MASTER EN SCIENCES INFORMATIQUES

Étude des mécanismes de traitements transactionnels et analyse de protocoles de coopération pour applications transactionnelles distribuées

Stas, Philippe

Award date:
1990

Awarding institution:
Universite de Namur

FACULTES
UNIVERSITAIRES
N.D. DE LA PAIX

NAMUR

INSTITUT D'INFORMATIQUE

Etude des mécanismes de traitements transactionnels et analyse de protocoles de coopération pour applications transactionnelles distribuées.

Philippe Stas

Promoteur : Professeur Jean Ramaekers

Mémoire présenté en vue de l'obtention
du titre de
Licencié et Maître en Informatique

Année académique 89 - 90

Résumé

Ce mémoire étudie de manière globale l'architecture d'un environnement transactionnel à hautes performances, supportant des applications distribuées entre sites informatiques hétérogènes. Le moniteur transactionnel à la base de celui-ci est TDS (Transactionnal Driven Subsystem) de Bull. L'analyse de cet environnement est faite par le biais d'une double approche. Au niveau des traitements locaux, l'étude est faite des concepts, des principes et des mécanismes transactionnels tels qu'ils sont envisagés par TDS. Cette étude peut servir de base de comparaison avec d'autres moniteurs transactionnels, présents sur le marché informatique. Au niveau des traitements globaux, les protocoles de coopération entre applications supportés par TDS sont analysés. Ces protocoles de coopération XCP1 et XCP2 permettent aux applications sous TDS de coopérer avec des applications supportées par les produits IMS et CICS d'IBM. Cette approche globale est complétée par une étude de la norme régissant la coopération transactionnelle interapplications telle que l'envisage l'ISO, instance internationale de normalisation, pour des systèmes ouverts.

Abstract

This dissertation deals, in a general way, with the design of a high performance transactionnal environment running applications shared between heterogeneous computer systems. The transactionnal monitor (the basis of the environment) is TDS (Transactionnal Driven Subsystem) by Bull. The analysis of this environment requires a double approach.On a local processing level, the study of concepts, principles and transactionnal mechanisms is done according to the manner in which they are taken into account by TDS. This study could serve as reference for a future comparison with other transactionnal products which are offered on the Market.On a large-scale processing level, cooperation protocols for distributed applications under TDS are analyzed. These protocols, XCP1 and XCP2 allow cooperation between applications running under TDS and applications running under IMS and CICS, both products of IBM.Finally, this global approach is completed by a study of the internationnal transactionnal processing standard proposed by ISO for the regulation of the communication between Open Computer Systems.

Avant propos

C'est avec un réel plaisir que je désirerai associer les personnes m'ayant soutenu et guidé lors de mes années d'études, et principalement durant cette année écoulée à l'accomplissement de ce travail.

Ma pensée va en premier au professeur J. Ramaekers, promoteur de ce mémoire. Je tiens à le remercier pour sa disponibilité et les conseils qu'il m'a apportés lors de mon stage et pour l'élaboration de ce travail.

Je remercie les membres de l'équipe de Bull chargée du développement de TDS: P. de Rivet, sans qui ce stage n'aurait pu avoir lieu, T.D. Luu pour l'accueil au sein de son équipe, J.M. Stéphan pour la confiance dont il a fait preuve à mon égard, toutes les personnes que j'ai cotoyées pendant ces cinq mois à Paris, dont l'enthousiasme était sans limite.

Ces études n'auraient pu s'accomplir sans professeur. Qu'ils soient remerciés pour leur patience, leur enseignement et les rapports humains qu'ils ont su conserver avec leurs élèves.

Enfin, je ne peux oublier mes proches amis, parents et Françoise qui ont su m'encourager tout au long de ces cinq dernières années.

Table des matières

Introduction

De nos jours coexistent sur le marché informatique de multiples utilitaires permettant, selon les concepteurs de ces programmes, d'élaborer et de gérer des applications transactionnelles "temps réel", dans un environnement fiable. Bon nombre d'entre eux, proposés en complément de gestionnaires de bases de données, possèdent des limites fonctionnelles.

Les principales limites concernent le nombre relativement faible d'utilisateurs pouvant être connectés en parallèle sur une application ainsi que les temps de réponse aux terminaux et de recouvrement en cas de pannes ou de problèmes, élevés.

De plus, la plupart de ces utilitaires souffre d'une incapacité à faire coopérer les applications dont ils ont la charge avec d'autres applications disponibles sur des sites distants ou non, hétérogènes ou non.

Par le biais de mon stage de fin d'études chez Bull à Paris, j'ai été amené à participer au développement d'un utilitaire répondant à ces critiques limitatives. Cet utilitaire est le **moniteur transactionnel de Bull, TDS**.

1. But du stage:

Le travail dont j'ai eu la charge, au sein de l'équipe chargée de l'élaboration du nouveau moniteur transactionnel de Bull, TDS/V6, a été de développer un module permettant la prise en charge des fonctions et commandes externes (utilisateur) du moniteur transactionnel actuel de Bull, TDS/V3, par le nouveau moniteur transactionnel TDS/V6. Ce travail d'intégration s'est déroulé en trois phases: tout d'abord une approche globale du "système transactionnel" de Bull, ensuite une étude détaillée des interfaces externes et des fonctionnalités des commandes de contrôle des deux moniteurs transactionnels, et enfin l'élaboration du module d'interface.

Le but recherché, à travers l'élaboration de ce module est de permettre une installation rapide et optimale du nouveau moniteur transactionnel TDS/V6, dans les sites sur lesquels est implémenté l'ancien moniteur transactionnel de Bull TDS/V3.

L'existence des deux environnements externes, celui de TDS/V3 et de TDS/V6, au sein de TDS/V6 permettra non seulement d'éviter les réticences et les rejets des utilisateurs potentiels du nouveau moniteur transactionnel de Bull (les plus réticents continueront à utiliser l'ancien environnement externe transactionnel), mais aussi d'habituer peu à peu ces utilisateurs aux fonctionnalités nouvelles du moniteur transactionnel TDS/V6, par ailleurs bien plus performant que TDS/V3.

2. Objectifs du mémoire:

Si la conception de ce module, fruit de ce stage au sein de l'unité de développement transactionnel de Bull, représente une expérience de travail intéressante dans un milieu de production logiciel, elle comporte néanmoins peu d'intérêts didactiques.

Toutefois, l'étude de l'environnement transactionnel de Bull m'a amené à constater une certaine carence des publications quant à une approche globale des moniteurs transactionnels et aux possibilités de coopérations interapplications que ces moniteurs peuvent offrir aux applications qu'ils supportent. Cette carence pourrait être expliquée non seulement par la complexité de ces utilitaires, mais aussi par les politiques plus ou moins protectionnistes des concepteurs de ces moniteurs transactionnels.

Sur base des informations disponibles au sein de la section transactionnelle de Bull et de l'expérience acquise durant ce stage, je me propose d'élaborer une telle approche globale pour TDS.

Une étude des mécanismes principaux et de l'architecture interne de base du moniteur transactionnel de Bull sera faite, ainsi que des possibilités qu'il offre au niveau de la coopération interapplications.

3.Contenu du mémoire:

Ce mémoire comportera deux parties distinctes:

La première, composée des six premiers chapitres, proposera une étude de l'environnement "système" et des techniques permettant au moniteur transactionnel TDS de remplir sa fonctionnalité de support fiable et performant d'applications transactionnelles.

Le **premier** chapitre sera consacré à l'étude de l'environnement et des ressources du moniteur transactionnel TDS.

Le chapitre **deux** aura pour but de définir les concepts transactionnels de base.

Un aspect essentiel de l'environnement transactionnel sont les possibilités d'accès aux bases de données et aux fichiers mis à la disposition du moniteur transactionnel et des applications qu'il supporte. Le chapitre **trois** proposera d'étudier les mécanismes de protection des données qu'ils contiennent afin de sauvegarder leur intégrité en cas de problèmes "hardware" ou "software" susceptibles de survenir.

Les deux derniers chapitres de cette première partie, les chapitres **cinq** et **six** seront consacrés à l'étude des mécanismes assurant la cohérence respectivement locale et globale des traitements transactionnels.

Dans la seconde partie, composée des chapitres sept à neuf, une étude de protocoles nécessaires à une coopération entre applications supportées par TDS avec d'autres applications s'exécutant sur des supports hétérogènes sera faite.

Ainsi, les protocoles de coopération XCP1 et XCP2 (eXtended Coopérative Protocol 1 et 2) de Bull, ont été élaborés afin de permettre des dialogues coopératifs entre applications supportées par TDS et des applications supportées par les produits IMS et CICS d'IBM. Ces deux protocoles seront les sujets des chapitres **sept** et **huit**.

Cette étude ne se limitera toutefois pas à ce cas de figure. Ainsi, l'aspect protocolaire au niveau des instances internationale sera abordé dans le chapitre **neuf**, à travers l'analyse de la future norme 10026 de l'ISO régissant la coopération entre applications pour systèmes ouverts.

Note

La majorité des informations nécessaires à l'élaboration de ce travail proviennent de documents internes à la compagnie Bull.

Ces documents, pour la plupart sont des avant-projets, des projets, des spécifications fonctionnelles de produits et de protocoles informatiques disponibles au niveau de l'unité de développement du moniteur transactionnel de Bull .

Une référence précise des informations contenues dans ce mémoire à ces documents ne peut être faite.

Aussi me suis-je proposé de référencer de manière globale les différentes parties de ce mémoire. Les ouvrages consultés seront donnés à la suite du titre des chapitres, des sections de chapitres .....

La liste des références complètes des documents sera proposée en annexe.

Partie 1 : Mécanismes transactionnels

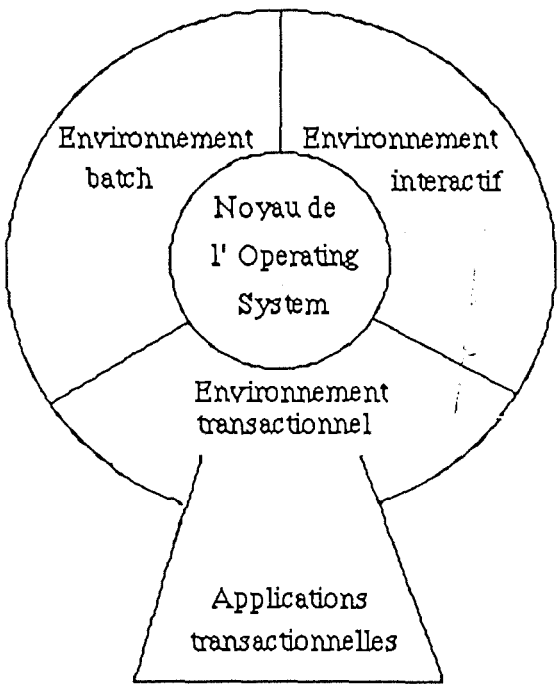
1. Présentation

1.1 Introduction

[B TDSC 89] [B TDS 89]

Le moniteur transactionnel dans l'environnement informatique est un sous-système particulier. L'ensemble de ses activités du point de vue fonctionnel est divisé en deux parties. Une première partie dont les fonctions s'apparentent à des activités de l'Operating System (O.S.) et une seconde se rapprochant plus de l'aspect "application" développé autour de ce même système.

On pourrait situer le champ d'action du moniteur transactionnel comme suit par rapport à l'environnement global de l'Operating System.



(Fig 1.1.)

L'environnement transactionnel est considéré comme un "monde" à part entière au côté du "monde" Batch et du "monde" interactif. Pour l'Operating System, une séparation existe entre ceux-ci ainsi qu'une indépendance fonctionnelle.
Ces trois environnements chez Bull coexistent au sein de l'Operating System développé par la Compagnie.

1.2 Présentation de l'environnement transactionnel

Dans une telle architecture informatique, le moniteur transactionnel est un sous-système qui fournit un environnement sécurisé et performant aux utilisateurs et aux développeurs d'applications "on line", c'est-à-dire orientées "temps réel".

Le champ d'action du moniteur transactionnel peut être scindé en deux parties: une partie applications et une partie contrôle de ces applications.

1) Applications

Cette partie concerne l'application ou les applications de gestion développées par des programmeurs, exécutées sous contrôle étroit du moniteur transactionnel. Celles-ci sont orientées surtout pour répondre à des besoins transactionnels présents sur le marché informatique et émanant principalement de sociétés dont l'activité globale repose essentiellement sur une activité "temps réel d'échanges d'informations" intense, sécurisée et disponible à tout moment (banques, assurances, entreprises publiques,...). Le code de ces applications permet l'exécution de transactions répondant à la demande d'utilisateurs connectés sur ces mêmes applications.

Ces applications sont développées dans des langages de haut niveau (Cobol, C, HPL) disponibles. Il est à noter que la disponibilité de tels langages ne limite pas le champ d'activité des programmeurs aux seules applications transactionnelles. D'autres champs d'application sont susceptibles d'être abordés dans un tel environnement.

Le contenu de cette première partie est fortement lié aux besoins spécifiques dictés par l'activité transactionnelle de ou des société(s) utilisatrice(s) du programme d'application.

2) Contrôle des applications

La deuxième partie est celle qui nous intéresse. Elle constitue le coeur de l'environnement transactionnel chez Bull, c'est le moniteur transactionnel en tant que tel: TDS.

TDS est un support essentiel pour un programmeur d'applications transactionnelles, en ce sens qu'il le libère de lourdes contraintes liées à la gestion des communications, à la gestion d'intégrité des données, ainsi qu'à la gestion des erreurs et de leur recouvrement. Ces contraintes sont essentielles dans un environnement transactionnel performant à haute disponibilité .

Concrètement, TDS se charge de :

- la gestion des communications de l'application qu'il supporte, tant au niveau des utilisateurs (terminaux) qu'au niveau des échanges de données avec des applications tournant sur systèmes hétérogènes.

- la gestion des sessions utilisateurs de l'application ainsi leur indépendance vis-à-vis l'une de l'autre et des autres utilisateurs de la machine. TDS peut gérer en toute sécurité jusqu'à 10.000 sessions "utilisateurs" connectées sur une même application.

- la gestion des interblocages résultant de l'accès simultané des programmes de l'application sur des données, exécuté pour répondre aux sollicitations de ces nombreux utilisateurs.

- la gestion des erreurs et des exceptions provenant de l'OS ou du code de l'application, sans toutefois arrêter cette dernière. Le traitement de ces problèmes est dynamique.

- la gestion des reprises automatiques et dynamiques de l'environnement transactionnel et de l'application en cas de problèmes graves nécessitant la réinitialisation de certains composants vitaux du système d'exploitation ou du sous-système transactionnel; les problèmes les plus graves étant la panne système ou la panne "média".

  Ces reprises doivent aboutir à la restauration d'un état d'exécution cohérent sans que ni la validité des données, contenues dans les bases de données de l'application et dans les fichiers de travaux, ni la validité des résultats ne soient altérées.

- l'optimisation des temps de réponse aux injonctions des utilisateurs. Pour ces derniers, les services qu'ils demandent (débit ou crédit sur un compte de banque par exemple) doivent être exécutés immédiatement ou dans un délai très raisonnable, à l'échelle humaine. Le temps utilisé par le moniteur transactionnel et par l'application, pour satisfaire une demande d'un utilisateur dans n'importe quel cas de figure, représente la mesure réelle de l'efficacité de l'environnement transactionnel en question.

L'ensemble des services qu'offre le moniteur transactionnel ne relève pas uniquement de son code. Son activité repose en grande partie sur les fonctionnalités de l'O.S. des gros systèmes BULL, GCOS7 .

1.3 Présentation de l'Operating System

La description succincte de l'Operating system propriétaire de BULL peut être la suivante:

Le système d'exploitation GCOS7 (General Comprehensive Operating System) est développé chez Bull pour une classe d'ordinateurs de haute puissance, allant du plus petit au plus gros DPS7 (Distributed Processing System), modulables et multiprocesseurs. C'est un système temps partagé supportant une architecture distribuée (DSA, Distributed System Architecture) offrant de nombreuses fonctionnalités.

Parmi ces fonctionnalités, citons

* Au niveau des télécommunications:

  Des possibilités d'accès aux réseaux publics et privés via DSA. Les possibilités de connexions via DSA sont étendues. En effet, DSA est non seulement conforme au modèle de référence pour l'interconnexion des systèmes ouverts tel que le définit l'Organisme de Normalisation International, (l'ISO), mais fournit aussi un accès au monde IBM basé principalement sur SNA.

* Au niveau des données et des accès fichier:

- Une gestion des fichiers avec contrôle de partage et de l'intégrité.
- Un système de gestion de base de données relationnelle IDS II (Integrated Data Store II) et des possibilités d'intégration d'autres systèmes de gestion de données comme Oracle.

* Au niveau des traitements:

- Un système IOF (Interactive Operation Facility) ou moniteur interactif, un système Batch.
- Un système transationnel TDS (dont les mécanismes seront traités dans ce mémoire).
- Tous les services de gestion des processus, des sessions utilisateur, de mémoire vive. de travaux nécessaires.
- Des mécanismes d'accès et de sécurité requis pour un tel système, des gestionnaires de recouvrement en cas de problèmes des ressources sous son contrôle.

* Au niveau des applications:

- des processeurs de langages évolués, des éditeurs, des utilitaires de trace ("debugger") etc.

Par rapport à d'autres systèmes d'exploitation, GCOS7 s'inscrit dans la lignée des gros gestionnaires de ressources comme VAX/VMS de Digital, BS2000 de Siemens, VM/CMS d'IBM.

1.4 Présentation des Ressources de l'operating system utilisées par le moniteur transactionnel TDS.

1.4.1 Introduction.

L'ensemble des ressources dont dispose le moniteur transactionnel TDS provient de l'"operating system" propriétaire Bull, GCOS7, plus particulièrement des modules le composant. La suite de ce chapitre a pour but de présenter les plus importants de ces modules et les services qu'ils offrent au moniteur transactionnel. Les services de ces modules sont là pour répondre chacun en partie aux fonctionnalités et aux besoins du moniteur transactionnel. Ceci est particulièrement vrai pour la gestion de la cohérence des données, et pour la gestion de l'aspect télécommunication du transactionnel, toutes deux vitales pour la sécurité et les performances du moniteur transactionnel.

1.4.2 Data Manager et Contrôleur général d'accès concurrent.

[B GAC 89] [GAC 84] [LUG 86] [ROSL 78] [SCOT 81]

1.4.2.1 Introduction.

Le gestionnaire de données "Data Manager" et le Contrôleur général d'accès concurrent GAC (General Accès Controler) sont directement liés à travers la complémentarité des fonctions qu'ils offrent au moniteur transactionnel. Tous deux sont à la base des accès et de la gestion cohérente des données que peut manipuler TDS. Notons que d'autres gestionnaires de données peuvent être mis à la disposition de TDS, comme Oracle. Dans ce cas, la gestion des données est supportée par Oracle. Voyons à présent les fonctions des deux utilitaires qui nous intéressent.

1.4.2.2 Data manager.

Ce module s'occupe de fournir des fonctions de création, de modification, de mise-à-jour des données contenues dans des fichiers. Il fournit au moniteur transactionnel une vue des fichiers, où une abstraction est faite de la gestion des données, de la méthode d'accès à ces données, de la méthode de transfert disque/mémoire vive, de la gestion des mémoires cache. Par son biais, TDS a non

seulement accès à des données stockées dans des fichiers accédés de manière traditionnelle, fichiers séquentiels ou indexés mais aussi à des données contenues dans des bases de données relationnelles (sous le système relationnel IDS2).

1.4.2.3. Contrôleur général d'accès concurrent (GAC).

1.4.2.3.1. Présentation.

La fonction du GAC est de surveiller les accès concurrents aux données que divers programmes, s'exécutant en parallèle pour une ou plusieurs applications, pourraient réaliser. Les performances de ce module sont un paramètre important pour le bon fonctionnement "temps réel" du moniteur transactionnel ainsi que pour les applications qu'il supporte. Ainsi, si le délai d'accès aux données est lent, l'attente se répercute au niveau de l'application supportée, et à fortiori, sur le temps d'attente de l'utilisateur connecté sur l'application, derrière son terminal.

D'une manière générale, le contrôleur général d'accès concurrent contrôle l'accès des données partagées, celles-ci pouvant être des fichiers, ou non.

Les travaux en temps réel nécessitent un temps de réponse très rapide pour leurs accès des fichiers partagés, il serait donc inacceptable de réserver la totalité d'un fichier en exclusivité à un programme d'application alors que d'autres sont en attente de ce même fichier.

Le contrôleur général d'accès concurrent permet de gérer de tels conflits de concurrence. Les "objets" susceptibles d'être alloués en exclusivité d'accès aux programmes d'applications sous TDS ne sont plus, comme dans la plupart des "Operating Systems" des fichiers, mais bien des " intervalles de contrôle", sorte d'enregistrements de taille fixe, de ces fichiers. GAC opère donc sur une échelle de granulosité plus précise que celles des "l'operating systems".

En particulier, le contrôleur général d'accès concurrent gère:

1) les accès aux fichiers séquentiels, relatifs, séquentiels indexés, aléatoires.
2) les situations de conflits d'attente sur ressources bloquées, ou des problèmes d'interblocage.
3) un seul protocole de blocage basé sur le principe de l'unité de Commit (voir point 1.4.2.3.2.).
4) des ressources associées aux fichiers et définies par la "méthode d'accès" comme des enregistrements ou des intervalles de contrôle.

1.4.2.3.2. Unité de Commit.

Pour le contrôleur général d'accès concurrent , une transaction peut être vue comme une séquence de références ou de modifications de données de fichiers, comme une unité de Commit.

Durant la durée de l'exécution de l'unité de Commit, les ressources doivent être cohérentes non seulement pour le détenteur de ces ressources, mais aussi pour les autres unités de Commit. Les données accédées durant cette "phase critique" par les unités de Commit concernées doivent être soumise à un verrouillage approprié.

Deux modes de verrouillage des données coexistent. Ce sont un mode Exclusif où seul une Unité de Commit "voit" les données, et un mode Partagé où plus d'une Unité de Commit peuvent accéder à ces données.

Ceci n'est pas suffisant pour garantir un accès cohérent et optimum aux données. Une politique de détection et de prévention des interblocages sur les données accédées doit être menée afin d'éviter les situations critiques d'attentes pour les performances du moniteur transactionnel. Des exemples précis d'interblocages classiques sont donnés par Date [DATE 82] au chapitre trois de son livre "An introduction to Database System".

Si l'unité de Commit exécute correctement son traitement, les données qui lui ont été réservées seront libérées et accessibles aux autres unités de Commit. La valeur des données est le résultat uniquement des traitements que cette dernière unité de Commit a exécuté.

Si, l'unité de Commit ne pouvait arriver à la fin de son exécution,de manière correcte, les ressources seraient libérées dans leur état avant verrouillage. Le mécanisme global de recouvrement des données en cas de problèmes sera détaillé dans le chapitre cinq de ce travail.

Comme cela a été dit auparavant, les performances du GAC sont un élément essentiel des performances globales du moniteur transactionnel et de la cohérence des données qu'il accède. Aussi n'est-il pas sans intérêt de décrire les mécanismes fonctionnels de base du GAC.

1.4.2.3.3 Politique de détection et prévention des interblocages.

Gac, pour répondre à sa fonctionnalité, mène une politique de détection et de prévention des interblocages.

Celle-ci peut être décrite comme suit:

Quand une unité de Commit T1 doit attendre un granule g déjà verrouillé dans un mode incompatible (fig 1.2),

Requête \ Verrou courrant	Partagé	Exclusif
Partagé	Compatible	Conflit
Exclusif	Conflit	Conflit

(Fig 1.2.)

le contrôleur général d'accès concurrent permet de gérer cette attente en prévenant toutes les situations d'interblocage.

Lors d'un conflit sur un granule g,

*si l'unité de Commit T1 n'a pas encore été tuée,

un mécanisme de détection d'interblocage est activé. Si un interblocage est effectivement détecté, T1 est avortée, toutes les ressources qu'elle a déjà acquises sont libérées et elle est mise en attente. Elle ne sera réactivée qu'à la libération du granule g réclamé et T1 recommencera au début de son code.

* si l'unité de Commit T1 a déjà été tuée,

c'est alors un mécanisme de prévention qui est activé. Celui-ci est basé sur l'algorithme "Die-Wait", proposé entre autres par Rosenkrantz, Stearns et Lewis [ROSE 78] .

Selon une étude sur les performances menées par J.P. Luguern [LUG 86] sur base d'un programme de recherche BULL du groupe SCOT (SCOT = System pour une Coopération Cohérente des Transactions) [SCOT 81] , cet algorithme assure un temps de réponse minimal des requêtes transactionnelles par rapport à d'autres méthodes de gestion et de prévention des interblocages. Elle a l'avantage de privilégier les transactions les plus vieilles, ce qui assure une répartition des possibilités d'accès équitable entre les unités de Commit concurrentes.

1.4.2.3.4 Description de l'algorithme de verrouillage utilisé.

La description détaillée de l'algorithme "Die-Wait" utilisé a été inspirée de travaux de J.P. Luguern [LUG 86]

Vocabulaire général.

Une unité de Commit T est assise sur un granule g si T a verrouillé g et ne l'a pas encore déverrouillé.

Un granule g donné peut se trouver dans trois états différents:

- g est dans l'état IDLE si aucune unité de Commit n'y est assise.
- g est dans l'état PASSIVE si une ou plusieurs unités de Commit y sont assises et si aucune unité de Commit n'est dans l'attente du verrouillage de g.
- g est dans l'état WAIT si une ou plusieurs unités de Commit y sont assises et si une unité de Commit est en attente du verrouillage de g.

Les deux modes d'accès autorisés sur un granule selon la définition de Rosenkrant sont : d'une part un mode Partagé
et d'autre part un mode Exclusif.

Règle générale.

Une transaction ne peut être mise en attente que de la terminaison ou de l'avortement de transaction strictement plus vieille qu'elle.

Règle d'estampillage

Les transactions se voient affecter des estampilles strictement croissantes lors de leur initialisation. Ainsi toutes les estampilles sont distinctes et plus une transaction est jeune, plus son estampille est grande.

Stratégie de prévention.

Cette stratégie se présente comme suit:

Supposons qu'une transaction Ti d'estampille i demande à verrouiller un granule g déjà verrouillé dans un mode incompatible par une autre transaction Tj (d'estampille j). Dans ce cas, et dans l'optique "Die-Wait": si Ti est plus vieille que Tj (Ti<Tj), alors Tj est avortée sinon Ti attend.

File d'attente.

Pour cet algorithme, une transaction Tj tuée peut toujours être mise en attente du granule qui a créé le conflit. En effet, l'unité de Commit Ti, qui l'a tué et qui possède g, est nécessairement plus vieille qu'elle. La règle générale permet donc à Tj d'attendre Ti.

Une seule file d'attente des unités de Commit (FWAIT) existe pour l'algorithme Die-Wait. Elle est ordonnée selon les estampillages décroissants de ces unités de Commit. Notons que cette file d'attente est limitée en taille et si une nouvelle transaction doit être mise en attente alors qu'il n'y a plus de place dans cette file, un avortement de la transaction doit être effectué.

Après déverrouillage d'un granule g, les plus vieilles transactions sont élues en premier.

Description détaillée de l'algorithme de demande de verrouillage: Req (Tx,m,g).

Soient: - Tx, demandant à verrouiller g dans un mode m.
-Assis, l'ensemble des transactions assises sur le granule g,
alors:

1) Si g est dans l'état IDLE, cette requête n'entre en conflit avec aucune autre sur g. Tx est alors autorisée à verrouiller g.

2) Si g est dans l'état PASSIVE, Assis est non-vide. Il contient : soit n transactions ayant verrouillé g en partage, soit une seule transaction l'ayant verrouillé de manière exclusive.

Cas n°1: "n accès partagés assis".

P1:

Si m = Partagé alors Tx peut verrouiller g en lecture pour son propre compte, il n'y a pas de conflit.

P2:

Si m = Exclusif alors il y a conflit effectif:

- Si Tx est plus jeune que tous les accès partagés, elle est mise en attente dans la file FWAIT.

- Si Tx est plus vieille qu'au moins un accès partagé assis: alors, il peut s'agir d' une seconde visite de g par Tx (Tx étant déjà un élément de Assis) alors :

- soit Tx est la seule transaction assise et elle est autorisée à verrouiller g en exclusif.

- soit Assis contient plus d'une transaction. Tx ne peut être mise en attente de sa propre terminaison. Une seule occurrence de Tx ne peut être reprise dans Assis.
  Le "Die-Wait" n'autorise pas les suicides mais seulement les meurtres des transactions plus jeunes que soi. S'il y a des transactions plus vieilles que Tx, Tx ne peut de toutes façons pas verrouiller g. L'option de tuer Tx est prise quitte à faire ainsi une entorse à la règle générale du "Die-Wait". On remarque cependant que cette entorse n'a aucune incidence ni sur le fonctionnement général de l'algorithme, ni sur le respect de la règle générale.

- Si Tx est plus vieille qu'au moins un accès partagé assis et s'il ne s'agit pas d'une seconde visite de g par Tx, Tx doit tuer toutes les transactions assises plus jeunes qu'elle. Ce n'est que dans ces conditions qu'elle pourra être mise en attente sur g .

Cas n°2: "Un accès exclusif assis"

P3: Que m soit partagé ou exclusif, on suit l'algorithme classique:

-Si Tx est plus jeune que la transaction Ta, Tx doit être mise en attente dans FWAIT;
-Si Tx est plus vieille que la transaction assise Ta, Ta doit être reprise;
-S'il s'agit d'une seconde visite et Tx étant la transaction assise, Tx est autorisé à réexécuter une mise-à-jour ou un accès partagé de g.

3) Si g est dans l'état WAIT, Assis et FWAIT sont non vides. De même que précédemment, Assis contient : soit "n accès partagés assis", soit "un accès exclusif assis".

Cas n°3: " n accès partagés assis"

P4:

Si m = Partagé, alors la transaction Tx n'entre pas en conflit direct avec les transactions assises. Cependant, FWAIT est non vide. La règle générale doit être respectée. Notons tout d'abord que si Ti est la plus vieille des transactions de FWAIT, alors Ti a demandé de verrouiller g en exclusif; remarquons ensuite que dans le cas contraire, Ti aurait été assise sur g.

Ainsi: - si Tx est plus vieille que Ti, elle est autorisée à s'assoir puisqu'étant plus vieille que toutes les transactions de FWAIT;
- s'il s'agit d'une seconde visite et Tx étant déjà un élément assis, Tx est autorisée à effectuer une nouvelle lecture de g;
- si Tx est plus jeune que Ti, elle ne peut s'assoir car Ti attendrait alors la transmission plus jeune qu'elle, mais elle peut être mise en attente dans FWAIT. En effet, toutes transactions assises étant plus vieilles que Ti sont, à fortiori, plus vieilles que Tx.

P5:

Si m=Exclusif, alors Tx est en conflit direct avec les transactions assises et la file FWAIT n'intervient plus. Ce cas est identique à P2.

Cas n°4: "un accès exclusif assis".

P6:

Que m soit partagé ou exclusif, Tx entre en conflit direct avec les transactions assises. On applique ici l'algorithme du "Die-Wait" classique. Ce cas est identique à P3.

1.4.2.3.4 Précisions

Dans le cas où la demande et l'attente d'un granule sont rejetées, l'unité de Commit doit être avortée et ses ressources libérées. Une reprise de son exécution au début de l'unité de Commit pourra être opérée dès que la ressource est libre.

Comme la gestion de l'unité de Commit n'est pas du ressort du contrôleur général d'accès concurrent, c'est les produits ayant demandé une protection de leurs ressources par le contrôleur général d'accès concurrent qui s'en chargent. Ces produits déterminent eux-même les actions appropriées ainsi que la marche à suivre. Ce qui est le cas de TDS. Le contrôleur général d'accès concurrent donne quant à lui les informations nécessaires à la résolution du conflit.

1.4.2.3.5 Limitation des attentes

La durée d'attente pour les unités de Commit, sous moniteurs transactionnels peut être limitée. Un temps limite ("longwait") au bout duquel, si la demande d'une unité de Commit n'a pu être honorée, peut être déterminé. Au bout de ce labs de temps, le contrôleur général d'accès concurrent renvoie le contrôle à TDS, qui choisit la marche à suivre. Cette démarche est généralement un avortement de l'unité de Commit. Il utilisera ce processus ainsi libéré pour effectuer d'autres travaux. Les verrous ne seront pas relâchés et lorsque la ressource redevient disponible, le contrôleur général d'accès concurrent notifiera à TDS le fait qu'il puisse redémarrer l'unité de Commit en question.

1.4.2.3.6 Liste des verrous

Il existe un nombre maximum de verrous contenus dans les tables gérées par le contrôleur général d'accès concurrent. A chaque unité de Commit, un nombre maximum de verrous est autorisé. En cas de dépassement, l'unité de Commit peut que suspendre son traitement ou avorter. Dans ce cas, elle garde la possibilité d'être relancée plus tard avec une garantie supérieure d'entrées dans la table.

1.4.2.3.7 Moyens de communication TDS / GAC

La communication entre le contrôleur général d'accès concurrent et le moniteur transactionnel s'effectue dynamiquement à l'aide d'un sémaphore à messages. Ce sémaphore est "monté" par GAC lorsqu'une information doit être transmise à TDS, il est dès lors pris en compte par TDS pour ses traitements et vice versa.

1.4.3 Gestionnaire des sessions utilisateurs (VCAM :Virtual Control Acces Manager)

[B TDST 89]

1.4.3 1 Rôle.

Ce module a la charge d'établir des sessions de communication entre des applications et/ou entre des applications et des terminaux ainsi que de gérer la bonne transmission des messages échangés sur ces sessions.

Le gestionnaire des sessions permet aux moniteurs transactionnels de faire communiquer leurs applications in abstracto de la nature du réseau, de la localisation des applications et des terminaux, de la présentation des données ou enfin des systèmes sur lesquels tournent ces applications. Précisons que chaque session n'engage que deux correspondants identifiés de manière univoque.

La qualité du module de communication constitue une garantie quant aux possibilités et aux conditions de distribution des applications transactionnelles supportées par un moniteur transactionnel. VCAM est un modèle du genre. BULL, très tôt, a compris l'intérêt d'adapter son module de communication et les protocoles qu'il supporte aux normes internationales de l'ISO, s'assurant par là même une grande capacité d'interconnexion avec des systèmes hétérogènes.

Pour des raisons commerciales, Bull a de plus complété sa capacité de communication avec les produits IBM en élaborant une passerelle vers SNA.

1.4.3 2 Objets connectables.

Chaque application tournant sous TDS est connue de VCAM sous la forme d'une station de travail à laquelle est attaché une ou plusieurs boîtes aux lettres servant au stockage et au transfert des messages entre applications ou entre applications et terminaux. Deux boîtes aux lettres d'applications coopérant sont reliées entre elles par une session. Plusieurs sessions sont allouables à une station de travail permettant ainsi la communication en parallèle des programmes de l'application.

1.4.3.3. Interface VCAM-Applications.

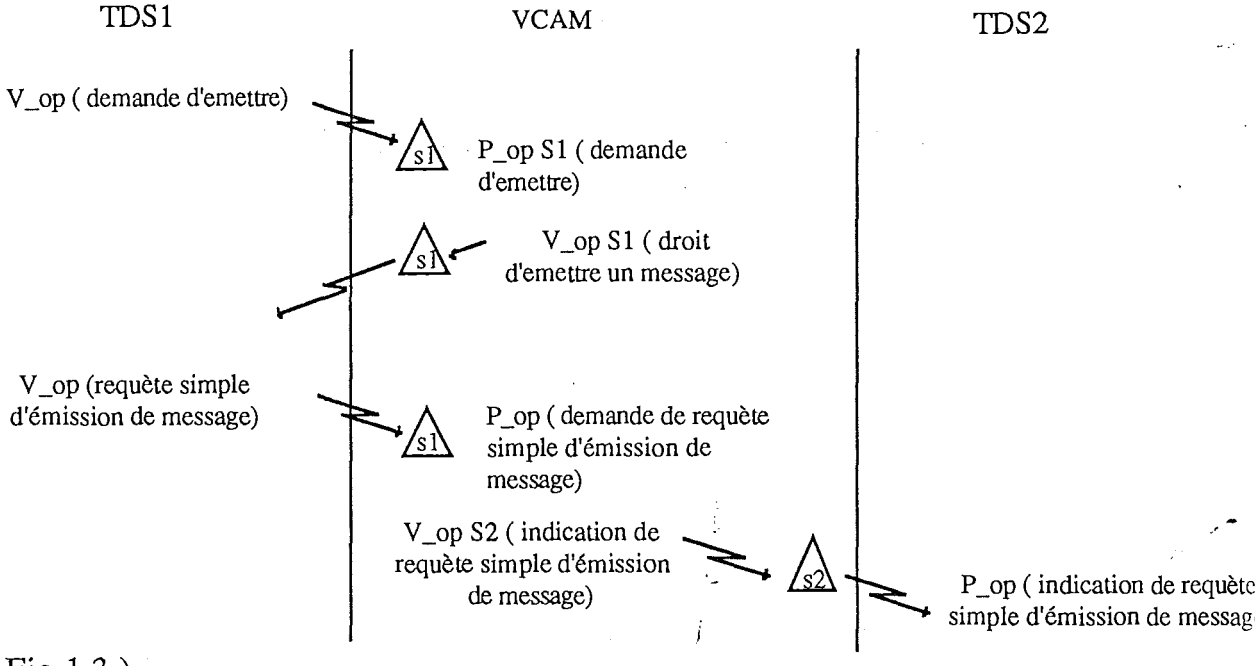
Du point de vue de VCAM, l'application est guidée par des événements qui lui sont renseignés via le moniteur transactionnel par un sémaphore à messages propre au moniteur transactionnel supportant l'application. Ce sémaphore joue un rôle similaire à celui disposé entre GAC et TDS. Ainsi, ce sémaphore à message permet à VCAM de piloter les communications de TDS et à TDS d'informer VCAM sur ses intentions de communiquer.

TDS a la possibilité de communiquer sous deux modes: soit sous un mode synchrone, soit sous un mode asynchrone.

Mode asynchrone.

Dans ce mode, l'accès télécommunication de l'application n'est pas bloqué pendant l'exécution de la requète.

Prenons l'exemple d'une requète simple d'envoi de données. La séquence suivante de dialogue à l'aide des sémaphores comprend une demande d'émettre du moniteur transactionnel 1, une autorisation d'émettre donnée par VCAM, et une indication de transmission de données de VCAM à destination du moniteur transactionnel 2.



(Fig 1.3.)

Mode synchrone.

Dans le mode synchrone, les possibilités de communication de l'application via TDS restent bloquées tant que VCAM n'a pas répondu de manière complète à la requète qui lui a été soumise .

Pour une requète simple, l'attente de l'application correspondra au temps nécessaire pour l'acceptation de l'émission de la requète par VCAM, plus le temps de transmission effectif de l'indication de cette requète au correspondant.

Pour une requète avec réponse, on doit y ajouter le temps de traitement de l'application correspondante et le temps de transmission de la réponse.

Durant ce temps, toutes les requêtes suivantes sont interdites. La logique pour le système transactionnel pousse à utiliser le mode asynchrone pour des raisons évidentes de disponibilité de ressources.

1.4.3.4 Fonctionnalités des sémaphores.

En fait, deux types de sémaphores sont utilisées par le moniteur transactionnel et par VCAM pour renseigner des événements à l'un et à l'autre.

Le premier type de sémaphores est relatif aux stations de travail.

Il permet à TDS:

- d'ouvrir le réseau de télécommunication (requète V-startup)
- de fermer ce réseau (requète V-shutdown)
- de recevoir des indications d'événements de VCAM (requêtes V-event, V-openreq...).
  Ces événements se déroulent en mode synchrone ou asynchrone. Un seul sémaphore de ce type existe par station de travail.

Le deuxième type est relatif aux sessions. Les évènements postés sur ces sémaphores visent directement les applications transactionnelles . Les possibilités d'actions sont entre autres:

- l'envoi de données (V-data)
- l'envoi de permissions d'émettre (V-credit)
- la réception d'événements:
  * ouverture session acceptée (V-openact)
  * destruction contexte session (V-close)
  * interruption (V-interrupt)

Ces événements et leur gestion se déroulent en mode asynchrone.

1.4.3.4. Fonctionnalités des boîtes aux lettres.

Les sémaphores ne sont que des indicateurs d'action à entreprendre par TDS et VCAM. L'ensemble des données communiquées transitant par le réseau aboutit dans les boîtes aux lettres des applications.

Celles-ci permettent aux moniteurs transactionnels :

- de pouvoir récupérer les messages qui leur ont été renseignés par les sémaphores.

- de ranger dans un endroit connu d'eux et de VCAM, les messages qu'ils vont vouloir envoyer à un correspondant après l'avoir signifié par un sémaphore à VCAM.

1.4.4 Gestionnaire des correspondants (Netgen).

Le gestionnaire des correspondants coordonne la gestion des différentes sessions de communication des utilisateurs connectés par le moniteur transactionnel, pour les besoins de l'application qu'il supporte. Son activité principale est de maintenir le réseau des correspondants prédéfini à la génération du moniteur transactionnel en parfaite opérabilité. Ceci inclut la génération du réseau et la gestion dynamique des problèmes survenant sur le réseau. Netgen s'appuie fonctionnellement pour remplir sa tâche sur les facilités de VCAM.

Le réseau contrôlé intègre principalement, comme correspondants: des terminaux, des applications IMS, CICS, TDS, des consoles, des terminaux esclaves (imprimantes, lecteurs-disques, de terminaux écran) et des correspondants fictifs nécessaires pour certaines transactions.

Si tous ces correspondants doivent être déclarés à la génération du moniteur transactionnel, tous ne sont pas automatiquement connectés lors du lancement de ce dernier. Le moniteur transactionnel gère la possibilité de connexion des correspondants de manière dynamique. Par son biais, une session, un correspondant ou un ensemble (un "pool") de sessions peut être soit connecté ou déconnecté, soit reconnecté ou non en cas de crash du réseau ou du système.

En conclusion, Netgen, dans l'environnement transactionnel, joue un rôle de superviseur du réseau nécessaire et potentiel afin de satisfaire les besoins des moniteurs transactionnels.

1.4.5 Autres ressources de l'operating system.

Les ressources utilisées par TDS ne se limitent évidement pas aux seules facilités décrites ci-dessus. Ainsi comme moniteur transactionnel, TDS peut disposer de toutes les facilités de contrôle des processus système dont il a besoin.

Processus:

Le moniteur transactionnel et les applications transactionnelles qu'il supporte est, du point de vue de l'operating system un programme particulier en ce sens que, pour des raisons de performances, plus d'un processus lui est nécessaire à dérouler son code. Dans un environnement de programmation traditionnel, ce serait l'operating system qui aurait la charge de l'affectation de ces processus aux différents programmes supportés. Ici, il n'en est rien, c'est le moniteur transactionnel qui se charge de l'affectation des processus dont il dispose.

Dans GCOS7, les processus sont groupés en "process-group". Le nombre de ces "process-group", sur DPS7 peut s'élever jusqu' à 256. Chacun d'eux est composé de plus ou moins 90 processus. Ce dernier chiffre peut être sujet à modifications

vers le haut étant donné l'évolution constante des performances des machines BULL et de leur operating system.

Le moniteur transactionnel TDS se voit attribuer, quant à lui, un "process group".

L'affectation des processus n'est donc pas confiée à l'operating system, comme c'est le cas lors de l'utilisation de la machine en temps partagé entre plusieurs utilisateurs, mais bien au moniteur transactionnel. GCOS7 se charge de fournir les renseignements et les facilités à TDS pour lui permettre de gérer de manière optimale cette affectation de processus

Comme application liée de près à l'OS, le moniteur transactionnel TDS doit disposer d'assez d'espace virtuel pour pouvoir fonctionner et d'assez de segments de mémoire centrale.

Segmentation de la mémoire

GCOS7 comporte des mécanismes de segmentation et de pagination de la mémoire centrale.

Ces deux mécanismes sont courants dans les systèmes de grande dimension: d'une part, la segmentation permet la protection des données en mémoire et d'autre part, la pagination permet d'optimaliser la gestion de la mémoire lorsque les segments sont volumineux.

A l'aide des attributs de segments, l'operating system vérifie à chaque accès que le processus en exécution, travaillant dans un segment possède bien les droits (lecture, écritue, exécution), ainsi que le niveau de privilèges adéquats. Notons que sous GCOS7, quatre niveaux de privilèges ou "anneaux" sont ainsi définis.

Ces mécanismes de protection de l'information sont particulièrement importants dans le cadre d'applications transactionnelles. En effet, il est vital d'assurer d'une part, l'étanchéité des données manipulées par les divers processus attribués aux programmes utilisateur se déroulant en parallèle (intérêt de la vérification qu'une information se trouve dans les limites d'un segment à chaque accès), et d'autre part, que seuls des programmes privilégiés ont l'exclusivité d'accès aux données nécessairement communes à un grand nombre d'usagers (pour exemple, les tables du moniteur transactionnel).

Le volume de mémoire virtuelle pouvant être attribué à un moniteur transactionnel par GCOS 7 est de plus de 28 Mégabytes.

2 Concepts transactionnels

[BULL AG 89] [BULL PG 89] [B TDSC 89] [B TDS 89]

2.1 Introduction

Un certain nombre de principes et de termes utilisés dans le monde transactionnel se retrouvent dans d'autres domaines d'applications ou d'autres environnements informatiques.

Pour limiter l'ambiguïté pouvant résulter d'une transposition des effets des principes ou des termes utilisés à l'extérieur de l'environnement ici étudié, il est nécessaire de les définir ou de les expliquer dans cet environnement.

Ce chapitre a pour but de le faire. Les définitions données ci-dessous sont caractéristiques au monde transactionnel BULL. Il est à noter que ces termes et principes se retrouvent chez nombre de fournisseurs de tels systèmes. La mise en oeuvre et l'appellation peut différer selon les systèmes.

2.2 Définition des termes

2.2.1 Transactionnal Processing Routine (TPR)

Une TPR est un programme ou une procédure d'application écrite dans un langage de programmation tel que Cobol, Fortran, GPL, C (restreint), respectant les règles de programmation d'usage du moniteur transactionnel décrites par la suite.

Chaque TPR est identifiée par un identificateur de programme. Les principales restrictions de programmation concernent les possibilités d'échanges de données avec un programme ou un terminaliste correspondant.

2.2.2 Transaction

C'est le bloc de base de l'application. Une transaction est une unité de travail d'une application contrôlée par le moniteur transactionnel que ce dernier décide d'exécuter en réponse à une sollicitation d'un utilisateur correspondant.

Cette sollicitation est un message contenant un code identificateur de transaction à effectuer.

Pour TDS, une transaction est une "opération atomique" ou une suite d'"opérations atomiques". Par "opération atomique", on entend une séquence de traitements possédant les propriétés fondamentales transactionnelles d'"atomicité", de "cohérence", d'"intégrité" et de "durabilité" telle qu'elles ont été définies entre autres par Bernstein, Hadzilacos, Goodman [BERN 87].

Que sous-entendent ces propriétés ?

Les modifications de contextes (fichiers ou autres) réalisées pendant la séquence de traitements sont validées dans leur totalité ou sont toutes complètement supprimées. Les données manipulées pendant la séquence de traitements sont inaccessibles pour une autre opération atomique. Lorsque les données ont été validées, il est impossible de supprimer ces modifications autrement que par l'exécution d'une autre opération atomique.

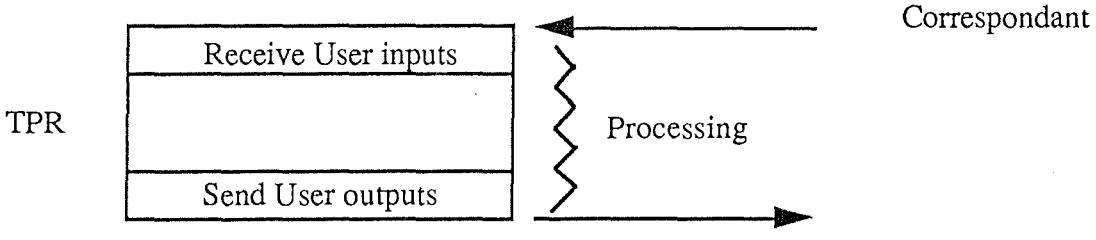
Pour TDS, une transaction est composée d'une suite de TPRs chaînées, agencées selon la logique de cette transaction.

Pour chaque correspondant, l'exécution d'un nombre de transactions ne peut être que séquentielle. A la réception d'un message, le moniteur transactionnel exécute le code relatif à la transaction demandée. Une fois la transaction exécutée, le correspondant reçoit la possibilité d'entrer le message suivant pour initialiser la transaction suivante.

2.2.3 Echange

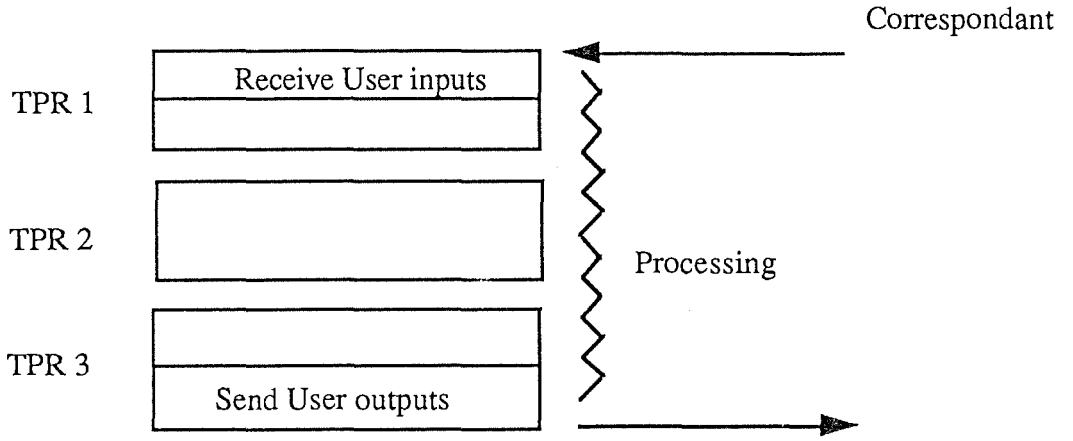
Un échange est une séquence d'exécution de TPRs, initialisée par l'envoi de données (un message) d'un correspondant vers le moniteur transactionnel, et clôturé lorsqu'en retour, le moniteur transactionnel répond à ce correspondant par un envoi d'un message.

Un échange peut avoir lieu au sein d'une TPR, (Fig 1.4.)



(Fig 1.4.)

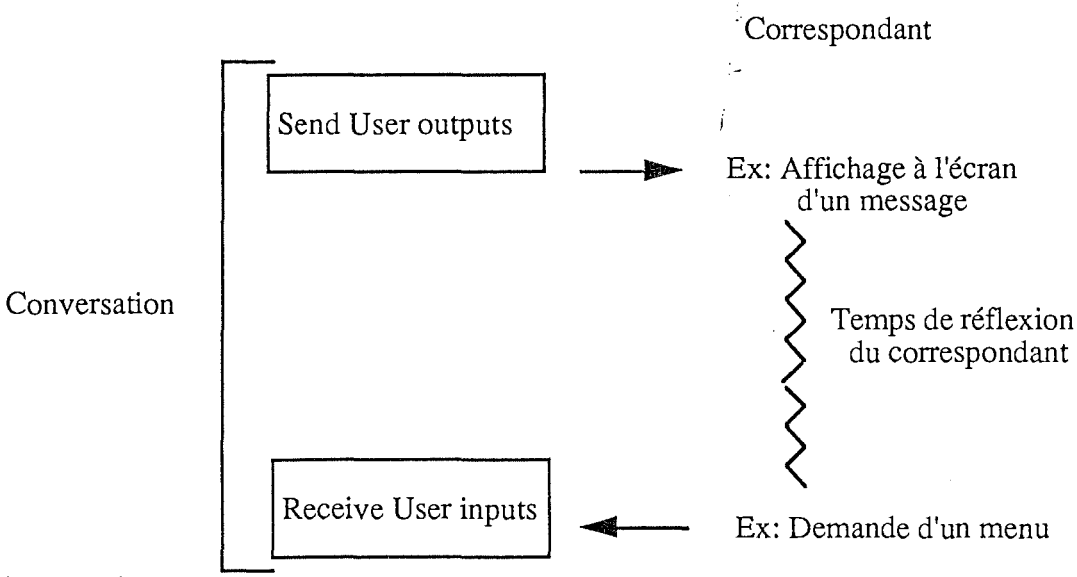
ou peut être mis en oeuvre par plusieurs TPRs, (Fig 1.5.)



(Fig 1.5.)

2.2.4 Conversation

Par le terme "conversation", on désigne une période de temps entre la réponse envoyée par la transaction via le moniteur transactionnel à un correspondant et la réception en retour de la pièce d'information suivante par le moniteur transactionnel en provenance de ce même correspondant. (Fig 1.6.)



(Fig 1.6.)

2.2.5 Règles d'ordonnancement

Au sein d'une transaction, différentes règles relatives à l'ordonnancement des TPRs, à l'existence des échanges et des conversations doivent être respectées.

Ces règles sont :

1. L'exécution du code de la transaction s'effectue durant les échanges, le correspondant quant à lui répond durant une conversation.

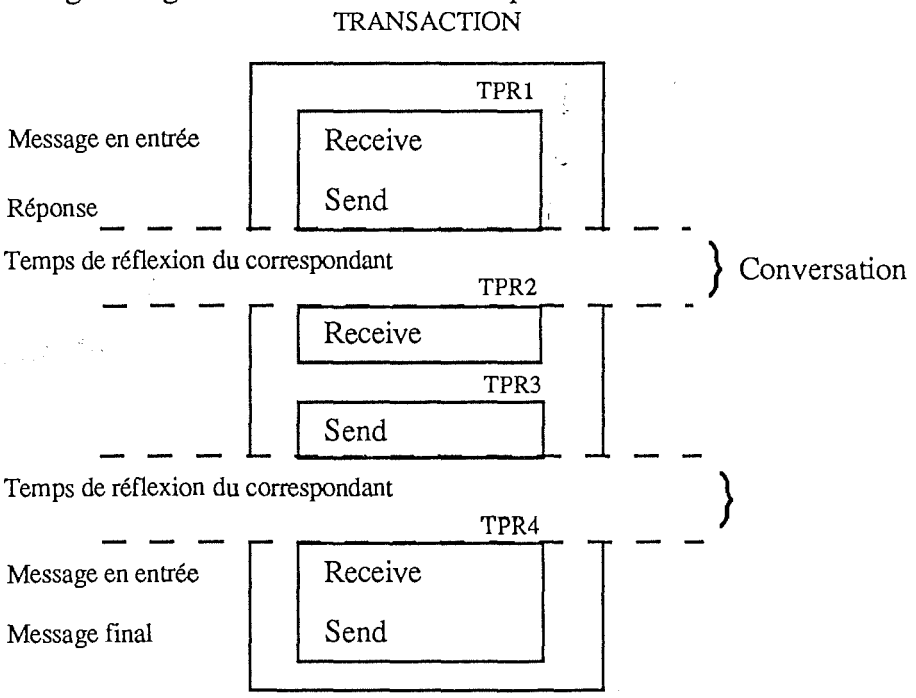
2. Une conversation sépare deux échanges de la même transaction de sorte que échanges et conversations se succèdent.

3. Aucune conversation n'est permise dans une TPR.

4. Une transaction consiste en au moins un échange mais n'inclut pas nécessairement une conversation.

5. Une conversation inclut le "temps de réflexion" entre une demande et une réponse. Celui-ci peut être long, particulièrement si le correspondant est un terminaliste. Dans ce cas ses actes physiques au point de vue du temps machine sont plus que disproportionnellement lents.

Cette découpe permet de distinguer clairement ce qui dans une transaction relève d'une part des actes du correspondant et d'autre part de l'exécution du code de la transaction. La figure Fig 1.7. illustre cette découpe.



(Fig 1.7.)

Il est de l'intéret du moniteur transactionnel de faire de cette distinction afin de permettre un relâchement du processus permettant ainsi aux TPRs d'autres

transactions de s'exécuter pendant le temps durant lequel le correspondant réagit. Ceci constitue une des bases des performances du moniteur transactionnel.

2.2.6 Point de Commit

Un point de Commit d'une transaction est un point d'exécution du code de la transaction après lequel toutes les actions exécutées par celle-ci sont acquises définitivement.

En cas de problèmes survenant par la suite , ce point de Commit fournit une base de reprise des traitement de la transaction.

Au point de Commit:

- toutes les ressources réservées pour la transaction (intervalles de contrôle, verrous de données, ressources virtuelles) sont libérées ,
- les données manipulées par la transaction sont laissées dans un état cohérent,
- les pointeurs courants des fichiers sont libérés.

Pour des raisons de performances, un point de Commit est toujours différé en fin de TPR. Si ce point de Commit ne se trouvait pas en fin de TPR, le moniteur transactionnel se verrait obligé de sauver l'état des variables locales des TPRs afin, qu'en cas de problème, le traitement transactionnel puisse être repris à cet endroit.

Notons que la fin d'une transaction est obligatoirement considérée comme un point de Commit pour TDS

2.2.7 Unité de Commit

Une unité de Commit est une unité d'exécution de TPRs délimitée par deux points de Commit. Durant l'exécution de l'unité de Commit, les TPRs concernées ont une vue cohérente des données.

Chaque unité de Commit est une opération atomique répondant aux propriétés fondamentales transactionnelles d'"atomicité", de "cohérence", d'"intégrité" et de "durabilité" telle qu'elles ont été définies au point 2.2.2.

3 Mécanismes de protection des données

Puisqu'un des principes du moniteur transactionnel est de protéger les données "coûte que coûte", des techniques de protection contre les incidents hardwares et softwares doivent être mises en coeuvre. Ces techniques sont centrées sur le concept d'unité de Commit tel qu'il a été défini dans le point 2.2.7.

3.1 Point de cohérence et de reprise

Tout système informatique est potentiellement apte à subir une défaillance. Ces défaillances sont multiples, et la plupart sont capables d'altérer la cohérence des traitements dus aux unités de Commit et l'intégrité des données que celles-ci manipulent.

Après un problème, le moniteur transactionnel doit être capable de reprendre les traitements sur base de données cohérentes. Pour ce faire, un "point de reprise" des traitements en cas de problème est défini; à ce point de reprise correspond un état de cohérence des données et des traitements. Pour le moniteur transactionnel, le début et la fin des unités de Commit constituent des points potentiels de reprise idéaux.

3.2 Journalisation

La "journalisation" des modifications des données de la base de données est la solution choisie, pour le moniteur transactionnel, pour préserver la cohérence et l'intégrité des données. Par journalisation, on entend ici sauvegarde des images de données modifiées de manière redondante dans une série de fichiers séquentiels appelés "journaux".

Deux journaux sont disponibles sous TDS.

Un journal "Before" contenant les images des données avant modification ainsi que les données identifiant l'unité de Commit, source de cette modification. L'écriture des images Avant est forcée sur le fichier Before avant la mise à jour des données dans la base de données.

Un journal "After" contenant les images des données après modification ainsi que les données identifiant l'unité de Commit, source de cette modification.

Notons que pour des raisons de performances liées aux nombres d'écritures disques qu'engendre la mise en oeuvre des deux journaux, le moniteur transactionnel laisse à l'initialisateur de l'application le choix de protéger ou non les données manipulées par l'application par les deux journaux en même temps ou par un seul journal. S'il le désire, il peut se passer des deux, mais à ses risques et périls.

Les images des données prises en compte sont des intervalles de contrôle de taille fixe, similaires à ceux que manipule le gestionnaire d'accès concurrent (voir Chap.1).

3.3 Techniques de reprise en cas de problèmes

Les deux techniques détaillées ici sont directement liées au type de journalisation sélectionné. Elles poursuivent un même but commun qui consiste en la remise dans un état cohérent des données afin que les traitements soient repris.

Les deux techniques de restauration bien connues sont le "Rollback" et le "Rollforward", encore connus sous le nom de "UNDO" et "REDO".

Le "Rollback" : consiste à annuler l'effet d'une unité de Commit sur les données de l'application. Le point de reprise au niveau des données est l'état des données avant traitements et restauration, les images Avant sont utilisées afin d'être appliquées sur la base de données.

Le "Rollforward" : consiste à refaire ce que l'unité de Commit a déja fait. Pour un raison physique, les images des données modifiées n'ont pu être écrites sur la BD, il suffira de réappliquer les mêmes images Après stockées dans le journal After pour remettre la base de données dans l'état cohérent acquis après traitements effectués par l'unité de Commit en question.

Ces deux techniques permettent de garantir la cohérence des données manipulées.

3.4 Technique de mise à jour "différée"

Pour des raisons de performances, l'écriture physique des données dans les fichiers de la base de données peut être différée. Dans ce cas, les modifications des données se font dans les "Buffers" de la base de données, les données n'étant protégées que par le journal After. Les avantages tirés de ce cas de figure sont une minimisation du nombre d'entrée/sortie exécuté sur la base de données; en effet, on peut écrire tous les intervalles de contrôle en une fois, et l'on évite ainsi les accès physiques aux fichiers qui seraient occasionnés par l'existence d'un journal Before.

3.5 Double protection des journaux et de la base de données

Sur base d'une sauvegarde de la base de données cohérente, il est possible en cas de problèmes ayant altéré l'intégrité de ses données, grâce à ses journaux, de reconstituer un nouvel état cohérent de cette base de données.

Encore faut il que ces problèmes ou d'autres n'altèrent pas l'intégrité des données de ces journaux, aussi, se doit on de les protéger. Les techniques de protection élaborées pour les bases de données peuvent être appliquées dans ce cas précis. Toutefois, pour ne pas altérer les performances du moniteur transactionnel, on peut se contenter simplement d'une double journalisation des images Après faite de manière asynchrone.

Dans le même ordre d'idée, il est possible de poursuivre cette politique de sauvegarde asynchrone pour une copie de la base de données. Dans ce cas, un "Rollforward" dynamique peut être mis en oeuvre sur base du double des images Après. Ainsi, en cas de gros problèmes occasionnant une perte ou une impossibilité d'accès de la base de données originale, le basculement quasi direct des accès vers son double peut être possible.

Ce mode de duplication et de mise à jour asynchrone constitue une garantie pour les accès base de données d'un moniteur transactionnel à haute disponibilité.

4 Gestion des traitements

4.1. Introduction

La gestion et la protection des données manipulées par l'application que supporte le moniteur transactionnel ne sont qu'un aspect du travail que doit accomplir ce moniteur transactionnel.

L'autre aspect consiste en la gestion de tous les traitements se déroulant pour satisfaire en parallèle les exigences des milliers de correspondants qui peuvent être connectés à l'application.

Pour plus de performances, le code du moniteur transactionnel et le code de l'application transactionnelle qu'il doit supporter sont fusionnés et forment un tout inamovible et non duplicable. Le tout est compilé en un programme exécutable. L'intérêt d'une telle disposition est la rapidité d'appel et d'exécution des divers programmes de l'application.

Comme le code de l'application est non duplicable, la réponse du moniteur transactionnel aux demandes de plusieurs correspondants se fera par l'exécution du code des même TPRs. Ici apparaît l'intérêt pour le moniteur transactionnel de disposer des techniques de verrouillage et d'isolation décrites dans les chapitres précédents.

L'architecture du moniteur transactionnel se compose essentiellement des cinq éléments suivants:

- un gestionnaire d'évènements (Event Manager),
- un module d'initialisation de TPRs et des contextes des correspondants s'y rattachant (TPR INIT),
- un module de gestion de fin de TPRs, (TPR TERM)
- un gestionnaire de Commits locaux ,
- un gestionnaire de synchronisme pour les transactions distribuées entre plusieurs applications.

Les mécanismes des deux derniers modules ne seront pas développés dans ce chapitre, mais dans les chapitres cinq et six qui suivent.

Avant de détailler plus ces différents modules, il est nécessaire de décrire le mécanisme permettant à TDS de multiplexer les processus dont il dispose.

4.2. Mécanisme de "Swapping"

Le multiplexage des processus mis à la disposition de TDS est possible grâce à un mécanisme de sauvetage des contextes des correspondants de l'application qu'opère TDS. Ces contextes pour des raisons de sécurité sont écrits physiquement sur un ensemble de fichiers appelé SWAP.

Le SWAP est composé d'un nombre d'entrées indépendantes entre elles correspondant aux sessions des correspondants ou utilisateurs. C'est sur ces fichiers que sont stoquées les données nécessaires au recouvrement complet des unités de Commit terminées, des transactions et des sessions utilisateur .

Chacune des entrées du fichier SWAP contient le contexte d'une session utilisateur. Chaque contexte est composé entre autre des données relatives: à la session, à la transaction exécutée, à la dernière TPR exécutée, aux communications entre TDS et les TPRs de la transaction, aux structures de contrôle d'écran ou de sortie, aux fichiers accédés...

Ces informations sont nécessaires pour relancer les TPRs des transactions suite à un incident ou à une interruption quelconque, volontaire ou involontaire, affectant l'exécution de ces TPRs.

Toutes ces entrées SWAP ne sont pas fonctionnellement identiques. Pour chaque session transactionnelle en cours, deux entrées SWAP lui sont allouées.

1) Une entrée Commit:

   Pour chaque session, l'entrée Commit du fichier SWAP contient le contexte de cette session transactionnelle au point de Commit atteint après exécution de la dernière unité de Commit pour la transaction en cours.
   Si la transaction en cours en est à sa première unité de Commit, l'entrée Commit du fichier SWAP contiendra le contexte de cette session transactionnelle au point de Commit atteint après exécution de la transaction précédente, c'est à dire le point de Commit situé en fin de la transaction précédente.

2) Une entrée courante:

   Pour chaque session, l'entrée courante du fichier SWAP contient le contexte de cette session transactionnelle à la fin de la dernière derniere TPR exécutée de la transaction en cours. Le sauvetage de contexte peut toutefois être différé, pour raisons d'économie d'accès au fichier SWAP, à une TPR suivante.

Le volume de contextes potentiellement accumulables dans le SWAP est non négligeable : il existe au moins un contexte par session par correspondant et jusqu'à 10.000 correspondants connectables en parallèle sur une application. Le nombre de fichiers composant le SWAP est important dans de telles conditions, tout comme le nombre de disques les supportant. Ainsi, plus il y a de fichiers et de disques, plus la capacité d'accès en parallèle aux données sauvées est élevée. Cette

multiplication des fichiers et des supports permet de moduler les performances du moniteur transactionnel au niveau de sa vitesse d'accès aux données.

A la vue de ces éléments, on peut deviner que la gestion du fichier SWAP est intimement liée à l'activité des modules d'initialisation et de terminaison des TPRs. La suite l'illustrera.

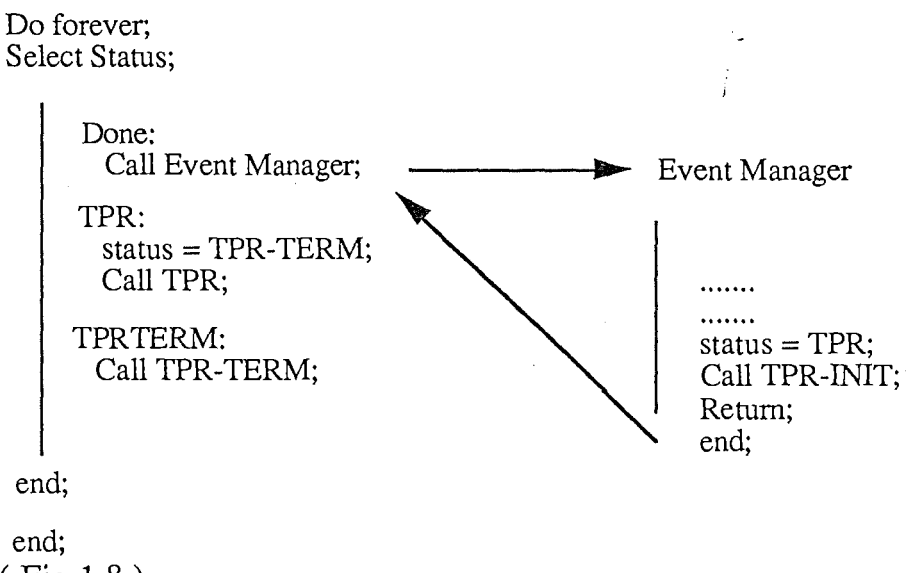
4.3. Schéma d'appel des TPRs

Le moniteur transactionnel dispose d'un nombre non négligeable de processus afin d'exécuter le code des TPRs. TDS ne contrôle pas directement tous ces processus, il les utilise. La gestion, qu'il opère au niveau de ceux-ci, est d'affecter chaque processus aux TPRs des différentes transactions. Cette affectation se fait en principe TPR par TPR.

Dans cette gestion, le moniteur transactionnel utilise les services de plusieurs de ses modules, le gestionnaire d'évènements (Event Manager) relatifs aux processus dont qu'il dispose, le module d'initialisation de TPRs (TPR INIT) et le module de terminaison de TPRs (TPR TERM).

Lorsque les processus ne sont par affectés à l'exécution d'une TPR, TDS leur assigne un programme de base permettant d'appeler et d'exécuter le texte code des programmes réalisant les diverses transactions. Ce programme de base est le programme "Main" dont la dynamique d'appel et les interactions avec les différents modules de TDS sont décrites ci-dessous.

Main:



(Fig 1.8.)

Comme on peut le constater, l'ossature du programme Main est assez simple. Seules trois options de traitements sont disponibles.

Une première option consiste à appeler l'Event Manager afin de déterminer quelle TPR sera exécutée par la suite. L'Event Manager se charge d'installer le contexte d'appel en lancant le programme d'initialisation des TPRs, TPR INIT. Cette action correspond au statut d'appel "Done".

La seconde option, statut d'appel "TPR" permet d'exécuter le code de la TPR sélectionnée.

La troisième option permet d'effectuer toutes les actions relatives à la terminaison de la TPR choisie, le statut d'appel est TPRTERM. La description des actions que doivent entreprendre les deux modules de gestion des Tprs, TPR INIT et TPR-TERM, sont reprises au point 4.5.

4.4. Fonction de l'Event Manager

L'Event Manager, comme son nom l'indique a la charge de la gestion des évènements internes ou externes (indication d'interruption provenant de l'OS, messages de VCAM, suspension d'exécution de TPR...) en relation avec le moniteur transactionnel.
Ses fonctions ne sont pas simplement liées à cette gestion, elles s'étendent à programmer l'ordre d'affectation des TPRs pour les processus disponibles, à appeler les modules TDS concernés.

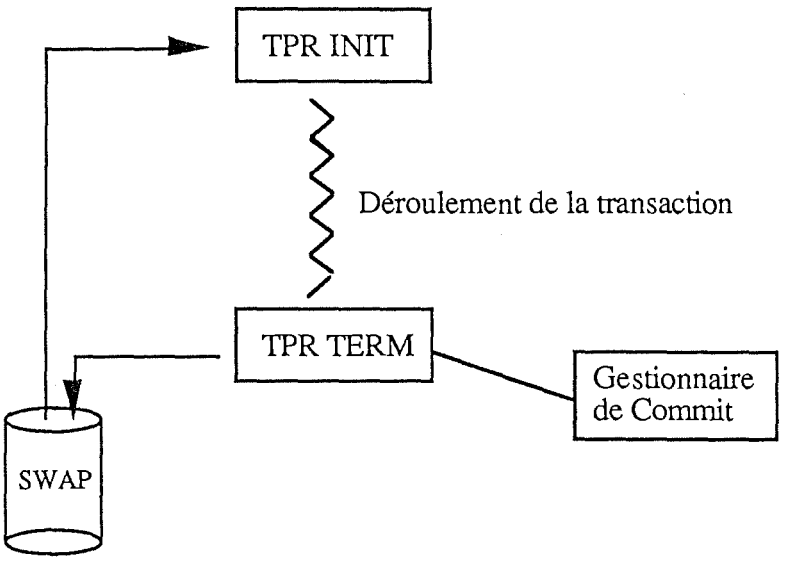
Le seul module qui nous intéresse est TPR INIT.

Cette gestion ne sera pas plus détaillée ici pour cause de complexité des appels, et par manque d'information quant aux traitements effectués par l'Event Manager.

4.5.Modules de contrôle des TPRs

4.5.1. Présentation

Ces module forment la base d'appel des TPRs. Ils gèrent le lancement (TPR-INIT) et la fin d'une TPR (TPR-TERM). Pour remplir leurs fonctions, ils se basent sur les fichiers SWAP. Le schéma suivant illustre les interactions entre le fichier SWAP, les deux modules de gestion des TPRs et le gestionnaire de Commit.



(Fig 1.9.)

4.5.2. Initialisation des TPRs

Le module d'initialisation des TPRs peut être appelé dans quatre cas de figures. Ces situations d'appels sont:

1) Le début d'une transaction
2) Le début d'une unité de Commit
3) Le début d'une TPR avec récupération du contexte sur le fichier SWAP
4) Le début d'une TPR sans récupération du contexte sur le fichier SWAP

TPR INIT opère à plusieurs niveaux:

Au niveau d'une TPR, il effectue :

- une recherche du nom de la TPR,
- une vérification de l'adressage, le chargement du code de la TPR et des données qui lui sont propres,
- une assignation de priorité d'exécution à la TPR,
- une mise en oeuvre d'une trace d'exécution de la TPR,
- une initialisation ou une réactivation du contexte écran ,
- une initialisation ou une réactivation du contexte GAC et des journaux,
- une initialisation ou une réactivation du contexte de la TPR,
- une activation de l'horloge de processus.

Au niveau d'une unité de Commit, il effectue :

- un contrôle de la non-concurrence entre les transactions,
- une gestion d'identificateurs de Commit,
- un appel au gestionnaire d'accès concurent GAC,
- des appels au journaux Before et After, si nécessaire.

Au niveau de la transaction, il effectue :

- l'initialisation du contexte de la transaction.

4.5.3. Terminaison des TPRs

Ce module est lancé en fin de TPR. Une TPR peut se situer en fin d'unité de Commit ou en fin de transaction. Les actions que ce module entreprend en fin de toute TPR sont entre autres:

- une désactivation de l'horloge de processus,
- un déchargement des données privées de la TPR,
- un calcul des adresses du journal After,
- une détection de fin de TPR, d'unité de Commit, de transaction,
- une détection des conditions d'enchaînement des TPRs,
- un contrôle des sessions,
- une prise en compte des évènements externes qui lui sont communiqués,
- une prise en compte des requètes au niveau de la TPR,
- un appel au module de trace,
- un envoi des messages de la TPR,
- un contrôle des conditions de terminaison vis-à-vis des règles et protocoles de TDS
- une fermeture du journal Before....

Si la TPR concernée est une fin d'unité de Commit, les actions supplémentaires suivantes peuvent être entreprises :

- une validation ou une invalidation des écritures fichiers,
- une validation ou une invalidation des requètes utilisateur,
- une fermeture des journaux Before et After,
- une relâche des ressources...

Ces actions sont mises en oeuvre par un appel au gestionnaire de synchronisme dont la logique d'action sera donnée dans le chapitre cinq.

Si la TPR concernée est une fin de transactions TPR TERM peut procéder:

- à une détection d'enchaînement de transaction,
- à un effacement des caractéristiques de la transaction...

Ces listes d'actions sont loin d'être exhaustives à la vue des nombreux cas particuliers qui peuvent illustrer le niveau de complexité de ces deux modules.

5 Gestionnaire de Commit.

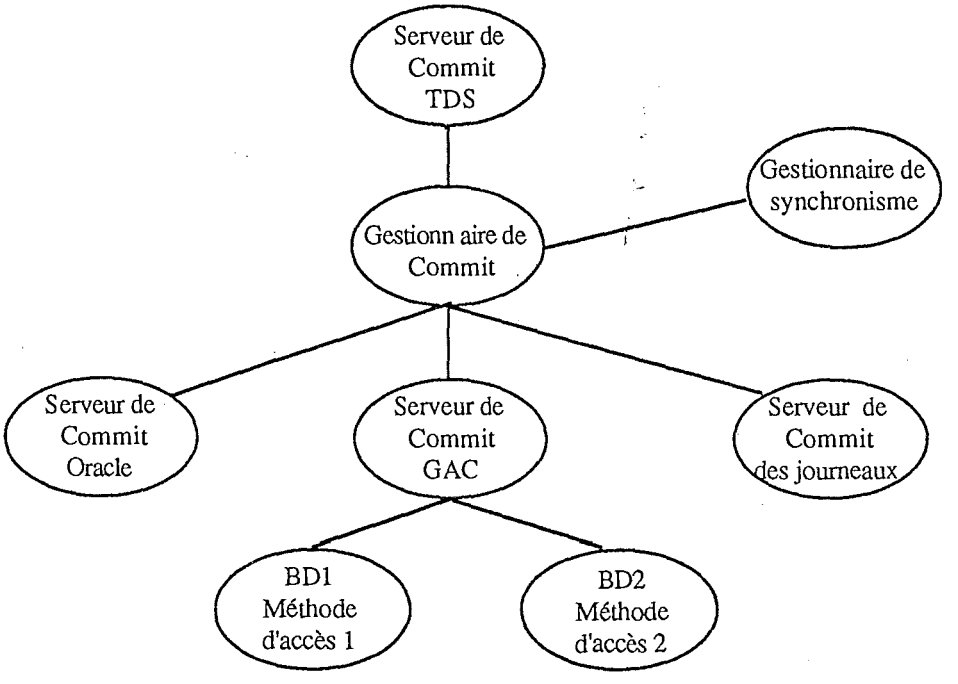
5.1 Introduction.

Le gestionnaire de Commit est un élément essentiel du moniteur transactionnel. Sa fonction est d'offrir les services nécessaires au moniteur transactionnel, afin qu'il remplisse son rôle de gestionnaire de cohérence des traitements.

Une partie de son travail est de synchroniser tous les serveurs de Commit prenant part dans l'élaboration du Commit local (journaux, gestionnaire d'accès concurrent, le sous-système transactionnel, les bases de données propriétaires, les bases de données externes supportées (Oracles e.a.).

Une autre partie de son travail est de répondre aux exigences du gestionnaire de synchronisme, s'occupant lui des synchronisations et des communications entre les applications inscrites dans une architecture transactionnelle distribuée. Un tel gestionnaire sous TDS est le gestionnaire de synchronisme.

Le schéma suivant illustre les interactions entre le gestionnaire de Commit, le gestionnaire de synchronisme et les différents serveurs de Commit intéressés



(Fig 1.10.)

La communication entre le gestionnaire de Commit et les différents serveurs de Commit se fait par le biais d'interfaces.

L'intérêt d'une telle disposition pour des interfaces est que tous les éléments constitutifs de l'architecture transactionnelle peuvent évoluer indépendamment l'un de l'autre sans mettre en péril tout le système transactionnel.

La description des interfaces relève de caractéristiques trop orientées "système" et est de peu d'intérêts. Notons que ces interfaces permettent :

1. d'informer le gestionnaire de Commit en permanence de l'état d'exécution des différentes unités de Commit à la charge des serveurs.

2. au gestionnaire de Commit de donner des ordres aux serveurs de Commit pour remplir sa fonction de gestionnaire d'unités de Commit de l'application et des entrées SWAP, (commandes début / fin de transaction, désactivation / activation des unités de Commit , Commit, Rollback, abandon des unités de Commit).

Le gestionnaire de Commit est avant tout un coordinateur travaillant dans une architecture supportant le Commit distribué. La suite du travail sur le Gestionnaire de Commit traitera de la dynamique du Commit distribué telle qu'il la supporte. Cette dynamique est un des piliers du système et la base fonctionnelle du moniteur transactionnel. Elle repose sur une exécution en deux phases: une préparation et une exécution du Commit.

5.2 Préparation du Commit

5.2.1 Présentation de la préparation au Commit.

Cette étape garantit la possibilité d'exécuter un Commit ou un Rollback de l'unité de Commit prévue et ce ,même si un accident apparaît par après. Si un des serveurs locaux ne peut garantir le Commit de l'unité concernée (celle dont il a la charge), le Commit est définitivement avorté. Si tous les serveurs sont en ordre de Commit, l'unité obtient: le statut "prêt à Commiter", si le Commit est distribué sur plusieurs sites transactionnels et le statut "Committed", s'il est local.

Si le Commit est distribué, l'unité de Commit entre dans une phase de doute.Celle-ci est gérée par le gérant de synchronisme qui décide ou non d'effectuer le Commit des unités de traitements. Le gérant de synchronisme possède un pouvoir de contrôle sur le gestionnaire de Commit dans ce dernier cas.

Cette étape passée, le Commit et le Rollback sont garantis pour toutes les ressources protégées.

- pour les enregistrements modifiés en mode "immédiat".

- Le Commit est garanti par les images Après écrites dans les fichiers par la méthode d'accès.
- Le Rollback est garanti par les images Avant écrites dans le journal Before lui-même sur fichier.

- pour les enregistrements modifiés en mode "différé".

  - Le Commit est garanti par les images Après écrites dans le journal After.
  - Le Rollback est garanti par les images Avant gardées dans les fichiers par la méthode d'accès.

- pour le contexte utilisateur

  - Le Commit est garanti par l'écriture du contexte de l'unité de Commit dans l'entrée courante du fichier Swap.
  - Le Rollback est garanti par l'existence du contexte précé-dent dans l'entrée Commit du fichier SWAP.

- pour Oracle

  - Le Commit n'est pas garanti. Le moniteur transactionnel à ce jour dialogue avec une version d'Oracle exécutant un Commit en une phase.
  - Le Rollback est garanti par le non-Commit de la base de données Oracle.

5.2.2 Préparation du journal Before

Avant chaque modification (écriture) d'enregistrement, les images Avant des données sont forcées dans le journal Before. De ce fait, les enregistrements du journal sont effectivement inscrites sur disques. Il n'y a donc rien de spécial à faire.

5.2.3 Préparation du journal After

La préparation consiste à forcer la rémanence des images Après, encore contenues dans les buffers, dans le journal After de l'unité de Commit journalisée. Ceci est nécessaire pour les fichiers en mode de mise à jour "différée" si l'unité de Commit doit tout de même être validée après apparition d'un problème.

5.2.4 Préparation du Data Manager et du General Access Controler

Cette fonction doit fournir un point stable pour toute la gestion des données. Les enregistrements modifiés des fichiers en mode de mise à jour immédiate doivent être écrits, ceux en mise à jour "différée", non.

Les contextes du GAC et du Data Manager en cours doivent être sauvés dans l'entrée courante du SWAP. Ce point stable est nécessaire pour être capable d'effectuer un Commit dans toutes les situations, y compris au cas où un problème apparaîtrait.

S'il n'existe pas de zone de doute, l'unité de Commit sera validée sauf en cas de problèmes. S'il existe une zone de doute, seul le gestionnaire de synchronisme décide de la validation de l'unité de Commit.

La préparation se fait en deux étapes .

1. Tous les verrous d'enregistrements, qui n'ont pas été modifiés par l'unité de Commit courante, sont libérés. Cette libération augmente les performances des transactions concurrentes, spécialement lorsque le temps séparant la préparation, du Commit est long. C'est le cas pour toutes les transactions utilisant un Commit distribué.

2. La réécriture de tous les enregistrements en mise à jour "immédiate" est faite et le sauvetage des contextes du Data Management, nécessaire à l'unité de Commit suivante, est exécuté. Ainsi les pointeurs de fichiers et les données en mise à jour "différée" ne seront pas perdus et pourront être utilisés par après.

5.2.5 Préparation d'Oracle

Pour des raisons de performances, Oracle ne fournit pas de primitives " Prêt à Commiter " pour cause de Commit en une phase. Seule une gestion heuristique de cette préparation est envisageable. Aussi faut-il allouer à Oracle un espace mémoire dans lequel on peut recevoir une information lorsque Oracle, est hors service, c'est à dire s'il n'est pas en état de satisfaire une demande de Commit.

Si Oracle est opérationnel, rien n'est à faire, sauf peut-être, si la base de données a été modifiée, resynchroniser le moniteur transactionnel et Oracle.

5.2.6 Préparation du moniteur transactionnel

Cette étape débute lorsque tous les autres serveurs de Commit ont répondu de manière positive à la préparation de Commit.

La décision est prise d'exécuter un Commit complet de l'unité de Commit locale au niveau du moniteur transactionnel. Si aucune erreur n'apparaît durant cette période, l'unité de Commit sera validée, sinon un Rollback de celle-ci sera effectué.

Si tout se déroule normalement, le moniteur transactionnel force le sauvetage du contexte de l'unité de Commit dans le fichier SWAP. A ce moment, les

modifications sur les bases de données et les fichiers utilisés par l'unité de Commit sont effectives.

5.3 Commit

5.3.1 Présentation du Commit.

Cette étape consolide les modifications engendrées par la préparation de l'unité de Commit et exclut la possibilité de revenir en arrière. Cette étape est rémanente.
Si pour une raison ou une autre, la consolidation n'est pas réussie, on peut la recommencer à nouveau. Le mécanisme de consolidation est idempotent .

La consolidation des ressources se fait comme suit:

Pour les enregistrements en mode de mise à jour "immédiate":

- Pour le Commit: rien à faire, l'écriture des enregistrements a été faite pendant la préparation.

- Pour le Rollback: on élimine la possibilité de Rollback en effaçant les images Avant écrites précédemment dans le journal Before. En cas d'échec de cet effacement, on recommence.

Pour les enregistrements en mode de mise à jour "différée":

- On consolide les données en forçant l'écriture des buffers, des fichiers les contenant, dans les fichiers. En cas d'échec d'écriture, on utilise les images Après du journal After pour les consolider.

- La possibilité de Rollback est éliminée par relâche des images Après écrites dans le journal After, après évidemment que la consolidation ait été un succès.

Pour le contexte utilisateur:

- Le Commit est garanti par la journalisation de l'état de l'unité de Commit dans l'entrée courante du fichier SWAP et en octroyant un état courant à une nouvelle unité de Commit. Pour la suite, cette nouvelle unité de Commit occupera l'entrée Commit du SWAP.

- La possibilité de Rollback est annulée par la libération de l'entrée Commit du SWAP de l'unité de Commit précédente .

Pour la base de données Oracle:

- Le Commit est garanti en demandant au serveur Oracle de valider toutes les modifications faites par l'unité de Commit courante.

5.3.2 Commit du journal Before

Si l'unité de Commit a été désactivée entre la préparation et le Commit, une réactivation de celle-ci doit être opérée. Le Commit sera consolidé par la relâche de toutes des images Avant associées à cette unité de Commit.

5.3.3 Commit du journal After

Si l'unité de Commit a été désactivée entre la préparation et le Commit, une réactivation de celle-ci doit être opérée. Le Commit sera consolidé par la relâche de toutes les images Après associées à cette unité de Commit.
Ces images ne pourront plus être utilisées pour le Rollback.

5.3.4 Commit du GAC et du Data Manager

1.Si l'unité de Commit a été désactivée entre la préparation et le Commit, une réactivation de celle-ci doit être opérée. La restauration des contextes de GAC et du Data Manager doit être faite, en lisant leurs contextes dans le fichier SWAP.

2. Les fichiers en mise à jour "immédiate", sont cohérents. Il reste à écrire tous les enregistrements modifiés en mise à jour différée pour les valider, si ceci n'avait pas été possible durant la phase de préparation (fichier non ouvert au point de vue Data Manager).

3. Libérer les verrous des enregistrements pour rendre ceux-ci visibles par les autres unités de Commit. Certains ne seront pas libérés, ce sont ceux dont une durée de verrouillage est supérieure à la durée de vie de l'unité de Commit concernée.

5.3.5 Commit Oracle

Pour des raisons de synchronisme et de cohérence entre la base de données Oracle et les bases de données locales, le Commit de l'unité de cohérence sous Oracle doit être exécuté avant l'unité de Commit de la BD propriétaire locale. Cet ordre permet d'éviter une mise à jour incohérente, des données des bases de données locales par rapport aux données sous Oracle, qui serait due à une déficience éventuelle du serveur Oracle pendant la phase de Commit.

Il existe une zone d'incohérence entre le début du Commit Oracle et la fin du Commit du moniteur transactionnel. Pour sécuriser le système, une démarche en trois étapes est opérée.

1.On effectue un sauvetage sur le fichier SWAP du contexte Oracle, avant d'entrer dans la zone d'incohérence.

On considère le processus de Commit Oracle comme une unité de Commit (entrée Commit du SWAP).

2. Une demande de Commit à Oracle est faite par le Gestionnaire de Commit local.

3. On quitte la zone d'incohérence en fin de Commit global en éliminant le contexte Oracle sauvé pour l'occasion. Cette dernière phase est exécutée lors de la mise à jour des écritures de Commit de l'ensemble de l'unité de Commit . Ceci permet d'éviter une multiplication des accès disques, coûteux au point de vue des performances du système transactionnel.

5.3.6 Commit TDS

Les actions à entreprendre dépendent du niveau de Commit.

Si ce Commit est non distribué, il n'y a rien à faire, tout a été fait pendant la préparation. L'unité de Commit est déjà dans un état "Commit".

Si ce Commit est distribué, on consolide l'état de l'unité de Commit. On change son état "Prêt à Commiter" en "Commit" sur le fichier SWAP. On sauve ensuite le contexte de cette unité de Commit TDS sur le fichier SWAP. A ce stade là, un Rollback ne peut plus être effectué.

5.3.7 Séquence du Commit

Un certain nombre de contraintes d'ordre temporel doivent être prises en considération dans cette phase de Commit:

1. Effectuer le Commit des unités de cohérence de GAC et du Data Manager avant de relâcher les verrous des enregistrements.

2. Relâcher les images Après, dès que le Commit du GAC et du Data Manager s'est terminé avec succès . Les images Après couvrant les mises à jour différées, leur relâche signifierait une impossibilité de reprise de la phase de Commit au cas où le GAC et le Data Manager connaîtraient des difficultés.

3. Il faut éliminer les images Après avant les verrous. Ainsi, pour la suite des traitements transactionnels, le journal After, après lâchage des verrous, contiendra les nouvelles images Après. Si cela n'est pas fait dans cet ordre, on risque de libérer une image Après déjà modifiée par l'unité de Commit suivante.

4. Les Commits s'effectueront dans cet ordre: Commit du journal Before, Commit du Data manager et de GAC, Commit du journal After, ensuite relâche des verrous des données modifiées. Le Commit de TDS se fait à n'importe quel moment.

5.4 Rollback

5.4.1 Présentation du Rollback.

Cette étape, ici présentée, permet d'annuler les effets des actions des unités de Commit transactionnelles. Les données et les variables liées à l'unité de Commit en question seront remises dans leur état initial avant traitement. Il est à noter que le mécanisme de Rollback est idempotent.

Au long de ce processus, toutes les modifications sur les ressources protégées subiront un Rollback.

Pour les enregistrements en mode de mise à jour "immédiate".

- Le Rollback est garanti en appliquant les images Avant écrites précédemment sur le journal Before. Après cette étape, on élimine les images Avant.

Pour les enregistrements en mode de mise à jour "différée":

- On invalide les modifications des enregistrements encore situés dans les buffers des fichiers. On élimine ensuite les images Après contenues dans le journal After.

Pour le contexte utilisateur:

- On oublie l'entrée courante du fichier SWAP et l'on restaure le contexte de l'unité de Commit précédente contenue dans l'entrée Commit du fichier SWAP.

- Le statut "Prêt à Commiter "de l'unité de Commit soumise à un Rollback est éliminé en substituant à cet état, l'état d'avortement "Abort".

- Si le Rollback échoue, l'état de l'unité de Commit trouvée sera "Abort" de telle sorte que cette unité de Commit sera soumise à un Rollback aussi longtemps que ce Rollback n'est pas terminé.

Pour Oracle

- Une demande de Rollback est faite à Oracle .
Cette requète prise en compte, Oracle garantit l'élimination de la possibilité de Rollback.

5.4.2 Rollback du journal Before

Si l'unité de Commit a été désactivée entre la préparation du Commit et le Rollback, une réactivation de celle-ci doit être opérée.

On applique ensuite les images Avant sur tous les enregistrements modifiés en mode de mise à jour "immédiate", ces enregistrements étant logiquement protégés par le journal Before.

5.4.3 Rollback du Journal After

Si l'unité de Commit a été désactivée entre la préparation du Commit et le Rollback, une réactivation de celle-ci doit être opérée.

On efface les images Après pour empêcher tout recouvrement dynamique ou statique des données à l'aide du journal After.

5.4.4 Rollback du GAC et du Data Manager

Si l'unité de Commit a été désactivée entre la préparation et le Rollback, une réactivation de celle-ci doit être opérée.

Une restauration des contextes GAC et Data Manager doit être exécutée en lisant leurs contextes sur le fichier SWAP.

On invalide enfin les modifications des enregistrements non encore écrits.

C'est à dire : - les enregistrements en mise à jour "immédiate" non encore écrits et les enregistrements en mise à jour "différée" en ne les réécrivant pas dans les différents fichiers.
- les autres enregistrements en mise à jour "immédiate" quant à eux en les réinitialisant à l'aide du journal Before.

5.4.5 Rollback d'Oracle

Ce cas est assez simple à traiter. Tant que TDS n'a pas demandé à Oracle d'exécuter un Commit, aucune consolidation des données au niveau de la base de données Oracle n'est faite.

TDS, en tant que gestionnaire de Commit, doit simplement demander à Oracle d'exécuter un Rollback.

5.4.6 Rollback deTDS

L'unité de Commit et le Contexte User sont réinstallés (lu du SWAP) s'ils ont été désactivés (sauvés dans le SWAP).

Trois cas de figures peuvent être envisagés :

1. Pas de préparation.
   On invalide l'unité de Commit au niveau TDS.

2. La préparation échoue.
   Aucune action spécifique n'est à entreprendre car l'unité de Commit est toujours considérée comme invalidée.

3. La préparation réussit.
   L'unité de Commit est invalidée, le dernier contexte est restauré.
   L'unité de Commit sera étiquetée "invalidée" , le dernier contexte est réactualisé. L'unité de Commit sera relancée avec le nouveau contexte.

5.4.7 Séquence du Rollback

Tout comme la phase de Commit, les différents Rollbacks locaux sont soumis à des règles d'ordre.

1. La relâche des verrous ne peut s'effectuer qu'après Rollback des unités de Commit de GAC et du Data Manager. Les enregistrements, en effet, ne doivent pas pouvoir être accédés durant la phase de restauration des valeurs initiales des données et des variables.

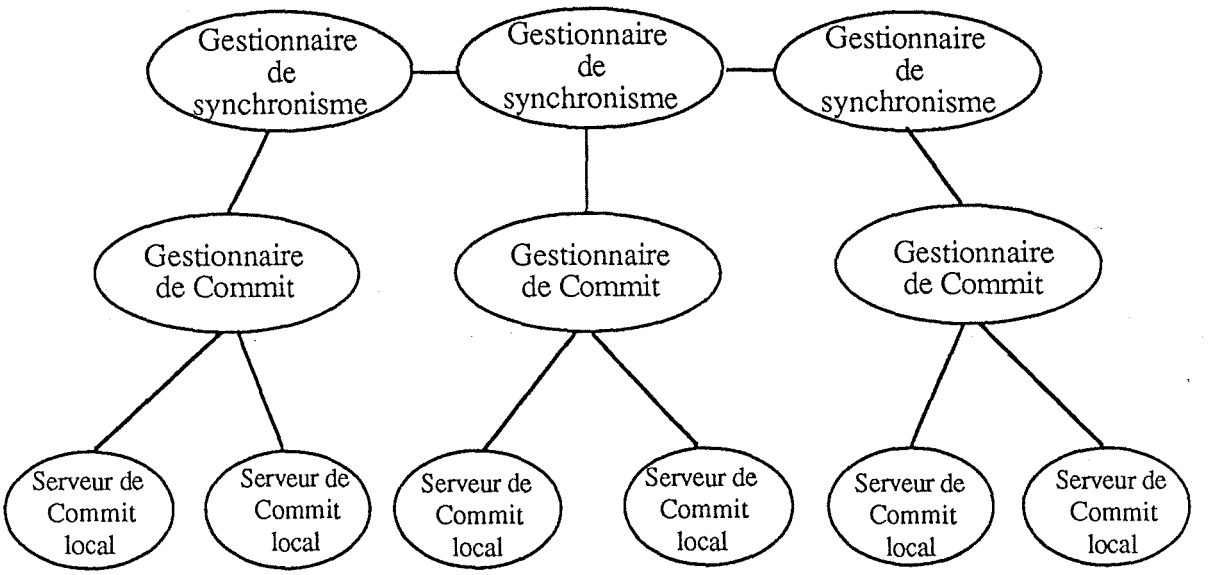
2. Le Rollback des données en fichiers les plus sensibles, doit être fait en premier.

3. L'ordre des Rollbacks pour les différents serveurs sont: le Rollback du journal Before, le Rollback de GAC et du Data Manager, le Rollback du journal After, vient ensuite la relâche des verrous de toutes les données.

4. Pour des raisons d'économie d'I/Os, il serait intéressant d'exécuter le Rollback de TDS avant tous les autres Rollbacks. Cette étape permettrait de grouper les accès aux fichiers nécessaires à la préparation des contextes pour les autres serveurs de Commit et de Rollback.

6 Gestionnaire de Synchronisme

Si le gestionnaire de Commit gère le contrôle de cohérence au niveau local, en cas de "zone de doute", c'est-à-dire au cas où des communications sont faites entre applications au sein d'une application distribuée, la coordination de la cohérence globale des traitements est à la charge des gestionnaires de synchronisme.



(Fig 1.11).

Du point de vue fonctionnel et local, le gestionnaire de Commit est soumis aux ordres du gestionnaire de synchronisme pour toutes les "zones de doute".

Au niveau de la distribution globale, pour une application distribuée, le gestionnaire de synchronisme est inscrit au titre de coopérateur dans une architecture réseau permettant la communication interapplications sur un même site ou entre des sites distants.

Il va agir comme le coordinateur des flux interapplications relatifs aux applications tournant sous le moniteur transactionnel auquel il est attaché.

Pour ce faire, il va dialoguer avec d'autres gestionnaires de synchronisme ou avec d'autres produits équivalents de manière protocolaire.

Au niveau de TDS, les protocoles sont :

XCP1 (eXtended Cooperative Protocole 1 compatible LUP (IMS / IBM),
XCP2 (eXtended Cooperative Protocole 2 +/- compatible LU6.2 (CICS / IBM))

Ces protocoles seront détaillés dans la seconde partie du travail relative aux communications entre applications.

Partie 2 : Communications entre applications

Introduction

Le temps où les applications développées sous moniteurs transactionnels avaient une portée strictement locale est dépassé. Ces dernières années ont vu un effort de développement et d'intégration important dans le domaine des télécommunications. Les possibilités de connecter des systèmes historiquement hétérogènes, se sont multipliées. Les échanges d'informations entre ces sites hétérogènes devenaient de plus en plus réalisables.

Hélas, dans la plupart des cas, le maintien de la cohérence des transferts s'effectuait de manière heuristique limitant ainsi la gestion des programmes de contrôle à un nombre limité de cas de figure de distribution transactionnelle entre sites hétérogènes.

L'élaboration de protocoles de coopération a été nécessaire face à ces besoins nouveaux de coopération entre programmes d'applications distribuées entre sites distants. Ces protocoles sont indispensables aux moniteurs transactionnels ayant la charge de maintenir les propriétés transactionnelles de base (voir point 2.2.2) des transactions liées à de telles applications distribuées.

Le but de cette seconde partie sera d'exposer les possibilités de coopération existant entre des programmes d'applications tournant sous des environnements à priori non-compatibles.

Dans un premier temps, une étude sera faite de protocoles développés alors qu'aucune normalisation au niveau international en cette matière existait , ISO principalement . Les deux protocoles présentés ne sont pas sans intérêts. Ils permettent l'interconnexion d'applications avec celles tournant sur des supports largement diffusés au niveau mondial que sont IMS (Protocole LU/P) et CICS (protocole LU6.2) d'IBM.

Vu sous l'optique Bull/TDS, ces protocoles se nomment respectivement , XCP1 (eXtended Cooperative Protocol 1) et XCP2 (eXtended Cooperative Protocol 2).

Dans un second temps, l'aspect normatif au niveau international sera abordé. Ceci sera fait par l'approche de la nouvelle norme, ISO / IEC 10026 relative à l'interconnexion des systèmes ouverts et au traitement des transactions distribuées. Cette norme sera adoptée officiellement en octobre 1990.

7 Le protocole de coopération XCP1

[BULL PXCP 87] [B TDS 89] [B TDST 89]

7.1 Introduction

XCP1 n'est pas en tant que tel un protocole complet de coopération pour applications distribuées. Il permet, en effet, uniquement le recouvrement des messages durant la communication entre une application transactionnelle et une autre. A ce niveau de coopération, la cohérence des traitements distribués est à la charge des applications désirant communiquer.

Au sein de la famille de protocoles XCP, XCP1 se contente de régir les problèmes de coopération des deux niveaux de sécurité suivants:

Niveau : NONE
où seuls des transferts de messages non protégés sont autorisés. Une quelconque méthode de recouvrement de messages entre les correspondants ou de synchronisation des dialogues est inopérante à ce niveau .

Niveau : Message Recovery
où seule une fonction de recouvrement de messages est disponible, Une synchronisation des traitements entre applications coopérant est exclu.

Notons dès à présent que des niveaux de synchronisation plus complets incluant des possibilités de coopérations étendues entre applications, les niveaux COMMIT et SYNCPOINT, sont pris en charge par un protocole de plus haut niveau étudié par la suite, XCP2.

XCP1 est , en offrant ces services, compatible avec le protocole LU/P d'IBM. Dans l'environnement transactionnel BULL, c'est le moniteur transactionnel TDS qui est chargé de gérer les échanges de messages soumis aux régles protocolaires XCP1. Le moniteur transactionnel TDS sera communément appelé fournisseur de services XCP1.

7.2 Définitions

7.2.1 Introduction

L'exposé de ce protocole nécessite la présentation d'éléments de base. Les notions de correspondant, session, chemin de coopération, attributs de session, et de dialogue sont définies ci-après.

7.2.2 Correspondant

Un correspondant est une entité, par exemple un terminal,un terminal intelligent ou une application, communiquant avec une application .

Celui-ci est défini par :

- un nom, c'est à dire un identificateur local par lequel le correspondant est connu par l'application, ou encore une adresse absolue connue du réseau DSA (nom de noeud + nom de session + nom de mailbox).

- des attributs: son type (XCP1 ou Terminal)
  son adresse (nom de noeud + nom de mailbox)
  son profil (rôle et option)
  primaire ou secondaire,
  actif ou passif.

Deux ou plus de deux correspondants avec des noms et des attributs différents peuvent référencer la même application. Pour plus de facilités, aucune distinction ne sera faite entre "correspondant" et "application", les deux termes seront utilisés indifféremment.

7.2.3 Session et chemin de coopération

Pour pouvoir correspondre, des applications doivent être reliées entre elles par un lien de communication. Pour ce faire, le fournisseur de services XCP1, TDS, se charge d'établir un ensemble de sessions de communication entre les correspondants. Cette série de sessions constitue le support sur lequel va venir se greffer une série de liens logiques de communication entre les programmes des applications. Ces liens logiques, constitués sous XCP1, sont appelés "chemins de coopération".

Par la suite, les mots "session" et "chemin de coopération" seront indifféremment employés.

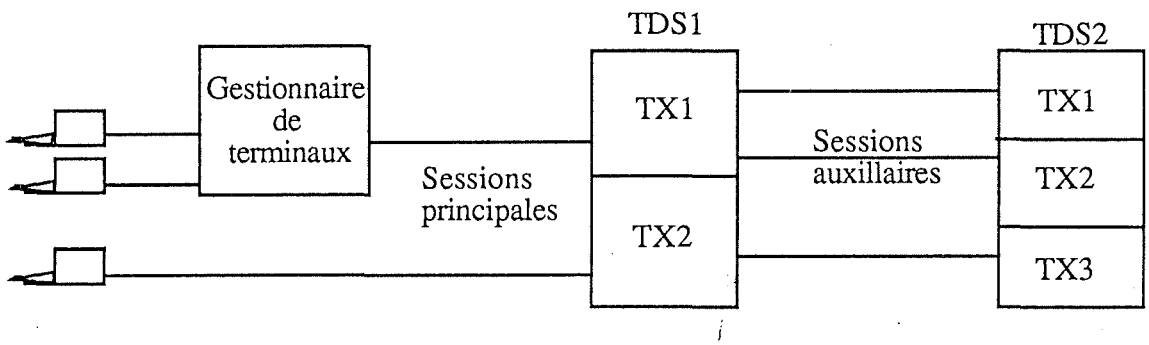
7.2.4 Attributs

Pour une application transactionnelle, deux différentes sessions doivent être considérées:

a. la "session principale" qui autorise la transaction à dialoguer avec le correspondant l'ayant initialisé et
b. la (les) "session(s) auxilliaire(s)" qui permet(tent) à la transaction de dialoguer avec d'autres correspondants.

La différence entre la "session principale" et la "session auxilliaire" est leur mode d'allocation. Lorsqu'une transaction démarre, la session principale est déjà activée et implicitement allouée, et l'utilisateur n'est pas averti de son existence. Une session auxilliaire, par contre, doit être explicitement demandée par toute transaction de l'application.

Le schéma suivant donne un exemple d'architecture de coopération transactionnelle distribuée entre terminaux et applications supportées par deux moniteurs transactionnels.



(Fig 2.1.)

7.2.5 Dialogue

A travers une session, des échanges de messages peuvent être opérés entre correspondants. Les deux applications concernées par une session peuvent être supportées par deux moniteurs transactionnels du même type (2 TDS) ou par deux moniteurs transactionnels de types différents (1TDS et 1 IMS). Les échanges de données entre deux applications constituent en fait un dialogue entre deux unités de travail.

7.3 Présentation du protocole

7.3.1 Attributs des correspondants et contrôle des droits de transmission

XCP1 est un protocole hiérarchique en ce sens que les deux correspondants n'ont pas les mêmes droits. L'un, le correspondant passif, est subordonné à l'autre, le correspondant actif. Cette subordination est matérialisée par l'existence d'un jeton d'émission. Ce jeton, un par session, donne le droit à son détenteur de contrôler et transmettre des données sur cette même session. Le correspondant passif, lui, à le droit de demander un transfert du jeton. La prise en compte de cette demande est soumise à la bonne volonté du détenteur du droit d'émission. Ce mode de contrôle est communément appelé mode de transmission "Two Way Alternate" .

Indépendamment de la possession du droit d'émission, une séparation fonctionnelle est entretenue entre un correspondant initialisant une session vers un correspondant, et ce dernier correspondant.

Le premier des deux se verra attribuer paradoxalement une fonctionnalité secondaire tandis que le second, lui, se verra attribuer une fonctionnalité primaire.

Les prérogatives propres à chaque correspondant sont les suivantes:

Pour le correspondant secondaire:

- avoir la possibilité de dialoguer avec un correspondant autre que son initiateur.
- pouvoir ouvrir des sessions gérées en pools vers un correspondant primaire. Ces sessions sont les sessions auxiliaires nécessaires pour dialoguer avec les correspondants primaires.
- avoir la permission d'initialiser une transaction vers le correspondant primaire par l'envoi d'un message .
- pouvoir recevoir des messages de requètes non sollicités venant du correspondant primaire.
- pouvoir utiliser la demande de resynchronisation pour un rollback ou pour un avortement de transaction.

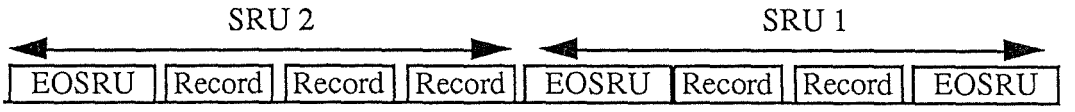
Pour le correspondant primaire:

- permettre à un correspondant secondaire actif de soumettre une transaction .
- avoir la permission d'envoyer une demande d'exécution d'une transaction vers un correspondant secondaire passif ou actif.
- pouvoir initialiser une resynchronisation
    - à l'établissement des sessions
    - au rollback ou à l'avortement d'une transaction.

7.3.2 Format des données transmises

Pour XCP1, les messages transmis entre deux correspondants d'une session représentent des unités potentielles de recouvrement. Chacun des messages est une " Session Recovery Unit " (SRU).

Ces SRUs sont en fait composées d'un ensemble d'enregistrements délimités par un enregistrement spécifique appelé " End Of Session Recovery Unit " (EOSRU).



(Fig 2.2.)

Ces messages recouvrables ne peuvent être ni perdus ni dupliqués. Le recouvrement est utile en cas de déconnexion ou d'un traitement anormal ayant altéré les données véhiculées.

7.3.3 Mécanisme de synchronisation et de reprise

Les SRUs envoyées, afin de ne pas être dupliquées, sont numérotées dans le protocole XCP1. Cette numérotation invisible de l'utilisateur, est gérée par le fournisseur de services XCP1, en l'occurence TDS.

Un contrôle de flux s'effectue dans le sens de la transmission des données sur une session, grâce à une numérotation des messages que la session véhicule et à laquelle se réfèrent les autorisations d'émission transmises dans l'autre sens.

Notons qu'à chaque session sont liés deux compteurs, un pour l'émission des messages (Send-SRU-Number, SSRU-Nb), et le second pour la réception des messages (Receive-SRU-Number, RSRU-Nb). Chaque enregistrement de SRU se voit attribuer, pour son transfert, le numéro de la SRU de laquelle il fait partie.

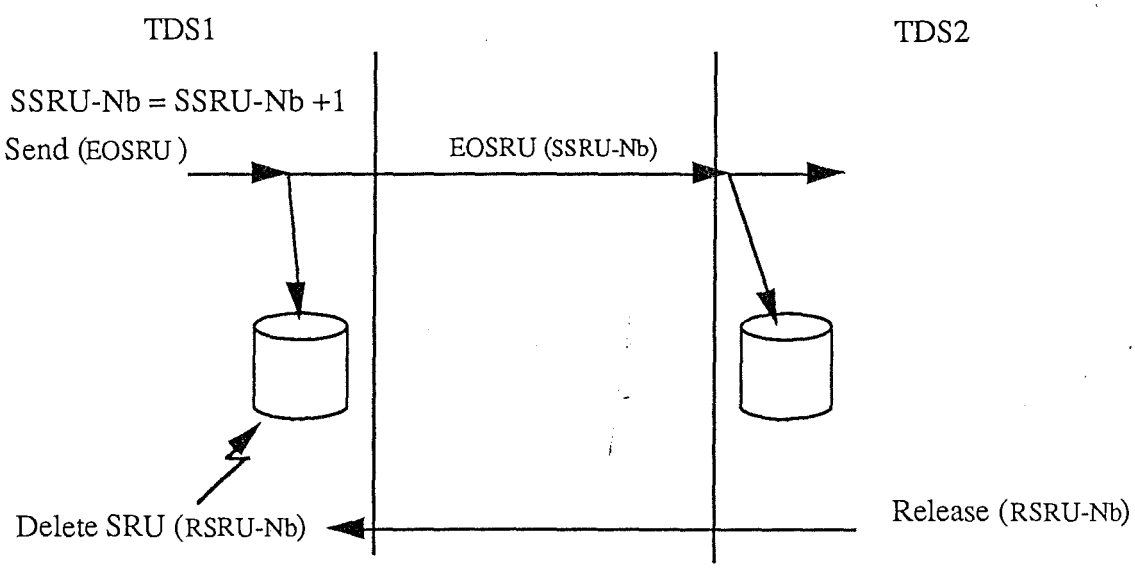
Une précision est à apporter au sujet de l'estampillage numérique des données transférées entre une application soumise au protocole XCP1 et une application soumise au protocole LU/P d'IBM. Pour un tel cas de figure, la numérotation des données échangées ne s'effectue pas au niveau des messages , mais bien au niveau des enregistrements. Pour l'application sous XCP1, la seule différence tient au nombre d'enregistrements par message - UN seul.

C'est au fournisseur de services XCP1, supportant l'application émettrice, d'incrémenter le compteur d'émission de la session, le SSRU-Nb, permettant ainsi de transmettre les messages avec une numérotation croissante.

Le fournisseur de services XCP1 de l'application réceptrice se charge, quant à lui, d'acquitter les messages qui lui sont transmis et d'autoriser l'application émettrice d'envoyer les messages suivants à l'aide du compteur RSRU-Nb dont il dispose.

Pour des raisons de sécurité, ces messages sont journalisés dans les fichiers SWAP alloués aux applications, tant au niveau de l'application émettrice que réceptrice. Toutefois, ces journalisations sont provisoires.

Ainsi, comme l'illustre le schéma suivant, le moniteur transactionnel TDS supportant l'application réceptrice permet au TDS supportant l'application émettrice de libérer les SRUs qu'il a journalisées.



(Fig 2.3.)

Lorsque la session est réétablie en cas de déconnexion, une reprise des échanges de messages et une resynchronisation de ceux-ci entre les correspondants sont exécutées.

Cette phase est faite par les moniteurs transactionnels TDS concernés et est basée sur la négociation des numéros de messages.

Cette phase permet à deux correspondants :

- de resynchroniser leur compteur, de déterminer la position du jeton d'émission,
- de décider de retransmettre un message,

- d'avorter ou de relancer un dialogue à partir d'un point de reprise de la conversation

En cas de désaccord, si la négociation échoue, les transactions sont gelées et un Rollback de celles-ci est effectué.

7.3.4 Règles de recouvrement

Afin que le recouvrement puisse être effectué, plusieurs règles doivent être respectées par les programmes de l'application transactionnelle distribuée, les TPRs (Transactionnal Processing Routines):

- Pour les sessions principales, tous les messages envoyés par une TPR de l'application via le moniteur transactionnel sont recouvrables s'ils ont subi un Commit. En d'autres termes, un point du Commit doit suivre l'envoi de ce message. Sous TDS, ce point se situe en fin de TPR.

- Pour les sessions auxiliaires, les messages envoyés par une TPR de l'application via le moniteur transactionnel sont recouvrables au niveau des deux correspondants si un point de Commit suit respectivement l'envoi et la réception des messages de part et d'autre de la session.

- Un message envoyé par une TPR non suivi d'un Commit ne peut être recouvert. Si une TPR envoie plus d'un message à un correspondant, seul le dernier message envoyé peut être recouvert si un point de cohérence est effectivement pris en fin de TPR.

7.3.5 Règles de transmission particulières au moniteur transactionnel

Pour des raisons indépendantes de XCP1, mais bien pour des raisons de performances du moniteur transactionnel et de l'application qu'il supporte, un message envoyé par une TPR est transmis toujours de manière asynchrone.

Deux cas peuvent se présenter:

1. Le message transmis par la TPR est le seul de cette TPR. Dès lors il sera transmis en fin de TPR.

2. Le message transmis par la TPR n'est pas le seul de celle-ci. Dans ce cas, la décision de transmission sera différée par le moniteur jusqu'au moment où la TPR enverra un nouveau message.

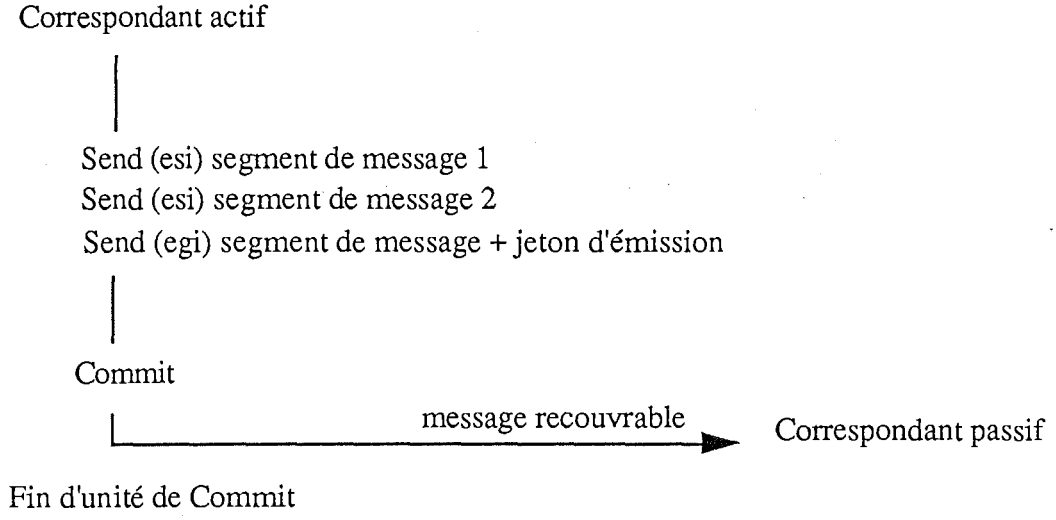
   Cet asynchronisme permet d'éviter les inconvénients de la transmission des données en temps réel et les blocages intempestifs et non planifiés

des ressources de communication que cette transmission engendre (CPU, session, réseau.etc.).

TDS met à la disposition des programmes trois types de messages à transmettre, chacun ayant une fonctionnalité spécifique. Ce sont:

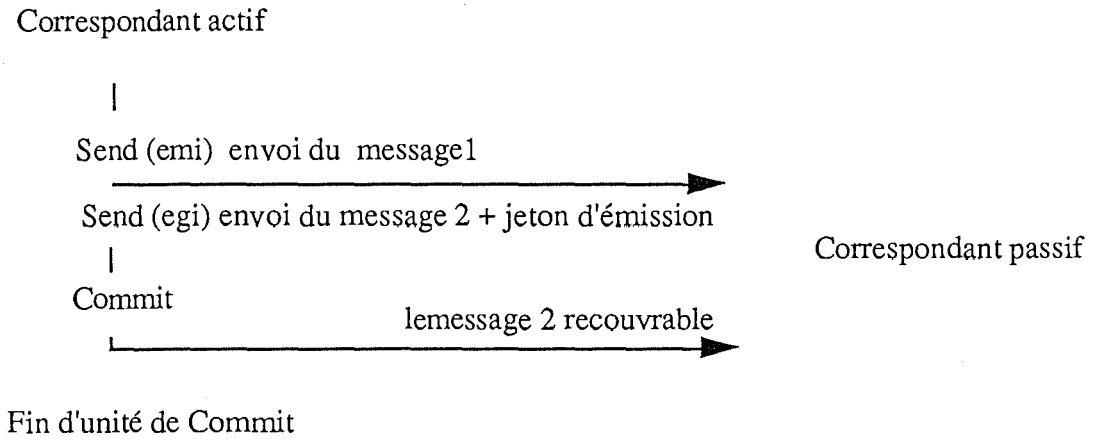
- les segments de messages sans obligation de transmettre (ESI),
- les messages avec obligation de transmettre (EMI), et
- les messages avec obligation de transmettre délégant le droit d'émettre (EGI)

Exemples : Cas 1: envoi et contention des segments de message jusqu'à la transmission effective.



(Fig 2.4.)

Cas 2: envoi de deux messages effectifs, l'un sans transfert du droit d'émission, l'autre avec transfert du droit d'émission.



(Fig 2.5.)

7.4 Primitives d'exploitation offertes

Le fournisseur de services XCP1, TDS, offre un certain nombre de primitives d'exploitation nécessaires aux applications désirant interagir avec d'autres applications. Ces primitives de services XCP1 et leurs paramètres repris ci-dessous ont été classés par groupes fonctionnels.

Gestion de la session

CP-ALLOCATE sert à allouer une session auxiliaire

Par: Cor-name (nom du correspondant)
Cp-ident (identificateur de la session)
Return-code
Status (résultat de la commande)

CP-FREE sert à libérer une session auxiliaire déjà allouée

Par: Cor-name (nom du correspondant)
Cp-ident (identificateur de la session)
Status (résultat de la commande)

Gestion des transactions

INIT-WORK sert à initialiser une transaction vers un correspondant distant

Par: Cor-name (nom du correspondant)
Cp-ident (identificateur de la session)
Transaction-name (nom de la transaction)
Security-level (niveau de sécurité 0 ou 2)
Status (résultat de la commande)

EXTRACT-WORK sert à recevoir les informations nécessaires relatives à la session principale

Par: Cor-name (nom du correspondant)
Cp-ident (identificateur de la session)
Transaction-name (nom de la transaction)
Security-level (niveau de sécurité 0 ou 2)
Status (résultat de la commande)

ABORT-WORK sert à faire avorter le dialogue avec une transaction distante que l'on a initialisé

Par: Cor-name (nom du correspondant)
Cp-ident (identificateur de la session)

Transaction-name (nom de la transaction)
Status (résultat de la commande)

SEND permet d'envoyer un message sur la session principale

RECEIVE permet de recevoir un message sur la session principale

CDSNX permet l'envoi d'un message sur n'importe quel type de session

Par: DCE-ind (Description du Correspondant)
CD (Description du message en sortie)
Status (résultat de la commande)

CDRVX permet de recevoir un message sur n'importe quel type de session

Par: DCE-ind (Description du Correspondant)
CD (Description du message en entrée)
Status (résultat de la commande)

Gestion des déconnections

ROLLBACK permet de demander le Rollback de la transaction

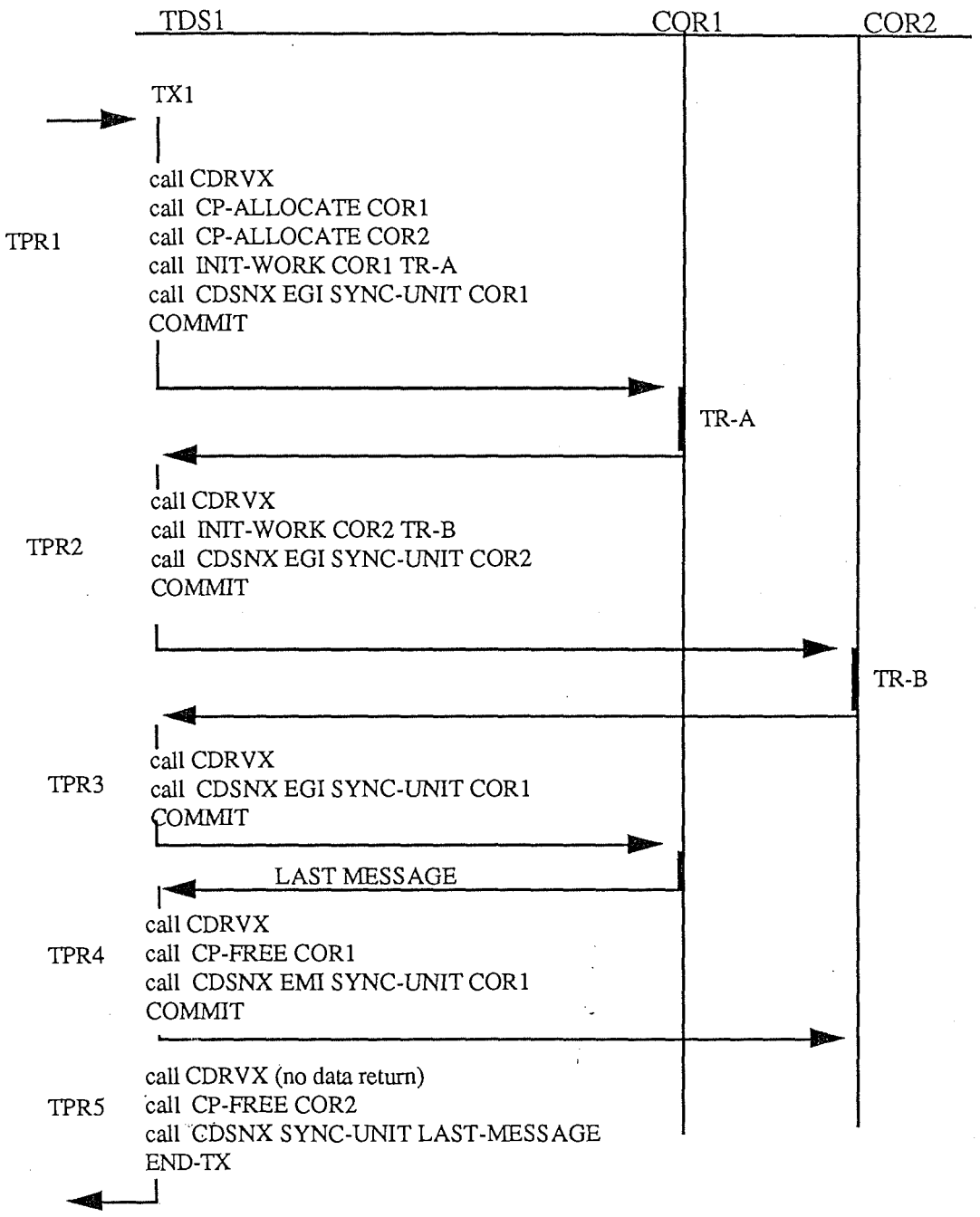
DFRECOV en cas de déconnexion permet de spécifier quelle TPR doit être relancée.

Ses paramètres sont:

Cor-name (nom du correspondant)
Cp-ident (identificateur de la session)
TPR-ident (identificateur de la TPR à lancer)
Status (résultat de la commande)

Les commandes du protocole XCP1 ne seront pas plus détaillées. Par ailleurs, le protocole de coopération XCP2 intègre ce niveau de coopération.

Le schéma suivant illustre une coopération transactionnelle impliquant trois correspondants transactionnels :



(Fig 2.6.)

7.5 Différents états liés aux conversations

Dans le protocole XCP1, un certain nombre d'états liés aux conversations sont imposés. Ces états et les transitions d'états, correspondent aux règles du protocole XCP1, sont les suivants:

Numéro d'état	Nom d'état	Significations principales
1	TO BE ALLOCATED	état initial d'une session auxiliaire
2	ALLOCATED	état initial d'une session principale ou état pour une session auxiliaire après requète CP-ALLOCATE
3	RECEIVE	la transaction doit émettre une requète RECEIVE ou CDRVX
4	SEND	la transaction courante doit émettre une requète SEND ou CDSNX
5	SEND avec SYNC-UNIT	la transaction courante doit émettre une requète SEND ou CDSNX avec paramètre SYNC-UNIT
6	SEND sans SYNC-UNIT	la transaction courante doit émettre une requète CDSNX sans paramètre SYNC-UNIT
7	COMMIT REQUESTED, RECEIVE PENDING	la transaction doit effectuer un Commit et la TPR suivante, après le point de Commit, doit émettre une requète RECEIVE ou CDRVX
8	COMMIT REQUESTED, SEND PENDING	la transaction doit effectuer un Commit et la TPR suivante après le point de Commit doit émettre une requète SEND ou CDSNX une requète SEND ou CDSNX
9	COMMIT REQUESTED, FREE PENDING	la transaction doit effectuer un Commit et la TPR suivante, après le point de Commit, doit émettre une requète CP-FREE sur une session auxiliaire ou terminer une session principale.
10	TPR-END REQUESTED	la transaction doit terminer la TPR courante sans exécuter un Commit, et la TPR suivante doit émettre une requète RECEIVE ou CDRVX
11	DISCONNECT	la transaction doit émettre une requète ROLLBACK ou DFRECOV
12	TO BE FREED	la transaction doit libérer la session

Ce tableau ne reprend que les transitions d'états principales sous XCP1. L'ensemble des possibilités de transition d'états nécessaires pour une gestion automatique de ces derniers est présenté de manière plus complète dans le document [BULL PXCP 87].

8 Protocole de coopération transactionnel XCP2

[APPC IBM 83] [GRAY 83] [MEI 87] [MUSS 89] [PAPPC IBM 86] [RAPA 85]

8.1 Introduction

8.1.1 Origine d'XCP2

Le fournisseur de services XCP2 et le protocole XCP2 furent développés, à l'origine, pour pallier aux carences en matière de traitements distribués du schéma ISO/DSA. Celui-ci était insuffisant dans la mesure où la session ISO/DSA ne pouvait offrir que des services partiels n'ayant de philosophie ni sur la synchronisation des traitements, ni sur l'intégrité des données distribuées, ni sur les reprises en cas de rupture de réseaux ou encore concernant les problèmes liés aux applications elles-même.

Ainsi quelle qu'étaient la qualité des supports et moyens de communication offerts par cette session, c'était aux applications d'acquitter et de vérifier le bon transit des informations au travers des connexions offertes (acquittement de bout en bout des données transférées).

La nécessité d'un protocole s'est avérée vitale pour le développement d'applications transactionnelles distribuées car le contrôle des connexions devenait lourd, compliqué à maintenir, voire perfectible. Le niveau de difficulté s'élève au cas où les applications sont distribuées sur des sites hétérogènes.

Une telle interface pour applications est utile dans des cas de figures transactionnelles les plus simples.

Prenons l'exemple des terminaux :

La tendance à cacher les caractéristiques et les particularités de ceux-ci renforce la notion de coopération entre applications. L'aspect terminal est devenu secondaire, surtout dans les réseaux. Ils sont pris en charge par des applications spécifiques du style "Terminal Manager" ou "Terminal Server". Une suppression de la vision directe du terminal n'est pas sans conséquences sur le degré de connectivité d'un environnement terminal et transactionnel. Le problème de sécurité des échanges d'informations peut être géré via une coopération entre les fournisseurs de services de niveau

application communicant et accédés selon des règles protocolaires.Cet exemple en est un parmi tant d'autres, mais un besoin de synchronisation d'applications existe et doit être satisfait.

Cette satisfaction passe nécessairement par des mécanismes de Commit et de recouvrements globaux et distribués. Ces mécanismes doivent pour des raisons évidentes de cohérence et de diffusion être réglés par voie protocolaire, d'où naissance du protocoles XCP2 de Bull lié au protocoles LU6.2 d'IBM.

De LU6.2, Bull en a retenu les aspects protocolaires nécessaires à une coopération transactionnelle efficace entre applications.C'est ainsi que l'aspect télécommunication relatif à la session SNA d'IBM a été écarté.

Néanmoins, sur bien des points, les éléments du protocole XCP2 sont semblables à ceux du protocole d'IBM.

8.1.2 Objectifs d'XCP2 et contraintes.

En l'absence de toute normalisation internationale, il a été décidé de promouvoir un protocole de coopération pour applications distribuées dont les objectifs étaient:

- être un protocole interne à Bull permettant aux différentes lignes de produits de s'interconnecter au niveau des applications de types transactionnelles et bureautiques.

- permettre l'interconnexion avec certains produits IBM en minimisant le coût du " gateway " propriétaire ISO/DSA/SNA et en ouvrant ainsi une porte à une plus large diffusion des produits transactionnels Bull.

XCP2 est inspiré du protocole LU6.2 d'IBM. La volonté d'IBM à travers LU6.2 était d'harmoniser en un protocole unique deux catégories de problèmes distincts:

- une première catégorie centrée sur la gestion du Commit distribué est liée essentiellement au domaine des " mainframes " centralisant les bases de données,
- la seconde vise l'échange intensif de données avec des micros ordinateurs, des machines spécialisées, des " mainframes " utilisés comme serveurs, etc.

Bull, dans la conception de son protocole s'est orienté vers la première catégorie des problèmes que prenait en charge LU6.2. Outre cette première mise

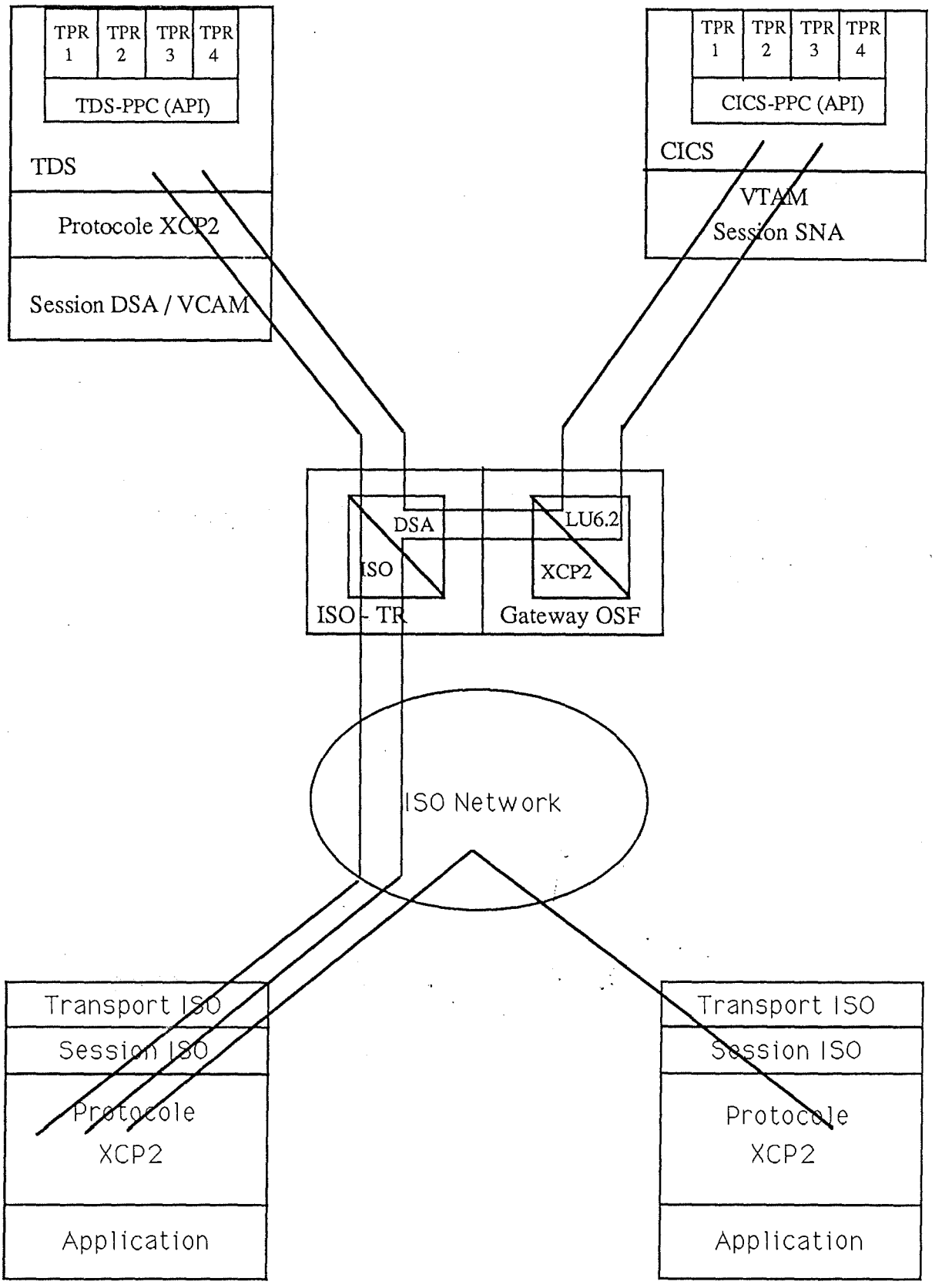
au point, Bull a pris ses distances face à la grande liberté d'action d'IBM vis-à-vis de ses protocoles réseaux, en particulier en ce qui concerne la session et les procédures administratives de raccordement SNA.

8.1.3 XCP2 et OSI de l'ISO.

Compte tenu des réserves soulevées, le fournisseur de services XCP2 et le protocole XCP2 se situeraient au-dessus de la couche session dans le modèle OSI. Il est considéré comme définissant une couche selon les critères OSI car il génère des éléments de protocoles spécifiques avec ses homologues sur des sites distants; mais il n'est pas une couche standardisée du point de vue de la norme internationale.

Basé sur la notion de coopération, XCP2 définit le comportement de programmes répartis sur plusieurs sites, ce en quoi il est plus global que le protocole de session qui ne décrit que la liaison entre deux correspondants sur le réseau. XCP2 peut gérer les relations entre N applications-programme (N non limité), selon un schéma transactionnel réparti.

Le schéma suivant présente les possibilités de connexions interapplications sous TDS.



(Fig 2.7.)

Commentaires: - API = Application Programming Interface,
- PPC = Program to Program Communication,

- les deux entités transactionnelles du bas peuvent tourner sous UNIX

8.2 Concepts XCP2, description des termes et entités

Un certain nombre de termes, de concepts doivent être définis pour la suite

8.2.1 Station de travail.

La définition est identique à XCP1. C'est un groupement logique de fonctions sur un site matérialisé par un ensemble de programmes transactionnels ayant des accès de télécommunication, c'est à dire ayant la possibilité de communiquer avec d'autres stations de travail par l'intermédiaire de sessions VTAM (IBM), VCAM (BULL) ou autres.

8.2.2 Application distribuée.

C'est une mise en relation de certains programmes d'une station de travail avec des programmes d'autres stations de travail dans le cadre d'un traitement distribué. Cette application respecte les propriétés d'atomicité, de cohérence, d'isolation et de durabilité.

8.2.3 Conversation.

Deux programmes d'une application distribuée communiquent au travers d'une conversation. Pour l'échange de leurs données, une conversation utilise une session ou un pool de sessions de communication préalablement établi entre deux stations de travail sur lesquelles tournent les deux programmes mis en conversation.

La communication des données s'effectue selon le mode TWA ("two way alternate"). Dans ce mode, l'émission et la réception des données sont dépendantes de la possession d'un jeton de droit d'émettre. Dans un dialogue, au niveau de l'échange de données, on ne peut à la fois émettre et recevoir. Les messages autres que les données ne sont pas subordonnés à cette règle. A toute conversation est associée un état de conversation .

Notons que le fournisseur de services XCP2 et le protocole XCP2 permettent des liaisons synchrones entre les divers programmes d'une application distribuée. Ils autorisent ainsi des exécution, de programmes coopérant en parallèle entre stations de travail. Une application peut ainsi avoir plusieurs conversations simultanées, chacune ayant un niveau de synchronisation indépendant des autres.

8.2.4 Pool de sessions

L'administration des pools de sessions se fait sous contrôle du fournisseur de services XCP2 (création, gestion, destruction, agrandissement...). Les pools sont typés à la création et peuvent avoir des caractéristiques différentes liées au niveau de synchronisation supporté.

L'affectation d'une session pour une conversation est du ressort du module supportant XCP2. En général, une conversation commençant sous une session se termine sous la même session, si celle-ci est rompue, dans ce cas, la reprise peut se faire sur une autre session.

Le multiplexage des conversations n'est pas de rigueur. L'affectation de sessions se fait de manière séquentielle et la fin d'une conversation libère la session.

Les pools sont typés selon quatre niveaux de synchronisation de reprise:

NONE:	aucun service de synchronisation.
XCP1:	synchronisation des messages échangés (ni doublons, ni pertes).
CONFIRM:	synchronisation par acquittement de bout en bout entre deux programmes mis en conversation.
SYNCPOINT:	Commit distribué

Ces différents niveaux sont précisés lors de la demande d'acquisition d'une session pour une conversation qui doit débuter. Ils permettent de choisir le pool de sessions adéquat duquel on extrait une session pour l'affecter à l'application.

Indépendamment de ces niveaux de synchronisation et de reprise, le choix de l'affectation pourrait dépendre d'autres caractéristiques comme la sécurité, la disponibilité, la confidentialité comme cela est d'ailleurs d'application chez IBM.

8.2.5 Commit distribué

Sous XCP2, la seule ressource pour laquelle on définit des points de reprise est la conversation de niveau SYNCPOINT. Les conversations de niveau CONFIRM ou NONE n'ont pas de points de reprise (ceci conformément au modèle LU6.2 d'IBM).

Dans un système à applications distribuées, un mécanisme de synchronisation est de mise. Les règles sont les mêmes que pour une unité de Commit locale : les mises à jour doivent être atomiques et irréversibles. Le fournisseur de services XCP2 se charge de propager les signaux de synchronisation dans tous les programmes impactés, c'est à dire ceux dont la conversation est de niveau SYNCPOINT. Le franchissement de tous les Commit locaux constitue le Commit global.

Ce Commit global constitue le point de reprise de ces conversations.
Notons que la cohérence des traitements répartis n'est pas du ressort de XCP2, elle est sous l'entière responsabilité des applications et dépend de chaque environnement d'application.

8.2.6 Arborescence des traitements

8.2.6.1 Présentation

L'ensemble des stations de travail mises en relation au sein de pools de sessions constituent un graphe des liaisons sur lequel va se greffer une application distribuée.

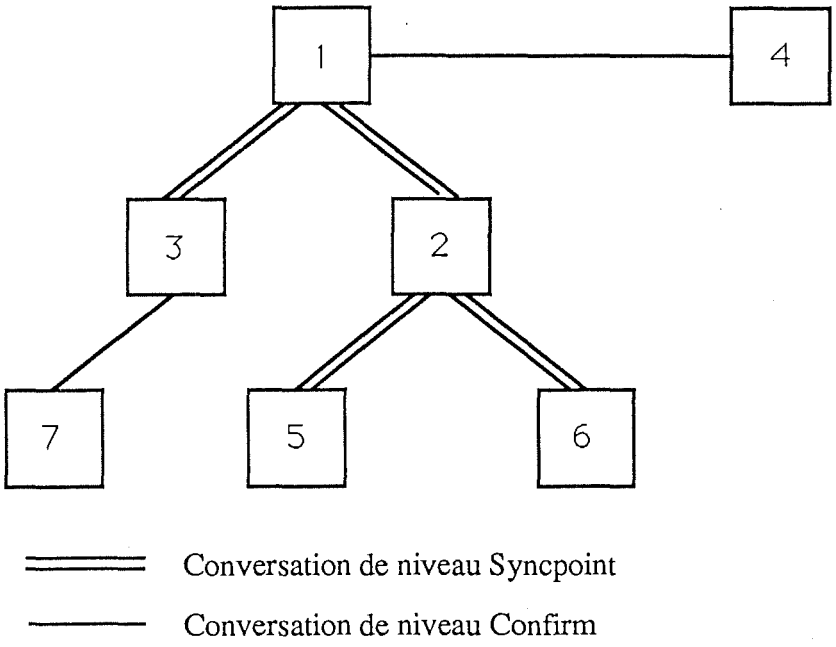
Une application distribuée va se constituer dynamiquement sur ce graphe par l'établissement de conversations entre programmes tournant sur ces stations de travail. Selon les besoins de l'application et l'emplacement des ressources impliquées, les programmes destinataires peuvent à leur tour établir une ou plusieurs conversations avec des programmes devant coopérer, ce qui conduit étape par étape à la construction d'une arborescence de traitements.

Une telle arborescence de traitements est donc constituée d'une racine, le programme qui initialise la première conversation, liée à ses subordonnés directs par des conversations représentant les arcs de l'arborescence. Chaque subordonné peut avoir ses propres subordonnés et ainsi de suite de niveau en niveau..

Pour éviter les circuits d'appels, les arborescences constituées sont de types hiérarchiques strictes. On remarquera qu'aucune entité n'a connaissance de l'arborescence complète, chacune d'entre elles n'ayant de relations qu'avec son supérieur ou ses subordonnés directs.

Le programme d'une application distribuée peut avoir autant de conversations simultanées qu'il le désire pour autant que les limites du pool de sessions soient respectées.

L'arborescence des traitements peut comporter des conversations de niveau de synchronisation différent. Le schéma suivant donne un exemple d'arborescence d'une application transactionnelle basée sur des conversations de niveau CONFIRM et SYNCPOINT.



(Fig 2.8.)

8.2.6.2 Arbre de synchronisation

Toutes les conversations sont indépendantes les unes des autres, sauf si elles sont inscrites dans un schéma de coopération de niveau SYNCPOINT. Pour un niveau de coopération autre que SYNCPOINT, la réponse à une demande de Rollback ou de prise de point de cohérence sera locale. Dans les autres cas, un Rollback ou un Commit provoquera une propagation de la demande sur l'ensemble du graphe des conversations de niveau SYNCPOINT.

Chaque programme dispose donc, ici, des mêmes droits. Ceci est fait pour autant que le fournisseur de service sXCP2 accepte de relayer l'émission de cette demande de proche en proche. Le fournisseur de services XCP2 ne connait que les conversations entre stations de travail connexes et leurs niveaux de synchronisation établis. Ainsi, aucune centralisation des données sur la structure de l'arbre des traitements n'est faite. Les services globaux de Commit et de Rollback sont réalisés dynamiquement par les fournisseurs de services XCP2.

8.3 Primitives d'exploitation

8.3.1 Introduction.

Les services offerts par XCP2 nécessitent pour leur fonctionnement, l'usage d'un nombre de commandes. Le nom des verbes génériques proposés peuvent selon le bon vouloir du programmeur différer d'une implémentation à l'autre. Ils constituent la base des possibilités de communication entre programmes.

Les paramètres associés à ces commandes, repris dans ce qui suit, sont les plus importants. Dans la mesure où leur fonction a été décrite pour une commande, ils ne seront plus expliqués par la suite. Ces commandes et leurs fonctionnalités peuvent être retrouvées assez facilement dans le cadre du protocole LU6.2 d'IBM. Ceci pour la raison évidente d'offrir des services compatibles LU6.2 et de s'inspirer largement de la notoriété et de l'expérience d'un protocole ayant fait ses preuves.

8.3.2 Commandes relatives à la session.

8.3.2.1 Allocation de session

Avant d'initialiser une conversation avec un correspondant, l'utilisateur doit acquérir une ressource session. Cette session est prise dans un pool de niveau de synchronisation et de reprise donné. Ces pools sont référencés par des noms qu'on suppose connus et standardisés.

La primitive utilisée est:

```
ALLOCATE- SESSION (     Cor-Name,
                        Service-class,
                        Queue-poss,
                        Imm-Alloc,
                        Conv-id,
                        Status )
```

ou

Cor-Name =	nom de correspondant, " mailbox " de station de travail chez Bull, adresse de LU chez IBM avec qui le programme veut dialoguer
Service-class =	fournit le niveau de synchronisation du pool de session demandé (NONE,CONFIRM, SYNCPOINT)
Queue-poss =	acceptation de mise en attente d'une session, si toutes les sessions possibles sont affectés à une conversation.

Imm-Alloc = permet la remise de la main au programme, s'il n'y a pas de session libre

Conv-id = identificateur univoque de la conversation qui va débuter. Celui-ci doit être répété pour chaque primitive

Status = fournit le résultat de la demande: rejetée ou acceptée

8.3.2.2 Désallocation de la session.

Une fois la conversation terminée (normalement ou anormalement), il faut rendre la session au pool ou demander de la conserver pour une autre conversation. L'initiative de la terminaison normale ne peut venir que du programme ayant la main (jeton d'émission) sur la conversation.

La désallocation s'effectue par l'envoi de la commande:

DEALLOCATE-SESSION: (Conv-id,
Dealloc-status,
Sync,
Status)

Le paramètre Dealloc-status peut prendre trois valeurs suivantes:

1) TERM: terminaison normale de conversation, la session est remise à la disposition du pool.
2) CONS: terminaison normale de conversation. Le programme désire garder la session pour débuter une autre conversation. Seul le programme ayant alloué la session peut le faire.
3)ABAN: la conversation est avortée et la session est rendue au pool. Peut être signifiée par l'un ou l'autre programme de la conversation.

Avant la désallocation effective, certaines actions peuvent être exigées via le paramètre:

Sync = Ce paramètre permet de disposer d'un certain nombre d'options relatives à la désallocation de la session et au niveau de synchronisation. Elle force l'exécution d'une fonction de synchronisation avant la réelle exécution de la désallocation.

Cela peut être une demande de confirmation (niveau CONFIRM) de la bonne réception du dernier message ou soit de la réalisation d'un Commit global (niveau SYNCPOINT).

8.3.3 Commandes relatives à la gestion des dialogues.

8.3.3.1 Commandes unitaires

8.3.3.1.1. Introduction.

Comme préalable à l'exposé des verbes disponibles pour la gestion des dialogues, attachons-nous à présenter les types de données échangés et les modes de conversation fournis par le fournisseur de services XCP2, et compatibilité aidant, par LU6.2.,

Quels sont les types de conversation disponibles ?

Conformément aux spécifications d'IBM, XCP2 connait deux types de conversations: l'une à l'usage de programmes systèmes (niveau "basic conversation"), l'autre pour les programmes d'application (niveau "mapped conversation").

La différence entre ces deux types de conversations se situe principalement au niveau du mode de réception et d'envoi des données:

1) Au niveau "basic conversation", la structuration des données est laissée à la charge du récipiendaire. Le fournisseur de services XCP2 ne "fait" que transmettre ce qu'il reçoit. Le programme tournant sur une station de travail dispose de deux options quant au mode de transmission des données:

a) un mode normal pour recevoir des enregistrements logiques et demander une nouvelle réception dans le cas où la longueur du message spécifié est inférieure à la longueur de l'enregistrement logique.

b) un mode " buffer " ou km pour recevoir une portion de message et suite à des demandes successives recevoir l'ensemble du message. Ici, c'est au programme de gérer la structuration des données. Ce mode lui permet, entre autres, d'accéder aux informations de contrôle des enregistrements logiques.

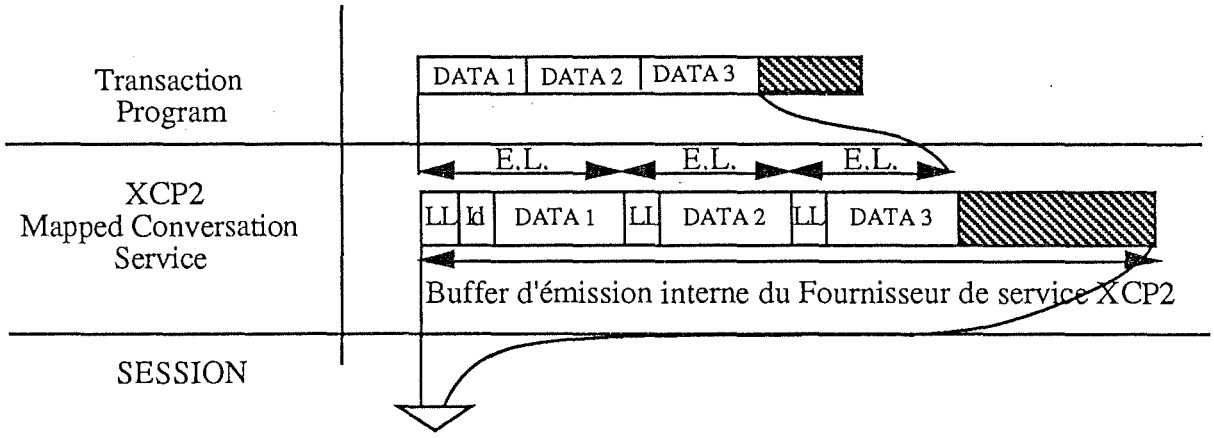
2) Il n'en est pas de même pour le mode "mapped conversation": celui-ci est orienté programmation de type utilisateur par opposition au précédent.

Si via le mode "basic conversation", le programme doit accéder à des informations de contrôle telles que la structuration de l'information transmise en enregistrements logiques, le mode "mapped conversation" libère, quant à lui, le programme de ces contraintes. Un "data mapping" est implémenté: celui-ci permet un certain nombre de transformations syntaxiques et sémantiques. Citons pour exemple le cas de la transformation d'une structure de données Cobol en une suite de caractères au niveau de l'émission. Celle-ci pourra être transformée en une structure Cobol différente, voire même en une structure de données d'un autre langage. Cette suite de caractères pourra aussi être "mappée" directement sur une structure d'écran.

Dans ce mode "basic conversation", l'utilisateur n'a pas la sensibilité de l'enregistrement logique. Selon les implémentations, le "end-user" (programme) n'a le droit de recevoir un message qu'en une fois. De plus, si la longueur demandée est inférieure à la longueur réelle, on remarque que le reste du message est perdu, ou bien que le programme reçoit le message par portion.

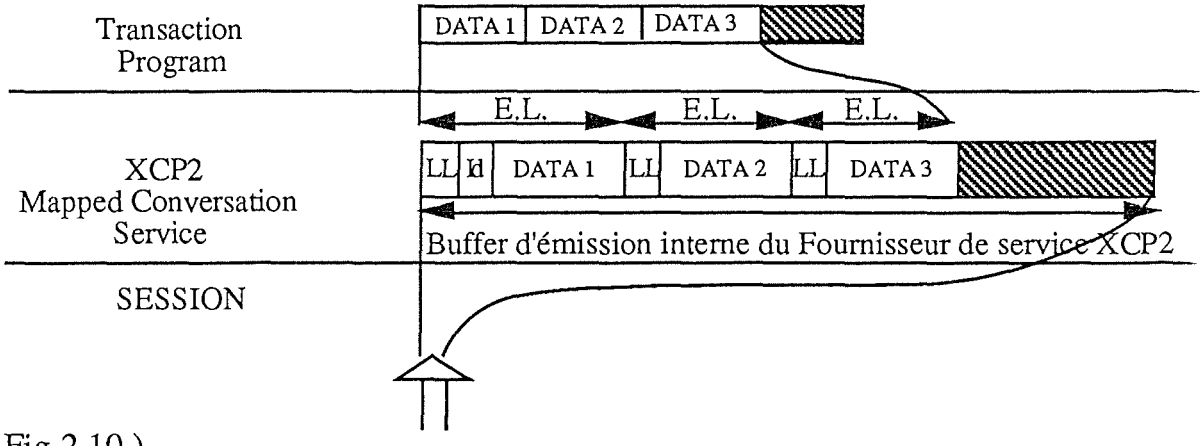
L'exploitation sous ces deux modes requiert un même type de ressource pour ranger les messages en réception et ceux en envoi, en l'occurrence un " buffer pool " au niveau du fournisseur de services XCP2.

Le schéma suivant illustre l'envoi des données en mode "Mapped conversation"



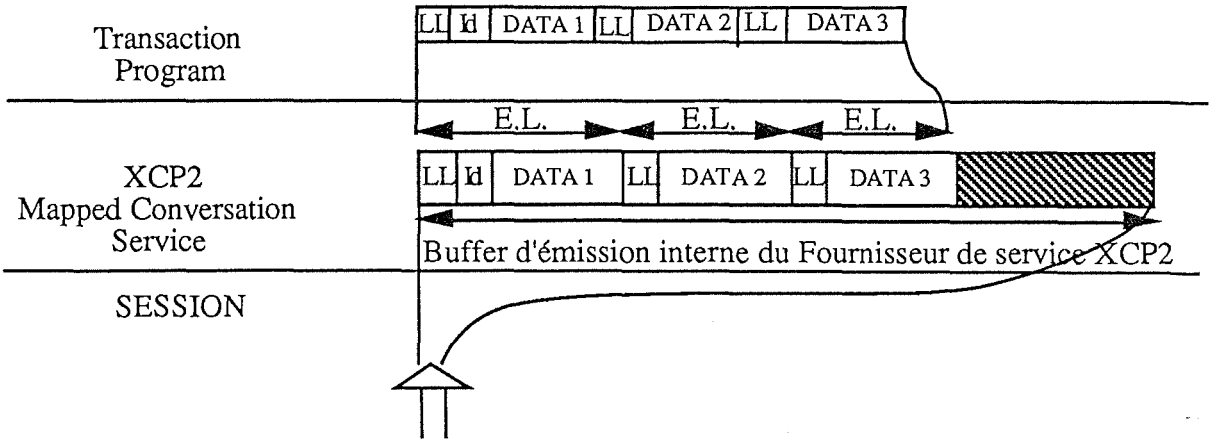
(Fig 2.9.)

Le schéma suivant illustre la réception des données en mode "Mapped conversation"



(Fig 2.10.)

Le schéma suivant illustre la réception des données en mode "basic conversation" au kilomètre.



(Fig 2.11.)

Remarques:

1) L'existence et la taille de ce " buffer pool " XCP2 sont importantes en ce sens qu'elles permettent à XCP2 de ne pas encombrer le réseau avec des messages trop courts, en groupant ceux-ci dans la limite du possible. Le fournisseur de services XCP2 va encore plus loin en adoptant une technique d'envoi différé: il prend acte d'une commande, la tamponne pour pouvoir l'ordonner dans une suite de commandes et est capable de déclencher cette commande par après, dans la limite de ses possibilités.

2) Il est intéressant de souligner la possibilité de transfert futur de nouveaux types d'objets communicables requérant un volume d'informations dont la structure n'est pas nécessairement traitable au niveau de modules tels que ceux implémentant XCP2 et LU6.2. Les images et les sons doivent pouvoir être utilisés dans des programmes coopérants. XCP2 et LU6.2 anticipent cette possibilité, à travers le mode de transfert "basic conversation" et le mode de transfert au km, tout en sachant que les problèmes de fluidité des informations dans les couches inférieures aux leurs ne sont pas résolues.

8.3.3.1.2. Initialisation d'une conversation.

Une fois la session acquise (commande Allocate-Session), le programme demandeur doit encore spécifier le programme avec lequel il veut converser et le mode de coopération effectif. En effet, le programme même s'il dispose d'un niveau de synchronisation SYNCPOINT peut s'il le désire se contenter d'un niveau inférieur, un niveau CONFIRM par exemple.

L'initialisation de la conversation se fait à l'aide de la primitive

INIT-CONVERSATION (Conv-id,
Tr-id,
Sync-Level).

où

Tr-id est l'identificateur, le nom de programme avec lequel on veut entrer en conversation.

SYNC-Level est le niveau effectif demandé sur les possibilités de la session il prend comme valeur:

soit NONE: pas de synchronisation à la charge d'XCP2
CONFIRM: acquittement de bout en bout
XCP1: Synchronisation des messages
SYNCPOINT: Synchronisation des Commits

La commande d'initialisation pour des raisons de performances sera tamponnée et bloquée jusqu'au moment où une commande subséquente forcera l'envoi du buffer d'émission ou lorsque ce dernier sera plein.

8.3.3.1.3. Désallocation d'une conversation.

La désallocation d'une session ne pose pas de problèmes. Elle peut être exécutée en utilisant la commande:

DEALLOCATE-CONVERSATION.(Conv-id)

8.3.3.1.4.Emission de messages.

L'envoi de messages à un correspondant se fait par le biais de la commande:

SEND-MESSAGE(Conv-id,
Workway-area,
Lenght,
Dialogue-id,
SYNC,
Mode,
Status)

Où

Workway-area = adresse de la zone de données à transférer

Lenght = longueur du message ou de la portion de message

Dialogue-id = indicateur de suite d'envoi, trois cas possibles:

- pas de fin d'envoi de messages
- fin de message, transfert du "jeton d'émission"

- fin de conversation

SYNC: optionnel selon le niveau de conversation déclaré, on demande que les mécanismes de confirmation (CONFIRM), de Commit ou de synchronisation (XCP1, SYNCPOINT) soient enclenchés.
La conversation , dès lors, est mise dans un état "prêt à commiter" ou "prêt à synchroniser" et est à l'entière disposition du programme. La conversation est donc mise dans un état intermédiaire où une invalidation est possible.

L'action d'invalider peut correspondre au choix d'un niveau de synchronisme plus restreint ou plus exigeant ou bien au choix de ne rien faire.

Mode: en cas de demande de confirmation de reception des données envoyées,le fournisseur de services XCP2 fournit deux mode de services: un synchrone et un autre asynchrone. Dans le cas du mode asynchrone, le fournisseur de services XCP2 rend le contrôle au programme et ce dernier recevra le résultat de la confirmation sur la commande WAIT. Dans le cas du mode synchrone, le fournisseur de services XCP2 garde le contrôle du dialogue enpèchant tout nouvel envoi de message sur la conversation usitée.

8.3.3.1.5. Réception de messages ou d'événements

RECEIVE-INFORMATION (Conv-id,
Press-type,
Lenght,
Working-area,
Length-rec,
Dialogue-id
Sync-evt,
Status)

où

Press-type = paramètre permettant à un programme pour des conversations de type "basique" de recevoir les données, enregistrement logique par enregistrement logique ou bien de les recevoir sous forme d'un message de longueur spécifié par le paramètre "lenght".

Lenght = longueur du message acceptée

Working-area = adresse de la zone de travail où doit être rangé le message.

Length-rec = correspond à la longueur réelle du message si la longueur du message < "working-area" ou correspond à la longueur du message non transféré si la longueur du message > "working-area"

Dialogue-id = indique qu'un changement de "turn" est demandé par le correspondant ou qu'une fin de conversation est en cours

Sync-evt = indique qu'un événement de synchronisation est demandé: une confirmation, un Commit global.

Status = indique en outre que la totalité du message n'a pas été transmise

8.3.3.1.6. Transmission des erreurs.

Une détection d'erreurs peut se produire dans n'importe quel état de conversation, en émission ou en réception. Elle peut être issue du programme conversant ou d'un fournisseur de services XCP2 ou autres. Dans tous les cas, les erreurs sont notifiées au programme distant ou local par l'intermédiaire du code d'erreurs Status présent comme paramètre de chaque commande.

La propagation de la détection d'erreurs doit être prioritaire aux traitements usuels. Pour ce faire le programme qui émet SEND-ERROR reçoit obligatoirement le droit d'émission du fournisseur de services XCP2 sur les conversations impliquées, pour envoyer des informations complémentaires au programme distant.

La syntaxe de cette commande est:

SEND-ERROR(Conv-id,
Error-type)

Il est à noter que le programme constatant l'erreur peut demander au fournisseur de services XCP2 auquel il est attaché de terminer proprement la conversation avant de lui donner le droit d'émission. Ceci peut être réalisé via une instanciation particulière du paramètre Error-type.

8.3.3.1.7. Demande de confirmation .

La demande de confirmation au programme distant de la réception d'un message peut être émise d'une manière autre que par le biais de la commande SEND-MESSAGE.

On peut utiliser de manière plus spécifique la commande:

CONFIRM(Conv-id,
Mode)

L'attente de la réponse peut être synchrone ou asynchrone (paramètre "Mode").

8.3.3.1.8. Réponse à la demande de confirmation.

La réception d'une demande de confirmation entraine la conversation dans un état spécifique où l'on ne peut que :

-confirmer la réception des données,
-Envoyer un message d'erreur,
-terminer brutalement la conversation

Les commandes: CONFIRMED (Conv-id),
SEND-ERROR,
DEALLOCATE-SESSION sont respectivement utilisées.

8.3.3.1.9. Altération du tamponnage des commandes

Pour raisons de performances, le fournisseur de services XCP2 peut tamponner les commandes et conserver les données à transmettre dans son buffer d'émission. La commande FLUSH force la transmission vers la couche session du buffer d'émission de la conversation dans l'état où il est.

Cette commande offre deux possibilités:

1) d'activer la prise en compte par le programme distant des commandes émises.
2) d'annuler une opération de synchronisation ou de Commit projetée. Le forcage de l'envoi du buffer indique au fournisseur de services XCP2 qu'il n'y a plus de phase de Commit active.

FLUSH (Conv-id)

8.3.3.1.10 Demande polie de droit d'émission.

Par cette requète, un programme demande le droit d'émettre sur une conversation.Cette demande est invoquée via la commande présentée ci-après. Le programme destinataire possède le droit d'accepter ou de refuser cette demande de transfert de contrôle.

REQUEST-TO-SEND (Conv-id).

8.3.3.2 Commandes relatives à la gestion simultanée de plusieurs conversations.

8.3.3.2.1 Introduction

Les programmes utilisant les services du module XCP2 ne se contentent pas toujours d'un seul lien conversationnel, ceci, entre autres pour des raisons de performances. De ce fait, chaque programme a la possibilité de mener plusieurs conversations en parallèle, il peut être en attente d'événements sur certaines de ces conversations et les traiter dans leur ordre d'arrivée. Même si celui-ci ne correspond pas à l'ordre d'initialisation ayant engendré l'attente de ces événements.

Le fournisseur de services XCP2 met à la disposition des programmes utilisateur un mécanisme de "dispatching" dédié à un ensemble de conversations qui au préalable ont été déclarées comme mises sous surveillance asynchrone. Ceci se fait pour l'utilisateur par les deux commandes suivantes: POST-ON-RECEIVE et WAIT-INFORMATION.

8.3.3.2.2 Déclaration des conversations "dispatchées".

Le choix des conversations "dispatchées" est à la charge du progamme, c'est à lui de déterminer le ou les conversations concernées et de transmettre son choix au fournisseur de services XCP2.

La déclaration d'appartenance à cet ensemble de conversations se fait dynamiquement, à l'aide de la commande suivante:

```
POST-ON-RECEIVE ( List-of-conv-id,
                  List-of-press-type,
                  List-of-lenght,
                  Status)
```

où List-of-conv-id = est la liste des identificateurs de conversations sur lesquels un dispatching est opéré. Instancié à "all", toutes les conversations sont impliquées.

List-of-pres-type = est la liste des formats de données relatifs aux données attendues sur chaque conversation (voir la commande RECEIVE-INFORMATION)

List-of-lenght = est la liste des longueurs des données relatives aux formats de présentation de la List-of-pres-type

8.3.3.2.3 Prise en compte d'événements sur conversations dispatchées

Le réveil et l'obtention d'événements sur conversation se fait à l'aide d'une commande disponible pour le programme ayant déclaré les conversations

"dispatchées". Le principe est que chaque exécution de cette commande est susceptible d'informer le programme de la disponibilité d'un événement sur une des conversations concernées. Pour obtenir les évènements de toutes les conversations, il est donc nécessaire de boucler sur cette commande:

WAIT-INFORMATION (List-of-conv-id,
Conv-id-awake,
Status)

où

list-of-conv-id = indique le sous-ensemble des conversations visées par la commande.

conv-id-awake = est l'identificateur de la conversation dont l'événement est disponible

status = est un paramètre permettant de distinguer le type d'événement reçu (données, changement de droit d'émission, fin de conversation, demande ou réponse de confirmation, demande de Commit, message d'erreurs...).

La mise à disponibilité d'un événement d'une conversation, via la commande WAIT-INFORMATION, a pour effet de retirer cette conversation de la liste des conversations en attente. POST-ON-RECEIVE doit être utilisé pour la réintroduire dans la liste.

La mise à disponibilité ne signifie nullement la réception des données liées à l'événement en question. Ces données peuvent être réceptionnées par le programme via la commande RECEIVE-INFORMATION.

8.3.3.3 Commandes relatives au Commit distribué et à la gestion des reprises.

Sous XCP2, l'ensemble de la phase de Commit est lancé via l'unique commande SYNCPOINT. Les effets de cette dernière est de propager la demande de Commit vers toutes les conversations de niveau SYNCPOINT. La synchronisation des traitements se fait selon un protocole de Commit à deux phases [BERN 87]. Les conversations de niveau NONE ou CONFIRM sont non-affectées.

La syntaxe de la commande est:

SYNCPOINT (Status)

Cette commande de synchronisation, comme toutes les autres commandes précédentes, est à la disposition de l'utilisateur. Le module de service supportant XCP2 devra, quant à lui, gérer ce processus via une dynamique de Commit en deux phases .

Le paramètre Status rend compte du résultat du déroulement des opérations de Commit dans l'ensemble de l'application distribuée. S'il y a refus de Commit d'un des programmes, toutes les conversations de niveau SYNCPOINT sont ramenées à leur dernier point de reprise. En cas de rupture de réseau pendant le Commit, le fournisseur de services XCP2 permet au programme utilisateur de gérer de

manière heuristique le problème. Si l'utilisateur ne saisit pas cette opportunité, la seule solution qui lui reste est Rollback qu'il pourra initialiser à l'aide de la commande BACKOUT (Status).

> Cette commande propage une demande de retour de toutes conversations de niveau SYNCPOINT à leur dernier point de reprise. Les conversations de type CONFIRM ou NONE sont avortées afin de remettre l'environnement coopératif transactionnel à un niveau cohérent.

Si tout se passe bien la prise de Commit définit un point de reprise des conversations.

8.3.4 Le protocole de commit.

Ce protocole se déroule en deux phases.

Phase n° 1:

Elle consiste à amener l'ensemble des programmes de l'application distribuée (du moins tous les programmes du graphe de synchronisation) à un état dit de "Commit pending".
L'état de "Commit pending" correspond au verrouillage des ressources impliquées dans le Commit, buffers, variables locales et conversations inclus. C'est dans cette phase qu'on obtient l'accord de tous les programmes pour valider les mises à jours des fichiers: le refus d'un programme est matérialisé par une demande de Rollback généralisé (l'application émet la commande BACKOUT).

Phase n° 2:

Cette phase n'est activée que si la phase 1 est réussie. C'est une phase de validation, c'est-à-dire que toutes les mises à jour sont considérées comme légales et devront être appliquées quelques soient les incidents. Cette phase correspond au déverrouillage des ressources; elle n'implique pas les écritures immédiates des mises à jour sur les fichiers concernés, celles-ci pouvant s'exécuter plus tard. Cela n'est qu'un problème d'implémentation. On a souvent interêt à ce que les phases 1 et 2 soient les plus courtes possibles.

Pour des raisons de performances, la mise à jour réelle des fichiers peut être différée. C'est ainsi qu'une sous-phase de mise-à-jour effective est souvent incluse dans la phase 2. En bonne logique, la validation n'est pas l'écriture.

8.3.4.1 Verbes réalisant le protocole de Commit en deux phases.

Quatre commandes entrent en action pour exécuter une procédure de Commit. Ces primitives : RQ-COMMIT, PREPARE, COMMITED et FORGET sont utilisées par les fournisseurs de services XCP2. Les deux premières commandes permettent de remplir le travail de la première phase du protocole, les deux autres de la deuxième phase.

RQ-COMMIT: permet au gestionnaire de Commit local de demander à un seul gestionnaire de Commit distant de prendre le jeton d'émission et d'exécuter un Commit.

S'il y a plusieurs conversations de niveau SYNCPOINT, le gestionnaire local va demander à tous les autres gestionnaires distants de lui soumettre l'indication RQ-COMMIT, cela en réponse à la commande PREPARE qu'il aura émise.

Le programme local entre dans l'état "commit pending" (phase 1).

PREPARE: le gestionnaire local demande à un gestionnaire distant de lui envoyer une requète RQ-COMMIT. Lorsqu'il recevra l'indication d'émission de cette requète, cela signifiera qu'il accepte la demande de Commit global, sinon à la place de la requète RQ-COMMIT, c'est la requète de Rollback BACKOUT qui lui sera envoyé.

COMMITED: verbe de la phase 2. Dès que le Commit local est validé, le gestionnaire local envoie vers tous les gestionnaires distants qui lui ont fait parvenir une indication RQ-COMMIT une indication "COMMITED".

FORGET: verbe de la phase 2. Lorsque le Commit local est validé, le gestionnaire de Commit local envoie une requète FORGET vers tous les gestionnaires distants qui lui ont fait parvenir une demande PREPARE. Le tout est de transmettre à l'initiateur du Commit distribué le résultat positif de sa commande.

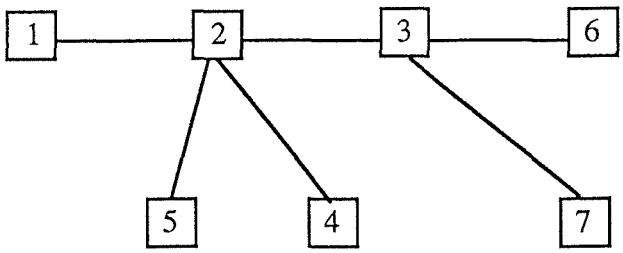
Le principe de l'algorithme est de sélectionner un gestionnaire qui ne reçoit que des ordres RQ-COMMIT et ne peut émettre le verbe PREPARE. C'est ce gestionnaire qui validera en premier et ensuite de proche en proche, les gestionnaires de Commit entreront à leur tour en phase de validation. Cet algorithme repose sur la hiérarchie des invocations de programme.

Citons un exemple pour montrer le fonctionnement de cet algorithme:

Exemple d'une application distribuée où 3 est le programme initiateur. Il est entré en conversation avec les programmes 6, 7 et 2.

Le programme 2 à son tour est entré en conversation avec les programmes 1, 3 et 4. La logique de l'application amène à ce que le programme 1 soit l'initiateur du Commit distribué.

Shéma de l'application distribuée:



(Fig 2.12.)

Commentaire sur la dynamique du Commit:

Pour plus de facilités appelons les programmes " P " et les gestionnaires de Commit dont ils dépendent " GC ".

Début de la première phase:

- P 1 émet une demande de Commit (SYNCPOINT)
- GC 1 en réponse émet la commande RQ-COMMIT vers P2
- P 2 accepte la demande de Commit
- P 2 émet une demande de Commit (SYNCPOINT)
- GC 2 en réponse émet la commande PREPARE vers P3, P4, P5
- P 4 accepte la demande de Commit
- P 4 émet une demande de Commit (SYNCPOINT)
- GC 4 en réponse émet la commande RQ-COMMIT vers P2
- P 5 accepte la demande de Commit
- P 5 émet une demande de Commit (SYNCPOINT)
- GC 5 en réponse émet la commande RQ-COMMIT vers P2
- P 3 accepte la demande de Commit
- P 3 émet une demande de Commit (SYNCPOINT)
- GC 3 en réponse émet la commande PREPARE vers P6, P7
- P 6 accepte la demande de Commit
- P 6 émet une demande de Commit (SYNCPOINT)
- GC 6 en réponse émet la commande RQ-COMMIT vers P3
- P 7 accepte la demande de Commit
- P 7 émet une demande de Commit (SYNCPOINT)
- GC 7 en réponse émet la commande RQ-COMMIT vers P3
- GC 3 ayant reçu toutes les indications RQ-COMMIT émet la commande RQ-COMMIT vers P2

La première phase est terminée.

- GC 2 ayant reçu toutes les indications RQ-COMMIT émet la commande COMMITED vers P1, P3, P4, P5

- En réponse, GC3 émet la commande COMMITED vers P6, P7
- Suite à çà,GC3, GC4, GC5, GC6, GC7 émettent la requète FORGET vers P2, P2, P2, P3, P3 et CG2 vers P1

8.3.4.2 Les problèmes posés par l'algorithme de diffusion du Commit

Du point de vue de XCP2, les programmes de 1 à 7 ont des droits identiques: chacun aurait pu être l'initiateur d'une demande de Commit.

Cet algorithme suppose que les temps de verrouillage des ressources ne jouent pas de rôle majeur. Dans l'exemple précédent, les programmes 1,5,4,7,6 gardent leurs verrous plus longtemps que les programmes 2 et 3. Or si le site où se trouve le programme 6 par exemple est celui où est géré la base de données principale, il est important de valider en premier lieu sur ce site. L'algorithme dans son fonctionnement ne le permet pas.

Le progamme qui reçoit la toute première demande de RQ-COMMIT est celui qui va valider en premier. Si dans l'exemple précédent, le programme 3 avait été l'initiateur d'une demande de Commit global, le gestionnaire de Commit du

programme 3 aurait eu le choix d'émettre RQ-COMMIT vers l'un des programmes 2, 6, 7 et PREPARE vers les deux autres.

Ces considérations montrent que, bien que l'on veuille cacher au programmeur le déroulement de la synchronsation des Commits, celui-ci doit en fait être conscient de l'algorithme utilisé afin de réduire au maximun les temps de verrouillage des ressources partagées jugées comme primordiales par l'exploitation.

D'où il peut-être utile dans la commande SYNCPOINT de permettre à l'utilisateur, lorsqu'il y a un choix entre plusieurs conversations, de diriger la première requète RQ-COMMIT.

La commande SYNCPOINT est du point de vue du programme appelant strictement synchrone. Le programme émetteur ne reprend le contrôle que lorsque le résultat a été acquis. Il serait important pour certains programmes systèmes d'avoir la visibilité des deux phases, permettant de reprendre ainsi le contrôle entre les deux phases et par exemple de déverouiller préventivement certaines ressources.

8.4 Etat d'une conversation.

8.4.1 Introduction.

Un certain nombre d'états conventionnels sont attachés à une conversation, au niveau des correspondants. Ce nombre d'états dépend du niveau de synchronisation requis lors de l'allocation de la conversation.

Les transitions d'état sont assurées par l'émission d'un verbe local ou par l'action d'une commande émise par l'application avec laquelle l'application locale correspond. Ces états sont DEFER, WAIT, RECEIVE, SYNCPOINT, CONFIRM, RESET, SESSION ALLOCATED.

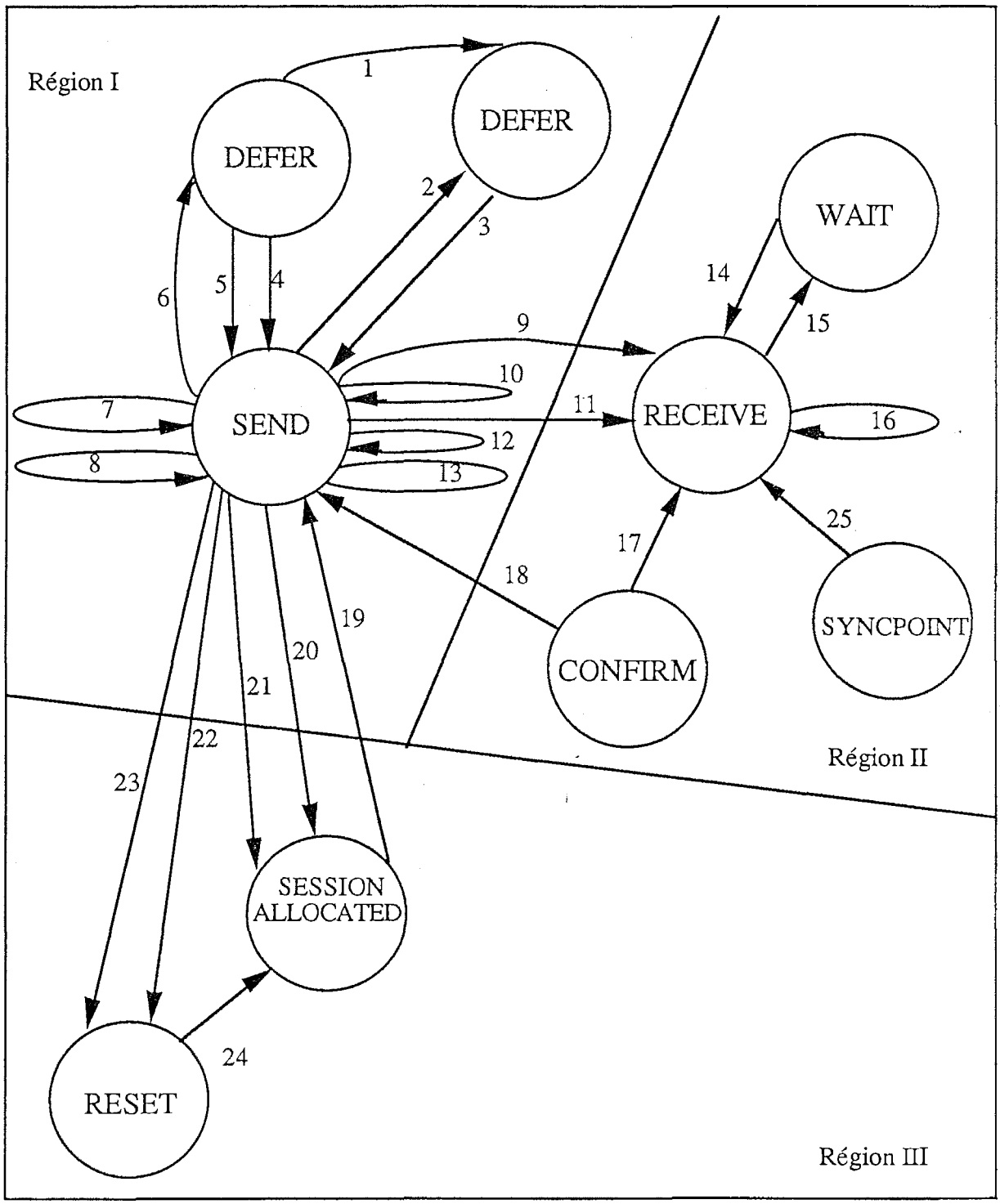
8.4.2 Régions d'état.

Pour plus de facilités, l'environnement des états d'une conversation peut être découpé en trois régions selon le critère du droit d'émission sous XCP2. Rappelons-nous que le mode conversationnel considéré est ici un mode "two way alternate" pour les échanges de données et un mode "two way simultaneous" pour les autres verbes en général.

Le changement d'état est signalé à l'application principalement par le biais du paramètre Status associé aux différentes commandes.

Le tableau suivant donne un aperçu des transitions possibles entre ces états. La liste complète de celles-ci peut être trouvée dans [RAPA 85].

Région I: "émission - droit de d'émettre".
Region II : "réception - pas de droit d'émettre".
Région III: "hors droit d'émission".



(Fig 2.13.)

Les commandes relatives aux arcs sont les suivantes :

arc 1 : CONFIRM asynchrone
arc 2 : SEND message + CONFIRM asynchrone
arc 3 : WAIT-INFORMATION
arc 4 : FLUSH
arc 5 : SYNCPOINT
arc 6 : SEND message + SYNCPOINT
arc 7 : SEND message
arc 8 : SEND-ERROR
arc 9 : SEND message avec jeton d'émission
arc 10 : SYNCPOINT
arc 11 : BACKOUT
arc 12 : SEND message + CONFIRM asynchrone
arc 13 : FLUSH
arc 14 : WAIT-INFORMATION
arc 15 : POST-ON-RECEIVE
arc 16 : RECEIVE-INFORMATION
arc 17 : CONFIRMED
arc 18 : SEND-ERROR
arc 19 : INIT-CONVERSATION
arc 20 : SEND message + fin de conversation
arc 21 : DEALLOCATE-CONVERSATION
arc 22 : DEALLOCATE + libération de la session
arc 23 : DEALLOCATE-SESSION
arc 24 : ALLOCATE-SESSION

---

9. PROTOCOLE ISO 10026 relatif au traitement des transactions distribuées

[ISO TP1 90] [ISO TP2 90] [ISO TP3 90]

9.1. Introduction

Tout comme XCP2, ce protocole ISO 10026 à été inspiré du protocole IBM LU6.2 du point de vue de sa fonctionnalité globale. Toutefois, une adaptation a du être opérée pour respecter l'esprit modulaire de l'ISO, ceci sur base des protocoles déjà existant et adoptés au niveau international. Les protocoles sur lesquels se base la norme 10026 sont principalement ceux existant au niveau 6 et 7 de la "pile" OSI permettant l'interconnexion de systèmes ouverts

Ce protocole ISO 10026 relatif au traitement des transactions distribuées comporte trois parties: un Modèle TP ("Transactionnal Processing"), un Service TP et un Protocole de communication lui permettant d'utiliser les services déjà disponibles au niveau de la couche application du modèle de référence OSI. Ce protocole OSI 10026 est un des standards structurant la couche application, il prend en charge les informations identifiables comme étant des transactions pouvant impliquer un ou plusieurs systèmes ouverts.

Notre attention va porter sur le Service TP. Celui-ci offre des facilités suffisantes pour supporter le traitement transactionnel distribué et pour établir un cadre de coordination pour les ressources TP multiples impliquées des systèmes ouverts coopérant dans le cadre de ce traitement.

Le corps de ce Service TP est constitué d'un fournisseur de services (TPSP="Transport Protocol Service Provider") et d'un ensemble de primitives permettant à un utilisateur (TPSUI="Transport Protocol Service User Invocation") d' accéder aux facilités offertes par ce TPSP. Ce sont sur ces primitives et sur la dynamique d' enchaînement de celles-ci que va porter notre intérêt. Auparavant, situons un peu l'environnement transactionnel tel que le définit l'ISO.

9.2. Situation du protocole ISO 10026 dans le modèle de référence ISO

9.2.1. Introduction

Le modèle de référence conçu par l'ISO se caractérise par une conception modulaire de l'activité de communication entre sites correspondant. Le but poursuivi est de permettre, avec un minimum d'accords techniques externes aux standards ISO, d'interconnecter des systèmes informatiques de différents constructeurs, et le plus souvent de techniques hardwares et softwares différentes.

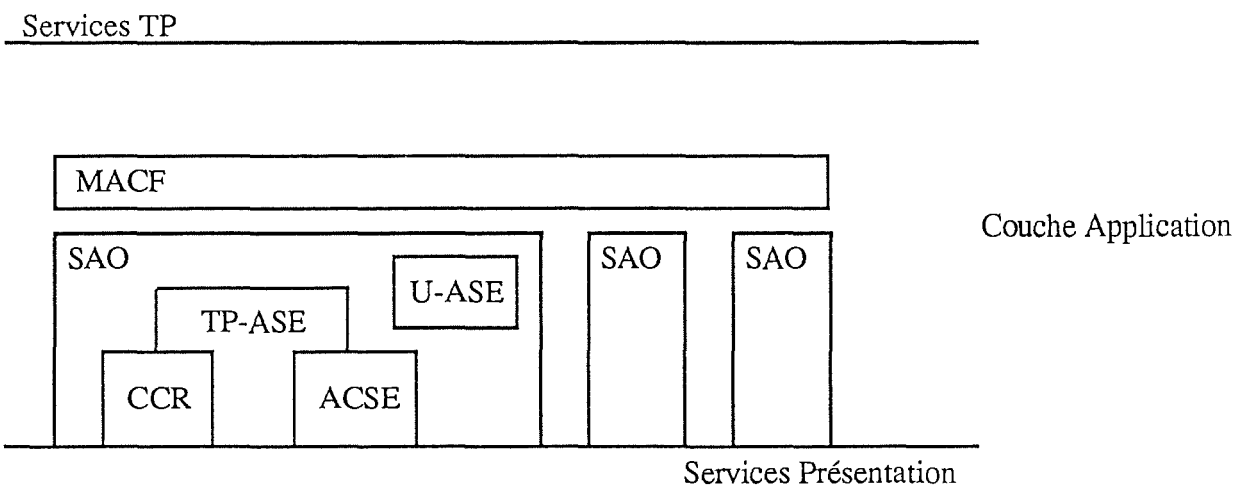
Certains protocoles de niveau Application et Présentation ont été, depuis longtemps, adoptés au niveau international. C'est sur une partie de ces éléments de protocoles que les services de traitements transactionnels reposent.

9.2.2. Vue schématique des éléments de protocole TP

Les primitives de services TP reposent en grande partie sur des primitives de services d'autres modules.

Ce sont : - le module ACSE,
- le module CCR.
- les services de la couche Présentation

Schématiquement, au niveau Application cela donne:



(Fig 2.14.)

où MACF = Multiple Association Control Function,
SAO = Single Association Object,
TP-ASE = Transactionnal Processing - Association Service Element,
U-ASE = User - Association Service Element
ACSE = Association Control Service Element,
CCR = élément de Commitment, Concurency et Recovery.

9.2.3. Relations entre éléments TP dans la structure de la couche Application

L'ensemble des services de niveau TP est contrôlé et accédé par chaque TPSUI via un élément MACF. Cet élément MACF transforme les primitives de service TP en primitives de service SAO. Comme on peut le constater sur le schéma précédent, un MACF peut contrôler plusieurs SAO, constituant par là un pool de SAOs.

Chaque SAO dans l'environnement de protocole TP ne gère qu'une et une seule association entre applications.

Ce SAO contient les éléments de service suivant:

TP-ASE: il repose sur les éléments de service CCR et sur les éléments de service de la couche Présentation et regroupe les fonctions spécifiques du traitement transactionnel;
ACSE: il est utilisé pour établir et terminer les associations. Les fonctions ACSE ne sont pas activées via les primitives de service TP, mais bien par le Fournisseur de services TP;
CCR: il fournit le support des fonctions de Commit, de Rollback et de recouvrement. Les fonctions CCR sont utilisées directement par le module TP-ASE ou par l'élément MACF;
U-ASE: il représente toutes les autres fonctions de niveau Application-Association fournissant des services directement transposables en services de niveau Présentation.

Le mode d'échange de données entre éléments impliqués dans le protocole TP ne diffère pas de ce qui existe au niveau des couches inférieures du modèle de référence OSI. Un mécanisme de concaténation des données et d'enlacement des données est de mise.

La structure TP, ici présentée devrait, selon les publication de l'ISO, combler les exigences des utilisateurs en termes de performances transactionnelles, de sécurité et de cohérence des échanges tout en préservant l'indépendance des applications envers les moyens de communication.

9.3. Présentation des concepts relatifs au traitement transactionnel sous ISO 10026

Tout comme les protocoles précédents XCP1 et XCP2, un certain nombre de concepts et termes doivent être définis.

Le protocole ISO 10026 étant fonctionnellement dérivé de LU6.2, il est intéressant de comparer ses termes et concepts avec ceux d'XCP2, protocole lui aussi dérivé d' LU6.2.

Nous retrouvons de part et d'autre des termes et concepts qui, sous un nom différent, remplissent les même fonctions. C'est ainsi que les concepts repris sous les noms TPSUIs, dialogues, arbres de dialogues se retrouvent sous XCP2 sous les termes, respectivement, de stations de travail, conversations et de graphes de liaisons entre stations de travail.

Sous le protocole OSI 10026, un arbre de synchronisation sera appelé arbre transactionnel. Une différence entre les deux protocoles à mettre ici en évidence, est que les niveaux de synchronisme dans l'arbre transactionnel sous 10026 sont limités à deux: soit NONE, soit COMMITMENT.

Les arbres de dialogues et les arbres transactionnels, sous le protocole 10026 ont la caractéristique d'être modifiables dynamiquement comme sous XCP2.

10026 se différencie d'XCP2 en mettant à disposition des utilisateurs un "canal de communication interapplications" supplémentaire à usage du traitement transactionnel. Ce "canal" permet aux applications désirant communiquer de ne pas passer par les TPSUIs, ceci pour des raisons de recouvrement ou de synchronisation.

Comme on peut le constater à ce niveau, les éléments conceptuels de base sont plus ou moins semblables.

9.4. Vue des services offerts par le Service TP

Le fournisseur de services TP et son protocole sont soucieux de créer un environnement dans lequel un ou plusieurs utilisateurs peuvent interagir en toute sécurité de manière complète et cohérente. A cette fin, les actions entreprises par un utilisateur de ces services peuvent être:

a) débuter un dialogue pour échanger avec un partenaire son identité et le mode de dialogue souhaité,

b) invoquer les services provenant d'autres éléments de services de niveau application sous contrôle du fournisseur de services TP,

c) coordonner le travail sous différents niveaux de coopération.

Soit - un échange de données seul (niveau de coopération NONE)
- un échange de données avec prise de Commit et Rollback disponible (niveau de coopération COMMITMENT),

d) délimiter les unités transactionnelles supportées par le TPSP,

e) préparer un Commit, effectuer un Commit ou un Rollback d'une transaction,

f) permettre un traitement heuristique d'un Commit en cas de problèmes,

g) signaler des erreurs de coopération,

h) terminer normalement un dialogue et permettre la libération de toutes les ressources allouées à ce dialogue,

i) terminer un dialogue anormalement, lorsque cela est nécessaire

j) synchroniser les traitements

k) avoir la possibilité de chaîner des transactions ou de les séparer les unes des autres.

Tous ces services sont accessibles via un ensemble de commandes standards exposé par la suite. Si l'on compare les fonctionnalités des commandes XCP2 ou LU6.2 aux commandes ISO 10026, on remarque que les primitives des protocole Bull et IBM remplissent la fonction de plusieurs commandes ISO. La paramétrisation des commandes Bull et IBM est plus élaborée.

9.5. Description des unités fonctionnelles TP

Dans le cadre de ce protocole, l'ensemble des services a été réparti en unités fonctionnelles liées au TPSP .

Les six unités suivantes ont été définies:

1. Kernel : Cette unité fonctionnelle regroupe l'ensemble des services de base pour débuter un dialogue, signaler des erreurs aux correspondants, recevoir des indications de problèmes de ses correspondants ou de son TPSP et terminer un dialogue normalement ou anormalement.

2. Shared Control: L'unité fonctionnelle "Shared Control" permet aux TPSUIs de correspondre suivant les contraintes "normales" sur les primitives.

3.Polarized Control: Cette unité fonctionnelle regroupe l'ensemble des fonctions permettant à un correspondant (TPSUI) d'avoir le contrôle sur un dialogue (en mode "Two Way Alternate"). Ce contrôle est déléguable. Les unités fonctionnelles "Shared Control" et "Polarized Control" sont mutuellement exclusives.

4. Handshake: L'unité fonctionnelle "Handshake" permet à une paire de TPSUIs de synchroniser leurs traitements l'un en fonction de l'autre.

5. Commit: L'unité fonctionnelle Commit supporte les Commits et les Rollbacks fiables sur les transactions

6. Unchained transaction: Cette unité permet aux TPSUIs de dialoguer en ayant un niveau de coordination nul ou de Commit selon leurs volonté de manière dynamique. Ils peuvent choisir ou non de se soumettre aux contraintes liées à une transaction spécifique.

9.6. Primitives relatives aux unités fonctionnelles

Les différentes unités fonctionnelles sont accédées par un ensemble de primitives de service qui leur sont associées. Chaque primitive permet d'initialiser une action relative à une fonction de l'unité fonctionnelle concernée.

La table, ici présentée énumère et associe les différentes primitives de services aux unités fonctionnelles du point 4.5. Le détail et l'explication de ces éléments de service seront donnés par la suite.

Unité fonctionnelle	Primitives de services	Paramètres
1. Kernel :	TP-BEGIN-DIALOGUE	Initiating-AE-Title Initiating-TPSU-Title Recipient-AE-Title Recipient-TPSUTitle
	TP-P-REJECT	Diagnostic [Rollback]
	TP-U-REJECT	[UserData] [Rollback]
	TP-END-DIALOGUE	
	TP-DATA	
	TP-U-ERROR	

	TP-P-ERROR	Diagnostic
	TP-U-ABORT	[UserData]
		[Rollback]
	TP-P-ABORT	Diagnostic
		[Rollback]

2. Shared Control: Aucune primitive de services particulière.

3.Polarized Control:
TP-GRANT-CONTROL
TP-REQUEST-CONTROL

4. Handshake:
TP-HANDSHAKE
TP-HANDSHAKE-AND-END
TP-HANDSHAKE-AND-GRANT-CONTROL

5. Commit:

TP-DONE	[heuristic Report]
TP-COMMIT	
TP-CONTINUE-COMMIT	
TP-COMMIT-RESULT	
TP-COMMIT-COMPLETE	
TP-PREPARE	
TP-READY	
TP-ROLLBACK	
TP-ROLLBACK-COMPLETE	
TP-DEFERRED-END-DIALOGUE	
TP-DEFERRED-GRANT-CONTROL	

6. Unchained transaction

TP-DEFER-NEXT-TRANSACTION	
TP-UNCHAIN-TRANSACTION	
TP-BEGIN-TRANSACTION	[Chain]

9.7. Relation entre le fournisseur de services TP (TPSP) et les utilisateurs (TPSUIs)

9.7.1. Introduction

Les principes de base de la dynamique d'enchaînement des primitives de services de niveau TP s'inscrivent dans la lignée des protocoles définis pour les autres couches de la pile OSI. Des requêtes sont adressées par un utilisateur des services TP, via le TPSP, à un utilisateur correspondant. Le TPSP, dès lors, en réponse à ces requêtes, transmet les indications de ces dernières au correspondant visé de manière effective.

Dans un souci de clarté, la présentation des interactions entre les TPSUIs et le TPSP ont été groupées de manière telle que les primitives utilisées se rapportent aux unités fonctionnelles auxquelles elles appartiennent.

9.7.2. Unité fonctionnelle de base, unité Kernel

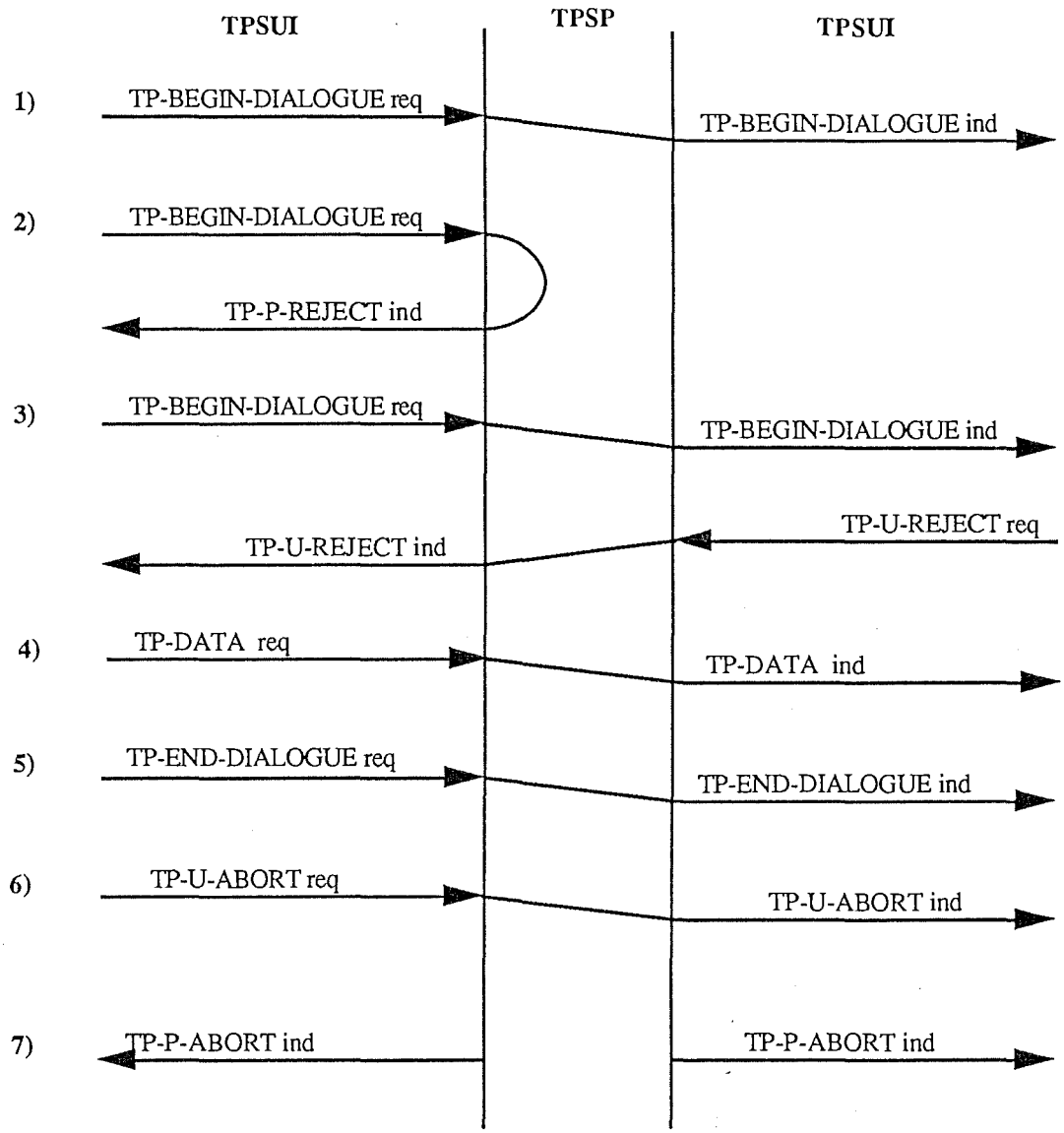
Les schémas dynamiques ci-dessous présentent la dynamique d'enchaînement des primitives transactionnelles assurant les services transactionnels de base nécessaire à une coopération entre les applications.

Ces services permettent:

- d'établir et de terminer normalement un dialogue,
- de signaler une erreur en provenance du TPSP ou du TPSUI correspondant,
- d'envoyer des données au TPSUI,
- de terminer un dialogue de manière anormale.

Les cas de figures suivants sont relatifs à:

1) à l'établissement d'un dialogue,
2) à l'établissement d'un dialogue et rejet de cette demande par le TPSP,
3) à l'établissement d'un dialogue et rejet de cette demande par le TPSUI correspondant,
4) au transfert de données,
5) à la terminaison normale d'un dialogue,
6) à la terminaison anormale de dialogue sous l'initiative d'un TPSUI,
7) à la terminaison anormale de dialogue sous l'initiative d'un TPSP.



(Fig 2.15.)

Notons que les paramètres des primitives TP-U-ABORT, TP-P-ABORT, TP-U-REJECT, TP-P-REJECT permettent, déjà à ce stade, au fournisseur de services ou au correspondant visé de demander un Rollback sur le dialogue établit. Ce Rollback n'étant possible que pour les dialogues ouverts de niveau COMMITMENT.

9.7.3. Mode de contrôle des dialogues

9.7.3.1. Présentation

A l'aide des services TP, deux modes de dialogues peuvent exister. Le premier, peu restrictif, est un dialogue en mode partagé ("Shared Control Mode"), l'autre, plus astreignant, est un mode de dialogue hiérarchique ("Polarized Control Mode").

La différence entre ces deux modes de dialogues tient aux possibilités de contrôler les échanges d'informations.

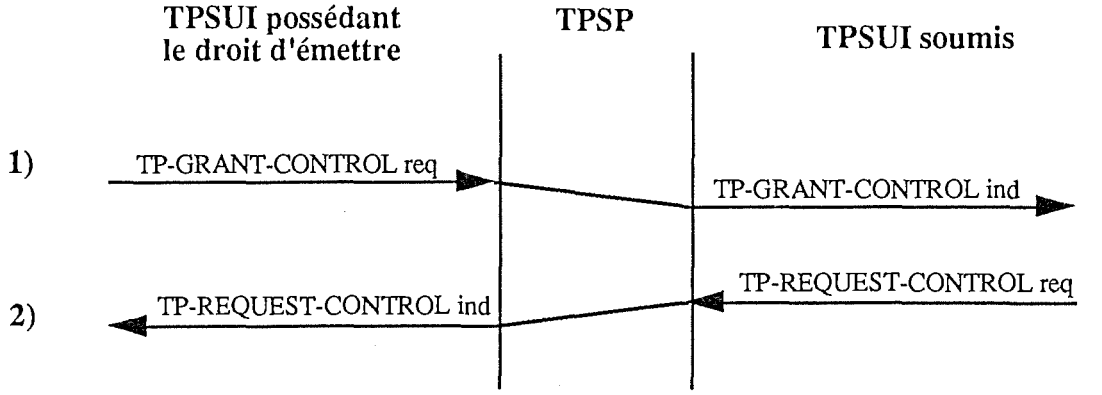
Dans le premier de ces modes, aucune contrainte hiérarchique entre les correspondants n'existe. Ils sont sur le même pied d'égalité quant aux droits d'émettre des primitives, de transmission des données, ou d'erreurs, etc.

Dans le second de ces modes, le mode "Polarized Control", des restrictions hiérarchiques supplémentaires sont introduites par rapport au mode "Shared Control". Un bon nombre de requêtes ne peuvent être émises que par le TPSUI contrôlant le dialogue, c'est-à-dire celui qui détient le droit d'émission. Les primitives de base de la gestion du droit d'émission sont TP-GRANT-CONTROL et TP-REQUEST-CONTROL.

9.7.3.2. Gestion du contrôle des dialogues

Le contrôle d'un dialogue ne s'effectue donc que sous mode "Polarized Control". Le TPSUI autorisé à gérer le dialogue possède un droit d'émettre. Ce droit est matérialisé par un jeton d'émission unique pour chaque dialogue.

Ce jeton d'émission peut être transmis par le TPSUI contrôlant le dialogue à son homologue correspondant. Le TPSUI n'ayant pas le contrôle du dialogue peut quant à lui demander une transmission du droit de contrôle du dialogue. Cette dernière requête sera soumise à la bonne volonté du TPSUI contrôlant le dialogue.



(Fig 2.16.)

1) Transfert du droit de contrôle
2) Demande de transfert de contrôle

9.7.4. Gestion des synchronisations

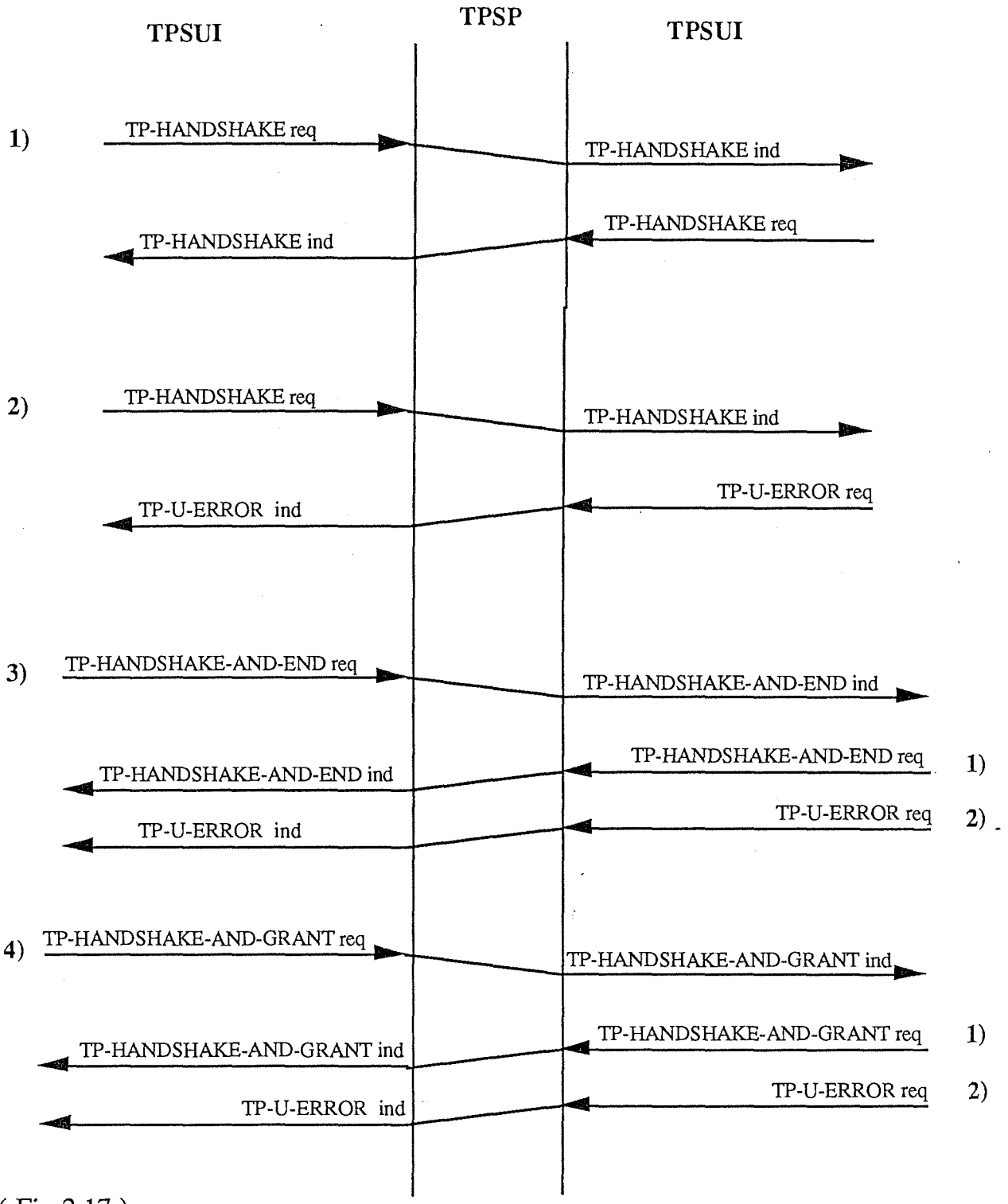
Les requêtes utilisées, ici, ont été regroupées dans l'unité fonctionnelle "Handshake". Ces primitives permettent au TPSUI local et à son TPSUI correspondant, qui peut être plus ou moins éloigné, de synchroniser leurs traitements l'un envers l'autre. La sémantique d'une synchronisation est: " une confirmation est demandée pour un point de traitements connu de l'utilisateur ". Ces points de traitements dépendent des applications transactionnelles développées et du degré de distribution de ces dernières.

La possibilité d'émettre une requête de synchronisation est liée au mode de contrôle des dialogues. Ainsi, en mode "Polarized Control", seul le TPSUI contrôlant le dialogue est autorisé à émettre une demande de synchronisation.

Il est à remarquer que quel que soit le mode de contrôle des dialogues, il n'existe pas de priorité des commandes de synchronisation sur les commandes usuelles telles que celles servant au transfert des données ou des erreurs. C'est ainsi que toutes les données en transit seront délivrées par le TPSP avant l'indication de demande de synchronisation.

Le protocole ISO 10026 offre un ensemble de commandes pour synchroniser les dialogues entre TPSUIs. Certaines de ces commandes ne mettent pas simplement en oeuvre un processus de synchronisation, mais aussi selon chacune de ces commandes, un transfert de contrôle de dialogue ou une demande de terminaison de dialogue.

Le schéma suivant illustre les possibilités d'agencement de primitives en vue de synchroniser les traitements de deux TPSUIs.



(Fig 2.17.)

1) Séquence de primitives avec synchronisation possible
2) Séquence de primitives avec synchronisation impossible
3) Séquence de primitives avec synchronisation + fin de dialogue possible (1) et impossible (2)
4) Séquence de primitives avec synchronisation + transfert de contrôle de dialogue possible (1) et impossible (2)

Les deux dernières séries de primitives (série 4 et 5) ont l'avantage de regrouper une suite d'actions et ainsi d'éliminer quelques communications inutiles. Il est à noter que l'émetteur de ces requêtes agrégées, ne peut rien transmettre entre le moment de l'émission de la requête et le réception de la confirmation ou de l'échec en réponse à cette requête en provenance du TPSUI correspondant.

9.7.5. Fonction de reprises

L'ensemble des primitives regroupées dans l'unité fonctionnelle de Commit permet de pallier aux problèmes de cohérence pouvant survenir lors d'une transaction distribuée. Les différentes primitives permettent de garantir les propriétés "ACID" (voir point 2.2.2.) des transactions concernées. Les deux mécanismes de base de cette gestion des reprises, bien connus, sont les mécanismes de Commit et de Rollback.

9.7.5.1. Gestion du Commit

Tout comme XCP2, la procédure de Commit se base sur un Commit à deux phases. La première phase sert à amener l'arbre de coopération transactionnel à l'état "READY". Dans cet état, tous les traitements et tous les transferts de données sont complètés. A ce stade, il existe une double possibilité d'action: soit d'atteindre le stade final (Commit), soit revenir à l'état initial de la transaction (Rollback).

Quel est l'enchaînement des primitives concernées?

Phase 1: Préparation du Commit

La primitive TP-PREPARE est la primitive de départ. Elle permet à un TPSUI de demander au sous-arbre transactionnel qui lui est soumis d'entrer dans l'état "READY". Lorsque le sous arbre en question est prêt, il l'indique à son TPSUI supérieur par l'envoi de la primitive TP-READY.

Les primitives TP-COMMIT et TP-CONTINUE-COMMIT sont utilisées pour indiquer que les données contrôlées par le TPSUI sont dans l'état "READY" et pour demander à tous les autres noeuds subordonnés de se mettre dans l'état "READY".

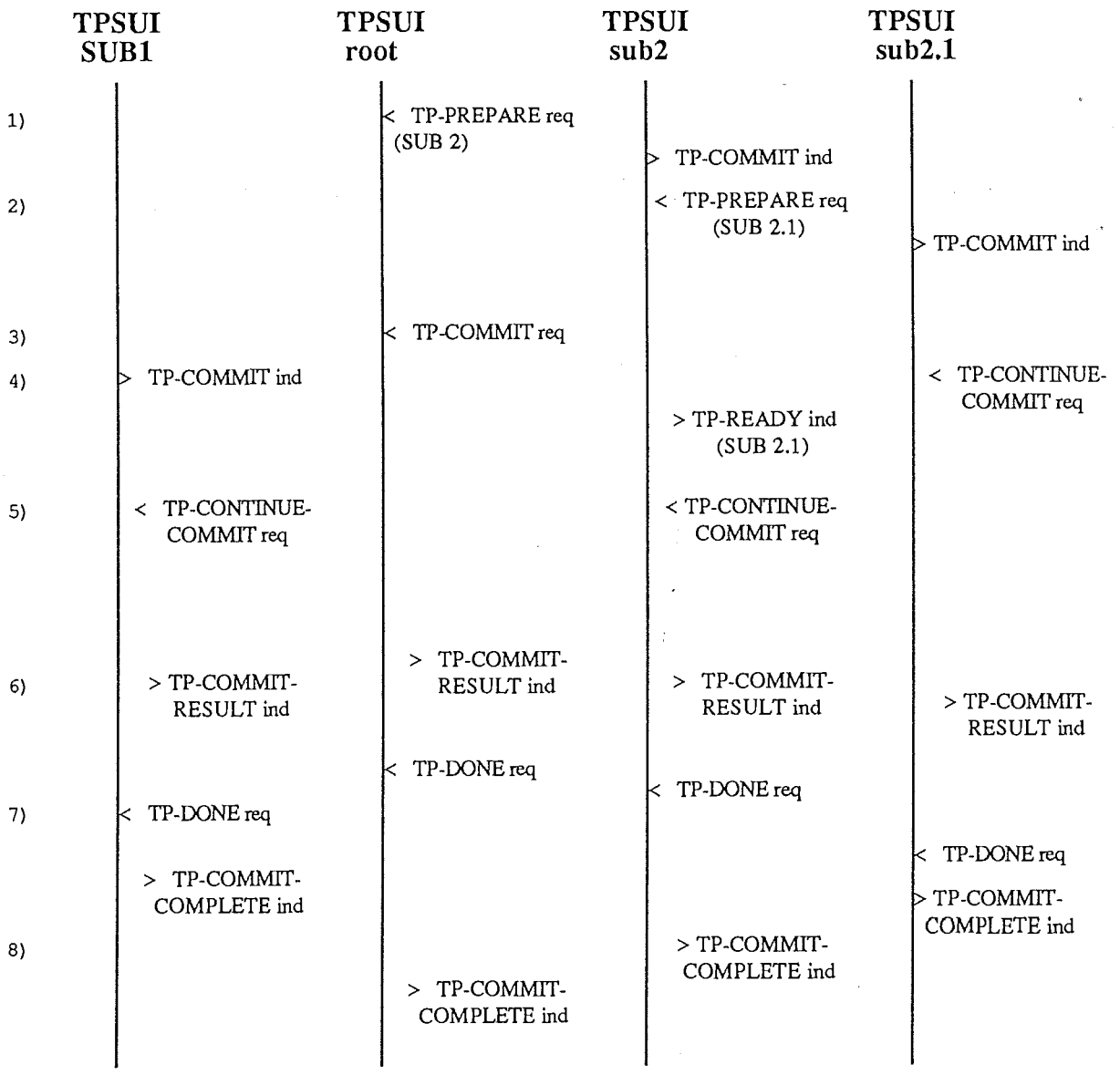
Arrivé à ce point, le TPSUI ne peut plus activer un Rollback et doit attendre une indication de TP-COMMIT-RESULT (ou TP-ROLLBACK) provenant des TPSUIs qui lui sont soumis.

Phase 2: Prise effective du Commit

La fonction de la primitive TP-COMMIT-RESULT ind informe donc un TPSUI que la transaction est en instance de Commit et demande à ce TPSUI de placer toutes les données sous un contrôle supérieur. A cette indication, le TPSUI concerné par cette demande va émettre une requête TP-DONE.

Suite à cette émission,le TPSUI devra attendre la primitive TP-COMMIT-COMPLETE ind. Cette dernière l'informe que la transaction dans laquelle il était inscrit s'est terminée de manière complète et cohérente à son stade. Ceci ne préjuge en rien de la réussite de l'activité de Commit en amont du TPSUI.

Le schéma suivant donne un exemple de l'enchaînement des primitives pour une transaction distribuée entre quatre TPSUIs lors d'un Commit initialisé par le TPSUI " racine " de l'arbre transactionnel. Cet exemple exclut tout problème en provenance du fournisseur de services TP ou d'un des TPSUIs.



(Fig 2.18)

Commentaires :

Dans cet exemple, seul le TPSUI sub2 est concerné par une préparation de Commit.

1) Le TPSUI "racine" de la transaction émet une requête de préparation de Commit, TP-PREPARE req, en direction du TPSUI sub2.

2) Le TPSP transmet au TPSUI concerné l'ordre de préparation, qu'il va lui-même transmettre à ses subordonnés.

3) Le TPSUI "racine" de la transaction émet une requête de Commit, TP-COMMIT, pour le TPSUI sub1 non impliqué dans la préparation de ce Commit .

4) Le TPSP indique au TPSUI sub1 adjacent, la volonté du TPSUI "racine" d'exécuter un Commit à son niveau.

5) Chaque TPSUI, à une demande de Commit ou à une préparation de Commit, émet une requête TP-CONTINUE-COMMIT. Dans un arbre transactionnel plus fourni, ceci aurait pour effet de propager l'ordre de Commit à tous les autres TPSUIs subordonnés coopérants.

6) Si toutes les requêtes TP-CONTINUE-COMMIT et TP-COMMIT ont été envoyées, le TPSP leur fait parvenir une indication TP-COMMIT-RESULT à chaque TPSUI pour confirmer que lui et tous les autres TPSUIs ont tous la possibilité d'exécuter un Commit.

7) Une fois le Commit complet au niveau local, chaque TPSUI envoie une requête TP-DONE.

8) Si tous les Commit locaux se sont déroulés sans problèmes, le TPSP conclut le Commit global par l'envoi de l'indication TP-COMMIT-COMPLETE à tous les TPSUIs.

9.7.5.2. Gestion du Rollback

Dans le cadre d'une conversation de niveau Commit, tous les TPSUIs, impliqués dans la transaction distribuée à un moment ou à un autre, peuvent demander aux TPSUIs partenaires de réinitialiser leurs traitements; en d'autres termes, revenir à l'état initial de la transaction.

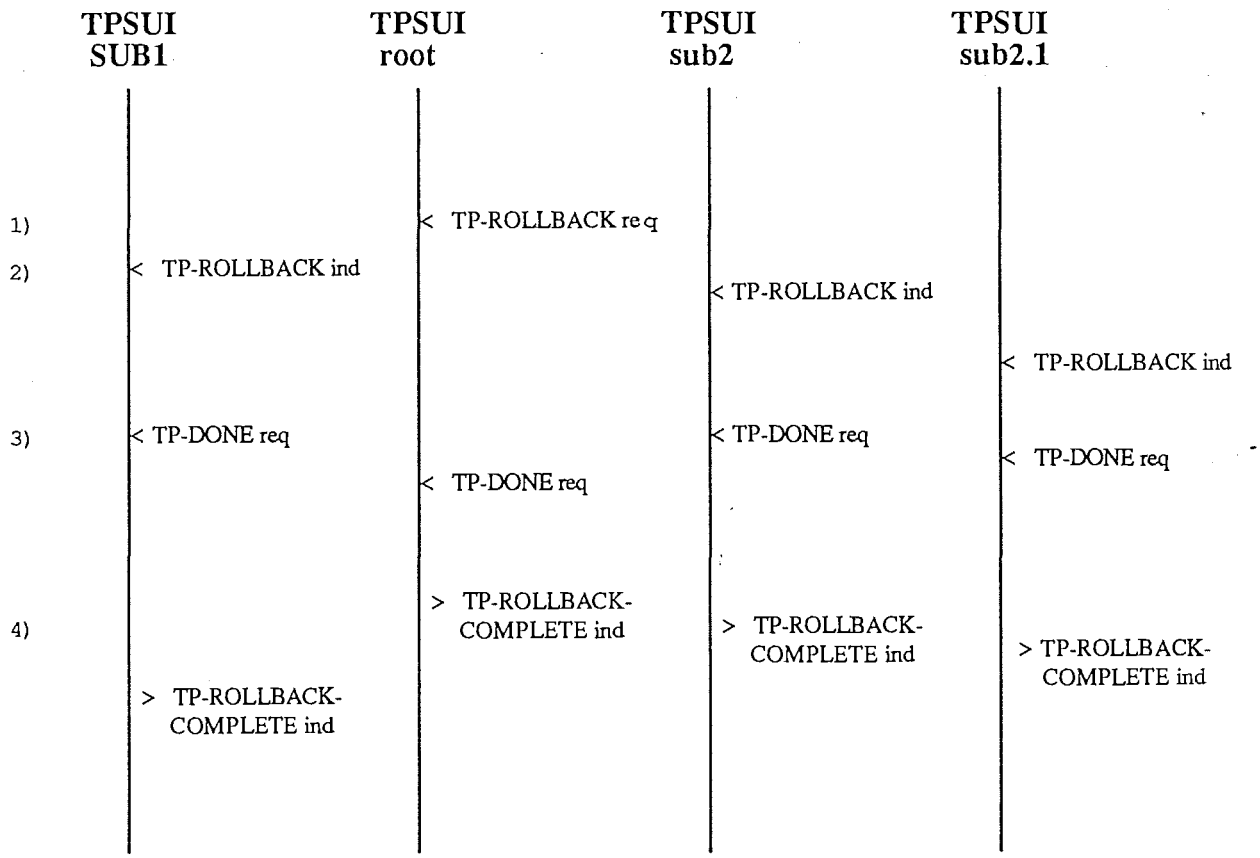
Pour réaliser sa demande, le TPSUI dispose de plusieurs commandes. La commande la plus directe est la requête TP-ROLLBACK. En réponse à cette requête, les autres TPSUIs de la chaîne de traitement concernés se devront d'émettre une requête TP-DONE lorsque leur activité de Rollback locale sera accomplie. D'autres commandes également disponibles pour une telle demande, sont les requêtes TP-U-ERROR et TP-U-ABORT où les paramètres Rollback sont instanciés.

L'ordre de Rollback peut aussi émaner du fournisseur de services TP. Au cas où une ou plusieurs conditions de bonne exécution de la transaction distribuée ne sont pas satisfaites, le TPSP indiquera aux TPSUIs impliqués qu'un Rollback est

nécessaire. Les primitives d'indications sont TP-P-ERROR et TP-P-ABORT où les paramètres Rollback sont instanciés.

Si nous suivons le déroulement normal de la phase de Rollback, une fois que le TPSUI a émis sa requête TP-DONE, on remarque qu'il se met en attente d'une indication de bonne exécution de Rollback de l'ensemble de l'arbre transactionnel. Cette exécution est matérialisée par la réception de la primitive TP-ROLLBACK-COMPLETE.

Le schéma suivant donne un exemple de l'enchaînement des primitives pour une transaction distribuée entre quatre TPSUIs lors d'un Rollback initialisé par le TPSUI " racine " de l'arbre transactionnel.



(Fig 2.19.)

Commentaires:

1) Une requête de Rollback est émise au niveau du TPSUI "racine" de la transaction

2) Le TPSP fait parvenir des indications TP-ROLLBACK aux TPSUIs concernés

3) Chaque TPSUI effectue un Rollback de ses variables locales et émet ensuite une requête TP-DONE

4) Une fois tous les TP-DONE émis, le TPSP conclut le Rollback en faisant parvenir une indication de fin de Rollback à tous les TPSUIs.

La dynamique de Rollback est séquentielle au niveau local.
Au niveau global, aucun ordre temporel n'est exigé.

9.7.5.3. Gestion heuristique

Un Rollback peut s'avérer excessif dans certains cas. C'est pour cela que la norme ISO 10026 autorise une possibilité de gestion heuristique des problèmes de cohérence pouvant survenir lors de la phase de Commit.

Ainsi, les primitives TP-DONE et TP-P-ERROR peuvent être utilisées respectivement par un TPSUI ou un TPSP pour indiquer non seulement que le point de cohérence ne peut être atteint, mais aussi qu'une décision heuristique de traitement du problème est souhaité. Pour ce faire, seule l'instanciation du paramètre "Diagnostic" de ces primitives est nécessaire pour déléguer une telle gestion au TPSUI supérieur .

9.7.6. Gestion dynamique du niveau de coordination transactionnel

Le protocole OSI 10026 autorise un dialogue à disposer des deux niveaux de synchronisation NONE et COMMITMENT à différents moments.

La possibilité de disposer de ces deux niveaux sur le même dialogue résulte d'une décision prise par le TPSUI ayant initialisé le dialogue. Celui-ci soumet le dialogue au contrôle de l'unité fonctionnelle "Unchained Transaction".

Sous ce type de contrôle, le TPSUI peut impliquer un TPSUI subordonné dans une conversation par l'usage de la requête TP-BEGIN-TRANSACTION. A ce niveau, deux cas de figure sont envisageables en fonction des paramètres de la requête émise.

1) si le paramètre "Chain " n'est pas instancié, le dialogue revient automatiquement à un niveau de coordination NONE dès que la transaction supportée est terminée, après un Commit ou un Rollback.

2) si le paramètre "Chain " est instancié, lorsque la transaction supportée est terminée, le niveau de coordination du dialogue persiste pour la transaction suivante.

Dans ce cas, deux primitives supplémentaires sont disponibles:
TP-DEFER-NEXT-TRANSACTION et TP-UNCHAIN-TRANSACTION

1) TP-DEFER-NEXT-TRANSACTION:

Elle est utilisé durant la phase active de la transaction supportée pour spécifier que le niveau de dialogue doit être remis à NONE.si la transaction subit un Commit. L'effet de cette primitive est nul si on effectue un Rollback de la transaction.

2) TP-UNCHAIN-TRANSACTION:

Elle est utilisée durant le Rollback ou le Commit de la transaction supportée lorsqu'une erreur apparaît dans l'arbre transactionnel. Par cette primitive, le TPSUI ayant initialisé le dialogue remet le niveau de synchronisation du dialogue à NONE en fin de transaction.

Conclusion

Ce travail, dans sa première partie, a proposé une approche globale d'un utilitaire performant, supportant des applications distribuées, à savoir le moniteur transactionnel de Bull TDS.

Nous avons vu que le moniteur transactionnel TDS n'est pas un programme "indépendant". Ses performances et le niveau d'interconnection avec des systèmes hétérogènes qu'il peut offrir aux applications qu'il supporte sont liés aux qualités des composants de l'Operating System de Bull, GCOS7.

Néanmoins, cette approche peut ne pas être strictement cantonnée à l'environnement transactionnel propriétaire de Bull. Ainsi, les concepts, mécanismes, et principes étudiés dans cette partie se retrouvent également dans d'autres utilitaires de fonctionnalité équivalente à TDS. Il va de soi que la mise en oeuvre et l'appellation de ces derniers peuvent différer selon les systèmes offerts par d'autres constructeurs.

Quoi qu'il en soit, cette première partie de ce mémoire peut constituer une base de référence quant à la comparaison de l'environnement transactionnel ici présenté avec d'autres environnements transactionnels offerts sur le marché.

Une telle étude comparative pourrait constituer un prolongement de cette analyse transactionnelle. Par la suite, une analyse des possibilités de "portage" du code des applications transactionnelles de l'environnement transactionnel de Bull vers un autre environnement transactionnel pourrait être faite. Cette dernière proposition n'est pas sans intérêts informatiques et commerciaux.

La deuxième partie de ce mémoire était plus protocolaire. Son intérêt n'était cependant pas des moindres. L'existence de tels protocoles permet de poser des bases solides pour des coopérations entre applications.

Bull, dans un souci d'ouverture transactionnel, a très tôt adapté ses protocoles de communication entre applications aux protocoles existants et largement diffusés chez IBM pour ses produits IMS et CICS. A long terme, à travers XCP1 et XCP2, le but recherché par Bull est d'offrir des possibilités d'installation de produits Bull sur les sites transactionnels sur lesquels IBM est présent.

Le dernier chapitre de ce mémoire à proposé l'étude de l'état de la normalisation des protocoles de traitements transactionnels au niveau d'une des principales instances internationales de normalisation.

La position qu'a adopté l'ISO, lors de l'élaboration de son protocole TP 10026, reprend une partie des objectifs de Bull, lors de l'élaboration de son protocole XCP2. C'est ainsi que dans un souci d'intégration maximale, le protocole OSI 10026 a été inspiré du protocole LU6.2 d'IBM ayant largement fait ses preuves. Cette démarche permettra sans aucun doute d'insérer, en un labs de temps assez court, le protocole 10026 dans l'ensemble des protocoles de référence au niveau du traitement transactionnel.

Pour finir, je dirai que ce mémoire a été élaboré comme une approche du "monde transactionnel". La complexité des produits transactionnels, les interactions des éléments de l'environnement transactionnel avec l'Operating System ne portent pas à une étude détaillée de tous les aspects transactionnel. Le sujet serait trop vaste.
L'intéret de ce travail est avant tout d'appréhender de manière globale, en partant d'un cas concret, l'architecture, les mécanismes et les facilités de coopération qu'offre un utilitaire comme le moniteur transactionnel, dont les bases fonctionnelles tiennent de disciplines informatiques aussi variées que la gestion des bases de données, la gestion d'un Operating System ou la gestion de services de télécommunication.

Table des figures

Bibliographie

[APPC IBM 83] "An introduction to advanced program to program Communication (APPC) ". International System. IBM Australia 1983.

[BERN 87] Bernstein,Hadzilacos,Goodman. "Concurency Control and Recovery in Database System" Addison Westley, 1987.

[BULL AG 89] "TDS Administrator Guide" BULL DPS7. GCOS7 Production System 1988.

[BULL ORA 89] "Oracle sous TDS" Documentation interne BULL 1988.

[BULL PG 89] "TDS Programers Guide" BULL DPS7. GCOS7 Production System 1988.

[BULL PXCP 89] "Transactionnal intercommunication Using XCP1 Protocol - User's guide" BULL DPS7. GCOS7 Production System 1988.

[B ARCH 89] "TDS Architecture" Conférence-cours BULL TP 1989.

[B FICH 89] "Intégrité des fichiers" Conférence-cours BULL TP 1989.

[B GAC 89] "General Access Controler" Conférence-cours BULL TP 1989.

[B TDS 89] "Transactional Driven System" Conférence-cours BULL TP 1989.

[B TDSC 89] "Overview of TDS concepts" GCOS7 Production System 1989.

[B TDST 89] "TDS et télécommunication" Conférence-cours BULL TP 1989.

[DATE 82] Date C.J. "An introduction to Database Systems volume II" Chapitre 3. Addison Wesley 1982.

[GAC 84] "General Access Controler GAC" Cours CNI, Centre National d'Intervention logiciel GCOS7 Paris 1984.

[GRAY 83] Gray J.P., Hansen P.J., Homan P., Lerner M.A., Potsefsky M. "Advanced program-to-program communication in SNA" IBM System journal,vol 22 n°4, 1983.

[ISO TP1 90] "Information Technology - Open Systems Interconnection - Distributed Transaction Processing - Model" Draft International Standard ISO/IEC DIS 10026-1, ANSI 1990.

[ISO TP2 90] "Information Technology - Open Systems Interconnection - Distributed Transaction Processing - Service Definition" Draft International Standard ISO/IEC DIS 10026-2, ANSI 1990.

[ISO TP3 90] "Information Technology - Open Systems Interconnection - Distributed Transaction Processing - Protocol Specification" Draft International Standard ISO/IEC DIS 10026-3, ANSI 1990.

[LUG 86] Luguern J.P. " Protocole de Contrôle de cohérence dans un SGBD réparti" ERNST. Paris 1986.

[MEI 87] Meijer A., Peters P. "Computer Network Architectures" Pitman 1987.

[MUSS 89] Musseta M.E "Présentation de LU6.2" Cours Bull Paris 1989.

[PAPPC IBM 86] "An introduction to Programming for APPC/PC" IBM Technical Support Center. USA 1986.

[RAPA 85] Rapaport R. "Eléments pour spécification fonctionnelles de l'eXtended Coopérative Protocol ". Bull 1985.

[ROSL 78] Rosenkrantz D.J. , Stearns R.E., Lewis P.W. "System Level Concurency Distributed Database Systems" ACM Transactions on database System, V2, pp 178,198,Juin 1978.

[SCOT 81] Groupe de recherche SCOT " Détection et prévention des interblocages dans un système transactionnel centralisé". Rapport de recherche n°17, centre de recherche CII. Honeywell Bull, Grenobles avril 1981.