Extension de l'outil client-serveur ODIS-X en un outil transactionnel multipoints

Caroline Andrae, Andry Rakotonirainy

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

1992
25ème anniversaire

N° 1681

Programme 1
Architectures parallèles, Bases de données,
Réseaux et Systèmes distribués

EXTENSION DE L'OUTIL CLIENT-SERVEUR ODIS-X EN UN OUTIL TRANSACTIONNEL MULTIPOINTS

Caroline ANDRAE
Andry RAKOTONIRAINY

Mai 1992

RR_1681

7 mai 1992

Extension de l'outil client-serveur ODIS-X en un outil transactionnel multipoints

CEDIA. INRIA - Rocquencourt

Caroline ANDRAE

collaborateur externe de CEDIA

Universitaet Hamburg
Fachbereich Informatik
Arbeitsbereich Rechnerorganisation
Vogt-Koelln-Str. 30
2000 Hamburg 54

e-mail: andrae@rzsun1.informatik.uni-hamburg.de

Andry RAKOTONIRAINY

Université Pierre et Marie Curie
Laboratoire MASI
INRIA Rocquencourt
B.P. 105
F-78153 Le Chesnay CEDEX

e-mail: andry@verdi.inria.fr

RESUME

Il est incontestablement opportun de chercher à valider une architecture de système transactionnel avec l'utilisation entière du protocole OSI TP, puisqu'il a été voté en norme internationale le 11 Avril 1992. Nous présentons dans ce document une nouvelle architecture qui intègre la norme OSI TP dans l'outil ODIS-X afin de fournir aux applications la sémantique de transaction atomique et durable. Notre architecture est une extension du modèle client-serveur en un modèle client-(N)serveurs en prenant en compte les travaux en cours sur le moniteur transactionnel de X/Open. De nouveaux services ont été rajoutés à X/Open afin de bénéficier pleinement des facilités de OSI TP et d'offrir le maximum de flexibilité au développeur d'application. Notre architecture est entièrement compatible avec les interfaces actuelles de X/Open.

Mots clés: OSI TP, ODIS-X, X/Open, transaction, sérialisation.

ABSTRACT

In order for transaction processing to be useful for general distributed programming we presents a new efficient architecture which combines the standard OSI TP and ODIS-X. This architecture gives a transactional semantic for applications to maintain atomicity and durability on distributed database systems. We extend the client/server model into client-(N)servers including X/Open concepts. We add new services in X/Open model to allow a full use of OSI TP facilities. A special care was given to allow our architecture to provide a very flexible programming mechanism and compatibility with X/Open interfaces to developpers.

Keywords: OSI TP, ODIS-X, X/Open, transaction, serializability.

Zusammenfassung

Die Konzeption eines verteilten Transaktionssystems unter Einbeziehung der von der OSI-Transaktionsverarbeitung (OSI TP) gebotenen Möglichkeiten ist eine naheliegende Aufgabe, die durch die Anerkennung von OSI TP als internationaler Standard im April 1992 noch an Bedeutung gewonnen hat.

Wir stellen in diesem Dokument eine neue Architektur vor, die OSI TP in das bestehende Werkzeug ODIS-X integriert. Dadurch soll den auf ODIS-X basierenden Anwendungen, die heute mittels RPC auf entfernte Datenbestände zugreifen, durch die Nutzung der Transaktionsverarbeitung zusätzliche Sicherheit und Flexibilität geboten werden. Unsere Architektur stellt eine Erweiterung des Client-Server-Modells in ein Client-(N)-Server-Modell dar, wobeidie aktuell von X/Open für die Transaktionsverarbeitung vorgeschlagene Systemarchitektur einbezogen wurde. Die von X/Open vorgeschlagenen Schnittstellen wurden um neue Dienste ergänzt, um alle MÖglichkeiten von OSI TP nutzen zu kÖnnen und zugleich den Anwendungsentwicklern ein Maximum anFlexibilität zu gewährleisten.

Stichworte: OSI TP, ODIS-X, X/Open, Transaktion, Serialisierbarkeit.

Sommaire

Nous tenons à exprimer notre reconnaissance aux auteurs de ODIS-X, A. CARISTAN et P. LEPUIL (INFOSERV-INRIA) pour leur esprit de coopération, S.SEDILLOT (CEDIA-INRIA), expert OSI TP à l'AFNOR et l'ISO, P.THOMAS (INRIA-BNP) et C.BROU pour les discussions, critiques et relectures de nos travaux.

1 Introduction

Le but de notre projet est l'integration de la norme OSI TP dans l'outil ODIS-X, actuellement utilisé en support d'applications de gestion pour les services administratifs de l'INRIA, ceci afin de fournir aux applications la sémantique de transaction atomique et durable. L'architecture obtenue est une extension du modèle client-serveur en un modèle client-(N)serveurs. Cette architecture utilise et étend les interfaces actuellement en cours d'élaboration dans le groupe 'transactionnel' de X/Open. La richesse des fonctionnalités OSI TP en matière de gestion de dialogues est offerte à l'application, au prix de modifications des interfaces X/Open. L'utilisation des gestionnaires de données est optimisé, on offre à l'application la possibilité de configurer dynamiquement l'arbre de transaction. La cohérence des transactions locales est prise en charge.

Afin de conserver l'architecture initiale d'ODIS-X, on lève la contrainte imposée par X/Open, consistant à exécuter application, Resource Manager (équivalent de gestionnaire de données) et Transaction Manager (gestionnaire de transaction) dans le même processus. Pour accéder le réseau, la machine protocolaire OSI TP (TPPM) utilise les sockets et TCP/IP. On assure donc la portabilité de ODIS-X sur les réseaux interconnectant les unités de Recherche de l'INRIA.

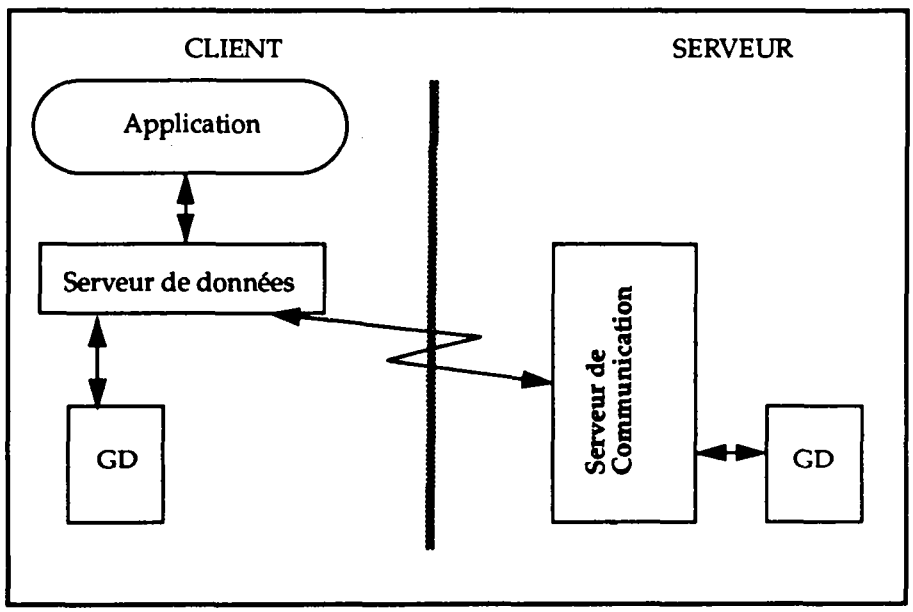
Ce travail a été accompli dans le cadre des activités de normalisation, en particulier relatives OSI TP, conduites à l'INRIA par Simone Sedillot.

Les trois premières parties présentent le produit ODIS-X, la norme OSI TP et le travail en cours à X/Open sur lesquels nous avons fondé notre architecture. Dans les trois dernières parties, nous présentons l'architecture ODIS-EX (Extented) en y introduisant de nouveaux formalismes tels les transactions partielles et globales et nous argumentons la solution retenue pour régler le compromis entre parallélisme et serialisation du traitement des événements.

2 ODIS-X

L'environnement ODIS-X offre un certain nombre d'outils afin de concevoir des applications intéractives, réparties ou non, sur des stations de travail graphiques UNIX. ODIS-X est utilisé à l'INRIA pour développer des applications de gestion spécifique de l'administration de l'INRIA. ODIS-X fournit entre autres aux applications une interface client-serveur et l'accès aux données, locales ou distantes. Les communications sont basées sur des appels synchrones RPC (Remote Procedure Call) qui sont exécutés sequentiellement. Une application peut accéder à des modules spécifiques, appelés gestionnaires de données (GD), locaux ou distants. La localisation des données est complètement transparente pour l'application. Le routage des appels est pris en charge par le serveur de données (SD).

FIGURE 1. Architecture de ODIS-X



3 OSI TP

OSI TP est une norme de la couche application dans l'architecture OSI (Open Systems Interconnection) de l'ISO (International Standard Organisation). OSI TP offre des services multi-points et a pour objectif de spécifier des règles d'interoperabilité pour exécuter des transactions distribuées entre différents systèmes ouverts, en garantissant l'atomicité et la durabilité de la terminaison d'une transaction répartie. Une transaction est une séquence d'opérations qui font passer le système d'informations réparties d'un état cohérent initial à un état cohérent final. Comme tous les normes de l'architecture OSI, OSI TP ne définit que les procédures de communication entre les systèmes.

Les procédures multipoints d'OSI TP sont une spécification de la validation à deux phases telles que présentées et étudiées largement dans les années 80. En outre, OSI TP offre des services point à point établissement/fermeture/abandon de dialogue, que nous utiliserons.

OSI TP ne pose aucune contrainte sur l'architecture locale d'un système transactionnel.

4 X/Open

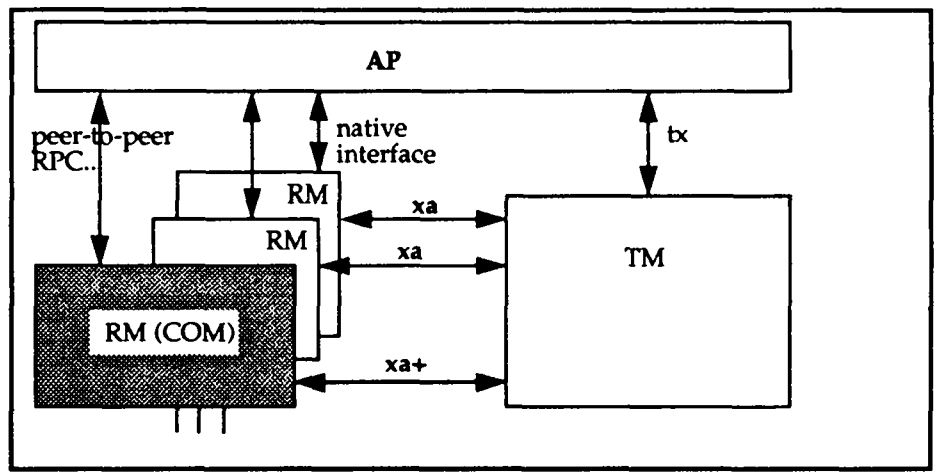
X/Open est un consortium indépendant, structuré en groupe de travail comprenant des utilisateurs (AT&T, ICL, OSF, Unix System Laboratories...) et des constructeurs (IBM, ICL, NCR, Sun Microsystems). X/Open a pour but de spécifier des standards UNIX, en prenant en compte les besoins du marché, afin d'améliorer la portabilité des applications et l'interopérabilité des produits. Le modèle X/Open Distributed Transaction Processing (DTP) permet à plusieurs applications (AP) de gérer plusieurs types de ressources encapsulées dans des 'Resource Managers' (RM). X/Open DTP spécifie les mécanismes qui coordonnent APs et RMs dans une transaction globale. Cette coordination est assurée par un Transaction Manager (TM). Des interfaces standards sont définis entre AP, TM et RM pour répondre aux besoins de portabilité, interoperabilité et modularité.

Le modèle X/Open DTP identifie les fonctionnalités des trois types de modules (guide book 91):

- Le module application (AP) définit et délimite la transaction et spécifie les actions qui constituent une transaction,
- Le module Resource Manager (RM) fournit les accès aux ressources partagées,
- Le module Transaction Manager (TM) donne un identificateur unique à chaque transaction, gère et garantit l'atomicité de chaque transaction.

La Figure 2 montre le modèle X/Open DTP.

FIGURE 2. Le modèle X/Open DTP



Dans le cadre d'une transaction locale, une application, est associée avec un TM et un ou plusieurs RMs. Elle communique avec chaque RM par l'interface native de ce RM. Cela implique que X/Open ne pose aucune restriction sur l'interface entre AP et RM. L'AP communique avec le TM par l'interface tx, qui actuellement n'est guère défini par X/Open. L'interface entre le TM et un RM est standardisée sous le nom 'interface xa'; la relation entre TM et un RM représente

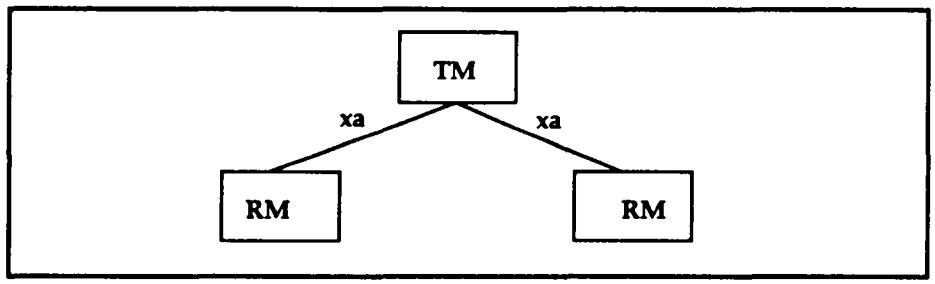
une branche de transaction. Les RMs sont, par exemple, des gestionnaires de données. A terme, X/Open prévoit que les SGBD du marché (ORACLE, INGRES...) supportent l'interface xa.

Actuellement, X/Open ne définit que des interfaces pour la gestion d'une transaction dite globale, en ce qu'elle fait appel à plusieurs RMs coordonnés, mais que nous qualifions de locale, puis qu'elle ne supporte pas des communications normalisés entre RMs et TMs répartis. Le document X/Open sur lequel on s'est appuyé date de Septembre 91. Notre modèle constitue une anticipation du standard DTP.

L'extension pour gérer les transactions réparties en utilisant OSI TP à travers l'interface xa+ est en cours de définition. Dans le cadre d'une transaction répartie, le TM se servira d'un RM spécifique, appelé RM-COM, pour la communication avec les TMs distants. L'interface entre TM et RM-COM est appelée xa+, car elle ne contient pas seulement des services xa, mais aussi les services nécessaires pour la gestion des TMs distants. Le RM-COM contient la machine protocolaire OSI TP (TPPM). L'application pourra utiliser les services du RM-COM pour communiquer en mode point à point (peer-to-peer) sur des dialogues OSI TP avec des applications distantes.

Selon X/Open, un arbre de transaction est construit comme dans la Figure 3.

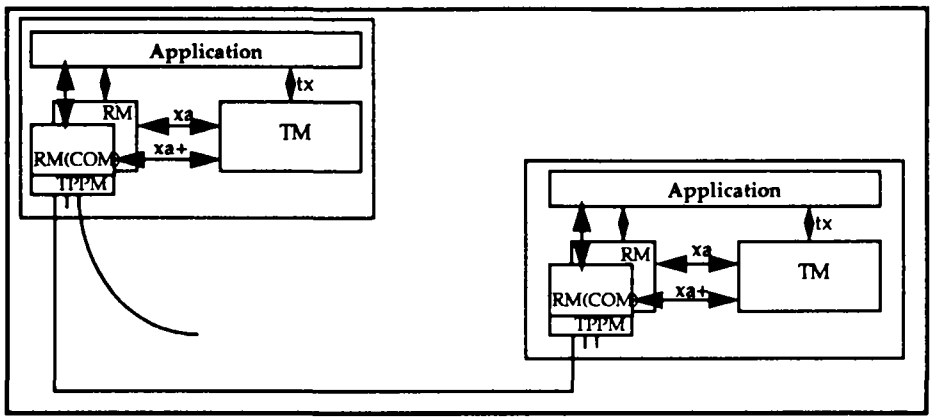
FIGURE 3. Arbre de transaction locale



Pour supporter des transactions réparties basées sur OSI TP, on interconnecte deux noeuds suivant la Figure 4.

On utilise ici la notion de noeud dans le sens d'OSI TP. Dans un arbre de dialogues (aussi appelé arbre transactionnel), un noeud est composé d'une application avec un ou plusieurs RMs, un TM et un module RM-COM. Dans le même site, une station de travail UNIX, par exemple, plusieurs noeuds peuvent coexister.

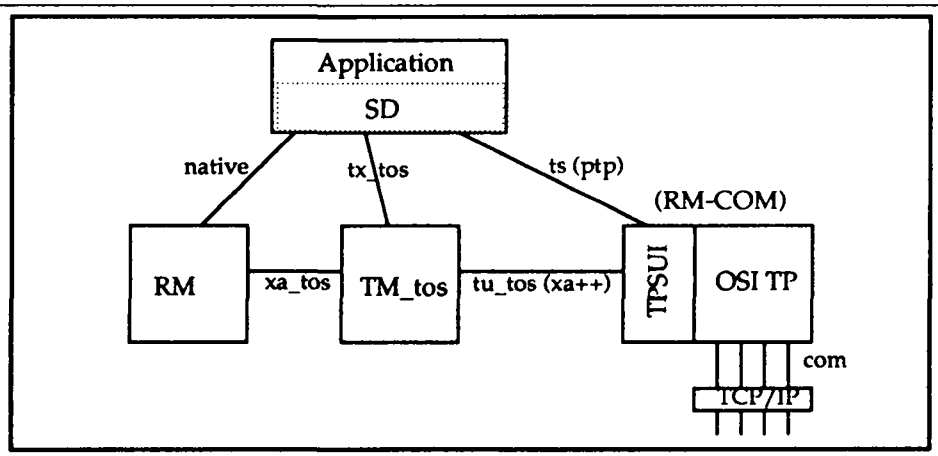
FIGURE 4. Arbre de transaction répartie



5 L'architecture ODIS-EX

L'architecture ODIS-EX, avec les extensions proposées ci-dessous, s'appuie sur le modèle X/Open, et est illustrée sur la Figure 5. Le serveur de données (SD) de ODIS-X a été inclus dans l'application, car selon le modèle X/Open, les fonctions réalisées par le SD font partie de l'application. Le Transaction Manager (TM_tos) gère les RMs locaux impliqués dans une transaction répartie. Le TM_tos utilise l'interface utilisateur défini par la norme OSI TP (représenté par le module TPSUI) pour accéder aux noeuds distants de la transaction. Pour le TM_tos, tout l'arbre réparti de la transaction est représenté par le module TPSUI.

FIGURE 5. Architecture choisie



5.1 Les modules

Les différents modules montrés dans Fig. 5 remplissent les fonctions suivantes:

Serveur de données (SD)

Le serveur de données assure la transparence de la localisation des données. Il route les requêtes vers le RM concerné, local ou distant.

Gestionnaire de données (GD) ou RM

Le gestionnaire de données pourra être un des GDs existants sous ODIS-X. Il sera cependant nécessaire que le GD synchronise ses actions avec le TM_tos en utilisant l'interface xa.

TM_tos

Le TM_tos représente le Transaction Manager du modèle X/Open. Ses fonctions comporte la génération d'un identificateur unique pour la transaction globale et la gestion locale de la transaction, avec les RMs locaux. On a personnalisé ce module en l'appelant TM_tos parce que, par rapport au TM de X/Open, on a ajouté des fonctions afin d'utiliser OSI TP. TM_tos traite le TPSUI (et avec lui l'arbre de transaction répartie) comme un RM local. TM_tos n'a pas d'échange de type point à point sur les branches de transaction répartie. La topologie de la distribution lui est transparent, cette dernière étant prise en compte par le (seul) TPSUI. TM_tos synchronise l'application (interface tx), les RMs locaux (interface xa_tos) et le TPSUI (interface tu_tos).

TPSUI

Le module TPSUI représente l'interface utilisateur défini par la norme OSI TP. Il est situé entre le TM_tos et la machine protocolaire OSI TP. Le TPSUI utilise les services OSI TP. Il offre l'interface peer-to-peer à l'application et gère toutes les dialogues et branches de transaction répartie ouverts par l'application.

TPPM

Le module TPPM représente la machine protocolaire OSI TP.

TCP

Toutes les communications vers le réseau passent par le module de communication TCP/IP. Le TPPM utilise les services de TCP au lieu de ceux de la couche Présentation (ISO), pour véhiculer les APDUs qu'il crée.

5.2 Les interfaces

Les interfaces et services entre les modules sont identifiés dans les tableaux 1 et 2.

TABLEAU 1. Identification des interfaces

Nom de l'interface	fonction
SD / AP	interface native (interne) (interface ODIS-X)
TM_tos / AP	interface tx_tos (extension de l'interface tx de X/Open)
RM / SD	interface native (interface ODIS-X)

TM_tos / RM	interface xa_tos (extension de l'interface xa de X/Open
TPSUI / TM_tos	interface tu_tos (préfiguration de xa+ de X/Open))
SD / TPSUI	interface ts_tos (peer-to-peer de X/Open)
TPPM / TPSUI	services OSI TP
TPPM / TCP	interface com (envoi des PDUs par TCP/IP)

- ts_tos qui permet à l'Application d'accéder directement aux services du TPSUI; les services ts_tos offrent au SD des services bi-point (orientés dialogue) de OSI TP. Cette interface permet, entre autre, de préparer une branche de transaction, contrairement à l'interface tx qui est une interface à service globale et correspond donc, entre autre, aux services globaux de OSI TP.
- tu_tos qui permet au TM_tos d'accéder au TPSUI. Cette interface offre les services globaux de gestion de transaction avec le noeuds adjacents (supérieur éventuel et subordonnées de l'arbre transactionnel).

Nous avons ajouté des extensions sur les interfaces existantes tx et xa de X/Open. Le détail de ces extensions est fourni par la suite, mais les motivations générales en sont les suivantes:

- offrir à l'application ce qu'offre OSI TP: Gérer simultanément des branches de transactions et des activités sur RMs hors transaction,
- pour une même application, permettre à un RM qui a travaillé hors transaction d'entrer dans une transaction globale,
- offrir à l'application une autre fonctionnalité de OSI TP: La préparation à la validation d'une branche subordonnée et de son sous-arbre,
- autoriser des RMs et APs distants à être subordonnés au sens OSI TP, c'est à dire serveurs,
- permettre à un TM subordonné d'indiquer et de négocier le début et la fin d'une transaction avec son supérieur

Pour nommer les services, on garde la convention de X/Open pour indiquer émetteur et récepteur du chaque service, c'est à dire

- les services de l'AP vers le TM sont nommés tx,
- les services du TM vers l'AP sont nommés xt,
- les services du TM vers un RM sont nommés xa,
- les services d'un RM vers le TM sont nommés ax.

Les services supplémentaires sont:

- xa_change_req: permet à TM_tos d'indiquer au RM de s'enregistrer dans une transaction globale.
- ax_done_req: permet au RM de dire à TM_tos qu'il a terminé un commit (validation) ou un rollback (abandon).
- ax_ready_req: permet au RM de dire à TM_tos qu'il est dans l'état ready (prêt à valider).
- xt_begin_ind: permet à TM_tos d'indiquer à l'application qu'il veut déclencher le début d'une transaction globale.
- xt_commit_ind: permet à TM_tos d'indiquer à l'application que la transaction en cours est validée.

- xt_prepare_ind - permet à TM_tos de prévenir l'AP que le TM veut exécuter la première phase de la validation.
- xt_complete_ind - permet à TM_tos d'indiquer la terminaison de la transaction en cours.
- xt_rollback_ind - permet à TM_tos d'indiquer à l'application que la transaction sera annulée.
- xt_error_ind - permet à TM_tos d'indiquer à l'application une erreur dans un service tx.
- tx_ready_req - permet à l'application de dire à TM_tos qu'elle est prête à valider.
- tx_begin_rsp - permet à l'application de confirmer le début d'une transaction.

TABLEAU 2. Répartition des services par interfaces

Nom	module émetteur	services
ts (SD / TPSUI)	TPSUI:	ts_open_dial_ind, ts_close_dial_ind, ts_begin_trans_ind, ts_ready_ind ts_abort_ind, ts_error_ind, ts_data_ind
	SD:	st_open_dial_req, st_close_dial_req, st_begin_trans_req, st_data_req st_receive_req, st_prepare_req
native (AP / SD)	AP:	sd_connect_req, sd_deconnect_req, sd_lire, sd_ecrire, etc.
	SD:	résultats
tx_tos (AP / TM)	AP:	tx_open_req, tx_begin_req/rsp*, tx_commit_req, tx_rollback_req, tx_rollback_rsp* tx_receive_req*[1],tx_ready_req*
	TM_tos:	xt_rollback_ind, xt_commit_ind, xt_complete_ind*, xt_error_ind* xt_prepare_ind*,xt_begin_ind*
native (SD / RM)	SD:	lire, ecrire, etc.
	RM:	résultats
xa_tos (TM_tos / RM (GD))	TM_tos:	xa_change_req*, xa_prepare_req*, xa_commit_req, xa_rollback_req, xa_complete_req
	RM (GD):	ax_reg_req, ax_ready_req*, ax_rollback_req, ax_done_req*
tu_tos (TM_tos / TPSUI)	TM_tos:	tu_change_req, tu_prepare_req, tu_commit_req, tu_rollback_req, tu_done_req, tu_complete_req, tu_receive_req
	TPSUI:	ut_reg_req, ut_prepare_ind, ut_commit_ind, ut_rollback_ind
tp (TPPM / TPSUI)	TPPM:	OSI TP service ind et cnf
	TPSUI:	OSI TP service req et rsp
com (TPPM / TCP)	TPPM:	connect_req, deconnect_req send_req
	TCP:	receive_ind

Notes:

Les sigles req (requête), ind (indication), rsp (réponse) et cnf (confirmation) sont utilisés avec la sémantique définie dans le modèle de référence ISO 7498.

Le service tx_ready_req est utilisé dans un noeud subordonné pour confirmer l'accord de l'application avec la validation de la transaction en cours. La sémantique de ce service correspond

1. Les services avec un * ont été rajoutés aux services tx ou ax de X/Open

au TP-COMMIT req de OSI TP. On n'utilise pas le tx_commit req pour ce besoin parce qu'il se peut que l'usage de ce dernier soit réservé par X/Open à un noeud racine.

Les services xa_complete_req et tu_complete_req suivent la définition du modèle X//Open. Ils servent à obtenir le résultat d'une opération asynchrone. Ces services n'ont pas de rapport avec les services TP-COMMIT-COMPLETE req/TP-ROLLBACK-COMPLETE req de OSI TP. Ces derniers sont reportés à l'application dans xt_complete ind.

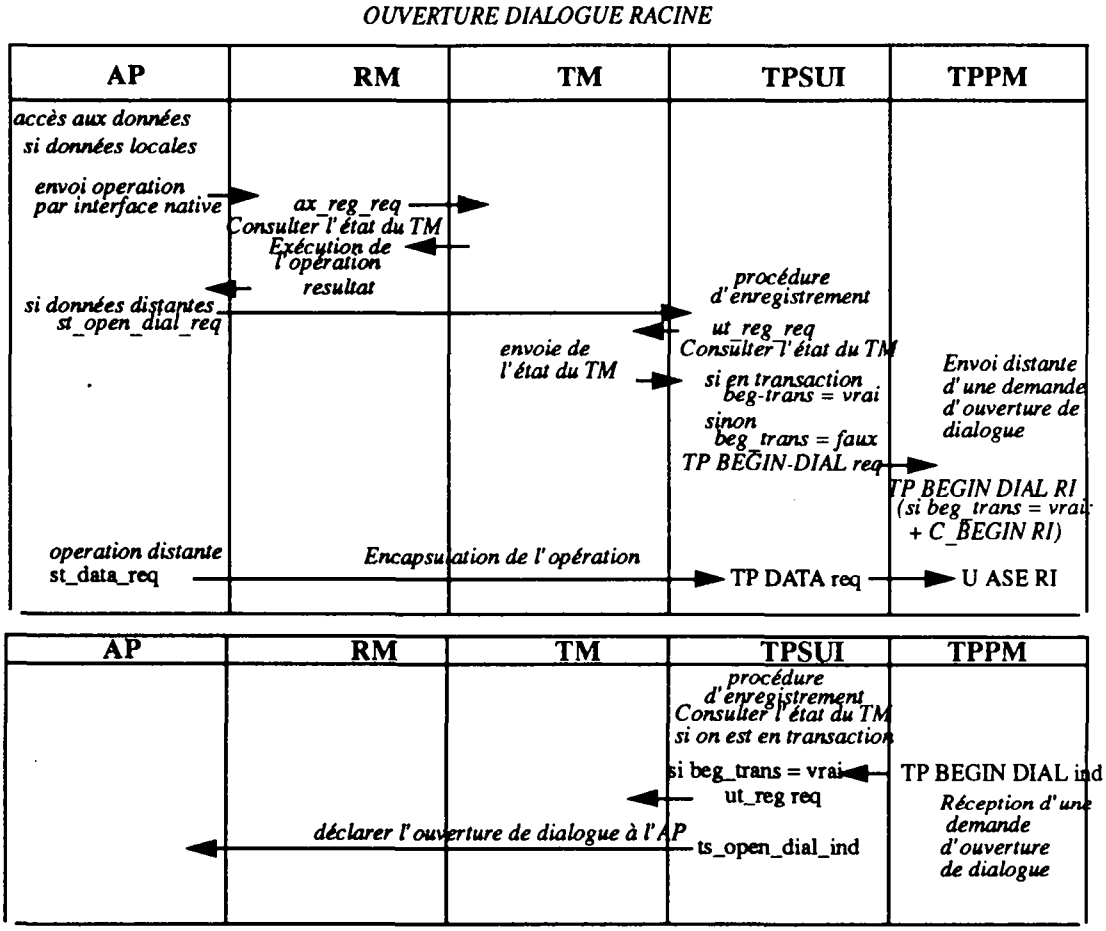
TABLEAU 3. Services OSI TP utilisés par le TPSUI

TP-BEGIN-DIALOGUE req/ind
TP-END-DIALOGUE req/ind
TP-PREPARE req
TP-COMMIT-req/ind
TP-ROLLBACK req/ind
TP-DONE-req
TP-BEGIN-TRANSACTION req/ind
TP-READY ind
TP-COMMIT-COMPLETE ind
TP-ROLLBACK-COMPLETE ind
TP-U-ABORT req/ind

5.3 Services utilisées pour les accès aux données

Les Figures 6 et 7 montre le déroulement d'un accès aux données distantes ou locales. Si les données sont sur un site distant alors on doit ouvrir un dialogue au sens OSI TP (Figure 6); ensuite l'opération est envoyé vers le site concerné (Figure 7).

FIGURE 6. Demande d'accès aux données locales ou distantes

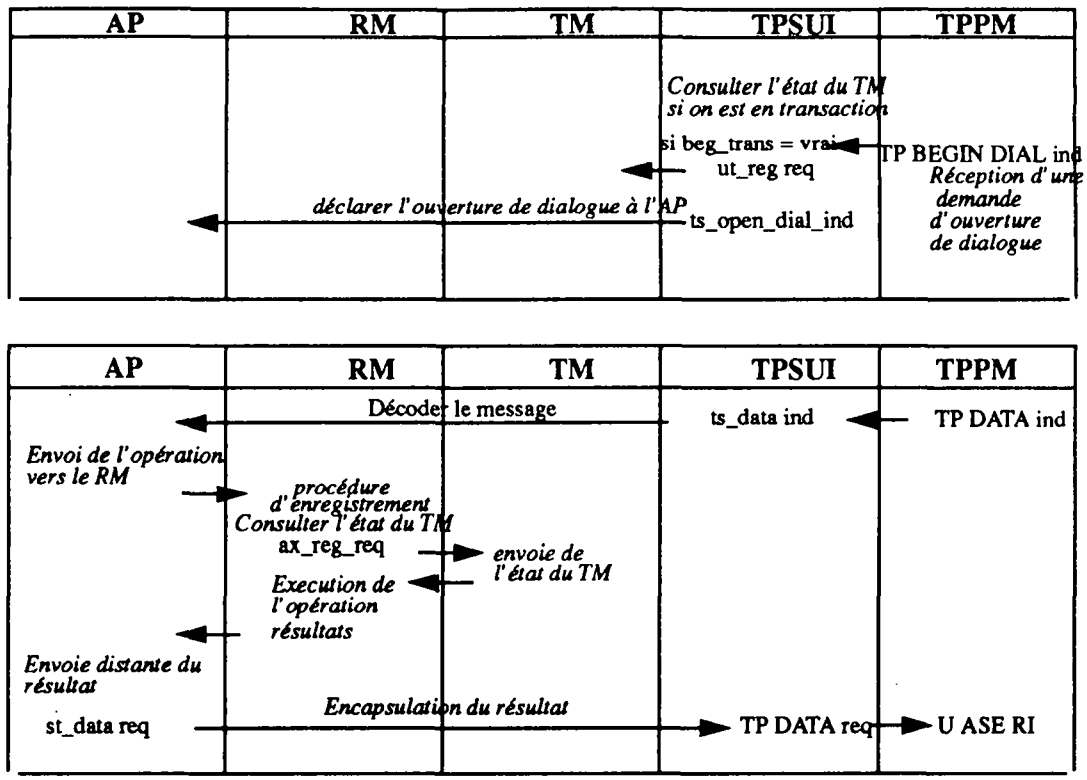


Note:

Le paramètre beg_branch dans ut_reg_req sera expliqué dans la section 6. Le paramètre beg-trans est défini dans le service TP-BEGIN-DIALOGUE req/ind de OSI TP.

On peut anticiper l'envoi de l'opération avec la demande d'ouverture de dialogue.

FIGURE 7. Réception de demande d'accès distante



5.4 Services utilisés pour la validation.

Les principes de déroulement de la validation, s'appuient sur le comportement de la TPPM. La validation en deux phases comprend.

- une première phase avec l'envoi des messages prepare vers tous les subordonnés, suivi de l'attente de tous les réponses ready (ou rollback)
- une deuxième phase avec l'envoi de l'ordre de commit (ou rollback) suivi de l'attente de toutes les confirmations.

Le TM_tos doit coordonner les RMs locaux et l'arbre de transaction géré par OSI TP.

.La Figure 8 illustre le scénario correspondant

- Le TM commence donc la validation avec l'envoi de l'ordre de préparation du sous-arbre reparti au TPSUI
- Le TPSUI genère un TP-PREPARE req pour chaque branche de transaction.
- le TM commande la préparation de tous les RMs locaux.
- Il attend tout les réponses des RMs locaux;
- les réponses des subordonnés sont mémorisés dans la TPPM.

- Tous les réponses positives des RMs locaux obtenu (le noeud local est donc prêt à valider), le TM envoie le tu_commit_req, qui déclenche le TP-COMMIT req vers la TPPM.
- Lorsque TM_tos a reçu à la fois le dernier ready et le TP-COMMIT req, le TPPM envoi TP-COMMIT ind au TPSUI, qui envoie l'ordre de validation commit sur les branches de transaction ou l'ordre d'annulation, si au moins une des réponses aux messages prepare est une demande d'annulation, c'est à dire un refus.
- Le TPSUI recevant TP-COMMIT ind envoie ut_commit ind à TM_tos
- TM_tos, qui s'était mis en attente du message du TPSUI, peut à ce moment-là envoyer le xa_commit_req aux RMs locaux.
- C'est donc la TPPM la dernière instance à prendre la décision pour commit ou rollback, dépendant de l'état du sous-arbre de transaction. Les messages TP-READY ind ne sont pas reportés àTM_tos. Ce scénario utilise pleinement et ne duplique pas dans TM_tos la fonction de coordination et de décision offerte par TPPM. Il permet de paralléliser les premières phases de validation sur RMs et dialogues. L'algorithme suivi par TM_tos racine est décrit dans la FIGURE 9. en pseudo-code.

FIGURE 8. Validation

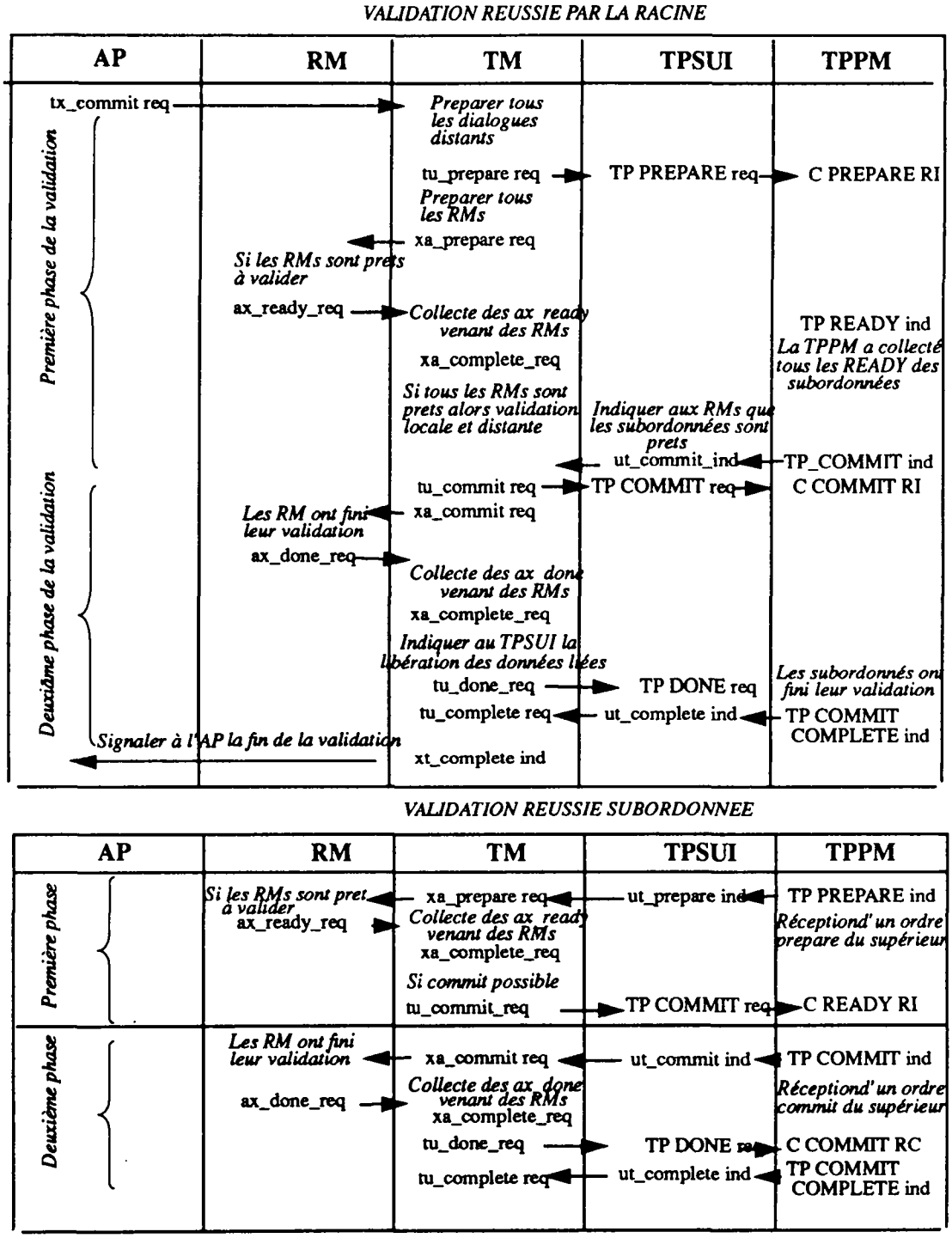


FIGURE 9. Algorithme du TM_tos racine sur réception du tx_commit req

```
DEBUT tx_commit req
cpt_done = 0; cpt_ready = 0;
tu_prepare_req();

Pour tous RMs locaux
 {
  xa_prepare_req()
 }
Tant que cpt_ready < ΣRM
   {
   attendre message ax;
   Si ax_ready_req alors cpt_ready++;
   Si ax_rollback_req alors --> Procedure Rollback du TM
}

tu_commit_req();
attendre message ut;
Si ut_commit_ind alors
   {
    Pour tous RMs locaux faire
       {
       xa_commit();
       Tant que cpt_done < ΣRM
          {
          attendre message ax;
          Si ax_done_req alors cpt_done++;
          Si timeout alors Procedure de Recovery locale;
          }
   }
}
sinon ut_rollback_ind ---> Procedure Rollback du TM

Si cpt_done = ΣRM alors
    tu_done_req;
attendre message ut;
Si ut_complete_ind alors
      xt_complete_ind(commit);

FIN tx_commit
```

6 Traitement des transactions partielles et globales

6.1 Extension du modèle X/Open

6.1.1 Utilisation d'OSI TP

Dans notre architecture, les modules TM_tos des différents noeuds communiquent en utilisant OSI TP. Le module contenant le TPSUI et la machine protocolaire OSI TP forme un RM-Multicom (RM-MC) dans le modèle X/Open. L'interface tu_tos entre TM_tos et RM-MC correspond a l'interface xa+ du X/Open, avec quelques extensions. Notre but sera de traiter les RMs et le RM-MC (qui à son tour représente les dialogues) de façon aussi symétrique que possible, par rapport au module TM_tos.

6.1.2 Les modèles de transaction

Avant d' analyser le démarrage des différents types de transaction dans les différents cas, on va préciser les notions d'une transaction locale, partielle et globale.

Transaction globale

Dans le modèle X/Open, une application peut démarrer une transaction globale utilisant le tx_begin_req. Après un tx_begin_req, tous les dialogues existants et tous les RMs passent en mode transactionnel. Tous les RMs et tous les dialogues seront inclus dans le processus de validation/abandon de transaction. Les transactions globales donnent la forme la plus générale du traitement transactionnel.

Transaction native

La notion d'une transaction native est un concept déduit du modèle X/Open. Par l'accès aux services natives des RMs, sans avoir fait tx_begin_req, l'application travaille hors d'une transaction globale. Nous allons distinguer deux cas différents:

- Nous appelons ce mode de travail une transaction native, si l'application utilise le service 'début de transaction' de l'interface native.
- Sinon, il s'agit d'un travail hors transaction.

Le TM de X/Open n'assure aucune coordination de validation/abandon de transaction entre RM et AP, avant qu'un tx_begin_req n'ait été reçu. De plus, le TM refuse un tx_begin_req si l'un au moins des RMs est utilisé hors transaction globale. De ce fait, transactions globales et transactions natives sont exclusives dans X/Open.

Ceci n'exploite pas toutes les facilités qu'offre OSI TP: Donner à l'application, via l'interface ts, qui représente l'interface peer-to-peer de X/Open, les services orientés dialogue de OSI TP, lui permet de changer un dialogue en mode 'none' à 'commitment', c'est à dire de travailler avec un RM distant, d'abord en mode hors transaction, puis en mode transaction.

Transaction partielle

Au niveau dialogue (dans l'interface peer-to-peer), l'application peut utiliser tous les services OSI TP qui sont orientés dialogue. Ceci doit permettre à l'application d'établir pour la même transaction certains dialogues au niveau 'commitment' et laisser d' autres dialogues au niveau 'none'. Seuls les dialogues de niveau 'commitment' seront inclus dans le mécanique de validation/abandon de transaction. Cette nouvelle possibilité nous conduit à introduire dans le modèle X/Open la notion de 'transaction partielle' et à examiner différents cas de coopération entre RMs, TM et AP, au niveau de la racine comme au niveau d'un subordonné.

6.2 La gestion des transactions

Dans une 'transaction partielle', les différentes branches locales et distantes sont ouvertes explicitement par l'application. Ce qu'on appelle une branche de transaction sera

- dans le cas locale, un RM travaillant en mode transaction par l'interface native,
- dans le cas distant, un dialogue au niveau 'commitment' géré par l'interface ts.

On suppose qu' à un instant donné, toutes les branches de transaction ouvertes par l'application font partie de la même transaction partielle. Contrairement à une transaction globale, une transaction partielle contient seulement des branches ouvertes explicitement par l'AP.

On suppose que toutes les transactions partielles (comme les transactions globales) seront vues et gérées par le TM. Ceci implique que le TM génère l'identificateur unique de la transaction; il est le seul à donner l'ordre commit ou rollback de la transaction. On exige donc, qu'un RM recevant sur son interface native (SQL, par exemple) un service 'début de transaction', s'adresse au TM pour obtenir cet identificateur globale. Ensuite, le RM reçoit les commandes de validation/abandon de transaction de la part du TM. L'application utilise les services du TM pour valider ou annuler la transaction. Ceci permet de garantir l'atomicité d'une transaction locale qui utilise plusieurs RMs.

Le choix entre une transaction globale et une transaction partielle est une décision locale à l'application. Dans un noeud subordonné, la transaction sera initialisée comme partielle, sur réception de demande d'établissement de dialogue venant du noeud supérieur. L'application subordonnée a, ensuite, la possibilité de changer l'état de la transaction en global.

Pour des applications qui ne souhaitent pas gérer leurs branches de transaction (ou qui ne possèdent que des RMs sans service 'début transaction' dans l'interface native), on va définir une unité fonctionnelle, qui contient seulement des transactions globales. Cette unité fonctionnelle choisie, le noeud subordonné entrera tout de suite dans l'état 'transaction globale' quand il sera inclus dans la transaction du supérieur. Dans ce cas, le démarrage d'une transaction partielle ne sera pas possible. La seule façon de démarrer une transaction sera une ouverture explicite d'une transaction globale.

A un moment donné, un TM_tos sera donc, par définition, dans un des quatre états suivants:

- *'hors transaction'*: l'application n'a pas encore travaillé avec des RMs locaux ou distants ou elle a travaillé sans démarrer une transaction

- *'transaction partielle'*: l'application a démarré une transaction partielle via l'interface native d'un RM ou via l'interface ts du TPSUI. Seuls les RMs et les dialogues qui sont explicitement déclarés comme tel par l'application font partie de la transaction
- *'transaction globale'*: l'application a démarré une transaction globale utilisant le service tx_begin_req. Tous les RMs utilisés et tous les dialogues existants font partie de la transaction globale
- *'terminaison de transaction'*: l'application a commandé la validation d'une transaction partielle ou globale.

Pour notre modèle, on a choisi de considérer tous les RMs comme des RMs dynamiques dans le sens de X/Open. Ceci implique, qu'aucun RM n'est inclus dans une transaction par le service xa_start_req. Les RMs dynamiques s'adressent eux-mêmes à TM_tos quand ils commencent à travailler pour une application. C'est ce que l'on appelle l'enregistrement. Ce comportement réduit au minimum le nombre de RMs qui sont gérés dans la transaction.

Afin de réaliser ces objectifs, on introduit deux paramètres supplémentaires dans les services ut_reg_req et ax_reg_req. Les paramètres ajoutés sont:

- boolean beg_branch;
  Le paramètre beg_branch est ajouté dans ax_reg_req et dans ut_reg_req. Si sa valeur est 'true', le paramètre indique le début d'une branche de transaction partielle avec le RM ou sur un dialogue, respectivement. Si sa valeur est 'false', l'application travaille hors transaction avec le RM ou sur un dialogue.
- INITIATOR_TYPE initiator;
  Ce paramètre est ajouté dans le service ut_reg_req. Il indique l'initiateur de la transaction a TM_tos. L'initiator a la valeur 'local' lorsqu'un ts_begin_trans_req est reçu de l'application locale. La valeur 'distant' indique qu'un TP_BEGIN_TRANSACTION_ind ou TP_BEGIN_DIALOGUE_ind (beg_branch = true) est reçu; dans ce cas, la transaction a été initialisée par une application distante, le TM devient donc subordonné dans la transaction.

6.3 Enregistrement dynamique des RMs et du TPSUI auprès de TM_tos

Le service ax_reg_req peut, selon la valeur des paramètres beg_branch et initiator, servir à

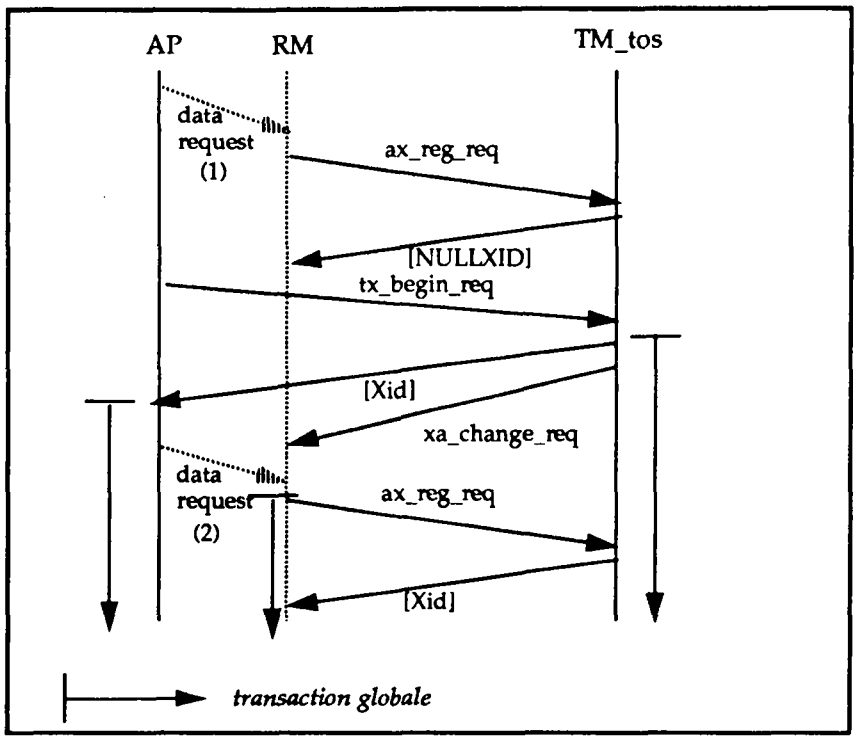
- informer le RM de l'état de TM_tos
- informer TM_tos que le RM commence à travailler pour son application
- informer TM_tos, si ce RM travaille pour une branche de transaction partielle.

Nous avons crée un nouveau service xa_change_req. Il est utilisé par le TM pour informer tous les RMs, qui ont travaillé hors transaction, du début d'une transaction globale. Afin de ne pas inclure inutilement des RMs dans la transaction globale, chaque RM n'entre dans la transaction que sur la première demande d'accès aux données reçue après le démarrage de la transaction globale. Un RM, qui n'est plus utilisé après le tx_begin_req ne participera donc pas aux phases de terminaison de la transaction. La réception du xa_change_req indique au RM, que son AP est maintenant en transaction globale et qu'il est obligé de se ré-enregistrer avec ax_reg_req avant sa prochaine accès aux données. Ce deuxième enregistrement lui servira à obtenir l'identificateur de la transaction globale et en même temps, à s'inclure dans les procédures de terminaison de transaction.

6.3.1 Principe de fonctionnement de xa_change req et tu_change req

L'utilisation du ax_reg_req et xa_change_req est illustré par le scénario 1.

SCENARIO 1. Enregistrement des RMs locaux dans une transaction globale



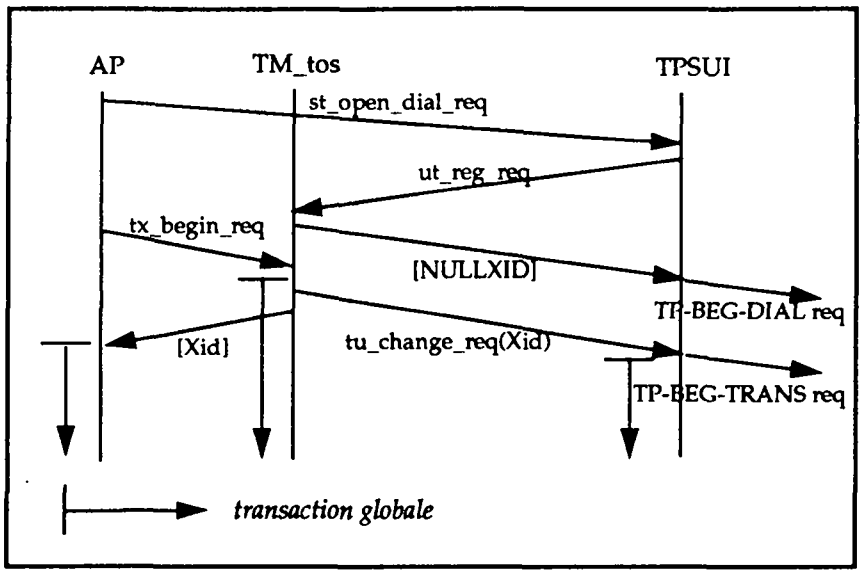
Le premier service natif (data_request 1)est hors transaction, TM_tos répond négatif au ax_reg_req. Sur réception du tx_begin_req, TM_tos envoie le xa_change_req au RM, parce que ce dernier fait ax_reg_req et est donc enregistré avec le TM_tos. Le RM répète le ax_reg_req avant qu'il ne traite le prochain service natif et reçoit, dans la réponse de TM_tos, l'identificateur de la transaction globale. Le TM prendra le RM en compte pour la phase de terminaison de transaction.

Pour le TPSUI, la situation est similaire (car l'interface tu ressemble à l'interface xa) aux différences suivantes près:

- le TPSUI fait ut_reg_req sur réception de la première requête de début de dialogue none, pour savoir si TM_tos est en transaction globale. Si une branche de transaction partielle est ouverte par l'application, chacun des TPSUIs dans les deux noeuds, supérieur et subordonné, font ut_reg_req pour informer leur TM_tos.
- le TPSUI reçoit le tu_change_req avec l'identificateur de transaction en paramètre si un tx_begin_req est reçu par le TM. Il n'est donc pas obligé de répéter le ut_reg_req pour demander le Xid. Le tu_change_req monte tous les dialogues du TPSUI au niveau 'commitment'. On n'attend pas le prochain envoi des données pour chaque dialogue avant qu'on ne le monte au niveau 'commitment', ceci parce que le service TP_BEGIN_TRANSACTION ind peut suffir à déclencher des actions (predéfinies) dans un noeud subordonné.

Le scénario 2) montre le principe de fonctionnement du ut_reg_req et tu_change_req.

SCENARIO 2. Enregistrement du TPSUI dans une transaction globale



Si le TPSUI reçoit le tu_change_req alors il est inclus tout de suite dans la transaction globale. Il envoie le TP-BEGIN-TRANSACTION req sur tous les dialogues ouverts.

Note: Une application, qui a ouvert des dialogues sur lesquelles elle ne veut pas travailler en mode transactionnel, doit fermer ces dialogues avant de faire tx_begin_req.

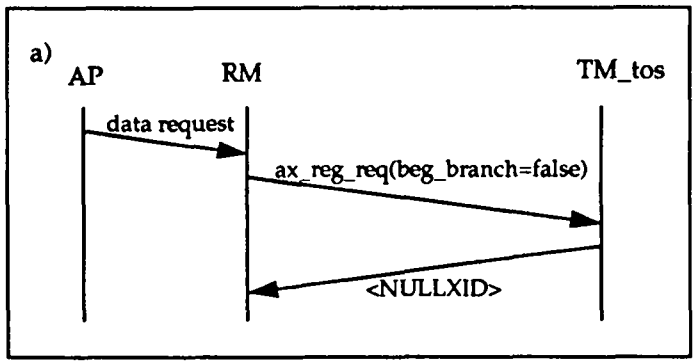
On va analyser dans les trois sous-sections suivantes, les différents scénarios d'enregistrement des RMs et de début de transaction. Les sous-sections sont groupés suivant l'état de TM_tos qui sont:

- hors transaction (sous-section 6.3.2)
- transaction partielle (sous-section 6.3.3)
- transaction globale (sous-section 6.3.4).

6.3.2 Enregistrement d'un RM ou TPSUI auprès de TM_tos dans l'état '*hors transaction*'

a) Enregistrement d'un travail local hors transaction

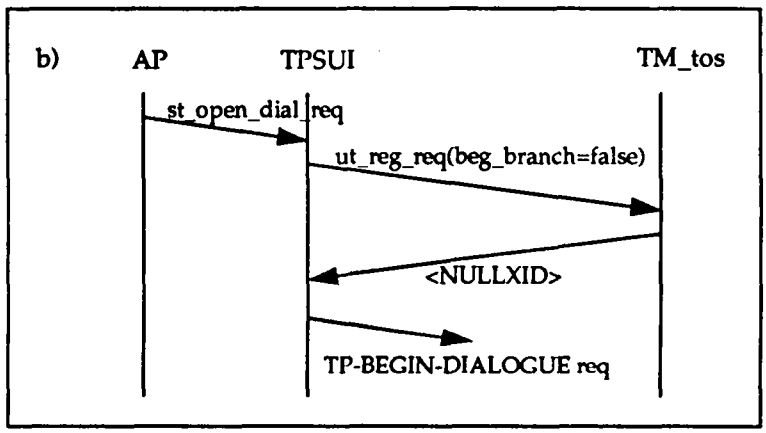
L'application commence à travailler avec un RM local sans démarrer une transaction locale. Avant son premier accès aux données, le RM envoie un ax_reg_req a TM_tos avec beg_branch = false. La réponse du TM est négative, parce qu'il n'est pas dans une transaction globale. Le TM mémorise, qu'il a reçu un ax_reg_req de ce RM.



b) Enregistrement d'un début de dialogue au niveau 'none'

L'application ouvre un dialogue au niveau 'none'. S'il s'agit du premier dialogue à ouvrir. Le TPSUI envoie a TM_tos un ut_reg_req avec beg_branch = false. La réponse du TM est négative, parce qu'il n'est pas dans une transaction globale. TM_tos mémorise, qu'il a reçu un ut_reg_req (beg_branch= false) du TPSUI. Le TPSUI ouvre le dialogue au niveau 'none'. Pour les prochains dialogues 'none', il n'y a pas d'enregistrement du TPSUI.

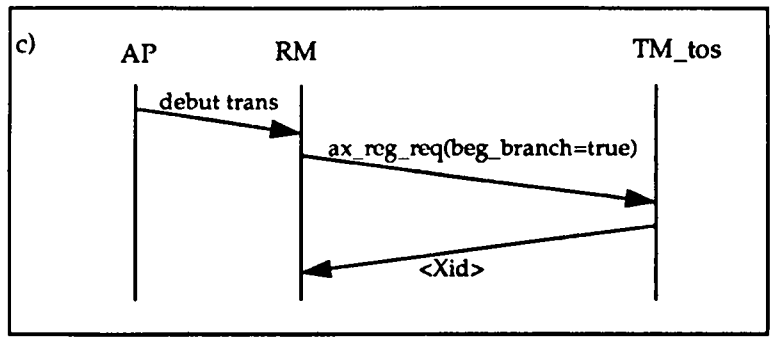
Dans le site distant, un AP et un RM seront démarrés. Le RM se retrouve dans le cas 6.4 a) (voir ci-dessus).



c) Enregistrement d'une branche de transaction native

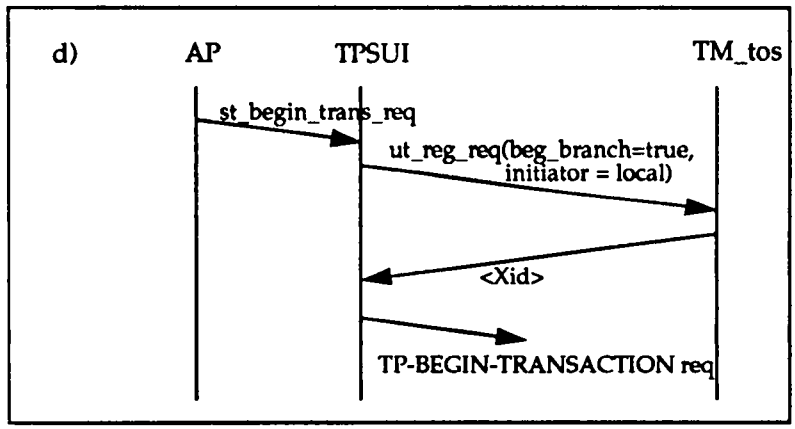
L'application ouvre une branche de transaction partielle avec un RM. Elle fait cela en utilisant le service natif du RM. Le RM envoie un ax_reg_req à TM_tos avec beg_branch = true. TM_tos, qui devient coordinateur d'une nouvelle transaction partielle, crée un

identificateur unique pour la transaction et passe dans l'état *'transaction partielle'*. Dans sa réponse, il envoie le Xid au RM.



d) Enregistrement d'un début de transaction sur un dialogue, coté supérieur

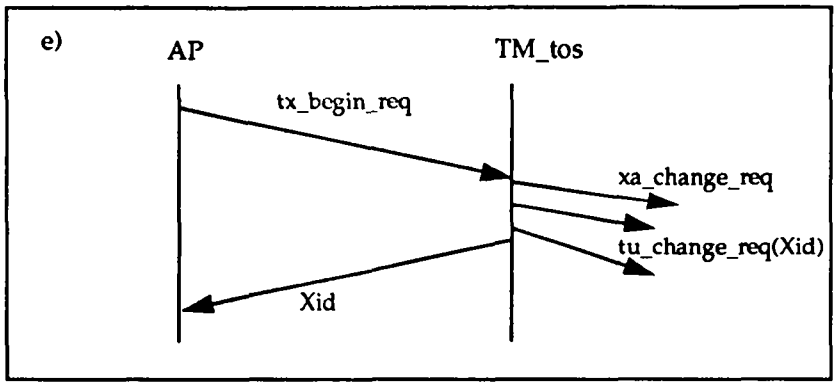
L'application ouvre un dialogue au niveau 'commitment' (ou lève le niveau d'un dialogue du 'none' au 'commitment'). Parce qu'il s'agit du premier dialogue au niveau 'commitment' (sinon TM_tos ne serait pas dans l'état *'hors transaction'*), le TPSUI envoie un ut_reg_req avec beg_branch = true et initiator = local a TM_tos. TM_tos, qui devient coordinateur d'une nouvelle transaction partielle, crée un identificateur unique pour la transaction et passe dans l'état *'transaction partielle'*. Dans sa réponse, il envoie le Xid au TPSUI. Le TPSUI ouvre le dialogue au niveau 'commitment' (ou envoie le TP-BEGIN-TRANSACTION req). Pour les prochains dialogues au niveau commitment, il n'y a pas de nouvelle enregistrement du TPSUI.



e) Enregistrement d'une transaction globale

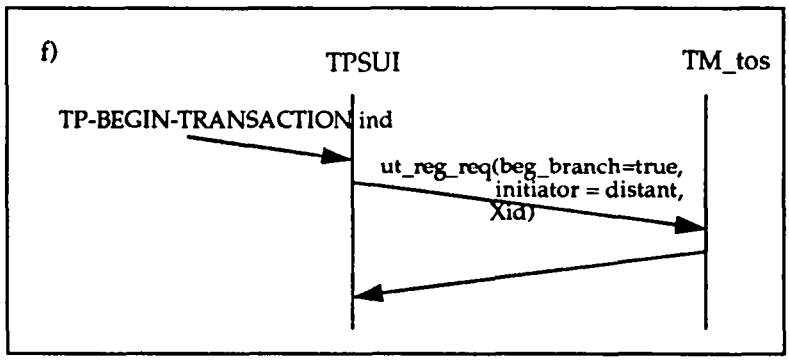
L'application démarre une transaction globale avec le service tx_begin_req. TM_tos devient coordinateur de la transaction, il crée un identificateur unique et passe dans l'état *'transaction globale'*. Le Xid est retourné à l'application. TM_tos envoie le service xa_change_req vers tous les RMs qui ont fait ax_reg_req (beg_branch = false), pour leur indiquer le démarrage d'une transaction globale. Avant son prochain accès aux données, le RM doit se re-enregistrer auparavant avec ax_reg_req. Un tu_change_req contenant le

Xid est envoyé au TPSUI si il a fait ut_reg_req(beg_branch=false). Le TPSUI va ensuite envoyer le TP-BEGIN-TRANSACTION req sur tous ses dialogues..



f) Enregistrement d'un début de transaction sur un dialogue, coté subordonné

Dans un site subordonné (dans l'arbre de dialogues), le TPSUI reçoit un début de transaction sur un dialogue (sous forme d'un TP-BEGIN-TRANSACTION ind ou TP-BEGIN-DIALOGUE ind (beg_trans = true)). Il fait un ut_reg_req avec beg_branch = true et initiator = distant. Dans le ut_reg_req il envoie aussi le Xid qu'il vient de recevoir. TM_tos devient subordonné dans la transaction, il passe en l'état *'transaction partielle'*.



L'application distante reçoit l'information du début de transaction partielle dans un ts_begin_dial_ind(beg_trans=true) ou ts_begin_trans_ind.

Si seules les transactions globales sont supportées dans le noeud subordonné, l'application reçoit un xt_begin_ind, le noeud entre dans l'état *'transaction globale'*. Cette option de configuration de TM_tos peut être envisagée pour simplifier TM_tos. Ceci est détaillé dans la section 6.7.

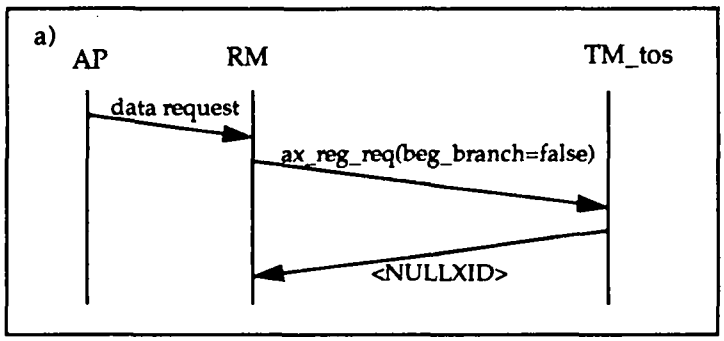
6.3.3 Enregistrement auprès de TM_tos dans l'état *'transaction partielle'*

Dans cet état, le TM a déjà créé un Xid pour une branche de transaction. On considère les cas a) à f), comme dans la section précédente.

a) Enregistrement d'un travail local hors transaction

L'application commence à travailler avec un RM local sans utiliser le service 'début transaction'. Le RM envoie un ax_reg_req à TM_tos avec beg_branch = false. La réponse

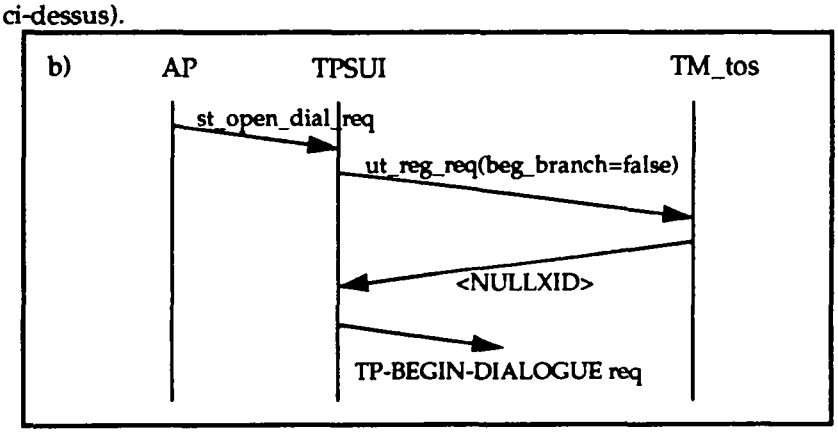
de TM_tos est négative, parce qu'il n'est pas dans une transaction globale. TM_tos mémorise qu'il a reçu un ax_reg_req de ce RM. Ce cas correspond au cas 6.4 a), où TM_tos était hors transaction.



b) Enregistrement d'un début de dialogue au niveau 'none'

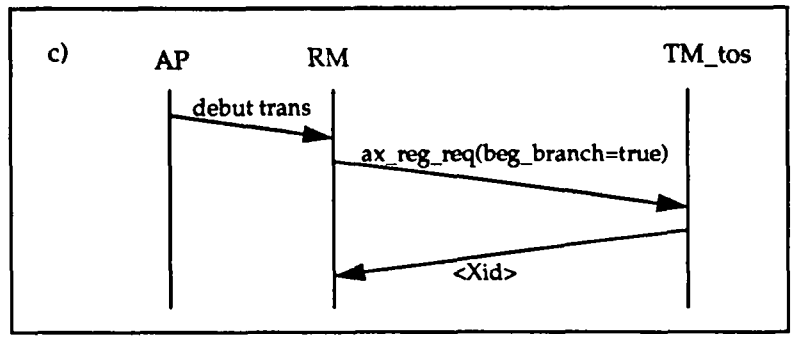
L'application ouvre un dialogue au niveau 'none'. S'il s'agit du premier dialogue à ouvrir, le TPSUI envoie a TM_tos un ut_reg_req avec beg_branch = false. La réponse du TM est négatif, parce qu'il n'est pas dans une transaction globale. TM_tos mémorise qu'il a reçu un ut_reg_req du TPSUI. Le TPSUI ouvre le dialogue au niveau 'none'. Pour les prochains dialogues 'none', il n'y a pas d'enregistrement du TPSUI.

Dans le site distant, un AP et un RM sont démarrés. Le RM se retrouve dans le cas 6.3.2.a) (voir ci-dessus).



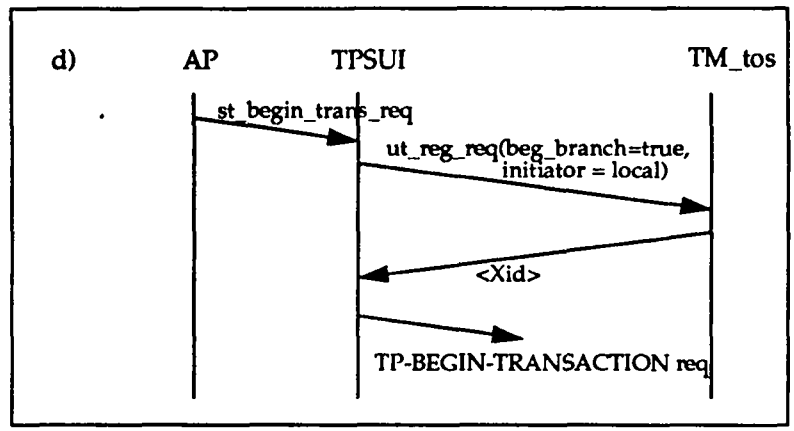
c) **Enregistrement d'une branche de transaction native**

L'application ouvre une branche de transaction partielle avec un RM en utilisant le service natif du RM. Le RM envoie un ax_reg_req a TM_tos avec beg_branch = true. TM_tos, qui est déjà dans une transaction partielle, envoie dans sa réponse le Xid au RM.



d) **Enregistrement d'un début de transaction sur un dialogue, coté supérieur**

L'application ouvre un dialogue au niveau 'commitment' (ou lève le niveau d'un dialogue du 'none' au 'commitment'). Le TPSUI envoie un ut_reg_req avec beg_branch = true et initiator = local a TM_tos. TM_tos, qui est déjà dans une transaction partielle, envoie dans sa réponse le Xid au TPSUI. Le TPSUI ouvre le dialogue au niveau 'commitment' (ou envoie le TP-BEGIN-TRANSACTION req).

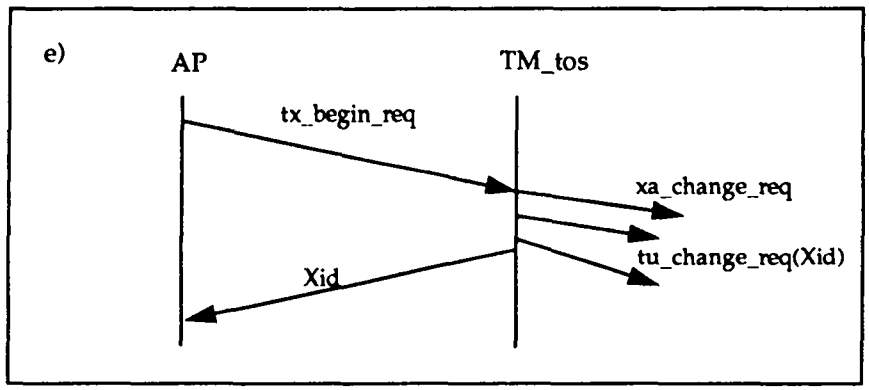


e) **Enregistrement d'une transaction globale**

L'application demande une transaction globale avec le service tx_begin_req. La transaction partielle en cours devient donc une transaction globale. TM_tos garde l'identificateur de transaction partielle et passe dans l'état *'transaction globale'*. Le Xid est retourné à l'application.

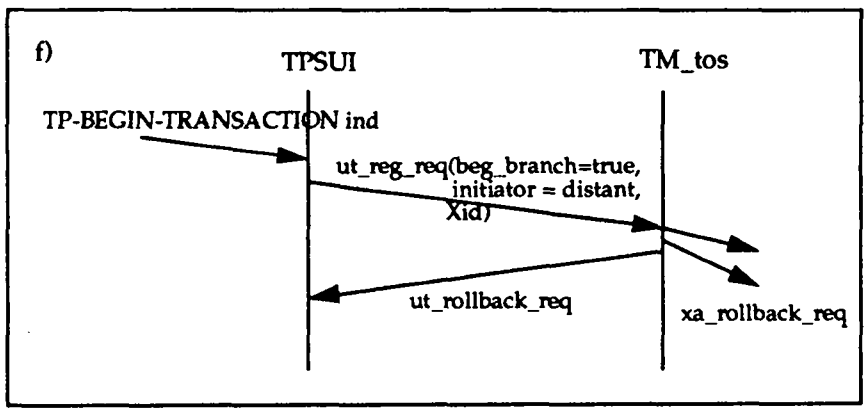
TM_tos envoie le service xa_change_req vers tous les RMs qui ont fait ax_reg_req (beg_branch = false), pour leur indiquer le démarrage d'une transaction globale. Avant son prochain accès aux données, le RM doit se ré-enregistrer avec ax_reg_req. Un

son prochain accès aux données, le RM doit se ré-enregistrer avec ax_reg_req. Un tu_change_req est envoyé au TPSUI s'il a fait auparavant ut_reg_req (beg_branch = false).



f) Enregistrement d'un début de transaction sur un dialogue, coté subordonné

Dans un site subordonné (dans l'arbre de dialogues), TM_tos est dans l'état 'transaction partielle'. Ceci est seulement le cas si l'application locale a commencé une transaction partielle, dans laquelle elle est racine. La réception d'un début de transaction sur le dialogue avec le supérieur doit provoquer le rollback des deux transactions. Si le TPSUI - et le TPPM font partie de la transaction partielle, le rollback est géré par la TPPM. Sinon, le TPSUI fait ut_reg_req avec beg_branch = true et initiator = distant. TM_tos, qui est déjà en transaction, commande le rollback de sa transaction en cours, ainsi que le rollback de la nouvelle transaction demandée par le supérieur. Une fois le rollback terminé, TM_tos passe dans l'état *'hors transaction'*.



g) Demande de terminaison de la transaction

Si l'application demande la validation ou le rollback de la transaction partielle en cours, TM_tos passe dans l'état *'terminaison de transaction'*. Il exécute le commit respectivement le rollback sur toutes les branches de la transaction, c'est à dire avec tous les RMs locaux qui se sont enregistrés avec beg_branch = true et avec le TPSUI, s'il s'est enregistré avec beg_branch = true.

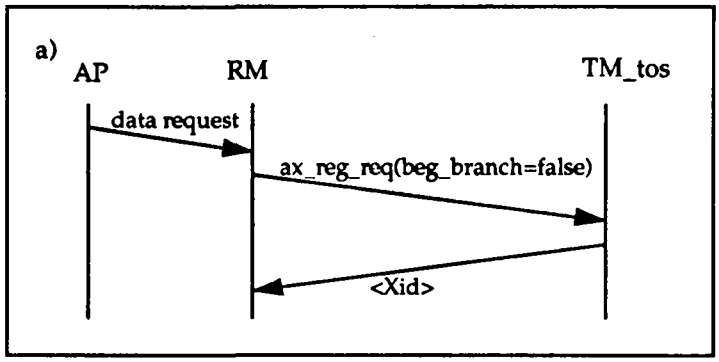
6.3.4 Enregistrement auprès de TM_tos dans l'état *'transaction globale'*

Dans l'état 'transaction globale' toutes les actions de l'AP et tous les dialogues font partie de la transaction. Quand l'AP commence à travailler avec un RM où sur un dialogue, ce travail devient automatiquement une branche de la transaction globale. Une application, qui a démarré une transaction globale, ne doit plus utiliser les services de début de transaction par branche.

On considère maintenant les cas déjà introduits.

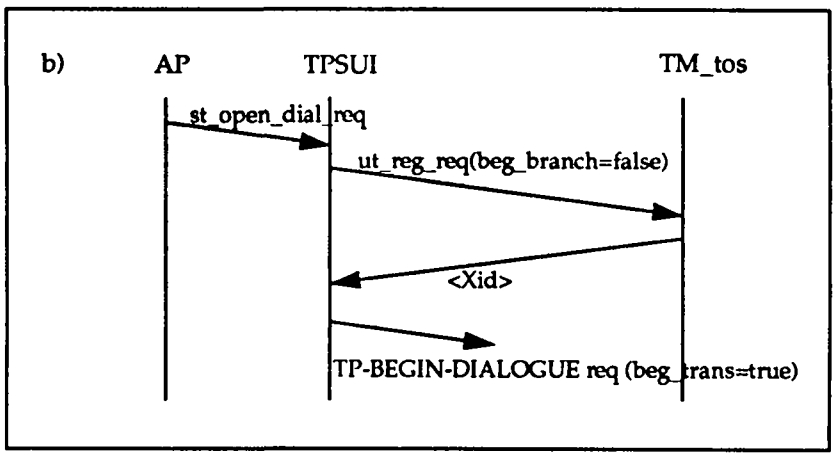
a) Enregistrement d'un travail local hors transaction

L'application commence à travailler avec un RM local sans utiliser le service 'début transaction'. Le RM envoie un ax_reg_req a TM_tos avec beg_branch = false. TM_tos, qui est en transaction globale, répond avec le Xid de la transaction en cours. Le RM en question est inclus dans la transaction globale. Ce cas est appliqué aussi pour la première requête de l'AP après la réception du xa_change_req.



b) Enregistrement d'un début de dialogue au niveau 'none'

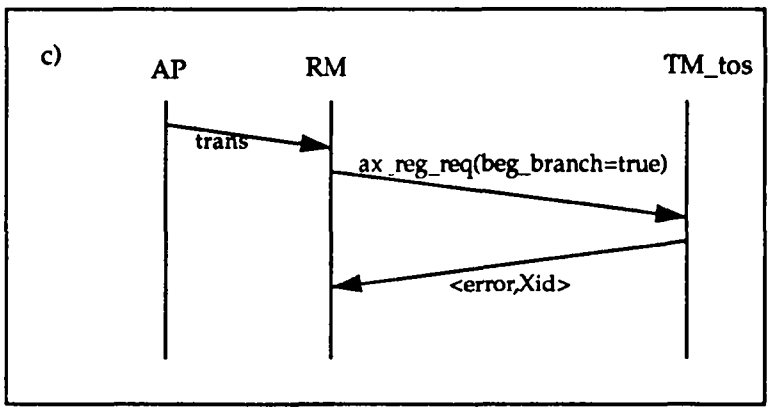
L'application ouvre un dialogue au niveau 'none'. Le TPSUI envoie a TM_tos un ut_reg_req avec beg_branch = false. TM_tos, qui est en transaction globale, répond avec le Xid de la transaction en cours. Le TPSUI est inclus dans la transaction globale, et ouvre le dialogue au niveau 'commitment'. Dans le site distant, un AP et un RM seront démarrés. TM_tos subordonné entre dans l'état *'transaction partielle'* ou *'transaction globale'*, dépendant de la configuration choisie



c) Enregistrement d'une branche de transaction native

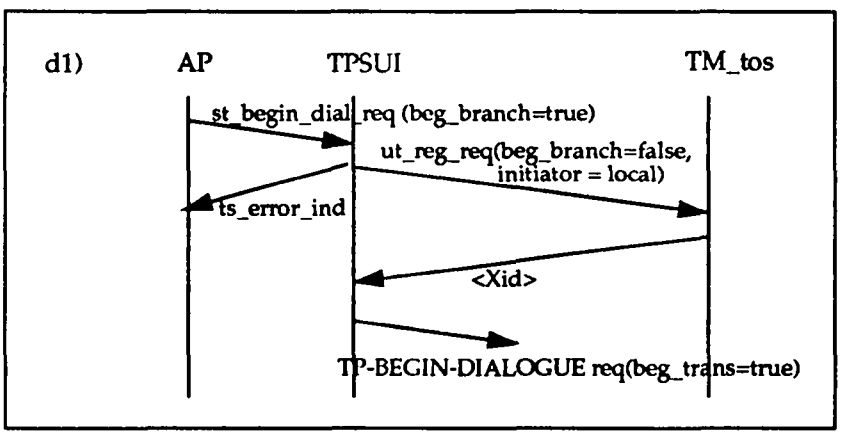
L'application ouvre une branche de transaction partielle avec un RM en utilisant le service natif du RM. Ceci est un erreur, parce que ce n'est pas à l'application d'ouvrir explicitement des branches, si il a fait tx_begin_req. Dans une transaction globale, TM_tos mettra automatiquement tous les activités de l'AP dans la transaction.

Si ce début branche est la première action vers le RM après le tx_begin_req, le RM envoie un ax_reg_req à TM_tos avec beg_branch = true. TM_tos indique l'erreur au RM mais envoie quand même envoie dans sa réponse le Xid au RM. Le RM en question est inclus dans la transaction globale. Si, par contre, l'application a fait une autre action avant, le RM a fait ax_reg_req (beg_branch=false) et a reçu le Xid. Il est alors déjà en transaction. Le traitement du deuxième début transaction est une question locale au RM..



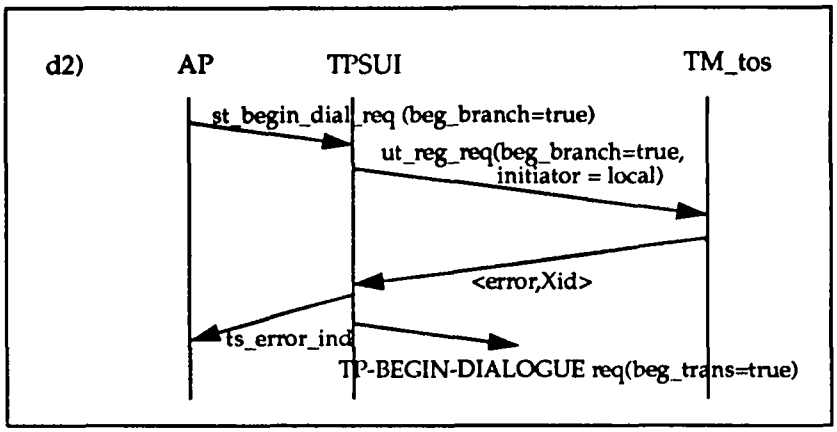
d) Enregistrement d'un début de transaction sur un dialogue, coté supérieur

L'application ouvre un dialogue au niveau 'commitment'. Si le TPSUI fait déjà partie de la transaction globale (pas la première ouverture de dialogue), il retourne un message d'erreur à l'application, (schéma d1). Le TPSUI ouvre un dialogue 'commitment' (voir cas 6.3.4 b).



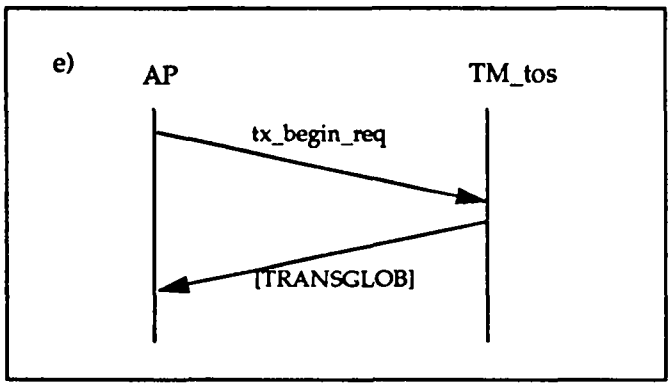
Sinon (il s'agit de la première utilisation de TPSUI) il envoie un ut_reg_req avec beg_branch = true et initiator = local a TM_tos. TM_tos, qui est déjà dans une transaction globale, indique l'erreur au TPSUI mais envoie quand même dans sa réponse le Xid au

TPSUI (schéma d2). Le TPSUI envoie un message d'erreur à l'application et ouvre le dialogue au niveau 'commitment'. Le TPSUI est inclus dans la transaction globale.



e) Enregistrement d'une transaction globale

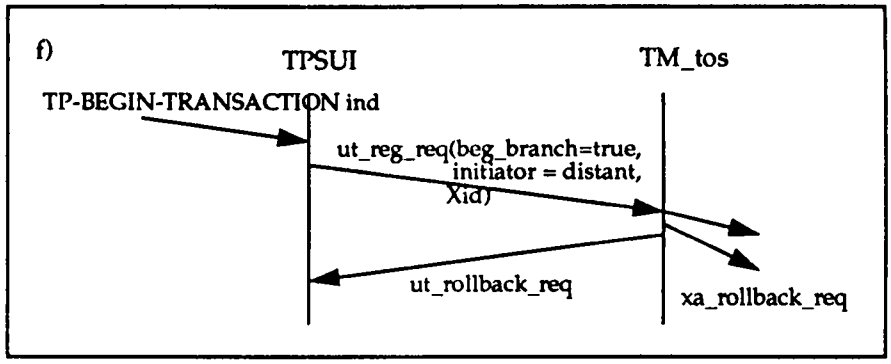
L'application demande la démarrage d'une transaction globale avec le service tx_begin_req. TM_tos refuse, parce qu'il est déjà en transaction globale. Il envoie une réponse négative a l'AP.



f) Enregistrement d'un début de transaction sur un dialogue, coté subordonné

Dans un site subordonné, le TPSUI reçoit un début de transaction sur un dialogue existant (sous forme d'un TP-BEGIN-TRANSACTION ind). TM_tos, qui est déjà en transaction globale, commande le rollback de sa transaction en cours, ainsi que le rollback de la

nouvelle transaction demandée par le supérieur. Une fois le rollback terminé, TM_tos passe dans l'état *'hors transaction'*.



g) **Demande de terminaison de la transaction**

Si l'application demande la validation ou le rollback de la transaction globale en cours, TM_tos passe dans l'état *'terminaison de transaction'*. Il va exécuter soit le commit soit le rollback sur toutes les branches de la transaction, c'est à dire avec tous les RMs locaux qui se sont enregistrés pour la transaction globale, et avec le TPSUI, si il s'est registré pour la transaction globale. Après la terminaison de la transaction TM_tos passera dans l'état *'hors transaction'*.

6.4 TM dans l'état *'terminaison de transaction'*

Dans cet état, l'ouverture des branches de transaction (locales ou distantes) n'est plus permise. Le changement d'une transaction partielle en transaction globale n'est permis non plus. On peut rendre la terminaison de la transaction brocante pour l'application pour l'empêcher de faire des actions interdites.

Pendant la terminaison d'une transaction partielle, l'ouverture des dialogues 'none' ou des RMs hors transaction, par contre, est possible. Une telle ouverture sera exécutée comme dans l'état *'transaction partielle'*.

6.5 Résumé: Table d'état pour TM_tos

Le comportement de TM_tos, vis à vis des demandes d'enregistrement, de validation et de début de transaction globale, est illustré par le table. Les services permis et l'état résultant de TM_tos sont indiqués dans les cases.

Pour les actions du TM on utilise les raccourcis suivants:

- RM_htr: Le RM est hors transaction. TM_tos le mémorise pour lui envoyer un xa_change_req dans le cas d'un tx_begin_req plus tard.
- DIAL_htr: Le dialogue est au niveau 'none'. TM_tos prendra en compte le TPSUI pour le tu_change_req dans le cas d'un tx_begin_req.
- RM_tr: Le RM est dans la transaction en cours.
- DIAL_tr: Le dialogue est dans la transaction en cours.
- change: TM_tos envoie xa_change_req ou tu_change_req si il a mémorisé des RMs locaux ou le TPSUI.

TABLEAU 4. Table d'état pour TM_tos

Service Etat	hors tr.	tr. partielle	tr. globale	term. tr.
ax_reg_req(b_b=false)	RM_htr	RM_htr	RM_tr	si globale: interdit
ut_reg_req(b_b=false)	DIAL_htr	DIAL_htr	DIAL_tr	si globale: interdit
ax_reg_req(b_b=true)	RM_tr tr. partielle	RM_tr	interdit	interdit
ut_reg_req(b_b=true)	DIAL_tr tr. partielle	DIAL_tr	interdit	interdit
tx_begin_req	change tr. globale	change tr. globale	interdit	interdit
ut_reg_req(b_b=true, init=distant)	tr.partielle	term. tr. (rollback)	term.tr. (rollback)	rollback
tx_commit_req	interdit	term. tr.	term. tr.	interdit

(htr=hors transaction, tr=en transaction, term=terminaison)

6.6 Configuration de TM_tos

Nous avons décrit le comportement d'un TM_tos qui supporte des transactions partielles. Pour des applications qui ne traitent que des transactions globales (coordination de la terminaison de transaction sur tous les RMs, locaux ou distants), on peut concevoir que TM_tos ne supporte que des transactions globales. Nous définissons donc deux unités fonctionnelles (FU), dont une doit être choisie à la configuration de TM_tos. Ce choix est local, un noeud qui utilise des transactions partielles et globales peut travailler avec un autre noeud qui ne gère que des transactions globales.

L'unité fonctionnelle pourra être choisie pendant la compilation des modules AP, TM et TPSUI ou dynamiquement dans, par exemple, le service tx_open_req. Un noeud subordonné, qui a choisi le FU 'transactions globales' entrera tout de suite dans l'état *'transaction globale'* sur réception d'un C_BEGIN_RI sur le dialogue avec le supérieur. Ceci sera signalé à l'application dans le service xt_begin_ind. Un noeud qui utilise également les transactions partielles entrera dans l'état *'transaction partielle'*, l'AP recevra un ts_begin_trans_ind ou ts_open_dial_ind(beg_trans=true). Il aura la possibilité d'entrer en transaction globale utilisant tx_begin_req.

Il est donc nécessaire que l'unité fonctionnelle choisie dans un noeud soit connu par TM_tos, le TPSUI et l'application.

L'unité fonctionnelle 'transactions globales' contient les services

- tx_begin_req pour démarrer une transaction globale (interdit dans un noeud subordonné)
- xt_begin_ind pour l'indication du début de transaction globale (dans un noeud subordonné seulement)
- st_open_dial_req avec beg_trans = false ou sans beg_trans pour ouvrir un dialogue au niveau 'none' ou 'commitment' dépendant de l'état de transaction globale

L'unité fonctionnelle 'transactions partielles' contient les services

- tx_begin_req pour entrer dans une transaction globale (racine et subordonné)
- st_begin_trans_req pour ouvrir une branche de transaction sur une dialogue
- st_open_dial avec beg_trans = true pour ouvrir un dialogue avec branche de transaction.

7 Sérialisation du traitement des services

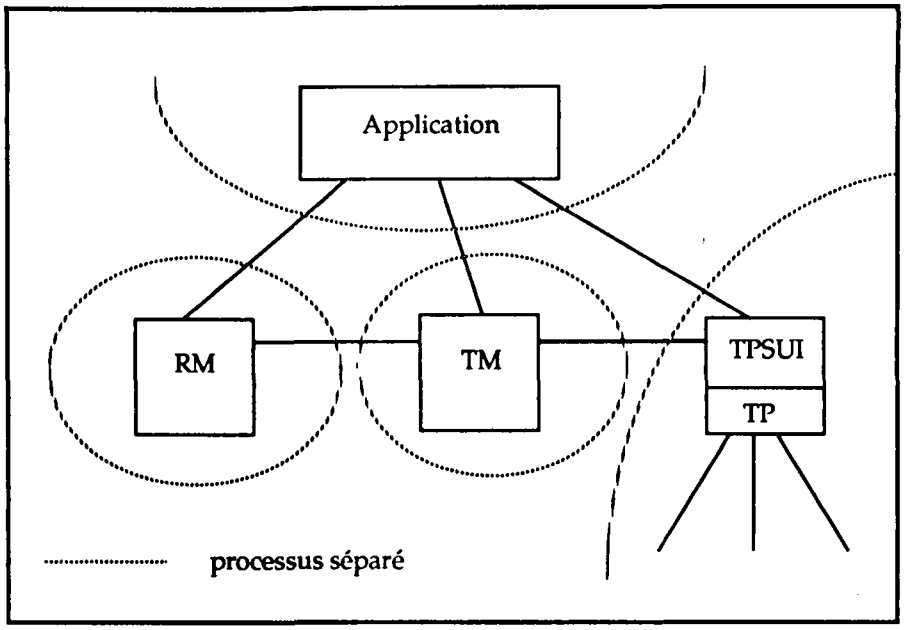
La philosophie de X/Open impose que l'application, le RM et TM_tos s'exécutent dans le même thread. Ceci implique, que le RM et TM_tos soient réalisés sous forme des fonctions, appelées par l'application. Ce type de réalisation ne permet aucun parallélisme entre les modules application, RM et TM. A un moment donné, une seule fonction dans l'un de ces modules est exécutée. Par contre, chaque événement est entièrement traité avant le suivant.

L'approche de X/Open est donc très contraignante, voir peu performante, en ce qui concerne la réalisation et conduit à la recherche d'une réalisation moins restrictive avec le plus de parallélisme possible. Nous préférons définir des processus séparés pour les différents modules. Nous considérons l'application et les RMs comme des processus séparés, parce qu'on désire utiliser des RMs et des applications existants.

Une telle architecture pose des problèmes au niveau de la communication entre processus. Toutefois ce choix est maintenu par le souci de préserver l'architecture initiale de ODIS-X.

La conception primaire consiste en ce que chaque module soit un processus, suivant la Figure 10.

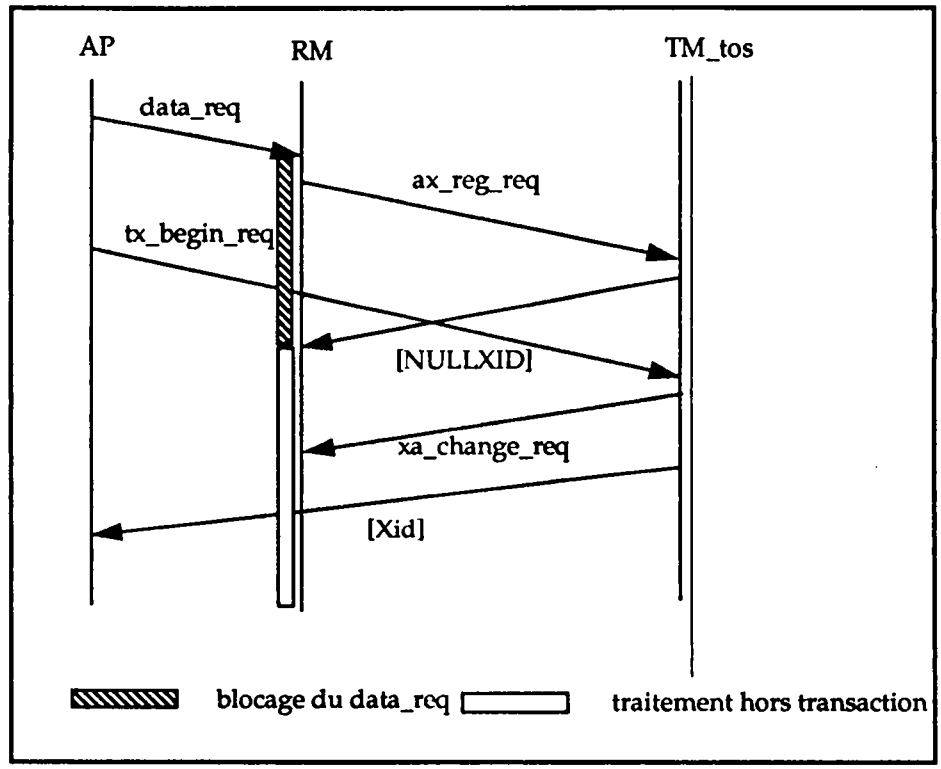
FIGURE 10. :Séparation des modules



Les interfaces entre les modules sont réalisées sous forme de IPC (communication inter-processus), les échanges de messages se déroulent de façon asynchrone. Les messages sont envoyés et placés dans des 'boites aux lettres' (files d'attente) et sont récupérés par le module destinataire sur la même file.

Avec une telle architecture, on ne peut pas garantir qu'un service reçu de l'AP soit complètement exécuté par tous les processus concernés avant la prise en compte du service suivant. En d'autres termes, on ne peut pas garantir que les traitements des services soient sérialisés. Chaque fois que sur un interface, peuvent s'enchainer demande d'accès aux données et demande d'enregistrement, la serialisation des traitement est nécessaire. Ne pas la garantir risque de rompre l'intégrité des données manipulées par les transactions. On note que ce problème ne se pose que vis à vis du même module. Un exemple en est l'envoi d'un service natif au RM suivi du tx_begin_req vers TM_tos. Le Scénario 3 montre le déroulement de cette séquence.

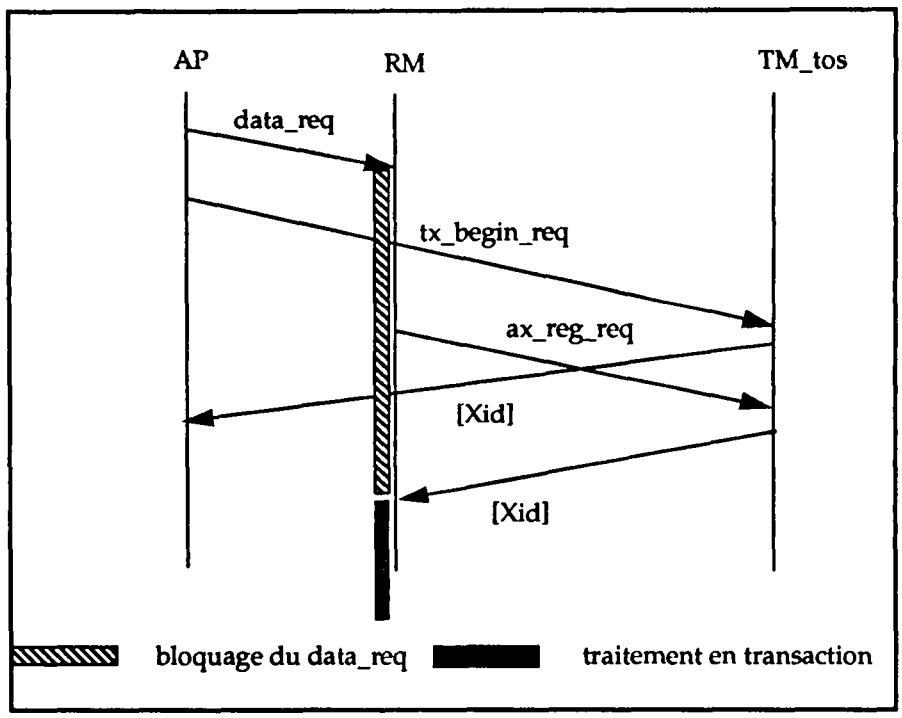
SCENARIO 3. Services serialisés



Ici, le résultat est ce qu'on souhaite: L'appel au RM, qui a été fait avant le tx_begin_req, ne fait pas partie de la transaction globale qui est démarrée. On rappelle que le data_req est exécuté immédiatement sur réponse à ax_reg_req. La réponse à ax_reg_req est négative. Le RM n'entre dans la transaction globale qu'avec le prochain service qu'il recevra de l'AP. A ce moment, le RM répète le ax_reg_req (parce qu'il a reçu le xa_change_req) et cette fois reçoit une réponse positive.

Malheureusement, on ne peut pas garantir que le data_req soit traité le premier. Par exemple, si TM_tos commence à traiter tx_begin_req avant que le RM ne traite le data_req, alors le résultat peut être différent comme le montre le Scénario 4.

SCENARIO 4. Services non serialisés



Dans ce deuxième cas, l'appel au RM entre dans la transaction globale, parce que TM_tos traite le ax_reg_req du RM après le tx_begin_req. TM_tos répond donc avec l'identificateur de la transaction globale. Le RM traite le data_req comme s'il faisait partie de la transaction globale.

Les algorithmes du TM_tos sont tels que la contrainte de sérialisation ne s'applique qu'aux services reçus sur les interfaces de l'AP, et relativement à certaines primitives.

Traiter les événements dans leur ordre d'occurrence ne peut être résolu pas un outil qui maintienne l'atomicité du traitement des événements. Ainsi le 'data_req' doit être complètement traité avant que TM_tos traite le tx_begin_req. Une telle atomicité ne peut pas être garantie, si l'on crée des processus indépendants avec des communications séparées.

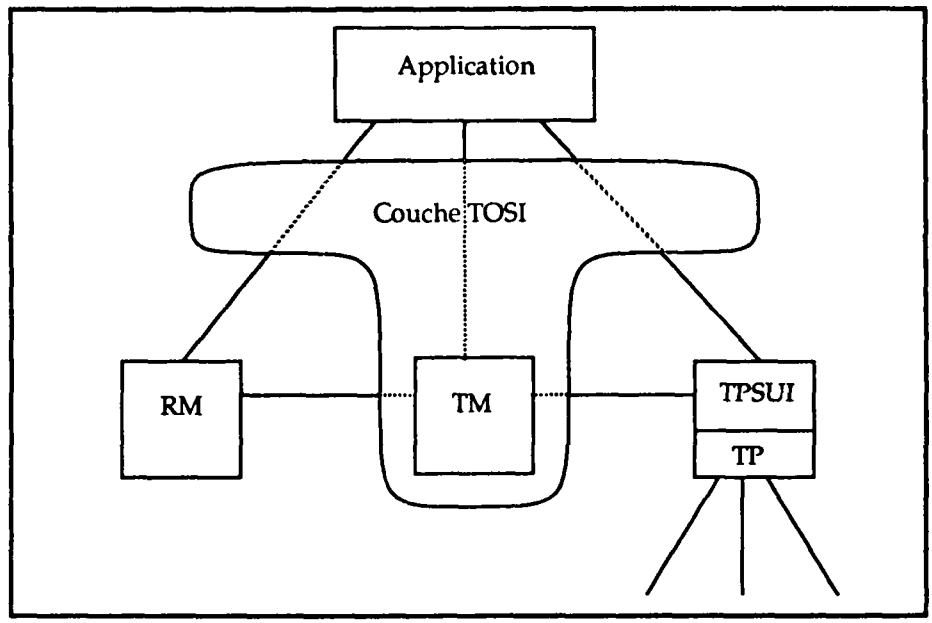
Nous voyons deux solutions pour résoudre ce problème. La première solution consiste à exécuter (comme l'exige X/Open), AP, RM et TM dans le même processus. Dans ce cas, tout traitement est complètement séquentialisé, ce qui n'est pas toujours nécessaire.

La deuxième solution consiste à ne pas grouper les modules dans le même processus mais créer un module 'gardien', qui garantisse la non préemption du traitement d'un service, lorsque nécessaire. Ce module 'gardien' doit donc contrôler toutes les interfaces concernées.

Dans notre cas, le rôle du 'gardien' sera assuré par la couche appelée TOSI. TOSI accepte tous les messages sur les interfaces et les met en files d'attente. Par définition, TOSI peut choisir de ne traiter qu'un message à la fois et donc garantir la non préemption du traitement.

Pour pouvoir filtrer tous les interfaces nécessaires, la couche TOSI est placée comme le montre la Figure 11.

FIGURE 11. Introduction de la couche TOSI



TOSI reçoit tous les messages envoyés sur les interfaces ts, tx_tos, tu, xa_tos et sur les interfaces natives des RMs. TOSI propage les messages vers le module destinataire.

Pour la conception de cette couche il y a quelques exigences supplémentaires à respecter:

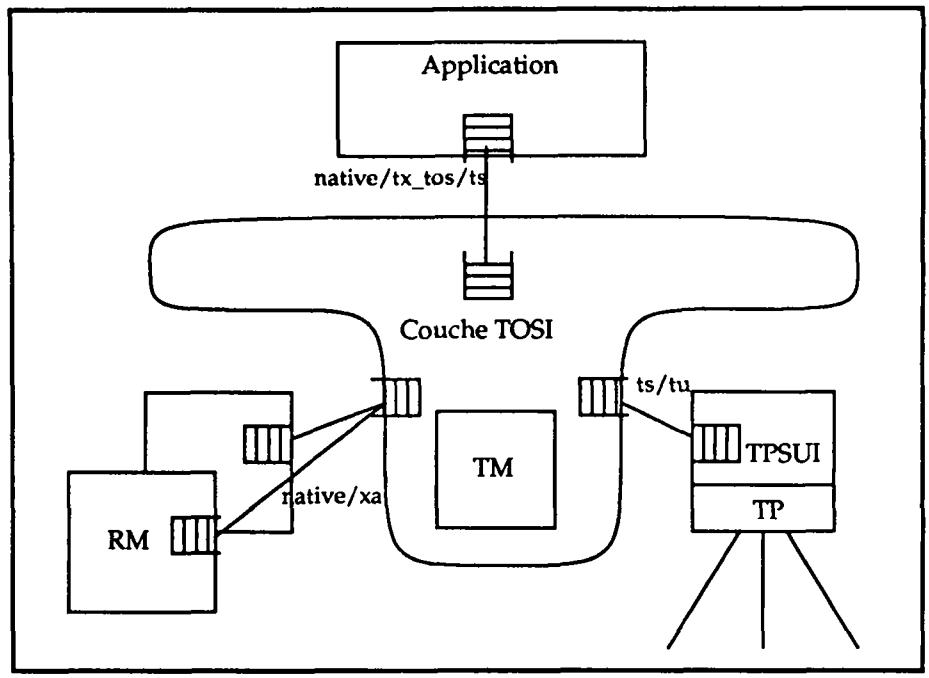
1. La présence de TOSI doit être transparente à l'application, aux RMs et au TPSUI.
2. Le traitement séquentiel des services ne doit pas bloquer l'application.
3. TOSI doit être capable de traiter les services reçus de la même source, par exemple sur les interfaces tx_tos et native, dans leur ordre d'arrivée.
4. La présence de TOSI ne doit pas trop compliquer l'interconnection des modules.
5. On ne veut pas dupliquer dans TOSI les fonctions d'analyse des services de TM_tos.

C'est à cause des exigences 2.) et 5.) que TOSI est placée dans le même processus que TM_tos, au lieu d'être réalisée sous forme de fonctions appelées par l'AP.

L'architecture avec TOSI se présente comme le montre la Figure 12. Cette architecture minimise le nombre de files d'attente entrante de TOSI.

Tous les événements arrivant du même module sont placés FIFO dans une seule file d'attente. TOSI a une seule file d'attente des messages venant du TPSUI, une seule file d'attente des messages venant de l'AP, une seule file d'attente des messages venant des RMs. L'AP, le TPSUI et chaque RM ont chacun une seule file d'attente pour tous les messages venant de TOSI.

FIGURE 12. L'architecture avec TOSI



Ainsi, des interfaces différents se rejoignent dans des files communes par module. Un interface est en fait un ensemble de messages échangés entre deux modules. Le récepteur des messages va faire la distinction entre les interfaces différents et traiter chaque message suivant son objectif.

On va ensuite préciser les restrictions à appliquer sur le traitement des services entrants. En fait, l'exigence d'un traitement atomique de tous les événements entrants est trop fort. Il y a beaucoup d'événements qui n'ont pas d'interférence et qui peuvent être traités en parallèle. Ce sont tous les services pour l'accès aux données, les services natifs sur différents RMs, les st_data_req qui sortent et les ts_data_ind qui arrivent en séquence.

La difficulté se situe au niveau des services délimiteurs des transactions (tx_begin_req et ut_reg_req avec beg_branch = true, initiator = distant), des services qui affectent le changement d'état d'une transaction (tx_commit_req, ut_prepare_ind, ut_commit_ind, tx_rollback_req, ut_rollback_ind) et des services d'accès aux données venant du même module. Il est nécessaire de traiter chaque accès aux données avant ou après un changement d'état de transaction, en préservant l'ordre d'arrivée de ces services en file d'attente.

Note: Les services qui indiquent le début d'une transaction partielle ne sont pas inclus dans la groupe des délimiteurs, ceci parce que le début d'une transaction partielle n'affecte qu'une seule branche de transaction, c'est à dire un seul RM ou un seul dialogue. Le groupe des délimiteurs ne contient que les services du TM qui affectent toutes les branches de transaction ou plusieurs RMs ou dialogues. Le service ut_reg_req (beg_branch = true) forme un cas spécial: Selon l'unité fonctionnelle choisie, il signale le début d'une transaction globale ou partielle.

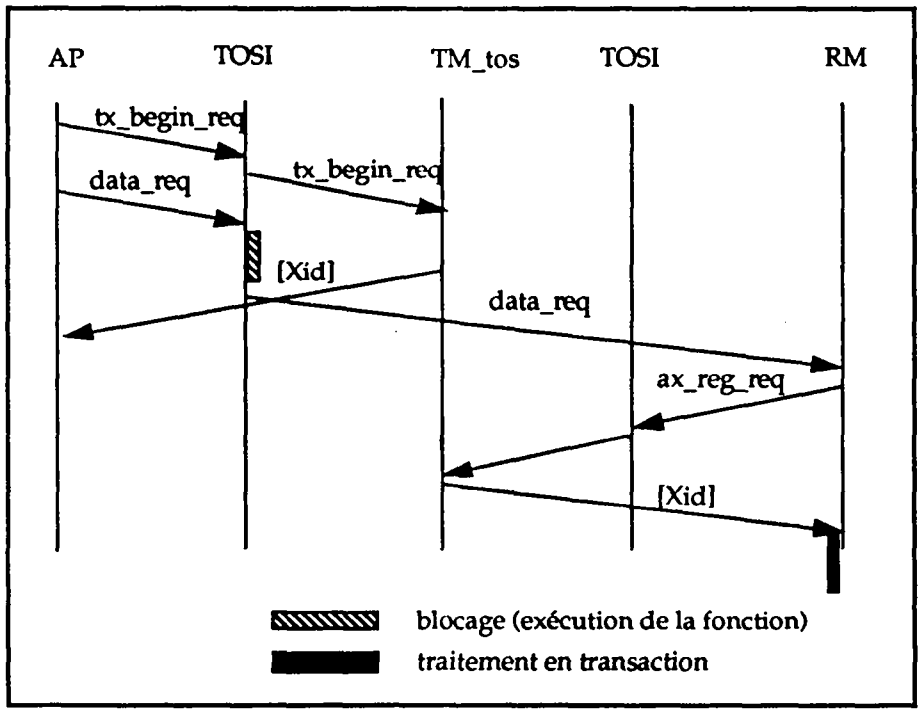
Pour les événements du premier groupe, il n'y a pas de restriction du parallélisme. Tous les st_data_req, ts_data_ind et tous les services natifs des RMs peuvent être traités sans atomicité. Pour les événements du deuxième groupe, qui sont tous des services du TM, l'atomicité est assurée par la façon dont ils sont implantés. En effet, comme TM_tos est dans le même processus que TOSI, les services du TM sont réalisés sous forme de fonction. Un appel de fonction garantit le traitement atomique. On doit seulement intégrer tout traitement qu'on souhaite exécuter atomiquement dans la fonction appelée.

Entre les deux groupes, on a deux situations d'interférence:

1. Les demandes d'accès aux données reçues après un service provoquant le changement d'état de la transaction, doivent être traitées après que le service qui change l'état de la transaction soit être complètement traité.
2. Le changement d'état de transaction après un (ou plusieurs) accès aux données. Ici on doit éviter que le service du TM double l'accès aux données lancé avant.

Le cas 1) pose en fait peu de problème, parce que les changements d'état se font d'une manière atomique (voir ci-dessus). Ceci assure qu'un accès aux données arrivant après un service du TM sera traité après l'exécution de ce dernier[2]. Un exemple est l'envoi d'un data_req après un tx_begin_req. Dans ce scénario, chaque ligne 'TOSI' représente une file d'attente entrante de TOSI.

SCENARIO 5. Traitement du data_req après tx_begin_req



2. La préservation de l'ordre est seulement garanti pour les événements qui arrivent dans la même file d'attente, l'ordre d'exécution pour des événements arrivant dans des files différentes doit être examiné séparément

Le tx_begin_req est traité complètement avant que le data_req soit envoyé au RM concerné. Ce comportement assure la préservation de l'ordre des services.

Le cas 2), par contre, est moins simple. Le premier problème est que TOSI ne connaît pas la fin d'un accès aux données. Comme on ne veut pas imposer de restrictions sur l'interface native entre RM et AP, on peut, par exemple, ne pas supposer que tout accès aux données sera confirmé.

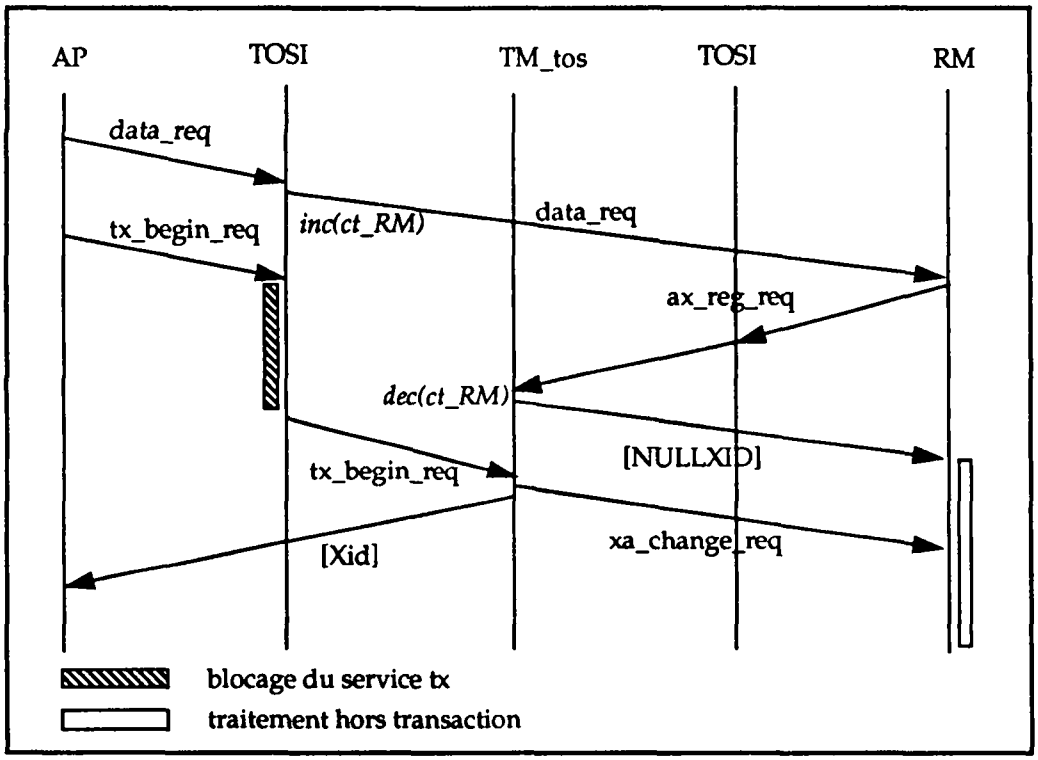
Pour les services venant de l'application, on peut préciser les cas à surveiller. Ce sont les cas où le traitement d'un service nécessite d'abord une enquête auprès du TM pour obtenir l'état de transaction. Un RM fait une telle enquête avec ax_reg_req, le TPSUI la fait avec ut_reg_req (beg_trans=false, initiator=local). Ici, il est important que ce ax_reg_req ou ut_reg_req se fasse avant que TM_tos traite un service tx envoyé après[3].

TOSI, qui est dans le même processus que TM_tos, peut connaître d'avance, quand un RM va faire ax_reg_req (ou le TPSUI va faire ut_reg_req). Il y a deux cas où un service native provoque un ax_reg_req: C'est le premier service envoyé à un RM après la fin de la dernière transaction et le premier service après le début d'une transaction globale (donc après un xa_change_req envoyé vers le RM). Ceci permet d'appliquer l'algorithme suivant:

TOSI garde un compteur (appelé ct_RM) qu'elle incrémente quand elle envoie un service vers un RM qui va provoquer ax_reg_req. Ce compteur sera décrémenté par TM_tos quand il a traité le ax_reg_req. Un service tx de l'AP (et avec lui tous les services envoyés après) va être bloqué si le compteur n'est pas égal zéro. Ceci nécessite que TOSI puisse accéder au tableau des RMs géré dans TM_tos. Le comportement qui résulte de cet algorithme est illustré par le Scénario 6.

3. Ce cas est illustré par les scénarios au début du chapitre.

SCENARIO 6. Traitement du tx_begin_req après data_req



Le TPSUI va faire ut_reg_req quand il recevra le premier ts_open_dial_req après la fin de la dernier transaction ou si le ts_open_dial_req marque le début d'une branche de transaction. Le TPSUI est surveillé de la même façon que les RMs, avec un compteur.

Pour les services venant du TPSUI, le traitement est un peu plus complexe. C'est l'application, qui reçoit les ts_data_ind et qui se charge des actions nécessaires. Un service ts_data_ind peut déclencher aucun, un ou plusieurs services natifs vers le(s) RM(s), ces services peuvent être confirmés ou non. Un ts_data_ind peut même déclencher un échange sur un dialogue avec un subordonné. Le ts_data_ind lui-même peut être confirmé ou non-confirmé. C'est seulement l'AP qui sait, quand son traitement d'un ts_data_ind est fini.

Pour assurer au niveau noeud, que tout traitement des ts_data_ind reçus avant est fini, on va introduire un échange explicite avec l'AP local avant un changement d'état global de la transaction. Cet échange n'est nécessaire que quand le changement d'état de transaction est dû à un message reçu sur un dialogue. (Si c'est l'AP qui demande le changement d'état, il doit avoir fini tout traitement avant.) Plus précisément, on a besoin d'un échange avec l'AP dans les situations suivantes:

- début de transaction globale dans un noeud subordonné (ut_reg_req(beg_trans=true, initiator=distant))
- réception du prepare dans un subordonné (ut_prepare_ind)
- réception du rollback dans un subordonné ou supérieur (ut_rollback_ind).

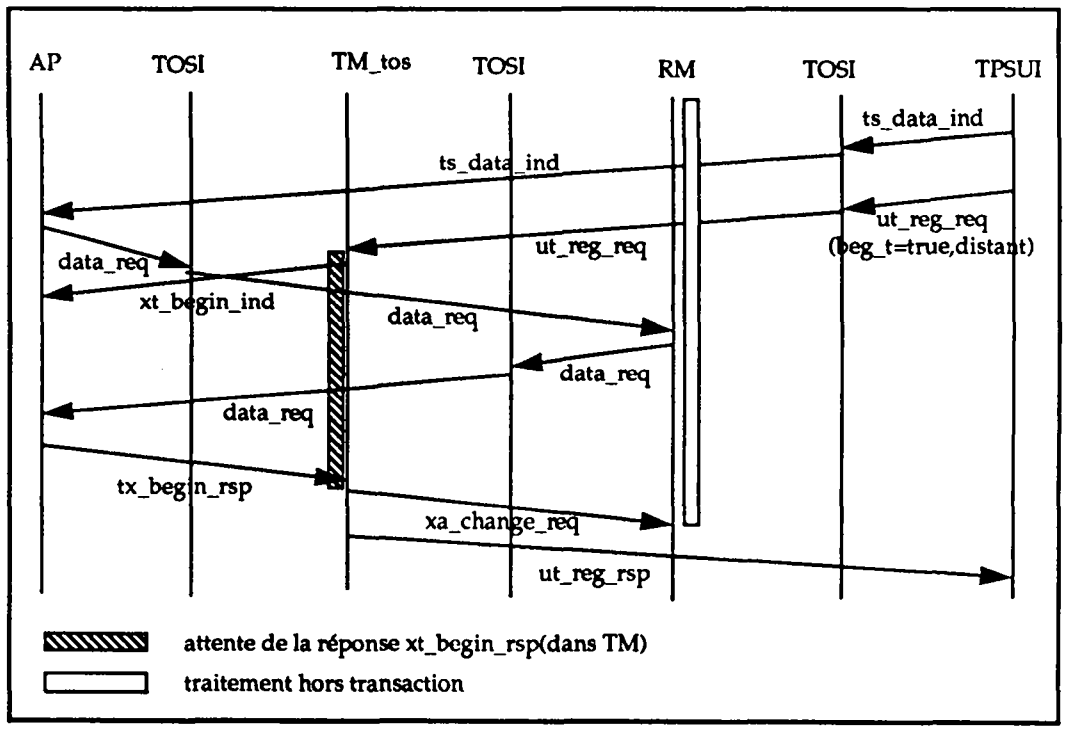
Dans ces trois cas, TM_tos, sur réception du message indiqué, fera un échange de messages avec l'AP avant d'envoyer des messages aux RMs ou de changer son état global. Ceci implique:

- un service confirmé pour xt_begin_ind
- un service confirmé pour xt_rollback_ind
- l'indication du prepare vers l'AP avec une réponse positive ou négative de l'AP

Ces échanges permettent à l'AP de finir le traitement de toutes les requêtes qu'il a reçu avant. TM_tos ne doit pas envoyer de services xa aux RMs avant l'acquittement de l'AP. Ce comportement nous permet de garantir la délimitation de transaction comme le garantit le protocole OSI TP.

Le Scénario 7 explique le séquencement avec un tel échange.

SCENARIO 7. Début de transaction dans un noeud subordonné



Ici, l'application attend la réponse du xt_begin_ind jusqu'à ce que le traitement du ts_data_ind soit fini. L'AP a également la possibilité d'éviter le démarrage d'une transaction. Dans ce cas (la réponse du xt_begin_ind est négative), TM_tos, qui n'est pas encore dans l'état 'transaction globale' doit envoyer un tu_rollback_req vers le TPSUI pour effectuer le rollback de la transaction avec le supérieur. Il n'envoie rien aux RMs, parce que ces derniers ne sont pas encore en mode transactionnel.

On définie donc les messages supplémentaires

- tx_begin_rsp

- xt_prepare_ind
- tx_ready_req (tx_rollback_req sert comme réponse négative)
- tx_rollback_rsp.

Pour les messages qui arrivent sur des interfaces différentes, TOSI ne peut pas garantir l'ordre d'exécution.

Elle peut, par contre, faire la purge dans les cas ou certains messages doivent être supprimés parce qu'ils ne sont plus permis (par exemple dans les cas de rollback). Les cas où, en même temps, des services pour TM_tos arrivent du supérieur et de l'AP, sont des cas de collision. Dans un tel cas, TOSI est capable de prendre les décisions nécessaires, parce qu'elle contient TM_tos. Ce dernier est toujours au courant de l'état de la transaction et des services permis et non permis.

Ces cas de collisions représentent des problèmes au niveau sémantique de l'application distribuée. Une application, qui anticipe un début transaction ou un tx_commit prend le risque d'un rollback dans des cas de collision, si une des applications subordonnées envoie encore des données.

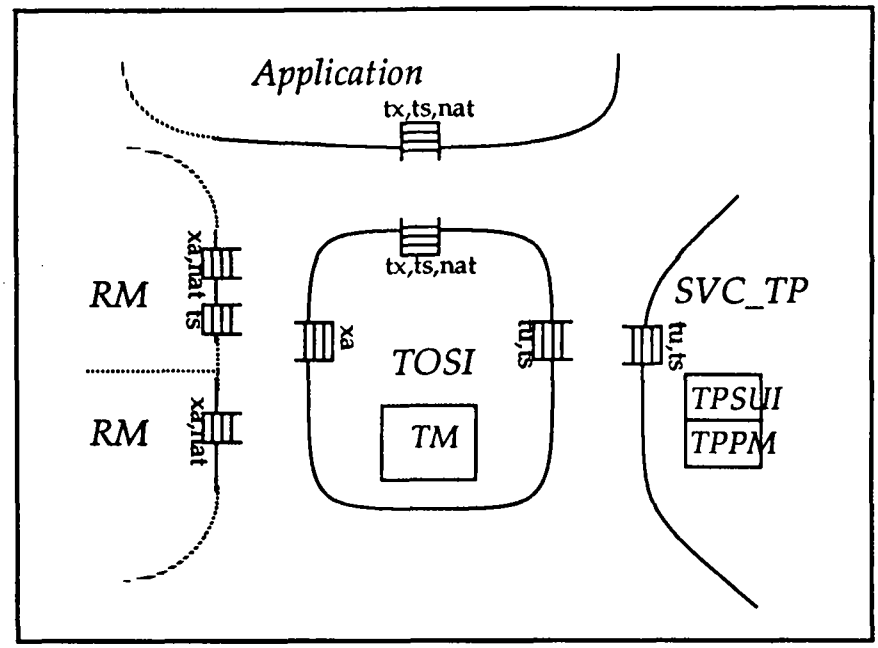
7.1 Conclusion

La couche TOSI

1. fournit sous forme de boîtes aux lettres des interfaces tx, xa, tu, ts et native_RM
2. route les services vers les boîtes aux lettres concernées
3. assure la délimitation des transactions par un échange explicite avec l'AP
4. garantit que tous les traitements des événements venant d'un processus (soit un RM, le TPSUI, l'AP) sont serialisés.
5. garantit qu'un message sur l'interface native qui déclenche un ax_reg_req sera traité avant qu'on traite un service tx qui suit
6. garantit qu'un ts_open_dial_req qui déclenche ut_reg_req sera traité avant qu'on traite un service tx qui suit

Pour obtenir une architecture plus flexible, qui permet par exemple que l'application et un RM soient dans le même processus ou qu'un RM utilise l'interface ts (peer-to-peer) pour ouvrir des dialogues, on va généraliser le modèle. Cette généralisation donne à peu près le schéma suivant.

FIGURE 13. TOSI et les boîtes aux lettres



TOSI a trois boîtes aux lettres dans lesquelles elle reçoit des messages:

- une boîte pour les interfaces tu et ts, les messages venant du TPSUI
- une boîte pour les interfaces tx, ts, et native, les messages tx venant de l'application, les messages ts et native venant de l'application ou des RMs
- une boîte pour les interfaces xa, les messages venant des RMs

TOSI sait distribuer les messages qu'elle a reçus dans plusieurs boîtes aux lettres:

- une boîte pour l'interface tx, l'interface native avec les RMs, côté application, et les services ts reçus sur le dialogue avec le supérieur
- une boîte par dialogue subordonné pour l'interface ts (réception)
- une boîte par RM pour l'interface native côté RM et l'interface xa
- une boîte du serveur SVC_TP, pour les interfaces tu et ts (envoi)

TOSI ne pose aucune restriction aux propriétaires des boîtes aux lettres destinataires. Toutes ces boîtes peuvent appartenir au même processus, ou chaque boîte peut appartenir à un autre processus. Si une application utilise un RM qui ne fait pas partie du même processus, elle est obligée d'utiliser les boîtes aux lettres fournis par TOSI pour la communication avec le RM. Les services natifs sont enveloppés dans des messages génériques. Chaque enveloppe va donc porter l'adresse d'une boîte aux lettres ou le nom d'un processus. Comme ça, il est aussi possible qu'un RM envoie des services natifs à un autre RM. TOSI va appliquer l'algorithme concernant le ax_reg_req également aux services natifs venant d'un RM vers un autre RM.

Plus précisément, ceci veut dire que TOSI vérifie pour chaque service natif envoyé vers un RM, si ce service déclenche un ax_reg_req, ainsi que pour chaque service ts_open_dial_req, s'il déclenche un ut_reg_req.

Une application peut aussi appeler un RM par fonctions, si le RM existe dans le même processus. Dans ce cas, les communications entre RM et AP ne sont pas visibles pour TOSI. (Ici, le problème de parallélisme entre tx_begin et ax_reg n'existe pas, parce que chacun des deux services est bloquant pour le processus AP/RM.)

L'algorithme de la couche TOSI, concernant la surveillance de l'ordonnancement des messages, est le suivant:

```
recevoir message de l'AP;
si message_tx en attente
    mettre message dans la file d'attente;
sinon
    si type = native
        si message va déclencher ax_reg_req
            ct_RM = ct_RM + 1;
        envoyer message dans la BAL du RM
    si type = ts
        si message va déclencher ut_reg_req
            ct_TPSUI = ct_TPSUI + 1;
        envoyer message dans la BAL du TPSUI
    si type = tx
        si ct_RM = zéro et ct_TPSUI = zéro
            appeler TM;
        sinon
            mettre message_tx dans une file d'attente

recevoir message du RM;
Si type = native
    si message va déclencher ax_reg_req
        ct_RM = ct_RM + 1;
    envoyer message dans la BAL du destinataire
Si type = xa
    appeler_TM;

recevoir message du TPSUI;
si type = ts
    envoyer message dans la BAL de l'AP
si type = ut
    appeler_TM;

si message tx en attente et ct_RM = zéro et ct_TPSUI = zéro
    récuperer message_tx;
    appeler_TM:
    vider file d'attente;
```

8 Conclusion

Il est incontestablement opportun de chercher à valider une architecture de système transactionnel avec l'utilisation entière du protocole OSI TP, puisqu'il a été voté en norme internationale le 11 Avril 1992. Il est non moins utile de chercher à insérer OSI TP dans une architecture aussi voisine que possible de ce que sera l'interface standard entre application et moniteur transactionnel. Ceci permet à la fois d'acquérir le savoir-faire dans le domaine et d'identifier les aspects architecturaux qui, s'ils ne sont pas pris en compte dans un produit, seront une contrainte pour le développeur d'application.

Nous avons étendu le moniteur transactionnel et les interfaces tx et xa tels que nous avons pu les comprendre des travaux courants de X/Open pour permettre au développeur d'application de programmer des transactions reparties en un arbre dont les noeuds sont sur des machines distinctes.

Chaque noeud comporte un moniteur transactionnel et un gestionnaire de communication constitué par OSI TP et TCP. Tout noeud peut contenir un processus application et un processus gestionnaire de ressources. Pour réaliser ceci, nous avons ajouté les services tx_prepare ind, tx_commit ind et tx_rollback ind.

Un processus application peut travailler avec un gestionnaire de données, local ou distant, en mode non transactionnel ou en mode transactionnel. Pour coordonner le passage de l'un a l'autre, entre gestionnaire de données et moniteur, nous avons crée le service xa_change req et introduit un paramètre dans le service ax_reg req.

Toutes les données impliquées dans le travail en mode transactionnel sont validées ou annulées atomiquement en fin de transaction. Sur option, le gestionnaire de transaction place tous les gestionnaires en mode transactionnel. Un processus application peut démarrer un travail avec un gestionnaire de données à tout moment, des lors que la transaction n' est pas en cours de terminaison et sous la responsabilité des moniteurs.

Application, moniteur, gestionnaire de données et gestionnaire de communications sont des processus asynchrones qui communiquent entre eux par messages. Ceci offre le maximum de flexibilité pour utiliser des serveurs de données existants. Nous avons définis des mécanismes de file d'attente et compteurs dans le moniteur transactionnel de façon à garantir la délimitation des données par rapport à l'instant de passage du gestionnaire de données en mode transactionnel. Les données modifiées ou transmises avant le passage en mode transactionnel sont considérées hors transaction, les données modifiées ou transmises après le passage en mode transactionnel sont considérées à l'intérieur de la transaction.

Cette architecture offre le maximum de garanties et de flexibilité au développeur d'application. Elle est compatible avec les interfaces actuelles de X/Open. Ce travail a été mené en cinq mois et n'est que partiellement réalisé: Les services sont spécifiés en C, les procédures OSI TP sont disponibles sous forme de librairie, et en partie testées. Ce travail doit être considéré comme un prototype et non comme un produit. En particulier, il doit être complété par les outils génériques de sécurisation des données synchrones entre le gestionnaire de données et le moniteur transactionnel.

GLOSSAIRE

AP	Application
DIAL_htr	Dialogue hors transaction
DIAL_tr	Dialogue en transaction
FIFO	First In First Out
GD	Gestionnaire de données (RM)
OSI TP	Open System Interconnection Transaction Processing
ODIS-X	Environnement de Développement d'Applications Interactives et Réparties à Interface Graphique INFOSERV
ODIS-EX	ODIS-X Extented
PDU	Protocol Data Unit
RM	Ressource Manager
RM-COM	RM communication
RM-MC	RM Multicom
RM_htr	RM hors transaction
RM_tr	RM en transaction
RPC	Remote Procedure Call
SD	Serveur de Données
TCP/IP	Transmission Control Protocol/ Internet Protocol
TM	Transaction Manager
TM_tos	TM TOSI (extension du TM)
TOSI	Transactionnel OSI
TPPM	Transaction Processing Protocol Machine
TPSUI	Transaction Processing Service User Invocation
Xid	UniqueTransaction Identifier
X/Open DTP	X/Open Distributed Transaction Processing

BIBLIOGRAPHIE

A. CARISTAN, P. LE PUIL, "Environnement de Développement d'Application Intéractives et Réparties a Interface Graphique Infoserv", Journées UNIX, Grenoble, 28/28 Octobre 1989.

P.A. BERSTEIN - V. HADZILACOS - N. GOODMAN, " Concurrency Control and Recovery in Database Systems", Addison-Wesley Publishing Company.

ISO/CEI-IS 10026 Distributed Transaction Processing (OSI TP) Modèle, Service, Protocole.

ISO/CEI-IS-7498 ISO Modèle de Référence.

E.JARRY, "Les Applications X/Open et OSI TP", Journées AFNOR Traitement Transactionnel réparti de Février 92

Journées OSI TP à l'AFNOR. NORMATIQUE,36, Avril 1992.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique

ISSN 0249 - 6399