nouveau verrou V à une transaction sur un objet si (et seulement si) V est compatible avec les verrous que d'autres transactions possèdent déjà sur le même objet. En plus, une transaction pourra posséder sur un même objet plusieurs verrous (lui donnant droit à utiliser l'objet de diverses manières) mais dans ce cas il ne devrait pas y avoir de relations d'hiérarchie entre ces verrous.

Pour satisfaire les règles antérieures, lorsqu'une transaction souhaite lancer une opération sur un objet, le système détermine d'abord le verrou que la transaction doit obtenir sur l'objet, et ensuite, le système octroie ce verrou s'il est compatible avec les verrous que d'autres transactions possèdent sur le même objet ; en cas d'incompatibilité, la demande d'opération sera mise en attente jusqu'à la disparition du conflit. Pour déterminer le verrou qu'une transaction t doit obtenir sur un objet o de type W, pour réaliser une opération OP, le système suit les règles suivantes :

- soit V_i le verrou que l'opération OP indique dans sa spécification pour le paramètre formel correspondant à l'objet o.

- Dans le cas où t ne possède aucun verrou sur o, le verrou à obtenir est V_i.

- Dans le cas où t possède déjà un ou plusieurs verrous sur o, on considère 4 alternatives possibles :

  a) le verrou V_i existe déjà : t n'a pas besoin de l'obtenir à nouveau ;

  b) au moins un des verrous existants, V_K, est plus "fort" que V_i (i.e. $TH_W(V_K, V_i) = 1$) : alors t n'a pas besoin d'obtenir le verrou V_i, et le verrou V_K est maintenu ;

  c) V_i est plus "fort" qu'un ou plusieurs verrous existants (i.e. $TH_W(V_i, V_K) = 1$ pour au moins un verrou existant V_K) : alors t doit obtenir le verrou V_i pour *remplacer* tous les verrous existants moins "forts" que lui ;

d) V_i n'a pas aucune relation d'hiérarchie avec les verrous existants (i.e. pour chaque verrou existant V_K, $TH_W(V_K, V_i) \neq 1$ et $TH_W(V_i, V_K) \neq 1$) : alors t doit obtenir le verrou V_i pour le posséder simultanément avec les autres.

1.3 Un exemple d'application.

Nous voulons illustrer l'utilisation d'objets, d'opérations et de verrous généralisés avec un exemple sur la *gestion de comptes bancaires*. Par sa nature, la gestion de comptes bancaires est un bon exemple d'une application transactionnelle répartie, où les utilisateurs soumettent au système des transactions interactives manipulant de façon concurrente divers objets, lesquels peuvent résider sur différents sites.

Nous considérons une application simple qui manipule 2 types d'objets : les "comptes" et les "clients" ; on suppose qu'un compte peut appartenir à plusieurs clients, mais qu'un client possède au maximum un compte.

Un objet *compte* contient : le code du compte (qui est son nom unique), la date d'ouverture, et le montant. Sur les comptes, les transactions peuvent effectuer les opérations suivantes :

- OUVRIR : pour créer un nouveau compte et enregistrer son code dans les objets clients propriétaires (paramètres : code du compte, date courante, montant initial , nombre de clients propriétaires, noms de ces clients).

- CREDITER et DEBITER : pour mettre à jour le montant d'un compte (paramètres : code du compte, somme à créditer ou à débiter).

- TRANSFERER : pour transférer une somme d'un compte à un autre (paramètres : codes des deux comptes, somme à transférer).

- SOLDE : pour consulter le montant d'un compte (paramètre d'entrée : code du compte, paramètre de sortie : montant),

- FERMER : pour éliminer un compte (paramètre : code du compte).

Un objet *client* contient : le nom du client, l'adresse, l'information personnelle sur le client (date de naissance, numéro national d'identité, occupation), et le code de compte du client.
Sur les clients les transactions peuvent effectuer les opérations suivantes :

- NOUVEAU-CLIENT : pour créer un nouveau client et enregistrer les informations le concernant (paramètres : nom du client, adresse, date de naissance, numéro national d'identité, occupation). Cette opération doit précéder l'ouverture d'un compte pour le client (opération OUVRIR).

- CHANGER-AD : pour changer l'adresse d'un client (paramètres : nom du client, nouvelle adresse).

- CONSULTER-INFO : pour consulter l'information personnelle sur un client (paramètre d'entrée : nom du client, paramètres de sortie : date de naissance, numéro d'identité nationale, occupation).

- CONSULTER-CODE : pour consulter le code de compte d'un client (paramètre d'entrée : nom du client, paramètre de sortie : code).

- RETIRER : pour éliminer un client (paramètre : nom du client).

Les *spécifications* correspondantes aux types client et compte sont alors les suivantes :

```
type client est
    composants
        nom : chaîne,
        adresse : chaîne,
        info-personnelle : composants
                             date-naissance : entier,
                             identité : entier,
                             occupation : chaîne
                           fin composants
        code-compte : chaîne
    fin composants
    opérations
       NOUVEAU-CLIENT (CL : out client V1, NOM : in chaîne, ADR : in chaîne,
                       DATE : in entier, ID : in entier, OCC : in chaîne),
       CHANGER-AD (CL : out client. adresse V2, ADR : in chaîne),
       CONSULTER-INFO (CL : in client. info-personnelle V3, DATE : out entier,
                       ID : out entier, OCC : out chaîne),
       CONSULTER-CODE (CL : out entier, OCC : out chaîne),
       RETIRER (CL : in out client V1)
    fin opérations
fin client.
type compte est
    composants
       code : chaîne,
       date ouverture : entier,
       montant : réel
    fin composants
    opérations
        OUVRIR (C : out compte VV1, CO : in chaîne, DATE : in entier, SOMME : in réel,
                N : in entier, CL (i, pour i = 1:N) : out client.code-compte V5),
        CREDITER (C : in-out compte. montant VV2, SOMME : in réel),
        DEBITER (C : in-out compte.montant VV3,
                 SOMME : in réel),
        TRANSFERER (C1 : in-out compte. montant VV3, C2 : in-out compte. montant
                    VV2, SOMME : in réel),
        SOLDE (C : in compte.montant VV4, SOMME : out réel),
        FERMER (C : in-out compte VV1)
    fin opérations
fin compte.
```

Les verrous sur un objet compte que les transactions peuvent demander au système, sont les suivants :

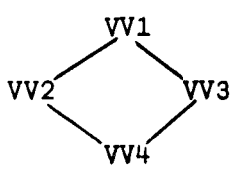
- VV1 : permet l'accès exclusif à tout le compte (utilisé par les opérations OUVRIR et FERMER),
- VV2 : permet lire et incrémenter le montant du compte (utilisé par les opérations CREDITER et TRANSFERER-en relation au deuxième paramètre),
- VV3 : permet lire et décrémenter le montant du compte (utilisé par les opérations DEBITER et TRANSFERER-en relation au premier paramètre) ;
- VV4 : permet lire le montant du compte (utilisé par l'opération SOLDE).

Le verrou VV2 est compatible avec VV2 (on permet des incrémentations concurrentes sur le montant d'un même compte), et le verrou VV4 est compatible avec VV4 ; toutes les autres combinaisons sont incompatibles. La Table d'Incompatibilités et la Table d'Hiérarchies pour le type compte sont alors les suivants :

TI_{compte} :

	VV1	VV2	VV3	VV4
VV1	1	1	1	1
VV2	1	0	1	1
VV3	1	1	1	1
VV4	1	1	1	0

hiérarchie



TH_{compte} :

	VV1	VV2	VV3	VV4
VV1	0	1	1	1
VV2	0	0	0	1
VV3	0	0	0	1
VV4	0	0	0	0

Fig. 2. Tables d'Incompatibilités et d'Hiérarchies pour le type compte.

Les verrous sur un objet client que les transactions peuvent demander au système, sont les suivants :

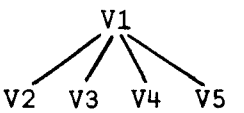
- V1 : permet l'éccès exclusif à tout le client (utilisé par les opérations NOUVEAU-CLIENT et RETIRER),
- V2 : permet modifier l'adresse du client (utilisé par l'opération CHANGER-AD),
- V3 : permet lire l'information personnelle du client (utilisé par l'opération CONSULTER-INFO),
- V4 : permet lire le code (de compte) du client (utilisé par l'opération CONSULTER-CODE),
- V5 : permet écrire le code (de compte) du client (utilisé par l'opération OUVRIR en relation aux derniers paramètres).

On permet des modifications concurrentes sur un même client si elles concernent différentes parties ; ainsi, le verrou V2 est compatible avec V3, V4 et V5, et le verrou V5 est compatible avec V2 et V3, mais pas avec V4. Le tableau d'Incompatibilités et le tableau d'hiérarchies pour le type client sont alors les suivantes :

TI_{client} :

	V1	V2	V3	V4	V5
V1	1	1	1	1	1
V2	1	1	0	0	0
V3	1	0	0	0	0
V4	1	0	0	0	1
V5	1	0	0	1	1

hiérarchie :



TH_{client} :

	V1	V2	V3	V4	V5
V1	0	1	1	1	1
V2	0	0	0	0	0
V3	0	0	0	0	0
V4	0	0	0	0	0
V5	0	0	0	0	0

Fig. 3 : Tableaux d'Incompatibilités et d'Hiérarchies pour le type client.

Un exemple d'une transaction interactive soumise par un utilisateur est le suivant (le symbole # dénote les réponses du système affichées sur le terminal et le symbole ∇ délimite les commandes de l'utilisateur) :

```
DEBUT-TR ∇
    CONSULTER-CODE (MATHIEU, CODE =?) ∇
      # CODE = C10 (réponse du système)
    SOLDE (C10, SOMME-10= ?) ∇
      # SOMME-10 = 600 000.
    DEBUT-INT ∇
        OUVRIR (C20, 'C20', 81 11 05, 250 000., 2, DUPONT, THIBAUT) ∇
        TRANSFERER (C10, C20, 100 000.) ∇
        DEBITER (C20, 50 000.) ∇
        SOLDE (C20, SOMME-20 = ?) ∇
        # SOMME-20 = 300 000.
    FIN 2 ∇
    DEBUT -INT ∇
        SOLDE (C5, SOMME-5 = ?) ∇
        #  SOMME-5 = 5000.
        TRANSFERER (C5, C10, 5000.) ∇
        FERMER (C5) ∇
    FIN 1 ∇
  FIN 2 ∇
```

Dans cet exemple l'utilisateur consulte d'abord le code du compte d'un client MATHIEU : le système établit le verrou V4 sur ce client et affiche le code : C10.
L'utilisateur consulte après le solde de ce compte C10 : le système établit le verrou VV4 sur C10 et affiche son solde : 600 000 F.

L'utilisateur décide à ce moment d'ouvrir une transaction interne où il va manipuler un nouveau compte : C20. Dans cette transaction interne, l'utilisateur demande d'abord l'ouverture du nouveau compte C20 appartenant aux clients DUPONT et THIBAUT : le système crée le compte C20 et établit sur lui le verrou VV1 ; en plus, le système verrouille les clients DUPONT et THIBAUT en mode V5 pour écrire le code du compte C20 dans leurs composants correspondants. Une fois que le compte C20 a été ouvert, l'utilisateur demande le transfert de 100 000 F du compte C10 vers C20 : à ce moment les verrous qui ont été octroyés à la transaction sont les suivants :

- V4 sur le client MATHIEU
- V5 sur le client DUPONT
- V5 sur le client THIBAUT
- VV4 sur le compte C10
- VV1 sur le compte C20.

Avant de réaliser le transfert demandé par l'utilisateur, le système devrait établir les verrous VV3 sur C10 et VV2 sur C20 : pour C10, le verrou VV3 va être en fait établit en éliminant le verrou existant, VV4, qui est inférieur ; pour C20, il n'y a pas besoin d'établir le verrou VV2 puisque le verrou existant, VV1, est supérieur.

D'autres opérations sur le compte C20 sont demandées par l'utilisateur dans la transaction interne, et pour finaliser l'utilisateur utilise la commande FIN 2 : à ce moment, le système va exécuter réellement les modifications des comptes C10 et C20 et des clients MATHIEU, DUPONT et THIBAUT (modifications qui jusqu'à là étaient enregistrées comme des intentions) et va libérer tous ces objets, sauf le compte C10 et le client MATHIEU lesquels sont référencés au niveau global.

Dans la suite de la transaction, l'utilisateur réalise une deuxième transaction interne pour manipuler un autre compte, C5, et en utilisant encore le compte C10. A la fin de toute la transaction, les intentions de modification restantes sont exécutées et les objets encore référencés sont libérés.

2. MISE EN OEUVRE DES TRANSACTIONS EMBOITEES AVEC OBJETS, OPERATIONS ET VERROUS GENERALISES.

Dans cette partie nous généraliserons les processus Producteur, Consommateur et Moniteur, décrits dans le chapitre IV, afin de réaliser des transactions emboîtées avec objets, opérations et verrous généralisés. Pour cela, nous présenterons d'abord une description du déroulement d'une transaction, ensuite les structures d'information nécessaires, et finalement un résumé des actions réalisées par les processus Producteur, Consommateur et Moniteur à la réception de chaque requête possible.

2.1 Description du déroulement d'une transaction.

2.1.1. Traitement normal.

- *Initialisation de la transaction :*

Lorsqu'un utilisateur soumet une commande "DEBUT-TR" pour initier une transaction, le Moniteur de son site assignera un identificateur unique, t, à cette nouvelle transaction (par le Mécanisme du Séquenceur Circulant de Le Lann [LELAN 79 A], décrit dans le chapitre III - partie 6.1), et activera son processus Producteur, P_t, au moyen d'un signal "DEBUT-TR (paramètres)". P_t initialise alors le descripteur de la transaction (DES), principalement avec les informations suivantes :

- identificateur de la transaction *tr* := t
- état de la transaction *etatr* := INITIAL,
- niveau d'emboîtement en cours *niveau* := ∅ ; ...

Après cette initialisation, l'utilisateur, peut soumettre les autres commandes de la transaction, c'est à dire : des commandes contenant des opérations à réaliser sur les objets du système, des commandes de contrôle pour commencer ou pour finir une transaction interne, et la commande de contrôle "ANNULER" pour demander une annulation partielle de la transaction.

- *Demande d'une opération* :

Une commande de l'utilisateur contenant une demande d'opération, présente la forme générale suivante :

"OP (σ_1, σ_2,...,σ_{n1}, c_1, c_2,...,c_{n2}, r_1 =?, r_2 =?,...,r_{n3} =?)"

où

σ_1, σ_2,...,σ_{n1} sont des noms d'objets du système,

c_1, c_2,...,c_{n2} sont des constantes (valeurs que l'utilisateur fournit)

r_1, r_2,...,r_{n3} désignent les valeurs que l'utilisateur attend en réponse.

Le *Producteur* de la transaction, P_t, examine les sites de résidence des objets paramètres de l'opération (σ_1, σ_2,...,σ_{n1}) dans la Table d'Objets Globaux (TOG) laquelle est un répertoire dupliqué sur chaque site. Pour chaque site qui est concerné par l'opération et qui ne possède pas encore un processus Consommateur de la transaction t, le Producteur P_t enverra une demande "DEBUT-TR (t, niveau, sitep, sitec)" qui déclenchera l'activation d'un tel processus Consommateur C_t (dans cette activation C_t initialise son Descripteur local de la transaction, et sa Pile d'Intentions). Ensuite, P_t envoie la demande d'opération au processus Consommateur C_t correspondant au site de résidence du premier objet paramètre σ_1 (l'opération OP étant définie pour le type d'objet auquel appartient σ_1) et se met en attente de la réponse "ACK-OP (r_1 = <valeur>, r_2 = <valeur>,..., r_{n3} = <valeur>)".

Toutes les demandes et réponses interchangées entre les processus Producteur et Consommateurs d'une transaction, passent par l'intermédiaire de leurs *Moniteurs* (la configuration d'un site du système en termes de processus, et le flux d'information déclenché par une commande ont été montrés dans les figures 6 et 7 du chapitre III - partie 6). Dans le cas présent, avant que la demande d'opération OP soit délivrée au Consommateur C_t chargé de la réaliser, le Moniteur de son site (*sitec*) doit établir les verrous nécessaires sur les objets locaux que le consommateur va accéder, d'après les règles de gestion de Verrous Généralisés qui ont été présentées dans la partie 1.2 de ce chapitre ; ces règles, basées sur l'examen des tableaux d'Incompatibilités et d'Hiérarchies relatifs aux types des objets servant de paramètres, permettront l'accès simultané à un même objet à diverses transactions qui ont établit auparavant des verrous compatibles ; en plus, plusieurs verrous sans relation d'hiérarchie sont permis à une transaction sur un même objet.
Dans le cas d'un conflit de concurrence de t avec une autre transaction t' (i.e.

t' possède sur un objet o_1 un verrou incompatible avec celui demandé par t sur o_1), le Moniteur place la demande d'opération OP dans la file d'Attente de t ($FATT_t$) ; en plus, pour prévenir les interblocages on donne toujours la priorité à la transaction la plus ancienne : ainsi, si t est plus ancienne que t' (d'après leurs identificateurs), le Moniteur communiquera au Consommateur $C_{t'}$ un signal "INTERBLOCAGE (o_1)" pour déclencher une annulation partielle de t' qui libérera l'objet o_1 en conflit.

Lorsque le *Consommateur* C_t (du *sitec*) reçoit la demande d'opération "OP (o_1, o_2,...,o_{n1}, c_1, c_2,...,c_{n2}, r_1 =?, r_2 =?,...,r_{n3} =?)", ce processus réalise les actions suivantes :

a) Préparer la pile d'Intentions :

La Pile d'Intentions de C_t va contenir les modifications à effectuer sur les objets locaux référencés par la transaction t ; pour chaque objet local o_i, dont un composant va être modifié par l'opération OP (i.e. la classe de o_i est <u>out</u> ou <u>in-out</u> d'après la spécification de OP), C_t introduit dans la zone en cours de la Pile d'Intentions : le type de l'objet o_i, son nom et la chaîne d'identificateurs délimitant la composant à modifier (cela en supposant que ces informations ne figurent pas encore au moment de l'opération).

b) Copier les objets paramètres non locaux :

Pour chaque objet σ_K, paramètre de l'opération OP et non-résident sur son site (*sitec*), C_t enregistre dans une table spéciale, TCOPIES, les informations suivantes : la classe de l'objet σ_K, son type, son nom et la chaîne d'identificateurs délimitant le composant intervenant dans l'opération OP ; ensuite C_t sollicite une copie de ce sous-composant de σ_K au site de résidence de σ_K (*sitec'*) en lui envoyant la demande "COPIER (<classe>, <type>, σ_K.<chaîne d'identif. du composant> <verrou>, sitec)".
Le paramètre <verrou> indique le verrou sur o_K exigé par la spécification de l'opération OP ; le Moniteur du *sitec'* doit établir ce verrou avant de délivrer la demande COPIER au Consommateur C_t, mais en cas de conflit de concurrence cette demande sera mise dans la File d'attente de t ($FATT_t$). Lorsque le Consommateur C_t du *sitec'* reçoit la demande COPIER, il introduit dans sa Pile d'Intentions les informations concernant σ_K (type, nom, et chaîne d'identificateurs) si cet objet va être modifié par l'opération OP (i.e. le paramètre

<classe> est out ou in-out) et si ces informations ne figurent pas encore dans la zone en cours de la Pile ; finalement, C_t du *sitec'* envoie à C_t du *sitec* la réponse "ACK-COPIER (o_K.<chaîne d'identif. du sous-composant>, <représentation>) contenant la version la plus récente du sous-composant de o_K qui est sollicité (cette version est la dernière indiquée dans la Pile d'Intentions si elle existe, ou sinon, celle indiquée par l'objet o_K même).

c) Exécuter l'opération :

Une fois que le consommateur C_t du *sitec* a reçu des copies de tous les objets paramètres non-locaux et a placé leurs représentations dans TCOPIES, il est prêt à réaliser l'opération OP en exécutant le code correspondant à son implémentation.Pendant cette exécution les écritures sur les objets paramètres locaux ne sont pas effectuées sur les objets mêmes, mais elles sont enregistrées dans la Pile d'Intentions ; les lectures sont réalisées à partir des dernières versions d'objets figurant dans la Pile d'Intentions, si elles existent, ou sinon à partir des objets mêmes. Par rapport aux objets paramètres non locaux, les écritures et les lectures sont effectuées sur les copies figurant dans TCOPIES.

d) Retourner les copies des objets paramètres non locaux :

A la fin de l'opération OP, C_t retourne aux sites d'origine les copies des objets non locaux qui ont été modifiés : pour chaque objet σ_K figurant dan TCOPIES dont la classe est out ou in-out, C_t envoie au site de résidence de σ_K (*sitec'*) la demande "RETOURNER (σ_K.<chaîne d'identif. du sous-composant>, <nouvelle représentation>, sitec)" contenant la nouvelle représentation du sous-composant de σ_K qui est intervenu dans l'opération.
Le consommateur C_t du *sitec'* enregistre dans sa Pile d'Intentions cette "écriture" sur σ_K et envoie à C_t du *sitec* la réponse "ACK-RETOURNER (σ_K)".

e) Envoyer la réponse au Producteur:

Une fois que le consommateur C_t du *sitec* a reçu des autres consommateurs toutes les réponses possibles du type "ACK-RETOURNER (o_K)", il élimine l'information contenue dans TCOPIES et envoie au Producteur P_t la réponse "ACK-OP (r_1 = <valeur> , r_2 = <valeur>,...,r_{n3} = <valeur>)" contenant les valeurs calculées par l'opération OP qui sont attendues par l'utilisateur.

Lorsque le *Producteur* P_t reçoit la réponse "ACK-OP (r_1 = <valeur>, r_2 = <valeur>,...,r_{n3} = <valeur>)", il affiche les valeurs reçues sur le terminal de l'utilisateur et il est prêt à recevoir la prochaine commande de la transaction.

- *Début d'une transaction interne* :

La commande de contrôle "DEBUT-INT", soumise par l'utilisateur pour signaler le début d'une transaction interne (et pour demander la création du point de sauvegarde associé), provoquera la diffusion de la demande "DEBUT-INT (niveau)" par le Producteur P_t vers tous les Consommateurs de la transaction. Lorsqu'un Consommateur C_t reçoit cette demande, il compare le niveau d'emboîtement de la transaction communiqué en paramètre, avec sa version propre : s'ils sont différents C_t communique cette anomalie au Producteur ; dans le cas normal C_t insère une marque dans sa Pile d'Intentions pour commencer une nouvelle zone, incrémente sa variable *niveau* (*niveau* := *niveau* +1), et envoie au Producteur P_t la réponse "ACK-DEBINT". P_t attend les réponses de tous les Consommateurs pour incrémenter à son tour sa variable *niveau*. L'utilisateur peut à ce moment soumettre les commandes qui correspondent à la transaction interne.

- *Fin d'une transaction interne* :

La commande de contrôle "FIN 1" (ou respectivement "FIN 2"), soumise par l'utilisateur pour signaler la fin de la transaction interne en cours, déclenchera un mécanisme de Validation en 2 étapes qui sera résistant aux défaillances:
D'abord le *Producteur* P_t diffuse à tous les consommateurs la demande "FIN 1 (niveau, contenu (LC))" (resp. "FIN 2 (niveau, contenu (LC))") contenant la liste de Consommateurs participant dans la transaction.
Lorsqu'un *Consommateur* C_t reçoit cette demande, il met à jour ses versions locales de la Liste de Consommateurs, et de l'état de la transaction : *étatr* := PREVALIDATION 1 (resp. PREVALIDATION 2) ; ensuite C_t envoie au Producteur P_t la réponse "ACK-FIN1" (resp. "ACK-FIN2").
Lorsque le *Producteur* P_t reçoit les réponses de tous les Consommateurs, il met à jour à son tour sa version de l'état de la transaction : *étatr* := VALIDATION 1 (resp. VALIDATION 2) : on peut dire alors que la transaction interne en cours est en point de Validation (de non-retour). Ensuite, P_t diffuse à tous les Consommateurs la demande "VALIDER 1" (resp. "VALIDER 2").

Lorsqu'un *Consommateur* C_t reçoit la demande "VALIDER 1", il réalisera une "Validation Différée" de la transaction interne en cours : d'abord la variable *étatr* est mise à jour : *étatr* := VALIDATION 1 ; ensuite, la dernière zone de la Pile d'Intentions est supprimée et les intentions correspondantes à cette zone sont incorporées à celles de la zone immédiatement précédante ; finalement C_t décrémente sa variable *niveau*, met à jour sa variable *étatr* (*étatr* := INITIAL) et envoie au Producteur P_t la réponse "ACK-VALIDER 1". Il s'agit alors de la fin de la transaction interne entraînant la suppression du point de sauvegarde associé mais pas la libération de ressources.

Par contre, lorsqu'un *Consommateur* C_t reçoit la demande "VALIDER 2", il s'agit d'un point cohérent de la transaction (i.e. l'utilisateur signale un état cohérent des ressources utilisées par la transaction) et en conséquence, les ressources qui ne vont plus être utilisées par la suite peuvent être libérées. C_t réalise alors une "Validation Permanente" de la transaction interne en cours : d'abord la variable *étatr* est mise à jour : *étatr* := VALIDATION 2 et toutes les intentions de la Pile sont exécutées sur les objets ; ensuite, C_t libère les objets qui ont été "ouverts" (i.e. utilisés par la première fois) dans la transaction interne en cours : pour cela C_t communique à son Moniteur des signaux du type "LIBERER (σ_i)" (le Moniteur enlévera les verrous mis par t sur les objets libérés et activera les demandes d'autres transactions qui sont en attente de ces objets) ; finalement, C_t supprime la dernière zone de la Pile d'Intentions, décrémente sa variable *niveau*, met à jour sa variable *étatr* (si *niveau* ≥ 0 alors *étatr* := INITIAL sinon *étatr* := COMPLET) et envoie au Producteur P_t la réponse "ACK-VALIDER 2".
Lorsque le *Producteur* P_t reçoit les réponses de tous les Consommateurs, il décrémente à son tour sa variable *niveau* et met à jour sa variable *étatr* (si *niveau* ≥ 0 alors *étatr* := INITIAL sinon *étatr* := COMPLET). Si *étatr* devient INITIAL l'utilisateur peut soumettre des nouvelles commandes de la transaction mais si *étatr* devient COMPLET il s'agit de la fin de toute la transaction : P_t diffuse alors la demande "TERMINER" à tous les consommateurs afin qu'ils deviennent inactifs ; lorsque P_t reçoit les réponses "ACK-TERMINER" de tous les consommateurs, lui-même devient inactif.

- *Création et destruction d'objets :*

Lorsqu'une commande soumise par l'utilisateur entraîne la *création* d'un objet σ_i, le Producteur de sa transaction, P_t, communiquera au Moniteur de son site le signal "CREER (σ_i, <type>, <site>)" indiquant : le nom de l'objet à créer, son type, et son site de résidence (ce site pourrait être spécifié par l'utilisateur dans un paramètre de la commande, ou si rien d'autre est spécifié, P_t peut supposer que c'est son propre site). L'envoi du signal "CREER" doit précéder chez le Producteur P_t, le lancement de l'opération qui va initialiser le nouvel objet. Le Moniteur de son site répertorie alors le nouvel objet σ_i dans sa copie locale de la Table d'Objets globaux (TOG), en consignant les informations reçues en paramètres, et diffuse le signal "CREER (σ_i, <type>, <site>)" à tous les sites du système, afin que les autres Moniteurs fassent de même ; en plus le Moniteur du site de résidence de σ_i doit créer la maquette correspondante à son type.

Lorsqu'une commande soumise par l'utilisateur entraîne la *destruction* d'un objet σ_i, le Consommateur C_t correspondant au site de résidence de σ_i enregistrera dans sa Pile d'Intentions l'intention de détruire σ_i (et cela dans le déroulement normal de l'opération correspondante à la commande). Ultérieurement, lors d'une Validation Permanente d'intentions, C_t exécutera l'intention de détruire σ_i en envoyant à son Moniteur le signal "DETRUIRE (σ_i, <type>)"; son Moniteur enlévera alors l'objet σ_i de sa copie locale de la Table d'Objets Globaux (TOG) et diffusera le signal "DETRUIRE (σ_i, <type>)" à tous les sites du système, afin que tous les autres Moniteurs fassent de même.

2.1.2 Traitement des situations d'exception.

Pour une transaction emboîtée travaillant avec objets, opérations et verrous généralisés, le traitement des situations d'exception (c'est à dire : les demandes d'annulation partielle faites par l'utilisateur, la détection et traitement des risques d'interblocage et la détection des défaillances de système affectant la transaction) est identique à celui présenté dans le chapitre IV (partie 3), sauf qu'au lieu de considérer des fichiers on considérera plutôt des objets généralisés.

Le traitement normal et le traitement des situations d'exception pour chaque transaction soumise dans le système, assurent en dernière instance la *préservation de la cohérence* des objets du système : en effet, d'après les justifications données dans le chapitre IV (partie 2), chaque transaction emboîtée comprend plusieurs unités de cohérence (délimitées par les commandes "FIN 2") et son exécution est équivalente à celle d'une séquence de transactions *Bien-Formées* et à *Deux-Phases* ; la gestion de verrous réalisée par chaque processus Moniteur produit un *ordonnancement légal* des transactions concurrentes au niveau de chaque site ; tout ceci assure la préservation de la cohérence dans le système (selon les principes de Eswaran et Gray [ESWAR 76]).

2.2 Tables utilisées par les processus producteur, consommateur et Moniteur.

2.2.1. Répertoires globaux.

La Table d'Objets Globaux (TOG) est un répertoire dupliqué sur chaque site qui indique pour chaque objet du système son nom, son type et son site de résidence. La version de cette table sur un site contient, en plus, pour chaque objet local, un pointeur à sa représentation (cette représentation sera en général un arbre des représentations des (sous) composants de l'objet). La TOG d'un site peut être consultée par tous les processus locaux (Moniteur, Producteurs ou Consommateurs) mais modifiée uniquement par le Moniteur ; sa configuration peut être représentée ainsi :

TOG (Table d'Objets Globaux):

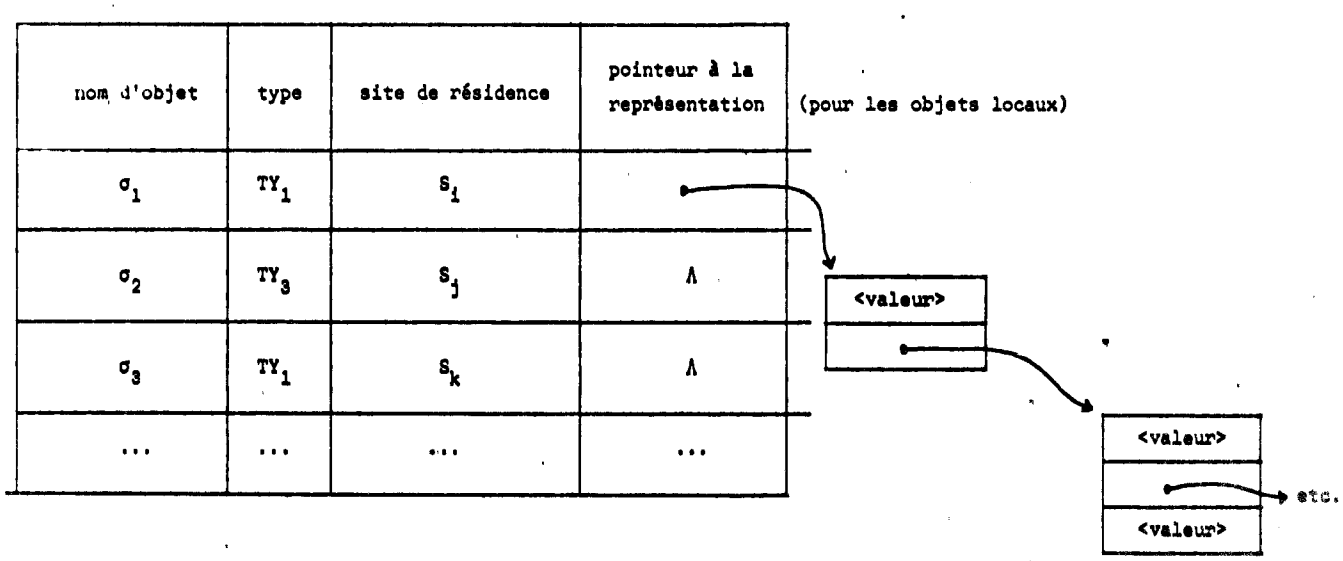


Fig 4 : Configuration de la Table d'Objets Globaux (TOG): version sur un site S_i .

Un autre répertoire dupliqué sur chaque site est la Table de Spécification des Types (TST), laquelle contient la spécification de tous les types d'objets du système et peut être consultée par tous les processus locaux d'un site (Moniteur, Producteurs et Consommateurs).
Pour chaque type d'objet la table TST indique : le descripteur des types des composants (qui peut être en général un arbre de descripteurs) et la spécification de ses opérations, laquelle définit les paramètres formels de chaque opération. Pour les types d'objets qui peuvent résider sur un site (types "locaux"), la version locale de TST indique, en plus, l'implémentation de leurs opérations. La configuration de TST peut être représentée de la manière suivante :

TST (Table de Spécification des Types) :

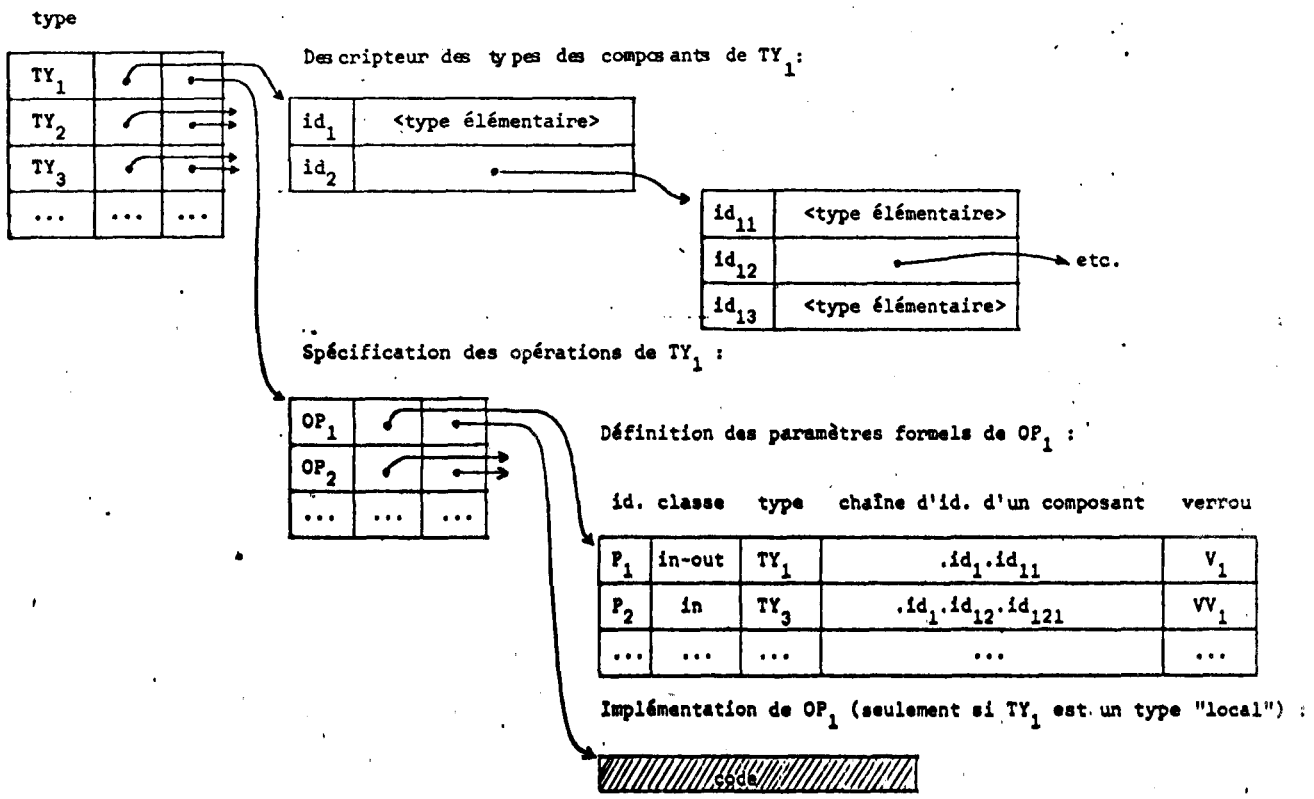


Fig. 5: Configuration de la table de Spécification des Types (TST): version sur un site qui va contenir des objets du type TY_1 .

2.2.2 *Tables propres aux processus Producteur et Consommateurs d'une transaction.*

Le Producteur d'une transaction gère la Liste de Consommateurs (LC) et le Descripteur de la transaction (DES) avec les informations déjà mentionnées dans le chapitre IV (partie 3.1.2).

Chaque consommateur d'une transaction t possède des versions locales de la Liste de Consommateurs (LC) et du Descripteur de la transaction (DES). En plus, chaque Consommateur gère dynamiquement une PILE contenant les intentions de modification des objets locaux ; la Pile d'Intentions est composée de diverses zones correspondant aux divers niveaux d'emboîtement des transactions internes de t.
La configuration d'une Pile d'Intentions peut être représentée ainsi :

PILE (d'Intentions) :

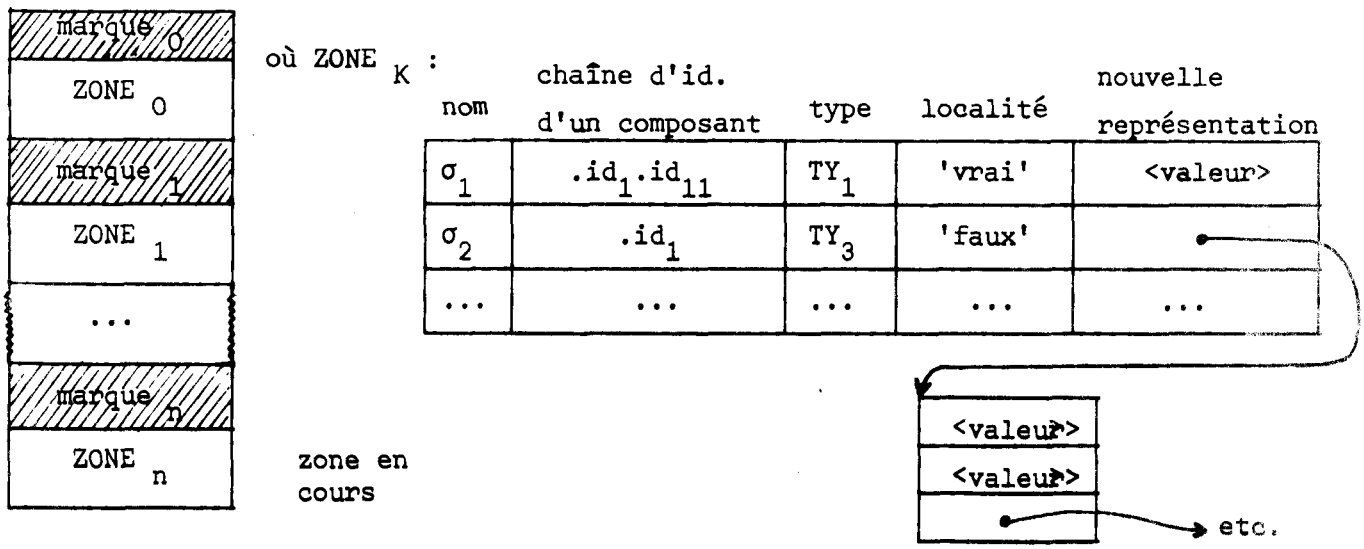


Fig. 6 : Configuration de la Pile d'Intentions (PILE) d'un Consommateur.

Pour chaque objet σ figurant dans une $ZONE_K$ de la Pile d'Intentions, les informations suivantes sont indiquées : son nom, son type, la chaîne d'identificateurs délimitant le (sous) composant à modifier, la nouvelle représentation de ce composant, et un indicateur de localité informant si l'objet σ a été référencé par la première fois dans $ZONE_K$ ou non.

Chaque Consommateur possède aussi une Table de Copies (TCOPIES) où il place des copies des objets non locaux qui sont des paramètres pour l'opération en cours. La configuration de cette table peut être représentée de la manière suivante :

TCOPIES (Table de Copies) :

classe	nom	chaîne d'id. d'un composant	type	représentation
in	σ_3	$id_1.id_{12}.id_{123}$	TY_1	<valeur>
in-out	σ_4	id_2	TY_4	
...	...	...	...	...

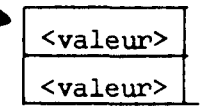


Fig. 7 : Configuration de la Table de Copies (TCOPIES) d'un Consommateur.

2.2.3 Tables propres au processus Moniteur d'un site.

Le Moniteur de chaque site gère une File de transactions en attente d'identification (FID), une Table de Producteurs locaux (TPRO) et une Table de Consommateurs locaux (TCON), déjà mentionnées dans le chapitre IV (partie 3.1.4).

Pour la gestion de verrous, le Moniteur de chaque site possède des copies du Tableau d'Incompatibilités (TI) et du Tableau d'Hiérarchies (TH) associés aux types d'objets qui peuvent résider sur le site (nous avons défini le contenu de ces tableaux dans la partie 1.2 de ce chapitre). En plus, le Moniteur de chaque site gère une Table de Verrous (TV) qui enregistre les verrous établis sur les objets locaux par les diverses transactions ; la configuration de cette table peut être la suivante :

nom	type	id. transaction	verrou
σ_1	TY_1	t1	V1
σ_1	TY_1	t2	V2
σ_2	TY_3	t2	VV1
...	...	...	...

Fig. 8 : Configuration de la table de Verrous (TV) établis sur les objets d'un site.

2.3 Processus Producteur.

Les tables suivantes résument les actions réalisées par le processus Producteur P_t d'une transaction t, à la réception des différentes requêtes possibles (ces requêtes peuvent être de 4 types : C : commande, D : demande, R : réponse, S : signal). Pour chaque état d'entrée de P_t et chaque requête arrivée, la table indique les actions déclenchées, les requêtes produites (qui sont passées au Moniteur) et l'état de sortie de P_t.

. Traitement Normal (P_t) :

ETAT D'ENTREE DE P_t	REQUETE ARRIVEE	TRANSITION DECLENCHEE (ACTIONS)	REQUETES PRODUITES	ETAT DE SORTIE DE P_t
Inactif	S:DEBUT-TR(t,...)	Initialiser DES : tr := t ; etatr := INITIAL ; niveau :=∅; RI := 'faux';... LC := ∅ (vide)		Etape I
Etape 1	C : $OP_K(\sigma_1,\ldots,\sigma_{n1}$ $c_1,\ldots,c_{n2}$, $r_1=?,\ldots,r_{n3}=?)$	Pour chaque σ_i, i=1:n1 faire sitec:=site de σ_i ; si σ_i n'existe pas encore alors fsi si sitec ∉ LC alors fsi fait sitec1 := site de σ_1	S : CREER (σ_i,type, sitec) D : DEBUT-TR (tr, niveau,...) vers C_t de sitec D : OP_K(<paramètres>) vers C_t de sitec1	Att-acKOP_K
Att-acKOP_K	R : ACK-DEBTR (sitec)	Ajouter sitec à LC		Att-acKOP_K
Att-acKOP_K	R : ACK-OP_K (r_1=valeur$_1$,..., r_{m3}=valeur$_{n3}$)	Afficher les valeurs reçues sur le terminal		Etape I
Etape I	C:DEBUT-INT		D : DEBUT-INT (niveau) vers tous les c_t ∈ LC	Att-acKdebint
Att-acKdebint	R : ACK-DEBINT	Si tous les ack sont arrivés alors niveau := niveau +1 sinon attendre fsi		Etape I Att-acKdebint
Etape I	C : FIN_j pour j=1,2		D : FIN_j (niveau, contenu (LC)) vers tous les C_t ∈ LC	Att-ackfinj
Att-ackfin_j	R : ACK-FIN_j pour j=1,2	si tous les ack sont arrivés alors étatr := $VALIDATION_j$ sinon attendre fsi	D : $VALIDER_j$ vers tous les C_t LC	Att-ac$Kvalider_j$ Att-ac$Kfin_j$
Att-ack-valider$_j$	R : ACK-$VALIDER_j$ pour j=1,2	si tous les ack sont arrivés alors niveau := niveau -1 ; si niveau ≥ 0 alors étatr:=INITIAL sinon étatr:=COMPLET fsi sinon attendre fsi	D : TERMINER vers tous les C_t LC	Etape I Att-ack terminer Att-ack valider$_j$
Att-ack terminer	R : ACK-TERMINER	Si tous les ack sont arrivés alors sinon attendre fsi		Inactif Att-ack terminer

BUS LILLE

---. Traitement des situations d'exception (P_t) :

ETAT D'ENTREE DE P_t	REQUETE ARRIVEE	TRANSITION DECLENCHEE (actions)	REQUETES PRODUITES	ETAT DE SORTIE DE P_t
Etape I	C : ANNULER		D : ANNULER (niveau) vers tous les $C_t \in LC$	Att-acKannuler
Att-acK annuler	R : ACK-ANNULER	Si tous les ack sont arrivés alors notifier à l'utilisateur: "transaction in erne annulée" sinon attendre fsi	---------- ----------	Etape 1 Att-acKannuler
état quelconque sauf {Inactif, Att-acKterminer}	S : INTERBL (ni)	Si RI = 'faux' alors si étatr=VALIDATION 1 et ni=niveau alors NA := ni-1 sinon NA := ni fsi RI := 'vrai' sinon si ni < NA alors NA := ni fsi fsi	D : ANN-INTERBL (NA, étatr) vers tous les $C_t \in LC$ D : ANN-INTERBL (NA, étatr) vers tous les $C_t \in LC$	Att-acKanninterbl
Att-ack-anninterbl	R : ACK-ANNINTERBL (ni)	si ni = NA et tous les ack sont arrivés alors niveau := NA ; étatr := INITIAL ; RI := 'faux' ; notifier à l'utilisateur : "transaction reprise au niveau NA" sinon attendre fsi	---------- ----------	Etape 1 Att-acKanninterbl
état quelconque sauf {Inactif}	S : NACK, ou S : PANNE-C, ou S : REINSERTION	si RI = 'vrai' alors sinon cas étatr = INITIAL : VALIDATION$_j$ (j=1,2) COMPLET fin-cas fsi	D : ANNULER (niveau) vers tous les $C_t \in LC$ D : VALIDER$_j$ vers tous les $C_t \in LC$ D : TERMINER vers tous les $C_t \in LC$	Att-acKannuler Att-acKvalider$_j$ Att-acKterminer
état E quelconque sauf {Inactif}	D : QETATR P (sitec)		R : ACK-QETATRP (étatr) vers C_t de sitec	état E

BUS LILLE

2.4. Processus Consommateur

Par la suite nous présentons les tables résumant les actions réalisées par un processus Consommateur C_t d'une transaction t, à la réception des différentes requêtes possibles.

. Traitement normal (C_t) :

ETAT D'ENTREE DE C_t	REQUETE ARRIVEE	TRANSITION DECLENCHEE (ACTIONS)	REQUETES PRODUITES	ETAT DE SORTIE DE C_t
Inactif	D : DEBUT-TR (t, ni, idp, idc)	Initialiser DES : tr := t ; étatr := INITIAL ; niveau := 0 ; RI := 'faux' ; sitep := idp ; sitec := idc ; ... Introduire marque$_{niveau}$ dans PILE ; Tant que niveau ≠ ni faire niveau := niveau +1 ; Introduire marque$_{niveau}$ dans PILE fait	R : ACK-DEBTR (sitec) vers P_t de sitep	Etape I
Etape I	D : $OP_K(\sigma_1,\ldots,\sigma_{n1}, c_1,\ldots,c_{n2}, r_1=?,\ldots,r_{n3}=?)$	Préparer PILE et TCOPIES : Pour chaque σ_i, i=1:ni faire Si σ_i est local alors si classe de σ_i=out/in-out et $\sigma_i \notin$ ZONE$_{niveau}$ de PILE alors Introduire informations de σ_i dans PILE fsi sinon introduire informations de σ_i dans TCOPIES ; sitec' := site de σ_i -----	D : COPIER ($\sigma_1,\ldots$, classe, verrou, sitec) vers C_t de sitec'	
		fsi fait ; Si il y a des σ_i non-locaux alors attendre ---------- sinon Exécuter OP_K (écritures sur PILE et TCOPIES) ; -----	----------	Att-acKcopier
		fsi	R : ACK-OP_K (r_1:=valeur$_1$, ..., r_{n3} = valeur$_{n3}$) vers P_t de sitep ----------	Etape I

ETAT D'ENTREE DE C_t	REQUETE ARRIVEE	TRANSITION DECLENCHEE (ACTIONS)	REQUETES PRODUITES	ETAT DE SORTIE DE C_t
Att-ACKcopier	R : ACK-COPIER (σ_i,...,(représentation))	Placer la représentation du composant de σ_i dans TCOPIES ; Si tous les ack ne sont pas arrivés alors attendre		Att-ACKcopier
		sinon Exécuter OP_K ; Pour chaque $\sigma_i \in$ TCOPIES faire Si classe de σ_i = out/in-out alors sitec' := site de σ_i	D : RETOURNER (σ_i,...,(nouvelle représentation sitec) vers C_t de sitec'	
		sinon Eliminer σ_i de TCOPIES fsi Fait Si TCOPIES n'est pas vide alors attendre		Att-ACKretourner
		sinon	R : ACK-OP_K (r_1 := valeur$_1$, ...,r_{n3}:=valeur$_{n3}$) vers P_t de sitep	Etape I
		fsi fsi		
Att-ACKretourner	R: ACK-RETOURNER (σ_i)	Eliminer σ_i de TCOPIES Si tous les ACK ne sont pas arrivés alors attendre		Att-ACKretourner
		sinon	R : ACK-OP_K (r_1 := valeur$_1$, ...,r_{n3}:=valeur$_{n3}$) vers P_t de sitep	Etape I
		fsi		
Etape I	D:COPIER (σ_i, ...,classe, verrou, sitec')	Si classe = out/in-out et $\sigma_i \notin$ ZONE$_{niveau}$ de PILE alors Introduire informations de σ_i dans PILE fsi	R : ACK-COPIER (σ_i, ...,(représentation)) vers C_t de sitec'	Etape I
Etape I	D:RETOURNER (σ_i,...,(nouvelle représentation),sitec)	Réaliser l'écriture de σ_i sur PILE	R:ACK-RETOURNER (σ_i) vers C_t de sitec'	Etape I
Etape I	D:DEBUT-INT (n_i)	Si ni = niveau alors niveau := niveau +1 Introduire marque dans PILE	R: ACK-DEBINT vers P_t de sitep	Etape I
		sinon	S : PANNE-C vers P_t de sitep	Att-annval
		fsi		
Etape I	D:FIN$_j$ (n_i, S1,S2,...,Sn) pour j = 1,2	Si ni = niveau alors introduire S1, S2,...,Sn dans LC ; NV := niveau étatr := PREVALIDATION j	R : ACK-FIN$_j$ vers P_t de sitep	Etape II
		sinon	S : PANNE-C vers P_t de sitep	Att-annval
		fsi		

ETAT D'ENTREE DE C_t	REQUETE ARRIVEE	TRANSITION DECLENCHEE (ACTIONS)	REQUETES PRODUITES	ETAT DE SORTIE DE C_t
état quelconque sauf {Inactif Att-anninterbl}	D : VALIDER j pour j=1,2	Cas etatr = PREVALIDATION 1 : Réaliser "Validation Différée" : faire etatr := VALIDATION 1 ; Incorporer les intentions de ZONE$_{NV}$ à ZONE$_{NV-1}$; Eliminer marque$_{NV}$ de PILE ; niveau := NV-1 ; etatr := INITIAL. fait ----------	R:ACK-VALIDER 1 vers P_t de sitep	-- Etape I
		PREVALIDATION 2 : Réaliser "Validation Permanente " : faire etatr := VALIDATION 2. Effectuer toutes les intentions de PILE sur les objets ; ---------- Libérer chaque objet σ_i "ouvert" dans ZONE$_{NV}$; ---------- Eliminer marque$_{NV}$ de PILE ; niveau := NV-1 ; ----------	pour les intentions de destruction : S : DETRUIRE (σ_i, type) S : LIBERER (σ_i) R:ACK-VALIDER 2 vers P_t de sitep	
		si niveau ≥ 0 alors etatr := INITIAL ---------- sinon etatr := COMPLET ---------- fsi fait	---------- ----------	-- Etape I -- Etape II
		INITIAL : ----------	R:ACK-VALIDER j vers P_t de sitep--	-- Etape I
		COMPLET : ----------	R:ACK-VALIDER 2 vers P_t de sitep--	-- Etape II
		fcas		
Etape II	D : TERMINER (sitep)		R:ACK-TERMINER vers P_t de sitep	Inactif

. Traitement des situations d'exception (C_t) :

ETAT D'ENTREE DE C_t	REQUETE ARRIVEE	TRANSITION DECLENCHEE (ACTIONS)	REQUETES PRODUITES	ETAT DE SORTIE DE C_t
état quelconque sauf {Inactif Att-anninterbl}	D : ANNULER (ni)	Annuler les intentions de $ZONE_K$ pour ni ≤ K ≤ niveau ; Libérer chaque objet σ_i "ouvert" dans les zones annulées ; niveau := ni ; étatr := INITIAL	S : LIBERER (σ_i) R:ACK-ANNULER vers P_t de sitep	Etape I
Etat E quelconque sauf {Inactif}	S : INTERBLOCAGE (σ_i)	Si σ_i va être libéré (i.e. σ_i a été "ouvert" dans $ZONE_{niveau}$ et étatr = PREVALIDATION 2) alors sinon chercher la $ZONE_K$ où σ_i a été "ouvert" ; NA := K ; RI := 'vrai' fsi	S:INTERBL (NA) vers P_t de sitep	- état E - Att-anninterbl
état quelconque sauf {Inactif}	D : ANNINTERBL (ni,etatrp)	Si RI = 'vrai' et NA < ni alors attendre sinon RI := 'vrai' ; NA := ni ; Si etatrp = VALIDATION j, pour j=1,2 et etatr = PREVALIDATION j alors Réaliser Validation Différée (pour j=1) ou Validation Permanente (pour j=2) fsi Annuler les intentions de $ZONE_K$ pour NA ≤ K ≤ niveau ; Libérer chaque objet σ_i "ouvert" dans les zones annulées ; niveau := NA étatr := INITIAL ; RI := 'faux' fsi	S : LIBERER (σ_i) R:ACK-ANNINTERBL (NA) vers P_t de sitep	- Att-anninterbl - Etape I
Att-ackcopier	S:NACK-COPIER		S:PANNE-C vers P_t de sitep	Att-annval
Att-ackmodifier	S:NACK-MODIFIER			

BUS LILLE

ETAT D'ENTREE DE C_t	REQUETE ARRIVEE	TRANSITION DECLENCHEE (ACTIONS)	REQUETES PRODUITES	ETAT DE SORTIE DE C_t
Etat quelconque sauf {Inactif}	S:REINSERTION	Si RI = 'vrai' alors ----------	S:INTERBL (NA) vers P_t de sitep	Att-anninterbl
		Sinon Cas étatr = INITIAL : ---------- Annuler les intentions de $ZONE_{niveau}$; Libérer chaque objet σ_i "ouvert" dans $ZONE_{niveau}$;	S:PANNE-C vers P_t de sitep S:LIBERER (σ_i)	
		----------	----------	Att-annval
		VALIDATION 1 : Réaliser Validation Différée ; ----------	R:ACK-VALIDER 1 vers P_t de sitep	- Etape I
		VALIDATION 2 : Réaliser Validation Permanente ; ----------	R:ACK-VALIDER 2 vers P_t de sitep	
		Si niveau ≥ 0 alors -	----------	- Etape I
		sinon -	----------	- Etape II
		fsi		
		COMPLET : ----------	R:ACK-VALIDER 2 vers P_t de sitep	- Etape II
		PREVALIDATION j, pour j = 1, 2 : ---	D:QETATRP (sitec) vers P_t de sitep	Att-ackqetatrp
		fcas fsi		
Att-ackqetatrp	R:ACK-QETATRP (etatrp)	Cas étatrp = INITIAL : ---------- Annuler les intentions de $ZONE_{niveau}$; Libérer chaque objet σ_i "ouvert" dans $ZONE_{niveau}$;---- étatr := INITIAL	S:PANNE-C vers P_t de sitep S:LIBERER (σ_i)	
		----------	----------	Att-annval
		VALIDATION 1 : Réaliser Validation Différée ; ----------	R:ACK-VALIDER 1 vers P_t de sitep	- Etape I
		VALIDATION 2 : Réaliser Validation Permanente ; ----------	R : ACK-VALIDER 2 vers P_t de sitep	
		Si niveau ≥ 0 alors ----	----------	- Etape I
		sinon ----	----------	- Etape II
		fsi fcas		

ETAT D'ENTREE DE C_t	REQUETE ARRIVEE	TRANSITION DECLENCHEE (ACTIONS)	REQUETES PRODUITES	ETAT DE SORTIE DE C_t
Att-ackqetatrp	S:NACK-QETATRP		D:QETATR C (sitec) vers tous les $C_t \in LC$	Att-ackqetatrc
Att-ackqetatrc	R:ACK-QETATRC (etatrc, ni)	Cas étatrc =		
		VALIDATION 1 : Réaliser validation Différée;	R:ACK-VALIDER 1 vers P_t de sitep	- Etape I
		VALIDATION 2 ou COMPLET : Réaliser Validation Permanente ;	R:ACK-VALIDER 2 vers P_t de sitep	
		Si niveau ≥ 0 alors		- Etape I
		sinon		- Etape II
		fsi		
		INITIAL : si ni = niveau		
		alors	S:PANNE-C vers P_t de sitep	
		Annuler les intentions de $ZONE_{niveau}$; Libérer chaque objet σ_i "ouvert" dans $ZONE_{niveau}$;	S:LIBERER (σ_i)	
		étatr := INITIAL		Att-annval
		sinon		
		Cas étatr =		
		PREVALIDATION 1 : Réaliser Validation Différée ;	R:ACK-VALIDER 1 vers P_t de sitep	- Etape I
		PREVALIDATION 2 : Réaliser Validation Permanente ;	R:ACK-VALIDER 2 vers P_t de sitep	- Etape I
		fcas		
		fsi		
		PREVALIDATION 1 ou PREVALIDATION 2 :		
		Si tous les ack ne sont pas arrivés alors attendre les ack		Att-ackqetatrc
		sinon attendre la récupération de P_t		Att-annval
		fsi		
		fcas		
Att-ackqetatrc	S:NACK-QETATRC	Si tous les autres ack ne sont pas arrivés alors attendre les ack		Att-ackqetatrc
		sinon attendre la récupération de P_t		Att-annval
		fsi		
état E quelconque sauf (Inactif)	D : QETATRC (sitec)		R:ACK-QETATRC (étatr, niveau) vers C_t de sitec	état E

BUS LILLE

2.5 Processus Moniteur.

Les tables suivantes résument les actions réalisées par le processus Moniteur de chaque site du système, à la réception des différentes requêtes possibles. On considère 2 groupes de requêtes : les requêtes externes, délivrées par le sous-système de transport, provenantes des processus situés sur d'autres sites, et les requêtes internes provenant des processus locaux (Producteurs ou Consommateurs de diverses transactions). Pour chaque requête arrivée à un processus Moniteur, les tables indiquent les actions réalisées par ce processus et les requêtes produites vers d'autres processus, externes ou locaux (pour les transitions d'un processus Moniteur, on considère un état unique d'entrée et de sortie : l'état "actif" prêt à traiter n'importe quelle requête). On utilisera dans les tables les abréviations DEM_p, DEM_c, REP_p et REP_c pour denoter les ensembles suivants :

DEM_p : demandes originées par un Producteur
= {DEBUT-TR, OP_K, DEBUT-INT, FIN1, FIN2, VALIDER1, VALIDER2, TERMINER, ANNULER, ANNINTERBL }

DEM_c : demandes originées par un Consommateur
= {COPIER, MODIFIER, QETATRP, QETATRC}

REP_p : réponses destinées à un Producteur
= {ACK-DEBTR, ACK-OP_K, ACK-DEBINT, ACK-FIN1, ACK-FIN2, ACK-VALIDER1, ACK-VALIDER2, ACK-TERMINER, ACK-ANNULER, ACK-ANNINTERBL}

REP_c : réponses destinées à un Consommateur
= {ACK-COPIER, ACK-MODIFIER, ACK-QETATRP, ACK-QETATRC}

SIG_p : signaux destinés à un Producteur
= {INTERBL, PANNE-C, NACK}

SIG_c : signaux destinés à un Consommateur
= {NACK-COPIER, NACK-MODIFIER, NACK-QETATRP, NACK-QETATRC}

. Traitement des requêtes externes par le Moniteur:

REQUETE EXTERNE ARRIVEE	TRANSITION DECLENCHEE (ACTIONS)	REQUETES PRODUITES VERS D'AUTRES PROCESSUS	
		externes	locaux
D : DEBUT-TR (t, ...) concernant une transaction t	Introduire information de t dans TCON : t ; Act_t := 'vrai' ; $FATT_t$:= Λ (vide)		D : DEBUT-TR (t,...) vers C_t
D : dem pour dem = DEBUT-INT/FIN$_j$ (j = 1, 2) / VALIDER j (j = 1, 2) / MODIFIER concernant une transaction t	Si il y a une autre demande de t en exécution ou en attente alors Introduire dem dans $FATT_t$ sinon Act_t := 'vrai' fsi	----------	D : dem vers C_t
D : OP_K ($\sigma_1,\ldots,\sigma_{n1}$, $c_1,\ldots,c_{n2}$, r_1 = ?,..., r_{n3} = ?) concernant une transaction t	Si il y a une atre demande de t en exécution ou en attente alors Introduire OP_K (...) dans $FATT_t$ sinon Réaliser "Examen Verrous" pour OP_K : A := 'vrai' ; i := 1 ; Tant que A = 'vrai' et i ≤ n3 faire Si σ_i est local alors V := verrou sur σ_i exigé par OP_K ; Si t possède déjà V sur σ_i ou un autre verrou plus fort que V alors B := 'faux' sinon B := 'vrai' fsi ; Si B = 'vrai' alors Si V n'entraîne aucun conflit de t avec une autre trans. t' alors Introduire V dans TV ; Eliminer les verrous de t sur σ_i moins forts que V sinon Introduire OP_K(...) dans $FATT_t$; si t<t' alors fsi A := 'faux' fsi fsi fsi i := i+1 fait ; si i > n3 alors Act_t := 'vrai' fsi fsi	---------- ----------	S : INTERBLOCAGE (σ_i) vers chaque $C_{t'}$ possible D : OP_K (...) vers C_t

REQUETE EXTERNE ARRIVEE	TRANSITION DECLENCHEE (ACTIONS)	REQUETES PRODUITES VERS D'AUTRES PROCESSUS	
		externes	locaux
D : COPIER (σ_i,...,V,...) concernant t	Si il y a une autre demande de t en exécution ou en attente alors Introduire COPIER (...) dans $FATT_t$ sinon Réaliser "Examen Verrous" pour COPIER : Si t possède déjà V sur σ_i ou un autre verrou plus fort que V alors Act := 'vrai' ----------	----------	D : COPIER (...) vers C_t
	sinon Si V n'entraîne aucun conflit de t avec une autre trans. t' alors Introduire V dans TV ; Eliminer les verrous de t sur σ_i moins forts que V ; Act_t := 'vrai' -------	----------	D : COPIER (...) vers C_t
	sinon Introduire COPIER (...) dans $FATT_t$; Si t < t' alors ------	----------	S : INTERBLOCAGE (σ_i) vers chaque $C_{t'}$ possible
	fsi fsi fsi fsi		
D : TERMINER (sitep) concernant t	Si t ∈ TCON alors ---------- sinon ---------- fsi	---------- R:ACK-TERMINER vers sitep	D :TERMINER (sitep vers C_t
D : dem pour dem = ANNULER / ANNINTERBL concernant t	$FATT_t$:= Λ (éliminer demandes en attente)		D : dem vers C_t
D : QETATRP (sitec') concernant t			D : QETATRP (sitec') vers P_t
D : QETATRP (sitec') concernant t			D : QETATRC (sitec') vers C_t
R : rep pour rep ∈ REP_P concernant t			R : rep vers P_t
R : rep pour rep ∈ REP_C concernant t			R : rep vers C_t
S : signal pour signal ∈ SIG_P concernant t			S : signal vers P_t
S : Signal pour signal ∈ SIG_C concernant t			S : signal vers C_t
S : REINSERTION			S : REINSERTION vers tous les processus locaux
S : JETON (ticket)	Tant que FID n'est pas vide faire Extraire un id-utilisateur de FID t := ticket Introduire t dans TPRO ---------- ticket := ticket + 1 fait ----------	---------- S:JETON(ticket) vers site successeur	S : DEBUT-TR(t,...) vers P_t
S : CREER (σ_i, type, site)	Répertorier σ_i dans TOG		
S : DETRUIRE (σ_i, type)	Enlever σ_i de TOG		

BIIS LILLE

. Traitement des requêtes internes par le Moniteur:

REQUETE INTERNE ARRIVEE	TRANSITION DECLENCHEE (ACTIONS)	REQUETES PRODUITES VERS D'AUTRES PROCESSUS	
		externes	locaux
C : DEBUT-TR	Introduire id-utilisateur dans FID		
D : dem pour dem $\in$ $DEM_P \cup DEM_C$, destinée à la transaction t, site S		D : dem vers site S	
R : rep pour rep $\in$ $REP_P \cup REP_C$ sauf {ACK-TERMINER} destiné à la transaction t, site S	---------- Si $FATT_t$ non vide alors Examiner prochaine demande de t : dem := tête de $FATT_t$; Cas dem = DEBUT-INT/FIN j/VALIDER j/MODIFIER : enlever dem de $FATT_t$; ---------- OP_K ($\sigma_1,\ldots,\sigma_{n1},c_1,\ldots,c_{n2}$, $r_1 = ?, \ldots, r_{n3} = ?$) : Réaliser "Examen Verrous" pour OP_k ; COPIER (σ_i, ..., V, ...) : Réaliser "Examen Verrous" pour COPIER ; ---------- fcas fsi	R : rep vers site S ---------- ---------- ----------	 D : dem vers C_t si non-conflit : D : dem vers C_t Si non-conflit : D : dem vers C_t
R : ACK-TERMINER destinée à la transaction t, site S	Enlever t de TCON	R:ACK-TERMINER vers site S	
S : signal pour signal = INTERBL/PANNE-C destiné à la transaction t, site S		S = signal vers site S	
S : LIBERER (σ_i, t)	Enlever de TV les verrous de t sur σ_i ; Pour chaque t' $\in$ TV (t' $\neq$ t) faire Examiner prochaine demande de t' (résolution d'attentes de verrous sur σ_i pour dem := tête de $FATT_{t'}$) ---------- fait	----------	Si non conflit : D : dem vers C_t.
S : CREER (σ_i, type, site)	Répertorier σ_i dans TOG	S : CREER type,site) vers tous les sites	
S : DETRUIRE (σ_i, type)	Enlever σ_i de TOG	S:DETRUIRE (σ_i, type, site) vers tous les sites	

BUS LILLE

3. LIGNES GENERALES D'UNE IMPLEMENTATION EN ADA.

Dans cette dernière partie, nous voulons proposer quelques lignes sommaires d'une implémentation en ADA pour le modèle généralisé de contrôle de cohérence présenté tout au long de ce chapitre.

Le langage de programmation ADA [BREND 81 ; ICHBI 79] s'avère très approprié pour une telle implémentation par les raisons suivantes :

- ADA offre le concept de type abstrait : il permet de spécifier des types d'objets et des opérations manipulant ces objets,

- ADA permet d'exprimer des contraintes de synchronisation entre processus concurrents,

- ADA va être largement diffusé dans les années à venir.

Les notions "paquetage" et "tâche" du langage ADA, nous semblent fondamentales pour une éventuelle implémentation de notre modèle généralisé de contrôle de cohérence :

La notion *paquetage* (*package*) va permettre la réalisation des objets du système : pour un type d'objets, on utilisera la partie *"interface"* d'un paquetage pour spécifier les composants de ces objets (tels qu'ils sont connus par l'utilisateur) et les opérations qui permettent de les manipuler ; on utilisera la partie *"corps"* du paquetage pour décrire le code des opérations (du type d'objets correspondant).

La notion *tâche* (*task*) va permettre la réalisation des processus du système : pour un processus, on utilisera la partie *"interface"* d'une tâche pour spécifier la liste des points de rendez-vous entre le processus en question et les autres processus du système ; cette liste indiquera en fait tous les requêtes possibles que le processus peut recevoir (i.e. les *entrées*) avec leurs paramètres. On utilisera la partie *"corps"* de la tâche pour décrire les actions réalisées par le

processus à la réception des diverses requêtes et les conditions d'activation de ces actions.

En utilisant le même exemple de gestion de comptes bancaires, présenté dans la partie 1.3 de ce chapitre, nous illustrons par la suite l'utilisation des notions paquetage et tâche pour implémenter les types d'objets du système et les processus Producteurs, Consommateurs et Moniteur de chaque site :

- *Interfaces des types d'objets clients et comptes :*

```
package CLIENTS is
    type INFORMATION is record DATE-NAISSANCE : INTEGER,
                               IDENTITE : INTEGER,
                               OCCUPATION : STRING
                        end record ;
    type CLIENT      is record NOM : STRING,
                               ADRESSE : STRING,
                               INFO-PERSONNELLE : INFORMATION,
                               CODE-COMPTE : STRING
                        end record ;
    procedure NOUVEAU-CLIENT (CL : out CLIENT, NM : in STRING, ADR :
                              in STRING, DATE : in INTEGER, ID : in
                              INTEGER, OCC : in STRING) ;
    procedure CHANGER-AD (CL : out CLIENT, ADR : in STRING) ;
    procedure CONSULTER-INFO (CL : in CLIENT, DATE : out INTEGER,
                              ID : out INTEGER, OCC : out STRING) ;
    procedure CONSULTER-CODE (CL : in CLIENT, CODE : out STRING) ;
    procedure RETIRER (CL : in out CLIENT) ;
end CLIENTS ;
```

```
with CLIENTS

package COMPTE is
    type COMPTE is record CODE : STRING,
                          DATE-OUVERTURE : INTEGER,
                          MONTANT : REAL
                   end record ;
    procedure OUVRIR (C : out COMPTE, CO : in STRING, DATE : in STRING,
                      SOMME : in REAL, N : in INTEGER, CLS : out array
                     (1..N) of CLIENT) ;
    procedure CREDITER (C : in out COMPTE, SOMME : in REAL) ;
    procedure DEBITER (C : in out COMPTE, SOMME : in REAL) ;
    procedure TRANSFERER (C : in out COMPTE, C2 : in out COMPTE,
                          SOMME : in REAL) ;
    procedure SOLDE (C : in COMPTE, SOMME : out REAL) ;
    procedure FERMER (C : in out COMPTE) ;
end COMPTE ;
```

- Interface d'un processus Producteur :

```
with CLIENTS, COMPTES
task P is
    entry DEBUT-TR (T : in INTEGER, ...) ;
    entry NOUVEAU-CLIENT (CL : out CLIENT, ...) ;
    entry CHANGER-AD (CL : out CLIENT, ...) ;
    ...
    entry OUVRIR (C : out COMPTE,...) ;
    entry CREDITER (C : in out COMPTE,...) ;
    ...
    entry DEBUT-INT ;
    entry FIN 1 ; entry FIN 2 ;
    entry ANNULER ;
    entry INTERBL (NI : in INTEGER) ;
    entry PANNE-C ;
    ...
    entry ACK-DEBINT ( SITE: in INTEGER) ;
    entry ACK-FIN 1 ; entry ACK-FIN2 ;
    ...
 end P ;
```

Cette interface indique toutes les requêtes qu'un processus producteur peut recevoir : des commandes venant de l'utilisateur, ou des demandes, réponses ou signaux venant du Moniteur.

De manière similaire on indiquera dans l'interface d'un processus Consommateur (C) toutes les demandes, réponses et signaux qu'il peut recevoir de son Moniteur.
Dans l'interface d'un processus Moniteur (M) on indiquera toutes les requêtes qu'il peut recevoir des processus Producteurs et Consommateurs locaux ou externes.

- *Corps d'un processus Producteur :*

Le corps d'un processus indiquera le code à exécuter lors de l'arrivée de chaque requête et les conditions d'activation de ces traitements. Ainsi par exemple, le corps d'un Producteur peut présenter la forme suivante :

```
with CLIENTS, COMPTES
task body P is
... déclarations des tables propres à un Producteur : DES et LC.

begin
      accept DEBUT-TR (T : in INTEGER,...) do
            DES.TR := T ; ETATR := INITIAL ;
      end   DEBUT TR ;
      ACTIF := TRUE ;
      while ACTIF = TRUE loop
        Select
          accept NOUVEAU-CLIENT (CL : out CLIENT,...) do ... end NOUVEAU-CLIENT ;
        or
          accept CHANGER-AD (CL : out CLIENT,...) do
                ...
                SITEC := ... déterminer dans TOG le site de
                         ... résidence du paramètre CL.
                M. CHANGER-AD (CL,...,SITEC);
                         ... envoyer la demande au Moniteur M
          end CHANGER-AD ;
        ...
        or accept FIN_2 do
                for I in 1 .. NC loop                ... NC est le nombre de consommateurs
                  SITEC := LC(I) ;
                  M. FIN2 (NIVEAU, LC, SITEC) ;      ... diffusion de la demande FIN2
                end loop ;
                for I in 1 .. NC loop
                  accept ACK-FIN 2 ;
                end loop ;
                ETATR := VALIDATION 2 ;
                for I in 1 .. NC loop                ... diffusion de la demande VALIDER 2
                  SITEC := LC(I) ;
                  M. VALIDER 2 (SITEC) ;
                end loop ;
                for I in 1 .. NC loop
                  accept ACK-VALIDER 2 ;
                end loop ;
                NIVEAU := NIVEAU -1 ;
                if NIVEAU > 0
                  then ETATR := INITIAL
                  else do ETATR := COMPLET ;
                          for I in 1 .. NC loop      ... diffusion de TERMINER
                            SITEC := LC(I) ;
                            M. TERMINER (SITEC) ;
                          end loop ;
                          for I in 1 .. NC loop accept ACK-TERMINER; end loop;
                          ACTIF := FALSE;
                       end if
          end FIN 2
        or accept ANNULER do ... end ANNULER
        ...
        end select ;
      end loop ;
end P ;
```

Dans le corps d'un processus Consommateur, le traitement de quelques requêtes entrainera l'invocation des opérations des types d'objets CLIENTS et COMPTES. L'implémentation de ces opérations (indiquée dans les parties corps des types d'objets CLIENTS et COMPTES) entrainera la consultation de la Table d'Objets Globaux (TOG) et la manipulation des tables propres au Consommateur appelant : c'est à dire la PILE d'intentions, la Table de copies d'objets externes (TCOPIES), etc.
En fait, c'est seulement lors de la réception de la demande VALIDER 2 que le consommateur mettra à jour des objets CLIENTS et COMPTES (catalogués dans TOG), en rendant effectives les intentions de modification accumulées dans sa PILE.

CONCLUSIONS GENERALES

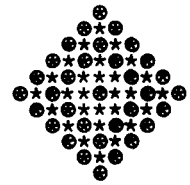
Par rapport aux systèmes informatiques centralisés, les systèmes répartis offrent une meilleure adaptation aux structures des organisations et aux besoins des utilisateurs, mais ils posent des problèmes de contrôle de cohérence que nous avons caractérisé dans le Chapitre I de ce travail. En vue de résoudre ces problèmes et d'offrir en même temps diverses options de contrôle à l'utilisateur interactif des systèmes répartis (options délimitées dans le Chapitre II), nous avons proposé dans les Chapitres IV et V un modèle de contrôle de cohérence basé sur l'utilisation de transactions emboîtées et sur la libération graduelle des ressources pendant le déroulement d'une transaction.

Pour l'utilisateur interactif, le modèle de transactions emboîtées présente quelques avantages par rapport au modèle conventionnel de contrôle de cohérence basé sur la réalisation de transactions atomiques (modèle décrit dans le Chapitre III). Entre ces avantages nous pouvons mentionner la meilleure performance en degré réel de concurrence et en temps de réponse et les options de l'utilisateur pour sauvegarder des états de sa transaction, pour annuler une partie de sa transaction et pour protéger ses opérations des annulations déclenchées par le système lors de défaillances (ou de risques d'interblocage).

La généralisation du modèle de transactions emboîtées que nous avons fait dans le Chapitre V, permet son utilisation dans des applications transactionnelles diverses, manipulant des objets de plusieurs types. Pour chaque application, le modèle prévoit la définition d'un ensemble de verrous qui soit approprié pour favoriser la concurrence entre transactions.

Quelques simplifications ont été faites dans le modèle de contrôle de cohérence proposé : pour une implémentation réelle il faudrait

tenir compte d'autres problèmes importants des systèmes répartis tels la protection des données confidentielles et l'évaluation et la décomposition des requêtes (afin d'assigner à chaque opération le site "optimal" pour son exécution). Il serait aussi intéressant de prendre en compte d'autres facilités conversationnelles souhaitées par l'utilisateur et d'étendre le domaine d'application du modèle de transactions emboîtées au cas des données dupliquées.



BIBLIOGRAPHIE

[ALSBE 76] : P.A. Alsberg, G.G. Belford, J.D. Day, E. Grapa,
"Multi-copy Resiliency Techniques",
CAC Document n° 202, University of Illinois-Urbana, May 1976,
(Réinprimé dans [IEEE 78]).

[ANDRA 80] : J. Andrade, F. Andre, R. Balter, G. Bogo, J.C. Chupin,
P. Decitre,
"Scot : A System for the Consistent Cooperation of Transactions",
Document du Centre Scientifique CII-Honeywell-Bull et IMAG,
Grenoble, 1980.

[BANIN 79] : J.S. Banino, C. Keiser, H. Zimmermann,
"Synchronisation for distributed systems using a single Broadcast channel",
Réunion du Groupe AFCET-TTI : Architecture des machines et systèmes informatiques, Lille, Publication n° 9 du Laboratoire d'Informatique de l'Université de Lille I, pp. 5-30,
Novembre 1979.

[BERNS 78] : P.A. Bernstein, J.B. Rothnie, N. Goodman, C.A. Papadimitrou,
"The concurrency control mechanism of SDD-1 : A system for distributed Databases (The fully redudant Case)",
IEEE Transactions on Soltware Eng., Vol. SE-4, n° 3, pp. 154-168, May 1978.

[BLASG 81] : M.W. Blasgen, M.M. Astrahan, D.D. Chamberlin, J.N. Gray et al.,
"System R : An architectural overview",
IBM System Journal of Research and Development, Vol. 20, n° 1, pp. 41-62, 1981.

[BOUCH 81] : P. Bouchet, A. Chesnais, J.M. Feuvre, G. Jomier, A. Kurinckx,
"PEPIN : An experimental Multi-micro computer data base Management System",
Proc. of the 2nd International Conference on Distributed Computing Systems, Paris, pp. 211-217, April 1981.

[BREND 81] : R.F. Brender, I.R. Nassi,
"What is ADA ?",
IEEE Computer, Vol. 14, n° 6, pp. 17-24, June 1981.

[CARRE 80] : C. Carrez,
"Structuration des programmes de gestion dans un environnement conversationnel",
RAIRO Informatique/Computer Science, Vol. 14, n° 2, 1980.

[CHAMB 76] : D.D. Chamberlin, M.M. Astrahan, K.P. Eswaran et al.,
"SEQUEL 2 : A unified approach to Data Definition, Manipulation and Control",
IBM Journal of Research and Development, Vol. 20, n° 6, pp. 560-575, Nov. 76.

[CHAMP 77] : G.A. Champine,
"Six approaches to Distributed Data Bases",
Datamation, May 1977, (Réimprimé dans [IEEE 78]).

[CHAMP 80] : G.A. Champine,
"Distributed Computer Systems : Impact on management, design and analysis",
North-Holland Publishing, 1980.

[CORNA 81] : Groupe CORNAFION,
"Les Systèmes Informatiques Répartis : Concepts et techniques",
Dunod, Ed., 1981.

[DELAT 79] : E. Delattre,
"Contribution à la répartition d'un système à structure de domaines sur un réseau local de micro-ordinateurs faiblement couplés",
Thèse Docteur-Ingénieur, Université de Lille I, Décembre 1979.

[ELLIS 77] : C.A. Ellis,
"Consistency and Correctness of Duplicate Database Systems",
Proc. of 6th ACM Symposium on Operating Systems Principles, pp. 67-84, Nov. 1977.

[ENSLO 78] : P.H. Enslow Jr.,
"What is a Distributed Data Processing Sustem ?",
IEEE Computer, Vol. 11, n° 1, pp. 13-21, Janv. 1978.

[ESWAR 76] : K.P. Eswaran, J.N. Gray, R.A. Lorie, I.L. Traiger,
"The notions of consistency and predicate locks in a database system",
CACM, Vol. 19, n° 11, pp. 624-633, Nov. 1976.

[FAULL 80] : B. Faulle G. Thomas,
"L'analyse des systèmes conversationnels",
Zero-Un Informatique, n° 137, Janvier-Février 1980.

[FRANK 79] : C. Franky-Toro,
"Contrôle de cohérence dans les systèmes intéractifs",
Rapport de D.E.A. en Traitement de l'Information, Laboratoire d'Informatique de l'Université de Lille I, Oct. 1979.

[FRANK 81] : C. Franky-Toro, C. Carrez,
"A Model of Control Consistency adapted for the interactive user of a distributed system",
Publication n° 15, Laboratoire d'Informatique de l'Université de Lille I, Feb. 1981.

[GRAY 75] : J.N. Gray, R.A. Lorie, G.R. Putzolu, I.L. Traiger,
"Granularity of locks and degrees of consistency in a shared data base",
R.J. 1654, IBM Research Laboratory, San-José, California, Sept. 1975.

[HAYES 81] : P. Hayes, E. Ball, R. Reddy,
"Breaking the Man-machine Communication Barrier",
IEEE Computer, Vol. 14, n° 3, pp. 19-30, March 1981.

[HERMA 79] : D. Herman, J.P. Verjus,
"An algorithm for maintaining the consistency of multiples copies",
Proc. 1st International Conférence on Distributed Computing Systems, Hunstville, Alabama, pp. 625, Oct. 1979.

[ICHBI 79] : J.D. ICHBIAH, J.C. Heliard, O. Roubine, J.G.P. Barnes, B. Krieg-Brueckner, B.A. Wichmann,
"Reference Manual for the ADA programming Language",
SIGPLAN Notices, Vol. 14, n° 6, part A, June 1979.

[IEEE 78] : IEEE Computer Society,
"Tutorial : Distributed Data Base Management",
Edited, by P.A. Bernstein, J.B. Rothnie and D.W. Shipman,
IEEE Catalog n° EHO 141-2, 1978.

[ISRAE 78] : J.E. Israel, J.G. Mitchell, H.E. Sturgis,
"Separating Data from functions in a distributed file system",
INRIA, Second Colloque International sur les systèmes d'exploitation, Oct. 1978.

[JAMES 80] : E.B. James,
"The user interface",
The Computer Journal, Vol. 23, n° 1, pp. 25-28, Feb. 1980.

[JOUVE 80] : M. Jouve, C. Parent, S. Spaccapietra,
"Principes des systèmes de gestion de Bases de Données Réparties",
RAIRO Informatique/Computer Science, Vol. 14, n° 3, pp. 253-274., 1980.

[KIM 79] : K.H. Kim,
"Error, detection, Reconfiguration and Recovery in Distributed Processing Systems",
Proc. of 1st International Conference on Distributed Computing Systems, Hunstville, Alabana, pp. 285-295, Oct. 1979.

[LAFUE 78] : J.M. Lafuente, D. Gries,
"Language facilities for programming user-computer dialogues",
IBM Journal of Research and development, Vol. 22, n° 2,
pp. 145-158, March 1978.

[LAMPO 78] : L. Lamport,
"Time, clocks and the ordering of events in a distributed system",
CACM, Vol. 21, n° 7, pp. 558-565, July 1978.

[LAMPS 77] : B. Lampson, H. Sturgis,
"Crash Recovery in a distributed Data Storage system",
Computer Science Laboratory Report, Xerox Palo Alto Research Center, California, 1977.

[LANCI 78] : D. Lanciaux,
"Mécanismes de structuration des systèmes à capacités",
Thèse d'Etat, Université de Lille I, Oct. 1978.

[LEBIH 76] : J. Le Bihan,
"La répartition des données dans les réseaux informatiques",
Congrès AFCET : Panorama de la Nouveauté Informatique en France,
pp. 55-65, Nov. 1976.

[LEBIH 80] : J. Le Bihan, C. Esculier, G. Le Lann, W. Litwin, G. Gardarin, S. Sedillot, L. Treille,
"SIRIUS : A french nation wide project on distributed data Bases",
Proc. 6th International Conference on VLDB, Montreal, 1980.

[LELAN 78] : G. Le Lann,
"Coherent Data sharing in failure-tolerant distributed processing systems",
INRIA, Projet SIRIUS, Ref. ALL-I-013, Janv. 1978.

[LELAN 79a] : G. Le Lann,
"An overview of distributed Control techniques",
SIRIUS-INRIA : Seminaire sur les systèmes répartis à partage de données, Mai 1979.

[LELAN 79b] : G. Le Lann, J. Boudenant, S. Sedillot,
"Principes de SCORE : système de cohérence et Résilience dans la réalisation SIRIUS-DELTA",
INRIA, Projet SIRIUS, Ref. INT.I.009, Août 1979.

[LELAN 79c] : G. Le Lann,
"An analysis of different approaches to distributed computing",
Proc. 1st International Conference on Distributed Computing Systems, Huntsville, Alabama, pp. 222-232, Oct. 1979.

[LELAN 81] : G. Le Lann,
"A distributed system for real-time transaction processing",
IEEE Computer, Vol. 14, n° 2. Feb. 1981.

[LISKO 74] : B. Liskov, S. Zilles,
"Programming with abstract data types",
ACM SIGPLAN, Vol. 9, n° 4, pp. 50-59, Avril 1974.

[MACCH 79] : C. Macchi, J.F. Guilbert,
"Téléinformatique",
DUNOD, Ed., 1979.

[MARTI 73] : J. Martin,
"Design of Man-computer dialogues",
Prentice-Hall, Englewood Cliffs, New-Jersey, 1973.

[MENAS 78] : D.A. Menasce, G.J. Popek, R.R. Muntz,
"Centralized and hierarchical locking in distributed data-bases",
Proc. of the ACM/SIGMOD International Conference on the Management of Data, Austin, Texas, 1978. (Réimprimé dans "Tutorial : Distributed Data Base Management" IEEE Catalog n° EHO-141-2, pp. 178-195, 1978).

[MENAS 79] : D.A. Menasce, R.R. Muntz,
"Locking and deadlock detection in distributed databases",
IEEE Tr. on Software Engineering, Vol. SE-5, n° 3, May 1979.

[MEYER 78] : D. Meyer, L. Dargent,
"Une base de données répartie pour une organisation étoilée",
Congrès de l'AFCET : Théorie et techniques de l'Informatique, Tome 2, pp. 131-139, Gif sur Yvette, Nov. 1978.

[MORGA 77] : H.L. Morgan, K.D. Levin,
"Optimal Program and Data locations in computer Networks",
CACM, Vol. 20, n° 5, pp. 315-322, May 1977.

[MUKER 79] : J. Mukerji, R.B. Kieburtz,
"A distributed file system for a hierarchical Multicomputer",
Proc. of 1st International Conference on distributed Computing Systems, Hunstville, Alabama, pp. 448-458, Oct. 1979.

[NOE 73] : J.D. Noe, G.J. Nutt,
"Macro E-Nets for Representation of Parallel Systems",
IEEE transactions on Computers, Vol. C-22, n° 8, pp. 718-727, August 1973.

[PAXTO 79] : W.H. Paxton,
"A client-based transaction system to maintain data integrity",
Proc. of 7th ACM Symposium on Operating Systems principles, Dec. 1979.

[RANDE 75] : B. Randell,
"System structure for software Fault tolerance",
IEEE transactions on Software Engineering, Vol. SE-1, n° 2, pp. 220-232, June 1975.

[REED 79] : D.P. Reed, R.K. Kanodia,
"Synchronization with Event counts and Sequencers",
CACM, Vol. 22, n° 2, Feb. 1979.

[ROLIN 80] : P. Rolin,
"Exploitation d'un modèle d'evaluation de Nutt au cours de la vie d'un produit",
INRIA, Projet SIRIUS, 1980.

[ROSEN 78] : D.J. Rosenkrantz, R.E. Stearns, P.M. Lewis,
"System level concurrency control for distributed database systems",
ACM Transactions on Database Systems, Vol. 3, n° 2, pp. 178-198, Juin 1978.

[ROTHN 77] : J.B. Rothnie, N. Goodman,
"A survey of research and development in distributed database management",
Proc. of the 3rd International Conference on Very large Data Bases, (VLDB), Oct. 1977, (Réimprimé dans "Tutorial : Distributed Data Base Management", IEEE Catalog n° EHO 141-2, pp. 30-44, 1978).

[RUSSE 77] : D.L. Russell,
"Process Backup in Producer-consumer systems",
Proc. 6th ACM Symposium on Operating Systems principles, pp. 151-157, Nov. 1977.

[SALES 80] : A. Sales,
"Utitisation de la notion de type abstrait générique en Bases de Données Relationnelles",
Actes du Congrès de l'AFCET-Informatique, pp. 565-575, Nancy, Nov. 1980.

[SCOT 81a] : Groupe Scot,
"Description du protocole SCOT",
R.R. Scot n° 9, Centre Scientifique CII-Honeywell Bull et IMAG, Grenoble, Janv. 1981.

[SCOT 81b] : Groupe Scot,
"Selection of a commitment and recovery mechanism for a distributed transactional system",
R.R. Scot n° 14, Centre Scientifique CII-Honeywell Bull et IMAG, Grenoble, Janv. 1981.

[SEGUI 79] : J. Seguin, G. Sergeant, P. Wills,
"A majority consensus algorithm for the consistency of duplicated and distributed information",
Proc. 1st International Conference on Distributed Computing Systems, Huntsville, Alabama, Oct. 1979.

[STERL 74] : T.D. Sterling,
"Guidelines for humanizing computerized Information Systems : a report from Stanley House",
CACM, Vol. 7, n° 11, pp. 609-613, Nov. 1974.

[STONE 79] : M. Stonebraker,
"Concurrency control and consistency of multiple copies of Data in Distributed INGRES",
IEEE Transactions on Software Eng., Vol. SE-5, n° 3, May 1979.

[THOMA 78] : R.N. Thomas,
"A solution to the concurrency control problem for multiple Data Bases",
COMPCON 78, Spring, 1978, (Réimprimé dans "Tutorial Distributed Data Base Management", IEEE Catalog, n° EHO 141-2, 1978).

[WILLS 79] : P. Wills,
"Etude et comparaison d'algorithmes de maintien de la cohérence dans les Bases de Données Réparties",
Thèse de Docteur-Ingénieur, I.N.P. Grenoble, Nov. 1979.

[WINOG 79] : T. Winograd,
"Beyond Programming Languages",
CACM, Vol. 22, n° 7, July 1979.

[ZIMME 81] : H. Zimmermann, J.S. Banino, A. Caristan, M. Guillemont, G. Morisset,
"Basic concepts for the support of distributed systems : the CHORUS approach",
Proc. 2nd International Conference on Distributed Computing Systems, Paris, pp. 60-66, April 1981.