Maintien de la cohérence dans les systèmes d'information répartis

Roland Balter

THESE

présentée à

l' Université Scientifique et Médicale de Grenoble
et à
l' Institut National Polytechnique de Grenoble

pour obtenir le grade de
DOCTEUR ES SCIENCES
«Mathématiques»

par

Roland BALTER

MAINTIEN DE LA COHERENCE DANS LES SYSTEMES D'INFORMATION REPARTIS.

Thèse soutenue le 2 Juillet 1985 devant la commission d'examen.

J. MOSSIERE	Président
M. ADIBA	
J. FERRIE	
G. GARDARIN	Examinateurs
S. KRAKOWIAK	
G. ROUCAIROL	

UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE

Année universitaire 1982-1983

Président de l'Université : M. TANCHE

MEMBRES DU CORPS ENSEIGNANT DE L'U.S.M.G.

(RANG A)

SAUF ENSEIGNANTS EN MEDECINE ET PHARMACIE

PROFESSEURS DE 1ère CLASSE

ARNAUD Paul	Chimie organique
ARVIEU Robert	Physique nucléaire I.S.N.
AUBERT Guy	Physique C.N.R.S.
AYANT Yves	Physique approfondie
BARBIER Marie-Jeanne	Electrochimie
BARBIER Jean-Claude	Physique expérimentale C.N.R.S. (labo de magnétisme)
BARJON Robert	Physique nucléaire I.S.N.
BARNOUD Fernand	Biosynthèse de la cellulose-Biologie
BARRA Jean-René	Statistiques - Mathématiques appliquées
BELORISKY Elie	Physique
BENZAKEN Claude (M.)	Mathématiques pures
BERNARD Alain	Mathématiques pures
BERTRANDIAS Françoise	Mathématiques pures
BERTRANDIAS Jean-Paul	Mathématiques pures
BILLET Jean	Géographie
BONNIER Jean-Marie	Chimie générale
BOUCHEZ Robert	Physique nucléaire I.S.N.
BRAVARD Yves	Géographie
CARLIER Georges	Biologie végétale
CAUQUIS Georges	Chimie organique
CHIBON Pierre	Biologie animale
COLIN DE VERDIERE Yves	Mathématiques pures
CRABBE Pierre (détaché)	C.E.R.M.O.
CYROT Michel	Physique du solide
DAUMAS Max	Géographie
DEBELMAS Jacques	Géologie générale
DEGRANGE Charles	Zoologie
DELOBEL Claude (M.)	M.I.A.G. Mathématiques appliquées
DEPORTES Charles	Chimie minérale
DESRE Pierre	Electrochimie
DOLIQUE Jean-Michel	Physique des plasmas
DUCROS Pierre	Cristallographie
FONTAINE Jean-Marc	Mathématiques pures
GAGNAIRE Didier	Chimie physique

.../...

GASTINEL Noël	Analyse numérique - Mathématiques appliquées
GERBER Robert	Mathématiques pures
GERMAIN Jean-Pierre	Mécanique
GIRAUD Pierre	Géologie
IDELMAN Simon	Physiologie animale
JANIN Bernard	Géographie
JOLY Jean-René	Mathématiques pures
JULLIEN Pierre	Mathématiques appliquées
KAHANE André (détaché DAFCO)	Physique
KAHANE Josette	Physique
KOSZUL Jean-Louis	Mathématiques pures
KRAKOWIAK Sacha	Mathématiques appliquées
KUPTA Yvon	Mathématiques pures
LACAZE Albert	Thermodynamique
LAJZEROWICZ Jeannine	Physique
LAJZEROWICZ Joseph	Physique
LAURENT Pierre	Mathématiques appliquées
DE LEIRIS Joël	Biologie
LLIBOUTRY Louis	Géophysique
LOISEAUX Jean-Marie	Sciences nucléaires I.S.N.
LOUP Jean	Géographie
MACHE Régis	Physiologie végétale
MAYNARD Roger	Physique du solide
MICHEL Robert	Minéralogie et pétrographie (géologie)
MOZIERES Philippe	Spectrométrie - Physique
OMONT Alain	Astrophysique
OZENDA Paul	Botanique (biologie végétale)
PAYAN Jean-Jacques (détaché)	Mathématiques pures
PEBAY PEYROULA Jean-Claude	Physique
PERRIAUX Jacques	Géologie
PERRIER Guy	Géophysique
PIERRARD Jean-Marie	Mécanique
RASSAT André	Chimie systématique
RENARD Michel	Thermodynamique
RICHARD Lucien	Biologie végétale
RINAUDO Marguerite	Chimie CERMAV
SENGEL Philippe	Biologie animale
SERGERAERT Francis	Mathématiques pures
SOUTIF Michel	Physique
VAILLANT François	Zoologie
VALENTIN Jacques	Physique nucléaire I.S.N.
VAN CUTSEN Bernard	Mathématiques appliquées
VAUQUOIS Bernard	Mathématiques appliquées
VIALON Pierre	Géologie

PROFESSEURS DE 2ème CLASSE

ADIBA Michel	Mathématiques pures
ARMAND Gilbert	Géographie

.../...

AURIAULT Jean-Louis	Mécanique
BEGUIN Claude (M.)	Chimie organique
BOEHLER Jean-Paul	Mécanique
BOITET Christian	Mathématiques appliquées
BORNAREL Jean	Physique
BRUN Gilbert	Biologie
CASTAING Bernard	Physique
CHARDON Michel	Géographie
COHENADDAD Jean-Pierre	Physique
DENEUVILLE Alain	Physique
DEPASSEL Roger	Mécanique des fluides
DOUCE Roland	Physiologie végétale
DUFRESNOY Alain	Mathématiques pures
GASPARD François	Physique
GAUTRON René	Chimie
GIDON Maurice	Géologie
GIGNOUX Claude (M.)	Sciences nucléaires I.S.N.
GUITTON Jacques	Chimie
HACQUES Gérard	Mathématiques appliquées
HERBIN Jacky	Géographie
HICTER Pierre	Chimie
JOSELEAU Jean-Paul	Biochimie
KERCKOVE Claude (M.)	Géologie
LE BRETON Alain	Mathématiques appliquées
LONGEQUEUE Nicole	Sciences nucléaires I.S.N.
LUCAS Robert	Physiques
LUNA Domingo	Mathématiques pures
MASCLE Georges	Géologie
NEMOZ Alain	Thermodynamique (CNRS - CRTBT)
OUDET Bruno	Mathématiques appliquées
PELMONT Jean	Biochimie
PERRIN Claude (M.)	Sciences nucléaires I.S.N.
PFISTER Jean-Claude (détaché)	Physique du solide
PIBOULE Michel	Géologie
PIERRE Jean-Louis	Chimie organique
RAYNAUD Hervé	Mathématiques appliquées
ROBERT Gilles	Mathématiques pures
ROBERT Jean-Bernard	Chimie physique
ROSSI André	Physiologie végétale
SAKAROVITCH Michel	Mathématiques appliquées
SARROT REYNAUD Jean	Géologie
SAXOD Raymond	Biologie animale
SOUTIF Jeanne	Physique
SCHOOL Pierre-Claude	Mathématiques appliquées
STUTZ Pierre	Mécanique
SUBRA Robert	Chimie
VIDAL Michel	Chimie organique
VIVIAN Robert	Géographie

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Année universitaire 1982-1983

Président de l'Université : D. BLOCH

Vice-Président : René CARRE
Hervé CHERADAME
Marcel IVANES

PROFESSEURS DES UNIVERSITES :

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

.../...

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

PROFESSEURS ASSOCIES

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

PROFESSEURS UNIVERSITE DES SCIENCES SOCIALES (Grenoble II)

BOLLIET Louis
Chatelin Françoise

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

RIEU Jean
SOUSTELLE Michel

CHERCHEURS DU C.N.R.S.

FRUCHART Robert	Directeur de Recherche
VACHAUD Georges	Directeur de Recherche

.../...

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

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

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

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

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

.../...

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

.../...

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

LABORATOIRES EXTERIEURS

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

Je tiens à remercier ici l'ensemble des personnes qui, par leurs conseils, leurs remarques et leurs encouragements, ont contribué à l'aboutissement de ce travail :

Monsieur Jacques MOSSIERE, Directeur du Laboratoire de Génie Informatique au sein duquel s'est effectuée cette étude, et Président du jury.

Monsieur Sacha KRAKOWIAK, Professeur à l'Université de Grenoble, mon Directeur de thèse.

Monsieur Jean FERRIE, Professeur à l'Université de Montpellier, et avec lui l'équipe du CRIM, pour les discussions enrichissantes que nous avons eues.

Monsieur Georges GARDARIN, Professeur à l'Université de Paris VI, qui a accepté de juger ce travail.

Monsieur Michel ADIBA, Professeur à l'Université de Grenoble, qui a bien voulu participer au jury.

Monsieur Gérard ROUCAIROL, Directeur de la Division Architecture et Logiciel du Centre de Recherche BULL, d'avoir accepté de participer au jury.

L'ensemble des participants au projet SCOT, et tout particulièrement Paul DECITRE sans qui ce travail n'aurait pas existé.

J'exprime aussi ma gratitude à Marie-Laure WOROWSKI pour la part prépondérante qu'elle a pris à la réalisation matérielle de cette thèse.

SOMMAIRE

V. CONCLUSION

REFERENCES BIBLIOGRAPHIQUES

ANNEXES TECHNIQUES

A. Description de quelques algorithmes de contrôle de l'accès concurrent.

B. Exemple de programmation d'une application répartie.

C. Bases de données distribuées : anatomie d'une transaction.

D. Diagramme d'état d'un agent.

Notes à l'attention du lecteur :

- Chaque chapitre est précédé d'un sommaire plus détaillé.

- La numérotation des pages est relative à chaque chapitre (de la forme numéro de chapitre - numéro de page).

- De la même manière, l'identification des figures est propre à chaque chapitre.

AVANT-PROPOS

Ce travail concerne les Systèmes d'Information Répartis, un domaine qui a fait l'objet, ces dernières années, de nombreuses actions de recherche et développement, en particulier dans le cadre du Projet Pilote SIRIUS dont le Projet SCOT fut une composante.

Issu de l'expérience d'un précédent projet : "POLYPHEME", réalisé en collaboration avec l'Université de Grenoble, SCOT s'est volontairement restreint à l'étude d'un nombre limité de thèmes, parmi lesquels le maintien de la cohérence dont il est question ici, dans la perspective d'un transfert technologique. Cette orientation a démarqué notre approche du contrôle de la cohérence par rapport à des études plus académiques menées à la même époque et référencées dans cet ouvrage.

Ce contexte particulier d'une recherche pré-industrielle explique le caractère parfois pragmatique de notre contribution qui porte autant sur l'évaluation et le champ d'application des algorithmes que sur leurs propriétés intrinsèques. Le fait que les résultats présentés ici aient fortement contribué à la définition de produits en cours de réalisation montre que l'objectif de recherche appliquée, qui était poursuivi, a été atteint.

Le maintien de la cohérence dans les Systèmes d'Information Répartis est un domaine qui comporte de multiples aspects. Notre travail a porté sur plusieurs d'entre eux et tout particulièrement sur le traitement des erreurs et des pannes. Cependant, afin de respecter une certaine unité de présentation, cette thèse décrit, à côté de notre contribution, des aspects complémentaires empruntés à d'autres auteurs.

0

Introduction

0. INTRODUCTION

L'évolution des systèmes informatiques vers des configurations réparties est la conséquence des développements technologiques qui ont marqué la dernière décennie. Les faits les plus marquants de cette évolution sont, d'une part, l'apparition des mini et micro-ordinateurs caractérisés par un rapport prix/performance très favorable, et d'autre part, le développement des réseaux de télécommunication.

Les applications téléinformatiques, nées de la possibilité de se connecter à distance à un ordinateur de traitement, se généralisent dans les années 70 (soumission de travaux à distance, systèmes conversationnels à temps partagé) et ne cessent ensuite d'évoluer, multipliant ainsi les perspectives des applications classiques (gestion transactionnelle, recherche documentaire, systèmes de gestion de bases de données). Dans tous ces exemples cependant, la structure du système est étoilée et la puissance de calcul reste concentrée sur l'ordinateur central.

La maîtrise des réseaux d'ordinateurs constitue un pas décisif dans l'évolution des applications téléinformatiques en autorisant le dialogue, non seulement entre un utilisateur et un système informatique, mais directement entre deux applications s'éxécutant sur des sites éloignés.

Nous ne nous étendrons pas ici sur les motivations conduisant à l'abandon des systèmes centralisés au profit des configurations distribuées (modularité, disponibilité, coût, ...) ni sur les critères de répartition des informations et des traitements associés. Nous nous intéressons plutôt à la manière de concevoir un système d'information réparti. Dans ce domaine, deux philosophies s'opposent selon qu'on transfère les données vers le centre de traitement ou au contraire qu'on transfère les requêtes vers le lieu de résidence des données.

La première catégorie regroupe les systèmes de Transfert de Fichier aujourd'hui disponibles sur de nombreux matériels. Cette technique est maintenant suffisament maîtrisée pour faire l'objet de tentatives de normalisation afin d'étendre son champ d'application à des systèmes hétérogènes. Cette approche a cependant montré rapidement ses limites dues à la lenteur des moyens de communication conventionnels et au volume d'informations inutiles transférées. C'est pourquoi cette voie est progressivement délaissée au profit de systèmes permettant une réelle répartition des données et des traitements. On désigne cette nouvelle génération de systèmes répartis sous le terme générique de Systèmes de Gestion de Bases de Données Réparties (en abrégé SGBD-R).

Depuis 1975, des recherches ont été entreprises dans ce domaine tant aux Etats-Unis qu'en Europe. Ces études ont porté sur les thèmes suivants :

- modèles globaux de données prenant en compte la répartition et l'hétérogénéité des modèles locaux,
- langages de description et de manipulation associés à ces modèles
- expression des contraintes d'intégrité et de confidentialité,
- exécution répartie des traitements,
- contrôle de la cohérence des données dans un environnement réparti, partagé et non fiable.

Les résultats théoriques de ces actions de recherche ont été validés par la réalisation de prototypes de systèmes d'information répartis parmis lesquels on peut citer : SDD-1 [SDD1 79], D-INGRES [Neuhold 76], R* [Williams 82], DDTS [Devor 80]. C'est dans cet esprit qu'en 1976, l'Agence de l'Informatique a lancé le projet pilote SIRIUS qui s'est achevé officiellement en 1982 avec de nombreuses réalisations à son actif [LeBihan 80]. Les résultats présentés dans cet ouvrage sont issus de deux projets successifs, **POLYPHEME** et **SCOT** qui se situent dans le cadre du projet pilote SIRIUS.

POLYPHEME [POLYPHEME 79] est un prototype de Système de Gestion de Bases de Données Réparties avec schéma global et langage de requêtes relationnel permettant la coopération de systèmes hétérogènes. Cette approche permet d'offrir au concepteur d'applications des langages de description et de manipulation des données qui font apparaître le système distribué comme un SGBD centralisé (la distribution des données est transparente) [Adiba 78, Caleca 78]. Cette recherche a permis de mesurer la distance séparant un prototype d'un produit et de déceler les limites actuelles d'une approche uniquement basée sur la répartition au niveau de la description des données [SCOT 83].

L'expérience acquise dans POLYPHEME a conduit à la définition d'un nouveau projet, SCOT (Système pour la COopération COhérente de Transactions) dans lequel la répartition s'exprime au niveau des programmes d'application. Ce type de répartition revient à considérer que chaque site est responsable de la gestion de ses propres programmes d'application (appelés transactions), considérés comme opérateurs d'accès aux données locales. Le système fournit les primitives qui permettent le contrôle de la coopération des programmes en vue de réaliser une application globale. C'est l'approche "Transactionnel Coopérant".

Par rapport aux Systèmes de Gestion de Bases de Données Réparties, l'approche "Transactionnel Coopérant" laisse un certain nombre de problèmes à la charge du concepteur d'application tels que la localisation des données et des programmes ou l'hétérogénéité des modèles de données. Cette simplicité devait permettre , d'une part de déboucher sur des produits dans un délai plus court, et d'autre part de récupérer en grande partie les programmes existants, allégeant ainsi l'effort de transition des applications vers des configurations réparties. En contrepartie de cette tendance à la simplification, justifiée par les nécessités du marché, le projet SCOT s'est plus particulièrement focalisé sur l'étude des techniques de contrôle de la cohérence, un sujet qui avait été volontairement délaissé dans POLYPHEME.

Cet ouvrage présente une synthèse des travaux portant sur le maintien de la cohérence, réalisés dans le cadre du projet SCOT. Le maintien de la cohérence désigne l'ensemble des techniques mises en oeuvre pour assurer la validité du système d'information par rapport au monde réel qu'il modélise. en fait, le terme générique "maintien de la cohérence" recouvre plusieurs aspects complémentaires.

- Les informations sensibles ou confidentielles doivent être protégées des accès non autorisés; c'est le domaine de la sécurité. Ce point n'est pas traité ici.

- Les informations enregistrées dans le système doivent être conformes à la sémantique des objets représentés. Les tests de conformité s'expriment à l'aide de conditions que les données doivent vérifier en permanence; c'est le domaine des contraintes d'intégrité qui sera partiellement évoqué dans le chapitre suivant.

- Le dernier point concerne les incohérences engendrées par des incidents non prévisibles tels que les erreurs, les défaillances ou l'accès incontrôlé à des informations partagées. Le contrôle de ces évènements incombe au système d'information. C'est cet aspect du contrôle de la cohérence qui a fait l'objet de nos recherches. Dans ce domaine, les techniques de contrôle sont fondées sur deux notions essentielles : une programmation modulaire basée sur le concept de transaction et l'utilisation de la redondance.

Le concept de transaction joue un rôle essentiel à la fois dans la spécification de l'application et dans le contrôle de la cohérence. Les systèmes d'informations correspondants sont dits "transactionnels" et constituent aujourd'hui l'essentiel des systèmes opérationnels dans des domaines très divers : réservation de place, systèmes bancaires, etc .. Depuis quelques temps, on observe des tentatives de plus en plus nombreuses visant à exploiter les propriétés de la transaction dans d'autres champs d'activité que la gestion de production (Génie Logiciel, Bureautique, CAO, etc.). Cela nous amènera à évoquer l'adéquation des techniques définies pour la gestion à ces domaines.

Le fait d'utiliser la redondance apparaît comme assez naturel quand il s'agit, comme nous l'avons indiqué, de prendre en compte les pannes et les erreurs. Nous retrouverons donc au cours des développements ultérieurs des techniques caractéristiques des systèmes tolérant les fautes. Une particularité des systèmes d'information transactionnels provient du couplage des mécanismes de sûreté de fonctionnement avec la notion de transaction.

Cependant, l'utilisation de la redondance est limitée par la charge supplémentaire qu'elle impose en terme de capacité de stockage et de temps d'exécution. Le coût est parfois incompatible avec l'objectif de performance lorsque celui-ci constitue le critère de satisfaction fondamental de l'application. Pour illustrer l'importance de la performance, on cite généralement l'exemple des systèmes de réservation des compagnies aériennes qui doivent supporter la charge induite par plusieurs milliers de terminaux connectés. En période de pointe, cette charge atteint plusieurs centaines de requêtes par seconde avec un temps de réponse qui ne doit pas excéder 2 à 3 secondes [Gifford 84]. Pour absorber un tel débit et garantir ces temps de réponse, la redondance est strictement limitée à l'essentiel et il est admis que des informations moins critiques puissent être perdues ou dénaturées. La détermination du mécanisme de contrôle "idéal" pour chaque application est un compromis entre la sûreté de fonctionnement et la performance. l'étude de plusieurs applications réelles montre que chaque système constitue en la matière un cas d'espèce.

En résumé, notre approche du contrôle de la cohérence a été constamment guidée par les deux postulats suivants :

- la cohérence et la performance sont deux objectifs contradictoires.

- les besoins des applications dans ces deux domaines sont variables.

Cette démarche dans laquelle on associe les aspects théoriques du contrôle de la cohérence, l'évaluation (qualitative et quantitative) des mécanismes correspondants et les contraintes imposées par les applications réelles a démarqué fortement notre approche et a influencé la nature des résultats présentés ici. Une façon de résoudre le tryptique "Cohérence - Performance - Besoins des applications" consiste à appliquer le principe de la "séparation des mécanismes et des stratégies" : à partir d'une analyse des propriétés des différents algorithmes et de leur champ d'application, on sélectionnera un ensemble de mécanismes sur lesquels il soit possible de bâtir diverses politiques de maintien de la cohérence. Ces mécanismes de base constituent un "noyau transactionnel", c'est-à-dire une machine virtuelle supportant la notion de transaction et constituant une base sûre et efficace pour la réalisation d'applications.

Le contrôle de la cohérence regroupe en fait deux mécanismes de base : le traitement des erreurs et des pannes (désigné dans la suite sous le terme "validation-reprise") et la synchronisation des accès aux objets partagés (appelé aussi "contrôle de l'accès concurrent"). Ce dernier point a suscité un véritable engouement dans la communauté scientifique, qui s'est traduit par la proposition d'une multitude d'algorithmes tant en centralisé qu'en réparti, dont un petit nombre seulement a été implémenté ou évalué. Aujourd'hui, ce domaine est bien maîtrisé sur le plan théorique mais pèche beaucoup sur celui de l'applicabilité et de l'étude de performance. C'est pourquoi notre effort a principalement porté sur l'évaluation comparative de plusieurs d'entre eux en vue de déterminer le meilleur choix pour une classe d'applications déterminée. Dans cette présentation, nous ne consacrerons pas d'étude de fond au contrôle de l'accès concurrent. Pour les aspects théoriques, nous renverrons le lecteur à des études de synthèse parues récemment et nous nous limiterons à évoquer, d'une part les relations avec le traitement des erreurs et des pannes, et d'autre part les résultats fondamentaux de nos études de performance.

Par comparaison, la littérature sur les techniques de traitement des erreurs et des pannes est beaucoup moins prolixe. Dans nos travaux, ce domaine a fait l'objet d'une attention particulière et constitue la part la plus significative de notre contribution.

Le chapitre 1 présente globalement le problème du contrôle de la cohérence indépendamment du type d'environnement (centralisé ou réparti). Les concepts de cohérence, de transaction et d'atomicité y sont précisés. Diverses formes de cohérence et d'atomicité sont évoquées qui correspondent, au niveau des applications, à l'expression de besoins différents. Ces besoins sont illustrés à l'aide d'exemples concrets extraits d'applications réelles.

Les chapitres 2 et 3, d'organisation similaire, sont consacrés à l'étude des propriétés des mécanismes de contrôle de la cohérence dans les systèmes centralisés puis dans les systèmes répartis. Dans ces présentations, l'accent est mis principalement sur le traitement des erreurs et des pannes. Ces deux chapitres constituent le prérequis nécessaire à la présentation de nos solutions.

Le chapitre 4 regroupe les résultats des travaux réalisés dans le projet SCOT. Les problèmes évoqués au chapitre précédent sont repris et diverses solutions sont examinées sous le triple aspect de la correction, de la performance et de l'adéquation aux applications. en ce qui concerne les mécanismes de validation et reprise, l'analyse est très complète et conduit à de nombreux résultats originaux. pour le contrôle de l'accès concurrent, notre contribution est plus restreinte et concerne essentiellement la famille des algorithmes de verrouillage.

L'ensemble de ces études débouche sur deux résultats majeurs :

- la définition d'un noyau transactionnel répondant à l'objectif de disponibilité annoncé et implémenté en partie dans la maquette SCOT,

- des recommendations à l'usage des concepteurs pour le choix d'un mécanisme approprié en fonction de la configuration. Ce point concerne essentiellement validation et reprise.

La conclusion dresse un bilan des retombées industrielles de nos travaux sur le contrôle de la cohérence et présente une voie de recherche nouvelle issue du mariage du transactionnel et des types abstraits.

Chapitre

I

Contrôle de la Cohérence

Présentation

SOMMAIRE DU CHAPITRE I

1. QU'EST CE QUE LA COHERENCE DANS UN SYSTEME D'INFORMATION ?

Ce chapitre présente les éléments de structuration d'un système d'information et introduit les concepts de base utilisés dans la suite de ce travail : intégrité, cohérence, atomicité, transaction,...

1.1. Introduction.

Il est usuel de distinguer deux aspects dans un Système d'Information, la Base de Données et les Opérations sur les données. La base de données représente un modèle du monde réel dans lequel sont décrits la structure des objets manipulés et les dépendances fonctionnelles entre ces objets. Cette description constitue un modèle de structure, celle des opérations est un modèle de comportement.

Exemple : un Système Bancaire.

La description de notre système bancaire simplifié met en jeu deux types d'objets : CLIENT et COMPTE dont la structure est définie ci-dessous :

```
Type CLIENT                    Type COMPTE
     Nom     : chaîne,              Numéro : entier,
     Age     : entier,              Solde  : réel.
     Adresse : chaîne.
```

Cette description est complétée par la définition des opérations susceptibles d'être effectuées sur ces objets. Pour les objets de type COMPTE par exemple, on définit les opérateurs : OUVRIR, FERMER, CONSULTER, CREDITER, DEBITER. L'opérateur DEBITER peut s'exprimer de la manière suivante :

```
Procédure DEBITER (C : entier, M : réel)
debut
* C est le numéro du compte à débiter (clé d'accès à l'objet)
* M est le montant du débit
  X : compte
  LIRE-Donnée (C, X)
  X.Solde : = X.Solde - M
  ECRIRE-Donnée (X, C)
fin
```

Il est possible également de définir des opérations mettant en jeu plusieurs objets. C'est le cas, par exemple, lorsqu'on veut décrire un transfert de fonds entre deux comptes, consécutif au traitement d'un chèque.

```
Procédure TRANSFERT (Cd : entier, Cc : entier, M : réel)
début
* Cd représente le compte à débiter, Cc le compte à
* créditer et M le montant du transfert.
  X : compte
  LIRE-Donnée (Cd, X)
  X.solde := X.Solde - M
  ECRIRE-Donnée (X, Cd)
  LIRE-Donnée (Cc, X)
  X.Solde := X.Solde + M
  ECRIRE-Donnée (X, Cc)
fin
```

Une écriture plus synthétique consiste à définir l'opération TRANSFERT de la manière suivante :

```
Procédure TRANSFERT (Cd : entier, Cc : entier, M :réel)
début
  DEBITER (Cd, M)
  CREDITER (Cc, M)
fin
```

Une application est ainsi décomposée en opérations elles-mêmes décomposables en opérations plus simples. Ce processus de décomposition introduit plusieurs niveaux d'abstractions caractérisés par un ensemble d'objets et d'opérations sur ces objets. Une opération d'un niveau d'abstraction est réalisée par une séquence d'autres opérations du niveau inférieur. Au niveau le plus bas de la hiérarchie on trouve des opérations élémentaires ou actions, considérées comme atomiques, en ce sens qu'elles ne se décomposent pas en entités plus fines.

Exemple : système bancaire

niveaux	Objets	Opérateurs
utilisateur	VIREMENT	TRANSFERT
Système d'Information	COMPTE-à-DEBITER COMPTE-à-CREDITER	DEBITER CREDITER
Système d'Exploitation	COMPTE-à-DEBITER.solde COMPTE-à-CREDITER.solde	LIRE-D, ECRIRE-D LIRE-D, ECRIRE-D

L'objet de type VIREMENT est défini de la manière suivante :

```
type VIREMENT
     Compte-à-débiter : COMPTE
     Compte-à-créditer : COMPTE
```

L'ensemble des valeurs prises par les objets constitue l'état de la base de données. La COHERENCE est liée à la validité des états successifs de la base par rapport au monde réel qu'elle modélise. Dans le système de représentation hiérarchisé évoqué ci-dessus, le concept de "cohérence" est très subjectif et n'a de signification que pour un niveau d'observation déterminé. Dans la suite de ce chapitre, nous aurons l'occasion de revenir sur les conséquences de la relativité de cette notion. Le terme "cohérence" recouvre deux aspects complémentaires selon qu'on s'intéresse seulement à la valeur des objets ou que l'on prend aussi en compte le déroulement des opérations.

Le contrôle de la cohérence doit pouvoir répondre aux deux questions suivantes :

a) Les informations stockées dans la base sont elles conformes à la sémantique des objets telle qu'elle a été spécifiée dans le modèle ? C'est le domaine des contraintes d'intégrité, traité au § 1.2.
b) Les opérations s'exécutent elles correctement ? Le terme "correct" fait référence ici à une éventuelle perturbation du déroulement de l'opération par des évènements externes non attendus : les erreurs, les pannes ou le partage incontrôlé des objets. Ce point est abordé au § 1.3.

On remarquera que cette définition de la cohérence est indépendante du fait que le système soit centralisé ou réparti. La distribution introduit uniquement un surcroit de difficulté dans la réalisation des mécanismes de contrôle correspondants.

1.2 Contraintes d'intégrité.

Pour contrôler la validité des états de la base, le concepteur spécifie des contraintes sur la valeur des objets. Ces contraintes s'expriment à l'aide de règles, communément appelées contraintes d'intégrité, qui sont des assertions ou prédicats que les objets doivent vérifier. La spécification et la vérification des contraintes d'intégrité constituent le contrôle d'intégrité.

Les contraintes d'intégrité s'expriment au niveau du modèle de structure et portent sur un objet ou un groupe d'objets. Dans le système bancaire mentionné plus haut, on peut imposer, par exemple, que le solde d'un compte soit toujours positif ou supérieur à une valeur représentant le découvert autorisé. Il est possible également de spécifier la plage de valeurs autorisée pour l'âge d'un client.

Exemple d'expression des contraintes d'intégrité :

```
type CLIENT                          type COMPTE
     Nom     : chaîne                     Numéro : entier
     Age     : entier (18-99)             Solde  : réel ( > 0)
     Adresse : chaîne
```

Un autre exemple de contrainte d'intégrité consiste à spécifier que la somme des soldes des deux comptes est un invariant de l'opération TRANSFERT.

A l'exécution, la violation d'une contrainte entraîne le refus d'exécuter l'action correspondante. L'utilisation de contraintes d'intégrité dans un système d'information permet de détecter des conditions anormales, et de protéger l'information contre des altérations dues à des opérations erronées. Le contrôle du respect des contraintes d'intégrité peut s'effectuer de deux manières : par interprétation de la structure lors de l'accès à l'objet, ou en générant, à la compilation, les séquences de test nécessaires.

Pour une analyse complète des techniques correspondantes, nous renvoyons le lecteur à [Delobel 82] et [Andrade 80]. Ce dernier ouvrage traite plus particulièrement le problème des contraintes d'intégrité dans un environnement réparti.

Pour les sytèmes d'information qui ne fournissent pas un mécanisme intégré de contrôle d'intégrité, les contrôles sémantiques sur la valeur des objets sont explicitement programmés dans le code de l'opération.

Exemple : contrainte "Solde positif" dans l'opération DEBITER

```
Procédure DEBITER (C : entier, M : réel)
début
   LIRE-D (C, X)
   X.Solde : = X.Solde - M
   Si X.Solde < 0 alors ERREUR
   Sinon ECRIRE-D (X, C)
fin
```

1.3 Opérations atomiques : les Transactions

Lorsque les contraintes d'intégrité associées aux objets sont vérifiées, la base est dans un état cohérent. Par extension, une opération est dite cohérente si les contraintes d'intégrité des objets qu'elle manipule sont vérifiées. Une opération cohérente fait passer la base d'un état cohérent initial à un nouvel état cohérent final.

Selon la complexité de l'opération ou des prédicats mis en oeuvre, le contrôle d'intégrité s'étend sur la durée de l'opération. Il est naturel d'en juger globalement les effets en fin d'opération. Pendant l'exécution de l'opération des incohérences temporaires sont inévitables qui correspondent à un état transitoire. Dans l'exemple de l'opération TRANSFERT, l'invariant (Cc + Cd) n'est pas vérifié à la fin de l'opération DEBITER. Cela n'est pas considéré comme une erreur car, du point de vue de la sémantique de l'application, on ne s'intéresse pas séparément à l'opération de débit ou de crédit mais au regroupement des deux.

On appelle **TRANSACTION** (*), une opération considérée comme atomique. L'atomicité exprime le fait qu'on ne s'intéresse pas aux actions prises séparément mais au résultat global d'une séquence d'actions. Ou bien toutes les actions sont effectuées, ou bien aucune ne l'est. Tout état intermédiaire est considéré comme incohérent. On retrouve ici une première illustration du caractère subjectif des notions telles que la cohérence ou l'atomicité en fonction du niveau d'observation auquel on se trouve : utilisateur, système d'information ou système d'exploitation. Dans l'exemple du système bancaire, LIRE-Donnée est une action atomique pour le système d'information alors qu'elle donne lieu à l'exécution d'une séquence d'instructions complexe à l'intérieur du système d'exploitation. De la même manière, DEBITER est une action atomique vue d'un utilisateur du système bancaire.

* Dans la suite, on désignera indifféremment sous le terme "transaction" l'opération et le processus qui l'exécute.

La définition des transactions est dictée par la sémantique de l'application. Si on reprend l'exemple du système bancaire, le traitement d'un retrait à un guichet se matérialise par une transaction constituée de la seule opération DEBITER. Le traitement d'un chèque est une transaction qui met en oeuvre deux opérations DEBITER et CREDITER.

Du point de vue du contrôle de la cohérence, le concept de transaction vient se superposer à la notion d'intégrité sémantique. Dans un système qui ne possède pas de mécanismes de contrôle de l'intégrité, la programmation explicite des contrôles sémantiques et la notion de transaction permettent le contrôle de la cohérence. Dans la suite, nous ne nous intéresserons plus au contrôle d'intégrité et le terme contrôle de cohérence fera exclusivement référence au contrôle de l'atomicité des transactions.

2. MAINTIEN DE LA COHERENCE DANS UN SYSTEME D'INFORMATION

Dans les systèmes d'information de première génération, la notion de transaction en tant qu'unité de cohérence n'apparaît pas clairement. Pour permettre la reprise après panne, une image cohérente de la base de données est sauvée périodiquement. La décision de prendre un point de cohérence est, soit confiée à l'opérateur, soit laissée à l'initiative du système. Dans ce cas, le critère le plus souvent adopté est le nombre de mises à jour ayant affecté la base depuis le point de cohérence précédent. Lorsqu'un point de cohérence est demandé, l'exécution de la procédure de sauvegarde est différée jusqu'à la terminaison des activités en cours. En cas d'incident entre deux points de cohérence, la base de données est rétablie dans l'état correspondant au point de cohérence précédent. Les modifications apportées à la base depuis ce dernier point sont perdues.

Un pas important est franchi dans la maîtrise de la cohérence lorsque celle-ci, au lieu d'être assujettie à des points définis arbitrairement, est associée aux traitements qui manipulent les données. C'est ainsi qu'est apparue la notion de **TRANSACTION** définie à la fois comme opérateur d'accès aux données et comme unité de cohérence. Dans cette approche, le système d'information passe par une suite discrète d'états cohérents qui correspondent à chaque terminaison d'une transaction.

2.1 Nature des problèmes à résoudre

2.1.1 Propriétés d'une Transaction

La transaction, définie comme unité de cohérence, suppose que les trois propriétés suivantes soient vérifiées :

P1. La séquence d'actions d'une transaction est **indivisible** : soit toutes les actions sont exécutées et le système atteint un nouvel état cohérent, soit aucune ne l'est et le système reste dans l'état initial. C'est le principe du "Tout ou Rien".

P2. Les actions d'une transaction sont **isolées** : pendant l'éxécution d'une transaction, le système est dans un état temporairement incohérent ; il est nécessaire de masquer cette incohérence aux transactions qui s'éxécutent en parallèle. De ce point de vue, la transaction définit un domaine de protection.

P3. Les effets des actions sont **permanents** (on dit aussi "rémanents"). Il est impossible d'annuler les effets des actions d'une transaction terminée. En particulier, les résultats d'une transaction survivent à une panne postérieure à la terminaison de cette transaction.

Les propriétés P1 et P2 caractérisent l'atomicité de la transaction.

L'atomicité est susceptible d'être remise en cause par l'occurrence d'un des évènements suivants : une erreur, une panne ou une interférence non contrôlée avec une autre transaction. Le principe du traitement de ces incidents est exposé dans les paragraphes suivants. Le respect de la propriété P3 sera examiné ensuite.

2.1.2 Erreurs sémantiques

Une erreur correspond à une anomalie dans le déroulement de la transaction, détectée par le système d'information ou le système d'exploitation sous-jacent (exemple : opération sur un compte inexistant). Un cas particulier d'erreur est provoqué par un arrêt programmé (exemple : test de compte à découvert dans l'opération de débit). La transaction est interrompue et un message explicitant la cause de l'erreur est transmis à l'utilisateur. Une partie seulement des actions de la transaction a été exécutée. L'objectif d'atomicité impose que les actions déjà exécutées soient annulées. Ce retour en arrière est désigné sous le terme générique "défaire".

2.1.3 Pannes

Une défaillance matérielle ou logicielle du système de traitement interrompt l'éxécution de toutes les transactions en activité. (le cas particulier d'une défaillance des mémoires secondaires est traité en 2.1.5.). L'objectif d'atomicité impose, comme dans le cas précédent, que les actions des transactions soient défaites. Il y a cependant trois différences essentielles dans le traitement des erreurs et des pannes.

- Une seule transaction est concernée par une erreur, tandis qu'une panne affecte toutes les transactions en cours.

- Lorsqu'une erreur se produit, le système est "vivant" et réagit aussitôt à cette condition par le traitement approprié. Dans le cas d'une panne, le traitement est différé jusqu'au redémarrage du système. Il fait l'objet d'une procédure de reprise destinée à remettre le système dans un état cohérent avant d'accepter l'exécution de nouvelles transactions.
- Une erreur est permanente ; elle est liée à la sémantique de l'application et se reproduira si la transaction est réexécutée sans correction préalable. En revanche, une panne est un événement incontrôlé, externe à la transaction, susceptible de ne pas se reproduire si la transaction est réexécutée. C'est pourquoi le traitement après panne est constitué de deux étapes : défaire les actions des transactions, puis refaire ces transactions.

2.1.4 Interférences entre transactions : Accès Concurrent

En l'absence d'erreur ou de panne, l'éxécution séquentielle des transactions garantit la cohérence du système. Cependant, pour des raisons d'efficacité évidentes, l'éxécution simultanée des transactions est nécessaire. Dans ce cas, les séquences d'actions de plusieurs transactions peuvent s'imbriquer dans le temps et provoquer des incohérences.

Exemple Deux opérateurs lancent l'exécution de la transaction DEBITER pour le même compte bancaire (opérations initialisées à 2 guichets indépendants). Ces transactions sont notées T_1 et T_2 et peuvent s'exécuter de la façon indiquée dans la figure 1.1 (t_i désigne le temps sur un axe vertical).

t_0	(T_1)	LIRE-D (C, X_1)	
t_1	(T_2)	LIRE-D (C, X_2)	Après exécution de cette
t_2	(T_1)	X_1.Solde = X_1.Solde - M_1	séquence d'actions, la va-
t_3	(T_2)	X_2.Solde = X_2.Solde - M_2	leur du solde est diminuée
t_4	(T_1)	ECRIRE-D (C, X_1)	seulement du montant M2.
t_5	(T_2)	ECRIRE-D (C, X_2)	L'opération T1 est perdue.

Fig. 1.1. Exemple d'interférence

D'autres exemples d'imbrications conduisant à des résultats incohérents sont décrits dans la littérature [Gardarin 78], [Lelann 79]. Ces scénarios montrent que l'accès incontrôlé à des objets partagés peut provoquer des incohérences telles que la lecture d'une valeur erronée d'un objet, la modification d'un objet en cours de consultation ou la perte d'une mise à jour (cas de l'exemple précédent). Le rôle des mécanismes de contrôle de l'accès concurrent consiste à ordonnancer les actions des transactions concurrentes de telle sorte que le résultat final soit équivalent à une exécution séquentielle : c'est la propriété de **sérialisation**. Les techniques de synchronisation classiques, telles que les moniteurs par exemple, sont insuffisantes dans la mesure où la synchronisation ne porte pas sur un seul objet partagé mais sur un groupe d'objets.

Le principe du contrôle de l'accès concurrent est basé sur la détection des dépendances entre transactions engendrées par les conflits d'accès aux objets partagés. Dans certains cas, un nouveau conflit peut entraîner le rejet d'une transaction. Le traitement consiste alors, comme pour une panne, à défaire les actions exécutées par la transaction, puis ultérieurement à refaire la transaction interrompue malgré elle.

2.1.5 Panne des mémoires secondaires

Pour garantir la propriété P3, à savoir la permanence des actions d'une transaction, un système d'information gère au moins deux représentations des objets : une version dite "active" consultée et modifiée par les transactions en cours d'exécution et une version dite "stable" composée des valeurs des objets relatifs aux transactions terminées. Le passage de la version active à la version stable est contrôlé par la procédure de validation. La version "stable" constitue l'état cohérent de référence du système d'information. Elle est conservée sur un support non volatile (généralement un disque).

Lorsque cette version de réference est détruite (panne d'un disque par exemple), les procédures de reprise évoquées précédemment à propos des pannes affectant le système de traitement ne sont plus suffisantes ; d'autres mécanismes s'avèrent nécessaires pour reconstituer une version cohérente des objets.

Le tableau suivant résume les actions exécutées en fonction du type d'incident.

erreur sémantique	défaire
panne du système	défaire, refaire
panne d'un support disque	reconstruire, refaire
accès concurrent	défaire, refaire

2.2 Niveaux d'atomicité

2.2.1 Introduction

Dans la définition d'une transaction (§ 1.3), nous avons soulevé le problème de la relativité de la notion d'atomicité par rapport au niveau d'observation auquel on se situe. Ce paragraphe examine l'impact de ces divers niveaux d'atomicité sur le contrôle de la cohérence. La figure 1.2 rappelle les principaux niveaux de réalisation pour une application de gestion d'information.

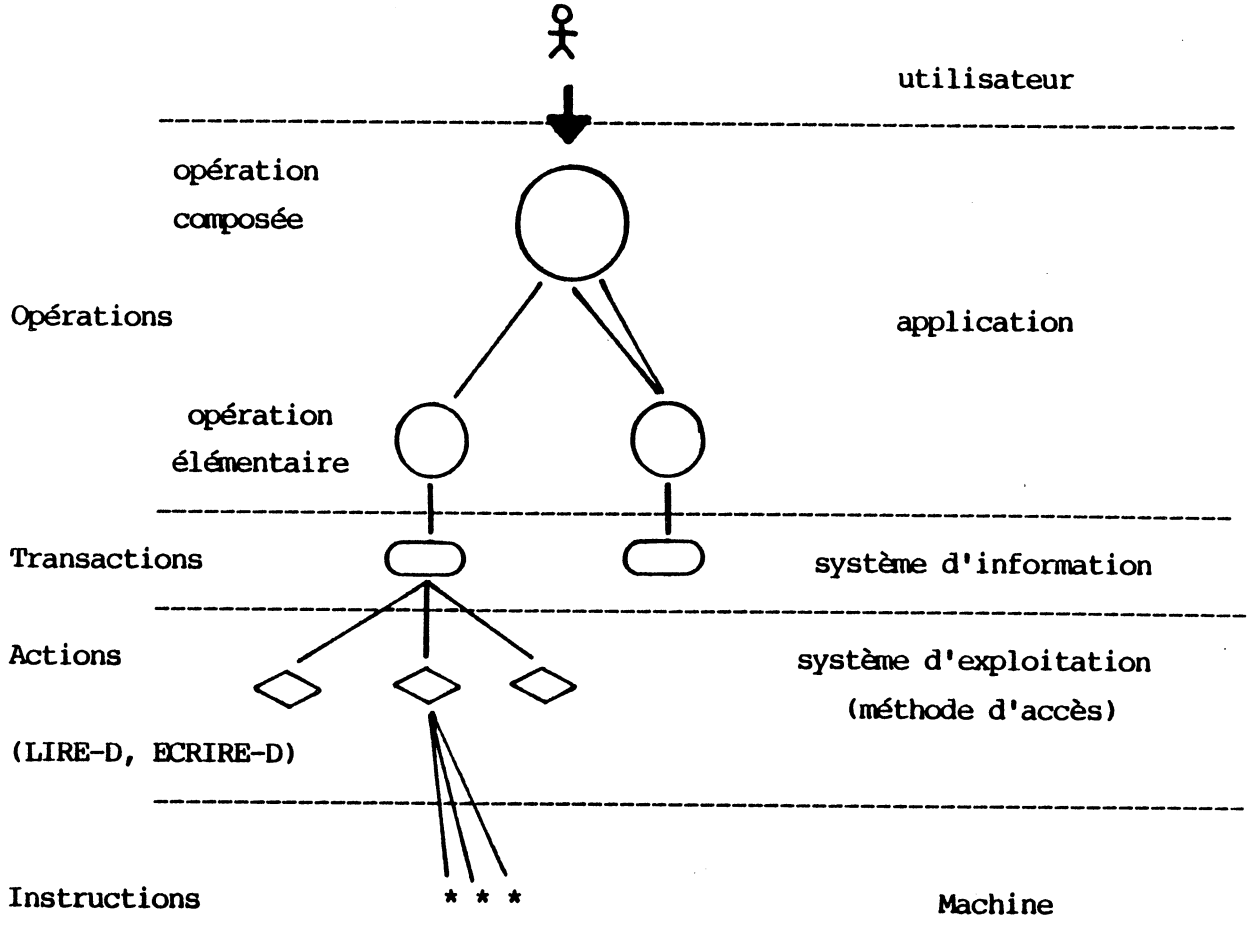


Fig. 1.2. Niveaux d'abstraction dans une application de gestion

Nous nous intéressons ici à la transition entre le niveau de l'application et celui du système d'information. La correspondance est, pour l'essentiel, définie par le concepteur de l'application mais elle peut également être imposée par le système d'information. Dans le premier cas, la décomposition est dictée par la sémantique de l'application ; dans le second cas, ce sont des contraintes de réalisation (performance, architecture du système) qui imposent ce découpage. Nous verrons des exemples de chacun d'entre eux dans la suite de ce paragraphe.

Dans la plupart des cas, les opérations définies à un niveau donné ne sont pas naturellement atomiques. En effet, l'exécution de la séquence correspondante au niveau inférieur peut être temporairement suspendue à tout moment pour différentes raisons.

Dans la suite, on s'intéressera essentiellement à la réalisation de l'atomicité au niveau du système d'information et on réservera le terme "Transaction" aux opérations de ce niveau pour lesquelles la propriété d'atomicité est vérifiée. Cette démarche signifie implicitement que nous considérons comme acquise l'atomicité des actions du niveau inférieur (c'est à dire LIRE-Donnée et ECRIRE-Donnée). Nous reviendrons sur ce point plus tard en étudiant, d'une part la possibilité d'introduire au niveau même du système d'exploitation la notion de "transaction atomique".

2.2.2 Pseudo-transactions

En théorie, la définition des transactions en tant qu'opération atomique est la responsabilité du programmeur d'application. C'est ainsi que les procédures DEBITER, CREDITER, ou TRANSFERT qui combine les actions de ces deux procédures, sont définies comme étant des "transactions". En réalité, la complexité de certaines applications ne permet pas toujours d'associer une "transaction au sens système du terme" à une opération dont la sémantique requiert pourtant qu'elle soit atomique. Pour cette raison, nous désignons ces opérations sous le terme "pseudo-transactions". Nous donnons deux illustrations de ce problème : les opérations de longue durée et les opérations interactives.

a) Opérations de longue durée

Certaines opérations s'étendent sur une période de temps qui peut atteindre plusieurs jours, voire plus. C'est le cas, par exemple d'une opération de transfert de fond entre deux banques étrangères dont le principe est représenté de manière très simplifiée (la distribution n'est pas représentée).

TRANSFERT-ETRANGER (Ca, Cb, M)
* Ca désigne le compte de la banque A, Cb celui de la banque B et
* M le montant du transfert dans l'unité monétaire de A.

1 - Vérification du solde du compte C_A
2 - Déclaration de l'opération auprès d'une banque de couverture dans le cas où les deux banques concernées par l'opération n'ont pas d'accord d'échange direct.
3 - Recherche du taux de change optimal : S --> S'
4 - Débit du compte C_A et transfert de la somme S' vers la banque de couverture.
5 - Transfert de la banque de couverture vers la banque B et crédit du compte C_B.

Les opérations notées 2 et 3 peuvent demander une intervention manuelle (responsable de la banque de couverture, agent de change), ce qui accroît encore le temps d'exécution de l'opération globale.

La propriété d'isolation stipule que les actions sur les objets de l'opération globale soient invisibles aux transactions concurrentes. Cela signifie dans notre exemple que les comptes Ca et Cb sont deux objets "inaccessibles" jusqu'à la fin de cette opération. Les autres transactions qui font référence à ces objets seront mises en attente ou rejetées. Plus généralement, une opération de longue durée, d'une part interdit tout partage de certains objets, et d'autre part monopolise des ressources physiques pendant une période de temps incompatible avec un environnement partagé. Une solution consiste à découper l'opération TRANSFERT-ETRANGER en plusieurs transactions qui s'exécutent en séquence (par exemple une transaction par opération élémentaire).

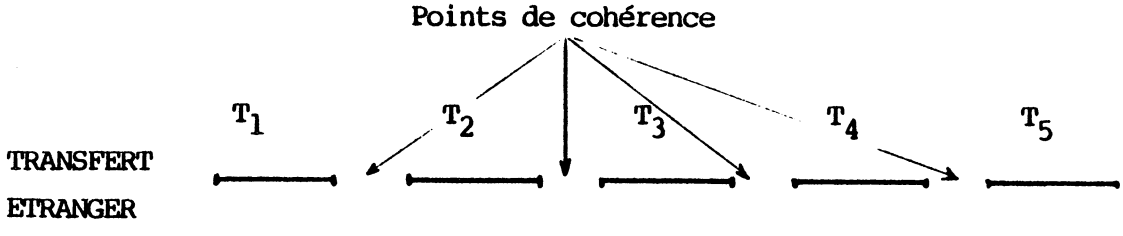


Fig. 1.3. Découpage d'une opération de longue durée

Le rôle du système se borne à garantir l'atomicité des transactions intermédiaires (notées T_i). Chaque terminaison d'une transaction intermédiaire est un point de cohérence pour le système, ce n'est pas

un point de cohérence pour l'application (Figure 1.3). Si une erreur interrompt une transaction intermédiaire, le système qui contrôle son exécution défait les actions de cette transaction. Malheureusement, la propriété P3 interdit d'annuler les effets des transactions précédentes. On résoud ce problème en programmant explicitement des transactions "inverses" dont le rôle consiste à remettre les objets modifiés dans un état "cohérent au sens de l'application". Cette technique assure que la propriété P1 est vérifiée pour l'opération globale mais ne garantit pas la propriété P2. En effet, les valeurs des objets manipulés par les transactions intermédiaires précédentes ont pu être consultées et utilisées par d'autres transactions.

b) Opérations interactives

Certaines opérations nécessitent de nombreux dialogues avec l'opérateur. Un exemple typique est une transaction de réservation de place dans laquelle un dialogue multiple s'établit entre le système et l'opérateur pour afficher les caractéristiques des vols disponibles (identification, horaire, tarification,..) et pour entrer les paramètres du passager à enregistrer (identification, abonnement, mode de paiement, ...). A l'échelle de temps d'un système informatique, une conversation est une action sensiblement longue; on se trouve alors dans un cas de figure analogue à celui des opérations de longue durée.

Certains systèmes imposent que chaque conversation corresponde à un point de cohérence. Les dialogues sont exclus des transactions. La structure d'une opération et les problèmes qui en découlent sont identiques à ceux du cas précédent. En cas d'échec d'une transaction intermédiaire, l'opérateur invoque explicitement une opération destinée à détruire les effets des transactions précédentes (annulation d'une réservation par exemple).

Remarque : Il ne faut pas confondre cette structuration avec l'utilisation de points de reprise à l'intérieur d'une transaction. Les points de reprise sont destinés à limiter la quantité de travail à refaire lorsque la transaction est provisoirement abandonnée puis réexécutée (reprise partielle). Un point de reprise n'est pas un point de cohérence, en particulier en ce qui concerne la visibilité à l'extérieur des objets manipulés par la transaction. Nous reviendrons en détail sur cette technique dans le chapitre 2.

2.2.3 Transactions imbriquées

La définition adoptée jusqu'à présent pour la transaction, à savoir séquence atomique d'actions, met en évidence le caractère linéaire de cette construction; on parle de transaction à 1 niveau. Cette structuration est en contradiction avec la démarche descendante, généralement adoptée dans la spécification d'une application dont le résultat définit une hiérarchie d'opérations emboitées, déjà évoquée au paragraphe 3.2.1, et représentée dans la figure 1.4.

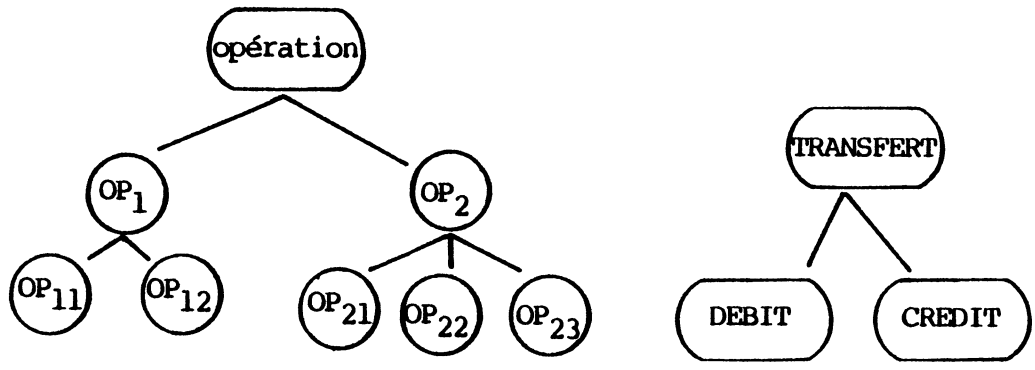


Fig. 1.4. Décomposition d'une opération complexe

Il est intéressant, dans certains cas, d'adopter pour une transaction une structuration identique ; on parlera alors de transaction à plusieurs niveaux ou bien de transaction imbriquée. On construit une transaction imbriquée en regroupant plusieurs opérations, appelées sous-transactions, dans une entité composée définie elle-même comme une transaction.

Exemple Dans un système transactionnel classique, la programmation de la transaction TRANSFERT ne fait pas apparaître explicitement la composition des transactions DEBIT et CREDIT. La structuration en transaction imbriquée autorise cette expression.

```
Transaction DEBIT  (C, S) début .... fin
Transaction CREDIT (C, S) début .... fin
Transaction TRANSFERT (Cd, Cc, S)
début
      DEBITER  (Cd, S)
      CREDITER (Cc, S)
fin
```

D'une manière plus générale, nous représenterons une transaction imbriquée par le schéma de la figure 1.5.

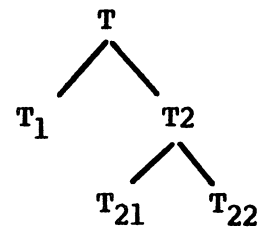


Fig. 1.5. Transaction imbriquée

La transaction imbriquée T est composée de deux sous-transactions filles Tl et T_2; T_2 est elle-même composée de deux sous-transactions T_{21} et T_{22}. Dans cette organisation hiérarchique, les noeuds et les feuilles de l'arborescence jouent des rôles différents :

- Seules les feuilles opérent sur les objets. Elles constituent donc la partie exécutive de la transaction.
- Les noeuds ont un rôle de synchronisation et de supervision des sous-transactions filles.

Par rapport aux transactions classiques, les transactions imbriquées présentent des propriétés intéressantes listées ci-dessous.

a) Synchronisation

Une transaction mère spécifie les conditions d'activation et de terminaison de ses filles. On permet ainsi l'exécution séquentielle ou parallèle de sous-transactions. Dans la transaction imbriquée TRANSFERT, par exemple, il est possible d'exécuter les sous-transactions DEBITER et CREDITER en parallèle (fig. 1.6-a). Il est possible également de forcer l'exécution de CREDITER seulement lorsque DEBITER est terminée (figure 1.6-b).

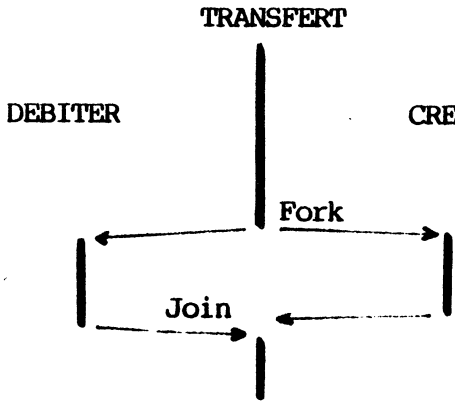


Fig. 1.6-a
Exécution parallèle

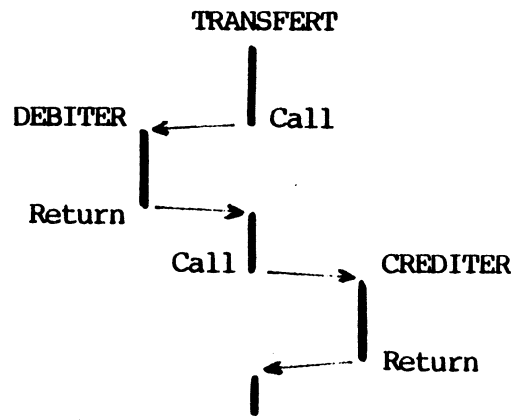


Fig. 1.6-b
Exécution procédurale

b) Atomicité

Une sous-transaction vérifie les propriétés P1 et P2. La propriété P3 est garantie seulement au niveau de la transaction globale. Ainsi, une sous-transaction ne peut valider séparément, les confirmations des actions des sous-transactions sont synchronisées au niveau de la transaction englobante. De plus, les ressources acquises par une sous-transaction peuvent être transmises à la transaction mère. De la même manière, les ressources acquises par la transaction mère sont héritées par les filles. Ce mécanisme de transmission et d'héritage permet d'assurer l'isolation des actions de la transaction englobante.

c) Reprise partielle

Une sous-transaction interrompue par une défaillance peut être reprise sans que le fonctionnement des autres sous-transactions soit affecté. Cette reprise n'affecte que l'arborescence issue de la sous-transaction. Cette possibilité permet de limiter les conséquences d'une panne à une partie seulement de la transaction.

d) Contrôle d'erreurs

Il est possible de récupérer au niveau d'un noeud les incidents affectant les sous-transactions filles. Ce mécanisme permet de limiter la propagation des conditions anormales en spécifiant pour chaque type d'erreur le traitement associé (mécanisme équivalent au ON-ERROR de PL/1). Si aucun traitement n'est précisé pour un type d'incident donné, la condition d'erreur remonte au noeud supérieur. Cette facilité est, à notre avis, l'aspect le plus important des transactions imbriquées car il permet de spécifier des stratégies plus fines que le "tout ou rien" de la transaction à un niveau.

La stucture d'une transaction imbriquée préfigure en partie celle d'une transaction répartie composée de sous-transactions s'exécutant sur des sites distincts (voir chapitre 3) avec, en plus, un mécanisme hiérarchisé de contrôle d'erreurs. En conséquence, il n'est pas curieux de constater que les techniques mises en oeuvre pour contrôler les propriétés énoncées s'apparentent à celles développées pour les sytèmes distribués. C'est le cas pour la synchronisation des terminaisons, la gestion des pannes et le contrôle des conflits entre deux sous-transactions d'une même transaction. Nous diffèrons donc la discussion de ces questions jusqu'au chapitre 3.

Les transactions imbriquées ont été étudiées en détail dans [Reed 78] et [Moss 82]. Une implémentation partielle de ces mécanismes existe dans le système SWALLOW développé au MIT [Svobodova 80].

2.3 Types de cohérence

Lorsque les trois propriétés énoncées en 2.1.1 sont vérifiées, l'atomicité des transactions est garantie; on dit alors que la cohérence est forte. Si l'une des propriétés n'est pas vérifiée, la cohérence est dans un état dégradé. Plusieurs auteurs se sont intéressés à caractériser et classifier ces formes de cohérence. Les résultats de ces études sont résumés dans les paragraphes 2.3.1 et 2.3.2. Le paragraphe 3.3.3 conclut en analysant les besoins d'applications réelles en terme de cohérence.

2.3.1 Niveaux de cohérence

La première publication traitant de cette question [Eswaran 76] proposait une classification en quatre niveaux de cohérence, notés de 0 à 3. Le niveau 3 correspondait à la cohérence forte ; les niveaux inférieurs étaient obtenus lorsque les propriétés P1 et P2 n'étaient pas vérifiées. Il est à noter que la propriété P3 n'était pas prise en considération dans ce modèle qui excluait les défaillances des supports de l'information permanente. Une part importante de cette étude a consisté à identifier les incohérences provoquées par le manquement à la règle P2 en différenciant les situations suivantes : une transaction consulte un objet en cours de modification ou une transaction tente de mettre à jour un objet en cours de consultation. Dans [Gray 78] on trouve une analyse du gain espéré en adaptant les mécanismes de contrôle de la cohérence aux exigences de chaque application. La conclusion de ce papier est que le niveau de cohérence fourni par le système n'a pas d'impact significatif sur les performances. J. Gray en conclut donc qu'il faut développer des mécanismes assurant la cohérence forte. On peut adresser deux reproches à cette analyse. D'une part, elle est basée sur des hypothèses de comportement simplistes et des évaluations très intuitives, d'autre part elle prend en compte exclusivement un système centralisé. Nous verrons dans le chapitre 3 que la répartition entraîne des différences de comportement beaucoup plus significatives en raison du nombre variable de messages induits par les mécanismes correspondants à ces niveaux de cohérence.

L'analyse en elle-même des dépendances entre les actions Lire-D et Ecrire-D est fondamentale, en particulier pour la définition des algorithmes de contrôle de l'accès concurrent. En revanche, la classification de la cohérence en quatre niveaux nous paraît être un raffinement superflu pour caractériser les besoins des applications. Dans la suite, nous parlerons plus simplement de cohérence forte pour qualifier un système qui vérifie les trois propriétés : indivisibilité, isolation, permanence. Par opposition nous parlerons de cohérence dégradée lorsqu'une de ces propriétés n'est pas vérifiée.

2.3.2 Cohérence forte et cohérence faible

Pour accroître la sécurité et la disponibilité, certains systèmes d'information exploitent le principe de la redondance des informations en maintenant plusieurs copies des objets. Cette approche est désignée sous le terme "base de données dupliquée" ou encore "base de données à copies multiples". La cohérence du système global exige qu'une modification d'un objet soit répercutée sur toutes les copies de cet objet. Il y a deux manières de concevoir le contrôle des copies.

Une première approche consiste à imposer que toutes les copies aient, à un instant donné, une valeur identique. Cette condition exige que les mises à jour des copies soient effectuées à l'intérieur d'une transaction. La conséquence de cette synchronisation très forte est une diminution sensible du taux de partage d'un objet. Une approche moins contraignante accepte un décalage entre la valeur des copies d'un même objet ; ce décalage correspondant au délai de mise à jour.

Dans ce contexte particulier, la cohérence forte caractérise un système qui fournit à tout instant la version cohérente la plus récente. Par opposition, la cohérence faible caractérise un système qui fournit une version cohérente éventuellement antérieure.

2.3.3 Analyse des besoins

On observe aujourd'hui un déphasage important entre la sophistication des techniques de contrôle de la cohérence publiées dans la littérature et la simplicité, parfois même le dépouillement des mécanismes mis en oeuvre dans les applications réelles. Les études théoriques donnant lieu à publications sont pour une large majorité consacrées à la cohérence forte. Les chercheurs justifient cette démarche en faisant observer que les algorithmes définis pour la cohérence forte traitent, a fortiori, les cas de cohérence dégradée. En fait, cette attitude ne tient pas compte du facteur performance qui conditionne en partie le choix d'un mécanisme. Un exemple de cette contradiction entre les objectifs de performance et de cohérence apparait clairement dans la justification des choix de réalisation du système de réservation de places de la compagnie TWA [Gifford 84] .

Depuis peu, on voit se dessiner quelques tentatives en faveur de mécanismes spécifiques, dérivés des mécanismes généraux, et adaptés à des profils de transactions particuliers (transactions en consultation, transaction de longue durée, etc .) [Garcia 83, Prädel 82, Lausen 82, Bernstein 83-b]. La suite de ce paragraphe présente quelques exemples d'applications dans lesquelles la cohérence forte n'est pas requise.

a) Consultation du solde d'un compte bancaire :

Une transaction de consultation est destinée à fournir au client une "valeur indicative" du solde de son compte. Si cette transaction est exécutée en parallèle avec une transaction de mise à jour du solde, la valeur retournée au client est une valeur cohérente qui n'est pas la plus récente. Cet exemple peut être considéré comme un cas de cohérence faible.

b) Statistique sur un ensemble de comptes

Dans un système bancaire, on considère une opération d'audit dé-

clenchée périodiquement. Par balayage de l'ensemble des comptes, cette opération est destinée à fournir des statistiques telles que montant total, solde moyen, découverts, ... C'est une opération longue qui opère sur un grand nombre d'objets. La cohérence forte exigerait que l'ensemble des objets soit inaccessible aux autres transactions jusqu'à la fin de l'opération. Cette contrainte est incompatible avec la disponibilité du système. Par conséquent, cette opération est exécutée sans contrôle de concurrence et elle rend un résultat approximatif, en conformité avec l'objectif statistique qui lui était assigné. Ceci est un exemple dans lequel la cohérence n'est pas assurée.

c) Réservation de place

Nous avons déjà vu dans un paragraphe précédent qu'une opération de réservation de place est découpée, uniquement pour des raisons de performance, en plusieurs transactions. La première fournit les caractéristiques du vol et le nombre de places encore disponibles, les suivantes permettent d'entrer les paramètres de la réservation (nom, paiement,....) ; la dernière enfin crée l'enregistrement lié à la réservation et met à jour le nombre de places disponibles. Si plusieurs réservations s'éxécutent en parallèle pour le même vol (ce qui n'est pas une situation rare), il est possible que le nombre de places disponibles ait été modifié entre l'affichage initial de l'état des réservations et l'enregistrement d'une nouvelle réservation (figure 1.7). On encourt ainsi le risque d'avoir plus de passagers enregistrés que de places disponibles. Cette incohérence est acceptée, voire encouragée par les compagnies aériennes (overbooking).

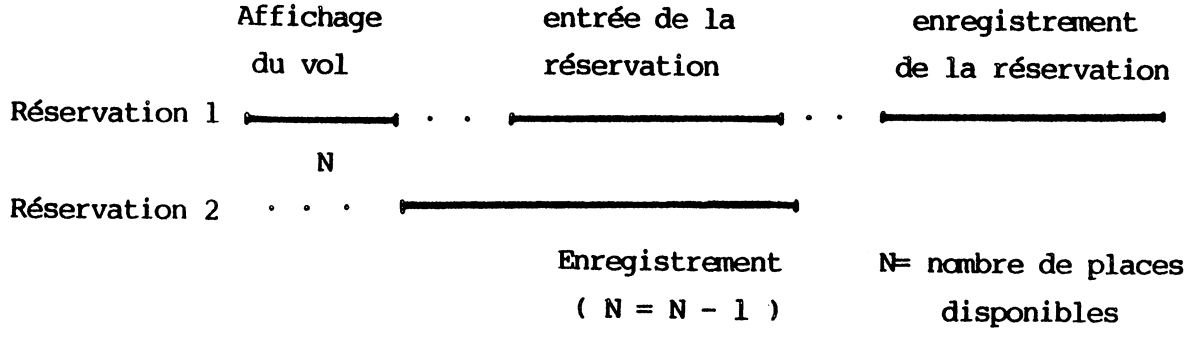


Fig. 1.7. Exemple d'incohérence

Le dernier exemple se situe également dans le domaine des systèmes de réservation de place. On considère maintenant une réservation multiple mettant en jeu plusieurs vols entre différentes escales. Chaque vol fait l'objet d'une réservation traitée à l'aide de la procédure que nous venons de voir. Supposons maintenant qu'il n'y ait pas de vol disponible entre les deux dernières escales et que l'utilisateur veuille annuler l'ensemble des réservations. Cette annulation ne peut pas être réalisée automatiquement faute d'avoir considéré la totalité des réservations comme une seule transaction. On résoud ce problème par une annulation explicite des réservations antérieures (opération ANNULER).

Conclusion

Dans un système d'information, le contrôle de la cohérence s'appuie sur le concept de transaction définie comme unité d'exécution atomique. La définition des transactions découle de la sémantique de l'application et incombe, de ce fait, au concepteur d'application. Nous avons vu que ce principe n'est pas toujours respecté et que le système impose parfois un découpage plus fin dicté à la fois par la gestion de ses ressources et aussi par la volonté de garantir un taux de disponibilité raisonnable aux objets partagés (cas des transactions longues ou interactives). Ce divorce illustre la relativité de la notion d'atomicité et montre qu'un système de contrôle intégré ne fournit pas une réponse appropriée dans tous les cas de figure.

Le modèle de transaction le plus répandu se limite à une structure linéaire : la séquence d'actions. Nous avons vu, à propos des transactions imbriquées, qu'il est parfois avantageux d'adopter une structuration plus riche (hiérarchique par exemple) présentant des propriétés intéressantes pour le contrôle des erreurs et la reprise après incident. Le dernier point important mis en évidence dans cette section est l'existence de plusieurs formes de cohérence. Dans la plus grande partie des recherches effectuées à ce jour, la cohérence "forte " constitue la version de référence. Notre démarche consistera à retenir des mécanismes qui permettent de prendre en compte des cas de cohérence dégradée sans perte d'efficacité.

Chapitre

II

Contrôle de la Cohérence dans les Systèmes d'Information Centralisés

SOMMAIRE DU CHAPITRE II

0. INTRODUCTION

Les mécanismes de contrôle de la cohérence en environnement réparti sont, dans une certaine mesure, des adaptations des mécanismes correspondants conçus pour les systèmes centralisés. Ce chapitre a pour principal objectif de faire le point sur ces techniques avant d'aborder ultérieurement le domaine des systèmes répartis.

La section 1 présente un modèle de système d'information transactionnel. Ce modèle permet d'introduire la terminologie élémentaire ainsi que les éléments de structuration d'une transaction et d'un système d'information orienté vers une utilisation transactionnelle. Les éléments du modèle sont largement utilisés dans ce chapitre et les suivants pour décrire et comparer les techniques de contrôle de la cohérence.

La section 2 décrit de façon détaillée les mécanismes de base entrant dans la réalisation des procédures de validation et de reprise après panne. Dans cette présentation, l'accent est mis sur le coût des mécanismes et diverses formes d'optimisation sont évoquées.

La section 3 est consacrée au contrôle de l'accès concurrent. Sur ce thème, notre exposé se limite à décrire le principe de fonctionnement des principales classes d'algorithmes en insistant sur les liens avec les procédures de validation et reprise. Une comparaison portant à la fois sur des critères qualitatifs et quantitatifs fait suite à cette présentation. pour conclure, on décrit brièvement les voies actuelles de recherche dans ce domaine.

1. MODELE D'UN SYSTEME D'INFORMATION TRANSACTIONNEL

1.1. Eléments du modèle

Les systèmes d'information disponibles aujourd'hui sur le marché, sont presque tous composés de deux parties : un module de gestion de transactions associé à un ou plusieurs systèmes de gestion de données structurées. On peut citer par exemple les associations suivantes : CICS/DL/1 et IMS/DB/DC chez IBM, TDS/IDS chez Cii Honeywell Bull. C'est pourquoi, nous reprenons une structuration analogue à celle décrite dans [Bernstein 81] pour décrire un système d'information. Cette architecture est représentée en figure 2.1 et comprend :

- un Module de Gestion de Transactions (noté MGT) qui contrôle l'exécution des transactions et assure le lien avec les usagers (contrôle d'identité, dialogue à travers un terminal, lancement et terminaison des transactions, ...).

- un Module de Gestion de Données (noté MGD) qui gère l'accès aux données : recherche des chemins d'accès, contrôle des droits d'accès, contrôle du partage et détection des conflits d'accès.

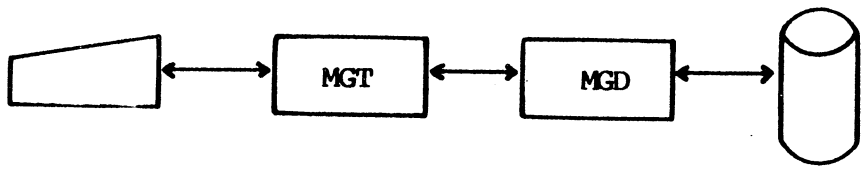


Fig. 2.1. Système d'Information Transactionnel

Dans ces systèmes, les programmes d'application sont structurés en transactions. Une transaction est définie comme une séquence atomique d'actions opérants sur des objets. Dans la suite, nous représenterons une transaction sous la forme :

T = [Ai, Oi / i=1,n] où Ai désigne une action sur l'objet Oi.

Les objets manipulés par une transaction appartiennent à l'une des trois catégories suivantes :

- Objets Temporaires.
  Ces objets sont locaux à une transaction et disparaissent lorsqu'elle est terminée ou abandonnée. Les variables locales entrent dans cette catégorie.

- Objets Permanents.
  Par opposition, les objets permanents sont partageables et les modifications réalisées par une transaction subsistent après sa terminaison. C'est le cas des enregistrements dans un fichier ou une base de données.
  Les actions LIRE-Donnée et ECRIRE-Donnée (en abrégé : LIRE-D, ECRIRE-D) s'appliquent aux objets temporaires ou permanents.

- Objets réels.
  Ces objets sont créés par la transaction, et subsistent après sa terminaison. Ils ne sont pas partageables.
  ECRIRE-Terminal, qui crée un objet de type message et SORTIR-Document, qui crée un objet destiné à être imprimer sont deux exemples d'actions portant sur des objets réels.

En résumé, on peut classifier les objets par rapport aux deux critères "Permanent" et "partagé".

	Permanents	Partagés
objets temporaires	non	non
objets réels	oui	non
objets permanents	oui	oui

Nous désignons sous le terme "temporaire", "permanente" ou "réelle" une action opérant sur un objet du type correspondant.

En plus de ces actions associées à la manipulation des objets, on trouve des actions de contrôle: DEBUT-Transaction, FIN-Transaction, ARRET, POINT-Reprise, dont la sémantique sera précisée plus loin, et qui constituent des délimiteurs de la transaction.

1.2. Atomicité et validation

La transaction, définie comme unité de cohérence, fait passer l'environnement d'un état cohérent initial Ei à un état cohérent final Ef. Chaque fin de transaction est un point de cohérence observable. L'environnement, en particulier les objets permanents, sont dans un état temporairement incohérent pendant l'exécution de la transaction.

Pour l'utilisateur, l'état de l'environnement est associé à l'issue de la transaction. Lorsque la transaction est terminée, l'environnement est passé dans l'état final. Si la transaction est abandonnée à cause d'une erreur ou d'une panne, l'environnement est revenu dans l'état initial. Lorsque la transaction est en cours d'exécution, l'environnement est dans un état non-observable ou indéfini (figure 2.2).

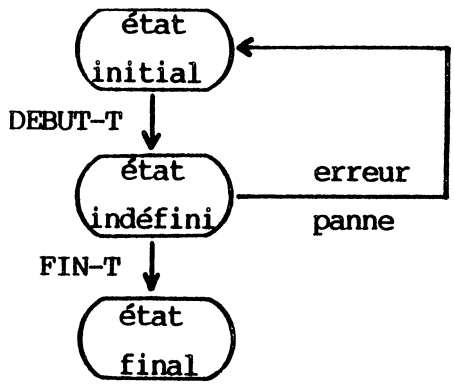


Fig. 2.2 Etats de l'environnement

Nous avons vu au chapitre précédent qu'un mécanisme de contrôle de la cohérence doit assurer les trois propriétés suivantes :

1. l'indivisibilité des actions de la transaction : toutes les actions sont exécutées une fois et une seule fois.
2. permanence des effets des actions d'une transaction.
3. isolation de chaque transaction : l'incohérence temporaire de l'environnement est masquée aux autres transactions.

La suite de cette section décrit les problèmes à résoudre et le principe d'une solution. La réalisation des mécanismes correspondants fait l'objet des sections 2 et 3.

1.2.1 Contrôle de l'indivisibilité fondé sur le "retour-arrière".

Lorsqu'un incident interrompt l'exécution d'une transaction, l'atomicité requiert que les actions exécutées jusque là soient "défaites".

a) Actions temporaires. Par définition, le résultat de ces actions disparait à la terminaison ou à l'abandon de la transaction. Le problème de "défaire" ces actions ne se pose donc pas.

b) Actions permanentes. Une solution consiste, avant d'exécuter une action permanente, à enregistrer le résultat de l'action inverse. Ainsi, dans le cas de l'action ECRIRE-D, on enregistre la valeur de l'objet avant modification, puis on exécute la modification. En cas d'incident, chaque objet est restauré dans son état initial à l'aide de l'image prise avant modification.

c) Actions réelles. La solution précédente ne s'applique pas. En effet annuler un document imprimé ou un message envoyé n'est pas réalisable sans intervention manuelle.

Le problème de ces actions "irréversibles" est partiellement résolu par la technique présentée au paragraphe suivant.

1.2.2 Contrôle de l'indivisibilité fondé sur les actions différées.

Dans cette méthode, les transactions sont exécutées en deux étapes : calcul et validation.

a) Etape de calcul : l'interprétation des actions est la suivante

DEBUT-T : Le MGT initialise pour T un espace de travail.

LIRE-D (X) : Le MGT recherche la valeur de la donnée X dans l'espace de travail de T. Si cette donnée existe, sa valeur est retournée à la transaction T, sinon le MGT émet une primitive "Lire-base" vers le MGD pour initialiser la valeur de la donnée X dans l'espace de travail de T. Cette valeur est ensuite rendue à T

ECRIRE-D (X) : Le MGT cherche la valeur de l'objet X dans le contexte de T. Si elle n'est pas présente le MGT émet une primitive "Préécrire-base" (*) pour l'obtenir du MGD. Cette valeur est ensuite mise à jour dans le contexte de T, sans écriture dans la base.

SORTIR-Document, ECRIRE-terminal : la description de chaque action avec les paramètres correspondants est enregistrée dans l'espace de travail. Les actions ne sont pas exécutées.

Pendant toute cette étape les actions permanentes et réelles sont simplement enregistrées dans un espace de travail propre à la transaction. L'état cohérent de référence de l'environnement est l'état initial. En conséquence, quelle que soit la nature de l'incident qui provoque l'arrêt de la transaction (erreur, panne ou occurrence de l'action ARRET), le système annule (invalide) toutes les actions enregistrées dans l'espace de travail de la transaction (on est ramené à l'état initial). Les actions n'ayant pas été exécutés, il n'est pas nécessaire de les "défaire".

* La distinction entre "lire-base" et "préécrire-base" est imposée par le contôle de l'accès concurrent (cf. section 3).

b) Etape de validation

Cette étape est exécutée par le système transactionnel lorsque la transaction a terminé l'étape de calcul (action FIN-T). La validation consiste à :

- Confirmer (on dit aussi valider) les actions enregistrées dans l'espace de travail. Le MGT exécute une primitive "Ecrire-base" pour tous les objets permanents qui ont été modifiés et les actions réelles sont exécutées.

- Signaler à l'usager la terminaison de la transaction.

- Relacher les objets acquis par la transaction pendant l'étape de calcul. L'effet des actions de la transaction est rendu visible aux transactions concurrentes.

Pendant toute cette étape, l'état cohérent de référence est l'état final. En conséquence, si cette étape est interrompue par une panne elle devra être réexécutée par le système transactionnel. Pour cette raison, l'étape de validation doit être idempotente.

Les incidents qui peuvent interrompre l'exécution d'une transaction sont classés en quatre catégories :

- les erreurs permanentes (ou fatales),
- les erreurs temporaires (ou récupérables),
- les défaillances du système central qu'elles soient d'origine logicielle ou matérielle,
- les défaillances des mémoires secondaires.

Nous examinons maintenant le principe de chaque traitement.

1.3. Erreurs fatales

Une erreur "fatale" est consécutive à un mauvais fonctionnement de la transaction ou à une intervention extérieure qui interdisent la continuation ou la reprise de la transaction.

- erreur dans le code de la transaction,
- arrêt programmé (action ARRET - T),
- arrêt commandé par l'opérateur,
- dépassement du temps imparti,
- violation des droits d'accès, etc ...

Une erreur fatale, correspondant à une anomalie pendant l'exécution de la transaction, ne peut affecter que l'étape de calcul. Par symétrie avec l'étape de validation nous appellerons étape d'invalidation le traitement correspondant qui consiste à :

- annuler les actions enregistrées sur l'espace de travail,

- signaler à l'usager la terminaison anormale de la transaction (un code d'erreur reflétant la nature de l'anomalie est retourné à l'usager).

- libérer les objets acquis par la transaction pendant l'étape de calcul (*).

Pendant cette étape, l'état cohérent de référence est l'état initial. Si cette étape est interrompue par une panne, elle devra être réexécutée (comme pour la validation, il est nécessaire qu'elle soit idempotente).

* Certaines techniques de synchronisation des accès aux objets permanents reposent sur l'octroi d'un "droit" de manipuler un objet. Le terme "libération" fait ici référence à l'abandon de ce droit en cas d'arrêt intempestif de la transaction (cf. section 3).

1.4. Erreurs récupérables

Au contraire de l'erreur fatale, l'erreur "récupérable" est susceptible de ne pas se reproduire si la transaction est réexécutée. Le cas le plus fréquent de ce type d'erreur est provoqué par une opération de préemption décidée par un gestionnaire de ressources. Le contrôle de l'accès concurrent entre dans cette catégorie (voir section 3).

Etant donné que la validation n'est ni consommatrice de nouvelles ressources, ni sujette à la préemption, une erreur récupérable ne peut interrompre que l'étape de calcul d'une transaction ; le traitement correspondant, appelé étape de reprise de la transaction consiste à :

- annuler les actions enregistrées dans l'espace de travail et libérer les objets.
- réexécuter la transaction.

Théoriquement, ce type d'incident est invisible à l'utilisateur. Aucun message d'erreur n'est émis et la reprise de la transaction est entièrement à la charge du système (à condition que le contexte d'initialisation de la transaction ait été sauvegardé dans l'espace de travail). En fait, cette transparence est illusoire dans le cas où la transaction dialogue avec le terminal, car alors, ce dialogue sera réexécuté (actions LIRE-Terminal, ECRIRE-Terminal).

L'état cohérent de référence est l'état initial. Comme dans les cas précédents, cette étape est réexécutée si elle est interrompue.

1.5. Défaillance du système

Dans le cas des erreurs fatales ou récupérables, le déroulement de la transaction est interrompu mais le système transactionnel continue de vivre. Il réagit immédiatement à l'incident pour exécuter le traitement correspondant (invalidation, ou reprise).

Dans le cas d'une panne, le système transactionnel est arrêté et, avec lui, l'ensemble des transactions qui sont sous son contrôle. Une panne du système a donc laissé l'environnement dans un état incohérent.

Au redémarrage du système transactionnel, il est nécessaire, au préalable à toute nouvelle exécution de transaction, de repositionner l'environnement dans un état cohérent. Le système réalise cette restauration en examinant pour chaque transaction l'étape qui était en cours d'exécution au moment de la panne. Nous avons déjà vu dans les paragraphes précédents quel traitement devait être appliqué lorsque les étapes de validation, d'invalidation ou de reprise étaient interrompues par une panne. Il nous reste à envisager le cas de l'étape de calcul pour laquelle le traitement est le suivant :

- annulation des actions de la transaction et libération des objets,
- réexécution de la transaction.

On constate que le traitement est le même lorsque la transaction exécutait l'étape de calcul et l'étape de reprise. Ces deux étapes n'ont donc pas à être différenciées pour le traitement après panne.

Le tableau suivant résume les opérations effectuées au redémarrage d'un système transactionnel après une panne.

Etape en cours au moment de la panne	Action exécutée au redémarrage du système
Calcul	Exécution étape de reprise
Reprise	Exécution étape de reprise
Validation	Exécution étape de validation
Invalidation	Exécution étape d'invalidation

1.6. Défaillance des mémoires secondaires.

Une défaillance de la mémoire secondaire a les conséquences suivantes

- les transactions en cours sont interrompues (ou seulement une partie d'entre elles s'il est possible d'isoler celles qui utilisent le dispositif en panne),

- l'information détruite par la panne doit être restaurée à partir d'une ancienne version et les modifications effectuées depuis cette sauvegarde doivent être réexécutées afin de revenir à un état le plus récent possible.

Ce type d'incident est rare (une ou plusieurs fois par an) et la reprise exige parfois plusieurs heures de calcul. Les systèmes à copies multiples, évoqués au chapitre précédent, sont un moyen de diminuer la probabilité de ce type d'incident. Après remise en état de la mémoire secondaire, la procédure de redémarrage exposée au paragraphe précédent est ensuite appliquée.

La section suivante détaille la réalisation d'un mécanisme de validation et reprise prenant en compte les points qui viennent d'être évoqués.

2. REALISATION D'UN MECANISME DE VALIDATION ET REPRISE

Dans le modèle de système d'information transactionnel qui vient d'être présenté, l'atomicité des actions d'une transaction repose sur l'utilisation d'un espace de travail et sur l'exécution en deux étapes : calcul et validation. La description de ces deux étapes a fait apparaître deux contraintes essentielles, à savoir :

- "défaire" les actions exécutées pendant l'étape de calcul si elle est interrompue par une erreur ou une panne,
- "refaire" les actions exécutées pendant l'étape de validation (ou d'invalidation) si elle est interrompue par une panne. Le paragraphe 2.1 décrit une technique de gestion de l'espace de travail compatible avec ce double objectif.

La description de la procédure de redémarrage du système après une panne (cf. § 1.5.) montre l'existence d'un point charnière dans la vie d'une transaction ; il s'agit de la transition entre l'étape de calcul et l'étape de validation (ou d'invalidation). Cette transition est généralement désignée sous le terme "point de validation" (ou d'invalidation) et doit être réalisée par une opération physiquement atomique. Ces points sont abordés en 2.2 et 2.3.

L'exécution des procédures de validation et de reprise entraîne une surcharge préjudiciable aux performances du système. Des possibilités d'optimisation sont examinées aux paragraphes 2.4 et 2.5.

2.1. Gestion de l'espace de travail : le JOURNAL

L'espace de travail d'une transaction est une zone de mémoire allouée au démarrage de la transaction. Les informations contenues dans l'espace de travail sont perdues lors d'un redémarrage du système transactionnel consécutif à une panne.

On désigne sous le terme de "journal" une extension de l'espace de travail dans laquelle seront stockées des informations destinées à survivre à un arrêt du système. Le journal est utilisé pour conserver

- la liste des transactions terminées,
- la liste des transaction abandonnées,
- pour chaque transaction en cours d'exécution :
  . le contexte d'initialisation,
  . l'identification de l'étape en cours d'exécution (calcul, validation, invalidation),
  . les informations nécessaires à l'annulation de l'étape de calcul ("défaire") ou à la reprise de l'étape de validation ou d'invalidation ("refaire").

L'utilisation du journal est précisée dans les paragraphes suivants en décrivant les transitions d'état d'une transaction et en détaillant la manière de réaliser les opérations "défaire" et "refaire".

Le plus souvent, le journal est réalisé à l'aide de fichiers sur disque. Nous évoquerons au paragraphe 2.4 l'impact d'une réalisation en technologie rapide.

2.2. Etats d'une transaction

L'indication de l'étape en cours d'exécution constitue l'état de la transaction. On distingue les états suivants :

- "ABSENT" avant exécution,
- "ACTIF" pendant l'étape de calcul,
- "VALIDE" pendant l'étape de validation,
- "INVALIDE" pendant l'étape d'invalidation,
- "TERMINE" après l'étape de validation,
- "ABANDONNE" après l'étape d'invalidation.

La figure 2.3 décrit le diagramme d'état d'une transaction.

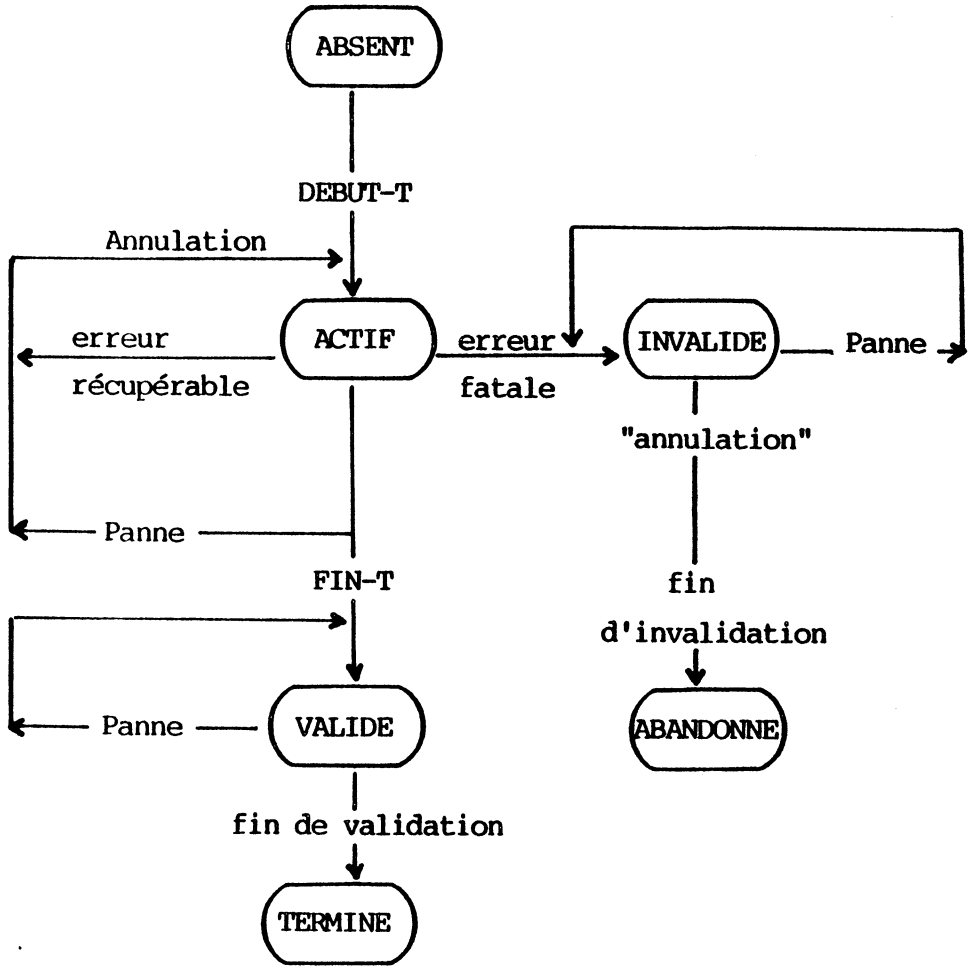


Fig. 2.3. Etats d'une transaction.

Les transitions d'état sont matérialisées par une écriture dans le journal.

La transition ACTIF --> VALIDE (respectivement ACTIF --> INVALIDE) est le point de validation (respectivement point d'invalidation) de la transaction.

2.3. Validation - Invalidation - Reprise

Dans ce paragraphe, nous abordons plus en détail le problème de la réalisation des procédures de validation, d'invalidation et de reprise. L'interprétation des actions d'une transaction est la suivante.

DEBUT-T

- création d'un espace de travail pour la transaction,
- création d'un identificateur de transaction. Cet identificateur peut être formé simplement à partir d'un compteur incrémenté à chaque lancement d'une nouvelle transaction (ou estampille).
- Ecriture sur le journal (en une seule opération) :
  - . identificateur de la transaction,
  - . contexte initialisation,
  - . état ACTIF.

LIRE-D, ECRIRE-D

Une image de l'objet est enregistrée dans l'espace de travail.

ECRIRE-T, SORTIR-D,

La description de ces actions, avec les paramètres correspondants, est enregistrée dans l'espace de travail.

FIN-T

- Les objets permanents modifiés et la description des actions réelles sont sauvegardés sur le journal,
- écriture sur le journal de l'état VALIDE : c'est le POINT DE VALIDATION. A partir de là, commence l'étape de validation qui consiste à exécuter les actions suivantes :
  - . écrire les objets permanents modifiés sur la base,
  - . exécuter les actions réelles (envoi de message, lancement de l'impression de documents, etc ..).
  - . envoyer un signal de terminaison au terminal,
  - . libérer les ressources acquises pendant l'étape de calcul.
- écriture sur le journal de l'état TERMINE.

ARRET ou erreur fatale

- écriture sur le journal de l'état INVALIDE. Cette écriture détermine le début de l'étape d'invalidation qui consiste à exécuter les actions suivantes :
  - . envoyer un message d'erreur au terminal (ou à l'activité ayant initialisé la transaction),
  - . libérer les objets acquis,
- écriture sur le journal de l'état ABANDONNE.

On notera que l'annulation des actions est implicite puisqu'elles n'ont pas encore été consolidées (elles n'ont même pas été recopiées sur le journal).

Erreurs récupérables

La remarque qui vient d'être faite s'applique également. L'espace de travail est purgé de la description des actions ; seules subsistent les informations enregistrées sur interprétation de la primitive DEBUT-T. Le journal n'est pas affecté.

Si une panne interrompt la transaction avant le point de validation, l'annulation des actions de la transaction est réalisée très naturellement en "oubliant" le contenu de l'espace de travail. Il n'est donc pas nécessaire de défaire des actions pour effectuer un retour-arrière. Le contexte d'initialisation, sauvegardé dans le journal, permet de relancer la transaction. Si la panne a interrompu la recopie des actions de l'espace de travail vers le journal, une simple opération de nettoyage suffit à restaurer l'état de celui-ci.

Si une panne interrompt la transaction après le point de validation, il est possible de réexécuter intégralement la liste des actions enregistrées sur le journal. Si cette réexécution ne comporte aucun risque pour les objets permanents, en revanche, elle peut engendrer une duplication des objets réels (message émis deux fois ou document imprimé deux fois) en contradiction avec la

propriété d'atomicité. Cette situation constitue l'inconvénient majeur d'une solution fondée sur l'opération "refaire" plutôt que sur l'opération "défaire". Nous justifions cependant le choix de cette solution en nous basant sur le fait que la probabilité de panne est bien moins grande pendant l'étape de validation que pendant l'étape de calcul (si on accepte le principe que la probabilité de panne est proportionnelle au temps d'exécution).

Au redémarrage du système transactionnel après une panne, les enregistrements écrits sur le journal font foi pour décider du sort de la transaction.

Etat transaction	Action exécutée au redémarrage du système
ACTIF	Exécution procédure de reprise
VALIDE	Exécution procédure de validation
INVALIDE	Exécution procédure d'invalidation
TERMINE	Pas d'action
ABANDONNE	Pas d'action

Il existe d'autres techniques de réalisation d'une procédure de validation-reprise, conçues pour des environnements systèmes appropriés. C'est le cas en particulier de la méthode dite des "pages fantômes" [Lorie 77], qui utilise les mécanismes de pagination pour réaliser les mises à jour atomiques de la base et la reprise après panne. Cette approche, mise en pratique dans le système R [Gray 81], présente l'avantage d'intégrer le contrôle de la cohérence dans les mécanismes de base du système d'exploitation et, par conséquent, de

simplifier la conception du système transactionnel. En revanche, cette technique ne permet pas de prendre en compte les actions portants sur les objets réels, pour lesquelles il est nécessaire de mettre en oeuvre une journalisation explicite.

En complément de cette présentation, le lecteur trouvera dans [Haerder 83] une taxonomie des techniques de réalisation pour la validation et la reprise.

2.4. Coût de la validation

Dans un système transactionnel muti-usagers, le journal constitue une ressource partagée de manière exclusive par les transactions. Le coût de la validation est fonction des paramètres suivants :

- attente de disponibilité du journal,
- temps d'exécution de la procédure de validation (ce paramètre conditionne en partie le premier).

Lorsque le journal est réalisé en technologie rapide non volatile, les accès sont suffisamment rapides pour que cette ressource ne constitue pas un point de congestion du système. Cette hypothèse est envisageable pour un système de taille limitée; compte tenu de la technologie, elle est aujourd'hui irréaliste au-delà d'une certaine charge (la prise en compte d'une centaine de transactions simultanées ayant un contexte d'exécution de 8 K. octets exigerait une mémoire de l'ordre du méga-octet pour le journal).

La solution classique consiste à réaliser le journal à l'aide d'un ou plusieurs fichiers sur disque. Dans ce cas, le temps d'accès au journal devient le paramètre clé.

Exemple : On suppose que le temps d'accès moyen au disque est de l'ordre de 20 millisecondes et que l'ensemble des informations propres à une transaction est transféré en une seule entrée-sortie sur le disque. Dans ce cas, le débit maximum du système est limité à 50 transactions par seconde. Ce chiffre est insuffisant pour certaines classes d'applications.

Une première façon d'améliorer ce rendement consiste à réaliser le journal sur plusieurs supports indépendants afin d'exécuter en parallèle une partie de la procédure de validation pour le compte de plusieurs transactions. La propriété de sérialisation exige alors que les enregistrements soient datés afin de reconstituer l'ordre effectif de validation lors d'une reprise éventuelle.

Une amélioration complémentaire consiste à regrouper les informations liées à la validation de plusieurs transactions en un seul enregistrement afin de limiter le nombre d'entrées-sorties physiques. Une solution de ce type est décrite dans [DeWitt 84]. Une transaction est exécutée en trois étapes : calcul, prévalidation et validation. Dans l'étape de pré-validation, l'enregistrement de validation de la transaction est simplement stocké dans une mémoire tampon prévue à cet effet. Les objets manipulés sont rendus visibles aux transactions concurrentes mais l'opérateur n'est pas averti de la terminaison de la transaction. La libération anticipée des objets d'une transaction crée des dépendances vis à vis des transactions qui manipulent ensuite ces objets. Ces dépendances n'ont pas de conséquences fâcheuses pour la cohérence tant que l'ordre de validation est respecté (voir section suivante traitant du contrôle de l'accès concurrent). Périodiquement, le tampon est vidé sur disque de telle sorte que l'étape de validation est commune à l'ensemble des transactions ; l'initiateur de chaque transaction est alors averti de la terminaison. Pèndant l'étape de prévalidation, l'état d'une transaction demeure "ACTIF"; une panne dans cet intervalle entraîne à la reprise, la réexécution des transactions en attente de validation. L'utilisation d'une mémoire permanente en technologie rapide pour la zone tampon élimine cet inconvénient et constitue ainsi un compromis coût/performance satisfaisant.

2.5. Coût de la reprise

Le coût d'un arrêt du système transactionnel est la somme des deux quantités suivantes :

- temps d'immobilisation jusqu'à la détection et la guérison de la défaillance,
- temps de redémarrage du système transactionnel. Ce temps comprend l'exécution de la procédure de reprise pour chaque transaction interrompue par la panne, et le temps de réexécution de ces transactions.

Nous avons vu que les journaux jouent un rôle essentiel dans la procédure de reprise. Si la reprise est impossible, en raison par exemple d'une défaillance des journaux, le prix à payer est beaucoup plus important car il est nécessaire de revenir à un état cohérent antérieur. la fiabilité des journaux est donc un facteur clé pour la disponibilité d'un système d'information. Ce point est abordé en 2.5.2.

En ce qui concerne le temps de redémarrage du système, il est plus rentable de s'efforcer de limiter le temps perdu à réexécuter les transactions interrompues que de chercher à optimiser la procédure de reprise. On utilise pour cela le concept de "point de reprise" présenté en 2.5.1.

Le paramètre le plus pénalisant dans un arrêt consécutif à une panne demeure cependant le temps d'immobilisation jusqu'à la remise en état du système. Pour éliminer, ou tout du moins raccourcir cette période, les systèmes dits à "haute disponibilité" utilisent une architecture tolérant les fautes basée sur la redondance des composants et la reconfiguration dynamique. Nous examinerons au chapitre 4 l'application de cette approche aux systèmes d'information transactionnels en vue de fournir aux utilisateurs un service "continu".

2.5.1. Reprise partielle : les points de reprise.

Jusqu'ici, il était implicitement admis que la transaction constituait à la fois l'unité de cohérence et de reprise. En cas d'erreur récupérable ou de panne du système, la transaction est réexécutée en entier. Ce type de fonctionnement est généralement désigné sous le terme de reprise globale. Dans un système où le temps d'exécution moyen des transactions est de l'ordre de la seconde, une reprise globale est acceptable compte tenu du fait que ce n'est pas une opération fréquente.

Cependant, nous avons vu au chapitre 1 des exemples d'applications qui mettent en jeu des transactions de longue durée (supérieure à la minute, voir de plusieurs heures), pour lesquelles une reprise globale est très pénalisante. L'exécution de la transaction peut même s'avérer impossible en cas de répétition d'incidents (cette situation est communément désignée sous le terme de "famine"). Pour éviter ce problème, on utilise le concept de **Point de Reprise** qui partitionne la transaction en éléments appelés **unités de reprise** (figure 2.4).

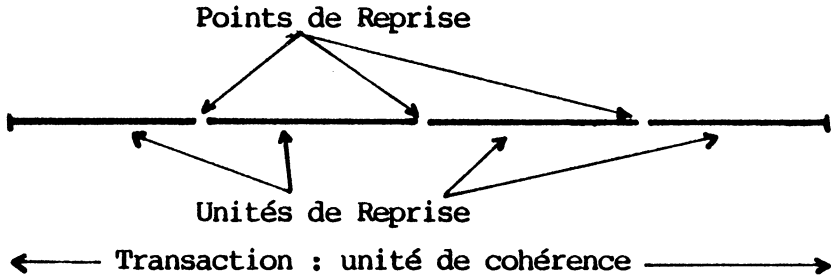


Fig. 2.4.

Les points de reprise sont définis par le concepteur à l'aide d'une action explicite notée POINT-Reprise. Sur occurrence de cette action, le système écrit dans le journal les informations suivantes :

- Indication d'un point de reprise (un début de transaction est un point de reprise implicite).
- Description des actions enregistrées dans l'espace de travail depuis le point de reprise précédent.
- Contexte d'exécution courant.

Le contexte d'exécution est constitué des objets qui sont globaux à plusieurs unités de reprise et dont la valeur constitue l'état initial de l'unité de reprise suivante. L'opération qui vient d'être décrite est équivalente à l'enregistrement d'un point de validation.

Il est important de noter la différence fondamentale entre les notions de point de reprise et de point de cohérence. En un point de reprise, les effets des actions d'une transaction ne sont pas rendus visibles à l'extérieur.

La périodicité des points de reprise est un compromis entre le cout d'un point de reprise (en terme d'overhead induit par la sauvegarde) et le coût d'une reprise (coût devant être pondéré par la fréquence des pannes). On trouve dans la littérature de nombreuses études d'évaluation portant sur ce thème [Gelenbe 77].

Ce mécanisme de reprise partielle ne s'applique pas systématiquement aux erreurs récupérables comme le montre l'exemple illustré par la figure 2.5.

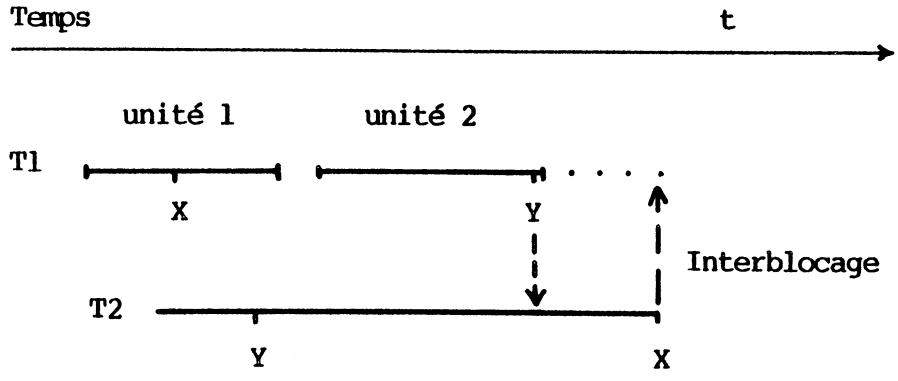


Fig. 2.5.

Dans cet exemple, un interblocage portant sur les objets X et Y est détecté entre les transactions T1 et T2 au temps t. Si l'algorithme de guérison de l'interblocage choisit d'abandonner T1, il est impératif de libérer l'ensemble des objets acquis par T1, en particulier X, acquis pendant la première unité de reprise. Par conséquent, la reprise ne peut pas être limitée à l'unité en cours.

2.5.2. Disponibilité de la mémoire secondaire.

Les mémoires permanentes sont le support des objets permanents (base de données) et des journaux sur lesquels sont stockées les variables d'états des transactions. Une panne a donc une répercussion sur l'un ou l'autre de ces deux ensembles de données.

a) Défaillance de la base

Lorsqu'une partie de la base est inaccessible ou erronée, il est nécessaire de restaurer l'information dans un état cohérent antérieur. Le principe de la méthode est le suivant :

- Périodiquement, une image de la base est sauvegardée sur un support indépendant. Cette sauvegarde constitue un point de contrôle.
- Les actions modifiant les objets permanents (et seulement celles-là) sont enregistrées chronologiquement sur un journal (généralement différent du journal utilisé en tant qu'extension de l'espace de travail des transactions.
- En cas de défaillance, la restauration s'effectue en deux temps. La base est restaurée avec l'image correspondant au point de contrôle antérieur, puis les actions permanentes enregistrées depuis ce point de contrôle sont réexécutées.

Une illustration de cette méthode est donnée dans la figure 2.6.

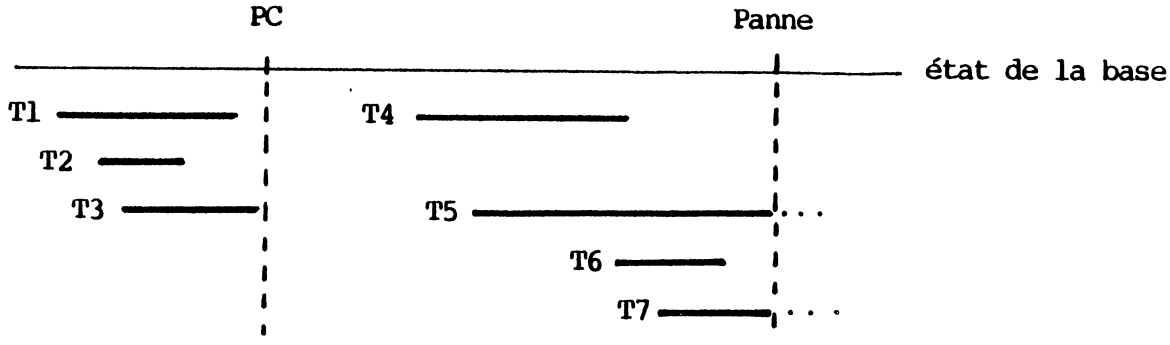


Fig. 2.6 Restauration après une défaillance de la mémoire secondaire

La reprise inclut les opérations suivantes :

. restauration de la base dans l'état correspondant au point de contrôle PC,
. réexécution des actions des transactions T4, T6 (T1, T2, T3 ne sont pas concernées par la panne),
. reprise des transactions T5 et T7 qui étaient actives au moment de la panne.

Comme pour les reprises partielles évoquées en 2.5.1, le paramètre critique dans la mise en oeuvre de cette méthode est la périodicité entre deux points de contrôle. Le meilleur choix résulte d'un compromis entre le coût d'un point de contrôle et le coût de réexécution des actions des transactions. Sur ce point, nous renvoyons le lecteur aux résultats des études déjà référencées en 2.5.1.

b) Défaillance des journaux

Les journaux détiennent des informations critiques pour la reprise des transactions. Leur disparition ou leur incohérence exige de revenir à un état cohérent antérieur.

La redondance des journaux est, de ce point de vue, un bon moyen d'accroître la fiabilité. [Lampson 77] décrit le principe d'une solution entièrement réalisée en logiciel, appelée "mémoire stable" mise en application dans le système XDFS [Mitchell 82]. Chaque opération d'écriture génère deux entrée-sorties sur deux supports indépendants. L'opération d'écriture est synchrone : on attend la terminaison des 2 entrées-sorties physiques et la seconde entrée-sortie n'est initialisée que lorsque la première est terminée avec succès. Il existe donc deux images physiques pour chaque information. En cas de panne, on vérifie la cohérence de chaque paire d'images physiques. Si les deux valeurs sont identiques, on est assuré que la panne n'a pas affecté l'opération d'écriture. Si elles sont différentes, la panne s'est produite

pendant l'opération d'écriture logique. Les deux images physiques représentent la valeur avant et après modification. Selon la procédure de reprise, l'une de ces deux images est prise comme valeur de référence. On trouve aujourd'hui des dispositifs matériels qui réalisent cette redondance au niveau du contrôleur disque. Cette solution désignée couramment sous le terme "disques miroirs" est coûteuse mais supprime le délai d'attente important engendré par une solution logicielle (figure 2.7).

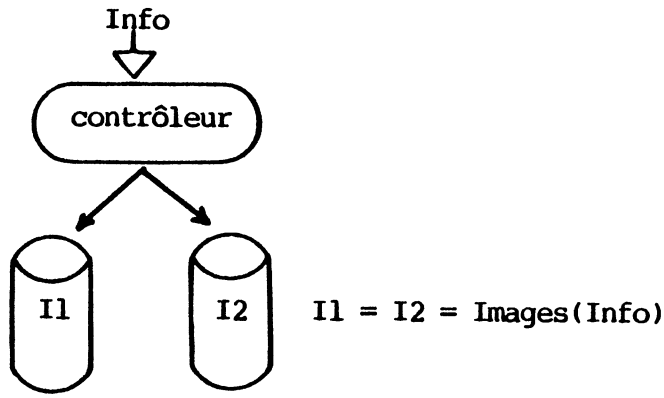


Fig 2.7 Disques miroirs

Ce dispositif peut naturellement s'appliquer aux objets permanents. Etant donné le coût de sa réalisation, il est aujourd'hui réservé aux journaux ou à un sous-ensemble de la base considéré comme critique.

Le système expérimental ENCHERE, réalisé à l'IRISA, dispose d'une mémoire stable rapide construite à partir de boîtiers 16 Kbits de RAM statique CMOS non-volatile. Cette mémoire fait partie de l'espace d'adressage du processeur, ce qui simplifie d'autant l'écriture des procédures de validation et de reprise [Banâtre 84].

3. VALIDATION ET CONTROLE DE L'ACCES CONCURRENT

3.1. Principes du contrôle de l'accès concurrent

Par définition, une transaction qui s'exécute sans incident conserve la cohérence de l'environnement. En conséquence, si les transactions sont exécutées séquentiellement, l'environnement passe par une suite discrète d'états cohérents. Cependant, l'exécution simultanée de plusieurs transactions est rendue nécessaire pour satisfaire un double objectif de qualité de service et de performance globale. Le parallélisme permet, d'une part l'accès simultané au système par le plus grand nombre possible d'utilisateurs interactifs et assure, d'autre part, une meilleure utilisation des ressources physiques du système (unité centrale, mémoire centrale, dispositifs d'entrée-sortie, etc.).

L'exécution en parallèle des transactions implique que plusieurs d'entre elles peuvent opérer simultanément sur les mêmes objets. Des exemples d'incohérences provoqués par l'accès incontrôlé à des objets partagés ont été présentés dans le chapitre 1 (§2.1.4).

Le principe du contrôle de l'accès concurrent consiste à ordonnancer les actions des transactions concurrentes, de telle sorte que le résultat de leur exécution soit équivalent à celui d'une exécution séquentielle. C'est la propriété de "**sérialisation**", associée à la notion de cohérence forte [Papadimitriou 79]. Les applications ne nécessitant pas la cohérence forte se satisfont d'ordonnancements plus généraux. Ceci se traduit par une diminution des synchronisations et donc par un accroissement du parallélisme. Les développements qui suivent s'appliquent à la cohérence forte; l'examen des cas de cohérence dégradée est reporté en fin de chapitre.

La sérialisation est basée sur la détection des conflits d'accès entre les transactions concurrentes. Il y a conflit d'accès lorsque deux transactions exécutent des actions incompatibles sur un objet.

Par exemple LIRE-D et ECRIRE-D ou bien ECRIRE-D et ECRIRE-D (deux actions LIRE-D sur le même objet ne sont pas incompatibles). Les conflits entre transactions concernent uniquement l'accès aux objets dits "permanents" selon la nomenclature définie au § 1.1. (les objets "temporaires" et les objets "réels" ne sont pas partageables). La détection des conflits porte sur les primitives Lire-Base et Préécrire-Base. Il est important de noter qu'un conflit n'est pas synonyme d'incohérence, comme le montre l'exemple de la figure 2.8.

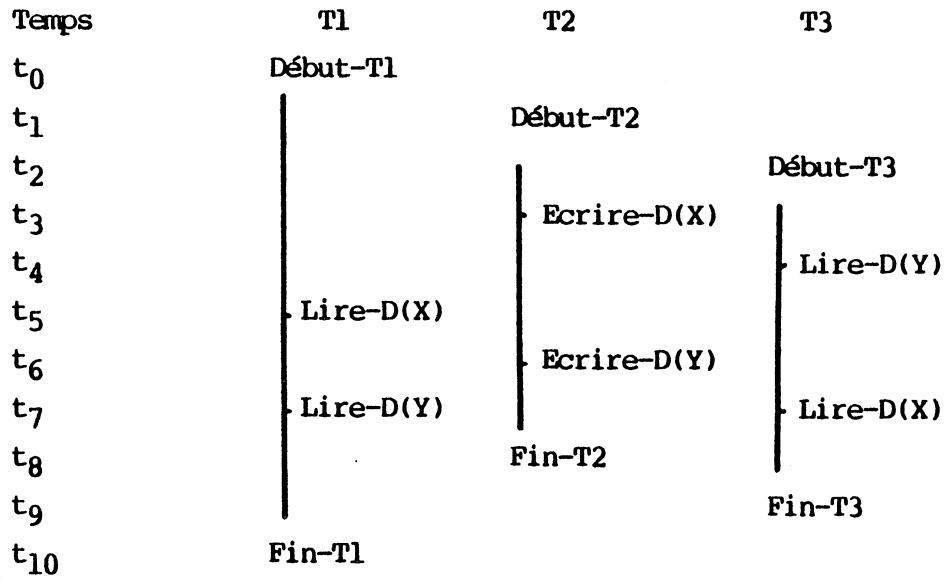


fig. 2.8 Exemples d'exécutions concurrentes

L'exécution concurrente de T1 et T2 est cohérente car elle est équivalente à l'exécution en séquence de T2 puis T1. Pourtant, cette exécution génère à deux reprises un conflit entre T1 et T2 : au temps t_5 pour l'objet X et au temps t_7 pour l'objet Y. En revanche, l'exécution concurrente de T2 et T3 est incorrecte, car elle ne correspond à aucune des deux séquences T2-T3 ou T3-T2.

L'apparition d'un conflit crée une relation de dépendance (notée $\prec$) entre les transactions qui tient compte des caractéristiques du conflit et de l'ordre chronologique des opérations sur l'objet partagé. Dans l'exemple de la figure 2.6, l'exécution concurrente de T1 et T2 crée deux fois la même dépendance (T1 $\prec$ T2) alors que l'exécution de T2 et T3 crée deux dépendances contradictoires (T2 $\prec$ T3 et T3 $\prec$ T2).

On peut résumer les relations de dépendances entre ces trois transactions à l'aide du graphe représenté en figure 2.9. C'est le graphe des dépendances.

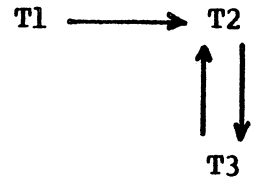


fig. 2.9 Graphe de dépendance

La présence d'un cycle dans le graphe des dépendances est le signe d'une exécution non sérialisable.

On trouve dans [Boksenbaum 84] une analyse des algorithmes de contrôle de l'accès concurrent fondée sur l'étude des propriétés du graphe des dépendances. Nous adopterons la classification des mécanismes de contrôle proposée par les auteurs, en deux groupes selon que le contrôle de dépendance :

- a lieu à chaque conflit : méthodes de **contrôle continu,**
- est reporté à la fin de chaque transaction (avant la validation) : méthodes de **contrôle par certification .**

Le problème du contrôle de l'accès concurrent a fait l'objet de très nombreuses publications et les algorithmes proposés sont variés. La description exhaustive de ces mécanismes n'est pas du domaine de cet ouvrage. Pour cela, nous renvoyons le lecteur aux références citées dans le cours de cette section. Pour chaque classe d'algorithmes, nous nous limitons à décrire le principe de la méthode et nous étudions les corrélations avec les mécanismes de validation et reprise.

3.2. Contrôle continu

Les méthodes qui entrent dans cette catégorie ont été les plus étudiées et sont aujourd'hui les plus répandues tant dans les produits que dans les prototypes de recherche. Elles ont fait l'objet d'une étude de synthèse très complète dans [Bernstein 81] et [Kohler 81]. Pour le détail du fonctionnement des algorithmes, nous renvoyons donc le lecteur à ces deux ouvrages. Cette classe se décompose elle-même en deux sous-ensembles : le verrouillage à deux phases et l'ordonnancement par estampillage.

3.2.1. Verrouillage à deux phases

a) Principe de fonctionnement

La relation de dépendance est définie uniquement par l'ordre chronologique des opérations sur l'objet partagé. Pour traduire cette dépendance en terme d'exécution séquentielle, on oblige la transaction à l'origine du conflit à attendre la libération de l'objet par la transaction concurrente. Dans cette méthode, le graphe des dépendances représente aussi un graphe des attentes entre les transactions. Pour détecter les conflits, les deux actions suivantes sont introduites :

- **VERROUILLER** (objet, mode exclusif ou partagé)
  Demande d'accès à un objet en mode exclusif ou partagé. Si l'objet est libre pour le type d'accès requis, le verrouillage est accepté et la transaction continue. Dans le cas contraire, il y a conflit : le verrouillage est retardé et la transaction est mise en attente.

- **RELACHER** (objet)
  Le déverrouillage d'un objet entraîne la libération d'une ou éventuellement plusieurs transactions en attente de cet objet.

Dans [Eswaran 76], les auteurs montrent que les conditions suivantes sont suffisantes pour assurer une exécution sérialisable :

* la transaction est "bien formée", c'est à dire :

  - tout accès à un objet est précédé d'une opération de verrouillage compatible avec le type d'accès envisagé,
  - tous les objets sont libérés en fin de transaction.

* le verrouillage est à "deux phases", c'est à dire qu'aucune demande de verrouillage n'est acceptée une fois que la première demande de libération a été exécutée. Il y a donc deux phases dans l'exécution d'une transaction : une première phase au cours de laquelle la transaction acquiert des objets, puis une deuxième phase pendant laquelle tous ces objets sont relâchés.

b) Liens avec les procédures de validation et reprise

L'exemple de la figure 2.10 montre que l'action RELACHER n'est pas réversible.

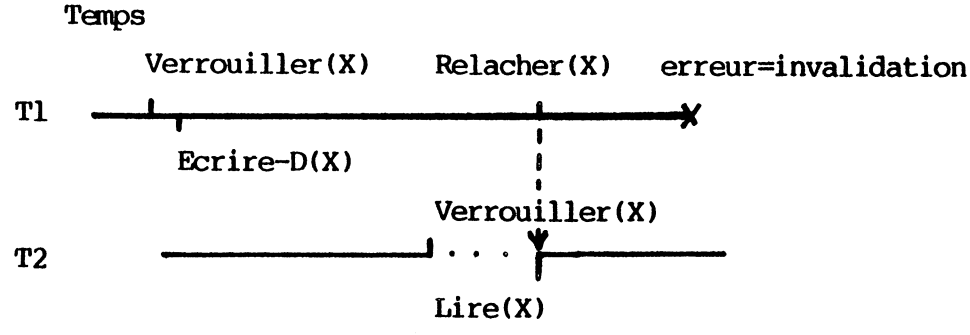


fig. 2.10.

Dans cet exemple, la règle du verrouillage à deux-phases est respectée. L'invalidation de T1 entraîne l'annulation de l'action effectuée sur l'objet X. Malheureusement, cette action a été rendue visible à l'extérieur par le biais de l'opération RELACHER.

La propriété P2 n'est pas respectée. Le retour à un état cohérent exigerait que T2 à son tour soit invalidée. C'est impossible si T2 a déjà été validée (propriété P3). Nous avons vu au §1.2.2 que les actions non réversibles doivent être différées jusqu'à la fin de la transaction : c'est le cas pour l'action RELACHER.

Les systèmes d'informations qui garantissent la cohérence forte utilisent la technique du "verrouillage implicite" dans laquelle les actions VERROUILLER sont implicitement associées aux primitives Lire-Base et Préécrire-Base tandis que les actions RELACHER sont associées à l'exécution de l'action FIN-Transaction. Les verrous associés aux objets non modifiés sont relachés immédiatement; ceux qui sont liés aux objets qui ont été modifiés sont relachés au cours de l'étape de validation après que les mises à jour aient été consolidées dans la base de données.

c) Gestion des interblocages

Le verrouillage dynamique et les attentes consécutives aux conflits créent les conditions d'un interblocage. Un interblocage se caractérise par un cycle dans le graphe des attentes. Il y a trois manières de tenir compte des interblocages :

- En les ignorant. Une borne supérieure est fixée au temps d'exécution de chaque transaction. Lorsqu'une transaction dépasse cette limite, elle est abandonnée, le dépassement étant interprété comme la conséquence d'un éventuel interblocage.

- En les détectant (DETECTION-GUERISON). La détection d'un cycle dans le graphe des attentes est aisée dans un système centralisé. L'abandon d'une transaction du cycle élimine l'interblocage. Le choix de la victime varie selon les méthodes.

- En empêchant leur formation (PREVENTION). On distingue deux sortes de prévention :

. La prévention statique qui consiste à attribuer globalement l'ensemble des objets nécessaires avant l'exécution d'une transaction (verrouillage statique). Aucun objet n'est alloué à la transaction tant que la totalité n'est pas disponible. Une solution plus souple consiste à allouer les objets en cours d'exécution en respectant le même "ordre" pour toutes les transactions [Lomet 78]. Ces techniques supposent que l'ensemble des objets manipulés soit connu à l'initialisation de la transaction. Cette contrainte est très forte et ne s'applique pas dans la plupart des cas, où leur identité est le résultat d'un algorithme. C'est pourquoi nous éliminons cette classe de solutions.

. La prévention dynamique [Rosenkrantz 78], désignée aussi sous le terme "évitement", consiste à exécuter à chaque conflit une procédure destinée à empêcher la formation d'un cycle dans le graphe des attentes. Dans la suite de ce rapport, le terme "prévention" fera référence uniquement à cette méthode. Cette technique est fondée sur l'existence d'une relation d'ordre entre les transactions (créée à l'aide des estampilles par exemple). Cette relation d'ordre fait foi à chaque conflit pour autoriser l'attente ou rejeter une transaction. Le lecteur trouvera en annexe A une description de ces méthodes.

Quelle que soit la technique adoptée, la gestion des interblocages entraîne parfois la reprise d'une transaction. C'est ce que nous avons appelé "erreur récupérable" dans le § 1.4.

3.2.2. Ordonnancement par Estampillage

a) Principe de fonctionnement

Le principe des techniques d'ordonnancement par estampillage consiste, lorsqu'il y a conflit à forcer les transactions à s'éxécuter selon un ordre de sérialisation pré-établi entre les transactions.

On utilise l'estampillage pour générer la relation d'ordre : à l'interprétation de l'action DEBUT-Transaction, le système crée une estampille (compteur universel) qui permet de "dater" toutes les actions d'une transaction. Lorsqu'il y a conflit, le contrôleur vérifie que les accès aux objets sont exécutés dans l'ordre pré-établi entre les transactions à l'aide de l'estampillage. Si c'est le cas, l'opération est acceptée. Sinon la transaction est abandonnée (conflits Lire-écrire), ou simplement retardée (conflit écrire-écrire : voir point b).

Dans cette méthode désignée aussi sous le terme d'estampillage statique, l'ordre de sérialisation est défini a priori et ne tient pas compte des conflits entre les transactions. Il existe une technique plus souple, désignée sous le terme d'estampillage dynamique, dans laquelle l'ordre de sérialisation n'est fixé qu'à l'apparition du premier conflit. De nombreuses autres améliorations ont été proposées dans la littérature en vue de diminuer le nombre de rejets [Reed 78, Thomas 79, Bernstein 80].

b) Liens avec les procédures de validation et reprise

Les opérations d'écriture (primitives Ecrire-Base) sont forcées de s'éxécuter dans l'ordre des estampilles. Cela conduit dans certains cas à retarder l'étape de validation d'une transaction. Etant donné que ces attentes sont ordonnées par les estampilles, il n'y a pas de risque d'interblocage. Comme pour le verrouillage, le rejet d'une transaction pendant la phase de calcul est considéré comme une erreur récupérable.

3.3. Contrôle par certification

a) Principe de fonctionnement

Cette approche laisse les dépendances s'établir pendant l'étape de calcul et diffère les contrôles jusqu'à la fin de cette étape. Une transaction est structurée en trois étapes : calcul, certification et validation selon le schéma représenté dans la figure 2.11.

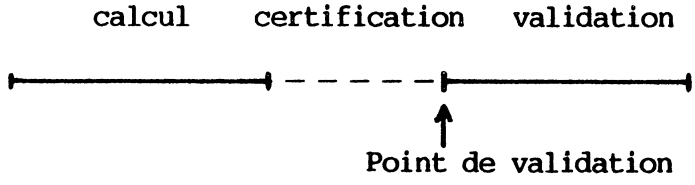


Fig.2.11. Contrôle par certification.

Au cours de l'étape de certification, on vérifie la consistence des actions effectuées par la transaction ; c'est donc un contrôle a posteriori. Si l'étape de certification décèle une opération incompatible avec celles des transactions concurrentes, la transaction est reprise. Dans le cas contraire, la transaction est validée.

On trouve dans la littérature plusieurs stratégies de certification. Dans la méthode décrite dans [Kung 81], les actions d'une transaction sont acceptées si elles ne sont pas en conflit avec celles des transactions qui ont été certifiées avant elle. l'ordre de sérialisation correspond donc à l'ordre chronologique de certification des transactions.

Dans [Boksenbaum 84] les auteurs proposent un raffinement du contrôle par certification qui prend aussi en compte les actions des transactions actives, autorisant ainsi un ordre de sérialisation différent de l'ordre chronologique des certifications.

b) Liens avec les procédures de validation-reprise

Il y a deux manières de considérer la certification dans la vie d'une transaction. Dans le premier cas, la certification fait partie de l'étape de calcul dont elle est l'aboutissement. L'état de la transaction est ACTIF pendant la certification et une panne entraîne la reprise totale de la transaction. L'alternative consiste à considérer la certification comme une étape autonome, atomique, représentée par l'état CERTIFIE. Si elle est interrompue par une panne, elle est réexécutée (figure 2.12). Pour permettre cette reprise, il est nécessaire que la description des actions permanentes ait été sauvegardée au préalable sur un journal. Cette précaution n'est pas nouvelle puisqu'elle constitue également un préalable à l'entrée dans l'étape de validation.

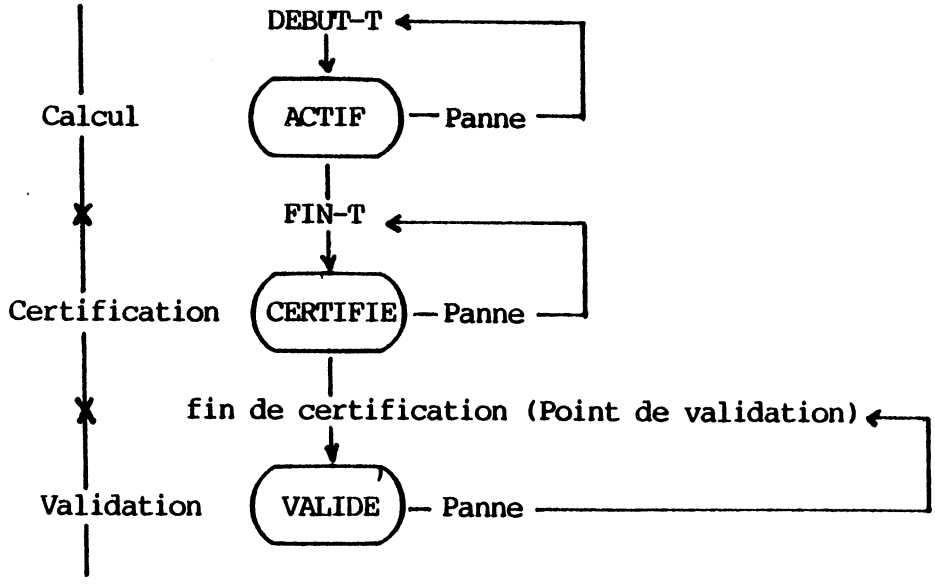


Fig. 2.12 Etats d'une transaction.

Le choix de la meilleure solution est un compromis entre le coût supplémentaire induit par la journalisation d'un nouvel état et le gain obtenu lors d'une reprise après panne.

3.4 Comparaison des techniques

Ce paragraphe présente une analyse comparative des diverses classes d'algorithmes établie à partir des critères d'évaluation suivants : risque de famine, granularité des objets, généralité de la méthode et performance.

a) Risque de famine

La famine (ou privation) désigne la propriété de certains algorithmes de ne pas pouvoir garantir la terminaison d'une transaction soit parce qu'elle attend indéfiniment, soit parce qu'elle est reprise et rejetée indéfiniment, soit parce que la reprise est retardée indéfiniment.

Dans le verrouillage à deux phases, il est possible d'adopter comme critère de sélection d'une victime (en cas de détection ou de prévention d'un interblocage) "l'âge" respectif de chaque transaction. Une stratégie, qui donne priorité à la transaction la plus ancienne, élimine le risque de famine. En effet, une transaction, abandonnée puis réexécutée, a "vieilli" par rapport à ses concurrentes et ses chances de succès en sont augmentées.

Cette propriété ne s'applique pas à l'estampillage ou à la certification. Ainsi, si on excepte le cas de l'estampillage conservatif, les méthodes d'estampillage et de certification sont sujettes à la famine. En cas de rejet répété d'une transaction, il est nécessaire de lui conférer une "priorité" absolue sur ses concurrentes lui garantissant les ressources indispensables à sa terminaison. Outre un accroissement de complexité, ce mécanisme s'accompagne d'une réduction du parallèlisme consécutive au blocage de certaines transactions en attente de l'exécution de la transaction déclarée prioritaire.

b) Granularité

La granularité désigne la nature des objets sur lesquels porte le contrôle d'accès (enregistrement logique, bloc ou page physique, partition d'une base,...). Une granularité fine réduit la probabilité de conflit mais accroît le coût de détection et de gestion de ces conflits. Une granularité forte a les effets inverses. [Ries 79] et [Potier 80] ont tenté de déterminer la granularité optimale pour un système qui utilise le verrouillage à deux-phases. En partant de lois de distribution des accès différentes, ces études parviennent à des conclusions opposées.

En fait, le choix d'une granularité est conditionné par la nature de l'application. Un système qui permet de gérer simultanément plusieurs types de granularité procure une souplesse d'utilisation appréciable. Le verrouillage à deux-phases est la seule technique qui remplisse cette condition au prix d'une complexité supplémentaire dans la gestion des verrous (verrouillage hiérarchique décrit dans [Gray 80]).

c) Généralité

Ce critère d'évaluation fait référence à l'aptitude des techniques de contrôle considérées à prendre en compte n'importe quel type de ressource et plusieurs niveaux de cohérence.

Les processus associés à l'exécution des transactions utilisent d'autres ressources que les objets permanents (mémoire, disque, réseau, etc.). L'estampillage et la certification ne permettent pas, sans aménagement sérieux, de gérer ce type de ressource. En revanche, le verrouillage s'applique sans restriction. Il est donc possible d'envisager à terme l'existence d'un mécanisme unique de gestion des ressources, au niveau du système d'exploitation.

L'ensemble des méthodes présentées assure la cohérence forte. Seul le verrouillage offre la possibilité de générer des niveaux de cohérence dégradés sans payer le prix de la cohérence forte. Il suffit pour cela de fournir au concepteur d'application une interface de programmation qui offre la visibilité des actions VERROUILLER et LIBERER. Il est alors possible de s'affranchir de la règle du verrouillage à deux phases et de contrôler entièrement l'acquisition et la libération des objets.

d) Performance

En théorie, un accroissement du parallélisme d'exécution devrait se traduire par une augmentation d'efficacité du système. En conséquence, l'algorithme qui permet le maximum d'ordonnancements parallèles pour un jeu de transactions donné, devrait constituer le meilleur choix. Le nombre d'ordonnancements parallèles est limité par l'établissement des dépendances entre transactions. Dans ce domaine, les facteurs clés sont : l'instant où s'établit la dépendance et le critère retenu pour la création d'une dépendance (nature du conflit). De ce point de vue, les méthodes de certification sont bien placées puisqu'elles suppriment toute forme de synchronisation pendant l'étape de calcul. A l'opposé, on trouve le verrouillage qui, non seulement crée une dépendance à l'apparition d'un conflit, mais aussi impose une attente parfois inutile, très préjudiciable au parallélisme.

En fait, ce critère d'évaluation est insuffisant pour caractériser le comportement d'un algorithme de contrôle de la concurrence car il ne tient pas compte de la perte d'efficacité entrainée par le rejet des transactions. C'est pourquoi nous préferons évaluer l'efficacité en mesurant le temps de réponse d'une transaction exprimé à l'aide de la formule suivante :

$$T = Te + Ts + Ta + Tp \qquad \text{où}$$

. Te représente le temps d'exécution effectif de la transaction.

. Ts représente le temps passé dans le système pour la synchronisation des accès (détection de conflit, établissement des dépendances).

. Ta représente le temps d'attente dû aux conflits.

. Tp représente le temps perdu à cause d'un rejet de la transaction. (c'est à dire le temps réel d'exécution jusqu'au rejet de la transaction).

Lorsqu'on considère le temps de réponse, on s'intéresse en fait à deux quantités : la valeur moyenne et l'écart autour de la moyenne (variance). L'étude de la variance est fondamentale car elle seule permet de mettre en évidence le phénomène de famine.

Te est une quantité incompressible quelque soit l'algorithme ; Ts est d'un ordre de grandeur inférieur à Ta ou Tp. En conséquence, la comparaison porte essentiellement sur l'évolution respective du temps d'attente Ta et du temps perdu Tp.

Dans la certification et l'estampillage, Ta est faible; il correspond à l'attente éventuelle dans l'étape de validation. En revanche, Ta est généralement important dans le verrouillage à deux phases. Tp est d'autant plus important que le rejet est tardif. La valeur la plus élevée est donc obtenue pour la certification qui décide le rejet uniquement en fin de transaction. Cette stratégie pénalise particulièrement les transactions de longue durée. Dans le verrouillage, le temps perdu dépend beaucoup de la méthode choisie pour la gestion des interblocages. La prévention accroît cette quantité dans une proportion non négligeable. En contrepartie, le temps d'attente est diminué, de telle sorte que le bilan est globalement positif [Balter 82]. Ce point sera développé en détail au chapitre 4.

Les études de performances publiées dans ce domaine [Lin 81, Madelaine 82, Galler 82, Agrawal 83, Kohler 83, Carey 84] sont difficilement exploitables car elles sont le plus souvent fondées sur des hypothèses de comportement différentes. De l'ensemble de ces travaux, on peut cependant dégager les éléments suivants :

- Les paramètres clés du comportement d'un mécanisme de contrôle de la concurrence sont le taux de conflit et la durée des transactions. Le taux de conflits est lui-même un paramètre délicat à estimer car il tient compte du nombre de transactions en concurrence, du nombre moyen d'accès et de leur type (partagés ou exclusifs), de la taille de la base, de la fréquence d'accès aux objets partagés, etc.

- Les méthodes de certification donnent d'excellents résultats lorsque le taux de conflit est faible. C'est pourquoi elles sont généralement désignées sous le terme d'"optimistes" car elles font l'hypothèse que les transactions s'exécutent sans conflit jusqu'à leur terminaison. Cette méthode donne d'excellents résultats lorsque les transactions sont courtes (*) ou lorsque la proportion de transactions en "consultation seule" est grande.

- Le verrouillage à deux phases est préférable lorsque le taux de conflit est fort ou lorsque les transactions sont longues. Les résultats complémentaires concernant les différences de comportement entre les méthodes de détection et de prévention sont présentés au chapitre 4.

- Les méthodes fondées sur l'estampillage se situent entre les deux. En outre, le gain de performance attendu en utilisant des améliorations de l'algorithme de base (règle de Thomas, versions multiples d'un objet, etc) n'est pas significatif.

* les qualificatifs "court" et "long" s'appliquent ici au nombre d'objets manipulés par une transaction.

3.5 Conclusion : des mécanismes flexibles

Les remarques précédentes montrent à l'évidence qu'aucune technique ne peut prétendre s'imposer dans toutes les configurations. Cette conviction a conduit plusieurs auteurs à rechercher des combinaisons possibles de plusieurs algorithmes dans le but de définir un mécanisme regroupant les avantages respectifs de chacun d'eux.

La première proposition en ce sens est décrite dans [Bayer 82] sous la forme d'un algorithme mixte intégrant le verrouillage et la certification. En fonction du type de conflit rencontré (Lire-Ecrire ou Ecrire-Ecrire), le système applique l'un des deux mécanismes. [Bernstein 81] propose un mécanisme équivalent basé sur l'utilisation du verrouillage et de l'estampillage. Dans [Lausen 82], le choix du mécanisme à appliquer (verrouillage ou certification) dépend du type de la transaction et non du type de conflit.

Ces techniques sont "statiques" en ce sens que la stratégie à appliquer est prédéfinie par la nature du conflit ou de la transaction. On trouve aujourd'hui des méthodes adaptatives appliquant dynamiquement des stratégies différentes en fonction de la configuration rencontrée. [Robinson 84] regroupe au sein d'un noyau de contrôle de concurrence le verrouillage et la certification. En fonction de la charge, le système décide l'utilisation d'une des deux méthodes et l'applique à toutes les transactions. Un prototype a été réalisé sur l'architecture multiprocesseur Cm* de Carnegie-Mellon. Dans [Boral 84], un seul algorithme de base est composé de plusieurs ensembles de règles de synchronisations inspirées du verrouillage et de la certification. En fonction de son profil, chaque transaction se voit appliquer la règle la plus appropriée.

Ces travaux présentent sur le plan théorique un intérêt majeur en démontrant formellement que la cohabitation de plusieurs mécanismes est possible. Des études complémentaires sont nécessaires pour confirmer, par des évaluations, l'intérêt effectif de ces méthodes mixtes malgré le surcroit de complexité qu'elles entraînent.

Chapitre

III

Contrôle de la Cohérence dans les Systèmes d'Information Répartis

SOMMAIRE DU CHAPITRE III

0. INTRODUCTION

Ce chapitre se présente comme le prolongement du chapitre précédent pour un environnement distribué. Il est organisé de façon identique.

La section 1 reprend le modèle de système transactionnel décrit dans le chapitre précédent et l'étend pour prendre en compte la répartition des données et des traitements. Ce modèle est suffisamment primitif pour permettre la description de n'importe quel type de système d'information réparti et, par conséquent, ne limite pas la portée des résultats qui suivent à des configurations particulières. Ce modèle met en évidence la nécessité d'une synchronisation par message entre les contrôleurs et en étudie l'impact sur les mécanismes de contrôle de la cohérence. Cette première section se termine par une description des hypothèses retenues pour le réseau de communication.

La section 2 est consacrée aux mécanismes de validation et de reprise. L'accent y est mis sur les conséquences d'un partitionnement du réseau en sous-réseaux disjoints susceptibles de diverger dans leurs décisions. Un protocole de base est présenté dont les détails de réalisation sont reportés au chapitre suivant.

Le contrôle de l'accès concurrent fait l'objet de la section 3. Comme dans le chapitre précédent, nous ne consacrons pas d'étude de fond à cet aspect et nous nous limitons à décrire le principe de chaque méthode et les liens avec les procédures de validation-reprise.

1. MODELE D'UN SYSTEME D'INFORMATION DISTRIBUE

1.1. Les éléments du modèle

Sur chaque site, un système transactionnel contrôle l'exécution de transactions qui sont des programmes s'exécutant sur ce site et ne manipulant que des objets locaux.

$T = [\ Aj\ ,\ Oj\ /\ j=1,m\ ]$ où Aj désigne une action sur un objet Oj local au site d'exécution de T

Lorsqu'on connecte entre eux plusieurs systèmes transactionnels, on peut réaliser une application distribuée en faisant coopérer les transactions. Cette coopération se traduit par le lancement de transactions à distance et l'échange de messages entre transactions s'exécutant sur des sites distincts.

On étend le concept de transaction à cette coopération en la désignant sous le terme de **transaction globale**. L'exécution des transactions coopérantes doit vérifier dans son ensemble la propriété d'**atomicité**.

Dans la suite, on désignera sous le terme d'**agent** chaque transaction coopérante et on réservera le terme **transaction** pour désigner le groupe d'agents pour lesquels les systèmes transactionnels garantissent la propriété d'atomicité.

$T = [\ Ti\ /\ i=1,n\ ]$ (Ti est un agent de T)

avec $Ti = [\ Aij\ ,\ Oj\ /\ j=1,m\ ]$

Le modèle de transaction présenté au chapitre 2 est modifié et enrichi des actions suivantes pour prendre en compte la coopération entre les agents d'une transaction.

DEBUT-Transaction : Début d'une transaction.

Un identificateur universel, l'estampille, est affecté à la transaction. Pour une analyse exhaustive des méthodes de génération d'estampilles en environnement réparti, nous renvoyons le lecteur à [Rahimi 82].

DEBUT-Agent : Début d'un agent.

Le MGT local crée un espace de travail pour cet agent.

Un début de transaction correspond en fait à l'initialisation du premier agent de cette transaction (agent initial). Pour cet agent particulier, les deux actions DEBUT-T et DEBUT-A sont confondues.

INIT-Agent (nom de site, nom de programme, paramètres)

Demande d'activation d'un nouvel agent pour exécuter un certain programme avec des paramètres initiaux.

Cette action n'est pas bloquante pour l'agent invocateur autorisant ainsi les traitements parallèles à l'intérieur d'une transaction. Le message généré par cette action véhicule, entre autres informations, l'estampille de la transaction. Nous verrons ultérieurement l'usage qui est fait de l'estampille dans le contrôle de cohérence.

L'exécution de cette action fournit en résultat une identification locale de l'agent distant (*). Celle-ci est utilisée ensuite dans les actions de synchronisation ENVOI-agent et ATTENTE-Agent.

(*) La description détaillée des structures d'adressage n'est pas indispensable à la compréhension des mécanismes de contrôle ; elle ne sera donc pas abordée ici.

ENVOI-Agent (identification d'agent récepteur, texte).
Envoi d'un message (texte) à un autre agent.

Cette action n'est pas bloquante et suppose que l'agent récepteur a déjà été activé.

ATTENTE-Agent (identification d'agent émetteur).
Attente d'un message en provenance d'un agent.
Cette action est bloquante.

FIN-Agent : Fin d'exécution d'un agent.
L'agent est prêt à exécuter le protocole de validation globale, destiné à synchroniser les validations de tous les agents de la transaction (voir section 2).

FIN-Transaction : Fin d'éxécution de la transaction.
Par symétrie avec l'action DEBUT-Transaction, les actions FIN-Transaction et FIN-Agent sont confondues pour l'agent initial.

ARRET-Agent : Abandon de l'exécution d'un agent.
Cette action déclenche l'exécution d'un protocole d'abandon destiné à arrêter l'exécution de tous les agents de la transaction.

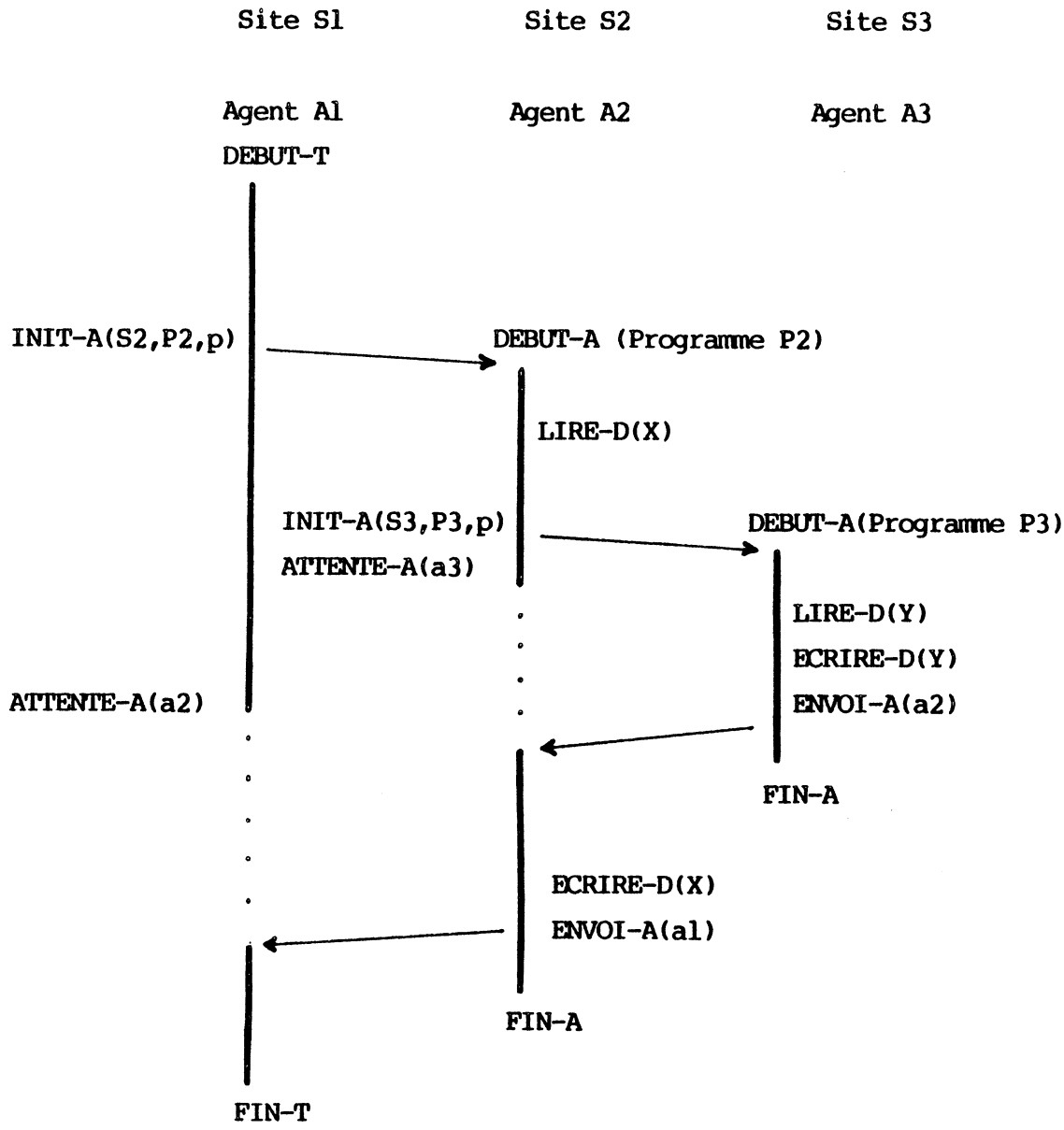
Dans la suite, ces actions seront désignées par les notations abrégées suivantes : DEBUT-T, DEBUT-A, ENVOI-A, FIN-A, ATTENTE-A, FIN-T, ARRET-A.

Les actions LIRE-Donnée, ECRIRE-Donnée, LIRE-Terminal, ECRIRE-Terminal et SORTIR-Document (notées en abrégé LIRE-D, ECRIRE-D, LIRE-T, ECRIRE-T et SORTIR-D) ont la même interprétation qu'en centralisé. Toutefois, LIRE-T et ECRIRE-T, qui assurent le dialogue entre la transaction et l'opérateur, ne sont permises que dans la programmation de l'agent initial.

Ce qui vient d'être présenté constitue un modèle simple de processus coopérants enrichi par le concept de transaction globale auquel est attachée la propriété d'atomicité. Ce modèle autorise une grande variété de scénarios d'exécution répartie : migration, appel procédural, exécution parallèle, co-routines. L'annexe B présente un exemple d'application répartie, inspirée du domaine bancaire, décrit à l'aide des éléments du modèle. L'annexe C montre comment le modèle peut également être utilisé pour décrire diverses catégories de systèmes d'information répartis.

Dans une approche du type "transactionnel coopérant", un agent éxécute un programme préenregistré sur un site. La programmation d'un agent fait apparaître explicitement les relations possibles avec les autres agents (activation, dialogue) et le contrôle des schémas d'exécution reste sous la responsabilité du programmeur d'application.

Une transaction est initialisée sur un site par l'exécution d'un agent (l'agent initial) qui peut demander l'exécution, sur des sites différents ou non, d'autres agents dont l'identité est dictée par l'application et peut être éventuellement le résultat d'un algorithme. Ce schéma est transitif et engendre un arbre d'invocation. Par la suite, deux agents d'une même transaction peuvent dialoguer en fonction des besoins de l'application. (figure 3.1).



a1, a2, a3 sont les identificateurs des agents A1, A2, et A3

Figure 3.1

Transactionnel coopérant : Anatomie d'une transaction

1.2. Contrôle de la cohérence dans un environnement réparti

1.2.1. Caractéristiques d'un environnement réparti

L'évocation générale des problèmes posés par la gestion de processus communiquants et l'accès à des ressources distribuées est en dehors des objectifs de ce paragraphe. Nous nous limiterons à évoquer les aspects fondamentaux pour le contrôle de la cohérence et nous renvoyons le lecteur à [Cornafion 81] pour une description complète des techniques d'adressage et de contrôle en environnement réparti.

Le facteur essentiel de discrimination entre un système centralisé et un système réparti est l'absence de mémoire commune qui se traduit par :

- l'absence d'état global du système. Chaque contrôleur n'a qu'une vue partielle de l'état du système,

- l'impossibilité de réaliser les synchronisations par les procédés traditionnels. Une synchronisation par message est nécessaire.

L'utilisation d'un support de communication pour acheminer les messages pose le double problème des délais de transmission d'une part et de la perte du message à cause d'une défaillance d'autre part. Dans un système réparti, la durée d'une opération de synchronisation entre deux processus distants est longue par rapport au temps qui s'écoule entre deux transitions d'état du processus émetteur. Il en résulte un décalage entre l'état réel du processus émetteur et la vue que peut en avoir le processus récepteur.

Ce phénomène est encore aggravé par la variance du délai de transmission : deux sites en copie d'un message expédié par un troisième ne sont pas assurés de les recevoir dans une même fenêtre de temps. Cet indéterminisme est particulièrement sensible dans les systèmes dits "faiblement couplés" dans lesquels les sites sont reliés par un réseau maillé à commutation de paquets (on rappelle que dans le réseau TRANSPAC, par exemple, le délai de transmission d'un message est de l'ordre de 200 millisecondes).

Dans un système centralisé, une panne entraîne l'arrêt de tous les processus en cours. Au redémarrage du système, l'exécution d'une procédure de reprise permet de repartir dans un état cohérent. Dans un système réparti la panne d'un site interrompt les activités sur ce site mais n'affecte pas directement le déroulement des activités sur les autres sites. La prise en compte des pannes nécessite deux étapes de synchronisation :

- lorsque la panne est détectée, les partenaires restant en lice se mettent d'accord sur la conduite à adopter : continuer ou arrêter,

- lorsque le site défaillant se reconnecte au système général, il doit prendre une décision compatible.

Une défaillance du réseau a des conséquences encore plus fâcheuses puisqu'elle crée un partitionnement du système réparti en sous-ensembles disjoints, c'est-à-dire, dans l'impossibilité de communiquer. Si des systèmes déconnectés laissent des agents coopérants poursuivre leur exécution, des incohérences peuvent apparaître des suites de décisions incompatibles : il y a donc un risque de divergence des sous-systèmes. Chacun d'eux réagit à ce type d'incident en décidant la continuation ou l'arrêt des agents qui coopéraient avec le sous-système disparu. L'incertitude née de l'absence d'état global ne permet pas d'apporter une solution dans tous les cas.

1.2.2. Conséquences sur le contrôle de la cohérence

Dans un environnement réparti, l'atomicité s'applique à la transaction globale. Les contraintes imposées par la répartition du contrôle ne permettent pas d'appliquer brutalement les solutions mises en oeuvre dans les systèmes centralisés. Des aménagements et souvent même de nouvelles techniques sont nécessaires. Il est curieux de constater que certaines de ces techniques, conçues pour un environnement réparti, ont été depuis lors appliquées avec succès dans des systèmes centralisés ; c'est le cas en particulier des algorithmes basés sur le verrouillage et la prévention des interblocages. Le but de ce paragraphe est de dégager les lignes directrices dans la conception d'un système de contrôle de cohérence en étudiant successivement le contrôle de l'accès concurrent et l'ensemble validation-reprise.

a) Accès concurrent

On distingue deux approches. La première vise à recréer les conditions d'un système centralisé en privilégiant un site. Elle permet alors de récupérer intégralement les mécanismes développés en centralisé et, de ce fait facilite l'évolution vers une configuration répartie. Si on ne dispose pas d'un réseau à diffusion ou au moins d'un réseau à grand débit, cette solution est peu performante car l'accès au site privilégié constitue un goulot d'étranglement. Cette solution pèche surtout au niveau de la sûreté de fonctionnement car la disponibilité du système réparti tout entier est conditionnée par la fiabilité du site privilégié.

L'alternative consiste, au contraire, à ne privilégier aucun site et à préserver l'autonomie des contrôleurs en limitant les synchronisations à l'essentiel. Les techniques correspondantes sont plus complexes mais autorisent plus de souplesse en ce qui concerne le fonctionnement en mode dégradé.

b) Validation et reprise

Les actions de manipulation des objets d'une transaction (Lire-D, Ecrire-D, Sortir-D, ...) n'ont de signification que sur le site où elles sont exécutées. Il est donc difficile d'essayer de se ramener aux conditions d'un système centralisé. La seule approche possible consiste à synchroniser les étapes de validation, d'invalidation et de reprise des agents de la transaction.

Cette brève introduction met en évidence l'importance de la synchronisation par messages dans la réalisation d'un mécanisme global de contrôle de cohérence. Nous verrons, tout au long de ce chapitre, que le choix d'une technique est un compromis entre la sûreté de fonctionnement obtenue à l'aide de synchronisations sophistiquées, et l'efficacité qui, au contraire, passe par l'autonomie des contrôleurs.

1.2.3. Bases de données dupliquées

On ne saurait terminer une présentation du contrôle de la cohérence dans un système réparti sans évoquer les bases de données dupliquées.

Il y a deux manières d'envisager la répartition des données dans un système d'information distribué. Dans le premier cas, il existe un exemplaire unique de chaque objet : on parlera de base de données partitionnée. Les critères de répartition des objets, propres à chaque application, ne sont pas traités ici. A l'opposé, il peut être avantageux de conserver sur des sites distincts plusieurs copies du même objet : on parlera alors de base de données dupliquée (on dit aussi base à copies multiples). La duplication peut être partielle ou totale (dans ce dernier cas, il existe un exemplaire de chaque objet sur tous les sites.) Les arguments avancés en faveur des bases dupliquées sont les suivants :

- l'accès en consultation est accéléré puisque l'information est disponible localement,

- la disponibilité du système global est accrue grâce à la redondance créée par la duplication. Une panne ne provoque pas l'inaccessibilité à une partie de l'information comme c'est le cas dans un système partitionné.

Les bases de données dupliquées posent un nouveau problème : celui de la mise en cohérence des copies. Cette question a déjà été évoquée dans le chapitre 1 en présentant les notions de cohérence forte et de cohérence faible selon que la mise à jour des copies est atomique ou qu'on accepte un décalage entre deux versions d'un même objet. Dans les systèmes de bases de données dupliquées, la mise à jour des copies est prise en charge par le système. Toute modification d'un objet est répercutée automatiquement sur les autres copies. On notera que cette méthode est bien adaptée aux Systèmes de Gestion de Bases de Données Réparties dans lesquels la localisation des données est transparente au programmeur d'application.

La mise en cohérence des copies multiples d'un objet constitue un cas particulier du contrôle de cohérence et peut être résolue par les algorithmes généraux. Cependant, de nombreux auteurs se sont intéressés à ce sujet et ont proposé des algorithmes conçus pour traiter ce type de redondance. On trouvera dans [Wilms 79] une analyse qualitative et dans [Garcia 79] une évaluation quantitative de certains algorithmes.

En ce qui nous concerne, les bases de données dupliquées sont considérées comme un cas particulier des bases partitionnées. En conséquence, le système ne prend pas en charge la mise à jour automatique des copies et aucun algorithme adopté à cette configuration particulière ne sera étudié.

1.3. Protocoles - Hypothèses sur le fonctionnement du réseau

Les paragraphes précédents ont mis en évidence la nécessité d'une coopération entre les systèmes transactionnels pour contrôler l'exécution répartie et garantir l'atomicité. Cette coopération est régie par des protocoles du niveau "application" au sens de la normalisation OSI, dont le principe est rappelé dans la figure 3.2.

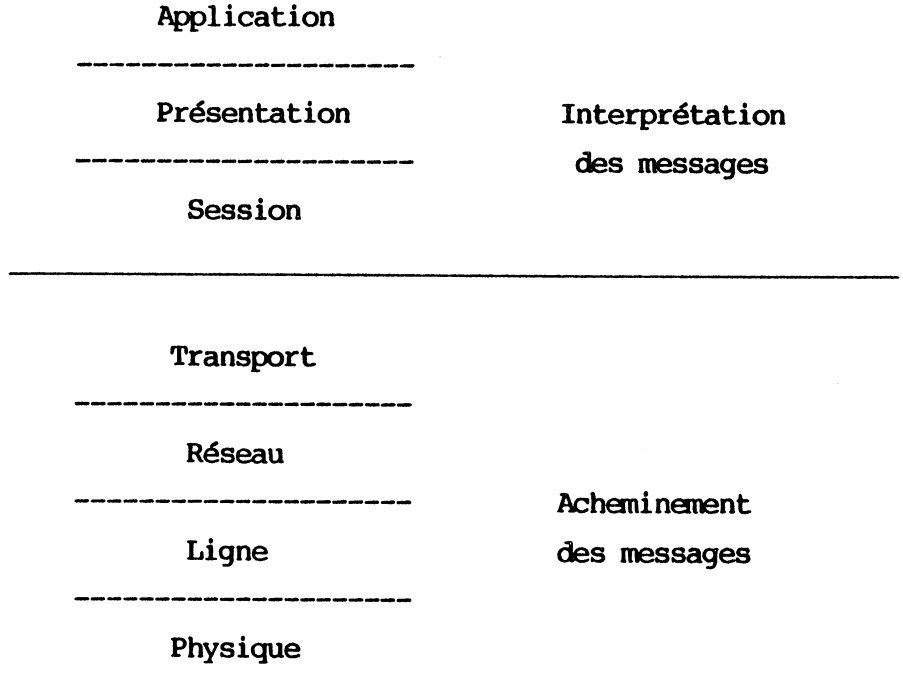


Figure 3.2

Architecture en couche OSI

Les quatre niveaux inférieurs, que nous désignerons globalement dans la suite sous le terme "protocoles de communication", contrôlent l'acheminement de l'information de "bout en bout" entre deux "stations de transport" qui assurent l'interface avec les niveaux supérieurs. Les services offerts par une station de transport sont les suivants :

- Acheminement des messages sans erreur, c'est-à-dire sans perte, sans duplication et sans altération de l'information. Toute condition anormale qui ne pourrait être rectifiée à ce niveau est interprétée comme une défaillance du service de transport et est signalée à l'application.

- Délivrance des messages dans leur ordre d'émission.

- Détection des pannes d'un site ou du réseau. Une panne de réseau a pour origine une défaillance d'un maillon quelconque de la chaîne de communication : ligne, noeud du réseau, coupleur, frontal, etc. Dans ce cas, les systèmes communiquants s'efforcent de trouver un autre chemin d'accès. Si cette tentative est infructueuse, la panne est permanente. Il y a partitionnement du réseau : un ou plusieurs sites se trouvent isolés de leurs partenaires. En résumé, une panne traduit simplement l'incapacité de communiquer avec un partenaire. Cette indication est transmise à l'application.

- Détection de la reconnexion d'un site qui avait momentanément disparu (la station de transport utilise pour cela une procédure de surveillance qui teste périodiquement la présence sur la ligne du partenaire disparu).

La nature de ces services est indépendante du type de réseau de communication (réseau longue distance de type TRANSPAC ou réseau local de type ETHERNET). Seul le délai d'acheminement des messages est fortement conditionné par ce paramètre.

Cette architecture s'applique exclusivement au cas des liaisons point-à-point. On ne trouve pas aujourd'hui de service équivalent dans le cas des liaisons multi-points (un émetteur et plusieurs récepteurs). Ce problème fait l'objet de recherches intensives sur divers types de réseaux à diffusion (ETHERNET, TOKEN RING, TOKEN BUS). La disponibilité d'une telle fonctionnalité aurait un impact majeur sur la défini-

tion des protocoles de contrôle de la cohérence [Scot 81, Khider 83]. Dans la suite de ce travail, nous supposons donc que les liaisons sont exclusivement de type point-à-point et que les communications multi-points sont simulées à l'aide de plusieurs liaisons point-à-point. Lorsque cela sera nécessaire, nous évoquerons l'impact d'un réseau à diffusion fiable.

La synchronisation par message, base des protocoles de contrôle de la cohérence, repose sur l'existence de liaisons point-à-point entre les processus coopérants. Nous avons déjà signalé au paragraphe 1.3.1 que les contraintes temporelles dues au réseau introduisent parfois des zones d'incertitude dans la réalisation de certaines opérations de synchronisation. Pour illustrer ce problème, nous considérons l'opération suivante entre deux processus P1 et P2 s'exécutant sur deux sites distincts A et B.

P1 exécute l'action A1 si et seulement si P2 a exécuté l'action A2 (figure 3.3).

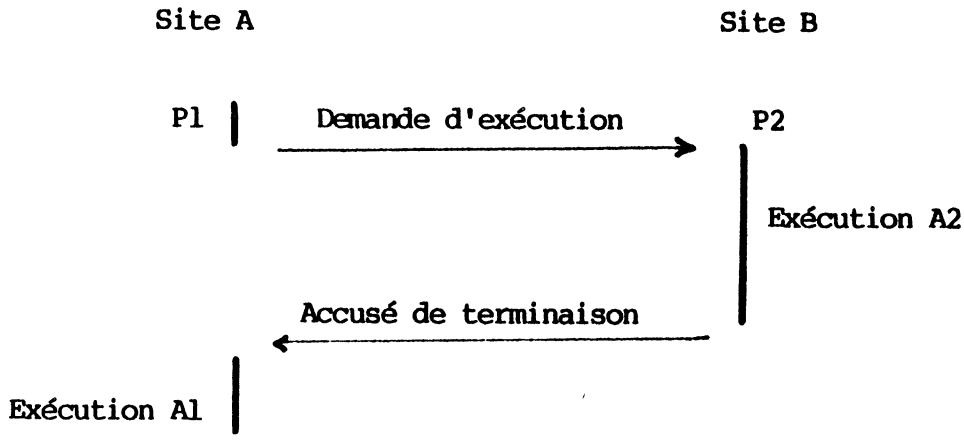


Figure 3.3

Un exemple de synchronisation

L'absence de message de terminaison en provenance de P2 interdit à P1 toute hypothèse quant à la réalisation de l'action A2 :

- la demande a pu ne pas arriver : A2 non exécutée,
- la demande est arrivée mais A2 n'a pas été exécutée (panne du site B),
- A2 a été exécutée mais l'acquittement a été perdu.

Seul le rétablissement du dialogue entre les deux processus permettra de lever l'ambiguïté. On parle alors de protocole bloquant.

Ces dernières années, un effort particulier a porté sur la réalisation, au niveau de la couche "Session", de mécanismes destinés à fournir aux applications un service de communication point-à-point sûr. En dépit de l'avantage certain que constitue la factorisation de ce service pour toutes les applications, cette approche n'offre qu'une solution partielle aux problèmes évoqués plus haut pour les raisons suivantes :

- un protocole d'accord entre deux entités de la couche session est non significatif du point de vue de l'application. En effet, la délivrance du message à la couche application n'est pas garantie (panne du site B pendant le transfert du message entre session et application),
- le protocole de niveau session est lui-même bloquant et on est ramené au cas précédent.

Seul, l'envoi d'un accusé de réception au niveau de l'application est significatif. Les échanges intermédiaires ont pour seul objectif de raccourcir les zones d'incertitude et d'accéler la procédure de reprise après panne. C'est pourquoi l'hypothèse d'un réseau fiable ne sera jamais évoquée dans la suite de ce travail.

Dans un système d'information réparti, on distingue généralement cinq protocoles :

- Protocole d'exécution répartie : lancement d'un agent à distance et dialogue entre agents.

- Protocole de contrôle de l'accès concurrent : contrôle de l'accès simultané par des transactions à des données partagées réparties sur plusieurs sites.

- Protocole de validation : garantit l'atomicité des actions d'une transaction lorsqu'elle se termine normalement.

- Protocole d'abandon : garantit l'atomicité lorsque la transaction est abandonnée à cause d'une erreur fatale.

- Protocole de reprise après panne : complément indispensable des deux protocoles précédents, il permet de synchroniser les actions des systèmes transactionnels après une panne d'un site ou du réseau.

Les protocoles de validation, d'abandon et de reprise sont décrits en détail dans la section 2. Les liens avec le contrôle de l'accès concurrent sont traités dans la section 3.

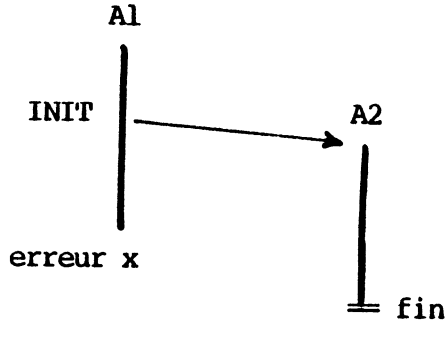
2. VALIDATION - REPRISE DANS UN SYSTEME DISTRIBUE

Cette section est consacrée aux techniques qui permettent d'assurer l'indivisibilité des actions exécutées par des agents coopérants. Dans un premier temps, on montre que l'application à chaque agent de la procédure de validation définie pour les systèmes centralisés ne suffit pas à garantir l'atomicité de la transaction : un protocole est nécessaire. Le principe de ce protocole est ensuite exposé tandis que la description détaillée des choix de réalisation est différée jusqu'au chapitre 4.

2.1. Nécessité d'un protocole : implications

L'application directe à chaque agent d'une transaction de la procédure de validation définie au chapitre 2 ne permet pas d'assurer en toutes circonstances l'atomicité de la transaction.

Exemple : Une transaction est constituée de 2 agents A1 et A2 s'exécutant en parallèle et validant séparément.



Une erreur dans le code de A1 a les conséquences suivantes :

A2 a validé ses actions
A1 a invalidé ses actions
A1 a invalidé ses actions

Figure 3.4

Une synchronisation des procédures de validation est nécessaire. Elle requiert la coopération des systèmes transactionnels locaux. Cette coopération s'établit sous forme de messages échangés entre les systèmes transactionnels et sera désignée dans la suite sous le terme de procédure de validation globale.

La difficulté principale de réalisation d'une procédure de validation globale est liée principalement à l'impossibilité de définir comme en centralisé un point de validation qui soit visible de tous les agents. Ceci est la conséquence directe de l'absence d'état global du système dans un environnement réparti.

Au point de validation se substitue, pour un agent, la notion de "zone de validation" telle que :

- avant entrée dans la zone, un incident entraîne l'abandon de l'agent,

- après sortie de la zone, un incident n'affecte plus l'agent (ses actions sont validées),

- dans la zone, l'occurrence d'un incident peut placer un agent dans un état indécidable. L'avis des partenaires est nécessaire pour sortir de cette situation. Si les partenaires sont inaccessibles (cas du site isolé), le rétablissement du dialogue est nécessaire pour décider du sort de l'agent. Cette situation est l'illustration de ce que nous avons appelé protocole "bloquant" dans le chapitre précédent. La conséquence majeure de l'attente qui en découle est l'indisponibilité des objets permanents accédés par l'agent.

Il est important de noter qu'il n'existe pas de protocole de validation résistant à un partitionnement du réseau. La démonstration formelle en a été faite dans [Skeen 81-a].

La multiplication des chemins d'accès entre plusieurs sites permet de limiter les risques de partitionnement du réseau. Le coût de cette forme de redondance s'avérant prohibitif, on préfère travailler à la définition de mécanismes qui s'efforcent de limiter la durée de la "zone de validation" afin de minimiser le risque évoqué plus haut. Certains éliminent la zone de risque pour au moins un agent de la transaction (cet agent joue alors un rôle privilégié dans la procédure de validation globale). Les techniques utilisées pour limiter la durée des zones de validation et donc pour parvenir à des procédures de validation plus "sûres" sont coûteuses car elles exigent des synchronisations plus sophistiquées, donc des échanges de messages supplémentaires. Un des objectifs de notre travail a consisté précisément à estimer le bien-fondé de ces mécanismes "chers" en fonction des besoins des applications.

2.2. Principe de la validation en environnement distribué

Nous décrivons deux méthodes pour contrôler le processus de synchronisation des étapes de validation.

2.2.1. Validation à contrôle distribué

Cette méthode est basée sur l'observation suivante : un agent peut valider si, et seulement si, les autres agents de la même transaction ont terminé leur exécution.

Chaque agent informe ses partenaires de sa terminaison (message 'Fin'). Un agent qui a reçu les messages de terminaison de tous ses partenaires valide localement (figure 3.5-a).

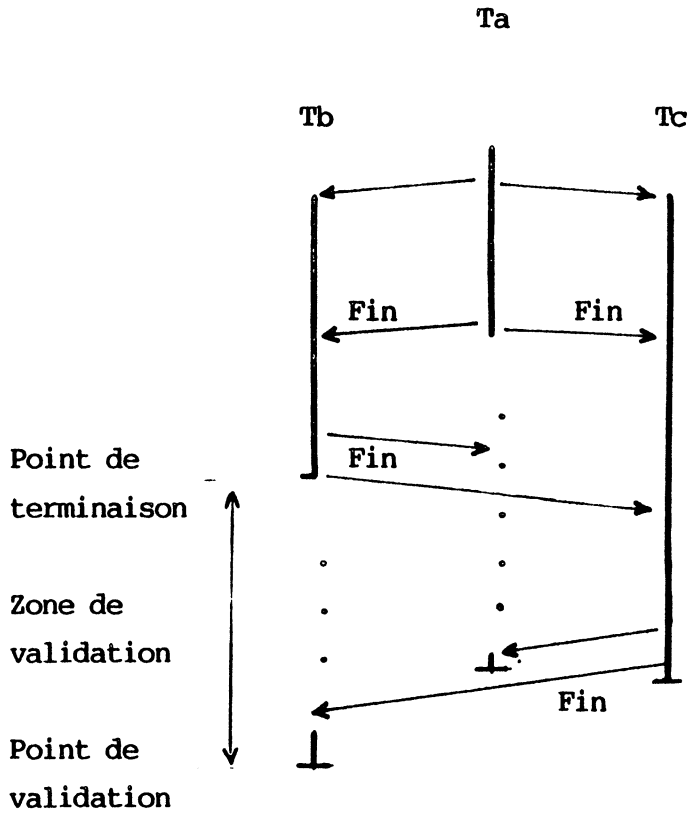


Figure 3.5-a

Ce protocole a les caractéristiques suivantes :

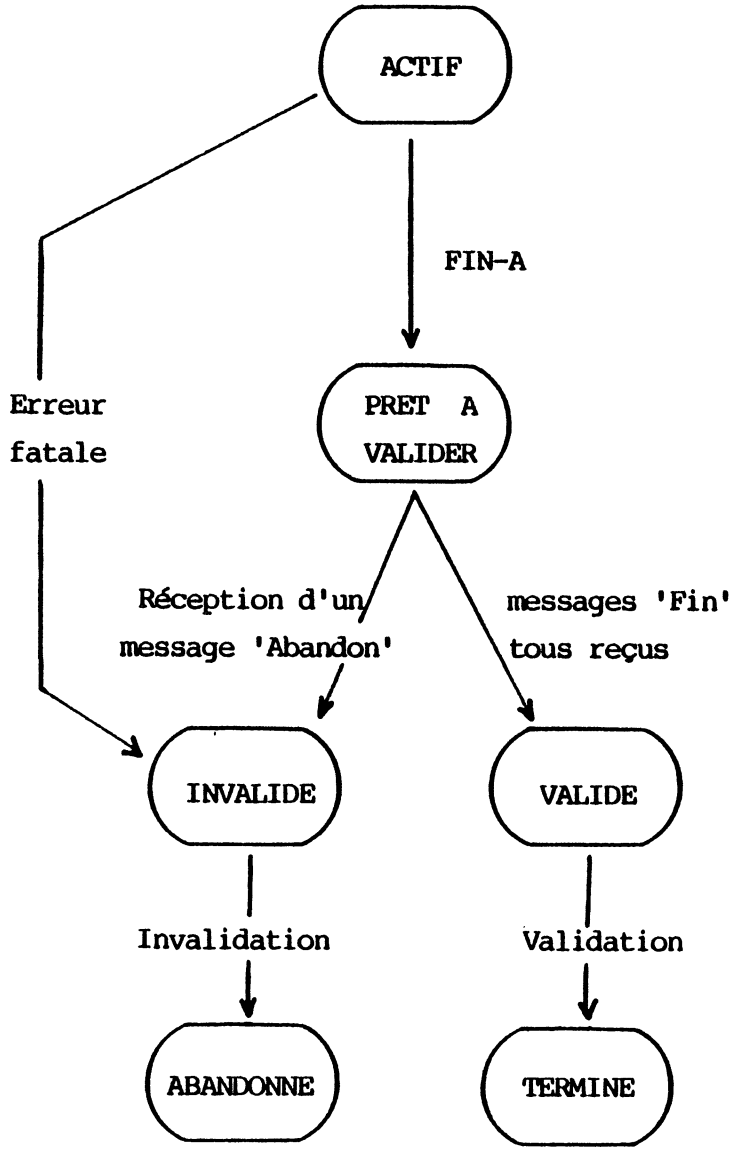
a) (N - 1) messages sont théoriquement nécessaires pour la validation globale si N est le nombre d'agents.

b) Il n'y a pas d'agent privilégié, chacun d'eux jouant un rôle équivalent dans le processus de validation.

c) Pour chaque agent, la zone de validation s'étend du point de terminaison jusqu'au point de validation. La panne d'un site distant ou du réseau pendant cette période ne permet pas au système transactionnel local de décider d'une issue pour un agent qu'il contrôle.

d) Cette méthode suppose en outre que chaque agent connait l'existence de tous ses partenaires. Cette connaissance mutuelle est plus ou moins complexe selon les scénarios.

Le diagramme d'état d'un agent est décrit dans la figure 3.5-b.



(Ce diagramme d'état est incomplet : le cas des pannes et des erreurs récupérables sera évoqué plus loin).

Figure 3.5-b

2.2.2. Validation à contrôle centralisé

Les deux inconvénients majeurs de la technique précédente sont, d'une part le nombre important de messages, et d'autre part l'existence d'une "zone de validation" pour tous les agents.

La validation à contrôle centralisé apporte une solution à ces deux problèmes. Cette technique est basée sur l'existence d'un agent privilégié appelé "**supérieur**" ou **coordonateur** de la validation. Par opposition, les autres agents sont appelés "**inférieurs**".

Le point de validation de la transaction est assimilé au point de validation de l'agent supérieur. Le principe du protocole est le suivant (Figure 3.6-a) :

1. Les inférieurs envoient un message de terminaison ('Fin') au supérieur.

2. Quand le supérieur a reçu tous les messages de terminaison, il valide localement et diffuse un message de validation ('Valider') aux inférieurs.

3. Un agent inférieur qui reçoit ce message valide ses actions.

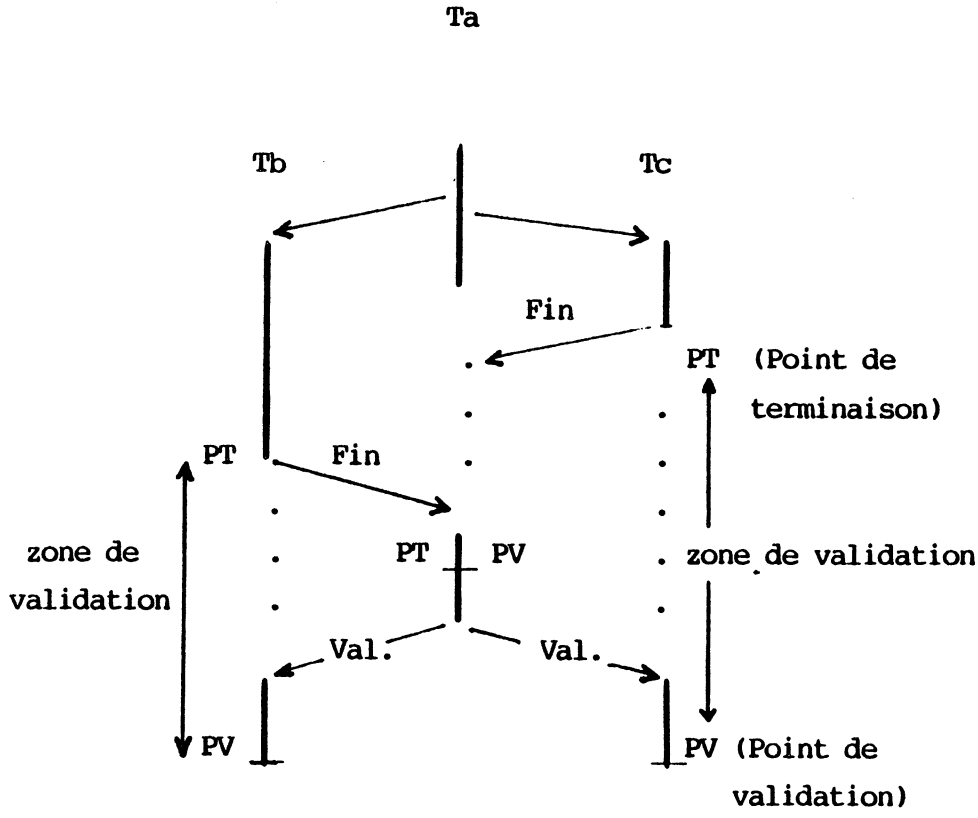


Figure 3.6-a

Dans la littérature, la description de ce protocole mentionne souvent l'envoi d'accusé de réception par les inférieurs qui ont validé. En fait, ces messages ne sont pas nécessaires à la correction du protocole ; on trouvera dans [Baer 80], une démonstration de cette affirmation. Le rôle des accusés de réception sera examiné dans le chapitre suivant.

Les caractéristiques de ce protocole sont les suivantes :

a) 2 (N-1) messages sont nécessaires pour la validation globale. A partir de 3 sites, le coût est donc inférieur à celui du protocole à contrôle distribué.

b) Le protocole suppose que les inférieurs connaissent le supérieur (envoi du message 'Fin') et que le supérieur connait tous les inférieurs (diffusion du message 'Valider'). Nous verrons plus loin que la présentation mutuelle peut être plus ou moins complexe en fonction de l'application et de l'identité du supérieur.

Le diagramme d'état d'un agent est décrit dans la figure 3.6-b.

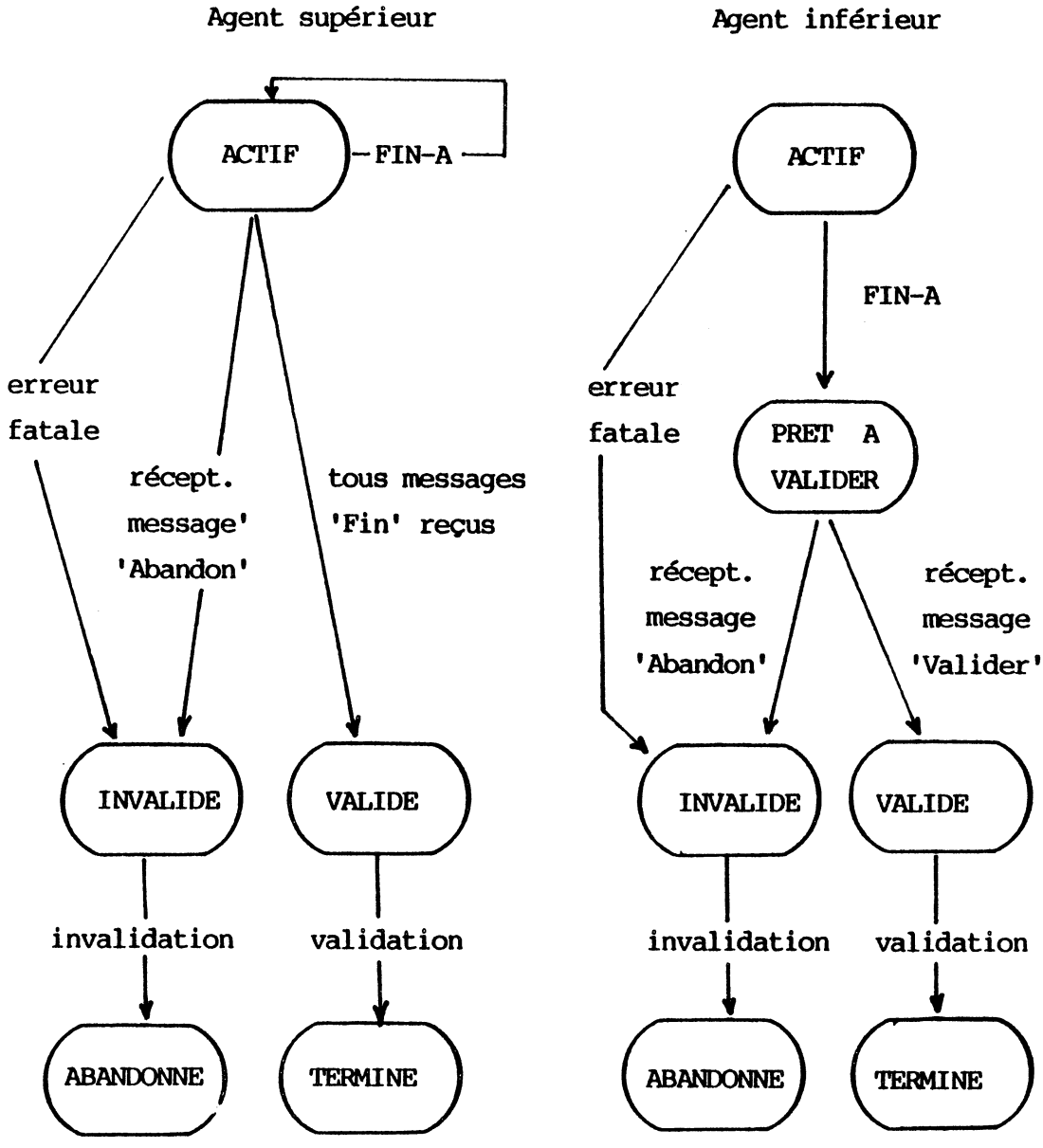


Figure 3.6-b

L'exécution d'un agent inférieur comporte trois étapes : Calcul, Préparation à la validation et Validation, caractérisées par les trois états ACTIF, PRET A VALIDER et VALIDE.

L'état de chaque agent est conservé sur un journal géré par le système transactionnel local. Les transitions d'état sont matérialisées par l'écriture d'un enregistrement dans ce journal.

Conclusion : la technique de contrôle centralisé présente deux avantages déterminants :

- Nombre de messages réduit (cet avantage devient caduque si on dispose d'un réseau à diffusion).

- Disparition de la zone de validation pour au moins un agent.

C'est pourquoi nous adopterons cette technique dans la suite de notre exposé.

2.2.3. Traitement des erreurs

Dans le chapitre 2, nous avons introduit une classification des incidents en trois catégories :

- les erreurs fatales, qui provoquent l'arrêt définitif de la transaction,

- les erreurs récupérables (dues essentiellement au contrôle de l'accès concurrent) qui provoquent l'abandon provisoire et la reprise de la transaction,

- les pannes qui arrêtent temporairement la transaction (jusqu'à la remise en état du maillon défaillant), la transaction est ensuite réexécutée. Nous traitons dans ce paragraphe le cas des erreurs, les pannes font l'objet du paragraphe 2.2.4.

La prise en compte d'une erreur s'effectue en plusieurs étapes :

. traitement sur le site où l'erreur a été détectée et diffusion d'un message d'erreur aux agents coopérants. Ce message précise la nature de l'erreur,

. chaque contrôleur qui reçoit un message d'erreur applique localement le traitement correspondant.

a) Erreurs fatales

Une erreur fatale correspond à une anomalie (ou à l'exécution de l'action ARRET-A) dans le code d'un agent. En conséquence, l'occurrence d'une erreur fatale n'est possible que dans l'étape de calcul (état "ACTIF") d'un agent. A la différence de la décision de valider qui appartient exclusivement au Supérieur, la décision d'abandonner une transaction peut être prise par un agent quelconque. Le traitement consiste à abandonner l'agent et à diffuser un message (noté 'Abandon') aux partenaires. Un agent qui reçoit un message d'abandon est invalidé.

b) Erreurs récupérables

Une erreur récupérable est générée par le contrôle de l'accès concurrent ; elle affecte l'étape de calcul. Le traitement consiste à interrompre l'exécution de la transaction et à la réexécuter après que le conflit sur la ressource ait été résolu. Selon la structure de la transaction, la reprise peut être partielle ou totale. Ce point sera détaillé dans le chapitre suivant. Pour la simplicité de l'exposé, nous supposons ici que la reprise est totale ; tous les agents autres que l'agent initial sont abandonnés et l'agent initial est repris. En résumé, le traitement d'une erreur récupérable est le suivant :

- le contrôleur qui détecte l'erreur diffuse un message d'erreur récupérable (noté "Reprise") aux partenaires,

- un agent, autre que l'agent initial, qui reçoit ce message exécute la procédure d'invalidation : le traitement équivaut à une erreur fatale,

- l'agent initial exécute la procédure de reprise.

Nous verrons dans la section consacrée au contrôle de l'accès concurrent qu'il existe des algorithmes qui autorisent la préemption d'un agent pendant l'étape de préparation à la validation. Ces algorithmes requièrent un traitement spécial dont nous différons la présentation jusqu'au chapitre 4.

La remarque faite à propos des accusés de réception dans le protocole de validation est valable ici : l'acquittement des messages d'erreur n'est pas indispensable.

2.2.4. Traitement des pannes

Une panne de site ou du réseau se traduit par l'incapacité de communiquer avec un système distant. La prise en compte d'une panne s'effectue en deux temps :

a) Chaque système s'efforce de trouver une issue aux agents locaux qui coopéraient avec des agents du site momentanément disparu :

   - un agent qui exécute l'étape de validation ou d'invalidation n'est pas affecté,

   - pendant l'étape de calcul un signal de panne est équivalent à un message d'abandon : l'agent est abandonné.

- le système ne peut pas décider seul pour un agent qui se trouve dans l'étape de préparation à la validation. L'avis des partenaires est nécessaire (voir § 2.2.5).

b) Un système qui redémarre après une panne prend, pour les agents qui coopéraient avec l'extérieur, une décision compatible avec celle prise en son absence. Les critères de décision sont ceux décrits en a.

2.2.5. La "zone grise"

Les paragraphes précédents montrent qu'il existe une différence fondamentale entre les processus de validation et d'abandon d'une transaction :

- La décision de valider appartient exclusivement au Supérieur.

- La décision d'abandonner peut être prise par n'importe quel agent, sans en référer préalablement au Supérieur, sauf dans le cas où l'agent est dans la zone de validation. Cette restriction ne concerne pas l'agent supérieur pour lequel la zone de validation n'existe pas.

Un agent inférieur qui a passé le point de terminaison ne peut être ni validé, ni abandonné unilatéralement par le système local. Il ne peut être abandonné que sur requête d'un partenaire et validé uniquement sur requête du supérieur. En l'absence de message du supérieur (panne du supérieur ou du réseau), un agent inférieur est suspendu. Il n'a, en effet aucun moyen de savoir si le supérieur a passé ou non le point de validation.

La propriété d'isolation impose que les objets manipulés par l'agent inférieur demeurent inaccessibles aux transactions concurrentes jusqu'au rétablissement de la connexion qui, seule, permettra au système isolé de prendre une décision en conformité avec celle de ses partenaires. Ce blocage a des conséquences graves sur le fonctionnement du système puisque des transactions, qui ne sont pas directement concernées par la panne, peuvent être indéfiniment retardées.

Nous appelons "zone grise" (*) la période correspondante qui s'étend du point de terminaison au point de validation. L'état "PRET A VALIDER" est un autre moyen de caractériser la zone grise. Un agent supérieur n'a pas de zone grise.

Dans les systèmes de bases de données réparties, il existe sur le site initial deux agents de la transaction, l'agent initial qui contrôle l'exécution des agents et assume le rôle de supérieur et un agent inférieur qui manipule des données sur ce site. Cette structuration ne modifie en aucune façon les résultats présentés dans cet ouvrage. Pour ne pas alourdir les schémas, nous assimilerons dans la suite le supérieur à un agent de la transaction manipulant des données.

* La terminologie est très riche pour désigner cette période sensible d'un agent inférieur : citons entre autres, "zone de doute", "zone d'incertitude" ou "zone de validation" comme nous l'avions désignée auparavant.

2.3. Description du protocole de base

Ce paragraphe décrit les procédures, exécutées localement par chaque système, et les messages, échangés entre les systèmes coopérants, qui constituent le protocole de **validation-reprise**. La description qui suit se limite à mentionner les actions essentielles de chaque procédure du protocole. Le lecteur trouvera en complément dans l'annexe D, un diagramme d'état complet pour chaque type d'agent.

a) Procédure de terminaison (ou pré-validation)

Cette procédure est activée localement sur occurrence de la primitive FIN-A qui marque la fin de l'étape de calcul d'un agent.

Supérieur	Inférieurs
- Attendre les messages de terminaison en provenance des inférieurs.	- Journaliser les ressources acquises (la justification de cette opération sera donnée plus loin) - Journaliser état PRET A VALIDER (entrée en zone grise). - Envoyer message 'Fin' au supérieur et attendre une réponse.
- Lorsque tous les messages sont arrivés, exécuter la procédure de validation.	- Exécuter la procédure de validation.

b) Procédure de validation

Cette procédure est exécutée par le supérieur lorsqu'il décide de valider la transaction et par un inférieur lorsqu'il reçoit le message de validation.

Supérieur	Inférieurs
- Journaliser état VALIDE	- Journaliser état VALIDE
- Envoyer message 'Valider' aux inférieurs	
- Consolider les actions locales	- Consolider les actions locales
- Libérer les objets locaux	- Libérer les objets locaux
- Message à l'utilisateur si agent initial	- Message à l'utilisateur si agent initial
- Journaliser état TERMINE	- Journaliser état TERMINE

c) Procédure d'invalidation (ou d'abandon)

Cette procédure est activée dans l'un des cas suivants :

- Erreur ou action ARRET-A dans le code d'un agent.
- Disparition d'un partenaire pendant l'étape de calcul.
- Réception d'un message d'abandon d'un partenaire.
- Réception d'un message de reprise (par un agent autre que l'agent initial).

Supérieur	Inférieurs
- Journaliser état INVALIDE	- Journaliser état INVALIDE
- Envoyer message 'Abandon' aux inférieurs (*)	- Envoyer message 'Abandon' aux partenaires (*)
- Annuler les actions locales	- Annuler les actions locales
- Libérer les objets locaux	- Libérer les objets locaux
- Message d'erreur à l'utilisateur si agent initial	- Message d'erreur à l'utilisateur si agent initial
- Journaliser état ABANDONNE	- Journaliser état ABANDONNE

* L'émission du message d'abandon correspond uniquement au cas où une erreur fatale est détectée localement.

On observe que la procédure est la même, que l'agent soit supérieur ou inférieur.

d) Procédure de reprise d'une transaction

La reprise d'une transaction est consécutive à une décision du contrôle de l'accès concurrent ou à un redémarrage du système après une panne. La reprise d'une transaction signifie :

. l'abandon des agents autres que l'agent initial,
. la réexécution de l'agent initial.

Dans la description qui suit, on suppose que la reprise est totale. Le problème de la reprise partielle sera examiné dans la section 3.

Agent Initial	Autres agents
	- Journaliser état INVALIDE
- Annulation actions locales	- Annulation actions locales
- Libérer les objets locaux	- Libérer les objets locaux
- Réexécution	- Journaliser état ABSENT

e) Procédure de redémarrage d'un système transactionnel

Le système utilise le journal local pour exécuter la procédure propre à chaque agent.

Si état = TERMINE ou ABANDONNE Alors pas d'action.

Si état = ACTIF Alors
Si agent initial Alors procédure de REPRISE
Sinon procédure d'INVALIDATION.

Si état = VALIDE Alors procédure de VALIDATION

Si état = INVALIDE Alors procédure d'INVALIDATION.

Si état = PRET A VALIDER Alors ENQUETE.
Le système local ne pouvant décider seul, l'avis d'un partenaire est nécessaire. La procédure ENQUETE est explicitée plus loin (cf. 3.3.3).

RESUME :

Les propriétés essentielles des protocoles de validation d'abandon et de reprise sont rappelées ci-dessous :

- Un agent joue une rôle privilégié dans le contrôle de la transaction : on l'appelle le Supérieur.

- Seul le supérieur décide la validation. Il transmet sa décision aux autres agents.

- L'abandon d'une transaction peut être initialisé par n'importe quel agent sans en référer préalablement au supérieur. Un message indiquant la cause de l'erreur est transmis aux autres agents.

- Les accusés de réception aux messages de validation et d'abandon sont théoriquement superflus.

- Pour tous les agents autres que le supérieur, il existe une période critique, appelée zone grise, pendant laquelle une rupture du contact avec le supérieur a des effets bloquants.

- En l'absence de redondance explicite, il n'existe pas de protocole de validation non-bloquant.

2.4. Eléments de choix pour la réalisation d'un protocole de Validation - Reprise

Les paragraphes précédents ont décrit les caractéristiques fonctionnelles d'un protocole de validation-reprise. Il existe plusieurs techniques de réalisation d'un tel protocole. Nous listons ci-dessous les points essentiels pour lesquels des choix de réalisation sont possibles. La description détaillée des solutions correspondantes fait l'objet du chapitre 4.

a) Choix du Supérieur

Il est assez naturel d'assimiler le supérieur à l'agent initial. Certaines applications se satisfont de ce choix, d'autres requièrent un choix différent.

b) Contrôle du processus de validation

Nous avons éliminé la validation à contrôle réparti en particulier à cause du nombre de messages. Un protocole de validation globale à contrôle centralisé est basé sur l'existence d'un dialogue entre le supérieur et les inférieurs. Ce dialogue peut s'établir de deux façons :

1. En utilisant l'arbre d'invocation. On parlera de validation à contrôle hiérarchisé.

2. Directement entre le supérieur et les inférieurs. On parlera de validation à contrôle direct.

L'identité du supérieur et la nature du réseau ont une influence considérable dans le choix d'une de ces deux méthodes.

c) Procédure de validation

La procédure de validation présentée comporte un seul échange de messages entre le supérieur et les inférieurs. Il existe d'autres protocoles, plus sûrs, mais qui exigent des échanges supplémentaires pour limiter les zones grises. Nous analyserons les propriétés et les performances comparées de ces algorithmes selon les scénarios d'application.

d) Procédure de redémarrage d'un système après panne

La procédure de redémarrage après panne s'efforce de trouver une issue à chaque agent dont l'exécution a été interrompue par la panne. Le système local ne peut pas décider seul pour les agents qui étaient dans l'état PRÊT À VALIDER. Deux approches sont possibles :

. Le supérieur prend l'initiative de renvoyer à l'inférieur le message de validation ou d'abandon.

. L'inférieur s'enquiert auprès du supérieur du sort de la transaction globale (Procédure d'enquête).

3. VALIDATION ET CONTROLE DE L'ACCES CONCURRENT

3.1. Caractéristiques du contrôle de l'accès concurrent en environnement réparti

On rappelle que le rôle de l'accès concurrent consiste à ordonnancer les actions des transactions de telle sorte que le résultat soit équivalent à celui d'une exécution séquentielle (propriété de sérialisation). Dans le cas où une transaction est constituée de plusieurs agents s'exécutant sur des sites distincts, la cohérence forte impose que l'ordre de sérialisation des agents soit identique sur tous les sites. Pour certaines méthodes, cette contrainte se traduit par la nécessité de synchroniser les contrôleurs. Les règles de synchronisation et les messages correspondants constituent le protocole de contrôle de l'accès concurrent.

La classification en deux catégories (contrôle continu et contrôle par certification) est reprise pour présenter le principe de fonctionnement des algorithmes. Comme dans le chapitre précédent, nous nous limitons à décrire le principe de fonctionnement de chaque technique et les corrélations avec les mécanismes de validation et reprise(l'annexe A donne pour chaque classe les algorithmes de base). Pour une étude plus complète, nous renvoyons le lecteur aux références citées dans le texte.

Les observations faites au chapitre précédent concernant les critères d'évaluation qualitatifs (risque de famine, granularité variable et généralité) restent valables dans un environnement distribué. Cependant, de nouveaux critères de jugement apparaissent, qui sont dûs en grande partie au besoin de synchroniser les contrôleurs.

. Complexité :

La synchronisation introduit un facteur de complexité supplémentaire dont le coût additionnel est fonction du degré de sophistication des échanges. De ce point de vue, nous considérons l'autonomie des contrôleurs comme une qualité essentielle.

. Robustesse :

Ce critère définit la sensibilité d'un mécanisme aux défaillances d'un site ou du réseau. De ce point de vue, l'autonomie des contrôleurs est également un facteur fondamental, dans la mesure où la défaillance de l'un d'entre eux n'a pas de répercussion sur le reste du système.

Nous considérons cependant que la performance du système demeure le critère de jugement décisif. L'impact d'une synchronisation par messages sur la performance du système est double. D'une part, l'attente inhérente à cette forme de synchronisation accroît le temps de rétention des ressources réduisant ainsi le degré de parallélisme. D'autre part, le nombre de messages échangés sur le réseau a une incidence directe sur le temps de réponse de la transaction. C'est pourquoi, certains auteurs, considérant sans doute qu'il s'agit là du paramètre clé, mesurent l'efficacité d'un algorithme uniquement en fonction du nombre de messages échangés sur le réseau [Garcia 79, Wilms 79]. Nous pensons que cette méthode est trop simpliste et qu'il convient de prendre en compte, comme en centralisé, le coût des attentes et des reprises pour obtenir une mesure plus réaliste. Dans le projet SCOT, de telles études ont été réalisées pour le cas des techniques de contrôle continu fondées sur le verrouillage. Les résultats les plus significatifs sont présentés dans le chapitre suivant et sont comparés à ceux d'autres études parues dans la littérature.

3.2. Contrôle continu

Cette dénomination recouvre deux classes de méthodes : le verrouillage à deux phases et l'estampillage des transactions.

3.2.1. Verrouillage à 2 phases

On examine successivement le principe de fonctionnement en environnement réparti, le problème des interblocages et les liens avec les procédures de validation-reprise.

a) Principe de fonctionnement

Chaque contrôleur maintient un graphe des attentes résultant des conflits détectés localement. L'existence de ces graphes permet de prendre en compte les interblocages locaux ; elle ne suffit pas à résoudre le problème des interblocages globaux, comme le montre l'exemple de la figure 3.7.

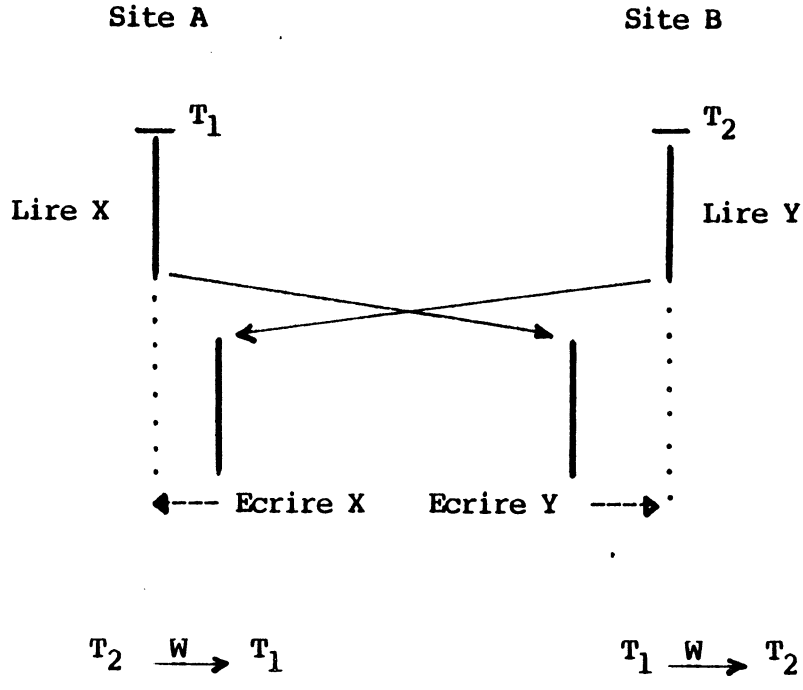


Figure 3.7

Interblocage global

Le graphe des attentes de chaque site indique une simple attente. Seule, la réunion des deux sous-graphes permettrait de reconstituer le cycle révélateur de l'interblocage. Cette situation est la conséquence de l'absence d'état global dans un système réparti.

b) Gestion des interblocages globaux

Nous avons vu au chapitre 2 qu'il y a trois manières de tenir compte des interblocages : les ignorer, les détecter et les guérir ou les éviter. Nous analysons brièvement l'application des techniques correspondantes à un environnement réparti.

. Ignorer les interblocages : méthode du temps limite.

On attribue à chaque transaction un temps maximum pour s'exécuter. Le dépassement de cette limite est interprété par le système comme une potentialité d'interblocage. Dans ce cas la transaction est reprise. On peut associer le temps limite à chaque agent ou à la transaction globale. Cette technique est facile à mettre en oeuvre et ne demande aucun développement supplémentaire pour s'adapter à un environnement distribué. C'est sans doute la raison pour laquelle les premiers systèmes transactionnels répartis commercialisés ont adopté cette technique. En revanche, son comportement est très sensible à la valeur du temps limite [Kohler 81], [SCOT 83]. Les modélisations montrent, d'une part que la valeur optimale du temps limite est celle qui correspond au temps de réponse effectif, et d'autre part que la plage de manoeuvre autour de cet optimum est faible. Le choix d'une valeur optimale,déjà délicat dans un système centralisé, l'est plus encore dans un système réparti à cause des fluctuations entraînées par le réseau sur la valeur du temps de réponse effectif d'une transaction. Cette technique est aujourd'hui pratiquement abandonnée.

. Détection et guérison des interblocages

La détection des interblocages globaux requiert la reconstitution du graphe global des attentes pour pouvoir tester la présence éventuelle d'un cycle. Deux techniques s'opposent selon qu'on choisit de reconstituer le graphe sur un site privilégié (contrôle centralisé) ou que les contrôleurs échangent périodiquement des informations en vue de reconstituer le graphe global (contrôle réparti).

Le contrôle centralisé permet l'application directe de toutes les techniques développées pour les systèmes centralisés. Comme pour la méthode du temps limite, cela garantit une évolution aisée des applications vers des configurations réparties. Par contre, cette méthode présente deux inconvénients majeurs :

- Chaque conflit engendre un message vers le contrôleur centralisé avec, pour conséquence, un nombre de messages prohibitif et un goulot d'étranglement au niveau du site privilégié.

- La défaillance du site privilégié paralyse le système global. Cette dépendance va à l'encontre de l'objectif de disponibilité.

Des aménagements ont été proposés pour diminuer le nombre de messages (en regroupant en un seul envoi l'information associée à plusieurs conflits) et pour accroître la disponibilité du système (en cas de panne, un contrôleur de secours est élu pour prendre le relai du contrôleur disparu). Le système D-INGRES [Stonebraker 79], qui utilisait cette technique à ses débuts l'a depuis lors abandonné en raison d'une certaine inefficacité.

Le contrôle réparti élimine en partie ces problèmes en augmentant l'autonomie des contrôleurs et en limitant les échanges de messages. Dans [Goldman 77], la détection repose sur la reconstitution de la chaîne d'interblocage par circulation d'un message contenant la liste des transactions interbloquées. De nombreuses autres méthodes ont été proposées depuis, basées sur l'échange de sous-graphes par les contrôleurs [Gouda 78, Menasce 79, Gligor 80, Obermarck 82]. Par rapport au contrôle centralisé, l'ensemble de ces techniques gagne en robustesse car la défaillance d'un site n'a de conséquence que pour les transactions qui coopéraient avec lui. Ces dernières années, les efforts des concepteurs ont porté essentiellement sur la réduction du nombre de messages et du délai nécessaire à la détection. Deux paramètres entrent en jeu dans ce processus d'optimisation : le nombre de sites auxquels on envoie un sous-graphe et la périodicité d'émission des messages. Le lecteur trouvera dans [Wazdi 83] une analyse très détaillée des propriétés de ces algorithmes.

Remarque :

A notre connaissance, la correction de ces algoritmes n'a pas été démontrée pour les schémas d'exécution parallèle.

. Prévention des interblocages

Pour mémoire, nous rappelons l'existence de la méthode de prévention statique basée sur l'annonce des ressources utilisées par la transaction. Cette méthode est facilement transposable à un système réparti [Lamet 78]. Nous l'écartons cependant pour les mêmes raisons que celles qui ont été évoquées au chapitre précédent et, dans la suite, le terme "prévention" fera exclusivement référence à la prévention dynamique.

Les techniques de prévention sont fondées sur l'existence de la même relation d'ordre sur tous les sites, générée à l'aide d'un estampillage universel [Rahimi 82]. Cette relation d'ordre introduit une priorité implicite entre les transactions qui est utilisée à chaque conflit pour faire attendre ou abandonner une transaction. Cette méthode est très facilement transposable à un système réparti. Il suffit en effet de donner la même estampille à tous les agents d'une transaction et d'appliquer la même relation d'ordre sur chaque site. Les techniques diffèrent par la nature de la relation d'ordre qui est appliquée [Rosenkrantz 78, Decitre 81]. Plusieurs algorithmes de cette catégorie sont décrits en annexe A.

Par rapport à la détection, la prévention accroît l'autonomie des contrôleurs puisqu'il suffit que chacun d'eux applique systématiquement la même relation d'ordre pour assurer la propriété de sérialisation. En revanche, le nombre de transactions abandonnées et réexécutées est plus important. L'impact de ce point sur les performances est étudié plus en détail au chapitre 4.

c) Liens avec les procédures de validation reprise

Nous ne reviendrons pas ici sur les techniques de propagation du signal d'erreur récupérable (message 'Reprise') qui permet d'abandonner tous les agents d'une transaction. Ce problème a été largement débattu dans la section précédente. Les remarques qui suivent sont indépendantes de la méthode choisie pour le traitement des interblocages globaux (détection ou prévention).

Par rapport à un système centralisé, l'élément nouveau est l'existence de la zone grise pour les agents inférieurs (état PRET A VALIDER). Un contrôleur ne peut pas décider unilatéralement la préemption d'un agent local en zone grise pour résoudre un cas réel ou potentiel d'interblocage. Il est possible, en effet, que dans le

même laps de temps, le supérieur ait déjà validé. La solution consiste à informer le supérieur d'une demande d'abandon et attendre une réponse à cette requête (figure 3.8). Pendant ce temps, la transaction à l'origine du conflit est provisoirement mise en attente. On constate ainsi que, contrairement aux systèmes centralisés, une opération de préemption n'est pas toujours immédiate; elle peut être précédée d'une attente. Les algorithmes exempts de préemption ne requièrent pas cette forme de synchronisation. Leurs décisions sont prises indépendamment les uns des autres, respectant ainsi l'objectif d'autonomie.

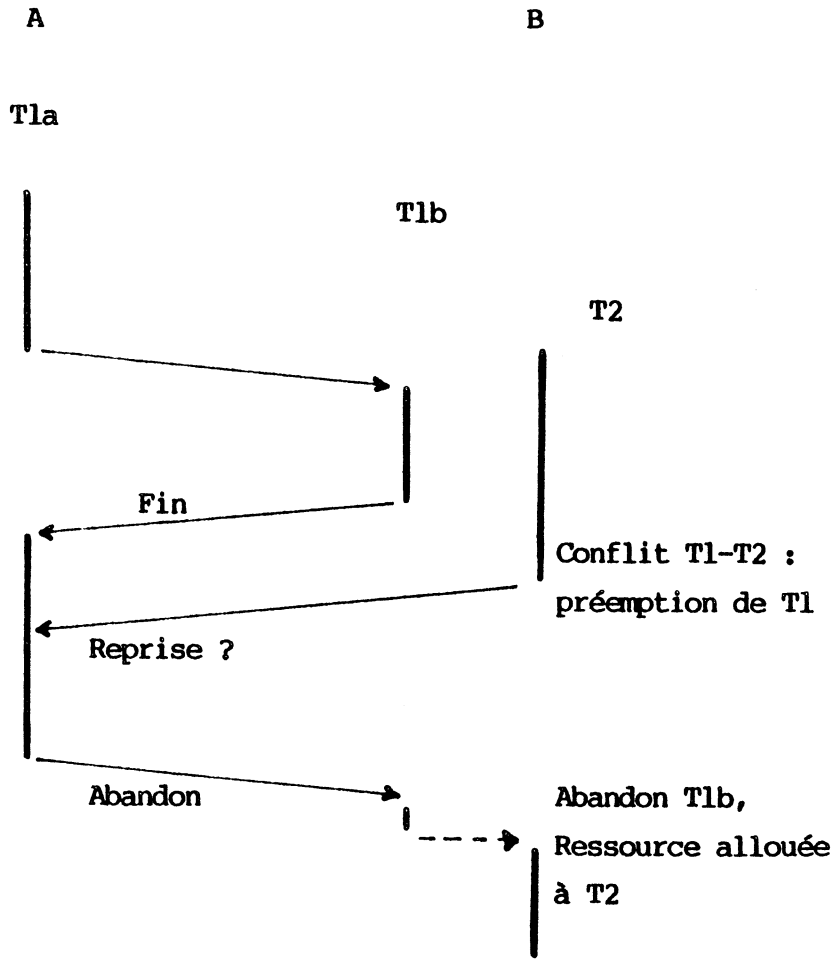


Figure 3.8

Préemption d'un agent en zone grise

3.2.2. Estampillage des transactions

a) Principe de fonctionnement

Ces méthodes utilisent une relation d'ordre préétablie entre les transactions à l'aide de l'estampillage universel. En cas de conflit, les transactions sont forcées de s'exécuter dans l'ordre imposé par les estampilles (cf. chapitre 2, §3.2.2). On distingue l'estampillage statique lorsque l'estampille est affectée à l'initialisation de la transaction et l'estampillage dynamique lorsque l'affectation de l'estampille est différée jusqu'à l'apparition du premier conflit. La propriété de sérialisation reposant sur l'application de la même relation d'ordre sur tous les sites, les agents d'une transaction doivent posséder la même estampille. Cette contrainte est facile à réaliser dans l'estampillage statique en passant l'estampille dans le message d'initialisation d'un agent. La technique d'estampillage dynamique appliquée à un scénario d'exécution parallèle pose un double problème, d'une part agréer une valeur d'estampille entre les agents (si plusieurs d'entre eux sont impliqués dans un conflit sur des sites différents), et d'autre part diffuser la valeur d'estampille ainsi élue aux autres partenaires. Le coût de cette négociation est cher en l'absence d'un réseau à diffusion. Le test de validité des actions d'un agent a lieu pendant l'étape de calcul. Un test négatif entraîne le rejet de l'agent et par conséquent de la transaction (protocole d'erreur récupérable). Pour le supérieur, le message de terminaison d'un agent représente donc une preuve de la validité des actions de cet agent et le protocole de validation globale constitue un moyen naturel de synchroniser les tests de conformité des agents.

b) Liens avec les procédures de validation reprise

Lorsque le test de conformité en écriture d'un agent inférieur est accepté, le contrôleur local doit garantir que la valeur de l'objet correspondant n'évolue plus jusqu'à la réception de l'ordre de validation. Cette contrainte se traduit par la nécessité d'exécuter les primitives Ecrire-base de deux agents concurrents dans l'ordre des estampilles. Cet ordonnancement peut amener un contrôleur à retarder éventuellement l'étape de validation d'un agent (figure 3.9). Ces attentes ne sont pas génératrices d'interblocages car elles sont ordonnées de la même manière sur tous les sites.

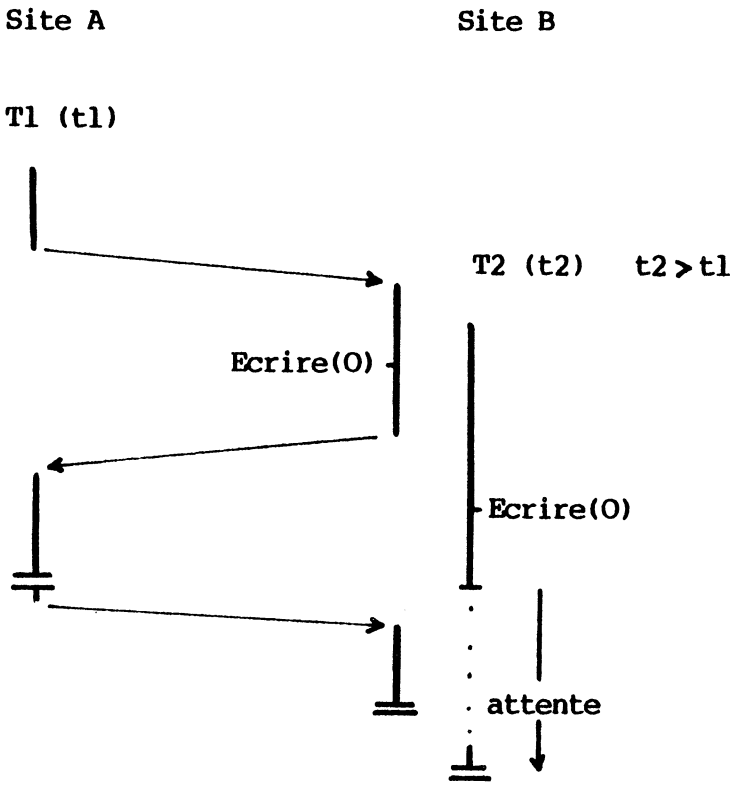


Figure 3.9

Estampillage et validation

3.3. Contrôle par certification

a) Principe de fonctionnement

Un agent est structuré en trois étapes : calcul, certification et validation (cf. chapitre 2, § 3.3). L'étape de calcul ne comporte aucun contrôle de dépendance. L'ensemble des contrôles est reporté en fin d'exécution d'un agent et constitue l'étape de certification. On rappelle qu'il existe plusieurs stratégies de certification selon qu'on teste les dépendances avec les transactions déjà certifiées ou que l'on prend aussi en compte les transactions en cours d'exécution. Pour les différentes méthodes de résolution en environnement réparti, nous renvoyons le lecteur à [Schlageter 81, Bhargava 82, Ceri 82, Viemont 82, Boksenbaum 85]. Les trois premières propositions citées sont des adaptations de la méthode centralisée décrite dans [Kung 81]. Les problèmes posés par les certifications parallèles, dans le but d'accroître le parallèlisme, sont évoquées dans [Schlageter 82, Boksenbaum 85].

b) Lien avec les procédures de validation-reprise

La définition d'un ordre de sérialisation cohérent sur tous les sites impose de :

- synchroniser les étapes de certification des agents. En outre, les méthodes qui utilisent un estampillage pour la certification requièrent une négociation entre les partenaires pour déterminer une estampille de référence.

- garantir que la valeur des objets pour lesquels les actions ont été certifiées, n'évolue plus jusqu'à la validation.La solution qui consiste à interdire tout accès à ces objets pendant la phase de validation est inacceptable dans un environnement distribué. Pour cette raison, les méthodes préconisées par les études mentionnées plus haut utilisent des protocoles complexes. Afin de minimiser l'impact sur les performances, leurs auteurs recommandent l'usage d'un réseau à diffusion.

3.4. Comparaison des techniques de contrôle

La répartition ne modifie en rien les conclusions du chapitre précédent en ce qui concerne le risque de famine et la granularité multiple. Outre les aspects présentés au chapitre précédent, le critère "généralité" s'applique également aux types de scénario d'exécution supportés par un mécanisme donné. De ce point de vue, on rappelle que le verrouillage avec prévention des interblocages et l'estampillage s'appliquent sans restriction. Dans le cas d'une transaction comportant des exécutions parallèles, la correction des algorithmes n'a pas été démontrée formellement pour le verrouillage avec détection et la certification.

En ce qui concerne la robustesse, nous avons d'emblée éliminé la méthode qui combine le verrouillage à deux phases et la gestion centralisée des interblocages. Aucune autre technique ne fait jouer un rôle privilégié à un site particulier et la robustesse est celle de l'étape de validation. Celà signifie que toutes ces techniques sont sensibles à l'existence d'une zone grise chez les agents inférieurs. On rappelle que, pour faire face à ce problème, l'identité des objets et la nature des actions correspondantes sont sauvegardées à l'entrée de la zone grise. Lors d'une reprise après panne, cette information est utilisée pour rétablir les dépendances qui existaient à l'instant où la panne s'est produite.

Le tableau suivant résume les observations faites dans les paragraphes précédents sur la complexité des protocoles. Le critère majeur retenu dans cette analyse concerne la nature des synchronisations entre les contrôleurs. la difficulté de mise en oeuvre d'un algorithme nous paraît être un critère trop subjectif pour pouvoir être mesuré.

Méthode	Estampillage universel	Nature des synchronisations
VERROUILLAGE		
Temps limite	non (1)	aucune
Détection	non	Echange de sous-graphes
Prévention :		
sans préemption	oui	aucune
avec préemption	oui	Enquête auprès du supérieur
ESTAMPILLAGE		
Statique	oui	aucune
Dynamique	oui	Diffusion de l'estampille après le premier conflit
CERTIFICATION	oui (2)	Négociation sur l'issue des étapes de certification

(1) L'estampillage peut être rendu nécessaire par la prévention du risque de famine.

(2) [Ceri 82] n'utilise pas d'estampille.

On rappelle qu'un dispositif d'estampillage universel est basé sur l'échange de messages pour garantir la cohérence des estampilles générées sur les différents sites.

Du tableau précédent, il ressort que la méthode du temps limite est la seule à ne solliciter aucune forme de synchronisation. Le verrouillage avec prévention (sans préemption) et l'estampillage statique requièrent simplement la génération d'estampilles universelles. Les autres méthodes imposent des synchronisations plus complexes, sources d'attente.

En environnement distribué, les informations quantitatives sur le comportement respectif des techniques de contrôle de l'accès concurrent sont peu nombreuses. L'interprétation des résultats correspondants est plus délicate encore qu'en centralisé en raison du nombre de paramètres supplémentaires introduits par le réseau.

Parmi les techniques présentées dans cette section, seul le verrouillage à deux phases a été étudié de façon approfondie. Paradoxalement, la détection des interblocages, qui a été analysée de manière approfondie en centralisé, n'a pas suscité un intérêt comparable en environnement réparti. Dans [Obermarck 82], on trouve, après la présentation de l'algorithme, la description d'un modèle simple de système distribué à partir duquel l'auteur calcule le nombre moyen de messages nécessaires à la détection d'un interblocage; aucune information n'est fournie quant au temps de réponse ou au débit du système. [Wazdi 83] compare plusieurs méthodes de détection, à la fois sur le plan qualificatif (correction, complexité) et quantitatif. Sur ce dernier point, les critères d'évaluation sont le délai et le nombre de messages pour réaliser la détection d'un interblocage. L'algorithme d'Obermarck et celui proposé par l'auteur sont de ce point de vue les meilleurs choix.

En comparaison les algorithmes de prévention ont donné lieu à des études plus nombreuses parmi lesquelles celles qui ont été réalisées dans le projet SCOT. Les résultats correspondants seront présentés et commentés dans le chapitre suivant. A notre connaissance, il n'existe pas de publications relatives à l'évaluation comparative de la prévention et de la détection des interblocages dans un système réparti.

Dans divers articles [Badal 80, Badal 81], l'auteur compare plusieurs techniques issues du verrouillage à deux phases, de l'estampillage et de la certification en utilisant des modèles probabilistes. Le critère de performance retenu est le degré de parallélisme autorisé par chaque méthode. De ce point de vue, la certification arrive en tête devant l'estampillage et le verrouillage. Ce résultat, conforme à la logique, ne préjuge en rien de l'efficacité des mécanismes en terme de temps de réponse (les attentes et les reprises ne sont pas prises en compte dans ce modèle).

[Lin 83] compare l'estampillage et le verrouillage à deux phases. L'auteur conclut sur la supériorité du verrouillage lorsque le taux de conflit est fort ou lorsque les transactions sont longues. l'estampillage est meilleur pour les faibles taux de conflit et pour les transactions très courtes (1 ou 2 accès). D'une manière générale, ces résultats confirment les observations faites à propos des sytèmes centralisés.

Chapitre

IV

SCOT

Un Noyau Transactionnel pour le traitement d'Informations distribuées

SOMMAIRE DU CHAPITRE IV

3. CONTROLE DE L'ACCES CONCURRENT : EVALUATION

3.1 Choix du verrouillage : justification

3.2 Evaluation des techniques de verrouillage : le modèle

3.2.1 Représentation du système : hypothèses de modélisation

3.2.2 Paramètres du modèle

3.2.3 Quantités mesurées

3.3 Gestion des interblocages : détection ou prévention ?

3.3.1 Détection des interblocages

3.3.2 Prévention des interblocages

3.3.3 Conclusion : détection ou prévention

3.4 Quel algorithme de prévention en environnement réparti ?

3.5 Synthèse : éléments d'un noyau transactionnel pour les systèmes d'information répartis

4. SYSTEME D'INFORMATION A HAUTE DISPONIBILITE

4.1 Redondance et disponibilité

4.2 Système en service continu

0. INTRODUCTION

Les arguments développés dans les chapitres précédents ont permis d'illustrer le postulat selon lequel :

- la cohérence et la performance sont deux objectifs contradictoires,

- les besoins des applications dans ces deux domaines sont très différents.

Dans l'optique d'un système d'information à vocation universelle, aucune technique de contrôle ne peut prétendre s'imposer. C'est pourquoi nous rejetons la solution habituelle qui consiste à réaliser un système de contrôle fermé, fondé sur une technique déterminée et transparent aux applications, pour nous orienter vers la définition d'un noyau sur lequel il soit possible de bâtir différentes stratégies de contrôle de la cohérence en fonction des paramètres de l'application (scénarios d'exécution, niveau de cohérence, temps de réponse, disponibilité, ...). Le rôle du programmeur d'application dans le choix d'une stratégie est fondamental. Le noyau est défini avec l'objectif de permettre la cohabitation de plusieurs mécanismes. Le choix du mécanisme le plus approprié étant laissé au programmeur d'application ; Le système fournit pour cela une interface de programmation qui permet l'expression de ces choix.

Ce chapitre présente les caractéristiques de ce noyau sous le triple aspect de l'architecture, des techniques de validation-reprise et du contrôle de l'accès concurrent.

La section 1 revient brièvement sur l'architecture d'un système d'information réparti. Les propriétés respectives d'un système de gestion de bases de données réparties et d'un système transactionnel coopérant sont comparées en termes de fonctions et de perspectives industrielles. Il ressort de cette analyse que ces deux approches ne sont pas antagonistes et que le transactionnel coopérant constitue en fait une étape intermédiaire vers les bases de données réparties.

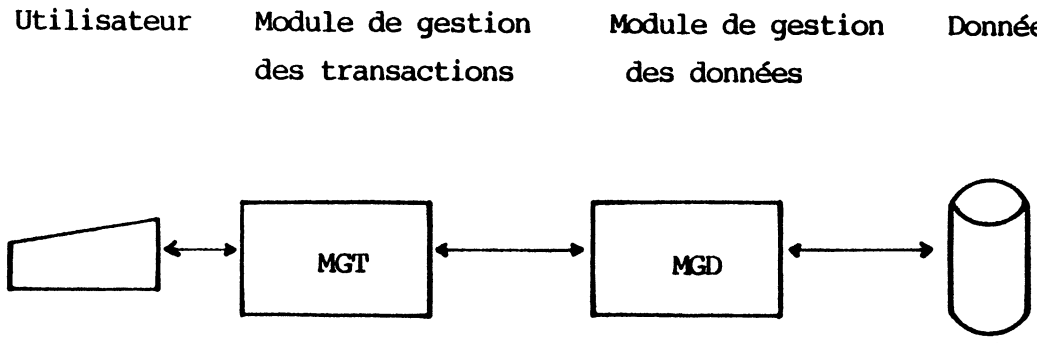
Les techniques de validation et reprise sont traitées dans la section 2. Pour chacune des composantes répertoriées à la fin de la section correspondante du chapitre précédent, plusieurs choix de réalisation sont décrits et comparés sur le triple aspect de la correction, de la performance et de l'adéquation aux applications. Cette analyse débouche sur la définition d'un mécanisme hybride ayant les propriétés requises pour être intégré dans le noyau.

La section 3 est consacrée au contôle de l'accès concurrent et plus particulièrement aux mécanismes fondés sur le verrouillage. Les résultats des différentes campagnes d'évaluation sont présentés et justifient le choix d'une famille d'algorithmes pour le noyau. Une synthèse des résultats principaux des sections 2 et 3 permet d'établir les éléments de base du noyau transactionnel recherché.

La section 4 enfin, décrit un exemple d'utilisation d'un noyau de système transactionnel coopérant pour la réalisation d'un système d'information à haute disponibilité.

1. ARCHITECTURE D'UN SYSTEME D'INFORMATION DISTRIBUE

Dès l'introduction de cet ouvrage, nous avons évoqué l'existence de deux approches pour la conception d'un système d'information réparti. Pour comparer leurs caractéristiques respectives, nous nous appuierons sur le modèle d'architecture présenté au chapitre 2 et rappelé ci-dessous.



- Le Module de Gestion des Données (MGD) gère l'accès aux données (contrôle du partage et détection des conflits d'accès).

- Le Module de Gestion des Transactions (MGT) contrôle l'exécution cohérente des transactions et assure l'interface avec l'usager.

Figure 4.1

architecture d'un système d'information
avec des capacités transactionnelles

On peut envisager la répartition au niveau de l'un ou l'autre de ces modules.

a) Répartition au niveau de la description des données :

La localisation et les contraintes d'intégrité s'expriment de manière statique à l'aide du langage de description des données.

Ce type de répartition, illustré dans la figure 4.2-a, permet d'offrir au concepteur d'application et à l'utilisateur des langages de description et de manipulation qui le font apparaître comme un système centralisé. C'est l'approche "Systèmes de Gestion de Bases de Données Réparties" (SGBD-R), dans laquelle la distribution des données est transparente au programmeur d'application.

La répartition peut être réalisée soit au niveau du schéma conceptuel (SDD-1, SIRIUS-DELTA, SABRE, POLYPHEME) [Sddl 79, LeBihan 79, Gardarin 82, Polyphème 79], soit au niveau du schéma interne (R*, D-INGRES) [Lindsay 79, Neuhold 76]. Dans le premier cas, le schéma est de type relationnel et l'unité de répartition est la relation. Dans le second cas, la distribution porte sur l'implantation des données et correspond grossièrement à une allocation de fichiers.

Un programme d'application est constitué d'une ou plusieurs requêtes émises sur un site. Une requête est analysée et décomposée automatiquement par le MGD local en fonction des informations de localisation. Le résultat de cette décomposition est un ensemble de sous-requêtes envoyées sur les sites correspondants pour y être exécutées (coopération entre les MGD).

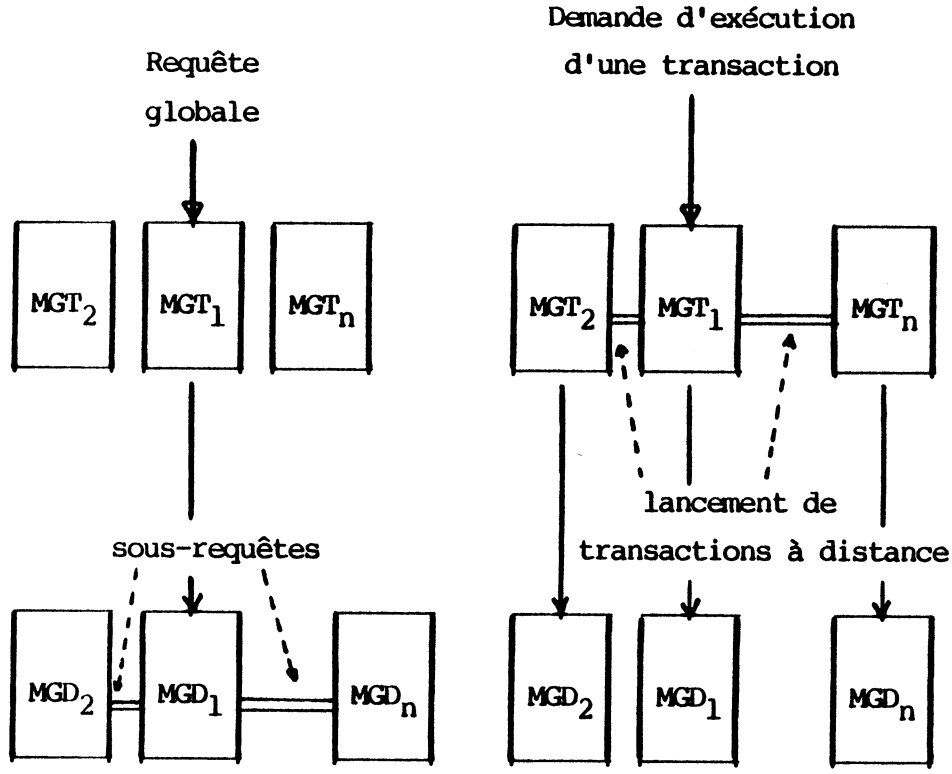
b) Répartition au niveau des programmes d'application :

Dans cette approche, les MGD sont indépendants et n'opèrent que sur des données qui leur sont locales. A l'inverse de l'approche précédente, la localisation des données et les contraintes d'intégrité sont exprimées directement dans les programmes d'application; les

schémas de description des données ne comportent aucune information de localisation.

Un programme d'application est constitué de une ou plusieurs transactions, s'exécutant chacune sous le contrôle d'un MGT particulier, et susceptibles de dialoguer entre elles (coopération des MGT).

L'architecture correspondant à cette approche, qui est illustrée par la figure 4.2-b, offre une solution simple à l'hétérogénéité des modèles de données puisque chacun d'eux n'est vu qu'au travers des transactions définies localement (absence de vue globale).



a. Distribution au niveau de la description des données (approche SGBD-R)

b. Distribution des programmes d'application (approche transactionnel coopérant)

Figure 4.2

Les deux types de distribution et les architectures correspondantes

Conclusion : un choix d'architecture

Le projet POLYPHEME est un exemple de réalisation d'un Système de Gestion de Base de Données Réparti avec schéma global, localisation automatique et langage de requêtes relationnel permettant la coopération de SGBD hétérogènes. Cette recherche a permis de déceler les limites actuelles de cette approche, dues essentiellement à la complexité et au coût du processus de décomposition automatique.

Par rapport aux Systèmes de Gestion de Bases de Données Réparties, l'approche "Transactionnel Coopérant" laisse un certain nombre de problèmes à la charge du concepteur d'application : la localisation des données, le découpage en transactions locales (absence de décomposition automatique) et l'hétérogénéité des modèles de données (absence de vue globale). En contrepartie, le système gagne en simplicité, donc en fiabilité et en performance.

En définitive, un argument déteminant a poussé à la fois les constructeurs et les utilisateurs à s'engager résolument dans cette voie. En effet, le transactionnel coopérant apparaît comme une solution de continuité pour les applications car il permet la récupération d'une grande partie des programmes existants. L'effort de transition vers une application répartie est sans commune mesure avec celui qui serait nécessaire pour reconcevoir complètement une application à partir d'un nouveau modèle de données. On trouve aujourd'hui plusieurs exemples de réalisations opérationnelles qui découlent de cette approche. Citons entre autres la coopération de systèmes CICS et IMS chez IBM et la coopération de systèmes TDS chez CII-HB. L'approche "Base de Données Distribuées", encore mal maîtrisée au niveau de la description des données (vue globale) et de la décomposition automatique des requêtes, fait actuellement l'objet de recherches complémentaires.

Les deux approches diffèrent essentiellement par la richesse du modèle de données qu'elles offrent au concepteur d'application. Il n'en reste pas moins qu'elles présentent de nombreuses analogies, notamment en ce qui concerne l'exécution répartie et le contrôle de la cohérence basé sur le concept de transaction. De ce point de vue, le transactionnel coopérant doit être considéré comme une étape intermédiaire vers les bases de données réparties. Si on se réfère à une structuration en couches, les fonctionnalités du transactionnel coopérant constituent la base nécessaire à la réalisation d'un système de gestion de bases de données réparties. On trouve dans SIRIUS-DELTA [Le Bihan 79] une illustration d'une telle architecture. Dans ce système, les couches "SER" (exécution répartie) et "SCORE" (contrôle de la cohérence) correspondent exactement aux fonctions du transactionnel coopérant et sont utilisées par la couche supérieure "SILOE", en charge de la gestion des données réparties.

Dans la suite de ce chapitre, on s'intéresse aux aspects "systèmes" de la répartition de l'information, à l'exclusion de tous les problèmes relatifs aux modèles de données. On s'appuiera pour cela sur le modèle de système transactionnel coopérant présenté au chapitre précédent. L'annexe C illustre le caractère canonique de ce modèle en montrant comment il permet de décrire plusieurs systèmes de gestion de bases de données réparties.

2. REALISATION D'UN PROTOCOLE DE VALIDATION REPRISE

Dans le chapitre 3, en conclusion de la section présentant les protocoles de validation et reprise, nous avons listé plusieurs points pour lesquels des choix de réalisation sont possibles : choix du supérieur, contrôle du processus de validation, nature de la procédure de validation et de la procédure de reprise. Ces sujets ont fait l'objet d'une étude approfondie dans SCOT et les résultats correspondants sont présentés ici.

2.1. Le "supérieur" de la validation globale

2.1.1. Propriétés du supérieur : Rappel

Nous avons vu que le supérieur joue un rôle privilégié dans le contrôle du processus de validation. La décision de valider ou d'annuler les traitements effectués par la transaction appartient au supérieur. En conséquence, le supérieur est caractérisé par les propriétés suivantes :

P1 - Terminaison assurée sans délai long (un délai long est par exemple l'attente d'une reconnexion d'un site).

P2 - Disponibilité des ressources locales assurée. Cette propriété est une conséquence de P1.

L'impact de ces deux propriétés est discuté à propos de plusieurs politiques de choix du supérieur. Les conséquences de ces choix sur le contrôle du processus de validation (hiérarchisé ou direct) seront examinées en 3.2.

2.1.2. Le supérieur est l'agent initial

Cette option est le choix le plus naturel. C'est aussi le cas le plus répandu dans les systèmes opérationnels.

L'agent initial dialogue avec l'utilisateur au terminal et rend compte du résultat de l'exécution de la transaction. La propriété P1 assure qu'en toutes circonstances, l'utilisateur est averti de la terminaison (normale ou anormale) de la transaction. De ce point de vue, l'interface entre le système distribué et l'utilisateur n'est pas différente de l'interface habituelle dans un système centralisé.

Cet argument perd une grande partie de son intérêt si l'interface avec l'initiateur de la transaction est asynchrone. C'est le cas en particulier lorsque l'utilisateur n'attend pas la fin de la transaction (déconnexion après lancement et absence de dialogue) ou lorsque la transaction est initialisée par un autre programme. Dans ce cas, le message de terminaison ou d'abandon est déposé dans une boîte aux lettres lorsque l'agent initial est validé ou abandonné.

Nous avons vu que le principe des échanges de la validation globale repose sur la connaissance mutuelle du supérieur et des inférieurs. Lorsque le supérieur est l'agent initial, sa présentation aux inférieurs est réalisée de manière très simple en utilisant l'arbre d'invocation. Il suffit de passer l'identité de l'agent initial dans le message d'initialisation d'un nouvel agent.

Le dernier argument qui plaide en faveur de l'agent initial est le temps de réponse. Dans le processus de validation globale, le supérieur valide le premier puis diffuse ensuite le message de validation aux inférieurs. Dès que le point de validation du supérieur est passé, le message de terminaison est transmis à l'utilisateur qui est donc informé de l'issue de la transaction avant même que tous les agents ne soient validés.

2.1.3. Le supérieur n'est pas l'agent initial

Dans certains scénarios d'application, le choix de l'agent initial comme supérieur de la validation n'est pas nécessairement la solution la plus efficace. Nous en donnons deux exemples ci-dessous.

Exemple 1 :

Cette application met en jeu une base centrale gérée par un gros serveur et des bases locales gérées par des mini-ordinateurs satellites. Les usagers sont connectés aux ordinateurs satellites. Les transactions initialisées sur les satellites consultent et modifient la base centrale.

Si le supérieur est l'agent initial, une panne d'un satellite ou du réseau peut laisser sur le site central un agent en zone grise, donc en attente de reconnexion avec des ressources bloquées. Cette situation est incompatible avec la disponibilité de la base centrale qui est, dans cette application, un critère de satisfaction important.

La solution à ce problème de disponibilité des ressources consiste à considérer l'agent invoqué sur le site central comme le supérieur de la validation.

Exemple 2 :

Cet exemple est inspiré de scénarios d'exécution du système R* [Lindsay 79]. On considère une transaction constituée de plusieurs agents s'exécutant séquentiellement (migration). La décision de valider incombe à l'agent qui se trouve au bout de la chaîne d'exécution. Le message de validation parcourt le chemin inverse pour valider successivement tous les agents de la transaction. Le dernier agent invoqué est donc le supérieur de la validation. Cette structure de contrôle, désignée sous le terme "validation imbriquée" minimise le nombre de messages nécessaires à la validation pour ce schéma d'exécution (figure 4.3).

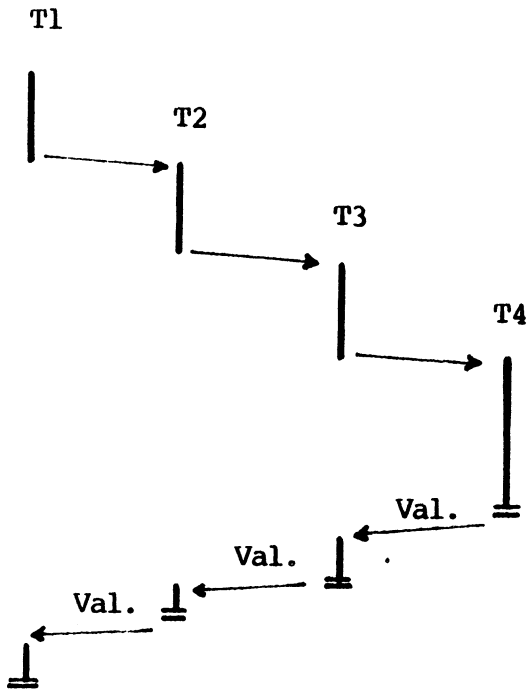


Figure 4.3

Validation imbriquée

Dans ces deux exemples, l'utilisateur est connecté à un agent qui n'est pas le supérieur de la validation. Une panne du supérieur ou du réseau lorsque l'agent initial est en zone grise ne permet pas à l'utilisateur de connaître l'issue de la transaction. Selon les applications, cela peut être un inconvénient majeur, en particulier parce qu'il n'a pas la possibilité d'enchaîner ultérieurement d'autres traitements dépendants du résultat de la transaction en cours.

Si on excepte le cas de la migration, la présentation du supérieur aux autres agents peut être complexe, en particulier si le scénario comporte plus de 2 agents. Ce point est développé en 2.2.

2.1.4. Définition du supérieur

La définition du supérieur est implicite lorsqu'il s'agit de l'agent initial. Dans le cas contraire, plusieurs solutions sont envisageables :

a) Définition statique

A la génération d'un système transactionnel, un agent est déclaré "supérieur" ou "inférieur". Chaque fois qu'un agent est initialisé pour le compte d'une transaction, le système vérifie la cohérence des définitions (un seul supérieur dans une transaction). Lorsque l'agent défini comme étant le supérieur est activé, il est nécessaire d'informer les partenaires de son identité. Selon les applications, cette présentation peut exiger des messages supplémentaires.

Cette solution présente un inconvénient majeur : un agent ne peut pas jouer plusieurs rôles (supérieur ou inférieur) selon la transaction dans laquelle il est impliqué.

b) Définition dynamique

L'identité du supérieur est déterminée par une négociation entre les agents. Cette négociation peut avoir lieu à l'initialisation ou à la terminaison (avant l'étape de validation) d'un agent.

Cette classe de solutions est plus souple que la précédente, mais plus difficile à mettre en oeuvre et aussi plus coûteuse en terme de messages échangés aux fins de présentation mutuelle.

Le choix de l'une ou l'autre de ces méthodes dépend fortement du type de contrôle adopté pour le processus de validation (contrôle hiérarchisé ou contrôle direct). C'est pourquoi nous différons cette discussion jusqu'au paragraphe 2.2.

c) Cas particulier des systèmes de gestion de bases de données répartis

Dans certains systèmes, la décomposition d'une requête, effectuée par l'agent initial, détermine la liste exhaustive des agents de la transaction. Par conséquent, le système a la possibilité de choisir parmi eux l'agent supérieur et de transmettre son identité à tous les autres agents dans les messages d'initialisation. Nous sommes ramenés à une situation équivalente à celle où le supérieur est l'agent initial. Il faut cependant noter, qu'à notre connaissance, aucun de ces systèmes n'exploite cette possibilité.

2.2. Contrôle du processus de validation

2.2.1. Présentation du problème

On s'intéresse ici aux flots d'informations échangés entre les agents pour exécuter l'étape de validation. Le principe de la validation globale, tel qu'il a été exposé précédemment repose sur deux types de dialogue.

1. Concentration des messages de terminaison vers le supérieur.

2. Diffusion de l'ordre de validation (ou d'invalidation) vers tous les inférieurs.

Il y a deux manières d'assurer la concentration ou la diffusion des messages entre les agents d'une transaction :

- Directement [Gray 78]

  Cela suppose la connaissance mutuelle du supérieur et des inférieurs. Nous verrons que cette condition est aisément réalisable lorsque le supérieur est l'agent initial. Elle l'est beaucoup moins dans le cas contraire.

- Par la voie hiérarchique [Colliat 79]

  On établit entre les agents une relation d'ordre partiel qui définit un arbre de validation dont la racine est le supérieur de la validation. Lorsque le supérieur est l'agent initial, l'arbre de validation est confondu avec l'arbre d'invocation. Dans le cas contraire, la définition de l'arbre de validation est plus complexe.

Ces deux types de dialogue sont référencés sous les termes :

- **Validation à contrôle direct** lorsque les liaisons s'établissent directement.

- **Validation à contrôle hiérarchisé** dans le second cas.

Nous analysons maintenant les avantages et les inconvénients de ces deux approches.

2.2.2. Validation à contrôle hiérarchisé

Nous examinons en premier lieu le cas simple où le supérieur de la validation est l'agent initial. L'arbre d'invocation est défini par les relations de filiation établies à l'initialisation de chaque agent. On utilise cette arborescence pour :

- faire remonter les messages de terminaison jusqu'à la racine,

- propager le message de validation de la racine à l'ensemble des agents constituant l'arborescence.

Le processus de validation globale s'initialise au niveau des agents qui constituent les feuilles de l'arbre. Chaque noeud attend les messages de terminaison de ses fils avant d'envoyer à son tour le message à son père. Les messages remontent ainsi l'arbre jusqu'à la racine. Le

supérieur valide et transmet le message de validation à ses fils. Chaque noeud qui reçoit le message valide l'agent local et transmet à son tour le message à ses propres fils (figure 4.4).

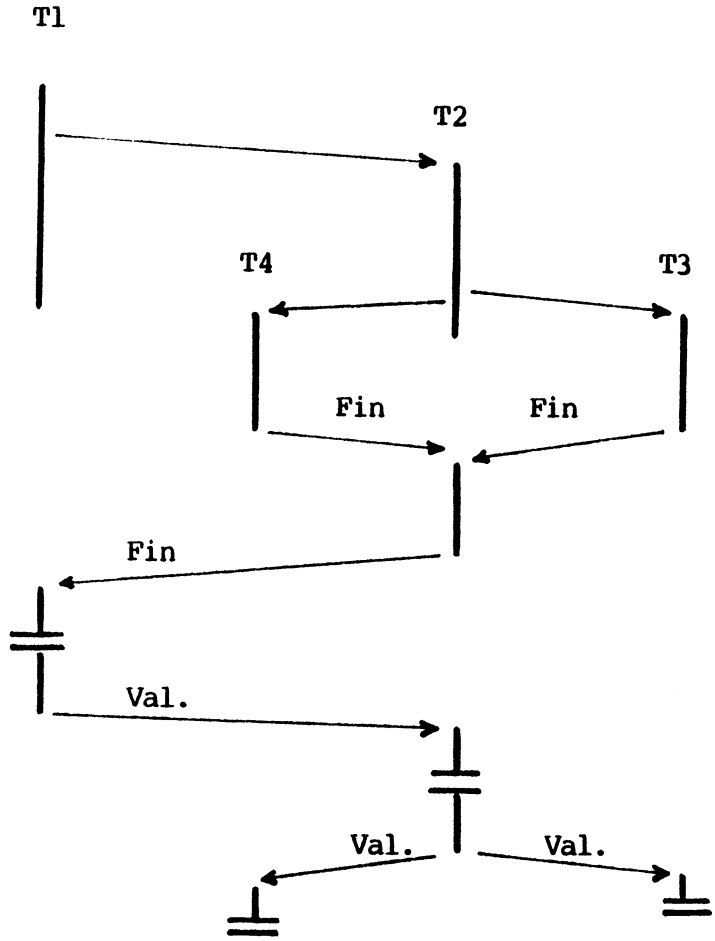


Figure 4.4

Cette technique est bien adaptée aux applications bâties sur un schéma d'exécution procédural. Cependant, dans certaines situations, cette technique s'avère pénalisante. Nous en donnons un exemple ci-dessous. Dans cette configuration, T3, initialisée par T2, s'exécute sur le même site que T1. Le fait que T1 ignore T3 dans la hiérarchie d'exécution implique le transfert sur le réseau d'un message inutile ('Valider') entre les sites A et B.

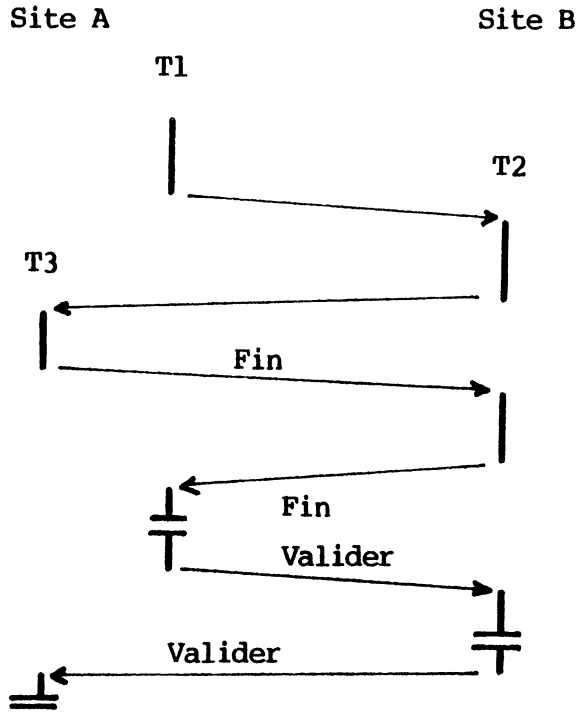
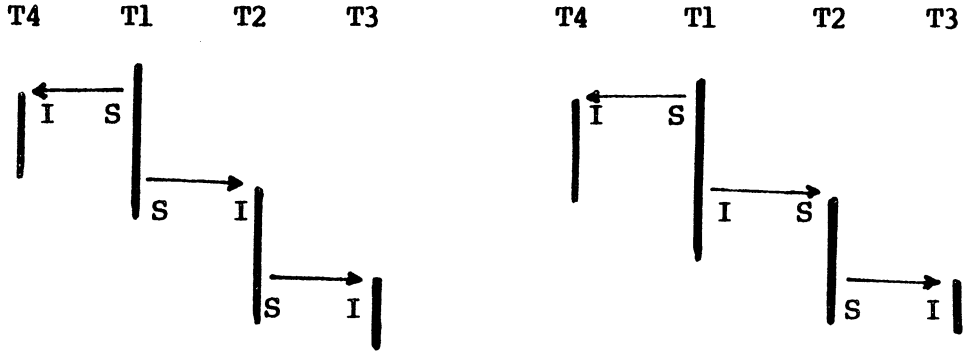


Figure 4.5

Nous examinons maintenant le cas où le supérieur n'est pas l'agent initial. L'arbre de validation est différent de l'arbre d'invocation. La technique utilisée pour définir l'arbre de validation est la suivante :

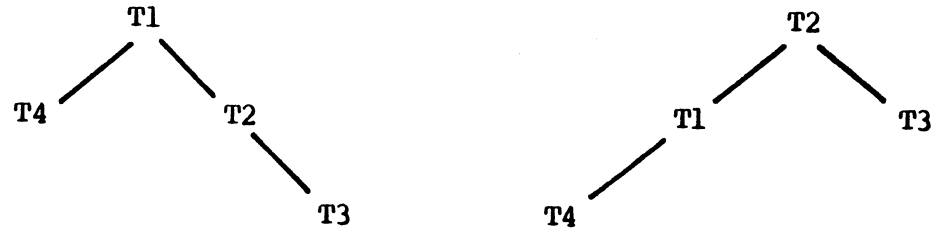
Pour chaque couple d'agents "invocateur-invoqué", on définit une relation "supérieur-inférieur". L'ensemble de ces relations définit l'arbre de validation (Figure 4.6-a)



T1 supérieur de la transaction T2 supérieur de la transaction

Figure 4.6-a

L'exemple de la figure 4.6-a, dans lequel T2 est supérieur, montre l'existence de deux arborescences distinctes reproduites en figure 4.6-b.



arbre d'invocation arbre de validation

Figure 4.6-b

Si l'identité du supérieur est connue à l'initialisation de la transaction, l'arbre de validation peut être construit sans problème en véhiculant l'identité du supérieur dans chaque message d'initialisation d'un agent.

Lorsque l'identité du supérieur n'est pas connue de l'agent initial, l'arbre de validation est construit dynamiquement. Il y a deux manières de construire l'arbre :

- à l'initialisation des agents,

- ou à la terminaison (avant l'étape de validation).

a) Construction de l'arbre de validation à l'initialisation

Dans le message d'initialisation, l'agent invocateur se présente comme "supérieur" ou "inférieur". Si la spécification de l'agent invoqué est incompatible avec la demande, un message d'erreur est renvoyé. Dans le cas contraire, aucune réponse n'est nécessaire, la relation est établie et l'acquittement est implicite.

A chaque étape de la construction de l'arbre, le système vérifie sa cohérence en appliquant les règles suivantes :

. Un agent ne peut pas avoir plus d'un correspondant supérieur.

. L'agent qui n'a que des correspondants inférieurs est le supérieur de la validation.

Le protocole de validation s'établit ainsi :

. Un agent attend les messages de terminaison de ses correspondants inférieurs (s'il en a).

. Lorsqu'il les a tous reçus, il envoie un message de terminaison vers son correspondant supérieur (s'il en a).

. Le supérieur de la validation (n'ayant pas de correspondant supérieur) valide localement, et envoie un message de validation à ses correspondants inférieurs.

. Un agent qui reçoit un message de validation, valide localement et renvoie le message de validation à ses correspondants inférieurs.

b) Construction de l'arbre de validation à la terminaison

La nature des messages échangés à la terminaison permet d'identifier le rôle de chaque agent dans le processus de validation (Figure 4.7).

. Un agent inférieur envoie un message 'Fin' à son père (agent invocateur).

. Un agent supérieur envoie un message différent, noté 'PaV' ('Préparer-à-Valider')à son père.

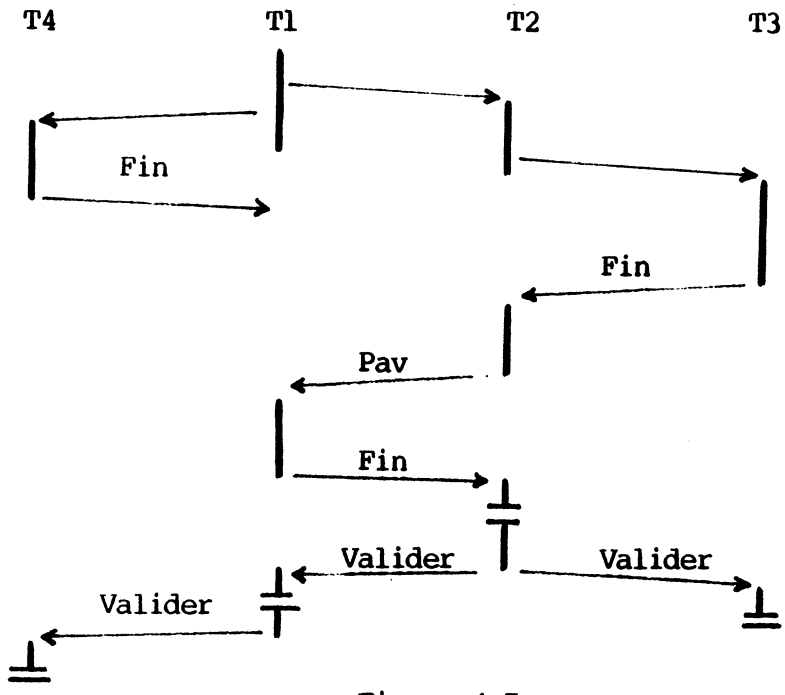


Figure 4.7

En différenciant ces deux messages, on permet ainsi à chaque agent de prendre connaissance du rôle des agents invoqués par lui dans le processus de validation. Le rôle de chaque agent est précisé à l'aide d'un paramètre de l'action FIN-Agent.

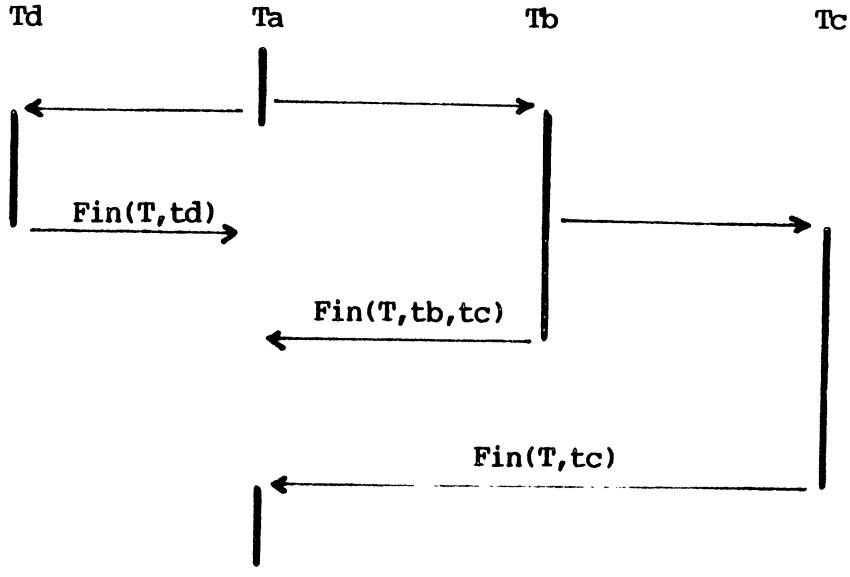
2.2.3. Validation à contrôle direct

Nous suivrons la même démarche que précédemment. Nous étudions en premier lieu le cas où le supérieur est l'agent initial, puis celui où le supérieur n'est pas l'agent initial.

Lorsque le supérieur est l'agent initial, son identité est passée à chaque agent dans le message d'initialisation. Un agent qui se termine envoie au supérieur un message comprenant :

- l'identificateur de la transaction,
- l'identité des agents invoqués par lui.

Cette technique permet d'avertir le supérieur de la terminaison de chaque agent et de lui faire connaître la liste exhaustive des inférieurs, comme le montre l'exemple de la figure 4.8.



T est l'identificateur de la transaction
ti est l'identificateur de l'agent Ti

Figure 4.8

Présentation des inférieurs au supérieur

Ta est le supérieur de la validation. Ta connait Tb et Td.

Si Tb se termine avant Tc, Ta prend connaissance de Tc par la liste (tb, tc) associée au message envoyé par Tb.

Si Tc se termine avant Tb, Tc se présente de lui même à Ta à l'aide de l'identificateur (T) inclus dans le message de terminaison.

Il faut noter que ces présentations réciproques ne nécessitent aucun message supplémentaire. Le coût marginal qu'elles introduisent est très faible car il correspond seulement à compléter certains messages par une liste d'identificateurs d'agents. Le supérieur maintient une liste des agents inférieurs qu'il met à jour chaque fois qu'il reçoit un nouveau message de terminaison. Lorsque tous les messages ont été reçus, le supérieur est assuré de connaître l'ensemble des agents qui coopèrent. Il valide localement et diffuse à tous les partenaires un message de validation.

Appliquée sans précaution aux schémas d'exécution procéduraux, cette technique peut s'avérer pénalisante comme le montre l'exemple de la figure 4.9. A la terminaison de T3, deux messages de synchronisation sont nécessaires : l'un vers T2 pour les besoins de l'application et l'autre vers T1 pour le protocole de validation. Cet inconvénient n'existe pas dans la méthode du contrôle hieérarchisé qui véhicule dans un même message les données de l'application et celles du protocole.

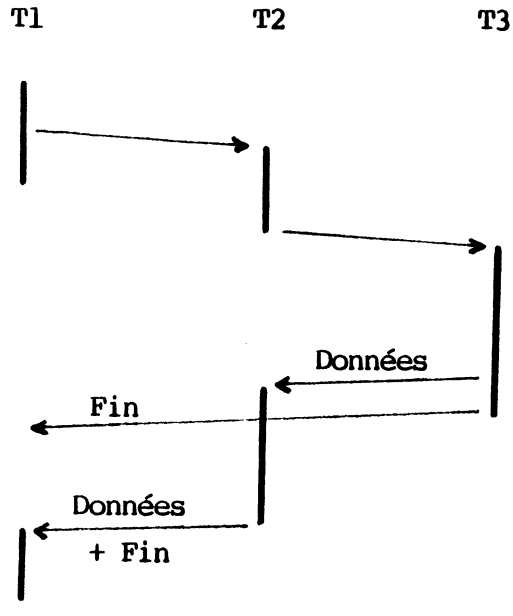


Figure 4.9

Nous examinons maintenant le cas où le supérieur n'est pas l'agent initial. Si l'identité du supérieur est connue de l'agent initial, nous sommes ramenés au problème précédent en passant l'identité de cet agent en paramètre du message d'initialisation.

Si l'identité du supérieur n'est pas connue de l'agent initial, il faut attendre l'initialisation de l'agent supérieur pour entreprendre sa présentation aux autres agents. L'agent invocateur est informé du fait que son fils est le supérieur. Il transmet cette information à sa filiation. De proche en proche, tous les agents prennent ainsi connaissance du supérieur. Si le nombre de sites concernés par la transaction est important (plus de 3), cette technique n'est pas réaliste en l'absence d'un réseau à diffusion.

2.2.4. Impact sur le protocole d'abandon

Une erreur fatale ou récupérable est détectée sur un site, puis la condition d'erreur est transmise à l'ensemble des partenaires. Nous avons vu au chapitre précédent (cf. § 2.2.3) le principe de cette propagation. Nous décrivons ici comment elle est mise en oeuvre pour une erreur fatale (la propagation d'une erreur récupérable s'effectue de la même manière).

La demande d'abandon peut être envoyée directement à tous les partenaires (contrôle direct) ou emprunter la voie hiérarchique (contrôle hiérarchisé).

a) Contrôle hiérarchisé

L'arbre d'invocation est utilisé pour propager le message d'abandon. Celui-ci est envoyé à la filiation directe de l'agent erroné et à l'agent invocateur (le message destiné à ce dernier contient l'origine de l'erreur). Un agent qui reçoit un message d'abandon exécute la procédure d'invalidation et redistribue le message d'abandon aux agents suivants :

- à sa filiation directe, à l'exception de l'agent qui était à l'origine du message reçu.

- à l'agent invocateur avec la cause de l'erreur.

On assure ainsi que tous les agents recoivent le message d'abandon et que l'origine de l'erreur remonte jusqu'à l'agent initial pour être fournie à l'utilisateur. Rappelons que les accusés de réception ne sont pas nécessaires entre l'agent émetteur d'un message d'abandon et les agents récepteurs (figure 4.10).

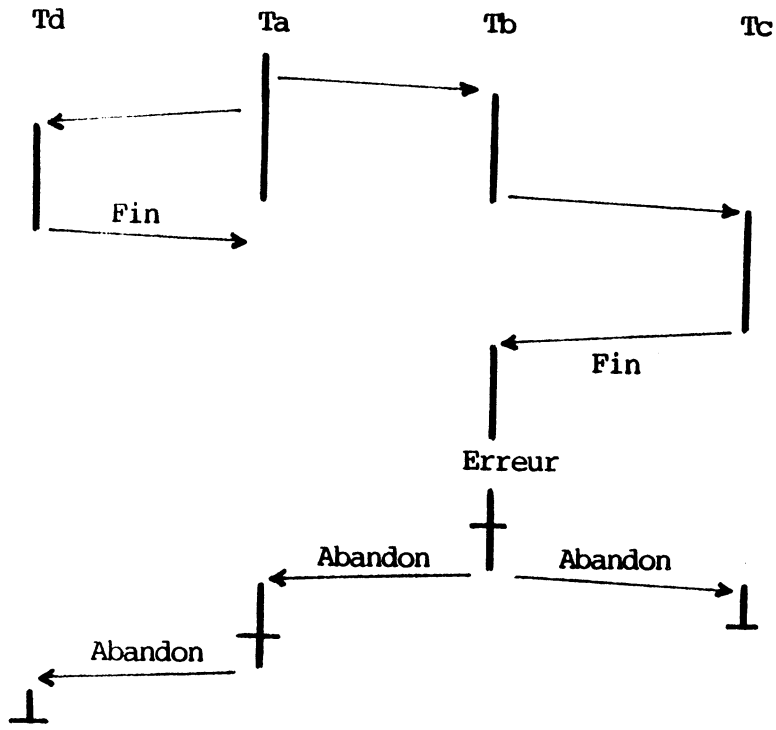


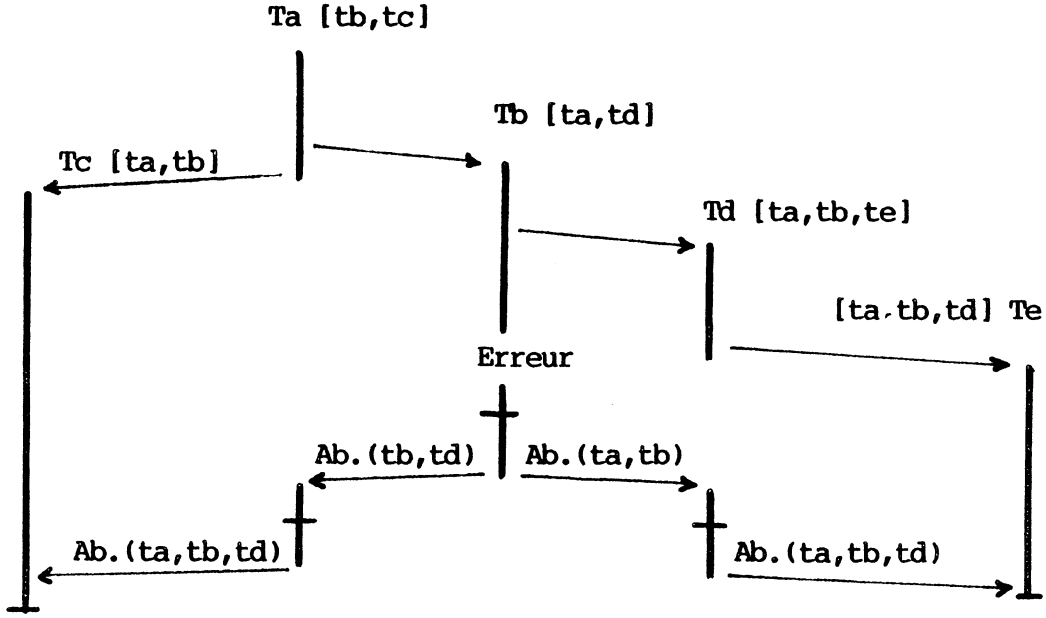
Figure 4.10

Abandon et contrôle hiérarchisé

b) Contrôle direct

Un agent qui a détecté une erreur envoie un message d'abandon à tous les partenaires qu'il connaît. Pour cela, chaque agent maintient une liste des partenaires connus. Cette liste lui est transmise à l'initialisation par l'agent invocateur puis est mise à jour chaque fois qu'il lance un nouvel agent. Cette liste n'offre qu'une vue partielle des partenaires.

Le message d'abandon, construit sur le site de l'agent erroné, comprend le code d'erreur et la liste des partenaires connus (appelée liste de diffusion). Ce message est adressé à tous les agents de la liste. Un agent qui reçoit un message d'abandon exécute la procédure d'invalidation et rediffuse le message aux agents, connus de lui, et qui ne sont pas inclus dans la liste de diffusion. A chaque étape de ce processus de rediffusion, la liste de diffusion est mise à jour dans le nouveau message. Par comparaison de la liste des partenaires connus et de la liste de diffusion, on assure que tous les agents reçoivent le message et que chaque agent n'en reçoit qu'un seul (figure 4.11).



[ ] Liste des partenaires connus

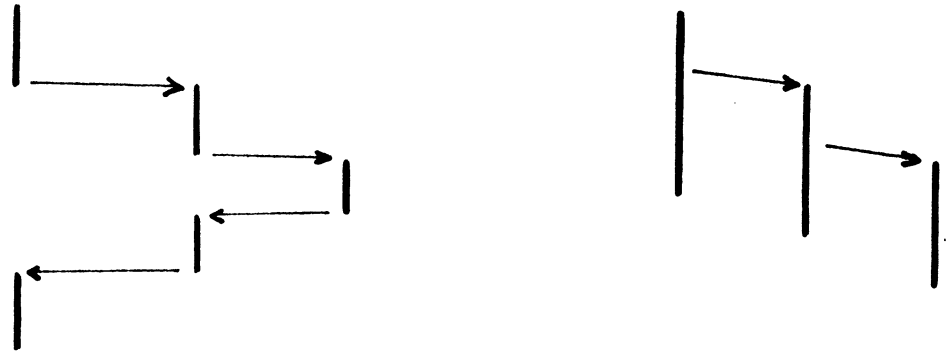
( ) Liste des partenaires qui ont recu le message

Figure 4.11 Abandon et contrôle direct

Le fait que le supérieur soit connu de tous les agents assure que le message d'erreur atteint l'agent initial, et par conséquent l'utilisateur, dès la première étape du processus de diffusion. Cette propriété n'est pas vraie dans le contrôle hiérarchisé.

2.2.5. Comparaison des contrôles hiérarchisé et direct

Dans ce paragraphe, on s'efforce de synthétiser l'ensemble des observations effectuées sur le choix du supérieur et sur la technique de propagation des messages des protocoles de validation et d'abandon entre les agents d'une transaction. L'objectif de cette analyse récapitulative consiste à déterminer le meilleur compromis pour un noyau transactionnel coopérant. Nous étayons notre argumentation sur les deux scénarios représentés en figure 4.12.



Exécution procédurale Exécution parallèle

Figure 4.12

Scénarios de référence

Dans cette étude, un fonctionnement en coroutines peut être assimilé à l'exécution procédurale. De même, la migration s'apparente à l'exécution parallèle. Dans la suite, on désigne par :

. n la profondeur d'exécution (la transaction a n+1 agents)

. t le temps de transfert d'un message entre deux sites. On suppose en outre que le temps d'exécution d'un agent sur un site est peit devant t.

La comparaison porte sur les trois aspects suivants : concentration des messages de terminaison vers le supérieur, diffusion du message de validation et diffusion d'un message d'abandon. Les critères d'évaluation sont le nombre de messages et le délai, calculé de la manière suivante :

. terminaison : délai entre la terminaison du dernier agent et l'arrivée du message correspondant au supérieur,

. validation : délai entre la validation du supérieur et celle du dernier agent,

. abandon : délai entre l'abandon du dernier agent et l'arrivée du message correspondant au supérieur.

Le tableau suivant résume les caractéristiques de chaque méthode pour les deux scénarios évoqués ci-dessus.

Contrôle	Terminaison		Validation		Abandon	
	m	d	m	d	m	d
Exécution Procédurale						
hiérachisé	n	nt	n	nt	n	nt
direct	2n-1	nt	n	t	n	t
Exécution Parallèle						
hiérarchisé	n	nt	n	nt	n	nt
direct	n	t	n	t	n	t

m : nombre de messages

d : délai d'acheminement du message

Pour des scénarios simples, les deux méthodes ont des comportements voisins : pour les transactions formées de 2 agents par exemple (n=1), les résultats sont identiques. Pour les schémas plus complexes, il convient de dissocier la terminaison d'une part, la validation et l'abandon d'autre part.

En ce qui concerne la propagation des messages de terminaison, la différence n'est sensible que dans une seule configuration : exécution procédurale et contrôle direct. Sur ce plan, la nature du contrôle ne constitue donc pas un critère de jugement décisif. En ce qui concerne le choix du supérieur, le contrôle hiérarchisé offre une souplesse que ne permet pas le contrôle direct. Pour cette raison, nous considérons qu'il constitue un meilleur choix.

Pour les messages de validation et d'abandon, le contrôle direct est plus avantageux car il diminue le délai d'acheminement des messages grâce au recouvrement de leur temps de transmission. Dans le contrôle hiérarchisé, du fait de l'utilisation de l'arbre d'invocation pour la propagation des messages , le délai d'acheminement est proportionnel à la profondeur de l'arbre. En résumé, la solution de compromis adoptée pour le noyau est la suivante :

- Les messages de terminaison sont acheminés par la voie hiérarchique : chaque agent attend la réception des messages de terminaison des agents invoqués par lui pour transmettre à son tour le message de terminaison à l'agent l'ayant invoqué. Le message de terminaison d'un agent indique le rôle joué par lui dans la validation (inférieur ou supérieur). Le choix du supérieur n'est pas systématique comme dans la plupart des systèmes existants; il incombe au programmeur d'application qui dispose pour cela de la primitive FIN-Agent.

- Les messages de terminaison ou d'abandon sont échangés directement entre les agents coopérants.

Le contrôle hiérarchisé repose sur l'existence de liaisons point-à-point uniquement entre les agents présentant un lien de filiation directe. Le contrôle direct requiert en outre des liaisons point-à-point entre le supérieur et les agents inférieurs. Ces liaisons sont utilisées pour acheminer les messages des protocoles et peuvent être différentes des liaisons nécessaires à la coopération des agents pour le compte de l'application. Le surcoût induit par la gestion de ces liaisons supplémentaires dans une architecture compatible OSI est très pénalisant. Cette observation nous a conduit :

- à utiliser pour l'acheminement des messages des protocoles des liaisons point à point pré-établies entre chaque paire de systèmes transactionnels coopérants,

- à développer un niveau "session" adapté au transactionnel coopérant. Les motivations et les caractéristiques de cette "session" sont décrites dans [Scot 83].

2.3. Procédure de validation

La procédure de validation que nous avons utilisée jusqu'à présent dans nos exemples est basée sur un "aller-retour" entre le supérieur et les inférieurs. Cette technique est appelée validation à 1-phase. Dans ce paragraphe, nous étudions les propriétés d'autres techniques ainsi que leur adéquation à divers scénarios d'application. Nous proposons enfin un protocole qui intègre les avantages respectifs de ces différentes méthodes afin de satisfaire la plus grande classe d'applications possible. Pour la lisibilité des exemples présentés, nous faisons ici l'hypothèse que le supérieur est la transaction initiale. Les résultats restent valables pour une configuration différente.

2.3.1. Validation à 0-phase : Auto-validation

La validation à 0-phase signifie en fait qu'il n'y a pas de procédure de validation globale. Un agent qui se termine valide immédiatement ses actions et envoie un message de terminaison au supérieur avec l'indication qu'il a validée.

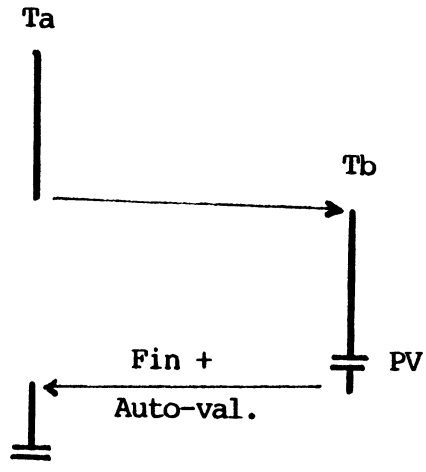


Figure 4.13

Auto-validation

Cette procédure ne garantit pas la cohérence de l'environnement distribué. En revanche, La disponibilité des ressources est immédiate. Il n'y a pas de zone grise.

2.3.2. Validation à 1-phase

Rappelons brièvement le principe de la validation à 1-phase.

- Les inférieurs envoient un message de terminaison au supérieur.
- Lorsqu'il a reçu tous les messages de terminaison, le supérieur valide localement et envoie à tous les inférieurs un message de validation (Figure 3.14).

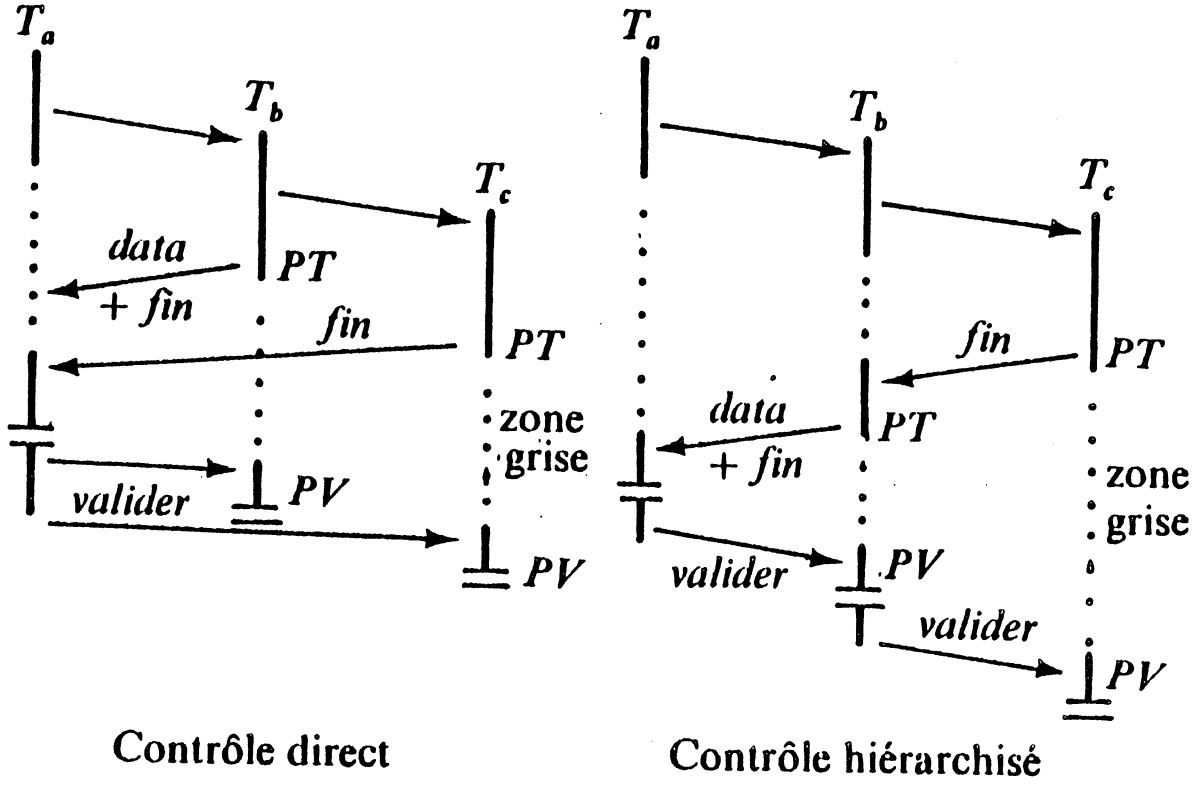


Figure 4.14

Validation à 1 phase

2.3.3. Validation à 2-phases

Le principe de la validation à 2 phases est le suivant :

- Les inférieurs envoient un message de terminaison au supérieur.

- Lorsqu'il a reçu tous ces messages, le supérieur diffuse un message de préparation à la validation (noté 'Pav'). Ce message indique à chaque agent inférieur que l'ensemble des partenaires est prêt à valider (tous les agents ont terminé).

- Les inférieurs acquittent cette information (message 'Prêt') ou demandent l'arrêt de la transaction (message 'Abandon').

- Lorsqu'il a reçu tous les messages 'Prêt' le supérieur valide localement et diffuse le message de validation (message 'Valider') (Figure 4.15).

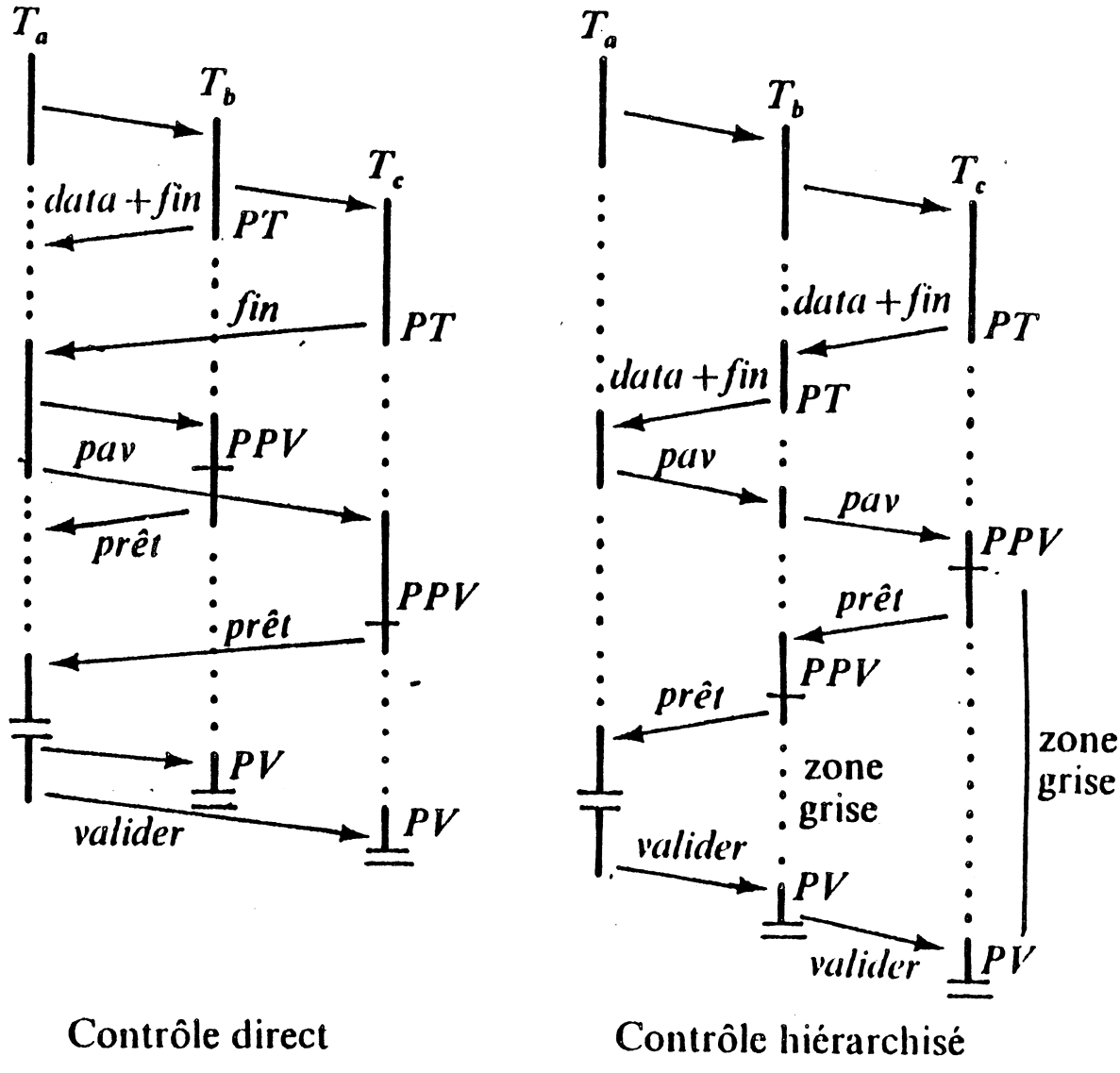


Figure 4.15

Validation à 2 phases

Ce protocole a les caractéristiques suivantes :

Il exige un aller-retour supplémentaire entre le supérieur et les inférieurs : 4(n-1) messages sont nécessaires si n est le nombre de sites. Le trafic réseau s'en trouve augmenté ainsi que le temps de réponse de la transaction.

Le point de pré-validation (noté PPV) d'un agent inférieur est distinct du point de terminaison. Il correspond à la transition d'état "ACTIF" → "PRET A VALIDER" après la réception du message 'Préparer-à-valider'.

Avant ce point, l'agent est considéré comme actif. Le système local peut donc décider d'abandonner l'agent (et libérer ses ressources) si une panne d'un partenaire ou du réseau est détectée.

La zone grise s'étend du point de pré-validation au point de validation. L'objectif essentiel du protocole à 2-phases est de synchroniser et de raccourcir les zones grises des agents inférieurs afin de minimiser le risque de panne pendant cette période. Cette propriété sera illustrée un peu plus loin en examinant des scénarios d'exécution.

Lorsque le contrôle de l'accès concurrent utilise la technique du verrouillage à deux phases, il est possible de libérer les objets manipulés en consultation au point de prévalidation sans compromettre la cohérence des données. Cette libération anticipée a des effets bénéfiques sur le degré de parallélisme.

Le supérieur peut exploiter la diffusion du message de préparation à la validation pour communiquer aux inférieurs la liste exhaustive de tous les agents de la transaction. Cette liste peut être utilisée par un agent inférieur, en zone grise et isolé du supérieur, pour enquêter sur le sort de la transaction. Il est possible, en effet, que la rupture de la liaison avec le supérieur soit postérieure à l'envoi par celui-ci de sa décision à un partenaire (fig. 4.16).

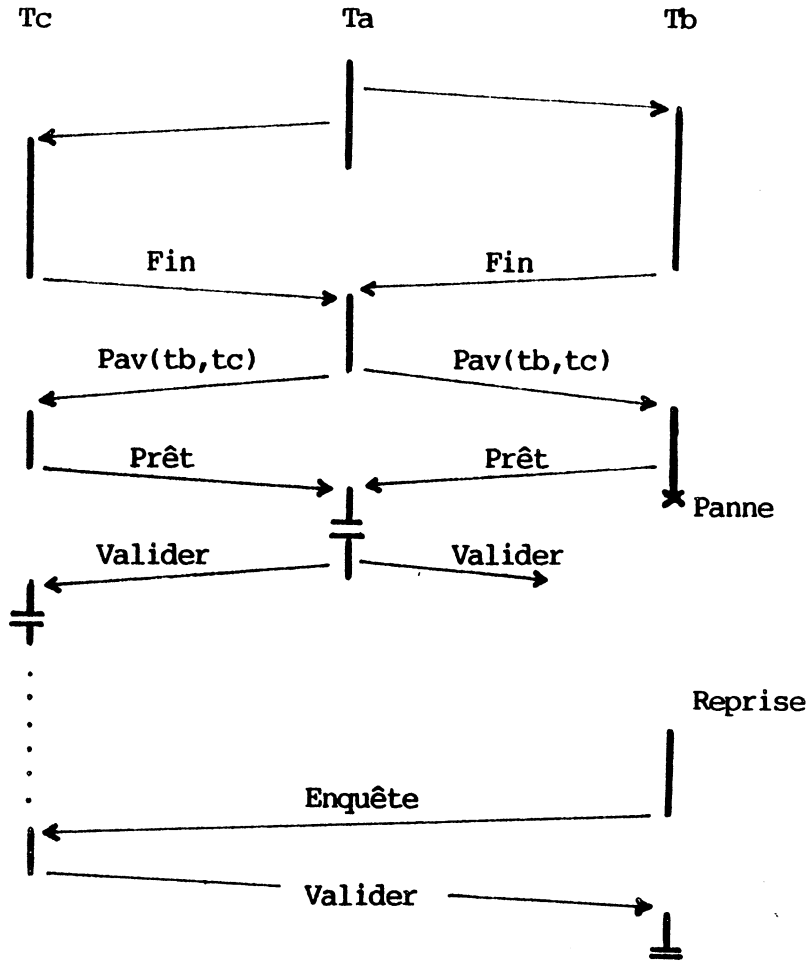


Figure 4.16

Procédure d'enquête

La procédure d'enquête fonctionne de la manière suivante :

- Diffusion auprès des partenaires connus d'une commande d'enquête (message 'Enquête').

- Si au moins un partenaire répond par un message 'Abandon', l'agent est abandonné.

- Si au moins un partenaire répond par un message 'Valider', l'agent valide localement.

- Si aucun partenaire ne répond ou si les seules réponses reçues font état de situations identiques (message 'Zone-grise'), l'agent ne peut pas conclure.

L'étude de ces différentes situations montre que le dialogue avec les partenaires permet dans certains cas, de conclure malgré l'absence de liaison avec le supérieur. La procédure d'enquête ne s'applique pas dans le cas où la panne isole l'agent inférieur du reste de ses partenaires.

Naturellement, il est possible d'appliquer cette technique au protocole à 1-phase. Cependant, la nature des échanges dans ce protocole ne permet pas à un agent de connaître à coup sûr tous ses partenaires (un agent ne connait que sa descendance directe et les agents qui étaient déjà connus de son père).

De nombreux aménagements ont été proposés depuis en vue d'améliorer le protocole à deux phases. Nous en décrivons deux exemples ci-dessous.

a) Validation imbriquée.

Le mécanisme de **validation imbriquée** [Gray 78], déjà mentionné au paragraphe 2.1.3, peut être appliqué en vue de réduire le nombre de messages entrant dans la réalisation du protocole de validation à deux phases (figure 4.17).

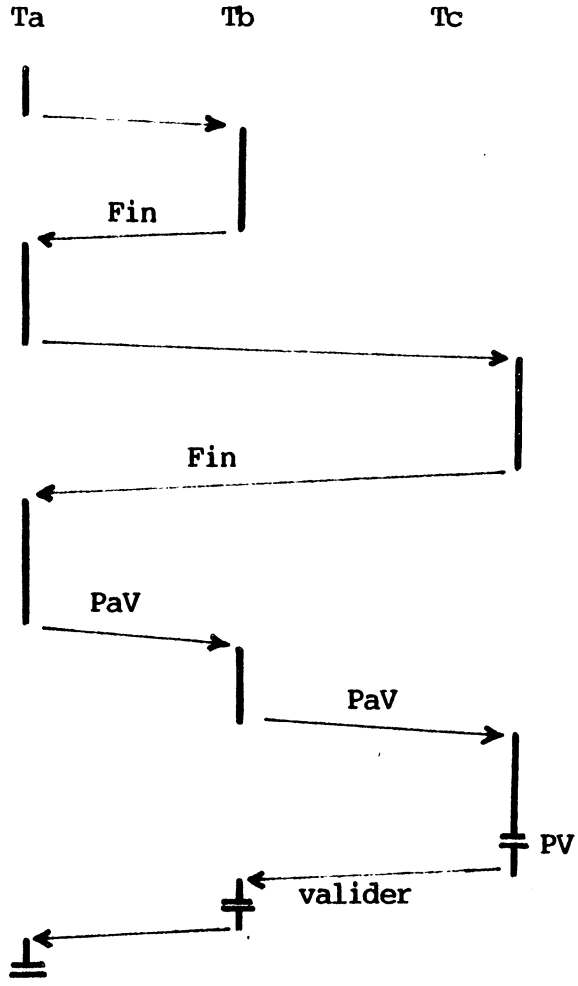


Figure 4.17

Validation imbriquée à 2-phases

L'agent initial ajoute au message de préparation à la validation la liste des agents inférieurs. Le message est envoyé au premier élément de cette liste qui l'achemine à son tour au suivant. Le dernier agent sur la liste est considéré comme le supérieur. Le message de validation est ensuite diffusé à l'ensemble des partenaires inclus dans la liste. Cette solution est particulièrement adaptée au cas des exécutions procédurales.

b) Validation à deux phases étendue.

Cette option consiste à généraliser le principe de l'enquête pour le cas d'un agent isolé du supérieur en se basant sur l'observation suivante : la procédure d'enquête a d'autant plus de chances d'aboutir que le nombre de sites mis au courant de la décision du supérieur est grand.

Le supérieur transmet l'ordre de validation, non seulement aux agents inférieurs concernés, mais aussi à un ensemble de sites dont l'identité est transmise dans le message de préparation à la validation. Cette méthode est particulièrement adaptée à un système distribué construit sur un réseau local à diffusion.

2.3.4 Validation à trois-phases [Skeen 82-a]

La validation à trois phases se présente comme une extension de la validation à deux phases étendue en vue d'accroître sa robustesse vis-à-vis des pannes du supérieur.

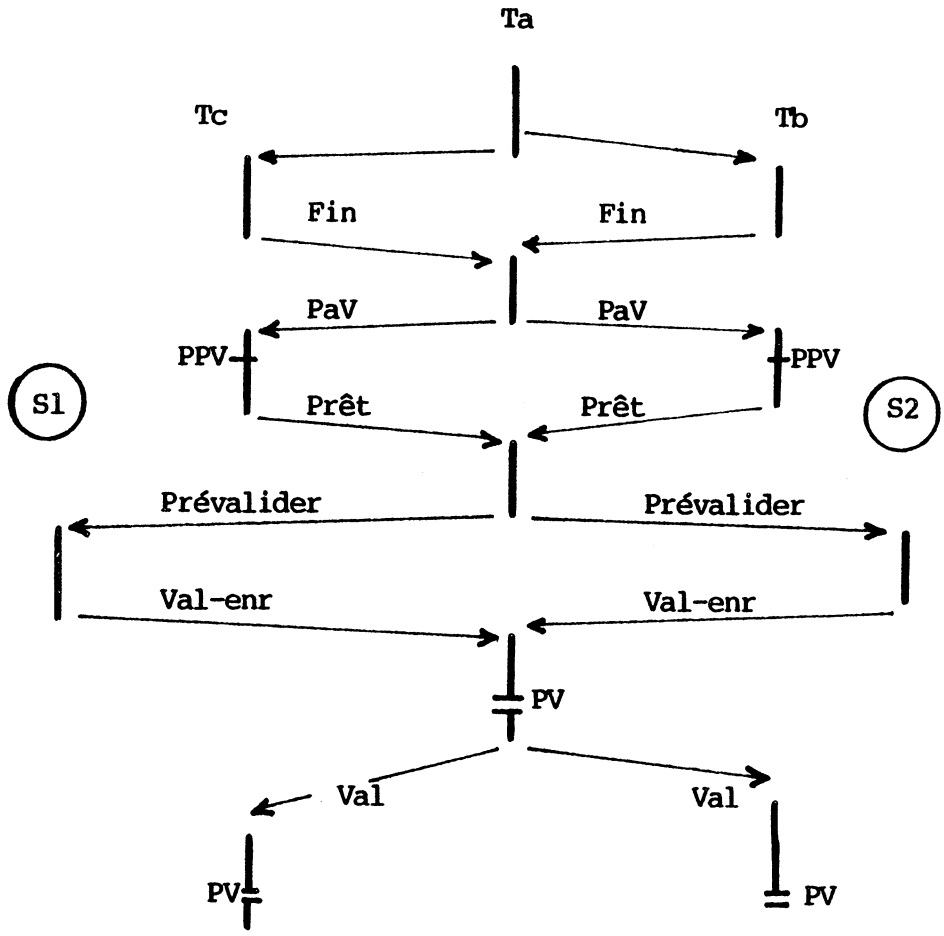
A chaque système transactionnel, est associé un ou plusieurs systèmes dits "de secours" situés sur des sites distincts qu'il tient informés en permanence de l'état des transactions qui sont sous son contrôle. Lorsqu'un site est défaillant, les systèmes de secours se concertent pour élire parmi eux un remplaçant auquel ils envoient toute l'information dont ils disposaient concernant le site disparu. Ce suppléant informe les sites des agents inférieurs de ses nouvelles prérogatives et répond aux demandes d'enquête émanant des inférieurs. Le protocole fonctionne de la manière suivante :

- les inférieurs envoient leur message de terminaison.

- lorsqu'il a reçu tous ces messages, le supérieur diffuse un message de préparation à la validation, acquitté par les inférieurs. Ceci correspond exactement à la première phase du protocole à deux phases.

- Lorsqu'il a reçu tous les messages, le supérieur diffuse un message de prévalidation (noté 'Prévalider') aux sites de secours qui accusent réception (message 'Val-enregistrée').

- Lorsqu'il a reçu tous les acquittements correspondants, le supérieur diffuse le message de validation (figure. 4.18).



S1,S2 : sites de secours

Figure 4.18

Validation à 3 phases

Ce protocole présente les caractéristiques suivantes :

- Il résiste aux pannes du supérieur. On trouvera dans [Skeen 82-b] une démonstration de cette affirmation. Dès que l'identité du suppléant est connue des sites supportant les agents inférieurs, une demande d'enquête peut être satisfaite.

- Il est cher : le nombre de messages et le temps de réponse d'une transaction s'en trouvent augmentés.

- La phase d'élection d'un nouveau supérieur, non représentée dans le schéma de la figure 4.18, est complexe car elle doit prendre en compte la disparition et la reconnexion dynamiques d'un élément de l'ensemble des sites de secours.

- Lors d'une reprise après panne, le supérieur doit se conformer à la décision prise par son suppléant. Ce protocole crée une zone grise chez le supérieur.

- Le protocole ne résoud pas le problème de l'agent inférieur isolé à cause d'un partitionnement du réseau.

Il existe plusieurs variantes de ce protocole. L'une d'entre elles a fait l'objet d'une réalisation expérimentale dans le système SDD-1 [Hammer 80].

2.3.5. Comparaison des procédures de validation

Le tableau de la figure 4.19 résume les observations effectuées dans les paragraphes précédents sur le comportement des procédures de validation. On note n le nombre d'agents de la transaction et t le temps de transmission d'un message du protocole. La comparaison porte sur les critères suivants :

- Nombre de messages de la procédure de validation.

- Temps de réponse de la transaction, limité ici à la période comprise entre la terminaison du dernier agent et la validation de l'agent initial.

- Durée de l'étape de validation, calculée comme la période comprise entre la terminaison du dernier agent et sa validation. On suppose qu'il y a un recouvrement parfait des temps de transmission des messages liés à une opération de diffusion (hypothèse optimiste dans le cas de communications point-à-point). Sur ce critère, nous distinguons deux valeurs selon que la validation s'exécute sans problème, ou qu'elle requière une enquête.

- Durée de la zone grise d'un agent inférieur. On rappelle que la probabilité de panne d'un agent inférieur en zone grise est proportionnelle à la durée de celle-ci.

- Capacité de l'algorithme à résister à une panne du site supportant le supérieur.

- Capacité de l'algorithme à résister à un partitionnement du réseau. Nous distinguons deux cas selon que la panne isole le supérieur ou un agent inférieur.

	1 phase	2 phases	2 phases imbriquées	2 phases étendues	3 phases
Nombre de messages	2(n-1)	4(n-1)	3(n-1)	4(n-1)+p (5)	4(n-1)+2p (5)
Temps de réponse	t	3t	(n+1)t	3t	5t
Durée validation : sans panne	2t	4t	nt	4t	6t
après enquête		5t+D (2)	nt+D (2)	5t+D (2)	7t+D (2)
Durée zone grise	2t+E (1)	2t	0 à (n-1)t (4)	2t	4t
Résistance panne du supérieur	non	oui si réponse à enquête (3)	oui si réponse à enquête (3)	oui si réponse à enquête (3)	oui
Résistance panne du réseau					
Supérieur isolé	non	oui si réponse à enquête	oui si réponse à enquête	oui si réponse à enquête	oui
Inférieur isolé	non	non	non	non	non

Figure 4.19

Comparaison des protocoles de validation

(1) E représente le temps d'exécution de la transaction depuis l'entrée d'un agent en zone grise. E est nul dans la procédure de validation à deux phases puisqu'aucun agent ne rentre en zone grise avant la terminaison de tous ses partenaires. Dans la validation à une phase, E peut être très important (cf. §2.3.6).

(2) D représente le délai avant déclenchement de la procédure d'enquête.

(3) Si la réponse à l'enquête fait état d'un ordre de validation ou d'abandon, l'agent inférieur peut conclure : le protocole n'est pas bloquant.

(4) La durée de la zone grise dépend de la place de l'agent dans l'arbre d'invocation.

(5) p désigne le nombre de sites auxquels est envoyé la décision du supérieur.

(6) p désigne le nombre de sites de secours.

Ce tableau récapitulatif montre que :

. Le temps de réponse (vu de l'opérateur) et le nombre de messages nécessaires à la validation croissent rapidement avec la complexité du protocole.

. La zone grise d'un agent inférieur est minimale pour le protocole à deux phases. De ce point de vue, on observe que le protocole à trois phases augmente cette valeur et, par conséquent, la probabilité de panne en zone grise, ce qui va à l'encontre de l'objectif souhaité.

- Par sa structure particulière d'acheminement du message de préparation à la validation, le protocole à deux phases imbriqué diminue le nombre de messages mais accroît le temps de réponse et la durée des zones grise.

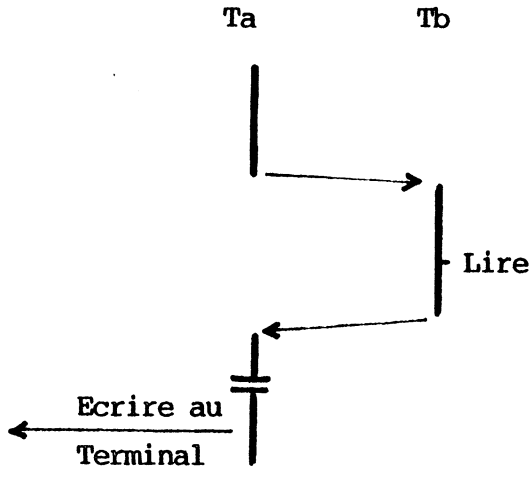
- On remarquera à nouveau qu'aucun protocole ne garantit la terminaison d'un agent inférieur en zone grise et isolé de ses partenaires.

En complément, le lecteur trouvera dans [Cooper 82] une étude comparable mais limitée aux protocoles à une phase, à deux phases et à la validation imbriquée à une phase, évoqué en 2.1.3. Cette évaluation utilise un modèle probabiliste d'un système réparti pour déterminer la durée des zones grises et en déduire le nombre moyen d'agents bloqués à cause d'un partitionnement du réseau. Selon cette étude, le protocole à deux phases est meilleur que la validation imbriquée à une phase, elle même supérieure au protocole à une phase.

2.3.6. Choix d'une procédure de validation

Les paragraphes précédents ont montré l'existence d'une gamme de protocoles de validation globale de complexité croissante et d'efficacité inverse. Cette augmentation de complexité, introduite par la gestion de la redondance, se traduit par un gain de fiabilité et de disponibilité.
En ce qui nous concerne, nous n'avons pas basé notre choix exclusivement sur l'analyse des propriétés théoriques ou des performances de ces mécanismes. Conformément à notre approche pragmatique, nous avons aussi pris en compte l'adéquation de ces techniques à différents types d'applications ; cette analyse est présentée ici.

Scénario a : consultation seule



Ta requiert le service de Tb pour consulter une donnée située sur B (état d'une réservation aérienne ou d'un compte bancaire par exemple)

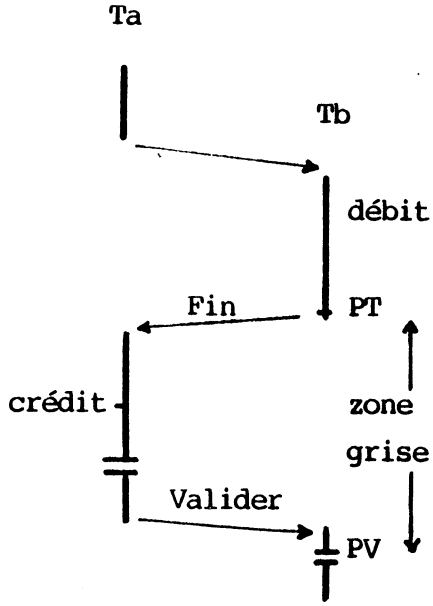
Cette information est simplement affichée au terminal de l'utilisateur

Figure 4.20

Dans ce cas très simple, la synchronisation des étapes de validation est inutile. On utilisera donc une procédure de validation à 0-phase.

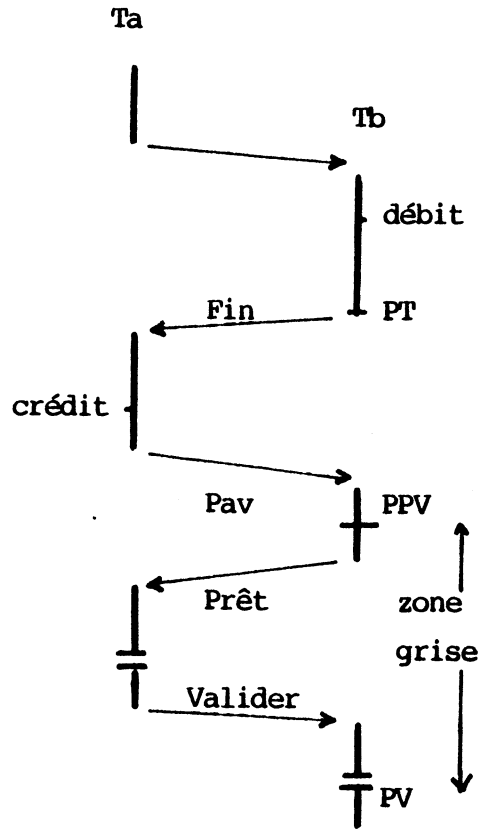
Scénario b : Mises à jours correlées (2 agents)

Deux mises à jours correlées sont réalisées par deux agents s'exécutant sur deux sites A et B. C'est le cas, par exemple, d'une opération de compensation bancaire qui crédite un compte sur un site et débite un compte sur un autre site.



Validation 1-phase

Figure 4.21-a



Validation 2-phases

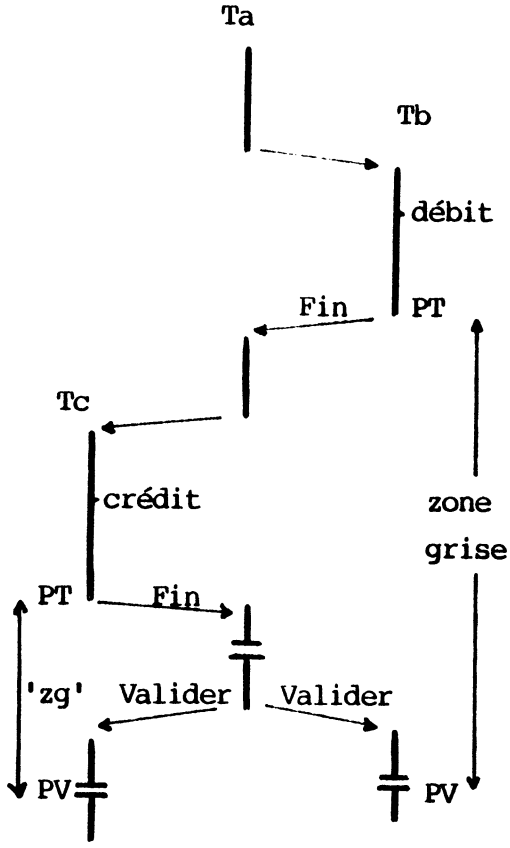
Figure 4.21-b

On considère dans cet exemple que le temps d'exécution de Ta après réponse de Tb est petit par rapport au temps de transfert d'un message sur le réseau. Dans les deux méthodes, la zone grise correspond à peu près à un échange de messages entre A et B. La règle d'atomicité interdit à Tb de relâcher ses ressources au point de pré-validation. On remarquera d'autre part que la procédure d'enquête n'a aucun intérêt dans le cas d'un scénario comportant deux agents.

Le protocole à 2-phases n'offre ici aucun avantage car il requiert un échange supplémentaire sans gain en disponibilité des ressources.

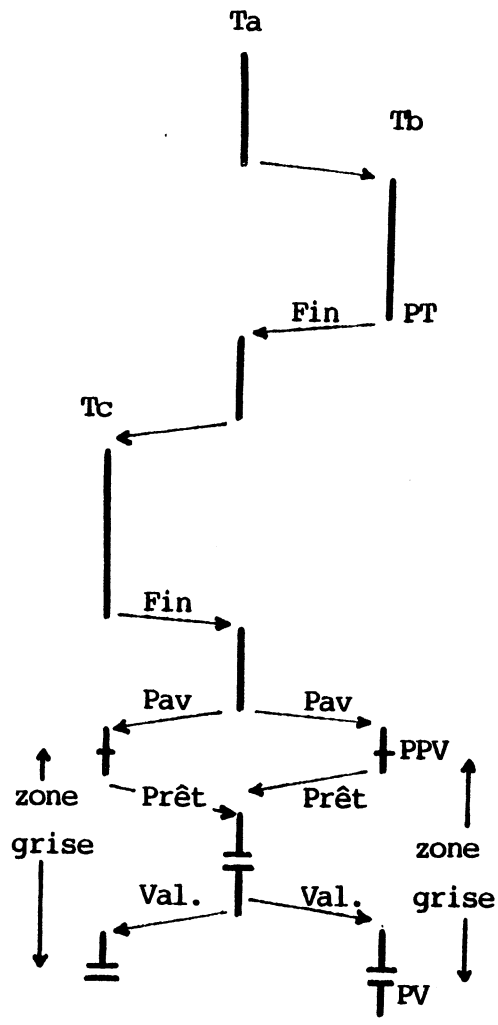
Scénario c : Mises à jour corrélées (3 agents)

On reprend l'exemple précedent mais, cette fois, l'opération de crédit s'exécute sur un troisième site C. La transaction globale est constituée de 3 agents s'exécutant selon un schéma procédural.



Validation 1-phase

Figure 4.22-a



Validation 2-phases

Figure 4.22-b

Dans le protocole à 2-phases (figure 4.22-b), les zones grises de Tb et Tc durent le temps d'un échange avec Ta.

Dans le protocole à 1-phase (figure 4.22-a), la zone grise de Tb inclut également le lancement et l'exécution de Tc. La probabilité d'avoir une panne pendant que Tb est en zone grise est donc considérablement augmentée.

Pour ce scénario, le protocole à 2-phases est recommandé si la disponibilité des ressources sur le site B est un paramètre critique.

2.3.7. Procédure de validation mixte

L'étude des scénarios précédents montre que :

- Le protocole de validation à 0-phase est indispensable pour les applications qui ne requièrent pas la cohérence forte.

- Certaines applications sont très pénalisées par le protocole à 2-phases, en particulier celles qui ont un schéma d'exécution très simple (les transactions constituées de deux agents entrent dans cette catégorie).

- Les applications qui requièrent un grand degré de fiabilité au niveau de la disponibilité des ressources nécessitent l'utilisation du protocole à 2-phases. Cette observation s'applique surtout à des schémas d'exécution tels que le scénario c. Nous pensons que le protocole à trois phases est trop pénalisant en rapport du gain en disponibilité qu'il procure; c'est pourquoi nous ne l'avons pas retenu dans SCOT.

Les trois protocoles sont nécessaires, le choix de l'un d'entre eux étant un compromis entre le coût supplémentaire induit par une synchronisation plus sophistiquée et une meilleure disponibilité des ressources. Le protocole de validation globale proposé dans SCOT fournit un mécanisme général dans lequel les trois méthodes cohabitent et laisse le choix de la technique appropriée au programmeur d'application [Balter 81]. Ce protocole de validation mixte fonctionne de la manière suivante :

- Chaque agent se voit affecter un type de protocole par le concepteur de l'application. Ce choix est spécifié en paramètre de la primitive FIN-Agent (*).

- Cette information est transmise au supérieur dans le message de terminaison.

- Le supérieur procède ensuite avec chaque agent inférieur selon le protocole indiqué.

* Cette approche permet d'introduire par la suite le protocole à 3 phases si une application l'exige.

Exemple : scénario c

Dans l'exemple illustré par la figure 4.23, Tb demande une procédure à 2-phases et Tc une procédure à 1-phase. Le supérieur exécute la phase de préparation à la validation avec Tb seulement. Sur réception de l'accord de Tb, la transaction globale est validée.

Cette technique permet de raccourcir la zone grise de Tb tout en évitant des échanges inutiles avec Tc.

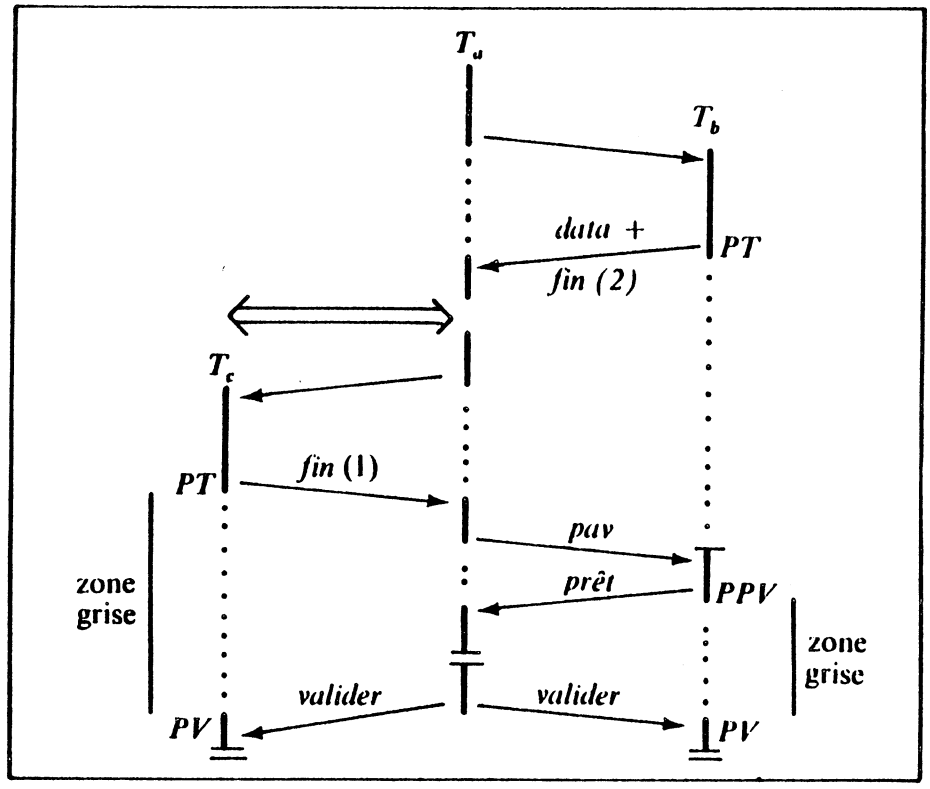


Figure 4.23

Validation mixte : scénario c

2.4. PROCEDURE DE REPRISE

Un système transactionnel contrôle la reprise des transactions dans les deux cas suivants :

- Redémarrage du système après une panne locale.
- Reconnexion d'un système coopérant qui avait disparu à la suite d'une panne du site ou du réseau.

2.4.1. Redémarrage du système local

Une procédure de reprise est exécutée, qui s'efforce de trouver une issue à tous les agents qui s'exécutaient sous le contrôle du système interrompu. Le journal local est utilisé pour déterminer le traitement propre à chaque agent en cours d'exécution à l'instant de la panne.

Si état = "ACTIF" ou "INVALIDE", la procédure d'invalidation est exécutée.

Si état = "VALIDE", la procédure de validation est exécutée.

Si état = "PRÊT À VALIDER" (agent inférieur en zone grise), le système transactionnel local ne peut pas prendre une décision sans consulter ses partenaires. Il y a deux manières de procéder.

1. Attendre une information du supérieur (reprise contrôlée par le supérieur).

2. Enquête auprès du supérieur (reprise contrôlée par l'inférieur).

Dans l'attente d'une information qui permette de sortir l'agent de la zone grise, la cohérence exige que les objets manipulés ne soient pas rendus visibles aux transactions concurrentes. Pour réaliser cette isolation, il est nécessaire que l'identité de ces ressources ait été enregistrée auparavant dans le journal local (cette journalisation a lieu au point de terminaison pour la validation à 1-phase et au point de prévalidation pour la validation à 2-phases).

2.4.2. Redémarrage d'un système distant

Nous désignons sous ce terme le rétablissement du dialogue avec un système momentanément disparu du fait d'une panne du site ou du réseau entre les deux sites.

Lorsqu'un système coopérant disparait, le système transactionnel local s'efforce de trouver une issue aux agents qui coopéraient avec des agents du site disparu. Pour chacun d'eux, le traitement suivant est appliqué :

Si état = "ACTIF", exécuter la procédure d'invalidation.

Si état = "VALIDE", aucune action.

Si état = "INVALIDE", aucune action.

Si état = "PRÊT À VALIDER", le système ne peut conclure seul dans le cas où le supérieur réside sur le site en panne. Si d'autres partenaires sont accessibles, une enquête auprès d'eux peut, dans certains cas, apporter une solution immédiate (cf. Figure 4.16). Dans le cas contraire (pas de partenaire accessible ou pas de réponse positive), l'agent est suspendu, en attente de reconnexion.

Lorsque la communication est rétablie, les systèmes transactionnels s'efforcent de trouver une solution aux problèmes laissés provisoirement en suspens par la panne. Comme au paragraphe précédent deux méthodes sont possibles suivant que l'initiative de la reprise est laissée au système qui contrôle le supérieur ou à celui qui contrôle l'inférieur. Ces deux techniques sont discutées dans les paragraphes suivants.

2.4.3. Reprise contrôlée par le supérieur

Dans cette approche, le rôle d'un agent inférieur est passif. L'initiative est laissée au supérieur, les inférieurs se bornant à répondre aux sollicitations du supérieur.

Avant d'envoyer les messages de validation, le supérieur enregistre sur son journal local la liste des inférieurs auxquels ce message est destiné. Un inférieur qui reçoit un message de validation, valide localement et renvoie un accusé de réception au supérieur (Figure 4.24).

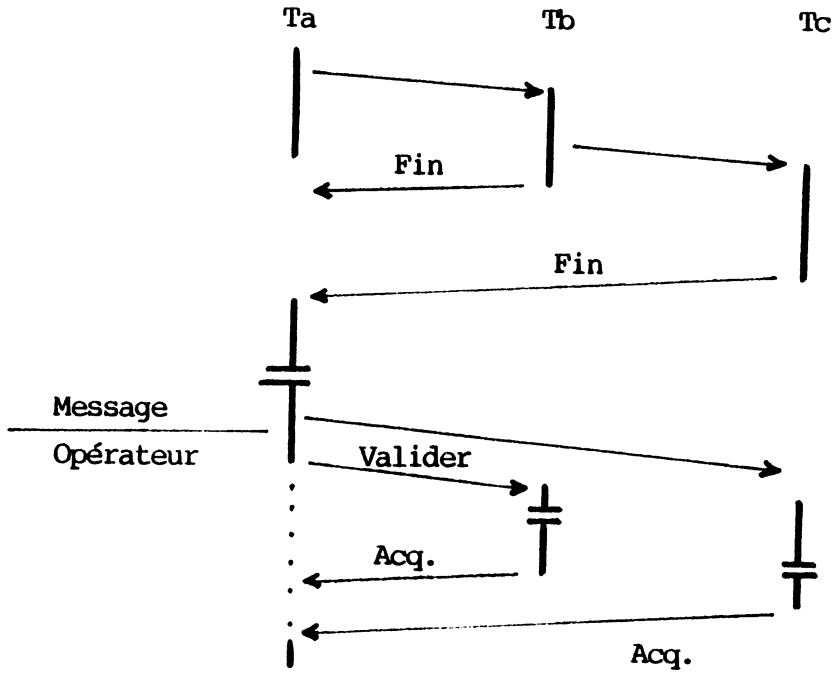


Figure 4.24

Reprise contrôlée par le supérieur

A chaque acquittement reçu, le supérieur enregistre sur le journal la validation de l'agent correspondant. Lorsqu'il a reçu tous les messages d'acquittement, le supérieur purge le journal des informations concernant la transaction. On notera que cette procédure d'acquittement ne retarde pas la terminaison de la transaction vue de l'utilisateur.

Au redémarrage du système après une panne locale, l'exploration du journal met en évidence les agents qui n'ont pas renvoyé leur accusé de réception. Pour ceux-là, le système renvoie le message de validation. Si le correspondant a déjà reçu le message (avant la panne), le nouveau message ne donnera lieu à aucun traitement, seul un acquittement sera renvoyé. Il est également possible de recevoir deux accusés de réception dont l'un est ignoré. De la même manière, lorsque la reconnexion d'un site distant est signalée, l'exploration du journal met en évidence les agents du site concerné qui n'ont pas renvoyé leur accusé de réception. La procédure de renvoi est ensuite la même que précédemment.

2.4.4. Reprise demandée par les inférieurs

Dans cette approche, le supérieur est passif. Seuls les agents qui se trouvent bloqués en zone grise envoient des messages d'enquête pour essayer de trouver une issue.

Au point de terminaison (validation à 1-phase) ou de prévalidation (validation à 2-phases), les inférieurs enregistrent dans leur journal l'identité du supérieur. L'agent supérieur n'exécute aucune action particulière.

Au redémarrage du système après une panne locale, les agents dans l'état "PRÊT À VALIDER" lancent une procédure d'enquête.

- Si un partenaire au moins répond 'Valider', l'agent est validé,
- Si un partenaire au moins répond 'Abandon', l'agent est abandonné.
- Si personne ne répond ou si toutes les réponses font état de situations pareillement ambiguës, l'agent ne peut conclure. Dans ce cas, l'agent est mis en attente de la reconnexion d'un site susceptible de le sortir de la zone grise.

Lorsqu'un système transactionnel est averti de la reconnexion d'un site distant, il recherche si des agents locaux dans l'état "PRET A VALIDER" sont susceptibles d'obtenir une réponse du système qui se reconnecte. Dans ce cas, la procédure d'enquête est exécutée.

2.4.5. Comparaison des deux méthodes

La reprise contrôlée par le supérieur est basée sur l'envoi des acquittements par les agents inférieurs. En fonctionnement normal, cet échange de messages est nécessaire. Au contraire, la procédure d'enquête requiert un dialogue seulement dans le cas précis où, après

une panne, il y a un agent dans l'état "PRÊT À VALIDER". Cette situation est très peu fréquente et par conséquent la surcharge en messages est largement moins importante que dans la méthode précédente. C'est pourquoi nous recommandons cette technique qui a été choisie pour SCOT.

En contre-partie la gestion des journaux est plus délicate. En effet, dans le premier cas, le temps de rétention de l'information associée à un agent en cours de validation (liste des inférieurs, acquittements reçus) est limité par la réception du dernier acquittement. Passé ce point, le journal peut être purgé. Dans le second cas, le temps de rétention n'est pas aisé à prévoir puisque la demande d'information est sollicitée par les agents inférieurs. Il y a donc un problème supplémentaire de nettoyage périodique du journal à résoudre.

Remarque : La même procédure d'enquête est utilisée dans deux situations :

- lorsque le supérieur a disparu, pour chercher chez les autres agents inférieurs un espoir de solution,

- à la reprise du système pour connaître l'issue de la transaction qui a été décidée pendant la panne.

Il est curieux de constater que de nombreux systèmes répartis l'utilisent dans le premier cas mais l'ignorent dans le second et lui préfèrent la méthode des accusés de réception qui est beaucoup plus lourde. C'est le cas, en particulier, de SDD-1 et du Système R*.

3. CONTROLE DE L'ACCES CONCURRENT : EVALUATION DE PERFORMANCE

3.1. Choix du verrouillage à deux phases : justification

A l'époque de la réalisation du noyau SCOT, la définition d'un mécanisme de contrôle de l'accès concurrent s'est posée en termes de choix entre le verrouillage à deux phases et l'ordonnancement par estampillage. Les techniques de contrôle par certification étaient encore peu connues, notamment en ce qui concerne les environnements distribués. Au vu de l'argumentation développée dans les chapitres 2 et 3, le verrouillage à deux phases nous a paru un meilleur choix pour les raisons suivantes :

- Il autorise la gestion simultanée de plusieurs degrés de granularité (enregistrement logique, page physique, partition d'une base, etc.).

- Il permet la prise en compte simultanée de plusieurs niveaux de cohérence; dans la perspective d'un noyau minimal, cet argument est fondamental. Les stratégies s'expriment au niveau de l'application à l'aide des actions VERROUILLER-RELACHER (verrouillage non deux phases). Cette flexibilité n'est pas aussi simple à réaliser avec l'estampillage.

- Moyennant certaines précautions, sur lesquelles nous reviendrons plus loin, l'évolution vers une configuration répartie est aisée. Cet élément constitue aussi un argument de poids dans la mesure où il permet la récupération des systèmes de contrôle existants. De ce point de vue, l'estampillage introduit des modifications majeures au niveau des méthodes d'accès pour prendre en compte l'estampillage des objets.

- Si on excepte la méthode de détection centralisée des interblocages, les autres algorithmes de verrouillage et les méthodes d'estampillage présentent les mêmes caractéristiques de robustesse vis à vis des pannes.

- Si on se reporte aux conclusions des sections consacrées au contrôle de l'accès concurrent dans les deux chapitres précédents, on constate que la performance ne constitue pas un facteur discriminatoire entre verrouillage et estampillage, chaque méthode ayant son champ d'application privilégié (transactions longues ou fort taux de conflits pour les premières, transactions courtes et faibles taux de conflit pour l'estampillage).

Le choix du verrouillage étant acquis, le reste de cette section est consacré à la détermination du meilleur choix parmi les variantes présentées au chapitre précédent. Cette analyse s'appuie sur les résultats des études de performance réalisées dans le projet SCOT à partir de modèles de simulation décrits et résolus à l'aide du progiciel QNAP [Merle 78]. Etant donnée la complexité de ces modèles, particulièrement en environnement distribué où le nombre de paramètres à prendre en compte est considérable, nous avons réalisé ces travaux en deux étapes :

a. Etude des algorithmes de verrouillage dans un système centralisé : cette phase exploratoire a permis d'ignorer la distribution et les problèmes posés par le réseau. Les efforts ont été concentrés sur la performance respective des algorithmes de détection et de prévention des interblocages.

b. Etude en environnement réparti : dans cette deuxième phase, un réseau de télécommunication et des scénarios d'exécution répartie inspirés par l'application bancaire sont introduits.

Dans cet ouvrage, nous nous limitons à décrire les résultats fondamentaux pour l'argumentation nécessaire à la définition du noyau. Le lecteur trouvera dans [Scot 81, Scot 82] tous les développements relatifs à ces études de performance. Les éléments du modèle, les hypothèses de modélisation et la nature des expérimentations effectuées sont brièvement décrites en 3.2. Le paragraphe 3.3 correspond à la première étape de nos travaux : il justifie le choix de prévention. Le paragraphe 3.4 enfin compare plusieurs méthodes de prévention en vue de déterminer le meilleur choix.

3.2. Evaluation des techniques de verrouillage : le modèle

3.2.1. Représentation du système : hypothèses de modélisation

Nous décrivons successivement la manière de représenter la base de données, les transactions, le système transactionnel, le système d'exploitation et le réseau de communication.

a) La base :

Sur chaque site réside une base indépendante constituée de N granules (dans nos expériences toutes les bases ont la même taille). Le granule est l'entité verrouillée par un agent local. L'impact de la granularité n'est pas pris en compte ici. Sur un site, il n'y a pas de conflit entre deux agents d'une même transaction globale et l'accès aux granules est aléatoire et uniformément réparti.

b) Les transactions :

On considère deux types de transactions. Les transactions locales, composées d'un seul agent, s'exécutent sur un site. Les transactions globales composées de deux (ou trois) agents s'exécutent sur deux (ou trois) sites selon un schéma d'exécution procédural (figure 4.25).

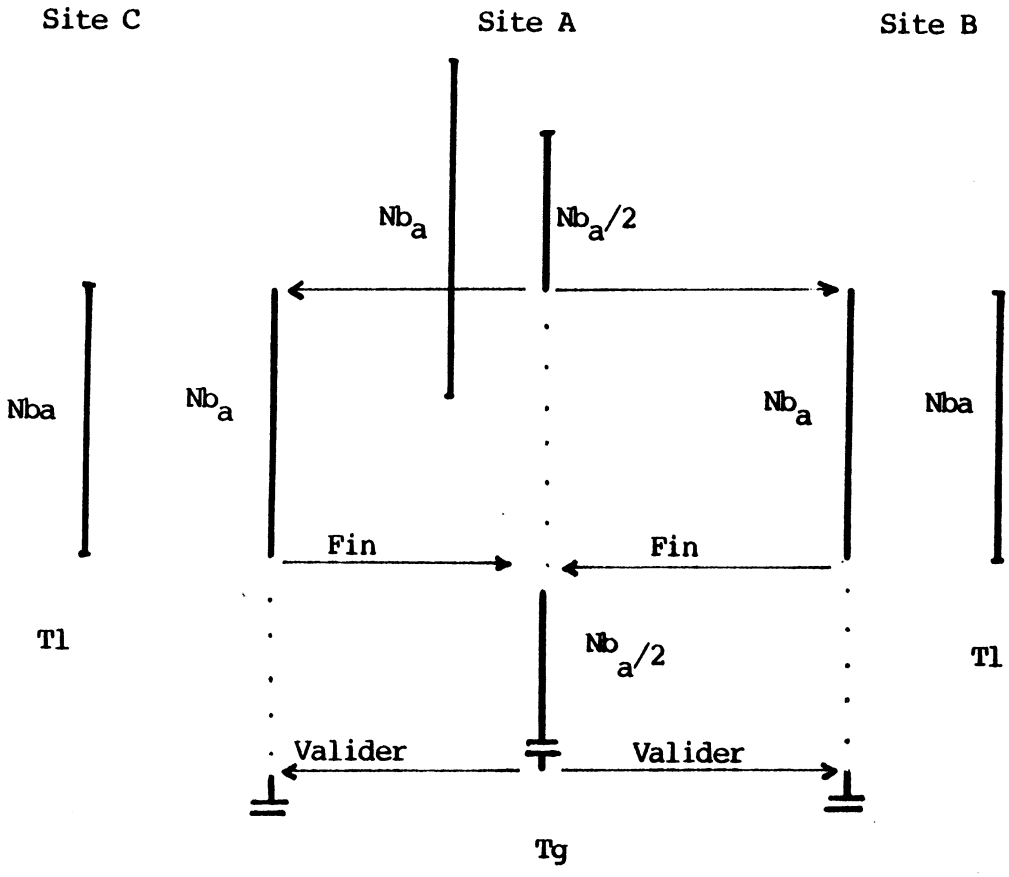


Figure 4.25

Dans chaque classe, on considère des populations homogènes de transactions ayant les caractéristiques suivantes :

. Tous les agents requièrent le même nombre d'accès à la base locale (noté Nb_a).

. La distribution des temps entre deux accès est exponentielle (de moyenne T_a). Ce temps comprend le temps d'accès au disque et le temps de calcul jusqu'à l'accès suivant.

. Le mode de verrouillage est exclusif. On ne fait pas de différence entre les accès en consultation et les accès en modification.

c) Le système transactionnel :

Le temps de verrouillage (noté Tv) est constant. Le temps d'initialisation et de terminaison d'un agent est constant. Il est réparti sur le temps T_a de manipulation de chaque granule.

Grâce à la technique des écritures différées, le coût de l'abandon temporaire d'un agent est négligeable. Il n'y a donc pas de surcharge en entrée-sortie associée à cette opération.

Les transactions globales utilisent un protocole de validation à 1-phase.

Sur chaque système, le niveau de concurrence, c'est à dire le nombre de transactions s'exécutant en parallèle est constant. Une transaction (locale ou globale) qui se termine est réinjectée dans le système (le modèle est fermé). On note T_l le nombre de transactions locales et T_g le nombre de transactions globales s'exécutant sur un site.

Les pannes ou erreurs ne sont pas considérées.

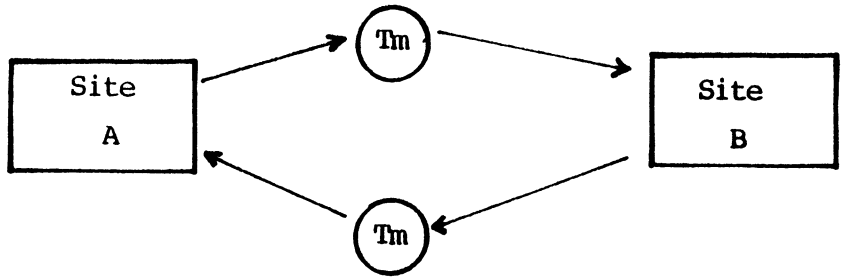
d) Le système d'exploitation :

Le délai de réflexion, le temps de soumission d'une transaction et la gestion des ressources physiques ne sont pas pris en compte dans les modèles.

e) Le réseau de communication :

Seul le cas des réseaux "longue distance" a été envisagé. On suppose que les transmissions ont lieu en mode "full-duplex". Pour cette raison, chaque liaison entre deux sites est modélisée par deux stations (une pour chaque sens) (Figure 4.26).

Station AB



Station BA

Figure 4.26

Modélisation du réseau

Le temps de transmission d'un message, c'est à dire le temps de service de la station (noté Tm) est constant. Ce temps comprend le traitement du message en émission, en réception et le délai de transmission proprement dit. On ne fait pas de différence entre des messages longs et courts. On considère deux types de stations pour représenter les lignes de communication :

. Ligne simple : la ligne est occupée pendant le temps de transmission d'un message (la station correspondante comporte une file d'attente). Cette technique permet de simuler simplement les phénomènes de congestion du réseau. Le temps total de transfert d'un message est la somme du temps de transmission Tm (donnée du modèle) et du temps d'attente (résultat de la simulation).

. Ligne multiplexe : les messages sont expédiés en "quasi-parallélisme". Il n'y a pas d'attente pour obtenir le service du réseau. Cette technique permet d'analyser les propriétés des algorithmes indépendamment de la charge du réseau. Le temps de transfert d'un message est égal au temps de transmission.

3.2.2. Les paramètres du modèle

Les paramètres du modèle sont les suivants :

- Taille des bases (N) : 500, 1000, 2000 granules

- Nombre de transactions : il y a le même nombre de transactions locales et globales.

  T = Tl + Tg : 6, 10, 20, 30, 40, 50

- Nombre d'accès par agent (Nb_a) : 5, 10, 20

  L'agent initial d'une transaction globale effectue ces accès en deux temps :

  - $Nb_a/2$ avant initialisation de l'agent sur le site distant,

  - $Nb_a/2$ après retour de cet agent.

- Temps entre 2 accès (T_a) : 100 millisecondes (moyenne du service exponentiel).

- Temps de verrouillage (T_v) : 10 millisecondes (constant)

- Temps de transmission d'un message (T_m) : 200, 250, 300 millisecondes (constant). Ce temps correspond aux observations effectuées sur TRANSPAC (soit 3 à 5 messages par seconde).

- Temps de simulation - Convergence des résultats : la durée des expériences a été calculée de telle sorte que l'écart relatif sur les principales quantités mesurées soit inférieur à 5%. Selon les expériences, cela a conduit à un temps réel de simulation de 2 à 30 minutes (temps UC sur Multics) pour un temps simulé de 200 à 500 secondes.

3.2.3. Quantités mesurées

- Temps de réponse d'une transaction : on s'intéresse à la fois à la valeur moyenne (notée R_l pour les transactions locales et R_g pour les transactions globales) et à la variation autour de la moyenne (écart-type).

- Débit du système (noté D) : exprimé en nombre de transactions terminées par unité de temps. D_l et D_g désignent le débit ramené aux seules transactions locales ou globales.

- Attente sur ressource indisponible : chaque fois qu'un granule est indisponible, un agent d'une transaction attend la terminaison d'une transaction concurrente. W désigne le temps d'attente correspondant.

- Reprises après abandon : Ab désigne pour une transaction le nombre moyen d'abandons dûs au contrôle de l'accès concurrent avant une exécution complète.

Le temps de réponse d'une transaction donnée s'exprime à l'aide de la formule suivante :

$$R = M + \sum_{i=1}^{Nb_a} (W_i + C_i) + \sum_{k=1}^{Ab} \left[M_k + \sum_{i=1}^{Nb_k} (W_i + C_i) + A_k \right]$$

où :

- Nba et Ab ont la signification indiquée ci-dessus.

- M est le temps de transfert des messages nécessaires à l'exécution de la transaction. Dans notre modèle :

  . M = 0 pour les transactions locales

  . M = 2 * Tm + Wm pour les transactions globales où Tm est le temps de transmission pur, Wm est le temps d'attente sur une ligne (Wm = 0 pour une ligne multiple).

- W_i est le temps d'attente pour le ième granule.

- C_i est le temps de calcul pour le ième granule. C_i est calculé à partir de Ta et de Tv.

- k désigne la kième tentative se soldant par un abandon.

- M_k est le temps de transfert des messages dans la kième tentative. Il s'agit à la fois des messages de l'application et des messages d'abandon (M_k = 0 pour les transactions locales).

- Nb_k est le nombre de granules accédés dans la kième tentative avant abandon de la transaction.

- A_k est le temps d'attente après le kième abandon et avant le démarrage de la tentative suivante. En effet, certains algorithmes introduisent une temporisation avant la réexécution d'une transaction consécutive à un abandon.

Tm, Nb_a, C_i sont des paramètres d'entrée du modèle calculés à partir des données décrites en 3.2.2.

Wm, W_i, Ab, M_k, A_k sont des quantités mesurées.

Le temps de réponse R peut s'exprimer également de la manière suivante :

$R = E + W + P$ où

. E designe le **temps d'exécution** proprement dit de la transaction (cette valeur est indépendante de l'algorithme de contrôle).

$$E = M + \sum_{i=1}^{Nb_a} C_i$$

. W designe la **somme des temps d'attente** dûs aux conflits pendant la phase d'exécution finale (sans abandon).

$$W = \sum_{i=1}^{Nb_a} W_i$$

. P désigne le **temps perdu à cause des abandons.**

$$P = \sum_{k=1}^{Ab} \left[M_k + \sum_{i=1}^{Nb_k} (W_i + C_i) + A_k \right]$$

Ce temps inclut :

- Le temps réel d'exécution des tentatives infructueuses (y compris les attentes dues aux conflits dans ces tentatives).

- Le temps d'attente après chaque tentative avant la réexécution suivante.

L'expression W + P constitue "l'overhead" dû au contrôle de l'accès concurrent.

3.3. Gestion des interblocages : détection ou prevention

Cette première série d'expérimentations concerne les systèmes d'information centralisés. Historiquement, l'objectif de cette étude consistait à obtenir des informations sur le comportement des mécanismes de prévention des interblocages, en particulier en les situant par rapport aux mécanismes de détection. L'absence de réseau se traduit par la simplification des quantités R, E, W et P, dans lesquelles les paramètres M et Mk ont disparu.

3.3.1. Détection des interblocages

La figure 4.27 exprime le débit D du système et le temps de réponse moyen R d'une transaction en fonction du niveau de concurrence.

Il existe un débit maximum obtenu pour Topt. transactions s'exécutant en parallèle. Au-delà, le système s'écroule avec, pour conséquence, une augmentation très rapide du temps de réponse moyen. Pour les valeurs du degré de concurrence supérieures à Topt, la dispersion des temps de réponse autour de la moyenne devient très forte (on observe des rapports supérieurs à 10 entre les transactions les plus rapides et les plus lentes).

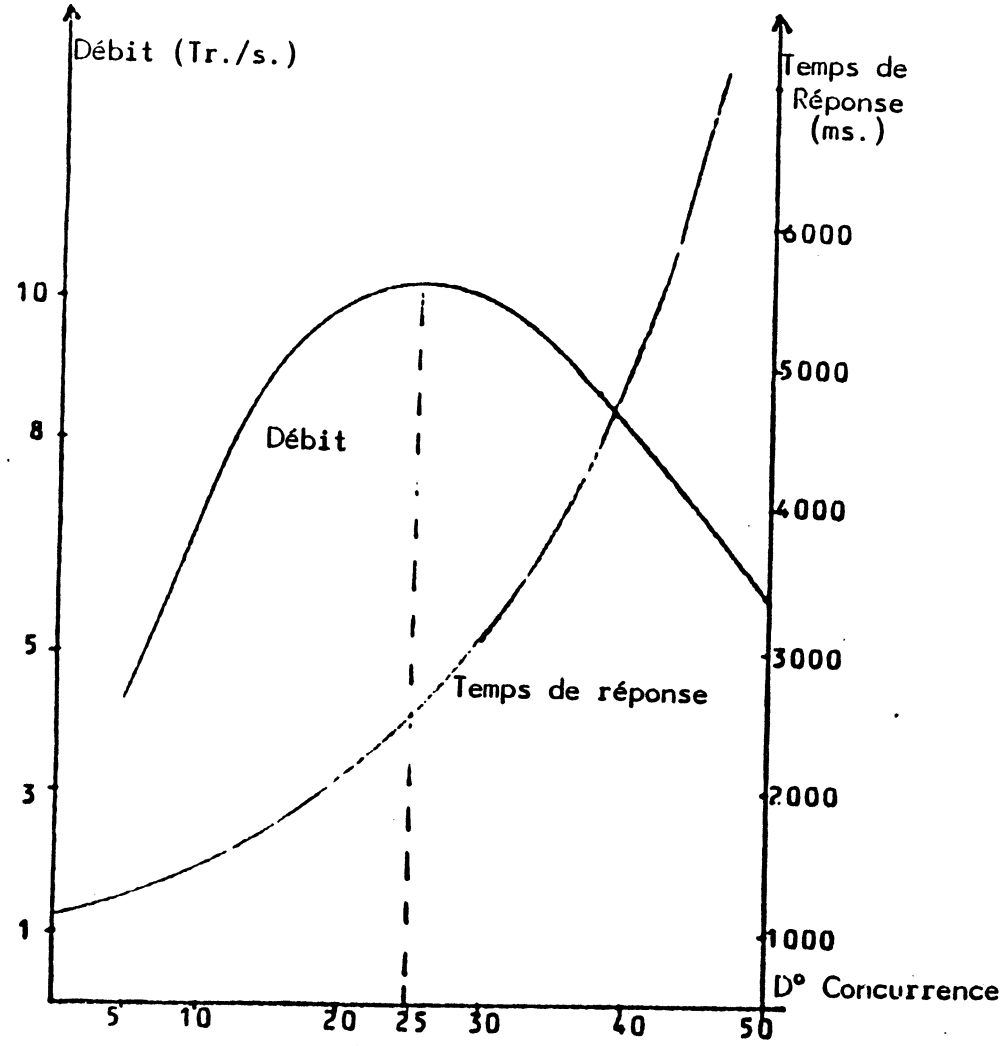


Figure 4.27

Remarques : En situation d'interblocage, deux problèmes se posent :

a) Quelle transaction doit être abandonnée ? Les critères de choix suivants sont possibles :

- l'une quelconque dans le cycle apparu dans le graphe des attentes,

- celle qui a provoqué l'interblocage,

- la plus récente (on suppose ici que les transactions sont datées),

- celle qui possède le moins de ressources (dans ce cas, le travail à refaire est minimal).

Les expériences réalisées montrent que la dernière stratégie est la meilleure. Notons cependant qu'une politique basée sur l'âge constitue également une bonne approximation de cet optimum.

b) La réexécution de la transaction est-elle immédiate ou différée ?

Différer la reprise de la transaction abandonnée jusqu'à la fin de la transaction concurrente donne les meilleurs résultats. En effet, la réexécution immédiate de la transaction risque de reprovoquer un conflit avec la transaction concurrente.

La figure 4.28 montre l'évolution respective du temps d'attente W et du temps perdu P avec le degré de concurrence.

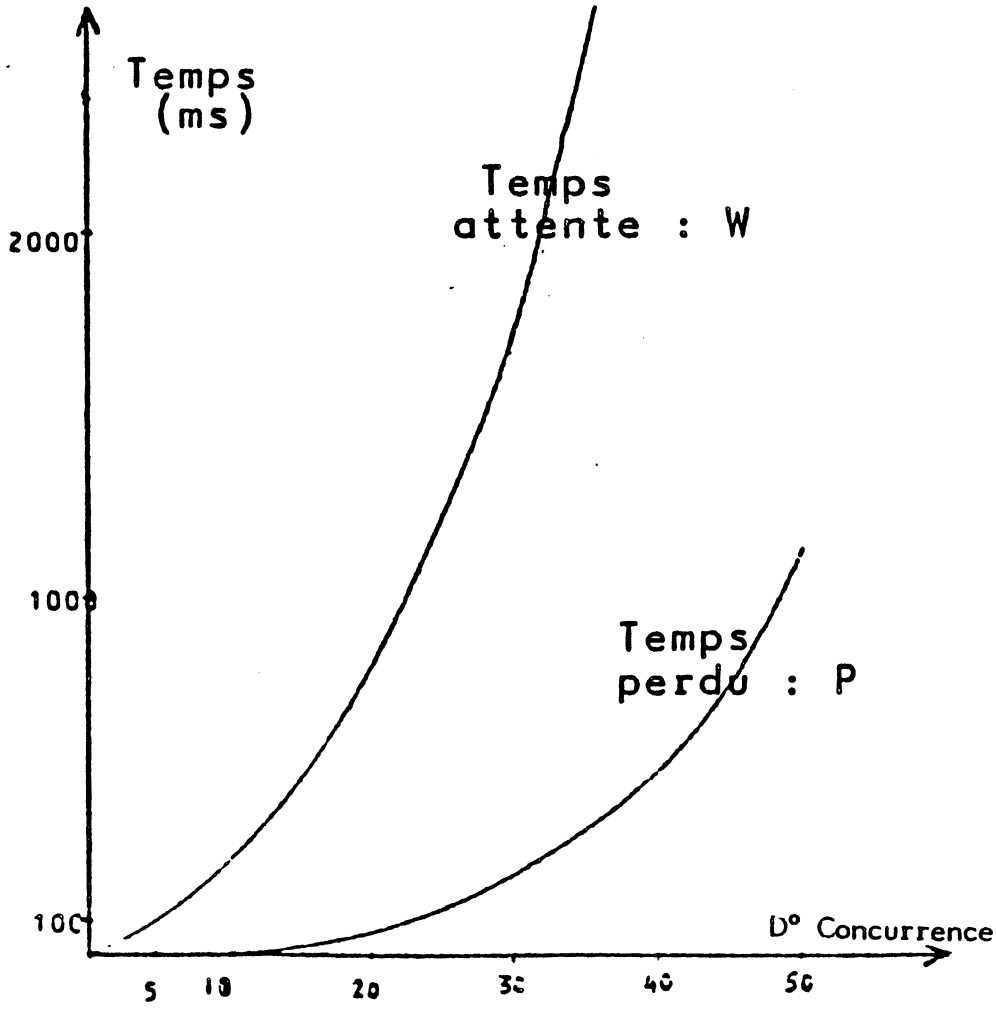


Figure 4.28

Le délai dû aux attentes simples est très supérieur au temps perdu à cause des abandons même pour les valeurs élevées du degré de concurrence pour lesquelles le nombre d'inter-blocages devient significatif. Cette remarque montre à l'évidence que l'impact des interblocages sur la performance du système n'est pas critique. Le facteur déterminant pour la performance est le temps d'attente en absence d'interblocage.

L'augmentation non contrôlée du degré de concurrence crée des configurations analogues au "Thrashing" dans les systèmes paginés. Par analogie avec les mécanismes de régulation des systèmes paginés, la solution consiste à limiter le degré de concurrence. Certains systèmes (TDS, CICS, ...) offrent la possibilité de spécifier à la génération un degré de concurrence maximum. Lorsqu'il est atteint, l'entrée d'une nouvelle transaction dans le système est refusée. Nous préférons un mécanisme de régulation plus souple, basé sur l'utilisation effective des ressources par les transactions concurrentes.

Une étude détaillée des phénomènes d'attente montre que la cause principale de dégradation des performances est la longueur des files d'attente. Dans une file donnée chaque transaction attend la terminaison successive des transactions placées devant elle. Pendant toute cette attente, les ressources déjà acquises par la transaction sont bloquées et donc inaccessibles aux transactions concurrentes.

L'algorithme suivant, exécuté pour chaque conflit, permet de limiter la propagation de ces ressources "dormantes".

> Une transaction est autorisée à attendre si et seulement si la longueur d'un chemin dans le graphe des attentes est inférieure à une limite préfixée.

Ce mécanisme, très simple à mettre en oeuvre, induit une régulation naturelle du degré de concurrence en n'autorisant à s'exécuter en parallèle que des transactions qui ne sont pas en conflit.

La figure 4.29 montre l'évolution du débit du système avec ou sans régulation. On remarquera que le mécanisme ne présente d'intérêt réel que pour les fortes charges. Il est légèrement pénalisant lorsque les charges sont faibles (perturbation inférieure à 10 %).

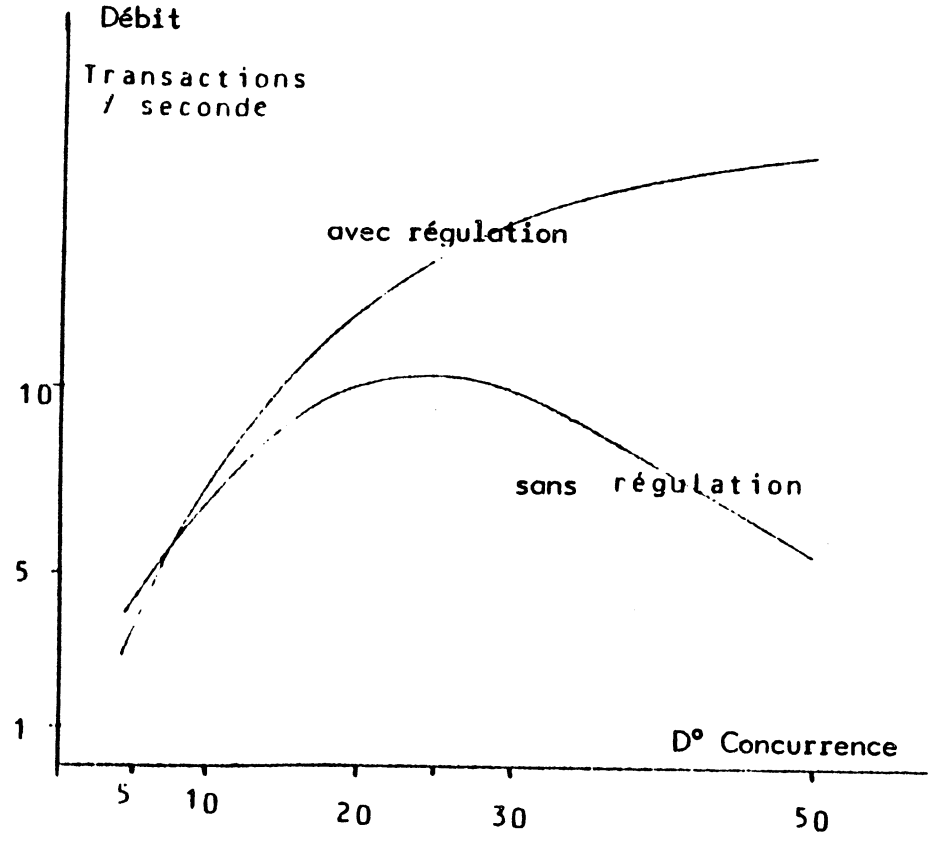


Figure 4.29

3.3.2. Prévention des interblocages

Les algorithmes qui ont été testés sont ceux qui sont décrits dans l'annexe A1 et qui sont référencés dans la suite sous l'appellation "WAIT-DIE", "DIE-WAIT" et "WOUND-WAIT".

La figure 4.30 décrit le temps de réponse moyen et le débit du système pour l'algorithme WAIT-DIE. La figure 4.31 représente l'évolution respective du temps d'attente W et du temps perdu P.

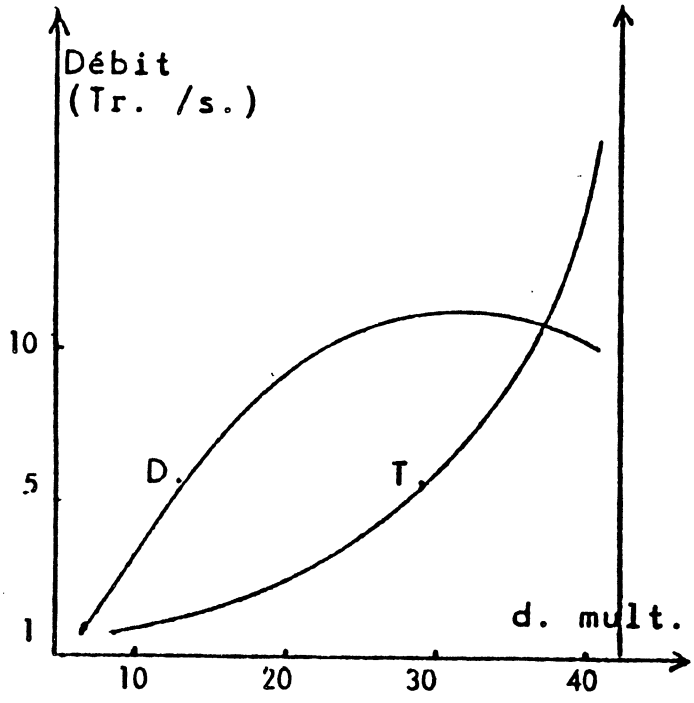


Figure 4.30

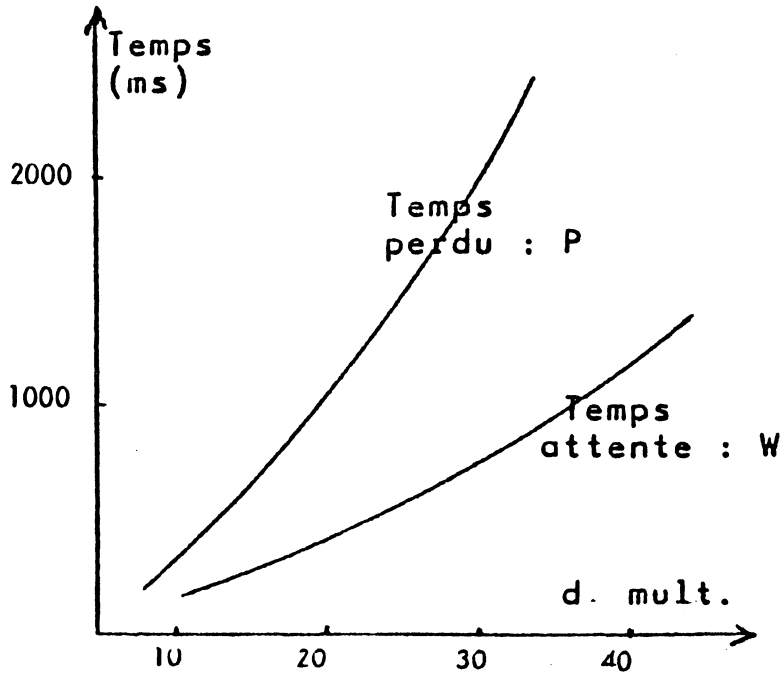


Figure 4.31

Ces courbes sont à comparer avec celles des figures 4.27 et 4.28. L'aspect général des courbes est le même : on retrouve en particulier le phénomène d'écroulement du système au-delà d'un certain niveau de concurrence. On notera cependant que la situation représentée dans les figures 4.28 et 4.31 est inversée : l'effondrement du système est maintenant causé par le nombre important d'abandons. Cette étude confirme un résultat intuitif, à savoir qu'un mécanisme qui provoque trop d'abandons détériore les performances du système.

Les figures 4.32 et 4.33 décrivent le temps de réponse moyen et le débit du système pour les trois algorithmes précédemment cités.

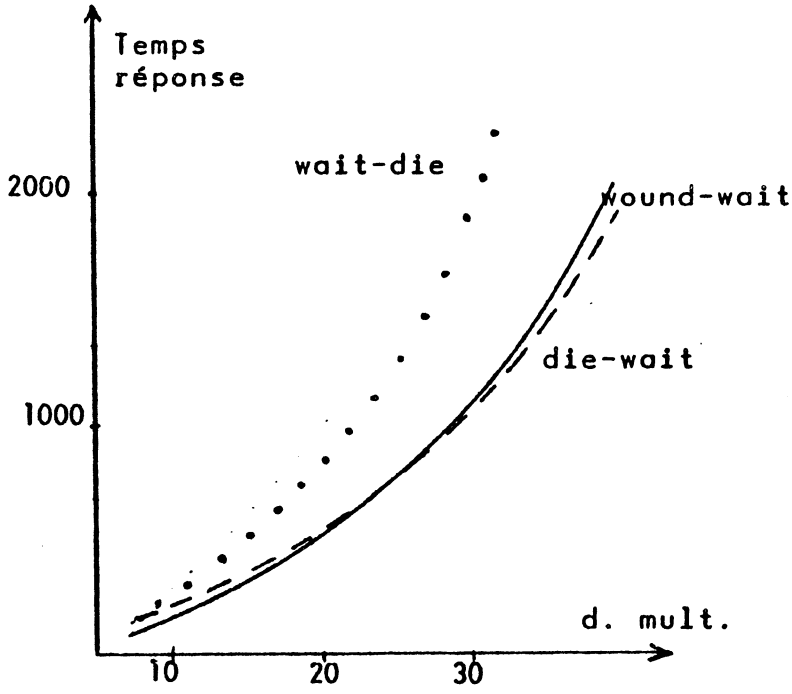


Figure 4.32

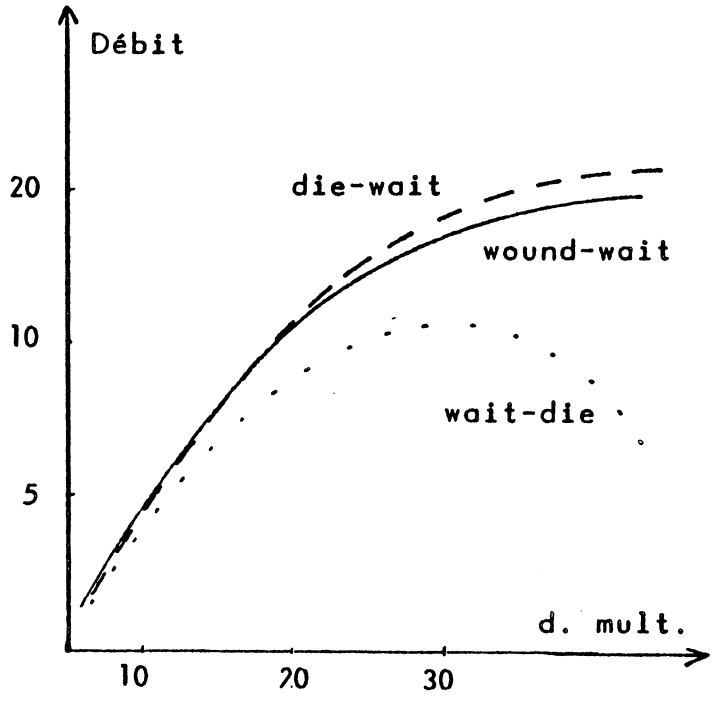


Figure 4.33

"DIE-WAIT" et "WOUND-WAIT" évitent l'écroulement du système en introduisant une régulation naturelle du degré de concurrence par le biais des abandons. Cependant, par rapport à "WAIT-DIE", le nombre d'abandons reste à un niveau raisonnable pour ne pas provoquer l'effet inverse.

Les algorithmes "DIE-WAIT" et "WOUND-WAIT" sont assez proches l'un de l'autre en performance. L'amélioration espérée qui consiste à "blesser" une transaction plutôt que de la "tuer" apporte effectivement un gain pour les faibles charges. Pour les charges fortes, le résultat est inversé.

La distribution du temps de réponse, non représentée ici, montre une grande dispersion pour "WAIT-DIE". Le meilleur centrage est obtenu pour "DIE-WAIT".

3.3.3. Conclusion : détection ou prévention ?

La figure 4.34 résume les observations précédentes ramenées à un critère de base : le débit du système.

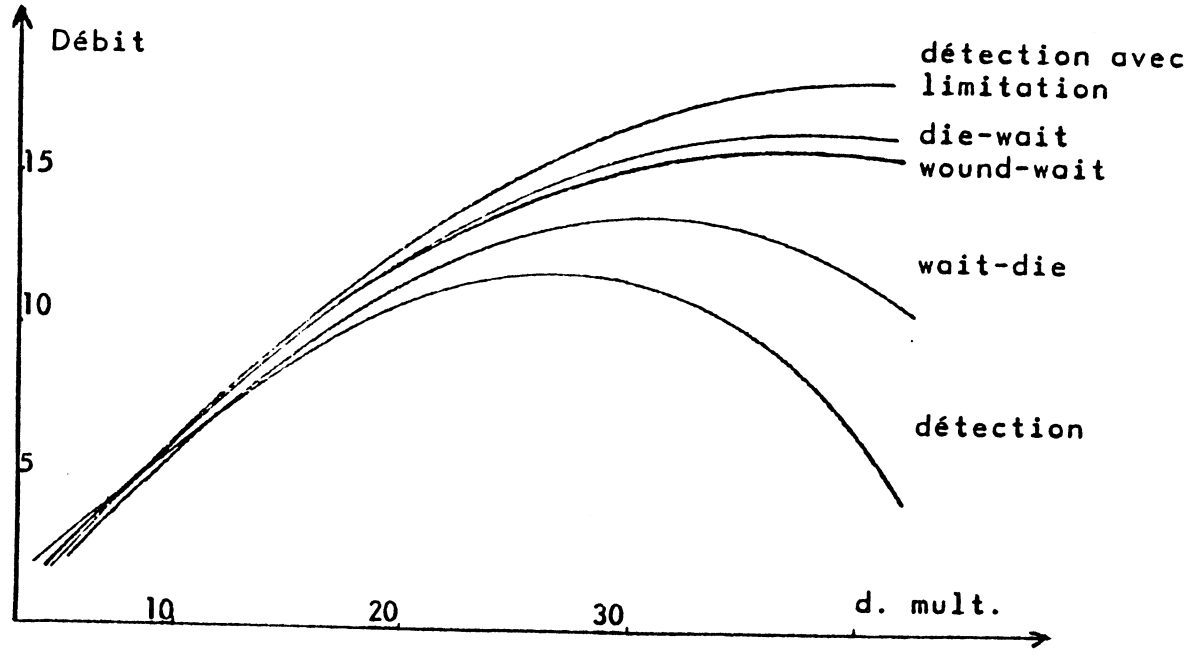


Figure 4.34

Lorsque le système est faiblement chargé (niveau de concurrence bas), l'algorithme classique de détection est meilleur. On notera cependant que les différences de performance ne sont pas significatives. En effet, dans cette zone où le nombre de conflits est faible, l'impact de l'algorithme n'est pas significatif.

Lorsque le taux de conflit devient élévé en raison d'une augmentation de la charge, le temps d'attente croît plus vite que le temps perdu. La dégradation des performances est due à la formation de longues chaînes d'attente. Dans ce cas, l'impact d'une régulation effective (limitation des chaînes d'attente) ou d'une auto-régulation (abandons provoqués par l'algorithme de contrôle) est prépondérant.

Cette supériorité des algorithmes de prévention sur ceux de détection a été depuis lors confirmée par d'autres études réalisées, soit à partir de modèles de simulation [Devor 82, Tay 84], soit d'après des modèles mathématiques [Batchelor 82].

Le bilan de cette première étape, consacrée à l'étude du comportement des techniques de verrouillage pour les systèmes centralisés peut être résumé comme suit :

- L'idée "a priori" qui consistait à préférer les techniques de détection parce qu'elles minimisent le nombre d'abandons se révèle sans fondement.

- Les abandons "à tort" provoqués par les techniques de prévention ont un impact bénéfique tant que leur nombre reste dans les "limites raisonnables".

- Les algorithmes de prévention, conçus pour un environnement distribué, s'appliquent également aux systèmes centralisés.

En conclusion, plutôt que d'entretenir la polémique sur les avantages respectifs de la détection ou de la prévention, nous plaidons en faveur d'un algorithme qui permette une régulation du degré de concurrence du système, quelle que soit la méthode adoptée pour gérer les interblocages. Ces résultats, obtenus dans le contexte d'un système d'information centralisé, ont amené les concepteurs à modifier les systèmes existants, soit en substituant la prévention à la détection, soit en complétant la détection par un mécanisme de régulation tel que celui que nous avons présenté.

3.4. Quel algorithme de prévention en environnement réparti ?

Dans un contexte distribué, le choix d'un algorithme de contrôle de l'accès concurrent fondé sur le verrouillage et la prévention des interblocages est justifié par l'argumentation suivante :

- L'évolution vers des configurations réparties est notoirement plus simple pour les méthodes de prévention qui nécessitent des synchronisations moins complexes (cf. chapitre 3 §3.3).

- Nous disposons de peu d'information sur le comportement chiffré des méthodes de détection dans un système réparti. Cependant, les résultats observés en centralisé (voir paragraphe précédent) et les estimations disponibles dans la littérature [Obermarck 82, Wazdi 83] ne laissent pas espérer des résultats remarquables. Néanmoins, cette impression demanderait à être vérifiée par une campagne de mesures.

Ayant éliminé les techniques de détection des interblocages, il nous reste à déterminer quel algorithme de prévention constitue le meilleur choix pour un système réparti. Si on se réfère aux résultats du paragraphe précédent, concernant les performances comparées des différents algorithmes en centralisé, on observe que, d'une part l'algorithme WAIT-DIE doit être éliminé en raison de ses performances médiocres, et

d'autre part que les différences de comportement entre DIE-WAIT et WOUND-WAIT ne sont pas significatives. Des expériences complémentaires, prenant en compte la distribution, sont nécessaires pour déterminer le meilleur choix.

Les courbes des figures 4.35 et 4.36 montrent l'évolution respective du temps de réponse moyen d'une transaction globale Rg et du débit du système Dg (exprimé en transactions globales terminées par unité de temps). L'allure générale des courbes portant sur les mesures propres aux transactions locales est similaire.

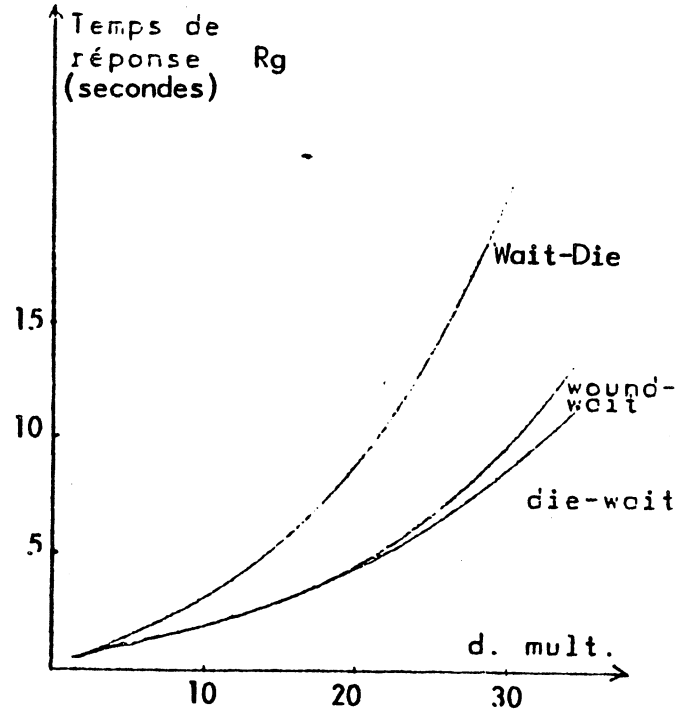


Figure 4.35

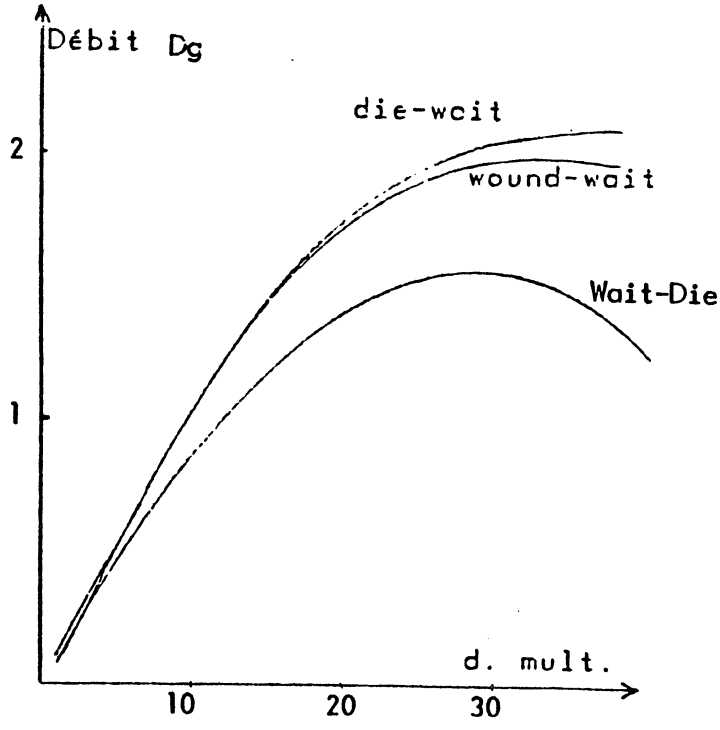


Figure 4.36

DIE-WAIT et WOUND-WAIT sont très proches en performance avec cependant un petit avantage à DIE-WAIT pour les fortes charges. On retrouve donc un résultat analogue à celui des systèmes centralisés. Le gain escompté en retardant l'abandon des transactions est contrebalancé par un accroissement du temps d'attente engendré par une augmentation du taux de conflit (le degré de concurrence est plus élevé), et du nombre de messages (messages de blessure). La dispersion des temps de réponse autour de la moyenne est minimale pour DIE-WAIT.

Des études complémentaires ont permis de mesurer l'impact du réseau, de la procédure de validation et de la stratégie de reprise d'une transaction après abandon. Les éléments correspondants sont décrits en détail dans [Scot 83] et sont résumés ci-dessous :

- L'augmentation du temps de transmission des messages, engendrée par une vitesse de ligne inférieure ou par un engorgement du réseau (utilisation de lignes avec files d'attente), a le même effet sur le temps de réponse et le débit du système qu'une augmentation du degré de concurrence. La position relative des deux algorithmes n'en est pas affectée.

- La prise en compte d'une procédure de validation à deux phases se traduit par une augmentation du temps de rétention des ressources, comparable à celui qui est introduit par une vitesse de ligne plus faible ou par un réseau congestionné. Les conséquences sont identiques.

- Le problème se pose de savoir, en cas d'abandon, s'il convient de réexécuter la victime immédiatement ou s'il est préférable de retarder cette exécution pour ne pas risquer de reproduire le même conflit. Les expériences réalisées conduisent aux conclusions suivantes. Les transactions locales doivent être retardées jusqu'à la terminaison de la transaction concurrente. Les transactions globales peuvent être réexécutées immédiatement. En effet, le délai inhérent au protocole d'abandon et au redémarrage de la transaction globale est, la plupart du temps, suffisant pour assurer la terminaison de la transaction concurrente.

En complément, le lecteur trouvera dans [Scot 83] une analyse de la validité des hypothèses de modélisation et de leur impact sur les mesures effectuées.

3.1 SYNTHESE : Eléments d'un noyau transactionnel pour les systèmes d'information répartis

Dans cette synthèse, on considère des systèmes d'information répartis construits sur le modèle transactionnel coopérant. Nous avons vu que la structure imposée par ce modèle est suffisamment générale pour s'appliquer à une large variété de systèmes d'information répartis. En outre, dans la perspective d'un usage aussi universel que possible, on considère comme acquis la nécessité de séparer clairement les mécanismes de base des stratégies d'utilisation. L'objectif visé est la définition d'un noyau de système distribué dédié aux applications transactionnelles, sur lequel il soit possible de construire diverses politiques de contrôle de la cohérence en fonction des applications. La définition des mécanismes du noyau découle principalement de l'expérience acquise dans le projet SCOT, mais prend également en compte les résultats d'autres recherches, menées en particulier dans le cadre du projet pilote SIRIUS. Nous listons ci-dessous les contraintes fondamentales du cahier des charges associé à la spécification de ce noyau :

- Les mécanismes du noyau doivent permettre de définir un protocole qui garantit la cohérence forte (y compris pour des applications présentant des scénarios d'exécution complexes si celà s'avère nécessaire : parallélisme entre agents par exemple).

- On peut influer sur les mécanismes de contrôle pour adapter leur comportement aux contraintes de l'application (cas de cohérence dégradée). Celà signifie que le contrôle de la cohérence est visible du programmeur d'application.

- Le noyau est facile à mettre en oeuvre. La complexité concerne trois aspects : les algorithmes exécutés sur chaque site, les synchronisations entre les sites (sur ce point, nous avons lourdement insisté sur le rôle de l'autonomie) et, enfin la possibilité d'évolution d'un système centralisé vers une configuration répartie.

- Il autorise des performances "acceptables" en fonctionnement normal. Nous écarterons les mécanismes qui seraient optimaux pour un environnement particulier mais qui se révéleraient trop pénalisants dans un contexte différent. De même, nous avons vu qu'à fiabilité égale, il est possible d'optimiser soit le fonctionnement en régime permanent,soit la reprise après panne ou erreur; il est difficile de concilier les deux. Partant de l'hypothèse que le dysfonctionnement constitue l'exception, nous donnerons la préférence à une technique qui privilégie le mode normal au détriment de la reprise après panne.

Parmi les nombreuses solutions analysées dans ce chapitre, nous sélectionnons une combinaison de plusieurs mécanismes, répondant aux critères du cahier des charges, et qui pourraient constituer les fondements du noyau transactionnel recherché. Cette synthèse est présentée en deux étapes. Dans un premier temps, nous nous attacherons exclusivement à la sélection d'un protocole de validation-reprise correspondant aux objectifs indiqués. La seconde phase consistera à déterminer un protocole de contrôle de l'accès concurrent compatible avec le choix précédent. L'ordre dans lequel nous procédons à cette analyse n'est pas indifférent. Il correspond à une hiérarchie dans l'importance relative que nous attribuons à ces deux facettes du contrôle de la cohérence.

Le protocole de validation mixte, défini en 2.3.6 est une illustration de la démarche suivie. D'une part, il permet la coexistence de plusieurs mécanismes adaptés à des niveaux de cohérence différents. D'autre part, la possibilité de spécifier dynamiquement le type de

protocole à appliquer (à l'aide d'un paramètre de la primitive FIN) rend possible le choix d'une stratégie au niveau de la programmation de l'application.

Une certaine souplesse dans la définition d'une politique de gestion de ressources impose de pouvoir choisir le supérieur de la validation selon des critères propres à chaque application. Les mécanismes qui rendent possibles l'expression de ce choix ont été décrits en 2.2. Dans le cas des réseaux "longue distance", on rappelle qu'ils reposent sur la désignation dynamique du supérieur à l'aide d'un paramètre de la primitive FIN-A, et sur le contrôle "hiérarchisé" pour l'acheminement des messages de terminaison. La même analyse a montré, qu'en toutes circonstances, le contrôle "direct" était la technique la mieux adaptée à la transmission des messages de validation et d'abandon.

Pour répondre au point quatre du cahier des charges, nous préconisons une technique de synchronisation des étapes de validation ou d'invalidation exempte d'accusés de réception. Nous avons vu que cette technique optimise le fonctionnement en mode normal. Dans ce cas, le protocole de reprise est contrôlé par les agents inférieurs et utilise la procédure d'enquête (cf. § 2.4).

En ce qui concerne le contrôle de l'accès concurrent, le premier argument que nous avons avancé en faveur du verrouillage à deux phases concerne la facilité d'évolution des applications existantes vers des configurations réparties. En outre, si on se réfère aux critères du cahier des charges, ce choix nous paraît a posteriori pleinement justifié à la lumière des résultats présentés dans les chapitres précédents. Par rapport aux autres méthodes, un facteur discriminant est la possibilité de maîtriser la gestion des objets partagés, autorisant ainsi la réalisation de stratégies d'allocation et de libération différentes selon les applications. Cette flexibilité repose sur la gestion simultanée de plusieurs niveaux de granularité, sur l'utilisation du verrouillage explicite et sur la libération anticipée des objets.

Le critère de simplicité de mise en oeuvre, qui prend en compte la complexité de l'algorithme et la nature des synchronisations, nous a amené à éliminer la détection des interblocages au profit d'un mécanisme de prévention. En dernier lieu, le choix de la technique "DIE-WAIT" résulte à la fois de considérations liées à la simplicité de réalisation et à la performance.

Le lecteur ne doit pas conclure de cette analyse que les techniques que nous avons exclues du noyau ainsi défini sont irrémédiablement condamnées. Il existe des applications spécifiques qui s'accommodent bien de ces méthodes. Ainsi, la technique des versions multiples, qui est un cas particulier de l'estampillage, est bien adaptée à la gestion d'historiques d'un objet, manipulés par exemple dans les applications de bureautique ou de génie logiciel. De même, la certification peut donner d'excellents résultats dans des applications où les transactions en consultation sont prépondérantes (recherche documentaire par exemple).

Un prototype de ce noyau transactionnel a été réalisé sur plusieurs machines Mini-6 interconnectées par le réseau TRANSPAC selon des protocoles compatibles OSI. Ce noyau permet la coopération entre systèmes transactionnels conçus pour cette machine. Il a servi de support pour la réalisation d'une application bancaire présentant une large gamme de scénarios d'exécution caractérisés par des exigences diverses en matière de cohérence.

4. SYSTEME D'INFORMATION A HAUTE DISPONIBILITE

L'évaluation du coût d'une panne (cf. chapitre 2 § 2.5) a mis en évidence un paramètre clé, à savoir la période d'indisponibilité du système jusqu'à la détection et la guérison de la défaillance. Selon la gravité de la panne, l'interruption de fonctionnement peut être plus ou moins longue et devenir incompatible avec la qualité du service attendu. Comme pour le maintien de la cohérence, les exigences en matière de tolérance aux pannes varient considérablement selon les classes d'applications. Cette section est consacrée à l'étude des techniques mises en oeuvre pour raccourcir le délai de redémarrage d'un système affecté par un incident. Dans ce domaine, l'objectif de "haute disponibilité" s'applique aux systèmes qui ne présentent aucune interruption de service apparente en présence d'une panne simple (c'est à dire une panne affectant un seul élément du système). Le principe de ces architectures appliquées aux systèmes d'information repose sur la redondance d'une part, et sur l'utilisation du concept de transaction d'autre part.

4.1. Redondance et disponibilité

On distingue deux graduations dans l'indisponibilité selon que la défaillance affecte directement l'information (panne d'une mémoire secondaire) ou l'exécution des programmes d'applications considérés comme opérateurs d'accès aux données (panne du système). Nous avons vu au chapitre 2 que le premier cas, d'une part est peu fréquent, et d'autre part peut être résolu de diverses manières (sauvegardes périodiques, doubles écritures, disques miroirs). Nous ne reviendrons donc pas sur ce problème et nous nous concentrerons sur les techniques à mettre en oeuvre pour minimiser les conséquences d'une défaillance du système de traitement, qu'elle soit d'origine matérielle ou logicielle.

Le partitionnement des données en sous-ensembles disjoints, sous la responsabilité d'un système de traitement distinct, est un premier moyen de limiter les conséquences d'une défaillance à un sous-ensemble seulement de l'information. Cette propriété est souvent présentée comme un argument en faveur des systèmes distribués. Malgré le progrès apporté par une configuration de ce type, une partie de l'information reste indisponible jusqu'à la réparation de la défaillance. L'objectif de disponibilité totale requiert une certaine forme de redondance. Cette redondance peut être mise en oeuvre de deux manières selon qu'elle s'applique à l'information ou aux programmes d'application.

La première technique est le domaine des bases de données dupliquées, évoquées à plusieurs reprises dans cet ouvrage. L'avantage généralement avancé en faveur de cette méthode est qu'elle ne nécessite qu'un investissement matériel mineur (doublement de l'espace disque). Un autre argument favorable concerne la souplesse de mise en oeuvre ; il est possible en effet d'appliquer la duplication seulement à la partie sensible de l'information. A côté de ces aspects positifs, cette approche présente deux inconvénients majeurs :

- Le contrôle du partage et la mise en cohérence des copies d'un même objet nécessitent des synchronisations complexes génératrices d'une dégradation du degré de parallélisme et du temps de réponse. Ce facteur est d'autant plus pénalisant qu'il intervient en fonctionnement normal, c'est à dire même en l'absence de panne.

- S'il est vrai qu'une défaillance d'un système n'a pas de conséquence directe sur les systèmes coopérants, en revanche la disponibilité n'est pas assurée pour les activités s'exécutant sur le système défaillant, à moins qu'elles ne disposent d'un moyen de s'exécuter sur un autre système, ce qui n'est généralement pas le cas.

L'alternative évoquée plus haut consiste à maintenir une seule copie de l'information et à appliquer le principe de la redondance aux traitements et aux chemins d'accès aux données.

Par rapport à la solution précédente, cette approche requiert une redondance au niveau matériel et par conséquence un investissement important. Le développement récent des sociétés proposant une offre de ce type (TANDEM, STRATUS, AURAGEN, ...) montre qu'il existe aujourd'hui un marché potentiel considérable, en particulier dans le domaine des applications bancaires et des systèmes de réservation de places [Smith 83].

Des deux formes de redondance qui viennent d'être évoquées, celle des chemins d'accès est la plus facile et la moins coûteuse à mettre en oeuvre. Cette option est d'ailleurs disponible sur des systèmes qui ne sont pas orientés vers la sécurité ou la disponibilité. Un support d'information permanente (un disque par exemple) est rattaché à deux contrôleurs indépendants, ces contrôleurs sont eux mêmes accessibles par deux voies différentes d'un ou plusieurs processeurs. Dans les réalisations les plus simples, on distingue un contrôleur primaire qui effectue les opérations et un contrôleur secondaire en "sommeil", qui prend le relai en cas de défaillance du premier. Les organisations plus sophistiquées exploitent l'activité parallèle des deux contrôleurs sur une grappe de disques, autorisant ainsi un plus grand débit d'entrées-sorties. Dans ce cas, la synchronisation des opérations sur un disque s'effectue soit au niveau du ou des processeurs, soit à celui des unités d'échange.

La redondance des traitements peut être conçue de diverses façons. Une première manière consiste à utiliser la technique mise en oeuvre dans les applications dites "temps réel" selon laquelle le même programme est exécuté par au moins deux processeurs différents. Si l'un tombe en panne le ou les suivants continuent le traitement. Il n'y a donc ni interruption de fonctionnement, ni même dégradation temporaire. Outre son coût exhorbitant, cette solution demeure sensible aux pannes logicielles (c'est à dire aux erreurs subsistant dans le système d'exploitation ou dans le système transactionnel). Or, il se trouve que, dans le contexte d'un système d'information complexe, cette source d'ennui est beaucoup plus fréquente que les pannes matérielles. Cette approche est utilisée dans le "Continuous Processing System" de la Société STRATUS Computer [Kastner 83].

Il existe des solutions moins brutales, et mieux adaptées au contexte particulier des systèmes d'information. Ces méthodes s'appuient sur les propriétés de la transaction pour limiter le coût de la redondance, tout en garantissant un fonctionnement en "service continu". Le principe de ces méthodes est détaillé dans le paragraphe suivant. En complément, le lecteur trouvera dans [Kim 84] une description de plusieurs réalisations opérationnelles fondées sur cette technique.

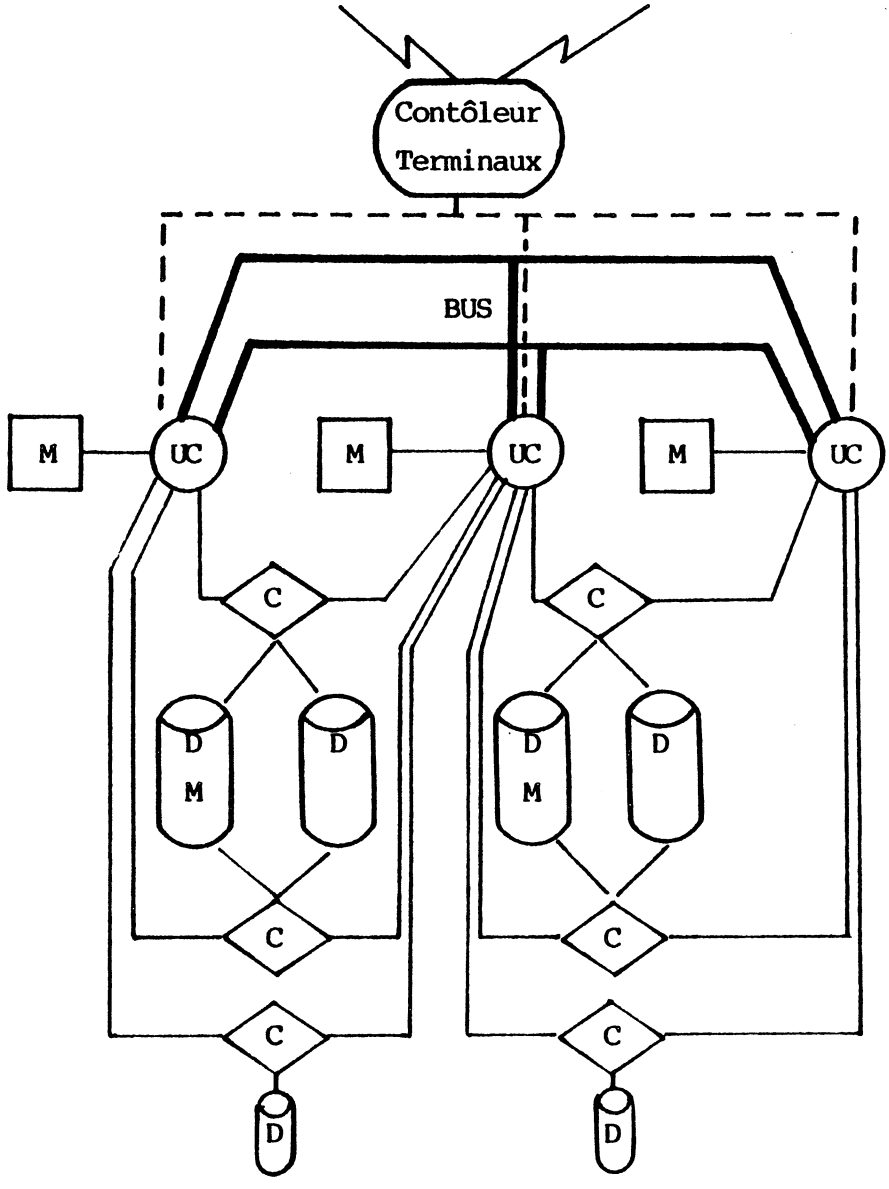
4.2. Système en service continu

Par système en "service continu" (on dit aussi "non-stop"), on désigne un système qui, vu de l'utilisateur, ne "tombe pas en panne", c'est à dire que le taux d'indisponibilité totale du système est extrêmement faible (de l'ordre de une à deux fois par an). L'occurrence d'une panne logicielle ou matérielle est masquée à l'utilisateur et se traduit seulement par une dégradation temporaire de la qualité du service (essentiellement en terme de temps de réponse).

Le service continu est obtenu grâce à la redondance du matériel et à l'association à chaque activité logicielle (dite "primaire") d'une activité logicielle, dite "de secours". Les activités primaire et de secours correspondant à une même fonction sont localisées sur des matériels différents.

L'activité de secours est en "sommeil" ; elle est simplement tenue informée des changements d'état essentiels de l'activité primaire qui lui est associée. En cas de défaillance du support de l'activité primaire, l'activité secondaire prend le relai, évitant ainsi une discontinuité de fonctionnement. La puissance de calcul nécessaire pour maintenir à jour l'activité de secours est faible de sorte que le processeur qui l'héberge peut consacrer l'essentiel de ses ressources à d'autres tâches. En retour, la synchronisation des processeurs supportant les activités couplées devient un élément critique de ce type d'architecture à la fois sur le plan de la fiabilité et sur celui de la performance. C'est pourquoi la majorité des systèmes présents sur

le marché utilisent une liaison à haut débit, elle même doublée pour des raisons de sécurité [Katzman 77, Bartlett 81]. Un exemple de ce type d'architecture redondante est représenté en figure 4.37.



UC : Unité Centrale
M : Mémoire
C : Contrôleur
DM : Disque Miroir (Journaux, système)
D : Données de l'application

Figure 4.37

Exemple d'architecture redondante

Cette architecture appartient à la classe des systèmes multiprocesseurs faiblement couplés, c'est-à-dire sans mémoire commune. Chaque processeur dispose de sa copie du système d'exploitation et du système transactionnel. Par rapport à une architecture fortement couplée, cette organisation présente deux avantages importants :

- elle diminue le risque d'occurrence de la même erreur logicielle sur les deux systèmes couplés du fait que les configurations correspondantes évoluent de façon disymétrique,

- elle permet une évolution plus souple du système. Ce point sera développé plus loin.

Le contrôleur de terminaux joue le rôle d'un aiguillage pour connecter dynamiquement les terminaux à l'un des processeurs, autorisant ainsi la reconfiguration en cas de défaillance de l'un d'entre eux.

Pour illustrer le principe de redondance des chemins d'accès, nous avons figuré deux formes possibles pour le double accès. Dans le premier cas, le même ensemble de disques est accessible depuis deux contrôleurs indépendants, eux-mêmes connectés à une paire de processeurs. Une configuration plus simple et plus économique se limite à un seul contrôleur pour une grappe de disques, connecté à deux processeurs. Enfin, le principe des "disques miroirs", évoqué au chapitre 2, peut être appliqué aux données critiques tels que les journaux, pour compléter la fiabilité de l'ensemble du système.

Le dispositif de détection des défaillances et de reconfiguration constitue un élément critique de ce type d'architecture. La description des mécanismes correspondants se situe en dehors des objectifs de ce document. Le lecteur trouvera dans [Kim 82, Kim 84] plusieurs exemples de réalisation dans ce domaine.

L'exécution en continu des applications utilise les propriétés de la transaction, considérée à la fois comme unité de cohérence et unité de reprise. A l'initialisation d'une transaction, une activité primaire

est créée sur un processeur et une activité de secours est créée sur l'autre processeur. Le choix du processeur est fonction de leur charge respective. L'activité primaire exécute le code de la transaction et communique à l'activité secondaire les informations suffisantes pour assurer, en cas de panne, la reprise de la transaction.

Dans une configuration totalement symétrique, les processeurs de traitement sont couplés deux à deux. Chaque processeur supporte à la fois des activités primaires et des activités de secours correspondant aux activités primaires du processeur dual. Cette double compétence est illustrée par le schéma de la figure 4.38.

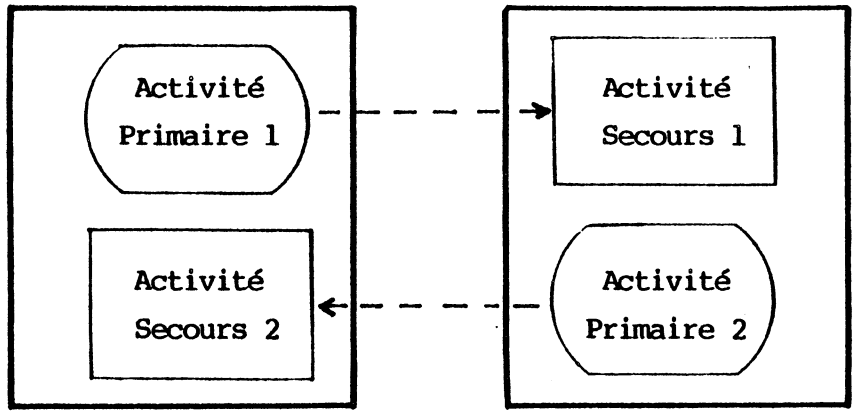


Figure 4.38

L'activité de secours maintient à jour une image mémoire du contexte d'état de la transaction. En cas de panne pendant l'étape de calcul, le contexte de l'activité de secours est donc dans un état qui correspond à l'état initial de la transaction : la reprise peut s'opérer immédiatement. En cas de panne pendant l'étape de validation (ou d'invalidation), le processeur de secours utilise directement l'information enregistrée par le processeur primaire dans le journal pour "refaire" l'étape interrompue (on rappelle que les disques sont accessibles depuis les deux processeurs de traitement). Le tableau suivant résume l'interprétation des actions d'une transaction dans le contexte d'un fonctionnement en service continu.

Action	Processeur primaire	Processeur de secours
DEBUT-T	. Initialisation espace de travail . Initialisation du journal . état = ACTIF	. Initialisation espace de travail . état = ACTIF
ECRIRE-D LIRE-D	. action enregistrée dans espace de travail	
FIN-T	. sauvegarde espace de travail sur journal . état = VALIDE . validation . état = TERMINE	. état = VALIDE . état = TERMINE
ARRET-T	. sauvegarde espace de travail sur journal . état = INVALIDE . invalidation . état = ABANDONNE	. état = INVALIDE . état = ABANDONNE
POINT-R	. enregistrement point de reprise sur journal	

On constate que l'activité du processeur de secours pour la transaction en cours d'exécution est très réduite. La puissance de traitement résiduelle peut donc être consacrée à d'autres tâches. La synchronisation entre l'activité primaire et l'activité de secours se limite à l'envoi d'un message sur occurrence de l'une des actions DEBUT-T, FIN-T et ARRET-T.

En cas de panne d'origine matérielle ou logicielle affectant un processeur, le système dual exécute, d'une part une procédure de reconfiguration afin de reconstituer des chemins valides, et d'autre part une procédure de reprise pour les activités interrompues. Une panne se traduit donc pour l'utilisateur par un léger accroissement du temps de réponse dû au basculement vers le processeur de secours et à la réexécution de l'étape interrompue.

Après une panne, le système fonctionne en mode dégradé et sensible à une nouvelle défaillance jusqu'à la réparation de la panne. Les systèmes en service continu résistent à la défaillance d'un élément. Seule une double panne provoque un arrêt complet du système. Les estimations basées sur la fréquence des divers types de panne montrent que le nombre d'arrêts devrait ainsi être de l'ordre de 1 par an.

Application aux systèmes d'information répartis :

Dans un système d'information réparti, l'obstacle majeur à la réalisation d'un mécanisme de contrôle de la cohérence sûr réside dans l'incapacité de définir un protocole de validation non bloquant. Nous avons vu au paragraphe 3.3.4 qu'il est possible de définir un protocole résistant à une panne du site contrôlant le supérieur. Le protocole de validation à 3 phases en est une illustration. Dans cette technique, le coût de la sécurité est considérable car il grève le temps de réponse de toutes les transactions en fonctionnement normal à cause de la synchronisation imposée par la phase supplémentaire. Une approche différente consiste à considérer un système distribué dans lequel les sites (ou seulement un sous-ensemble d'entre eux) répondent aux caractéristiques d'un système transactionnel en "Service Continu". Si on fait abstraction du coût imposé par la redondance matérielle, cette architecture présente sur le plan théorique deux avantages notables :

- Elle résiste à une panne du supérieur si celui-ci s'exécute sur un système en service continu (cette propriété n'est plus vérifiée en cas de panne multiple.)

- Elle n'induit aucune surcharge préjudiciable au temps de réponse des transactions en fonctionnement normal.

En théorie, il est possible de généraliser le principe de redondance à l'ensemble des composants du système réparti, donc au réseau, et d'éliminer ainsi le risque de blocage pour les agents inférieurs. Dans une configuration de ce type, le réseau offrirait, en fonctionnement normal, deux chemins physiques indépendants pour le transport des messages. Cette solution est irréaliste dans le cas d'un système géographiquement réparti, construit sur réseau "longue distance".

Elle devient concevable pour des systèmes multi-processeurs ou des systèmes localement distribués dans lesquels les communications utilisent un bus ou empruntent un réseau local.

Considérons à nouveau l'architecture redondante décrite dans cette section, qui est fondée sur le couplage d'activités s'exécutant sur des processeurs interconnectés. Outre le gain en disponibilité, cette approche présente tous les avantages d'une architecture ouverte et localement distribuée. Cette ouverture se présente sous différents aspects :

- Puissance de traitement extensible par adjonction d'éléments supplémentaires (unité centrale, sous-système disque, etc.).

- Croissance modulaire de l'application : on peut faire évoluer la base de données et les programmes de traitement sans arrêter la production.

- Evolution du système transactionnel : il est possible en effet de charger sur le processeur de secours une nouvelle version du système transactionnel et de réaliser in-situ le basculement des applications vers ce processeur, sans arrêt du système global.

- Stratégies de répartition de l'information modulables selon les applications ou les configurations. En fonction des besoins, une base de données peut être partagée entre des processeurs (chemins d'accès multiples) ou, plus simplement partitionnée.

Chapitre

V

Conclusion

La section 1 dresse un bilan de notre contribution à la fois sur le plan conceptuel (étude des mécanismes) et sur celui des retombées industrielles. En partant de l'expérience acquise dans le domaine du contrôle de la cohérence, la section 2 suggère une voie d'étude axée sur la définition d'un noyau de contrôle associant les concepts de transaction et de type abstrait.

1. Bilan : Contribution à l'étude du contrôle de cohérence.

Après un bref rappel des motivations et des contraintes qui ont influé sur le déroulement du projet SCOT, nous résumons les éléments significatifs de notre contribution au contrôle de la cohérence, en examinant successivement les mécanismes de validation-reprise, de contrôle de l'accès concurrent et d'exécution répartie. Nous décrivons ensuite les prolongements industriels de ces travaux.

Le projet SCOT est né de la convergence de deux courants d'idées. Le premier est d'ordre conceptuel et concerne l'héritage du projet POLYPHEME (bilan et orientations nouvelles). Le second, beaucoup plus pragmatique, est l'expression d'un besoin à l'intérieur du Groupe Bull sous la poussée des utilisateurs de systèmes d'information recherchant une évolution de leurs applications vers des configurations réparties. Cette demande issue des entités opérationnelles explique en partie certaines des orientations du projet SCOT (architecture de type transactionnel coopérant, réseau général, prise en compte des applications). De ce point de vue, SCOT doit être considéré comme un banc d'essai pour l'évaluation de mécanismes de contrôle de la cohérence. Par évaluation, nous entendons à la fois une étude de comportement (propriétés intrinsèques, comparaison avec d'autres mécanismes) mais aussi une étude du champ d'application. Cette démarche dans laquelle on associe des aspects théoriques, l'évaluation et l'adéquation aux applications, a démarqué fortement notre approche et a influencé la nature des résultats qui ont été présentés ici.

Dès les premières pages de cet ouvrage, nous avons souligné la variété des besoins des applications en matière de cohérence. Ce postulat a constamment guidé nos travaux et orienté nos choix vers des mécanismes qui autorisent des formes de cohérence moins contraignantes que la cohérence stricte, indispensable mais chère.

Comme le lecteur a pu le constater, un effort particulier a porté sur l'étude des mécanismes de validation et de reprise. A partir d'un modèle classique de transaction répartie, nous avons étudié en détail les éléments majeurs entrant dans la réalisation d'un mécanisme de validation-reprise. Pour chacun d'eux plusieurs solutions ont été présentées et comparées en fonction de l'environnement ou des scénarios d'application. Cette analyse a porté essentiellement sur le choix du supérieur de la validation, la technique de propagation des messages du protocole, les procédures de validation et celles de reprise. Cette étude a montré, d'une part qu'il n'y a pas de solution optimale pour l'ensemble de ces critères, d'autre part que la nature du réseau et de l'application supportée ont un impact majeur dans le choix d'une solution. Lorsque cela a été possible, nous avons proposé un mécanisme autorisant la cohabitation de plusieurs solutions ; ce fut le cas en particulier pour la procédure de validation mixte. Pour le reste, nous avons édicté des recommandations à l'usage des concepteurs pour le choix du mécanisme approprié en fonction des caractéristiques de l'environnement d'exécution.

Le contrôle de l'accès concurrent a été peu étudié sur le plan théorique. Le contexte industriel évoqué plus haut nous a conduit à nous intéresser plus particulièrement aux mécanismes de verrouillage (afin de faciliter l'évolution vers des configurations réparties des produits existants qui, précisément, l'utilisent). Le choix d'une technique de verrouillage à deux phases avec prévention des interblocages est le résultat d'une campagne d'évaluation portant sur le comportement respectif des techniques de détection et prévention des interblocages. Les résultats originaux obtenus dans ce domaine ont été confirmés depuis lors par d'autres études publiées dans la littérature.

Les travaux ayant trait à l'éxécution répartie ont été peu évoqués dans cet ouvrage. Ils ont constitué cependant une partie importante du projet et ont donné lieu à des contributions significatives, d'une part sur la définition d'un niveau "session" adapté au transactionnel coopérant et, d'autre part sur l'impact de la "diffusion fiable" sur la définition des protocoles.

L'ensemble des résultats et expériences issus des projets POLYPHEME et SCOT a fortement contribué à la spécification d'un Système Transactionnel Coopérant au sein des groupes Honeywell et Bull. Ce système permettra la coopération de systèmes transactionnels hétérogènes fonctionnant sur des machines de puissance très variable, connectées entre elles par un réseau de type "X25". Plusieurs techniques retenues dans notre noyau transactionnel, parmi lesquelles la procédure de validation mixte et le contrôle de l'accès concurrent basé sur le verrouillage et la prévention des interblocages, ont été adoptés pour ce futur produit. Ces mêmes résultats sont également réinvestis aujourd'hui dans la spécification d'un système transactionnel en service continu en cours de définition.

2. Perspectives : une nouvelle approche du contrôle de la cohérence

Il existe différentes voies de poursuite des travaux entrepris sur le contrôle de la cohérence. Dans la suite, nous en décrivons brièvement quelques unes avant de nous attarder plus particulièrement sur celle qui nous paraît offrir les perspectives les plus intéressantes.

Le nombre de mécanismes proposés dans la littérature est considérable, notamment en ce qui concerne le contrôle de l'accès concurrent. Un effort important reste à faire en ce qui concerne l'évaluation et le champ d'application de ces mécanismes. Les articles récents, proposant des techniques hybrides regroupant les potentialités de plusieurs algorithmes, vont dans ce sens.

Une autre voie d'étude concerne l'adéquation de ces techniques de contrôle à d'autres environnements tels que la bureautique, la CAO ou le Génie Logiciel. L'évolution de ces applications vers des configurations réparties pose naturellement le problème du contrôle de la cohérence. La nature des objets traités (grande taille, structure hiérarchique) et des opérations sur ces objets (transactions de longue durée) exige, soit des aménagements des techniques classiques, soit de nouvelles méthodes. On trouve aujourd'hui plusieurs amorces de solutions qui, pour la plupart, exploitent les propriétés des transactions imbriquées.

Nous souhaiterions, en dernier lieu, revenir sur le concept de noyau de contrôle de la cohérence évoqué à la fin du chapitre précédent. Nous avons justifié la réalisation d'un "noyau minimal" par la capacité qu'il offre d'exprimer au niveau des applications des stratégies de contrôle variées. Ces stratégies sont mises en oeuvre par le concepteur de l'application, à l'aide d'une interface de programmation élémentaire (appel de primitives du noyau) et sont dictées par la connaissance sémantique qu'il a de l'application. D'un point de vue formel, cette approche présente un inconvénient majeur, à savoir qu'une part importante du contrôle de cohérence est diluée dans la programmation de l'application, autorisant ainsi des effets de bord incontrôlables. Nous étudions ici une voie qui permettrait de restituer au noyau un certain nombre de contrôles sémantiques tout en conservant la capacité d'exprimer des politiques de contrôles variées. Cette possibilité passe par l'enrichissement sémantique des objets et des opérations qui sont manipulés au niveau du noyau transactionnel et qui constituent les fondements du contrôle de cohérence.

L'exemple du système bancaire, déjà exploité en diverses occasions, est repris pour illustrer les carences du modèle transactionnel classique.

Exemple : Pour les besoins de notre argumentation, on étend la sémantique de l'objet de type "Compte bancaire" qui devient :

```
Type COMPTE
début
     Numéro : entier,
     Solde  : réel,
     Découvert-autorisé : réel,

     Fonction CONSULTER, retour : réel,
     Procédure CREDITER (M : réel)
     Procédure DEBITER (M : réel)
     Procédure CHANGER-DECOUVERT (D : réel)
fin
```

Les opérations susceptibles de s'appliquer à ce type d'objet sont les classiques CONSULTER, DEBITER et CREDITER auxquelles nous rajoutons CHANGER-DECOUVERT, qui consiste à modifier la valeur du découvert autorisé. Si on se réferre à la description de ces opérations (cf. Chapitre 1, §2.1), on constate qu'elles s'expriment à l'aide des seules actions élémentaires LIRE-donnée et ECRIRE-Donnée. La synchronisation des accès aux objets partagés porte précisément sur ces actions. En opérant ainsi, on banalise les actions LIRE-D, ECRIRE-D et on perd la connaissance de l'opérateur dont elles sont la traduction.

Si le mécanisme de synchronisation adopté pour le noyau est le verrouillage à deux phases, l'exécution concurrente des opérateurs CONSULTER et DEBITER (ou CREDITER) est impossible. Seule une programmation explicite au niveau de l'application permet d'imposer cette stratégie. On remarquera au passage que la méthode de certification autorise cette exécution. En revanche, aucune méthode n'autorise l'exécution concurrente des opérations CREDITER et CHANGER-DECOUVERT alors que cette combinaison a un sens dans la réalité. Cette restriction découle du fait que le mécanisme de contrôle du noyau, opérant au niveau de l'action ECRIRE-D, a perdu l'information sur l'origine de cette primitive. Nous verrons plus loin d'autres exemples de restrictions imposées par la perte d'information sur l'origine des actions exécutées sur les objets.

L'exemple qui vient d'être développé montre qu'un enrichissement sémantique du contrôle de la cohérence est souhaitable. Il suppose qu'on sache résoudre les deux problèmes suivants :

1 - exprimer au niveau d'un objet ou d'un groupe d'objets des règles de cohérence incluant les conditions de synchronisation, de validation, de reprise après panne ou de traitement des erreurs,

2 - réaliser au niveau du noyau transactionnel des mécanismes de contrôle qui puissent exploiter ces spécifications.

Les types abstraits sont un début de réponse à la première question à condition qu'on puisse compléter la définition d'un type avec la description des règles de cohérence. Les avantages attendus de l'utilisation des types abstraits sont nombreux. Ils constituent une méthode sûre et homogène de description des traitements associés à une classe d'objets et ils favorisent la construction modulaire d'applications en autorisant la définition de types construits. Au lieu d'être seulement une séquence d'actions LIRE-D et ECRIRE-D, une transaction peut être définie comme une hiérarchie d'opérations typées sur des objets.

Exemple : Un transfert de fonds entre deux comptes peut s'exprimer par l'exécution de l'opérateur TRANSFERT appliqué à un objet du type "Virement" construit à partir du type élémentaire "Compte".

```
Type VIREMENT
début
     Cd : COMPTE
     Cc : COMPTE

     Procédure TRANSFERT (M : réel)
     début
       DEBIT (Cd,M)
       CREDIT (Cc,M)
     fin
fin
```

Dans la déclaration d'un type, un opérateur peut être déclaré comme une "Transaction". Dans ce cas, le système assure l'atomicité des actions liées à l'opérateur. S'il opère sur un type construit et fait référence à d'autres opérateurs eux-même définis comme des transactions, on obtient une structuration en **transactions imbriquées.**

Le dernier atout en faveur d'une approche basée sur les types abstraits est l'existence de supports linguistiques pour la programmation des applications. Il existe aujourd'hui plusieurs tentatives visant à combiner les avantages respectifs d'un contrôle basé à la fois sur les propriétés d'un type et sur le concept de transaction [Liskov 82, Weihl 83-a, Allchin 83]. Il est à noter que certaines de ces études dépassent le seul cadre des systèmes d'information. Une expérience similaire a été tentée dans le projet SCOT avec le langage ADA dans lequel le concept de transaction a été introduit [Scot 83] (dans cette étude, la définition des types ne comportait pas de règles de cohérence). Dans la suite nous décrivons quelques exemples d'expression de règles de cohérence et les méthodes de contrôle correspondantes.

a) contrôle de l'accès concurrent

En l'absence de connaissance sur la sémantique des objets, la sérialisation des transactions est un moyen systématique de synchroniser les accès à un ensemble d'objets partagés. L'exemple introductif, développé dans la section précédente, a montré que la sérialisation constitue parfois une forme de synchronisation trop contraignante.

A la notion d'ordonnancement sérialisable, on substitue celle d'ordonnancement sémantiquement correct basé sur l'application d'un ensemble plus restreint de synchronisations en accord avec la sémantique des objets. Dans la présentation des mécanismes de contrôle de l'accès concurrent au chapitre 2, nous avons insisté sur le rôle joué par l'étude des dépendances entre transactions

dans la détermination d'un algorithme de synchronisation optimal. On rappelle qu'une dépendance s'établit entre deux transactions lorsqu'elles tentent de manipuler le même objet. L'étude du graphe des dépendances permet de déterminer avec précision les situations pouvant conduire à une incohérence (matérialisée par un cycle dans le graphe de dépendance). Malheureusement, dans les systèmes d'information classique, l'étude des dépendances porte exclusivement sur les deux opérateurs LIRE-D et ECRIRE-D, ce qui diminue considérablement les possibilités d'exécutions concurrentes.

Dans une approche typée du contrôle de la cohérence, l'étude des dépendances est étendue aux opérateurs du type. Ainsi, dans l'exemple du compte bancaire, il est possible de préciser la compatibilité des opérateurs DEBITER, CONSULTER, CREDITER et MODIFIER de la manière suivante :

- DEBITER et CREDITER sont incompatibles; (la cohérence forte est requise dans le cas d'un transfert de fond).

- CONSULTER est compatible avec DEBITER et CREDITER (la cohérence forte n'est pas requise lorsqu'on se contente d'une valeur indicative pour le solde du compte).

- MODIFIER est compatible avec CREDITER mais ne l'est pas avec DEBITER (l'exécution d'un débit tient compte de la valeur du découvert autorisé).

Plusieurs auteurs proposent des mécanismes de contrôle de l'accès concurrent qui prennent en compte des règles spécifiques à chaque type pour contrôler de manière plus souple, et donc plus efficace, l'accès aux objets partagés. Nous présentons ici une technique basée sur le verrouillage et désignée sous le terme "verrouillage typé". Cette technique s'appuie sur la déclaration de "classes de verrou" spécifiques de chaque type et d'une matrice de compatibilité entre les opérateurs et ces différentes classes.

Exemple : Pour le type COMPTE, on définit quatre types de verrous (notés "Consultation", "Débit", "Crédit" et "Découvert", associés aux quatre opérations définies plus haut.

Le verrou "Consultation" indique qu'une opération CONSULTER est en cours sur le compte, qui n'est pas encore validée. Les verrous "Débit", "Crédit" et "Découvert" ont une signification analogue pour les opérations DEBITER, CREDITER et CHANGER-DECOUVERT. La matrice de compatibilité est construite à partir des déclarations portant sur la compatibilité des opérateurs du type COMPTE.

Opération demandée	Verrou existant			
	"Consu"	"Débit"	"Crédit"	"Découvert"
CONSULTER	oui	oui	oui	oui
CREDITER	oui	non	non	oui
DEBITER	oui	non	non	non
CHANGER-D	oui	non	oui	non

Lorsqu'une demande est incompatible avec le verrou existant, un conflit est détecté. Le traitement correspondant peut varier selon les types (attente, abandon, opération différée jusqu'à la validation, etc .). Le lecteur trouvera dans [Bernstein 83-b, Korth 83, Schwartz 84] des exemples de mise en oeuvre d'un contrôle de l'accès concurrent typé. [Korth 82] décrit en outre une méthode de prévention des interblocages adaptée à cette forme de contrôle.

b) traitement des erreurs.

A chaque opérateur d'un type, on peut associer une action à exécuter en cas de dysfonctionnement (erreur sémantique, panne d'un agent coopérant ou du réseau, etc...). Les actions peuvent différer selon les types et les opérateurs : arrêt, message d'erreur, traitement correcteur, etc . Si aucun traitement n'a été spécifié au niveau d'un opérateur pour un incident donné, le signal d'erreur est transmis à l'opérateur englobant. Cette structuration permet de généraliser le mécanisme de gestion hiérarchisé des erreurs déjà évoqué à propos des transactions imbriquées.

c) validation et reprise après panne.

Dans le modèle général de système transactionnel, l'indivisibilité et la permanence des actions d'une transaction ont été présentés comme des fondements de l'atomicité. Les formes de cohérence dégradée s'accomodent de certains manquements à ces règles et apportent, en contrepartie, un gain de performance. Ainsi, la sémantique d'une application accepte, dans certains cas, de rendre visibles les effets d'une action avant la fin d'une transaction. Cette technique ne garantit pas la cohérence forte mais accroît la disponibilité des objets. Lorsque la permanence des effets d'une transaction n'est pas impérative, pour ce qui est de certains objets, il est inutile de forcer immédiatement leur écriture en mémoire secondaire à la fin d'une transaction. Cette opération peut être différée et réalisée en parallèle avec l'exécution d'autres transactions. La même remarque s'applique à la recopie des actions sur les journaux, au cours ou à la fin d'une transaction. Ces optimisations ponctuelles se traduisent par un gain de performance, sans garantie toutefois de pouvoir défaire ou refaire les actions correspondantes [Spector 83, Schwartz 84].

L'approche qui vient d'être brièvement décrite présente deux avantages. D'une part, elle autorise une plus grande souplesse dans la définition des niveaux de cohérence; d'autre part, elle permet d'harmoniser les traitements pour une classe d'objets déterminée. L'existence de supports linguistiques renforce l'aspect formel de cette approche et devrait permettre à la fois une programmation plus sûre et plus facile [Weihl 83-b, Liskov 83]. Dans le futur, de nouvelles investigations seront nécessaires pour la définition de mécanismes de synchronisation, de validation et de reprise qui puissent exploiter les spécifications propres à chaque type.

Références Bibliographiques

[Adiba 78] M. Adiba, "Un modèle relationnel et une architecture pour les systèmes de bases de données réparties : application au projet POLYPHEME". Thèse de doctorat d'Etat, Université de Grenoble, septembre 1978.

[Agrawal 83] R. Agrawal, D. DeWitt, "Integrated concurrency control and recovery mechanisms : Design and Performance evaluation". Technical report n° 497. University of Wisconsin, février 1983.

[Allchin 83] J. Allchin, M. McKendry,"Synchronization and recovery of actions." Proc. of 2nd Principles of distributed Computing Conference, Août 1983.

[Andrade 80] J.M. Andrade. "L'intégrité et la mise à jour dans un système de gestion de bases de données réparties. Projet POLYPHEME". Thèse de Docteur ingénieur, Université de Grenoble, novembre 1980.

[André 78] E. André, P. Decitre, "On providing distributed applications programmers with control over synchronization". Proc. of Computer Networks Protocol Symposium, Liège, janvier 1978.

[Badal 80] D. Badal, "On the degree of concurrency provided by concurrency controls mechanismes for distributed databases". Proc. of International Symposium on distributed systems, INRIA, Paris, 1980.

[Badal 81] D. Badal, "Concurrency control overhead or closer look at blocking VS. nonblocking concurrency control mechanisms". Proc. of 5th Berkeley workshop on distributed data management and computer networks, février 1981.

[Baer 80] J. Baer, G. Gardarin, C. Girault, G. Roucairol, "The two-step commitment protocol modeling, specification and proof methodology". Proc. of 5th Conference on Software Engineering, San Diego, Mai 1980.

[Balter 81] R. Balter, "Selection of a commitment and recovery mechanism for a distributed transactional system". IEEE Symposium on Reliability in Distributed Software and Database Systems, Pittsburgh, Juillet 1981.

[Balter 82] R. Balter, P. Bérard, P. Decitre, "Why control of the concurrency level in distributed systems is more fundamental thand deadlock management". ACM-SIGACT-SIGOPS Symposium of Principles of Distributed Computing. Ottawa, Août 1982.

[Balter 84] R. Balter, P. Decitre, "Validation et reprise dans le système transactionnel coopérant SCOT". TSI, Vol.3 (2), 1984.

[Banâtre 84] M. Banâtre, "Le système ENCHERE : une expérience dans la conception et la réalisation d'un système réparti". Thèse de Doctorat d'Etat, Rennes, Mars 1984.

[Bartlett 81] J. Bartlett, "A Non-stop kernel. 8th ACM Symposium on Operating Systems Principles, Décembre 1981.

[Batchelor 82] R. Batchelor, "Analysis of concurrency control". Research Report, HIS Billerica, Décembre 1982.

[Bayer 80] R. Bayer, H. Heller, A. Reiser, "Parallelism and Recovery in database systems". ACM Transactions on Database Systems, Vol. 5 (2), 1980.

[Bayer 82] R. Bayer, K. Elhardt, J. Heigert, A. Reiser, "Dynamic timestamp allocation for transaction in database systems". Proc. of 2nd Symposium on Distributed Databases, Berlin, Septembre 1982.

[Benmakrouha 82] F. Benmakrouha, "Evaluation de performance de systèmes transactionnels répartis". Thèse de Docteur Ingénieur, Université de Rennes, Mars 1982.

[Bernstein 80] P. Bernstein, N. Goodman, "Timestamp based algorithm for concurrency control in distributed database systems". Proc. of 6th Conference on VLDB, Montréal, 1980.

[Bernstein 81] P. Bernstein, N. Goodman, "Concurrency control in distributed database systems". ACM Computing Survey, 13 (2), 1981.

[Bernstein 83-a] P. Bernstein, N. Goodman, V. Hadzilacos, "Recovery algorithms for database systems". Proc. of IFIP, Paris, Septembre 1983.

[Bernstein 83-b] P. Bernstein, N. Goodman, M. Lai, "Analysing concurrency control algorithms when user and system operations differ". IEEE Trans. on Software Engineering, vol.SE-9 (3), Mai 1983.

[Bhargava 82] B. Bhargava, "Resiliency features of the optimistic concurrency control approach for distributed database systems". Proc. of 2nd IEEE Symposium on Reliability in Distributed Software and Database, Pittsburgh, Juillet 1982.

[Boksenbaum 84] C. Boksenbaum, M. Cart, J. Ferrie, JF. Pons, "Certification by intervals of timestamps in distributed database systems". Proc. 10th conference on VLBD, Singapour, Août 1984.

[Boksenbaum 85] C. Boksenbaum, M. Cart, J. Ferrie, JF. Pons, "Concurrent certifications in distributed database systems". Proc. of 8th Int. Computing Symposium ACM ICS Florence, Mars 1985.

[Boral 84] H. Boral, I Gold, "Towards a self-adapting centralized concurrency control algorithm". Proc. of ACM SIGMOD, Boston, Juillet 1984.

[Borr 81] A. Borr, "Transaction monitoring in ENCOMPASS : reliable distributed transaction processing". Proc. of 7th conference on VLDB, Cannes, Septembre 1981.

[Borr 84] A. Borr, "Robustness to crash in a distributed database : a non shared memory multi-processor approach". Proc. of Conference on VLDB, Singapour, Août 1984.

[Caleca 78] J.Y. Caleca, "Projet Polyphème : l'expression et la décomposition de transactions dans un système de bases de données réparties". Thèse de 3ème cycle, Université de Grenoble, Septembre 1978.

[Carey 84] M. Carey, M. Stonebraker, "The performance of concurency control algorithm for database management systems". Proc. of Conference on VLDB, Singapour, Août 1984.

[Ceri 82] S. Ceri, S. Owicki, "On the use of optimistic methods for concurrency control in distributed databases". Proc. of 6th Berkeley Workshop on Distributed Data Management and Computer Networks, Berkeley, Février 1982.

[Colliat 79] G. Colliat, C. Bachman, "Commitment in a Distributed Database". Database Architecture, G. Bracchi and G. Nijissen eds., North Holland. 1979.

[Cooper 82] E. Cooper, "Analysis of distributed commit protocols". Proc. of ACM SIGMOD Conference on Management of data, Orlando, Juin 1982.

[Cornafion 81] Groupe Cornafion, "Systèmes informatiques répartis : concepts et techniques". Ed DUNOD, 1981.

[Decitre 80] P. Decitre, E. André, "POLYPHEME project : the DEM distributed execution monitor". Proc. of Symposium on distributed databases, Paris, North Holland Ed, Mars 1980.

[Decitre 81] P. Decitre, "A concurrency control algorithm in a distributed environment". Proc. of National Computer Conference, Chicago, 1981.

[Delobel 82] C. Delobel, M. Adiba, "Bases de données et systèmes relationnels", Dunod 1982.

[Devor 80] C. Devor, J.A. Weeldreyer,"DDTS : A testbed for distributed database research". Proc. of ACM PACIFIC 80 San Francisco.

[Devor 82] C. Devor, "The effects of locking mechanism design on database management system performance". Research Report, Honeywell CCSC Minneapolis, Septembre 1982.

[DeWitt 84] D. DeWitt, R. Katz, F. Olken, L. Shapio, M.Stonebraker, D. Wood, "Implementation techniques for main memory database systems". Proc. Of ACM SIGMOD, Boston, juin 1984.

[Eswaran 76] K. Eswaran, J. Gray, R. Lorie, I. Traiger, "The notions of consistency and predicate locks in a database system". Communication ACM, 19 (11) Novembre 1976.

[Galler 82] B. Galler, "Concurrency control performance issues". Ph. D. Thesis, University of TORONTO, 1982.

[Garcia 79] H. Garcia-Molina, "Performance of update algorithms for replicated data in a distributed database". Ph. D. Thesis, Standford, Juin 1979.

[Garcia 83] H. Garcia-Molina, "Using semantic knowledge for transaction processing in a distributed database". ACM Transactions On Database Systems, vol 8 (2), juin 1983.

[Gardarin 78] G. Gardarin, "Résolution des conflits d'accès simultanés à un ensemble d'informations. Application aux bases de données réparties". Thèse de doctorat d'Etat, Paris, Avril 1978.

[Gardarin 80] G. Gardarin, "Integrity, consistency, concurrency, reliability in distributed database management systems". Distribued Database, North Holland ed, 1980.

[Gardarin 82] G. Gardarin, P. Bernadat, N. Temmerman, P. Valduriez, Y. Viemont, "SABRE : a relational database system for a multi-microprocessor machine". Proc. of the International Workshop on Database Machines, San Diego, Août 1982.

[Gelenbe 77] E. Gelenbe, "On the optimum checkpoint interval". Rapport de Recherche IRIA, n° 232, 1977.

[Gifford 84] D. Gifford, A. Spector, "The TWA reservation system". Communications of the ACM, Juillet 1984.

[Gligor 80] V. Gligor, S. Shattuck, "On deadlock detection in distributed systems". IEEE Transactions on Software Engineering, vol SE-6, Septembre 1980.

[Goldman 77] M. Goldman "Deadlock detection in computer networks". Tech. Report MIT/LCS/TR-185, MIT, Septembre 1977.

[Gouda 78] M. Gouda, "A hierarchical controler for concurrent access in distributed databases". Proc. of 4th Workshop on Computer Architecture, Syracuse, Août 1978.

[Gray 78] J. Gray. "Notes on database operating systems". Research report RJ2188 IBM, Research Laboratory, San Jose, Février 1978.

[Gray 80] J. Gray, "A transaction model". 8th International Conference on language and programming, 1980.

[Gray 81-a] J. Gray, "The transaction concept. Virtues and Limitations". Research Report, Tandem Computers, Avril 1981.

[Gray 81-b] J. Gray, P. McJones, M. Blasgen, B. Lindsay, R. Lorie, T. Price, F. Putzolu, I. Traiger, "The recovery manager of the system R database manager" ACM Computing Survey, Vol 13 (2), Juin 1981.

[Haerder 83] T. Haerder, A. Reuter, "Principles of transaction-oriented database recovery". ACM Computing Survey, Vol 15 (4), Décembre 1983.

[Hammer 80] M. Hammer, D. Shipman, "Reliability mechanisms for SDD-1 : a system for distributed databases". ACM Transactions On Database Systems, vol. 5 (5), Décembre 1980.

[Kastner 83] P. Kastner, "A fault-tolerant transaction processing environment". IEEE Q. Bul. on Database Eng. 6,2 Juin 1983.

[Katzman 77] J. Katzman, "System architecture for non-stop computing". Proc. of IEEE Compcon, Février 1977.

[Khider 83] A. Khider, "Protocole de diffusion fiable pour réseaux locaux à diffusion". Thèse de Docteur Ingénieur, Université de Grenoble, Mai 1983.

[Kim 82] W. Kim, "AUDITOR : a framework for High Availability of DB/DC systems". IBM Research Report RJ3512, Juin 1982.

[Kim 84] W. Kim, "Highly available systems for database applications". ACM Computing Surveys, Vol.16 (1), Mars 1984.

[Kohler 81] W. Kohler, "A Survey of techniques for synchronization and recovery in decentralized computer systems". ACM Computing surveys, Vol.13 (2), Juin 1981.

[Kohler 83] W. Kohler, K. Wilner, J. Stankovic, "An experimental evaluation of locking policies in a centralized database system". Proc. of ACM SIGMOD Conference, San Jose, Mai 1983.

[Korth 82] H. Korth, "Deadlock freedom using edge locks". ACM Transactions On Database Systems, Vol.7 (4), Décembre 1982.

[Korth 83] H. Korth, "Locking primitives in a database system". Journal of ACM, vol.30 (1), Janvier 1983.

[Kung 81] H. Kung, J. Robinson, "On optimistic methods for concurrency control". ACM Transactions On Database Systems, 6 (2), Février 1981.

[Lampson 77] B. Lampson, H. Sturgis, "Crash recovery in a distributed data storage system". Reseach Report, Xerox Palo Alto research Center, 1977.

[Lausen 82] G. Lausen, "Concurrency control in database systems : a step towards the integration of optimistic methods and locking". Proc. of ACM National Conference, Dallas, Octobre 1982.

[LeBihan 79] J. le Bihan, C Esculier, G. LeLann, L. Treille, "SIRIUS DELTA : Un prototype de système de gestion de bases de données distribuées". Proc. of Symposium on Distributed Databases, Paris, Mas 1980, North Holland Ed.

[LeBihan 80] J. LeBihan, C. Esculier, W. Litwin, G. Gardarin, S. Sedillot, L. Treille, "SIRIUS, a french nationwide project on distributed databases". Proc. of Conference on VLDB, Montréal, 1980.

[Lelann 79] G. Lelann, "Consistency and concurrency control in distributed database systems". Proc. of EEC advanced course on distributed databases, Sheffield, Cambridge University Press, 1979.

[LeLann 81] G. LeLann, "Consistency issues in distributed databases". Proc. of On-time Conference on Distributed Databases, London, mars 1981.

[Lin 81] W. Lin, "Performance evaluation of two concurrency control mechanisms in a distributed DBMS". Proc. of ACM SIGMOD International Conference on Management of Data, An Arbor, Avril 1981.

[Lin 83] W. Lin, J. Nolte, "Basic timestamp, Multiple version timestamp, and Two phase locking". Proc. of Conference on VLDB, Florence, Novembre 83.

[Lindsay 79] B. Lindsay, P. Selinger, C. Galtieri, J. Gray, R. Lorie, T. Price, F. Putzolu, I. Traiger, B. Wade, "Notes on distributed programs". Research Report RJ2571, IBM lab., San Jose, Juillet 1979.

[Liskov 82] B. Liskov, "On linguistic support for distributed programs". IEEE Transactions on Software Engineering, vol SE-8, n° 3, Mai 1982.

[Liskov 83] B. Liskov, R. Scheifler, "Guardians and Actions : Linguistic support for robust distributed programs". ACM Trans. on Programming languages and systems, vol.5 (3), Juin 83.

[Lomet 78] D. Lomet, "A practical deadlock avoidance algorithm for database systems". Proc. of ACM SIGMOD, Toronto, Août 1977.

[Lorie 77] R. Lorie, "Physical integrity in a large segmented database". ACM Trans.On Database Systems 2 (1), Mars 1977.

[Madelaine 82] J. Madelaine, "Evaluation d'algorithmes de contrôle de concurrence". Rapport INRIA n° 164, Octobre 1982.

[Menasce 79] D. Menasce, C. Popek, "Locking and deadlock detection in distributed databases". IEEE Trans. on Software Engineering, Vol SE-5, n° 3, Mai 1979, pp 195-202.

[Menasce 82] D. Menasce, T. Nakanishi, "Optimistic versus pessimistic concurrency control mechanisms in Database Management Systems". Information Systems 7 (1), 1982.

[Merle 78] R. Merle, D. Potier, M. Veran, "QNAP. A tool for computer performance analysis". Proc. of ICPCI. North Holland, 1978.

[Mitchell 82] J. Mitchell, J. Dion, "A comparison of two netword-based file servers". Commununications of ACM, vol 25 (4), avril 1982.

[Moss 82] J. Moss, "Nested transactions : an approach to reliable distributed computing". Ph. D. Thesis, MIT, Avril 1981.

[Neuhold 76] E. Neuhold, M. Stonebraker, "A distributed database version of INGRES" Proc. 2nd Berkeley Workshop on Distributed Data and Computer Networks, Mai 1976.

[Obermarck 82] R. Obermarck, "Distributed deadlock detection algorithm". ACM Transactions On Database Systems, vol 7 (2), Juin 1982.

[Papadimitriou 79] C. Papadimitriou, "Serializability of concurrent updates". Journal of ACM, vol.26 (4), Octobre 1979.

[Polyphème 79] CII-Honeywell Bull, IMAG, "POLYPHEME : un système de bases de données réparties". Rapport de fin de contrat. Octobre 1979.

[Potier 80] D. Potier, Ph. Leblanc, "Analisys of locking policies in database management systems". Communications of ACM, 23 (10), Octobre 1980.

[Praëdel 82] U. Praëdel, G. Schlageter, R. Unland, "Two problems of optimistic concurrency control : long transactions and starvation". Research Report, FernUniversität, Hagen, 1982.

[Rahimi 82] S. Rahimi, W. Franta, "Fair timestamp allocation in distributed systems". Proc. National Computer Conference, Houston, Juin 1982.

[Reed 78] D. Reed, "Naming and synchronization in a decentralized computer system". Ph. D. Thesis, MIT, Septembre 1978.

[Ries 79] Ries, Stonebraker, "Locking granularity revisited". ACM Transaction On Database System, Juin 1979.

[Robinson 84] J. Robinson, "Separating policy from Correctness in concurrency control Design". Software Practice and Experience, vol.14, Septembre 1984.

[Rosenkrantz 78] D.J. Rosenkrantz, R.E. Stearns. P.M. Lewis, "A system level concurrency control for distributed database systems". ACM Transactions on Database Systems, Vol.3 (2) Juin 1978.

[Schlageter 81] G. Schlageter, "Optimistic methods for concurrency control in distributed database systems". Proc. 7th Conference on VLDB, Cannes, Septembre 1981.

[Schlageter 82] G. Schlageter, "Problems of optimistic concurrency control in distributed database systems". ACM SIGMOD Record, Vol.12(3), Avril 1982.

[Schwartz 84] P. Schwartz, A. Spector, "Synchronizing shared abstract types". ACM Transactions On Computer Systems, vol.2 (3), Août 1984.

[Scot 81] Groupe SCOT, "Adaptation du protocole SCOT à un réseau local à diffusion fiable". Rapport de Recherche n° 19, Cii-Honeywell Bull, Juin 1981.

[Scot 83] Groupe SCOT, "Rapport final du projet SCOT". Centre de Recherche Cii-Honeywell Bull, Grenoble, janvier 1983.

[Sdd-1 79] Groupe SDD-1, "SDD-1 : a system for distributed databases". Technical report CCA 02-79, Computer Corporation of America, janvier 1979.

[Sergeant 80] G. Sergeant, L. Treille, "SER : a system for distributed execution based on decentralized control techniques". Proc. of 5th Conference on Computer Communication, Atlanta, Octobre 1960.

[Skeen 81] D. Skeen, M. Stonebraker, "A formal model of crash recovery in a distributed system". Proc. of 5th Berkeley Workshop on Distributed Data Management and Computer Networks, Mai 1981.

[Skeen 82-a] D. Skeen, "A quorum based commit protocol". Proc. of 6th Berkeley Workshop, Février 1982.

[Skeen 82-b] D. Skeen, "Non blocking commit protocols". Proc. of ACM SIGMOD, Conference on Management of Data, 1982.

[Smith 83] L. Smith, K. Madsen, "Nonstop Transaction Processing". Computer Design, Mars 1983.

[Spector 83] A. Spector, P. Schwartz, "Transactions : a construct for reliable distributed computing". CMU Research report, Janvier 1983.

[Stonebraker 79] M. Stonebraker, "Concurrency control and consistency of multiple copies of data in distributed INGRES". IEEE Transactions on Software Engineering, SE-5, 3 Mai 1979.

[Svobodova 80] L. Svobodova, "Management of object histories in the SWALLOW repository". Research Report, MIT, 1980.

[Tay 84] Y. Tay, R. Suri, "Choice and performance in locking for Databases". Proc. of 10th Conference on VLDB, Singapour, Août 1984.

[Thomas 79] R. H. Thomas, "A majority consensus approach to concurrency control for multiple copy database". ACM Transactions On Database Systems, vol 4, n° 2, juin 1979.

[Verhofstadt 78] J. Verhofstadt, "Recovery techniques for data base systems". ACM Computing Surveys, vol 10 (2), Juin 1978.

[Viemont 82] Y. Viemont, G. Gardarin, "A distributed concurrency control based on transaction commit ordering". 12th Conference on FTCS, Los Angeles, Juin 1982.

[Wazdi 83] F. Wazdi, "Etude de l'interblocage dans les systèmes répartis". Thèse de Docteur Ingénieur, Université des Sciences et Techniques du Languedoc, Octobre 1983.

[Weihl 83-a] W. Weihl, "Data dependent concurrency control and recovery". Proc. of 2nd Principles Distributed Computing Conference, Août 1983.

[Weihl 83-b] W. Weihl, B. Liskov, "Specification and implementation of resilient, atomic data types". Proc. of SIGPLAN Symposium on Programming language issues in Software Systems, 1983.

[Williams 82] R. Williams, D. Daniels, L. Haas, G. Lapis, B. Lindsay, R. Obermach, P. Selinger, A. Walker, P. Wilms, R. Yost, "R* : an overview of the architecture". Proc. of Int. Conference on Database Systems, Jerusalem, juin 1982.

[Wilms 79] P. Wilms, "Etude et comparaison d'algorithmes de maintien de la cohérence dans les bases de données réparties". Thèse de Docteur Ingénieur, Grenoble, Novembre 1979.

[Wilms 83] P. Wilms, B. Lindsay, P. Selinger, "I whish I were over there : distributed execution protocols for data definition in R*". Proc. of ACM SIGMOD Conference on Management of Data, San Jose, Mai 1983.

ANNEXES TECHNIQUES

ANNEXES TECHNIQUES

A. **Principe de quelques algorithmes de contrôle de l'accès concurrent**

B. **Exemple d'application répartie**

C. **Bases de données distribuées**
(Anatomie d'une transaction)

D. **Diagramme d'état d'un agent**

ANNEXE A

PRINCIPE DE QUELQUES ALGORITHMES DE CONTROLE DE L'ACCES CONCURRENT

A.1. VERROUILLAGE A DEUX PHASES ET PREVENTION DES INTERBLOCAGES

Une transaction notée T_j (d'estampille t_j) possède un verrou sur l'objet 0.
Une transaction T_i (d'estampille t_i) demande l'accès à l'objet 0 dans un mode incompatible avec le verrou possédé par T_j.

L'une des méthodes suivantes peut être appliquée lors du conflit (*) :

a) WAIT-DIE

Si T_i est plus vieille que T_j ($t_i < t_j$) alors T_i attend
Sinon ($t_i > T_j$) T_i est abandonnée.

b) DIE-WAIT

Si T_i est plus vieille que T_j alors T_j est abandonnée
Sinon T_i attend.

c) WOUND-WAIT

Si T_i est plus vieille que T_j alors T_j est blessée
Sinon T_i attend.

* Pour des raisons de simplicité de l'exposé, nous avons adopté une terminologie légèrement différente de celle proposée par les auteurs dans leur papier de présentation [Rosenkrantz 78].

Dans WAIT-DIE, une transaction qui provoque un conflit attend ou "se suicide". Le mécanisme est donc non-préemptif. Dans DIE-WAIT et WOUND-WAIT au contraire, une transaction peut provoquer l'abandon d'une autre transaction en cours d'exécution.

Dans WOUND-WAIT, une transaction "blessée" est autorisée à poursuivre son exécution tant qu'elle ne se met pas en attente d'une nouvelle ressource. WOUND-WAIT minimise le taux d'abandons en offrant à une transaction impliquée dans un premier conflit une chance de "survivre" si elle se termine sans être de nouveau impliquée dans un second conflit où elle devrait attendre.

Une transaction "blessée" qui entre à nouveau en conflit avec une autre transaction :

- soit blesse sa concurrente si cette dernière est plus jeune,

- soit est abandonnée si la concurrente est plus vieille.

Une transaction abandonnée libère ses ressources et doit être réexécutée (selon les cas, la réexécution peut être immédiate ou différée). Une transaction qui est réexécutée conserve la même estampille. Cette méthode lui assure un "vieillissement" relatif par rapport à ses concurrentes et augmente ainsi ses chances de succès lors de sa nouvelle exécution (absence de famine).

A2. ORDONNANCEMENT PAR ESTAMPILLAGE (Algorithme de base)

Une transaction possède une estampille qui permet de "dater" toutes les opérations qu'elle effectue. Un Objet possède une estampille en lecture et une estampille en écriture qui "datent" les dernières opérations de lecture et d'écriture qui ont été autorisées.

On note : E(T) l'estampille de la transaction,

El(0) l'estampille en lecture de l'objet,

Ee(0) l'estampille en écriture de l'objet,

E(T) désigne la date de démarrage de la transaction T et identifie de ce fait toutes les opérations effectuées par la transaction.

El(0) désigne l'estampille de la transaction la plus récente ayant lu l'objet 0.

Ee(0) désigne l'estampille de la transaction la plus récente ayant modifié l'objet 0.

Le contrôle d'accès porte sur les primitives "Lire-Base" et "Préécrire-Base" issues de l'interprétation des actions LIRE-D et ECRIRE-D. L'algorithme suivant est appliqué :

a) Lire-Base (0,E(T))

Si Ee (0) < e (T) alors exécuter la lecture

El(0) := Max(El(0), E(T))

Sinon abandonner T.

(Si l'objet n'a pas été modifié depuis le début de la transaction T, la lecture est permise).

b) **Préécrire-Base (O,E(T))**

Si El(O) < E(T)
et Ee(O) < E(T) alors exécuter la préécriture
Ee(O) := Max (Ee(O), E(T))
Sinon abandonner T.

(Si l'objet n'a été ni lu ni modifié depuis le début de la transaction T, l'opération est acceptée).

Cette technique impose qu'une transaction abandonnée soit redémarrée avec une nouvelle estampille. Elle est donc sujette à la famine.

La manipulation des estampilles associées aux objets impose une complexité et un coût supplémentaires en ce qui concerne la gestion de la mémoire permanente.

A3. CONTROLE PAR CERTFICATION (Algorithme de base)

Une transaction est exécutée en trois phases : calcul, certification, validation. Le système maintient un compteur unique (noté C) incrémenté chaque fois qu'une transaction passe l'étape de certification avec succès. Une transaction est caractérisée par trois estampilles notées T-début, T-Fin et T-val.

- T-début est la valeur courante du compteur lorsque la transaction démarre son étape de calcul.

- T-fin est la valeur courante du compteur lorsque l'étape de calcul est terminée (avant l'étape de certification).

- T-val est la nouvelle valeur du compteur lorsque la certification a réussi (avant la validation).

On note : [Ol(T)] l'ensemble des objets lus par une transaction T.
[Oe(T)] l'ensemble des objets modifiés par T.

Le test de conformité, exécuté dans l'étape de certification, consiste à vérifier que les objets lus par T n'ont pas été modifiés par une autre transaction ayant validé dans l'intervalle [T-début, T-fin].

Pour toute transaction T_i telle que T-début < T_i-val < T-fin
Faire :

Si [$O_l(T)$] $\cap$ [$O_e(T_i)$] = $\emptyset$ alors C := C + 1
T-val := C

Sinon T est abandonnée.

Une transaction abandonnée est redémarrée avec la même estampille. Cette méthode est donc aussi sujette de la famine. Par rapport à l'ordonnancement par estampillage, elle ne nécessite pas la manipulation d'estampilles au niveau des objets mais requière la gestion des ensembles [Ol(T)] et [Oe(T)] pour chaque transaction.

ANNEXE B

EXEMPLE D'APPLICATION REPARTIE

L'exemple présenté ci-dessous illustre l'utilisation des primitives décrites au chapitre 3 (§1.2) pour la programmation d'une application répartie. Cet exemple est inspiré d'une application bancaire et concerne une opération de transfert de fond entre deux comptes, consécutive au traitement d'un chèque. On suppose que les deux comptes sont localisés sur des sites distincts B et C et que le traitement du chèque est initialisé sur un troisième site A. Cette application met en jeu trois opérations.

TRANSFERT qui contrôle le traitement du chèque.

Les paramètres de cette opération sont entrés par l'utilisateur au cours d'un dialogue préliminaire. Il s'agit de :

Cc : identification du compte à créditer
Cd : identification du compte à débiter
S : somme à transférer.

On suppose que chaque site dispose d'un catalogue lui permettant d'évaluer, à partir de l'identification d'un compte, le site de résidence de ce compte.

DEBIT qui exécute le débit d'un compte.

Les paramètres d'appel de cette opération sont :

C : identification du compte à débiter
S : somme à débiter.

CREDIT qui exécute le crédit d'un compte.

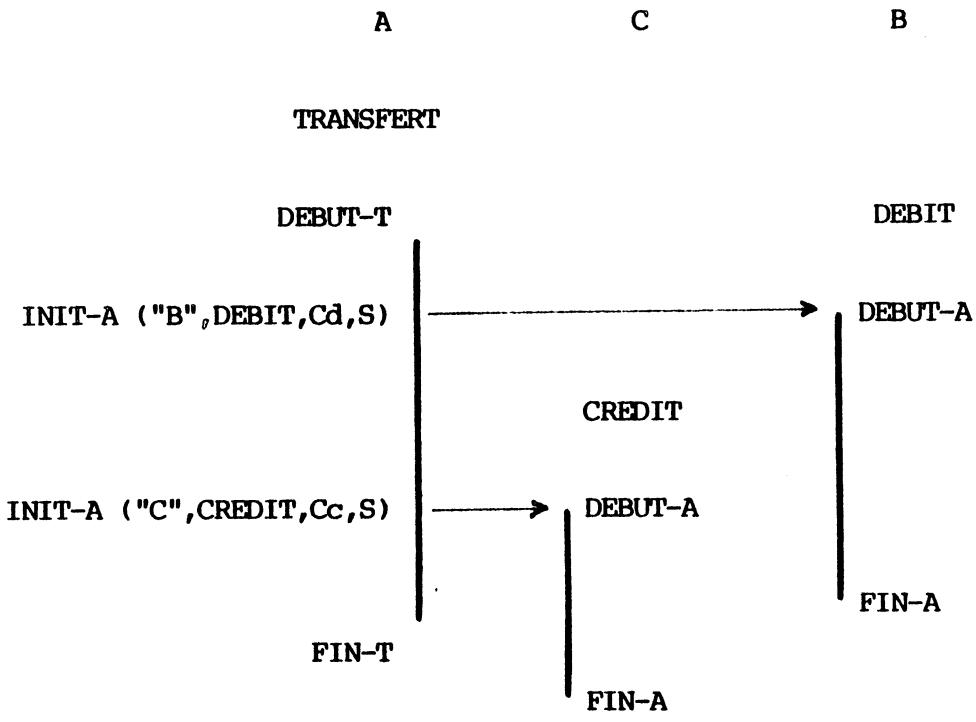
Les paramètres d'appel sont identiques à l'opération Débit.

Les programmes associés aux opérations TRANSFERT, DEBIT et CREDIT existent sur tous les sites afin de pouvoir initialiser une opération de transfert de fond depuis un site quelconque.

Une première manière de programmer ces opérations est décrite ici :

```
CREDIT : PROCEDURE (C : compte, S : réel)
              DEBUT-A
                   LIRE-D (C.solde)
                   C.solde := C.solde + S
                   ECRIRE-D (C.solde)
              FIN-A


DEBIT :  PROCEDURE (C : compte, S : réel)
              DEBUT-A
                   LIRE-D (C.solde)
                   C.solde := C.solde - S
                   Si C.solde < 0 alors ARRET-A(solde négatif)
              * la primitive ARRET-A engendre l'abandon de l'
              * agent et, par voie de conséquence, de toute
              * la transaction.
                   Sinon ECRIRE-D (C.solde)
              FIN-A


TRANSFERT : TRANSACTION
              DEBUT-T
                   Lire-Terminal (Cd, Cc, S)
                   Calculer le site de résidence de Cd
                   INIT-A ("B", DEBIT, Cd, S)
                   Calculer le site de résidence de Cc
                   INIT-A ("C", CREDIT, Cc, S)
              FIN-T
```

Cette programmation correspond au schéma d'exécution suivant :



C'est un schéma d'exécution parallèle. La synchronisation des terminaisons des transactions (non représentée ici) est effectuée par le système. C'est le rôle du protocole de validation mis en oeuvre sur chaque site lorsque la primitive FIN-A est rencontrée.

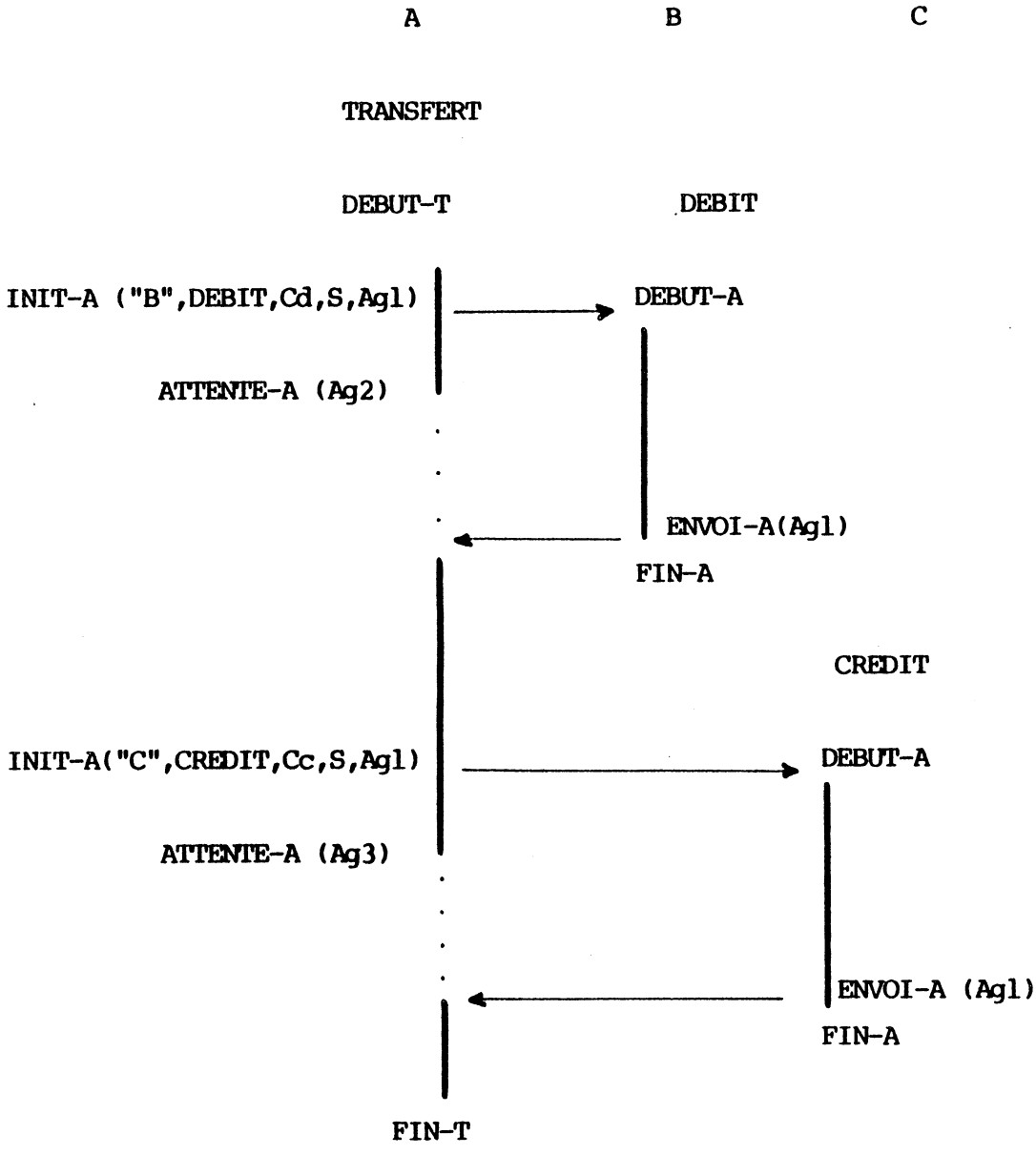
Il existe d'autres manières de programmer cette application. Nous en donnons un exemple ci-dessous.

```
CREDIT : PROCEDURE (C : compte, S : réel, Ag : id-agent)
              DEBUT-A
                   LIRE-D (C.solde)
                   C.solde := C.solde + S
                   ECRIRE-D (C.solde)
                   ENVOI-A (Ag, "ok")  * Ag est l'id. de l'agent
              FIN-A                      appelant
```

```
DEBIT :  PROCEDURE (C : compte, S : réel, Ag : id-agent)

        DEBUT-A
            LIRE-D (C.solde)
            C.solde := C.solde - S
            Si C.Solde < 0 alors ENVOI-A (Ag,"solde négatif")
            Sinon ECRIRE-D (C.solde)
                  ENVOI-A (Ag, "ok")
        FIN-A
```

```
TRANSFERT : TRANSACTION

        * Ag1, Ag2, Ag3 sont des identifications d'agents
        * Ag1 est le résultat de la primitive DEBUT-T
        * Ag2 et Ag3 sont le résultat des primitives INIT-A

        DEBUT-T
            Lire-Terminal (Cd, Cc, S)
            Calculer le site de résidence de Cd
            INIT-A ("B", DEBUT, Cd, S, Ag1)
            ATTENTE-A (Ag2)
            Si message = "ok" alors ARRET-A("solde négatif")
            Sinon
                calculer le site de résidence de Cc
                INIT-A ("C", CREDIT, Cc, S, Ag1)
                ATTENTE-A (Ag3)
                écrire-terminal (ok)
        FIN-T
```

Cette programmation correspond au schéma d'exécution suivant :



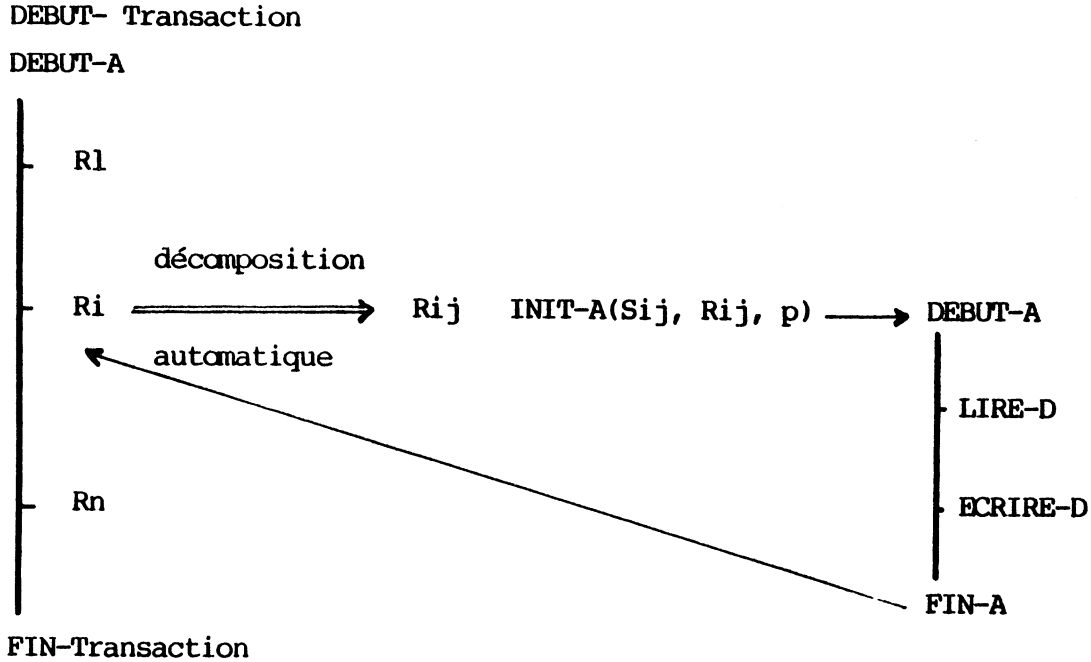
C'est un schéma d'exécution procédural contrôlé par l'agent initial.

ANNEXE C

BASES DE DONNEES DISTRIBUEES

(Anatomie d'une transaction)

Dans l'approche "Bases de Données Distribuées", une Transaction est consituée d'une ou plusieurs requêtes. L'agent initial, créé par l'action DEBUT-T analyse les requêtes. Pour chaque sous-requête issue de la décomposition d'une requête, l'agent initial demande le lancement d'un agent sur le site d'exécution de cette sous-requête.



Ri désigne le ième requête
Rij désigne une sous-requête de Ri
Sij désigne le site d'exécution de Rij

Figure C.1

Ce modèle d'exécution présente deux particularités :

- Le rôle de l'agent initial se limite à l'analyse des requêtes et au contrôle de l'exécution des agents invoqués. Il n'effectue aucune manipulation directe sur les objets.

- Le message généré par l'action INIT-Agent véhicule, non pas l'identification d'un programme, mais le contenu de la sous-requête à exécuter.

La richesse sémantique des sous-requêtes et la complexité des schémas d'exécution varient beaucoup selon les systèmes. Dans les systèmes SDD-1 et D-INGRES, l'arbre d'invocation comporte un seul niveau et les sous-requêtes sont limitées à des séquences d'actions LIRE-D, ECRIRE-D (figure C.2). Dans le système R*, la profondeur de l'arbre d'invocation n'est pas limitée mais le schéma d'exécution est restreint au contrôle procédural (figure C.3). Le système SIRIUS-DELTA autorise des scénarios plus sophistiqués. Pour chaque agent invoqué, l'agent initial créé un PEX (Plan d'EXécution). Ce PEX contient des variables globales de synchronisation qui permettent de gérer l'enchaînement entre les différents agents (conditions d'activation, dialogue, etc.). Le plan d'exécution est transporté dans le message d'initialisation (figure C.4).

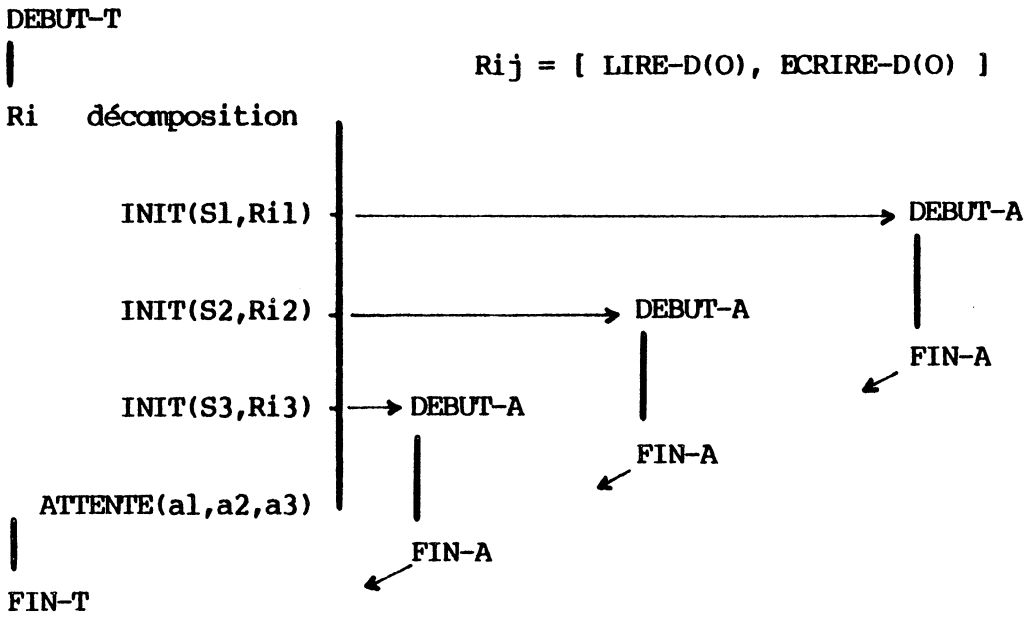


Figure C.2

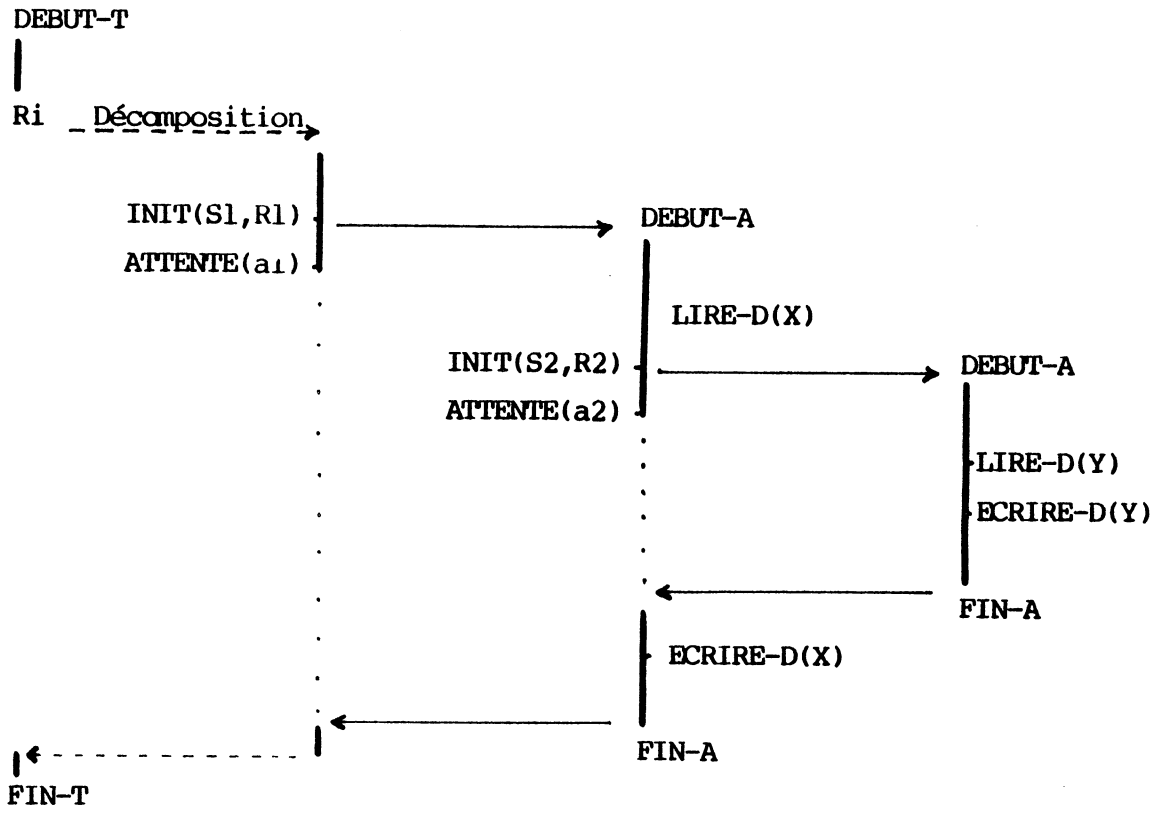
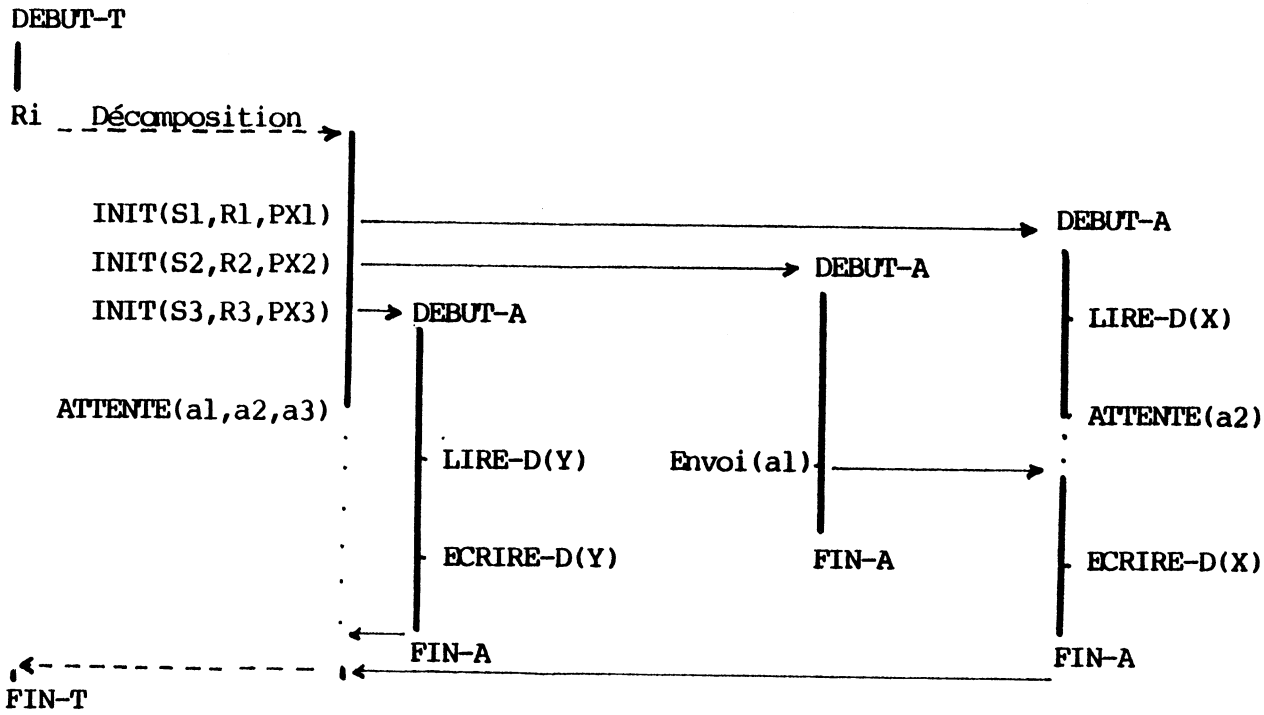


Figure C.3



PXi est le plan d'exécution associé à l'agent Ai. Dans cet exemple, PX1 et PX2 partagent une variable globale de synchronisation.

Figure C.4

ANNEXE D

DIAGRAMME D'ETAT D'UN AGENT

Le diagramme d'état d'un agent est représenté sous la forme d'une table à double entrée dans laquelle chaque élément représente la transition d'état associée à l'occurrence d'un évènement (lignes) dans un état initial donné (colonnes). Les éléments non représentés signifient que l'évènement est impossible pour l'état correspondant.

Les états possibles sont : ABSENT, ACTIF, PRET-A-VALIDER, VALIDE, TERMINE, INVALIDE et ABANDONNE

Les évènements significatifs sont les suivants :

- Primitives DEBUT-A, FIN-A, ARRET-A
- Erreurs fatales ou récupérables
- Panne locale
- Panne d'un autre élément du système distribué
- Réception d'un message 'Fin', 'Valider', 'Abandon', 'Reprise'
- Fin de l'étape de validation ou d'invalidation.

La description qui suit traite séparément le cas des agents supérieur et inférieur. Elle se limite au cas de la procédure de validation à une phase. Le lecteur trouvera dans le rapport SCOT n° 9 (Janvier 80), une description plus détaillée précisant, outre les transitions d'état, les actions principales exécutées dans chaque cas (cette description utilise le formalisme des réseaux de Nutt).

Agent Supérieur

	ABSENT	ACTIF	PRET-A-VAL	VALIDE	INVALIDE
DEBUT-A	actif				
FIN-A		actif / valide [1]			
ARRET-A		invalidé			
Erreur Fatale		invalidé			
eur récupérable		actif [2]			
Panne locale		invalidé		validé	invalidé
Panne externe		invalidé		validé	invalidé
'Fin'		actif / validé [1]			invalidé
'Valider'					
'Abandon'		invalidé			invalidé
'Reprise'		actif [2]			invalidé
in de validation				terminé	
in d invalidation					abandonné

1. **Le nouvel état est 'Validé' si et seulement si tous les messages de terminaison ont été reçus.**

2. **L'agent est réexécuté.**

Agent Inférieur

	ABSENT	ACTIF	PRET-A-VAL	VALIDE	INVALIDE
DEBUT-A	actif				
FIN-A		pret-a-val			
ARRET-A		invalidé			
Erreur Fatale		invalidé			
rreur récupérable		invalidé			
Panne locale		invalidé	? [1]	validé	invalidé
Panne externe		invalidé	? [1]	validé	invalidé
'Fin'		actif / validé			
'Valider'			validé		
'Abandon'		invalidé	invalidé		invalidé
'Reprise'		invalidé	invalidé		invalidé
Fin de validation				terminé	
Fin d'invalidation					abandonné

1. état indécidable : une enquête est nécessaire.

AUTORISATION de SOUTENANCE

'U les dispositions de l'article 5 de l'arrêté du 16 avril 1974

'U les rapports de présentation de Messieurs

. J. FERRIE, Professeur
. G. GARDARIN, Professeur
. S. KRAKOWIAK, Professeur

Monsieur Roland BALTER

:st autorisé à présenter une thèse en soutenance en vue de l'obtention du grade)OCTEUR D'ETAT ES SCIENCES.

Fait à Grenoble, le 10 juin 1985

e Président de l'U.S..I.G

1 4 JUIN 1985

Le Secrétaire Général

P. BALME

Le Président de l'I.N.P.-G

D. BLOCH
Président
de l'Institut National Polytechnique
de Grenoble

P.O. le Vice-Président,