N° d'Ordre : 06/90 - M/IN

THESE

PRESENTEE

A L'UNIVERSITE DES SCIENCES ET DE LA
TECHNOLOGIE HOUARI BOUMEDIENNE

pour l'obtention du

MAGISTER

Specialite : INFORMATIQUE

par :
Fazia BETOUCHE

CONCEPTION, SPECIFICATION FORMELLE ET REALISATION D'UN SERVICE D'APPEL DE PROCEDURE A DISTANCE DANS UN RESEAU HETEROGENE.

Soutenue le **03** Octobre 1990, devant **la** commission d'examen :

Mr A. AINOUCHE ..... Président
Mr C.B. BENYELLES .....Rapporteur
Mr N. BADACHE .....Examinateur
Mr M. BENHAMADI .....Examinateur
Mr A. BOULARAS .....Examinateur
Mr G. JUANOLE ..... Examinatem

Conception, spécification formelle et réalisation d'un service d'appel de procédure à distance dans un réseau hétérogène

KEMERCIEMENTS

Je remercie le Professeur A. Ainouche, directeur du centre de calcul pour l'honneur qu'il me fait en présidant ce jury de thèse.

Je suis fort reconnaissante envers:

- Monsieur N. Badache, Charge de cours A l'USTHB,
- Monsieur M. Benhamadi, Directeur du CERIST,
- Monsieur A. Boularas, Maitre de conference à l'USTHB,
- Monsieur G. Juanole, Professeur A l'université Paul Sabatier (Toulousej,

pour l'honneur qu'ils me font en participant à la commission d'examen.

Mes remercieinents et ma gratitude vont au Professeur C.8. Ben-Yelles pour l'interêt qu'il a porté A aon travail, son aide et ses critiques constructives.

Je remercie tout particulièrement Monsieur M. Benhamadi pour son acceuil ail CERIST et Monsieur N. Eadache pour son aide constante et sa disponibilité.

Je tiens à exprimer toute ma reconnaissance au professeur G. Juanole qui m'a acceullie au LAAS (Touiouse) et dont les critiques et conseils ont influé sur non travail.

Enfin, je tiens à exprimer ma profonde gratitude A Monsieur. M. Ayoub pour l'aide précieuse qu'il m'a apportée, A mes parents et plus particulièrement à ma soeur Mademoiaelle N. Betouche pour ses encouragements et sa presence constante à mes cotés, à la sympatique équipe de la direction procédé de l'ENEP et à tous mes amis.

Pour pallier à ces problèmes, il est nécessaire d'adopter une méthodologie de conception qui permettra de specifier de façon formelle une application répartie. Cette description formelle servira à son tour, de Lase pour la verification du comportement de cette **application** et cela avant son implantation.

De plus, il est nécessaire d'assurer aux programmeurs d'applications réparties, un certain degré **de** transparence à la répartition. Ceci peut être réalisé par i'utilisation d'un procédé bien connu dans les syatemes centralisés à savoir i'appel de procédure. Le service doit être &ten?;;de manière à offrir un niveau d'abstraction élevé du système réparti tel que celui-ci soit vu conime local. Les différents problèmes inhérents A la répartition seront gérés par le service lui même et **ne** serunt donç **pas** pris en compte par les programmeurs.

Le but du travail **expose** dans ce document est la conception, la spécification formelle et la réalisation d'un service d'appel de procedure à distance dans un réseau hétérogène.

Le document s'oryanise comme suit:

Danv **ie** premier chapitre, **apses** un exposé des différents problèmes inhérents à la répartition, nous décrivons la conceptiun d'un service de contLôie d'applicat.ions réparties, basé sur l'appel de procédure à distance. L'abstraction **de** ce **service** consiste **en** la mise en oeuvre de procédures de contrôle appelées stubs qui sont invoquées lors des appels à distance. **Ces** procedures sont produites par un générateur de stubs **que** nous décrivons dans ce chapitre. A partir des différents problèmes liés à l'exécution d'une application répartie, nous déduisons une architecture en couches qui permet la décomposition d'une application en sous fonctions plus faciles à appréhender.

Le deuxième chapitre a pour but ia vérification du service de contrôle **d'appel de** procédure à distance et du protocole sous-jacent. La verification nhcessite d'abord **la** representation du comportement des algorithmes dans un formalisme pouvant supporter des techniques de preuves **formelle**. Nous avons utilisé pour cette vérification l'outil PIPN développé au L.A.A.S. Ce système est base sur l'interprétation en Prolog des cornportement décrits en réseaux de Petri étiquetés.

Le troisième chapitre est consacré à l'implantation du service **d'appel** de procedure à distance, à titre expérimental dans un réseau homogène.

CHAPITRE I

Généralités sur les applications réparties et l'appel de procédure à distance

La communication dans les systèmes répartis est caractérisée par les transferts d'information et de contrôle entre des programmes autonomes s'exécutant sur des machines distinctes.

Les primitives de communication par messages sont l'outil de base pour la communication dans ces systèmes et peuvent donc être utilisées directement pour décrire l'organisation des activités réparties. Cependant, la répartition géographique du traitement et la nature du matériel connecté peuvent engendrer des situations exceptionnelles (exemples: panne d'un site, perte, duplication ou déséquencement de messages) qui ne sont pas prévues par les primitives de communication et qui doivent être prises en compte par les programmeurs.

Par conséquent, on considère souvent que les primitives de communication par messages fournissent un moyen d'expression de bas niveau, pouvant conduire à des structures d'exécution dont on ne peut contrôler la complexité. Un système d'exploitation réparti doit donc offrir des mécanismes de contrôle rendant transparents aux programmeurs d'applications réparties les mauvais fonctionnements du système survenant lors de l'exécution de ces applications.

Nous nous proposons dans ce chapitre, d'utiliser les messages pour concevoir un mécanisme de contrôle de traitements répartis, basé sur l'appel de procédure. La détermination des différentes fonctions nécessaires au contrôle des traitements répartis, nous permettra d'aboutir à une architecture fonctionnelle d'un service de contrôle d'appel de procédure à distance.

I.1 Généralités sur les traitements répartis

I.1.1 Définitions

Une application répartie est réalisée à l'aide d'un ensemble d'objets communicants appelés processus ou entités de traitement, s'exécutant sur des sites distincts et utilisant un protocole de coopération spécifique à l'application. Chaque entité est caractérisée par ses données locales, son traitement local et les échanges éventuels d'information avec les autres entités.

Les processeurs et les mémoires qui constituent ces sites sont sujets à des pannes. De même, le système de communication peut subir des défaillances (perte, altération, déséquencement, duplication des messages). Une défaillance quelconque provoque une terminaison anormale de l'application répartie, pouvant entraîner l'incohérence de l'état global des données. Par conséquent, la mise en oeuvre de mécanismes de traitement d'erreurs et de reprise s'avère nécessaire pour garantir la sûreté de fonctionnement et la terminaison normale d'un traitement réparti.

I.1.2 Détection des défaillances

Il existe deux types de défaillances [Mon 87, Nel 81]:

a. Défaillances de la communication

Elles sont prévisibles lors de la définition de l'application et donc prises en compte. Parmi ces situations nous citons le déséquencement, la perte et la duplication des messages, détectables par des mécanismes de numérotation, d'acquittement et de retransmission temporisée ou la corruption de l'information détectable au moyen de codes détecteurs introduisant une information supplémentaire (un exemple simple est celui du bit de parité).

b. Défaillances du matériel connecté

Elles ne sont pas directement liées au traitement mais à son environnement. Ce type regroupe toutes les situations exceptionnelles liées à la répartition de la communication ou la panne d'un site connecté. Dans la plupart des systèmes, le mécanisme de base utilisé pour détecter ce genre de défaillances est l'acquittement: l'absence d'acquittement au bout d'un certain temps est interprété comme un signal d'exception. Ce type d'erreurs peut entraîner une interruption de la communication (cas de la coupure physique de la ligne de communication) et/ou la perte des données résultant d'un traitement local (en cas de panne de site), engendrant ainsi, un état incohérent du système. Il est donc nécessaire de tenir compte de ces situations et de mettre en oeuvre un mécanisme de reprise après panne afin de garantir l'atomicité du système.

I.1.3 La cohérence

L'exécution d'une application répartie met en jeu les valeurs de l'ensemble des données des différentes entités participant au traitement. Ces valeurs sont liées par un ensemble de relations, appelées contraintes d'intégrité ou de cohérence qui expriment les spécifications du système.

Une des caractéristiques importantes au niveau des applications réparties est la cohérence du traitement. Le maintien de la cohérence de l'état global des données utilisées dans une application répartie caractérise un traitement atomique.

Une action atomique est une opération qui fait passer, en un temps fini, un ensemble d'objets d'un état initial à un état final et qui est caractérisée par les deux propriétés suivantes:

- la totalité: l'état du système après exécution d'un traitement est soit celui qui résulte de l'exécution complète de ce traitement, soit identique à l'état avant exécution du traitement, c'est-à-dire qu'en cas de défaillance au cours de l'exécution de l'application répartie, l'ensemble des données participant à ce traitement retrouve son état initial.

- la sérialisabilité: l'ensemble des objets participant au traitement est inaccessible aux autres applications (propriété d'indivisibilité). Autrement dit les accès concurrents à des objets partagés doivent être équivalents à des accès séquentiels ou sérialisés.

Un état global des données qui satisfait l'ensemble des contraintes d'intégrité est donc défini comme un état global cohérent. Cet état global est obtenu lors de la terminaison globale du traitement réparti, c'est-à-dire lorsque tous les processus y participant ont effectué normalement leurs traitements locaux et signalé leurs terminaisons locales.

I.1.4 Le mécanisme de reprise après panne

Le mécanisme de base de reprise est la mémorisation sûre de l'état de chaque processus de traitement, sur un support dont le contenu n'est pas affecté par une défaillance (mémoire stable). Ce mécanisme dit à mémorisation sûre, peut être réalisé par utilisation de versions dupliquées sur disque, des circuits de mémoires redondants ou auto-alimentés [Kra 87], [Mon 87], etc..

Pour garantir la cohérence de l'état sauvegardé de cette façon, il est nécessaire de disposer d'une primitive d'écriture atomique sur ce support.

Afin de pallier à toute défaillance, toute entité interagissant avec un processus distant doit mémoriser de façon sûre son état et garantir que lors de la reprise après défaillance, il sera restauré en l'état précédent la défaillance. La technique permettant l'exécution cohérente d'une application répartie en présence de défaillances repose sur le principe de la validation appelé aussi protocole d'engagement à deux phases. Il permet:

1- la sauvegarde des versions finales des données lors d'une terminaison globale normale. Cette partie du protocole est appelée phase d'engagement.

2- l'annulation du traitement local effectué, la restauration et la sauvegarde des versions initiales des données dans le cas d'une terminaison globale anormale. On dira qu'une terminaison globale d'une application répartie est anormale si au

moins un processus participant au traitement a effectué une terminaison locale anormale. Cette phase est appelée phase d'annulation ou de restauration.

I.2 Contrôle d'un traitement réparti

Les différents sites constituant un système réparti étant faiblement couplés (absence d'une horloge physique globale), les entités s'exécutent de façon asynchrone. Il est donc indispensable pour les entités participant à une application répartie d'échanger des informations sur leurs états locaux afin de déterminer le type de terminaison (normale ou anormale) de l'application répartie. Ces échanges et la détermination du type de terminaison d'une application répartie seront réalisés par une entité de contrôle distincte de l'entité de traitement. Sur chaque site du système réparti s'exécute une instance de cette entité.

Une entité de contrôle doit permettre aux entités de traitement de coordonner leurs comportements et leurs échanges afin d'assurer la cohérence du traitement et des données résultantes. En outre, elle doit être capable de détecter et éventuellement traiter les situations exceptionnelles (telles que panne de site, erreurs de communication, etc.). Les principales fonctions d'une entité de contrôle sont donc les suivantes:

- gérer et coordonner les échanges entre instances de traitement s'exécutant de façon asynchrone sur les différents sites,
- résoudre les problèmes de conflits résultants de l'échange simultané d'informations contradictoires entre entités de traitement,
- détecter et traiter les erreurs de communication et les pannes de sites et maintenir les données du système dans un état cohérent,
- gérer les relations entre entités de traitement. On distingue deux classes de relation entre les entités [Mon 87], [Mak 89]:

Relations simples

- deux entités données échangent simultanément des informations de même type (relation d'égal à égal). Ceci peut conduire à un conflit si les deux entités initient des informations contradictoires.

- une entité locale interagit avec une entité distante, l'entité locale étant l'initiatrice de l'activation. Ce type de relation sera appelé dans ce document relation 1 à 1.

Relations multiples

- une entité locale interagit avec un ensemble d'entités distantes, l'entité locale étant l'initiatrice des différentes activations (relation 1 à N).

- plusieurs entités distantes interagissent avec une entité locale (relation N à 1). Dans ce cas, l'entité locale peut recevoir des informations contradictoires de la part des entités distantes.

Si le programmeur d'une application répartie devait mettre en oeuvre les différentes fonctions d'une entité de contrôle, sa tâche serait fastidieuse, voire même impossible. Pour lui éviter ces contraintes, une entité participant à un traitement réparti peut communiquer avec une entité distante participant au même traitement en accédant à l'instance de l'entité de contrôle s'exécutant sur son site. Le moyen le plus simple d'accéder à cette instance est d'utiliser un procédé bien connu, ayant fait ses preuves dans les systèmes centralisés, à savoir l'appel de procédure. Ce mécanisme permet une exécution simple, tout en assurant un niveau d'abstraction très élevé et une sûreté de fonctionnement non négligeable.

De cette manière, lorsqu'une entité de traitement désire communiquer avec une autre entité distante, il lui suffit d'invoquer l'entité de contrôle locale qui se charge de localiser le site distant et de transférer la requête à l'entité de contrôle de ce site. Cette dernière invoque localement l'entité de traitement concernée par la requête. Par conséquent, les entités de traitement ignorent qu'elles coopèrent à l'exécution d'un traitement réparti. Seules les entités de contrôle le savent.

I.3 Appel de procédure à distance

I.3.1 Definition

L'appel de procédure à distance est une généralisation de l'appel de procédure ordinaire au cas où l'appelant et l'appelé s'exécutent sur des sites distincts. Ce mécanisme a le grand avantage de fournir un mode de structuration uniforme pour les activités locales et distantes. L'appel de procédure à distance met en jeu deux processus appelant et appelé, un même processus pouvant jouer les deux rôles, et deux entités de contrôle, l'une du coté de l'appelant et l'autre du coté l'appelé. Ces entités assurent la gestion des contextes distants de façon à préserver leur cohérence mutuelle en cas de défaillances.

Les processus appelant et appelé fonctionnent en mode client/serveur. Ce mode d'interaction possède les caractéristiques suivantes [Svo 85, Mon 87, Nra 87]:

- toutes les communications ont lieu entre un client et un serveur, l'activation étant transmise par le client au serveur.
- un serveur est défini par son interface qui spécifie les services qu'il fournit et leur mode d'utilisation. Le serveur possède ses propres données locales (serveur non idempotent) qui sont sauvegardées après terminaison de chaque appel.
- un signal de terminaison est transmis par le serveur au client.

L'abstraction de haut niveau à fournir à un client, consiste en l'invocation d'un service qui sera réalisé par un ou plusieurs serveurs situés sur des sites distants, imbriqués ou activés en parallèle.

Un appel de procédure à distance fait apparaître trois types d'entités de traitement:

- entité client: entité de traitement qui active le service pour son compte,
- entité serveur: entité de traitement qui est activée par le client et qui n'invoque aucun service,
- entité client-serveur: l'entité de traitement activée et qui doit à son tour invoquer un service.

Il faut remarquer que dans tous les cas, les résultats des différentes activations sont contrôlés par l'entité client. Le client doit donc avoir une vision globale du service.

I.3.2 Problèmes liés à la conception d'un service d'appel de procédure à distance

Un service de contrôle assurant les différents types d'interaction et possédant les propriété d'un appel de procédure locale doit tenir compte des différents problèmes inhérents à la répartition, parmi lesquels nous citons:

1. L'existence indépendante des processus appelant et appelé nécessite un mécanisme d'établissement du lien logique, permettant une fois ce lien établi, de réaliser l'invocation distante.

2. Une procédure appelée à distance peut être composée de deux parties entremêlées [Mon *873*:
   - une partie traitement local.
   - une partie traitement distant.

Du point de vue de l'appelant, la procédure est indivisible alors que de façon interne, il est possible d'atteindre des états intermédiaires incohérents. On doit donc garantir une terminaison non ambiguë du programme en dépit des pannes des noeuds ou des erreurs de communication qui peuvent survenir durant l'exécution.

3. Pour respecter un mode d'interaction procédural (appel/réponse) et pour coordonner l'échange de données entre un client et un serveur, la fonction de synchronisation est nécessaire.

4. Le processus appelant et appelé peuvent fort bien se trouver sur des machines ne disposant pas des mêmes représentations internes des données et être écrits dans des langages n'appréhendant pas les mêmes types de données, il est donc nécessaire de fournir un mécanisme de codage et de décodage des données, tant pour des raisons de transfert de données entre

machines hétérogènes que pour des problemes de compatibilité de types de données entre langages de programmation différents [Dia 85].

5. Tout appel de procedure à distance se traduit à un moment donne par une transmission de messages. Une fonction de communication est donc ndcessaire pour assurer les transferts reels des donndes entre les sites. Cette fonction sera réalisée par un service de communicatior ;ui sera accessible au service de contrôle par l'intermédiaire d< ,rimitives de service.

Un service d'appel de procddure à distance serait identique à un service d'appel local si ces différentes fonctions dtaient offertes de fapon transparente aux programmeurs d'applications rdparties.

1.3.3 Fonctionnement d'un appel de procddure à distance

I.3.3.1 Comportement normal

Le comportement normal (absence de pannes et d'erreurs de communication) d'un appel de procddure à distance peut être décrit de la manière suivante: le client (programme appelant) fait un appel local normal sur son site avec l'intention d'invoquer une procddure sur un site distant. Cet appel est intercepté par l'entité de contrôle du client qui localise le site de l'appelée, réalise une connexion loqique avec ce site, forme le message d'appel contenant, entre autres, les paramhtres d'appel puis envoie ce message au site de l'appelée et se met en attente. Ce message sera repu par l'entité de contrôle du serveur qui extrait les paramhtres, puis invoque localement la procedure concernée par l'appel distant.

Au retour de la procédure, l'entitd de contrble du serveur forme le message retour contenant les résultats puis le transmet au site appelant. L'entité de contrble du client reqoit le message retour, récupère les résultats puis effectue un retour local au programme appelant (fig. I-1).

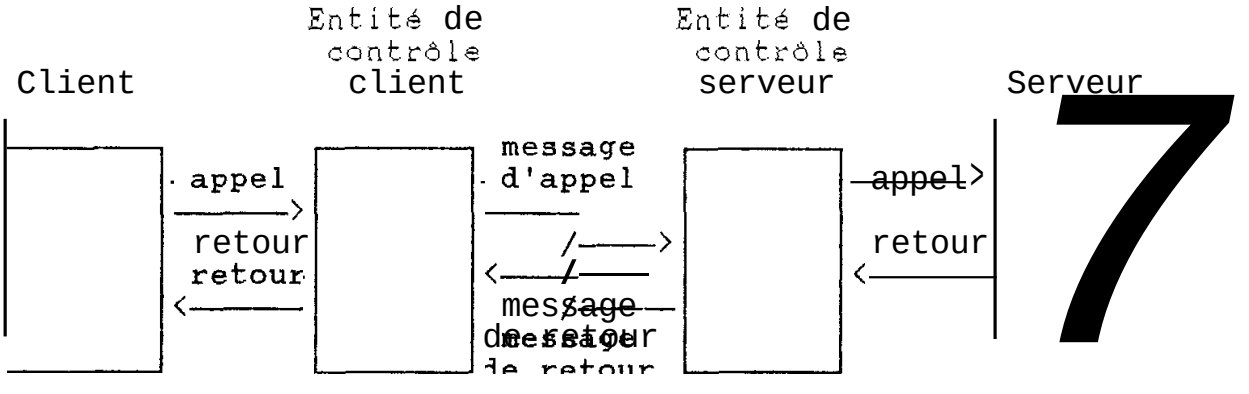

fig. I-1 Sequences d'bvbnements lors d'un appel de procddure a distance

I.3.3.2 Comportement anormal

Tout appel de procédure à distance est implanté à l'aide de primitives d'émission et de réception de messages [Sta 82]. Le canal véhiculant ces informations peut ne pas être sûr. Les messages peuvent donc occasionnellement être altérés ou se perdre. De plus, le site client peut tomber en panne immédiatement après que le serveur ait commencé l'exécution de l'appel et initié, éventuellement d'autres exécutions, sur son site ou sur des sites distants (cas d'appels imbriqués). La réémission du message d'appel provoquerait plus d'une exécution du serveur, pouvant entraîner l'incohérence de l'état du système. Par conséquent, il faut pouvoir détruire les processus (dits orphelins) résultants d'appels non terminés, éviter de réexécuter plus d'une fois la procédure appelée en cas de perte et réémission par l'appelant et enfin si l'exécution est atomique, il faut pouvoir restaurer l'état précédent les défaillances.

La spécification précise et la réalisation de l'appel de procédure à distance doivent être examinées avec soin pour prendre en compte les risques de terminaisons anormales dues à la défaillance du système de communication ou de l'un des sites. Selon les propriétés d'exécution, des mécanismes de contrôle de la communication de données sûres et de reprise après erreur doivent être mis au point. B.j.Nelson [Nel 81] a établi la classification de ces propriétés comme suit:

1. propriété "au moins une fois"

L'appelant est certain d'obtenir des résultats de l'exécution demandée. Pour cela, il doit retransmettre périodiquement le message d'appel jusqu'à la réception d'un message réponse, au prix d'exécutions répétées de la même procédure. Ceci peut se produire en cas de perte ou de retard des messages. Pour permettre au client de réémettre le message d'appel, il faut disposer d'un algorithme redemandant l'exécution distante en cas de retard, jusqu'à la réception du premier message retour, éliminant ainsi, les messages retour suivants. Cet algorithme garantit, l'exécution de la procédure au moins une fois mais l'appelant ne sait pas à quelle exécution correspondent les résultats reçus. Il se pose alors un problème quant à la cohérence des données locales de la procédure appelée.

2. propriété "la dernière fois"

Une variante intéressante de l'algorithme précédent est un algorithme qui permet de sélectionner le dernier message retour. Pour garantir qu'il s'agit bien du dernier message retour, l'algorithme doit inclure un numéro de série dans chaque message d'appel. De plus, à chaque message retour est affecté le numéro de série du message d'appel correspondant. La comparaison du numéro du dernier message d'appel et celui d'un message retour permet de rejeter les résultats des messages d'appel antérieurs

et de garder ceux du dernier. Dans ce cas, les données locales de la procédure appelée sont correctes vis à vis des résultats reçus.

3. propriété "au plus une fois"

La procédure appelée doit s'exécuter une fois ou pas du tout. L'algorithme doit donc etre pourvu de mécanismes détectant des appels dupliqués et assurant l'atomicité de plusieurs procédures éventuellement imbriquées. Dans ce cas un appelant peut recevoir une réponse négative à son appel.

4. propriété "exactement une fois"

La procédure appelée s'exécute exactement une fois et contrairement au cas précédent, l'appelant ne peut pas recevoir une réponse négative à son appel. Il exige une terminaison correcte de l'appelé. Cela nécessite donc un algorithme de reprise après panne.

Le choix de la propriété d'exécution d'un appel dépend des caractéristiques des procédures et du degré de fiabilité du réseau. Ainsi, dans le cas de procédures idempotentes (c'est-à-dire sans variables locales), les propriétés "au moins une fois" et "la dernière fois" sont équivalentes du point de vue des résultats reçus par l'appelant. L'atomicité globale s'obtient simplement car toute procédure se termine, sans que se pose la question de défaire ce qui a été fait en cas de panne. Cependant, dans tous les cas, il faut mettre en oeuvre un algorithme d'élimination des procédures en cours d'exécution et dont les initiateurs ont disparus (orphelins).

I.3.4 Interactions entre les entités de traitement et le service de contrôle

Dans le cas de cette étude, le service de contrôle prend en compte les appels simples (1 à 1) et imbriqués. Nous distinguons alors les trois types d'interaction suivants:

1. entité client/service

Les interactions client/service sont illustrées par la figure suivante (fig. I-2):

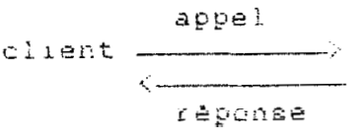

fig. 1-2 Appel et Réponse

Nous distinguons deux types de réponse:

> réponse positive: lorsque le ou tous les serveurs (cas d'appels imbriqués) ont été engagés, c'est-à-dire ont terminé normalement leur exécution.

> Réponse négative: lorsque l'appel a été annulé par le serveur (ou l'un des serveurs dans le cas d'appel imbriqués; en cas d'erreurs dans le message d'appel ou par le service lui même si un temps limite d'attente a été fixé au préalable ou s'il détecte une défaillance quelconque (panne du site distant ou de la ligne de communication).

2. service/entité serveur

Après invocation du serveur par le service (stub serveur), deux cas peuvent se présenter:

> une réponse initiée par le serveur (fig. I-3),

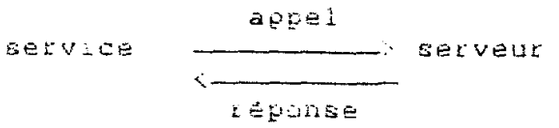

fig. I-3 Appel et Réponse

> une annulation initiée par le service (fig. I-4): le serveur est annulé par le service si lui même reçoit une annulation de l'entité service de contrôle distante ou s'il détecte une défaillance de la ligne de communication ou la panne du site appelant.

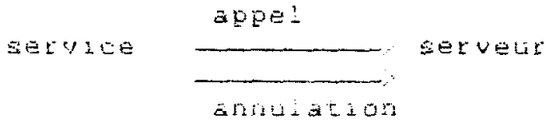

fig. I-4 Appel et Annulation

3. service/entité serveur-client

L'entité serveur-client est d'abord activée par le service de contrôle et peut:

- répondre (fig. I-5),
- activer le service (fig. I-6),
- être annulée (fig. I-7).

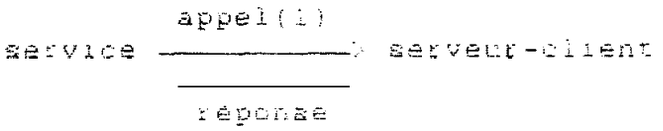

fig. I-5 Appel et Réponse

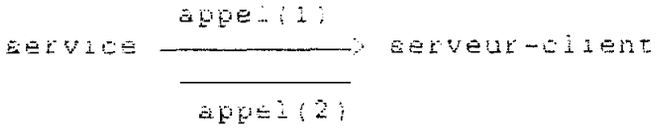

fig. I-6 Appels imbriqués

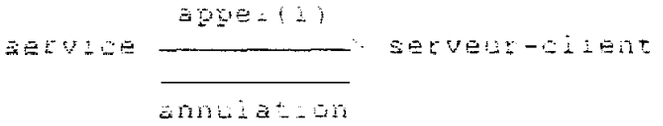

fig. I-7 Appel et Annulation

Remarque :

Après réception d'un message réponse, le service de contrôle client peut annuler ou confirmer (engager) un appel. La réception d'un engagement se traduit au niveau du service de contrôle serveur par la mise à jour des données du serveur. Dans le cas d'une entité serveur-client, l'engagement est soumis à la décision du client initial.

I.3.5 Conception du service d'appel de procédure à distance

Après l'étude de quelques systèmes d'appel de procédure à distance existants [Nel 81, Shr 82, Bir 84, Sat 85, Mon 87], nous avons conçu un service d'appel de procédure à distance qui reprend certains avantages de ces systèmes et apporte des solutions à leurs inconvénients.

Lors de l'exécution d'un appel de procédure à distance, nous distinguons deux étapes:

- la localisation de la procédure appelée afin d'établir le lien logique entre le site appelant et le site appelé,

- le contrôle de l'appel qui consiste en la formation et l'envoi du message d'appel, l'attente et la réception du message retour ainsi que la prise en compte des défaillances.

Nous allons dans ce qui suit, décrire la façon dont ces problèmes ont été appréhendés dans les systèmes existants et les solutions que nous proposons.

I.3.5.1 Etablissement du lien logique

L'invocation d'une procédure à distance nécessite la connaissance de son nom. Cependant, le nom de la procédure n'est pas suffisant pour établir un lien logique entre le site appelant et le site appelé. Il faut en outre connaître l'adresse du site appelé.

La désignation et la localisation d'un processus distant peuvent être réalisées de façon statique (arrangement au préalable entre les processus sur les noms et les adresses) ou de façon dynamique par interrogation d'un système de gestion d'objets répartis dans un réseau (Birrell et Nelson utilisent le système Grapevine [Bir 84])

1. Désignation

Un aspect important des systèmes répartis est la façon dont ils permettent aux entités communicantes de se désigner mutuellement. On appelle désignation la fonction d'identification d'un objet à partir d'une information appelée identificateur [Leg 88a, Leg 88b].

Dans le cas du service d'appel de procédure à distance conçu, chaque procédure (serveur) est désignée par un nom symbolique ne contenant aucune information concernant l'adresse de son site de résidence. Ce qui assure la transparence à la localisation. En outre, le nom d'une procédure pouvant être invoquée à distance est unique dans le système. En effet, si deux procédures offrant des services différents portaient le même nom, il se poserait un problème de conflit d'accès lors de l'appel de l'une ou de l'autre. L'appelant risque alors d'obtenir un service ne correspondant pas à sa demande.

Ce cas peut se présenter si les deux procédures possèdent la même description, c'est-à-dire le même nombre de paramètres et de mêmes types.

2. Localisation

La localisation du processus concerné par un appel distant se fera de manière dynamique (l'appelant n'a pas besoin de connaître l'adresse du site appelé). En l'absence de système de gestion d'objets répartis (base de données répartie par exemple), nous avons mis en oeuvre un catalogue (répertoire) dupliqué sur chaque site. Le catalogue contient pour chaque procédure pouvant être invoquée à distance, son nom symbolique, l'adresse réseau du site de résidence qui est unique ainsi que d'autres informations telles que l'état qui est égal à actif si le site de résidence de la procédure est actif, à absent si le site est en panne et à absent-temporairement si la procédure a été supprimée temporairement. Dans le cas où cette procédure est instanciée sur plusieurs sites, le répertoire contient en plus, le nombre de sites, l'adresse et l'état de chacun d'eux (fig. I-8).

nom de la procédure	n	adresse site1	état1	...	adresse siten	étatn	autres informations

fig. I-8 Structure du catalogue

Lorsqu'un utilisateur désire rendre sa procédure accessible aux autres sites, il diffuse un message contenant le nom symbolique ainsi que l'adresse du site de résidence. On parlera alors d'**exportation de la procédure.**

Lorsque le message d'exportation arrive sur un site donné, le catalogue local est mis à jour.

L'inconvénient majeur de la méthode réside dans la difficulté d'assurer une mise à jour cohérente de toutes les copies du catalogue. Exemple, si au même instant, deux utilisateurs travaillant sur le même site ou sur des sites distincts désirent exporter une procédure de même nom, une simple consultation des catalogues locaux de chacun des sites n'est pas suffisante pour accepter la demande de l'un ou de l'autre. En effet, les deux sites doivent coopérer dans la décision d'accorder le droit d'exportation de la procédure à l'un d'eux si la procédure n'existait pas déjà. Un algorithme de maintien de la cohérence doit donc être mis en oeuvre pour éviter les cas de conflit d'accès et d'incohérence des différentes copies du catalogue.

I.3.5.2 Contrôle de l'appel de procédure à distance

La plupart des systèmes d'appel de procédure à distance étudiés ont été réalisés sur des réseaux homogènes et ne permettent qu'un seul langage bien spécifique entre le client et le serveur. Parmi ces systèmes, nous citons: le système Eden [Alm 85] qui utilise le langage de programmation Eden, une extension d'Euclid concurrent, le système Argus de Barbara Lismov [Lis 83] qui exige des programmeurs l'utilisation du langage Argus, une extension de Clu, et l'écriture de leurs propres routines de codage et de décodage pour les types composés, et enfin le système Emissary, partiellement implanté par Nelson [Nel 81], qui utilise le langage Mesa.

Dans chacun de ces systèmes, le client et le serveur doivent être écrits dans le même langage et résider sur le même type de machine. Les compilateurs doivent être alors modifiés de telle sorte qu'ils puissent distinguer un appel local d'un appel distant et émettre un code pour représenter les données dans les messages. Pour cette raison, ces systèmes ne peuvent pas être utilisés dans des environnements hétérogènes [Gib 87].

D'autres systèmes fournissent aux utilisateurs un service de contrôle qui est invoqué lors d'un appel à distance et qui réalise les différentes fonctions nécessaires à l'exécution d'une application répartie [Bir 84, Mon 87, Gib 87, Sun 87].

La première solution a été écartée en raison des inconvénients qu'elle présente tant du point de vue implantation (indisponibilité du source des compilateurs) que du point de vue intégration dans des systèmes hétérogènes. La deuxième solution pourrait être, à son tour implantée de deux manières différentes:

1ère méthode

Il existe une seule procédure de contrôle qui gère les appels distants sur chaque site. Donc sur un site donné, invoquer une procédure distante revient à appeler la procédure de contrôle en lui donnant le nom de la procédure concernée par l'appel.

Exemple: soit **APD$CONT** le nom de la procédure de contrôle, l'invocation à distance d'une procédure Proc se fera de la manière suivante: **APD$CONT(Proc).**

Si de plus la procédure Proc possède des paramètres d'appel et de retour, l'appel à la procédure de contrôle se fera comme suit: **APD$CONT(Proc,** tpa_1**,**par_1**,..**tpa_1**,**par_1**,...,**tap_n**,**par_n
tpr_1**,**res_1**,...,**tpr_1**,**res_1**,...,**tpr_n**,**res_n**);**
où tpa_i et tpr_i sont respectivement les types des paramètres d'appel et de retour, par_i sont les paramètres d'appel et res_i sont les paramètres de retour.

Il est indispensable d'indiquer les types des paramètres d'appel et de retour car ils nous renseignent, entre autres, sur la manière de représenter les paramètres et les résultats dans les messages et servent également au calcul de la taille de ces derniers.

Le service d'appel de procédure à distance developpé pour les stations de tarvail SUN [Sun 86] qui est basé sur cette méthode utilise en outre, une bibliothèque contenant un ensemble de routines pour coder et décoder les paramètres d'appel et les résultats. Ce service d'invocation à distance peut être utilisé pour des environnements d'appel hétérogènes. Cependant, les programmes clients doivent explicitement appeler l'une de ces routines pour chaque argument dans chaque appel distant. De plus, les programmeurs doivent construire leur propre routines de codage et de décodage pour des types de données complexes.

L'inconvénient de cette méthode réside dans le fait que les procédures pouvant être invoquées à distance n'ont pas toutes le même nombre de paramètres. Or le nombre de paramètres de la procédure de controle est fixe. Deux solutions sont alors possibles:

1. concevoir une procédure de contrôle par groupe de procédures ayant le même nombre de paramètres. Ne connaissant pas, à priori, le plus grand nombre de paramètres qui peut exister dans une procédure, on ne peut donc pas savoir combien de procédures de contrôle il faut concevoir.

2. limiter le nombre de paramètres de la procédure de contrôle à deux variables qui sont des pointeurs vers des listes. Chaque liste possède deux champs : un champ valeur et un champ type de la valeur. La première liste sera réservée aux paramètres d'appel et la deuxième aux paramètres de retour.

Exemple: APDSCONT(LA,LR), où LA est le pointeur vers la liste des paramètres d'appel et LR le pointeur vers la liste des paramètres de retour.

Les inconvénients de cette méthode sont les suivants:

1. dans le cas où les paramètres d'appel ou de retour sont de types structurés, la structure des listes serait plus complexe.

2. l'utilisateur serait dans l'obligation de construire avant tout appel à une procédure distante, la liste LA et d'extraire au retour, les résultats de la liste LR. Cela nécessiterait donc une connaissance approfondie de la manipulation des listes et de leurs structures. Le degré de transparence offert aux utilisateurs n'est donc pas assez élevé et de ce fait la syntaxe d'un appel local et celle d'un appel distant différent.

2ème méthode

Cette méthode consiste à mettre en oeuvre deux procédures de contrôle, appelées stubs [Nel 81] ou bouts [Kra 87], l'une du coté client et l'autre du coté serveur et cela pour chaque procédure pouvant être invoquée à distance. Pour invoquer une procédure à distance, le client appelle donc le stub client qui porte le même nom que la procédure distante et qui joue le rôle de représentant de cette procédure. Le stub serveur jouera, quant à lui, le rôle d'interface entre le site appelant et la procédure appelée. Les stubs déchargent le client et le serveur de toutes les contraintes inhérentes à la répartition, permettant ainsi de garder la même syntaxe quelque soit le type d'appel (local ou distant).

Pour éviter aux utilisateurs l'écriture manuelle des stubs, un générateur de stubs est nécessaire. Ce dernier reçoit en entrée une description de la procédure ou déclaration d'interface et donne en sortie le stub client ou le stub serveur (fig. I-3).

La déclaration d'interface contient:
- le nom de la procédure,
- le nombre de paramètres de la procédure,
- l'identificateur, le type et le mode de passage de chacun d'eux,
- le langage dans lequel le stub sera généré.

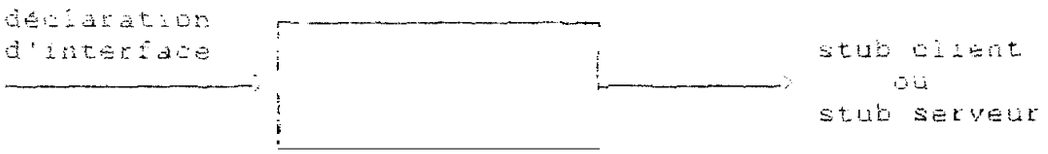

fig. I-9 Générateur de stubs

Dans un système distribué où il y'a L langages différents et M types de machine, un service d'appel de procédure à distance doit soit fixer au départ qui peut communiquer avec qui, soit prévoir L^M générateurs de stubs. De nouvelles routines de conversions doivent être implantées sur chaque site chaque fois qu'un nouveau langage ou un autre type de machine est rajouté au réseau.

Une solution à ce problème serait de mettre en oeuvre une représentation externe commune qui définit une représentation standard des différents types de données dans les messages. Ainsi, lors de l'appel à distance, les paramètres d'appel sont convertis de leur représentation dans l'environnement d'apoel du client à la représentation externe. Lorsque le message arrive au site serveur, ils seront convertis de la représentation externe à la représentation interne du serveur. Les résultats subiront le même traitement.

L'utilisation d'une représentation externe commune nécessite beaucoup de conversions des données. De plus, une représentation externe donnée peut être incapable de représenter efficacement les valeurs de certains types de données. Cependant, seulement deux routines de conversions doivent être écrites dans chaque environnement d'appel: l'une pour convertir à la représentation externe et l'autre pour convertir de la représentation externe. En outre, le stub est libéré de la détermination, au moment de l'exécution, de la représentation à utiliser. Enfin, de nouveaux langages et d'autres types de machine peuvent être rajoutés au réseau sans modification du système.

Les systèmes développés par Birrell et Nelson [Bir 84] et Black et al [Bla 85] utilisent un générateur de stubs. Cependant, les informations permettant de résoudre l'hétérogénéité des langages et des machines sont dépendantes du code du générateur de stubs. Cela rend difficile toute mise à jour et exige un générateur de stubs distinct pour chaque environnement d'appel, c'est-à-dire LxM générateurs si L est le nombre de langages différents et M le nombre de machines différentes.

Afin de résoudre ce problème, nous avons conçu un générateur de stubs indépendant des langages et des machines. Pour produire un stub client ou serveur, le générateur de stubs doit avoir en entrée, en plus de la déclaration d'interface, la spécification du langage dans lequel le stub sera généré et la spécification de la machine de résidence de ce stub (fig. I-10).

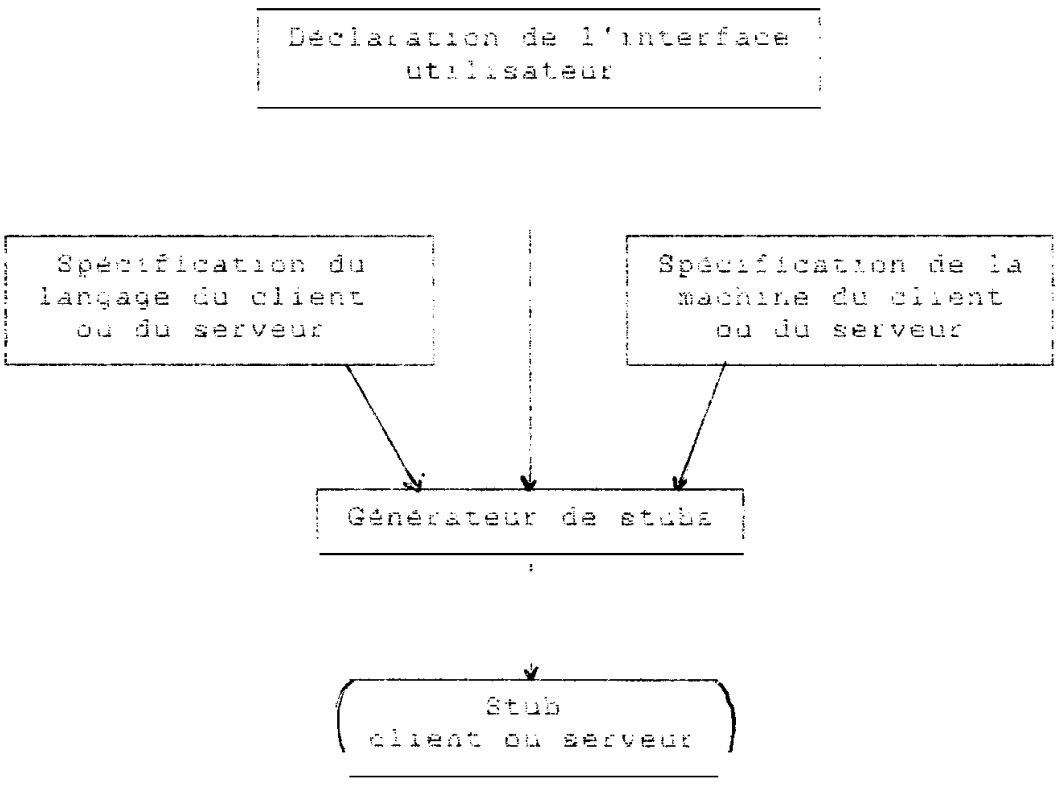

fig. I-10 Génération de stubs pour des environnements hétérogènes

La spécification d'un langage contient un ensemble de commandes interprétables par le générateur de stubs ainsi que des informations qui vont permettre de:

1. redéfinir les différents types du langage,
2. émettre l'entête de la procédure stub (client ou serveur) ainsi que les différentes parties qui constituent le stub, c'est-à-dire les informations pour:
   - déclarer les variables locales,
   - former les messages d'appel et retour,
   - allouer et désallouer les buffers,
   - envoyer et recevoir des messages.

La spécification de la machine contient des informations dépendantes de la machine, nécessaires pour la représentation des données dans les messages. Parmi ces informations, nous

distinguons pour chaque type de données, la taille en nombre de bits et le type de représentation (complément à deux, ascii, etc.).

Choix de la représentation externe

La représentation externe que nous avons choisi est de la forme type, longueur, valeur. C'est-à-dire que chaque argument d'appel ou de retour sera représenté dans le message par:

- le code du type (exemple: 1 pour integer, 2 pour real, etc.),
- la longueur de la représentation,
- la représentation de la valeur.

La valeur de l'argument peut être constitué d'autres éléments décrits de la même façon. C'est le cas des arguments de type enregistrement (record).

Génération des stubs client et serveur

L'appel de procédure à distance nécessite l'existence des stubs client et serveur respectivement sur les sites appelant et appelé. La question qui se pose alors est la suivante: à quel moment ces stubs doivent ils etre générés?

En ce qui concerne le stub serveur, il est généré après ou avant l'écriture du corps de la procédure. Ensuite, il suffit de réaliser l'edition de liens entre le stub serveur et la procédure elle même.

Quant à la génération du stub client d'une procédure, nous avons envisagé deux solutions possibles :

1. le stub client est généré, sur un site donné, à la réception d'un message d'exportation provenant d'un site distant; c'est-à-dire après le rajout d'une procédure dans le catalogue local. Le stub ainsi généré est ensuite inséré dans une bibliothèque que nous appellerons LIBsRPC. De cette façon un degré de transparence très élevé est assuré puisque, pour invoquer une procédure à distance, il suffit de réaliser l'édition de liens entre le programme appelant et la bibliothèque LIBsRPC.

Les inconvénients de cette méthode sont:

a. un stub client créé sur un site donné, pourrait ne jamais etre invoqué. A la longue, il se poserait un problème de saturation des ressources (mémoire).

b. une certaine quantité d'informations constituant la déclaration d'interface doit être envoyée par le site serveur lors de l'exportation en même temps que le nom de la procédure. Ceci est un inconvénient surtout dans le cas où le réseau n'est pas fiable.

c. de plus, cette méthode ne présente aucune symétrie entre le côté appelant et appelé. En effet, d'un côté le stub serveur est généré à la demande et de l'autre le stub client est produit systématiquement à la réception d'un message d'exportation envoyé par le site serveur.

2. le stub client est généré à la demande du client. Avant d'invoquer une procédure à distance, l'utilisateur demande la génération du stub client. Une fois ce stub produit il est inséré dans la bibliothèque LIB$RPC s'il n'y existait pas déja. Ensuite, il suffit de réaliser l'édition de liens entre le programme appelant et la bibliothèque.

Cette méthode présente un inconvénient lorsqu'au niveau du programme appelant, il existe plusieurs appels à des procédures distantes. Dans ce cas, l'utilisateur serait obligé de demander la génération du stub client de chaque procédure invoquée dans son programme en fournissant la déclaration d'interface de chacune d'elles.

L'avantage par rapport à la première méthode est que le stub client n'est généré qu'à la demande donc diminution des risques de saturation des ressources. De plus, il existe une symétrie entre le côté appelant et appelé puisque les stubs client et serveur sont générés de la même manière, c'est-à-dire à la demande.

Choix de la méthode

La méthode que nous avons choisie est un compromis entre les deux méthodes précitées. Le stub client est créé à la demande du client et nécessaire. Les informations contenant la description de la procédure distante seront envoyées par l'utilisateur en même temps qu'il exporte le nom de la procédure. De plus pour éviter la saturation des ressources (mémoire) un stub client donné sera supprimé de la bibliothèque LIB$RPC après le premier appel, s'il n'existe aucune autre demande d'importation de la procédure de la part d'un autre client. Sur chaque catalogue, nous disposons donc d'un compteur de demandes d'importation d'une procédure donnée. Ce compteur est diminué de un à chaque appel de cette procédure. Lorsqu'il atteint la valeur zéro, le stub client est supprimé de la bibliothèque LIB$RPC.

I.3.5.3 Traitement des erreurs

1. Erreurs de communication

Les erreurs de communication telles que perte, désèquencement, duplication de messages sont détectées soit par le service de communication [Shr 82] soit par le service d'invocation de procédure à distance [Dir 84].

Dans le cas où le service de communication utilisé ne détecte pas les erreurs de communication, le service de contrôle d'appel de procédure à distance ou stub doit être doté d'un

mécanisme de détection de ces erreurs. Le principe de ce mécanisme est basé sur la retransmission des messages à chaque suspicion de perte. Les duplications sont éliminées grâce à des numéros de séquence et l'utilisation d'un identificateur unique pour chaque appel [Bir 84].

2. Pannes de sites

a. Panne du site serveur

Elle est détectée soit par le service d'appel si la réponse à un appel n'est pas reçue avant écoulement d'un délai de garde prédéterminé [Shr 82], soit par le système de communication par l'échange de messages "PROBE" (testant simplement la présence du serveur) [Bir 84] ou "BUSY" (équivalents aux messages "PROBE" mais dont les échanges sont initiés par le serveur) [Sat 86]. Ces messages doivent être acquittés durant l'exécution.

La panne d'un site serveur peut être détectée soit au moment de l'établissement de la connexion soit durant les échanges entre les deux sites. Il se présente alors deux cas possibles:

- la panne est temporaire: le stub client doit redemander l'établissement de la connexion jusqu'à son établissement (si la panne survient au moment de l'établissement de la connexion), ou réémettre le message (si la panne survient au moment des échanges) jusqu'à la réception de la réponse.

- la panne peut être permanente: dans ce cas, le stub client signale cette panne aux autres sites et le client reçoit une réponse négative à son appel. Lorsque le site serveur est remis en fonctionnement, il doit restaurer l'état précédent la panne et signaler sa reprise aux autres sites.

b. Panne du site client

La panne d'un site client peut être signalée au stub serveur soit par le service de communication, soit en ne recevant pas d'acquittement au message "BUSY" [Sat 86], le traitement est alors annulé. La mise en oeuvre d'un algorithme d'élimination d'orphelins s'avère nécessaire afin de terminer brutalement l'exécution des serveurs à des appels de clients disparus. En effet, si une panne d'un site appelant est signalée au site appelé, les serveurs locaux sont annulés. Dans le cas d'appels imbriqués, un message d'annulation est envoyé à tous les sites serveurs concernés. Pour cela, il faut sauvegarder en mémoire tous les appels sortants d'un site liés à un appel rentrant à ce site.

c. Détection et élimination des orphelins

L'élimination des orphelins nécessite un protocole d'abandon des procédures en cours d'exécution, suite à la signalisation de la disparition d'un appelant. Cette signalisation peut être

originaire du site local ou d'un site distant. B.J.NELSON [Nel 81] présente plusieurs méthodes d'élimination des orphelins, nous discuterons, dans ce qui suit de quelques unes d'entre elles:

- La première méthode consiste à faire tomber en panne toutes les machines quand un site tombe en panne. Cette solution est, bien sûr, inacceptable mais adoptée, quand même par plusieurs systèmes.

- La deuxième solution consiste à faire tomber en panne juste les machines sur lesquelles s'exécutent les orphelins. Cette solution reste extrême.

- Une amélioration de la deuxième solution, consiste à éliminer, seulement, les processus qui exécutent les appels des procédures orphelines. Cette solution est, bien sur la plus adéquate et c'est celle que nous adopterons. Il se pose alors un problème de détection des processus orphelins, surtout dans le cas d'appel de procédures imbriquées. Ce problème peut être résolu par la mémorisation sûre des appels sortants correspondants à chaque appel rentrant sur un site donné. Cependant, le suivi devient impossible si les orphelins s'exécutent eux même sur des sites en panne.

I.4 Service de communication

Tout appel de procédure à distance se traduit à un moment ou à un autre par un transfert physique de données entre les entités participant à l'appel. Ce transfert est réalisé par un service de communication. Un service de communication doit offrir un ensemble de fonctions d'émission et de réception pour le transfert de messages et des fonctions de contrôle qui assurent un certain degré de fiabilité dans le cas où le taux d'erreurs de transmission n'est pas négligeable. Il existe deux types de service de communication :

1- type circuit virtuel (orienté connexion)

Ce type de service permet d'établir une voie virtuelle (connexion) sur laquelle des informations peuvent être échangées. Le terme de circuit virtuel est réservé à une voie virtuelle comportant contrôle d'erreurs et régulation de flux et sur laquelle des informations sont reçues sans duplication et dans l'ordre où elles ont été émises. De plus, ce service permet de signaler des erreurs telles que la coupure d'une ligne physique de communication ou la mise hors tension d'un des sites connectés.

2- type datagramme (non orienté connexion)

Le service datagramme permet le transfert de blocs de données de taille fixe, d'une adresse origine à une adresse destination. Ce service ne fournit aucun moyen de détecter les erreurs qui peuvent affecter l'information échangée, telles que duplication, perte, corruption et déséquencement. Il est surtout

utilisé dans des environnements composés de calculateurs homogènes interconnectés par un réseau local à haut débit et faible taux d'erreurs. La qualité de ce service dépend donc de celle du réseau de communication.

Choix du service de communication

Le service datagramme est un service minimal. Il offre donc une grande efficacité au niveau temps d'exécution et espace mémoire. Le service circuit virtuel bien que plus complexe et donc moins performant, dispose d'un ensemble de services généraux accessibles par les entités de traitement.

Le choix du type de service de communication dépend du type de traitement et du réseau de communication. Dans un environnement constitué de calculateurs homogènes interconnectés par un réseau local de faible taux d'erreurs, le service datagramme est suffisant. Si certaines fonctions s'avèrent nécessaires, elles pourront être mises en oeuvre par les utilisateurs de ce service.

Dans le cas d'un environnement ouvert, composé d'un ensemble de calculateurs hétérogènes interconnectés par un grand réseau de communication, le service orienté connexion est plus adéquat.

Ainsi les systèmes d'appel de procédure à distance conçu par Birrell et Nelson [Bir 84] et Shrivastava et Panzieri [Shr 82] utilisent, pour le transfert de messages de petite taille, un service datagramme (réseau IP de Xerox pour Birrell et service de la boucle de Cambridge pour Shrivastava). Les paquets de données de grande taille sont transmis, dans le cas de Birrell en plusieurs messages avec acquittement de chacun d'entre eux et un numéro de séquence pour détecter les pertes. Shrivastava a, quant à lui, réalisé un service datagramme évolué permettant l'échange de données de taille importante (UDS : Universal Datagramme Service). Par contre, le service d'appel de procédure à distance conçu par Mondain [Mon 87] pour un environnement ouvert, utilise, pour le transfert des données, un service orienté connexion.

I.5 Architecture du service d'appel de procédure à distance

Un appel de procédure à distance se décompose en deux étapes bien distinctes qui sont les suivantes :

1- appel au stub: cet appel est effectué par l'entité de traitement (programme utilisateur).

2- la deuxième étape se décompose elle même en plusieurs étapes:

- localisation de la procédure distante,
- accès au service de communication pour établir la connexion logique et effectuer les transferts physiques des messages,
- codage et décodage des paramètres d'appel et de retour,
- traitement des erreurs,

> synchronisation des échanges,
> gestion des appels simples ou imbriqués.

Les deux étapes décrites ci-dessus ainsi que les différentes fonctions d'une procédure stub nous ont conduit à proposer une architecture fonctionnelle pour l'exécution d'un appel de procédure à distance décrite sous forme de couches hiérarchiques (fig. I-11). Chaque couche réalise un service en utilisant les services offerts par les couches inférieures, et offre par conséquent, des services plus élaborés à la couche de niveau supérieur. Les entités adjacentes à un service interagissent par l'intermédiaire de primitives de service.

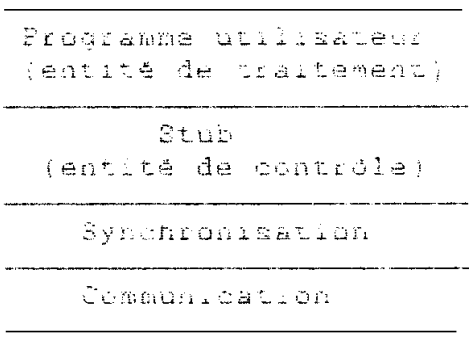

fig. I-11 Architecture fonctionnelle

Cette structuration en couche rejoint la structure du modèle de référence de l'ISO (International Standard Organization) qui est constitué, quant à lui de 7 couches (fig. I-12).

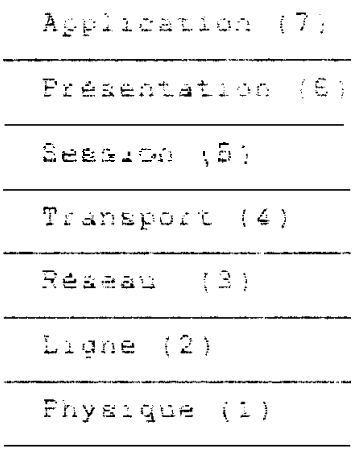

fig. I-12 Modèle de référence de l'ISO

Selon le principe de structuration en couches, chaque couche ajoute de nouvelles fonctionnalités à l'ensemble des fonctionnalités fournies par les couches inférieures de manière à satisfaire, au niveau de la couche la plus haute, les besoins des applications réparties [Mon 87, Alg 82].

L'abstraction d'une couche N et de toutes les couches inférieures (N-1, N-2,..) est appelée service (N). Les entités réalisant ce service sont les entités (N) et les entités utilisant ce service sont les entité (N+1). Les règles de coopération entre les entités (N) sont définies par un protocole (N).

Un protocole de niveau N est mis en oeuvre par l'échange d'unités de données du protocole (N) ((N)PDU : (N)Protocol Data Unit), par l'utilisation des services de niveau (N-1).

Les unités de données le protocole (N+1) échangées par les entités (N+1) sont transmises à la couche N en émission et reçues de la couche N en tant qu'unités de données de service (N) ou primitives de service.

La conception d'une couche d'un niveau N donné s'effectue en trois étapes:

- définition des services à réaliser par rapport aux besoins identifiés des utilisateurs du niveau supérieur N+1,
- définitions des fonctions à réaliser au sein de cette couche,
- définition du protocole de niveau N à réaliser en tenant compte des services offerts par le niveau N-1.

I.6 Conclusion

Les mécanismes de reprise d'erreurs de communication (retransmission temporisée et acquittement des messages) permettent d'assurer à notre service la propriété "exactement une fois" en l'absence de pannes de sites et la propriété "au plus une fois" lors de l'occurrence de telles situations. Ceci est dû au fait que la durée d'une panne d'un site donné est indéterminée donc pouvant être permanente. Dans ce cas, le client peut recevoir une réponse négative à son appel.

D'autre part, notre service permet la réalisation de traitements répartis atomiques en utilisant le mécanisme d'engagement et d'annulation nécessaire au maintien de la cohérence d'un traitement réparti.

De plus, la possibilité d'hétérogénéité des différents éléments est prise en compte, facilitant ainsi l'intégration de ce service dans des environnements hétérogènes. La décomposition du service d'appel de procédure à distance en couches hiérarchiques, permet une implantation modulaire. Par conséquent, chaque service offert par une couche donnée peut être spécifié, développé ou modifié indépendamment des autres couches.

CHAPITRE II

Specification formelle du service d'appel de procedure à distance

II.1 Introduction

Les échanges entre entités de traitement participant à une application répartie sont en général, complexes et sources d'erreurs difficiles à corriger. Les protocoles régissant ces échanges sont d'autant plus difficiles à concevoir que les systèmes sont complexes et réalisent des applications sophistiquées. i'occurence d'un mauvais fonctionnement peut avoir des consequences graves, ii est donc essentiel de prendre en compte cet aspect dès le stade de la conception. Afin d'assurer le bon fonctionnement et la sureté des systèmes, la conception de protocoles nécessite l'utilisation d'une méthodologie de conception riyoureuse. supportée par des outils adéquats.

L'étape premiere de cette méthodologie consiste a spécifier de façon formelle le fonctionnement d'une application répartie en incluant le conportement séquentiel et les interactions réparties. Cette spécification formelle permrt d'obtenir un modèle formel qui doit permettre de représenter de façon compacte et plus explicite le fonctionnement du protocole considéré. D'autre part l'utilisation d'un modèle formel permet l'application de techniques de validation grâce auxquelles nous pourrons vérifier le bon comportement. du protocole.

Alors que le comportement local d'une entité peut être en général, décrit par un automate séquentiel, la modélisation du fonctionnement global d'un protocole nécessite un formalisme permettant d'exprimer le parallélisme. L'outil de modélisation choisi doit permettre:

- l'obtention de modèles de spécification formelle non ambigus et les plus simples possible,

- l'application sur ces modèles des techniques de vérification des propriétés caractéristiques des protocoles.

Les approches actuellement développées reposent sur deux bases: un modèle d'architecture et un modèle de comportement. Le modèle d'architecture utilisé est le modèle en couches de l'ISO ou certaines variantes. Plusieurs modèles de comportement ont été utilisés, nous citons:

- machines à états finis [Boc 78]
- modèles mixtes [Kel 76, Raz 80],
- réseaux de Petri [Dia 82, Jua 84, Cou 84, Aya 85],

- types de données abstraits [Ger 80].

Parmi ces différents outils, nous avons choisi le formalisme à réseaux de Petri pour les raisons suivantes:

- leur puissance de description: ils permettent d'exprimer explicitement le parallélisme et la synchronisation entre processus,
- la possibilité d'interconnecter les différents modèles: la modélisation du protocole pourra s'effectuer par etapes,
- la disponibilité d'un logiciel de vérification PIPN (Prolog Interpreted Petri Nets) developpé au LAAAS/CNRS et qui est basé sur les réseaux de Petri.

II.2 Réseaux de Petri

II.2.1 Définition

Formellement, un réseau de Petri est un quintuplet (P, T, Pré, Post, M_0) [Bra 83] où:

P = (p_1, p_2, ..., p_m) est un ensemble fini de places;
T = (t_1 ,t_2, ..., t_n) est un ensemble de transitions;
Pré : PxT ——————> N est l'application d'incidence avant ou application d'entrée des transitions (N est l'ensemble des entiers naturels);
Post : PxT ——————> N est l'application d'incidence arrière ou application de sortie des transitions;
M_0 : P ——————> N est l'application de marquage initial; un marquage M associe à chaque place p un nombre de marques (jetons) tel que M(p)>=0.

II.2.2 Règles de fonctionnement

Une transition t est sensiblisée par un marquage M si et seulement si :

V p ∈ P, M(p) >= Pré(p,t).

Cette pré-condition de franchissement étant vérifiée, la transition t peut être tirée. Cela va provoquer le changement du marquage M en M' tel que :

V p ∈ P, M'(p) = M(p) + C(p,t) où C est la matrice d'incidence définie par:
C : PxT ——————> Z (Z est l'ensemble des entiers relatifs)
(p_i,t_j) ——————> c_{ij} = Post(p_i,t_j) - Pré(p_i,t_j)

La règle d'évolution ci-dessus définit une relation d'accessibilité sur l'ensemble des marquages d'un réseau de Petri. Un marquage M_k est accessible depuis un marquage M_1 si et seulement si il existe une séquence de transitions t_1, t_2, ..., t_{k-1} successivement tirables (appelée séquence de tir), qui conduit de M_1 à M_k.

Le comportement dynamique d'un réseau de Petri est donc défini par l'ensemble de ses marquages accessibles depuis M_0. Il est représenté sous forme de graphe dont chaque arc (M_1,M_2) est étiqueté par la transition correspondante du réseau. Ce graphe est appelé graphe de marquage [Bra 83].

II.2.3 Propriétés des réseaux de Petri

L'analyse des modèles décrits à l'aide des réseaux de Petri permet la vérification de deux catégories de propriétés [Aig 82, Aya 85, Jua 84, Aze 88]:

1. Propriétés générales

Elles fournissent une information sur le système, son fonctionnement et sa structure. Ces propriétés sont indépendantes de la mission confiée au système. Elles garantissent que le modèle défini possède "un comportement qui paraît correct et cohérent" [Aze 88, Aya 85].

Les propriétés générales sont définies en fonction d'un marquage initial M_0 et leur établissement nécessite l'élaboration du graphe des marquages conséquent de M_0.

Les propriétés générales d'un réseau de Petri sont: borné, vivant et réinitialisable.

La propriété "borné" garantit que le système a un nombre fini d'états et peut donc être implanté. La propriété "vivant" garantit que le système n'a pas de blocage. La propriété "réinitialisable" garantit un comportement cyclique du système.

2. Propriétés spécifiques

Une propriété spécifique d'un protocole est une propriété qui dépend de l'interprétation, c'est-à-dire de la sémantique que le concepteur associe aux éléments du modèle qui sont les places, les transitions [Aze 88, Aya 85].

II.2.4 Extensions des réseaux de Petri

Les réseaux de Petri se prêtent bien à la modélisation des systèmes notamment ceux à évolution parallèle. De nombreuses extensions ont été ensuite développées pour augmenter leur puissance de modélisation. En effet, il existe maintenant une large variété de modèles étendant les réseaux de Petri initiaux. Parmi ces modèles, nous citons:

- les réseaux de Petri temporels [Rou 85],
- les réseaux de Petri numériques [Aya 85],
- les réseaux de Petri à arcs inhibiteurs [Bra 83],
- les réseaux de Petri prédicat/action appelés aussi réseaux étiquetés [Aze 88].

C'est ce dernier type de réseaux de Petri que nous allons utiliser pour modéliser le service d'appel de procédure à distance.

II.2 Contexte de la modélisation

II.2.1 Les réseaux de Petri étiquetés

La modélisation à l'aide des réseaux de Petri étiquetés permet de distinguer la partie "contrôle" et la partie "données" de la spécification d'un système. La partie contrôle est décrite au moyen d'un réseau de Petri sauf c'est-à-dire pour lequel il n'y a au maximum qu'un jeton dans chaque place. A chaque transition du réseau est ajoutée une étiquette de la forme: **"si prédicat alors action"** (notre prédicat/action) dans laquelle le prédicat et l'action portent sur les variables globales du système, représentant de manière explicite la partie "données" du système spécifié. Le prédicat peut être une réception de message et/ou la vérification d'une fonction quelconque. De même, l'action peut être une émission d'un message et/ou une modification de valeur ou la vérification d'une fonction quelconque.

Dans ce modèle, une transition sera tirable si elle est sensibilisée au sens des réseaux de Petri classiques et si le prédicat associé à cette transition est évalué à vrai. L'action est alors exécutée de façon indivisible (fig. II-1).

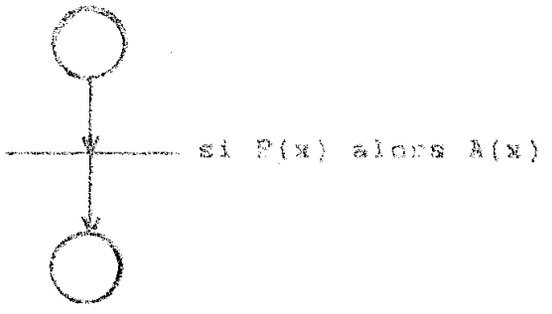

fig. II-1 Transition dans un réseau de Petri étiqueté

Dans le cas de notre spécification, l'étiquette associée à une transition représente l'échange de messages du système avec son environnement. Pour représenter l'étiquette nous avons utilisé la notation de Hoare [Hoa 78] suivante : ?M1/!M2. Cette notation signifie qu'à la réception d'un message M1(?M1) le message M2(!M2) est émis. Il faut noter que l'étiquette peut ne représenter que la partie prédicat ou la partie action seulement.

L'analyse des modèles décrits en réseaux de Petri étiquetés se fera à l'aide de l'interpréteur PIPN que nous décrirons succinctement dans le paragraphe suivant.

II.3.2 L'interpréteur PIPN

L'interpréteur PIPN est un outil d'analyse des réseaux de Petri étiquetés implanté en prolog (PIPN: Prolog Interpreted Petri Nets).

Un système modélisé en PIPN est décrit par un ensemble de places, un ensemble d'états initiaux et un ensemble de transitions. La description se fait au moyen de prédicats Prolog et obéit à la syntaxe suivante:

```
    place (liste des places du modèle)
    ident (ensemble des transitions)
    init (ensemble des états initiaux)
    tr (idf-tr,
label(input(ip-in(message)),output(ip-out(message))),
         pre (pré-conditions),
         post (post-conditions)).
```

place, ident, init, tr, label, pre, post, input et output sont des mots réservés. tr est suivi de l'identificateur de la transition considérée, de la description de son étiquette, des pré-conditions et enfin des post-conditions. ip-in et ip-out sont les points d'interaction avec l'environnement.

Exemple: Soit un processus qui émet une requête, attend une réponse puis se termine, le modèle réseau de Petri étiqueté décrivant le comportement de ce processus est constitué de trois places (début, attente, fin) et de deux transitions (envoi de la requête, réception d'une réponse) (fig. II-2).

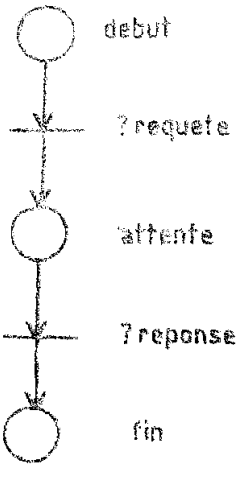

fig. II-2 Modèle réseau de Petri étiqueté du comportement du processus considéré

La description en PIPN du comportement de ce processus est la suivante:

```
place ([début, attente, fin]).

ident ([émission, reception]).

init([début]).

tr (émission,
   label ([output(ip-out(requête))]),
   pre ([début]),
   post ([attente]).

tr (réception,
   label ([input(ip-in(réponse))]),
   pre ([attente]),
   post ([fin]).
```

L'interpréteur PIPN permet une vérification exhaustive par génération et analyse des automates correspondants aux réseaux de Petri étiquetés du modèle. Dans le cas d'un système composé de plusieurs éléments (c'est-à-dire plusieurs réseaux connectés), l'analyse des comportements peut se faire de façon modulaire:

- analyse de l'automate partiel: analyse du fonctionnement de chaque élément,
- analyse de l'automate global: analyse du fonctionnement du système global.

L'analyse de l'automate permet de vérifier les propriétés générales et spécifiques. L'analyse des propriétés spécifiques d'un automate est réalisée par réduction de celui-ci. Cette réduction permet de visualiser un comportement du système réduit à un sous ensemble des événements possibles. Ces événements sont dits événements observables. Cette méthode est encore appelée méthode de projection. PIPN utilise deux méthodes de projection:

- la première est basée sur l'équivalence de langage (ou projection classique). Dans cette technique, la séquence des événements visibles ne dépend pas des événements invisibles. Ces événements correspondent à des interactions protocole/utilisateur qui sont des primitives de service. Tous les autres événements tels que la réception ou l'envoi d'une unité de données du protocole sont dits inobservables.

- la deuxième technique est basée sur l'équivalence observationnelle. Cette technique donne une vue plus fine du comportement du système. En effet, dans ce type de projection, si des événements définis inobservables possèdent une influence ultérieure sur les événements observables, ils peuvent alors apparaître dans l'automate. Cette technique permet d'analyser le comportement du protocole, le service perçu localement par un

utilisateur et le service global fourni aux utilisateurs (séquences possibles d'interactions entre les deux entités de traitements distantes).

II.3.3 Modèle d'architecture pour l'analyse

La méthodologie de modélisation actuellement utilisée dans les études sur les systèmes distribués et les protocoles repose sur une représentation par réseaux de Petri des tâches se déroulant dans un cadre d'architecture proche de celui donné par le modèle de référence de l'ISO. Chaque couche est constituée de plusieurs entités coopérantes selon un protocole. Les entités utilisent le service fourni par la couche immédiatement inférieure et fournissent elles mêmes un service à la couche immédiatement supérieure.

Ainsi, la spécification d'un protocole décrit le comportement de chaque entité en réponse aux requêtes venant des entités de la couche supérieure et en réponse aux indications de service venant des entités de la couche inférieure. Ces dernières interactions externes correspondent soit à des messages provenant d'entités distantes, soit à des évènements engendrés par le service de la couche inférieure sur sa propre initiative.

L'analyse du comportement d'un niveau N revient donc à l'analyse du couple service N /protocole N en tenant compte du service N-1. Ces conventions nous conduisent à considérer l'architecture suivante [Aya 85] (fig. II-3):

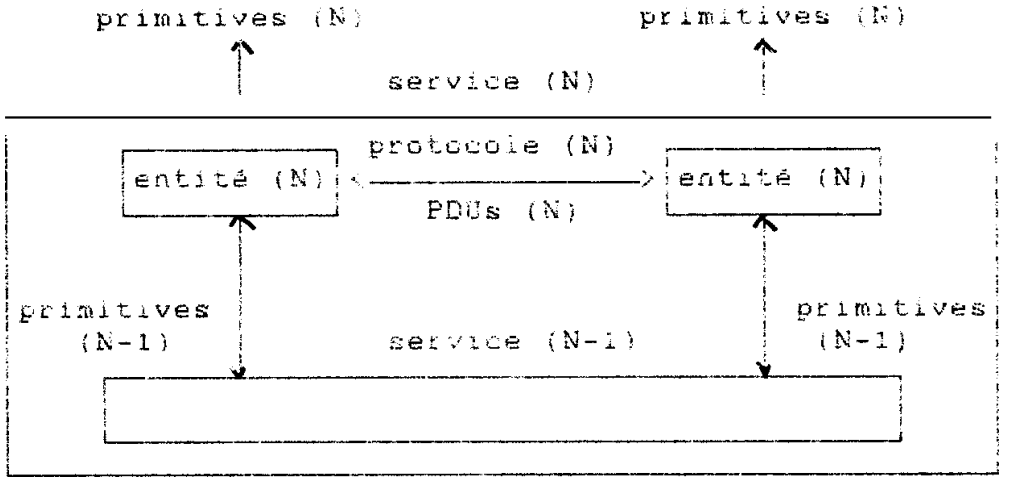

fig. II-3 Modèle d'architecture pour l'analyse

L analyse du comportement du couple service (N)/protocole (N) se fera à partir:

- des modèles locaux des entités (N),
- du modèle du service (N-1) (service (N-1) fourni par la touche inférieure).

```
{Connexion du module stub-serveur au module medium}
connet (stub-serveur(output(ss-m-out(X))),
        medium (input(m-ss-in(X)))).

{Connexion du module medium au module stub-serveur}
connet (medium(output(m-ss-out(X))),
        stub-serveur(input(ss-m-in(X)))).

{Attacher les points d'interaction sc-in et sc-out avec des
 points d'interaction équivalents d'un module utilisateur
 client}
attach (APD(input(c-in(X))), stub-client(input(sc-in(X)))).
attach (APD(output(c-out(X))),
stub-client(output(sc-out(X)))).

{Attacher les points d'interaction ss-in et ss-out avec des
 points d'interaction équivalents d'un module utilisateur
 serveur}
attach (APD(input(s-in(X))), stub-serveur(input(ss-in(X)))).
attach (APD(output(s-out(X))),
stub-serveur(output(ss-out(X)))).
```

Remarque:

La connexion des points d'interaction permet de fusionner les transitions en émission d'un module et en réception de l'autre module. L'interpréteur PIPN offre la possibilité de connecter deux points d'interaction par l'intermédiaire de files d'attente de longueur quelconque pouvant être infinie. Cependant, dans le cas où la file est de longueur infinie,l'interpréteur ne permet pas de générer des exceptions.

Nous allons présenter dans ce qui suit les états des modules stubs client et serveur ainsi que les différentes unités de données du protocole (PDUs) échangées entre les deux entités et les requêtes et indications (primitives de service) échangées entre les entités et l'environnement (stubs/utilisateurs et stubs/service de communication).

a. Description de l'entité stub client

Les états de l'entité stub client sont:
- état-initial: pas d'appel en cours,
- attente-retour: attente des résultats de l'appel,
- verif-res: vérification des résultats de l'appel,
- fin-appel: fin de l'appel après envoi d'un message d'engagement ou d'annulation.

b. Description de l'entité stub serveur

Les différents états d'un stub serveur sont:
- état-initial: pas d'appel en cours,
- verif-param: vérification des paramètres reçus dans le message d'appel,

- attente-réponse: attente réponse du serveur invoqué localement,
- attente-engt-ou-annul: attente décision du stub client,
- fin-appel: fin de l'appel après réception d'un message d'engagement ou d'annulation.

Remarques:

- La vérification des paramètres d'appel par le stub serveur est nécessaire dans le cas où le message d'appel serait altéré durant le transfert. En outre, le fait de donner la possibilité au client de décrire la procédure distante lors de la génération du stub client, peut engendrer des erreurs du genre incompatibilité de types.

- La vérification des resultats par le stub client est nécessaire en cas d'altération du message retour.

c. Les unités de données du protocole (PDUs)

Les différents PDUs échangés entre le stub client et le stub serveur sont:
- pdu-appel: message d'appel envoyé par le stub client au stub serveur,
- pdu-retour: message de retour envoyé par le stub serveur au stub client,
- pdu-annul: message d'annulation envoyé par le stub client au stub serveur ou vice versa.
- pdu-engt: message d'engagement envoyé par le stub client au stub serveur.

d. Les primitives de service

Il existe deux sortes de primitives de service: les requêtes et les indications. Les requêtes sont:

- r-appel: requête d'appel reçue par le stub client,

- r-retour: requête de retour reçue par le stub serveur. Le retour peut être positif (r-retour(+)) dans le cas d'une exécution complète et sans erreurs de la procédure invoquée ou négatif (r-retour(-)) s'il ya occurrence d'une erreur quelconque lors de l'exécution.

Les indications sont les suivantes:

- i-appel: indication d'appel transmise par le stub serveur au serveur,

- i-retour: indication de retour envoyée par le stub client au client. Dans ce cas là aussi, on distingue le retour positif (i-retour(+)) et le retour négatif (i-retour(-)),

- i-annul: indication d'annulation envoyé par le stub serveur au serveur,

- i-abort: indication d'erreur émise par le medium dans le cas où il peut générer des exceptions.

L'interpréteur PIPN a été utilisé pour vérifier les propriétés du protocole et du service offert. Cette vérification a été réalisée par génération et projection sur l'automate global.

Nous allons dans ce chapitre, étudier le comportement du protocole et son influence sur le service offert. Pour des raisons de clarté et de lisibilité et afin de faciliter l'analyse, nous allons considérer séparément différents cas de figure de ce protocole. Le premier cas décrit un comportement normal. Les deux autres cas sont obtenus à partir du premier soit en rajoutant un pdu-annul envoyé par le client après le pdu-appel et avant le pdu-retour (asynchronisme du client), soit en considérant un medium de communication pouvant générer des erreurs (pannes ou pertes de messages) et les signaler aux entités communicantes. Dans les deux premiers cas nous considérons un medium de communication parfait, géré de façon fifo.

II.4.2 Comportement normal

Le comportement du protocole d'appel de procédure à distance est dit normal si le medium de communication ne génère aucune indication d'exception (pas de panne de site et pas de perte de messages) et si le stub client ne prend pas l'initiative d'annuler un appel distant après l'envoi du message d'appel au stub serveur. Les modèles obtenus sont:

1. Modèle du stub client

Ce modèle est représenté sur la figure II-5. La sémantique des places est:

état 1: état-initial,
état 2: attente-retour,
état 3: vérif-res,
état 4: fin-appel.

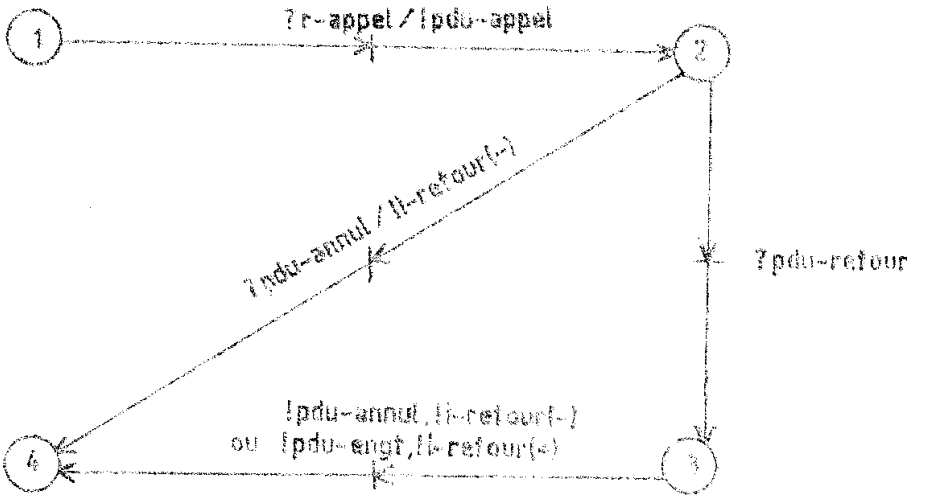

fig. II-5 Modèle du stub client

2. Modèle du stub serveur

Ce modèle est représenté sur la figure II-6. La sémantique associée aux places est la suivante:

état 1: etat-initial,
état 2: vérif-param,
état 3: attente-réponse,
état 4: attente-engt-ou-annul,
état 5: fin-appel.

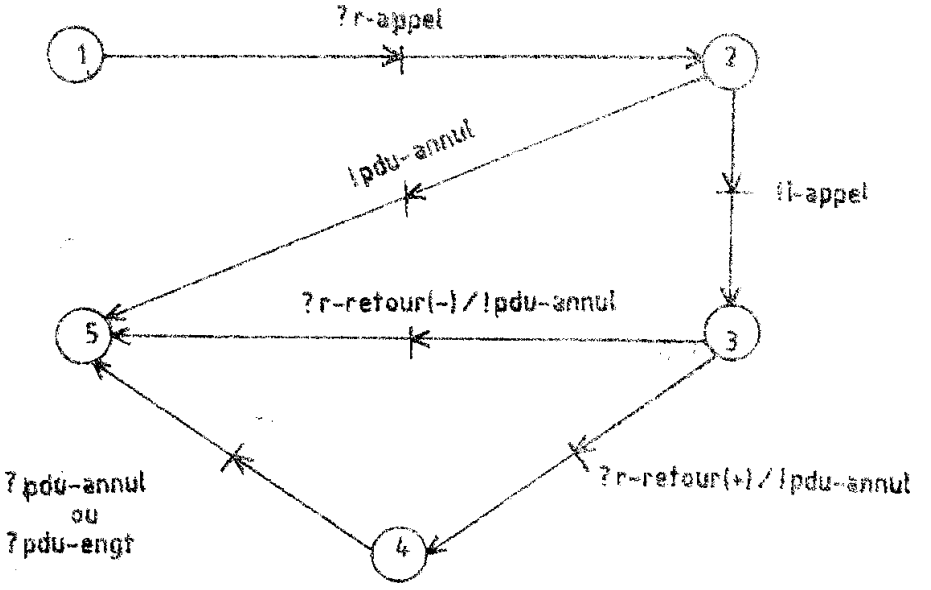

fig. II-6 Modèle du stub serveur

3. Analyse globale

L'analyse globale est basée sur le graphe des marquages caractérisant le modèle global obtenu par interconnexion des modèles stub client et stub serveur à travers le modèle du medium. Ce graphe des marquages définit l'automate global. Le modèle du medium n'est pas représenté ici car c'est un modèle bien connu [Aya 85]. Cette analyse permet de vérifier les propriétés générales et spécifiques.

3.1 Propriétés générales

L'automate global est composé de 10 états et de 12 transitions. L'analyse par PIPN a permis de vérifier l'absence de blocage du protocole et de l'utilisateur (pas d'états puits).

3.2 Propriétés spécifiques

La vérification des propriétés spécifiques est réalisée par projection sur le service de l'automate global. Dans ce cas, seuls les événements correspondant à des primitives de service sont observables.

i. Projection avec équivalence langage

L'automate projeté (6 états et 8 transitions) avec équivalence langage est le suivant (fig. II-7):

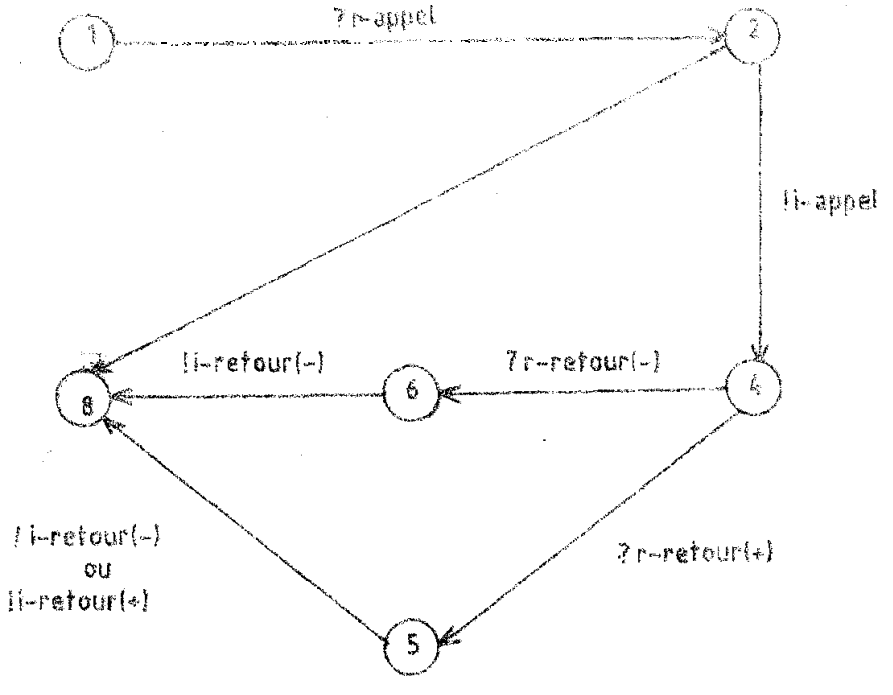

fig. II-7 Automate du service (projection langage)

A partir de cet automate, nous distinguons plusieurs comportements possibles du service global fourni. Nous allons donner ci-dessous deux exemples de séquences:

- séquence 1, 2, 4, 5, 8: cette séquence décrit un comportement normal du protocole. En effet, le stub client reçoit une requête d'appel provenant du client (?r-appel), il envoie un message d'appel au stub serveur qui émet une indication d'appel vers le serveur (!i-appel). Le serveur exécute normalement la procédure puis retourne le contrôle au stub serveur qui reçoit une requête de retour positif (?r-retour(+)). Le stub serveur envoie, à son tour le message retour au stub client qui vérifie les résultats puis transmet une indication de retour positif si les résultats sont corrects ou une indication de retour négatif si les résultats sont non conformes à ceux attendus.

- séquence 1, 2, 8: le stub client reçoit une requête d'appel (?r-appel) puis envoie au client une indication de retour négatif (!i-retour(-)). Cette séquence d'événements décrit un comportement anormale. Cependant, la projection avec équivalence langage de l'automate global ne nous permet pas d'expliquer ce comportement. Nous allons donc projeter l'automate global avec équivalence observationnelle afin de faire apparaître l'événement qui a induit la séquence 1, 2, 8 de l'automate précédent.

2. Projection avec équivalence observationnelle

L'automate projeté avec équivalence observationnelle possède 6 états et 8 transitions dont une n'est pas observable (fig. II-8).

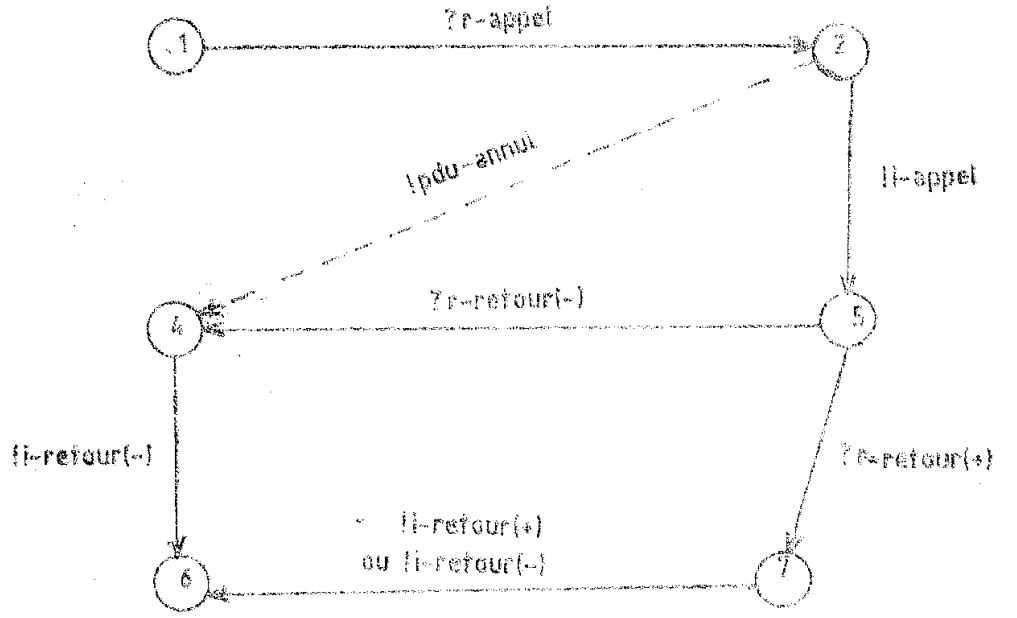

fig. II-8 Automate du service (projection observationnelle)

A partir de ce graphe, nous pouvons déduire que l'apparition de la séquence 1, 2, 8 du graphe précédent a été engendrée par l'envoi d'un message d'annulation (!pdu-annul: arc (2,4) par le stub serveur au stub client; le stub serveur ayant effectué une vérification sur les paramètres d'appel.

Cette abstraction du service global fourni aux deux entités ne correspond pas à la perception que les utilisateurs en ont: cet automate présente les séquences d'interactions visibles par un observateur global des deux sites. Les entités de traitement client et serveur ne disposent pas d'une telle vision et doivent se conformer aux services visibles localement.

II.4.3 Comportement asynchrone du stub client

Dans ce cas de figure, le stub client possède un comportement asynchrone, c'est-à-dire qu'il peut envoyer un pdu-annul après avoir envoyé le pdu-appel. Ceci se présente dans le cas où l'attente d'une réponse lui paraît trop longue. On obtient alors les modèles suivants:

1. Modèle du stub client

Le modèle réseau de Petri étiqueté décrivant le comportement du stub client est donné par la figure II-9.

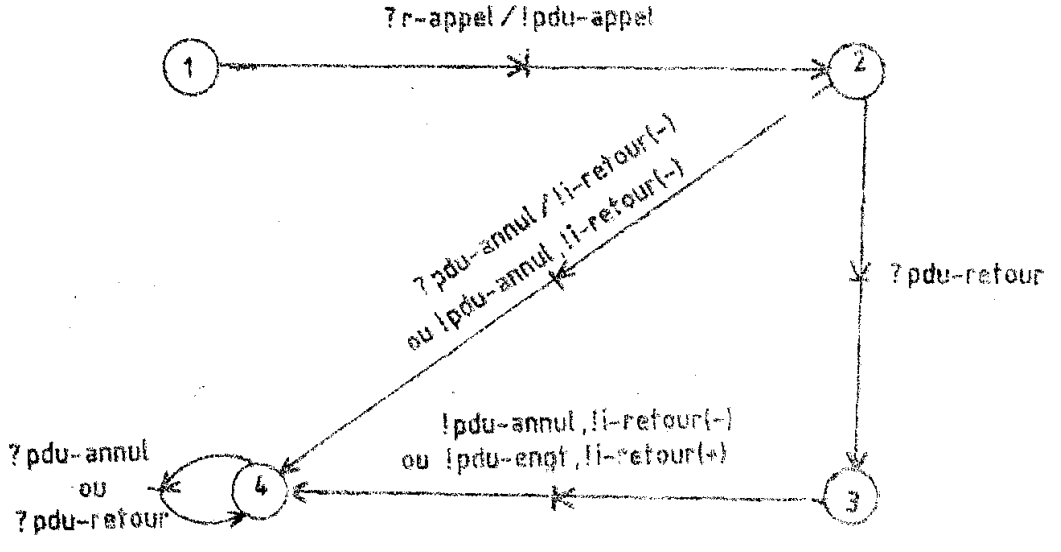

fig. II-9 Modèle du stub client

L'événement supplémentaire par rapport au modèle de la figure II-5 consiste en l'annulation de l'appel par le stub client et l'indication d'un retour négatif vers le client. Cet événement est représenté par la transition étiquetée par:

!pdu-annul, !i-retour(-). Dans ce cas, la boucle sur l'état 5 permet de consommer les messages non normalement attendus. En effet, si le stub serveur n'a pas encore reçu le message d'annulation, il peut envoyer un message d'annulation (!pdu-annul) ou de retour (!pdu-retour).

2. Modèle du stub serveur

Le modèle réseau de Petri étiqueté du stub serveur est représenté sur la figure II-10.

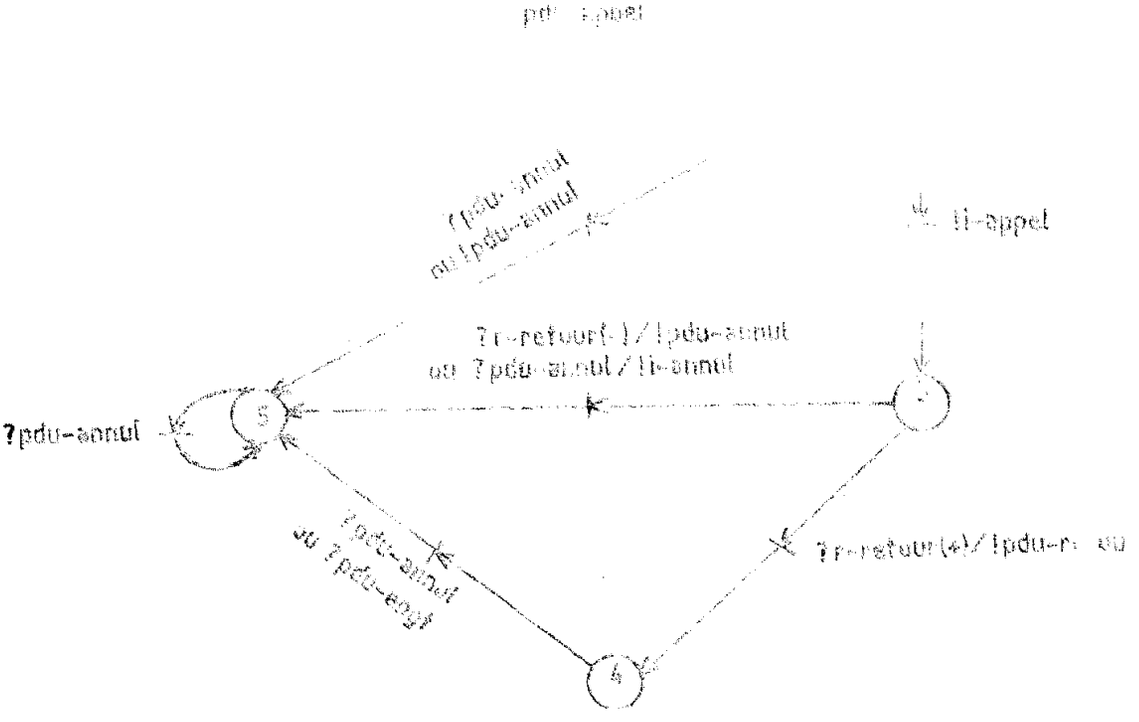

fig. II-10 Modèle du stub serveur

L'élément supplémentaire par rapport au modèle de la figure II-6 est la réception par le stub serveur d'un pdu-annul (annulation de l'appel par le stub client). Ce message peut être reçu soit avant l'activation du service (transition étiqueté par ?pdu-annul) soit lors de l'exécution de ce dernier et dans ce cas le stub serveur émet une indication d'annulation vers le serveur (transition étiquetée par ?pdu-annul/!i-annul).

La boucle sur l'état 5 permet, comme dans le cas du modèle du stub client d'absorber les messages non normalement attendus. En effet, le stub serveur peut envoyer un message d'annulation (!pdu-annul) ou de retour (!pdu-retour) et se terminer puis recevoir un pdu-annul envoyé par le stub client qui n'a pas encore reçu le message envoyé par le stub serveur.

3. Analyse globale

3.1 Propriétés générales

L'automate obtenu possède 18 états et 31 transitions. L'analyse par PIPN a permis de vérifier l'absence de blocage du protocole et de l'utilisateur (pas d'états puits).

3.2 Propriétés spécifiques

a. Projection avec équivalence langage

L'automate obtenu par projection avec équivalence langage (8 états et 13 transitions) est représenté sur la figure suivante (fig. II-11).

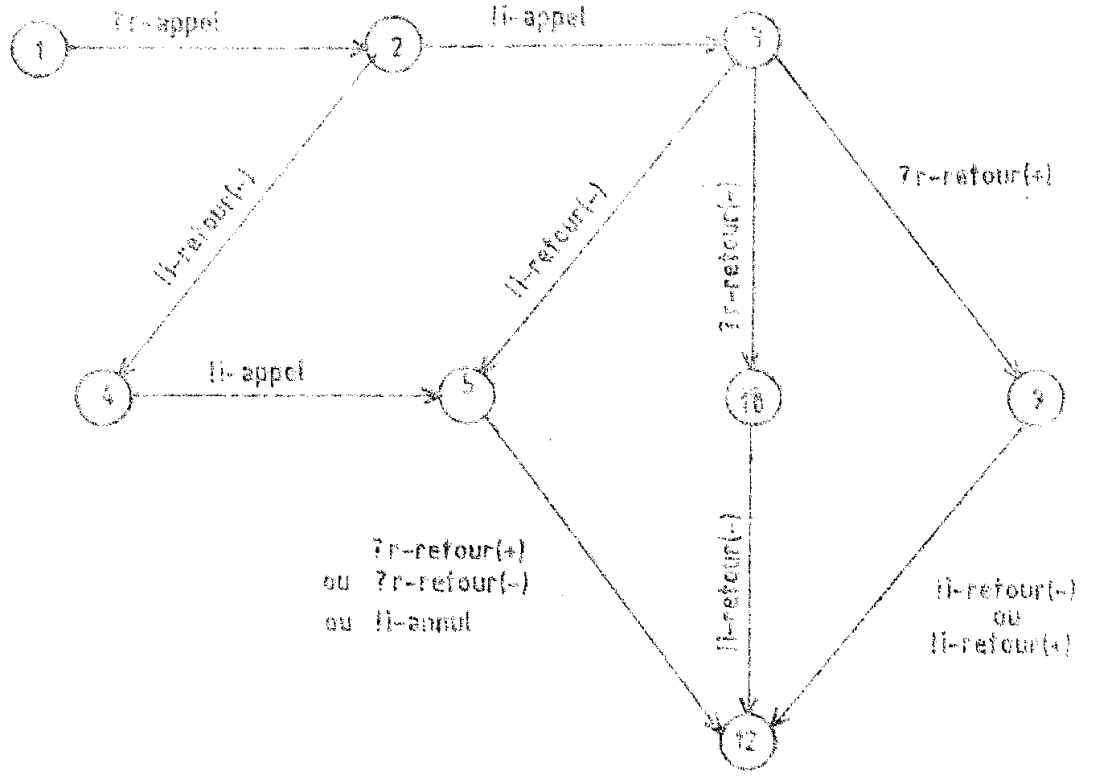

fig. II-11 Automate du service (projection langage)

L'automate obtenu met en évidence plusieurs séquences décrivant le comportement du service global fourni. Parmi ces séquences, nous citons:

- séquences 1, 2, 3, 9, 12 et 1, 2, 3, 10, 12 décrivent un comportement normal du protocole c'est-à-dire sans apparition de l'asynchronisme du client. Dans ce cas une requête d'appel r-appel (1, 2) est suivie d'une indication d'appel i-appel (2, 3) qui sera suivie à son tour soit par une requête de retour positif

(arc (3, 9)= exécution complète et sans erreurs de la procédure) ou bien d'une requête de retour négatif (arc (3, 10) : exécution non complète). La requête de retour positif est suivie soit par une indication de retour négatif (résultats reçus erronés) ou positif (résultats corrects). Pour la requête de retour négatif elle sera suivie d'une indication de retour négatif.

- séquence 1, 2, 3 décrit un comportement anormal du service fourni. En effet, dans le cas de cette séquence, une requête d'appel est immédiatement suivie par une indication de retour négatif. Cela signifie que l'occurrence d'un événement inobservable sur la frontière entre les

- séquence 1, 2, ..., 8 est une autre séquence décrivant aussi un comportement anormal du service. En effet, une requête d'appel (?r-appel) est immédiatement suivie par une indication de retour négatif (!i-retour(-)).

2. Projection avec équivalence observationnelle

La projection de l'automate obtenu avec équivalence observationnelle nous permettra de voir l'influence des événements inobservables sur la séquence des événements observables et d'expliquer les comportements anormaux du service fourni. L'automate obtenu possède 8 états et 10 transitions dont 3 sont invisibles (fig. II-12)

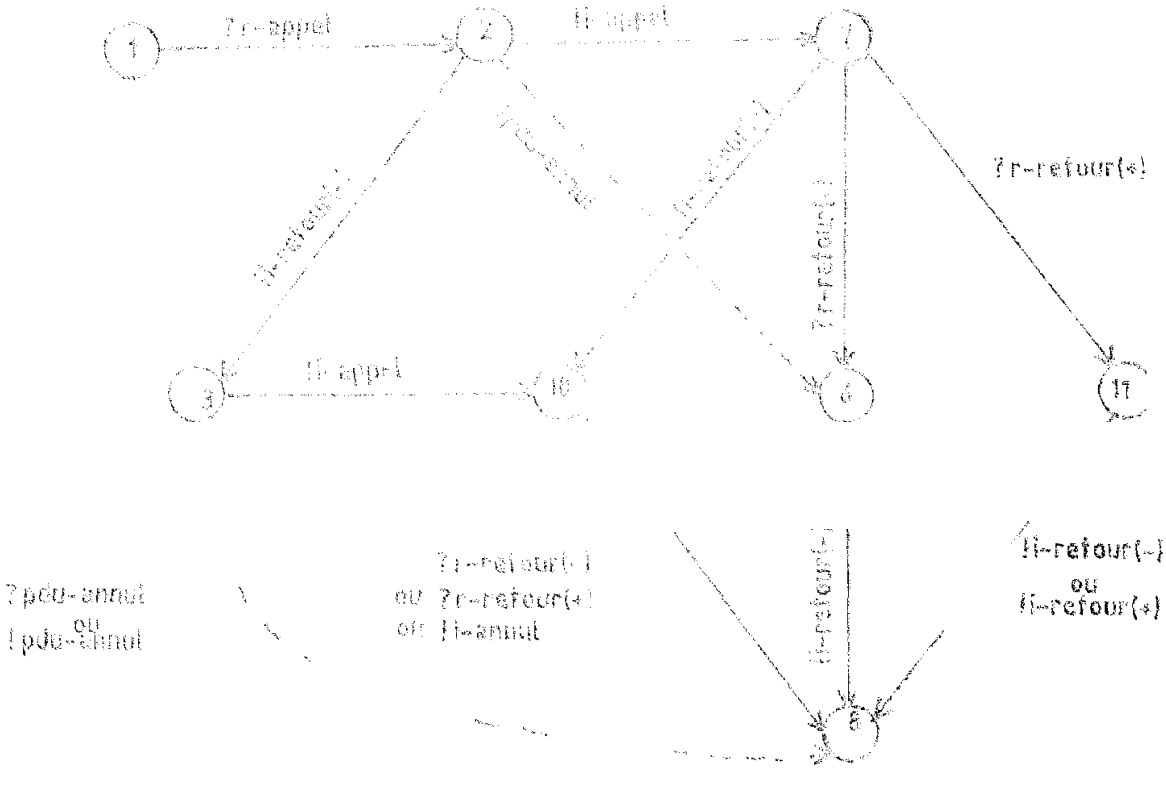

Fig. II-12 Automate du service (projection observationnelle)

Cet automate nous permet de déduire que la séquence 1, 2, 4 de l'automate prdcddent a été engendrée par l'envoi ou la reception d'un pdu-annul (arc (3,8)) qui représente respectivement l'annulation de l'appel par le stub client (attente trop longue de la réponse) ou par le stub serveur (paramêtres reçus non conformes à ceux attendus).

Quant à la sequence 1, 2, 3, 5, elle est dûe à l'envoi d'un pdu-annul par le stub client et ce avant la réception des résultats.

En outre, cet automate montre que l'événement !i-retour(-) (arc (10,12)) de l'automate précédent peut être aussi engendré par l'envoi d'un pdu-annul par le stub client (arc (2,6)).

L'automate projetd avec equivalence observationnelle a permis de mettre en evidence l'asynchronisme du stub client.

II.4.4 Indication de perte de messages et de panne par le medium

Nous allons dans ce qui suit, considérer un medium de communication qui peut générer des pannes, purger les PDUs en transit quand une panne a lieu et signaler leu pannes aux entitds communicantes.

1. Modèle du medium

Le modèle rdseau de Petri étiqueté décrivant ce medium **[Ayo 90]** (fig. 11-13) est composd de:

- deux files de capacité 1 représentant le transfert de PDUs entre le site client et le site serveur. Chaque file est constituée de deux places: CS1, CS2 du cot& client et SC1, SC2 du coté du serveur. La place CS1 (respectivement SC1) indique que le medium coté client (respectivement coté serveur) eat vide tandis que la place CS2 (respectivement SC2) indique qu'il est plein.

- deux places NEC et NES qui représentent la condition "pas d'erreurs" respectivement du coté client et serveur. L'occurence d'une panne de communication est représentée par la transition te.

- deux places EC et ES représentent une condition d'erreur respectivement sur le site client et le site serveur. La transition tp permet de purger le PDU en transit et une indication de panne i-abort est transmise aux entités stub client et stub serveur.

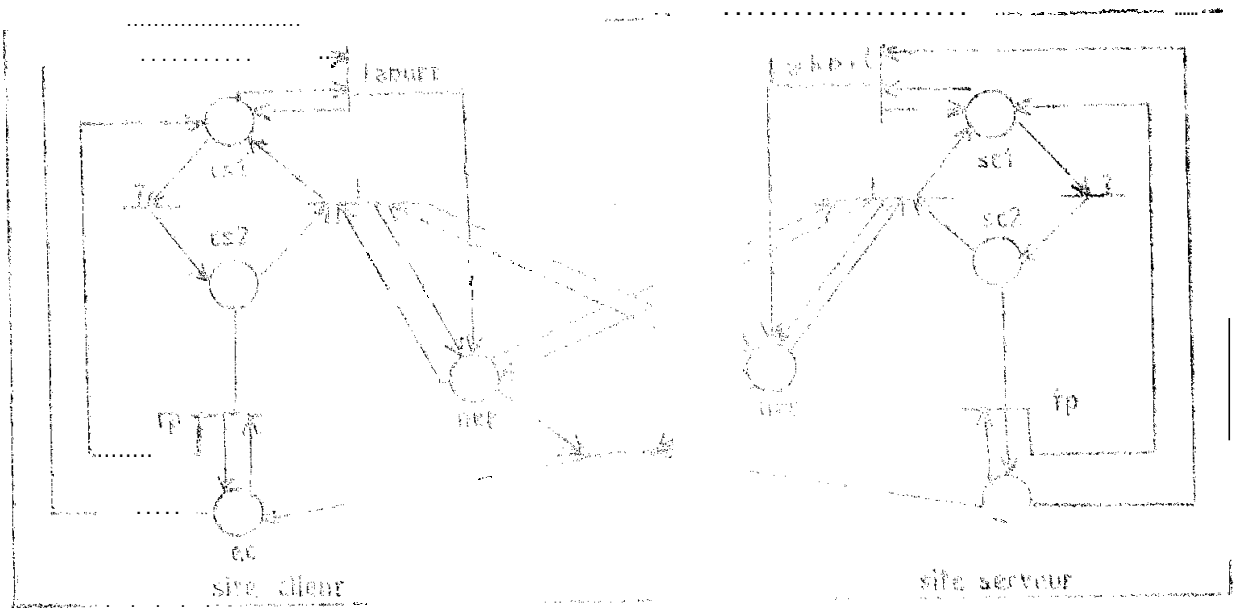

Fig. II-13 Modèle du service de communication

2. Modèle du stub client

Ce modèle est représenté sur la figure II-14.

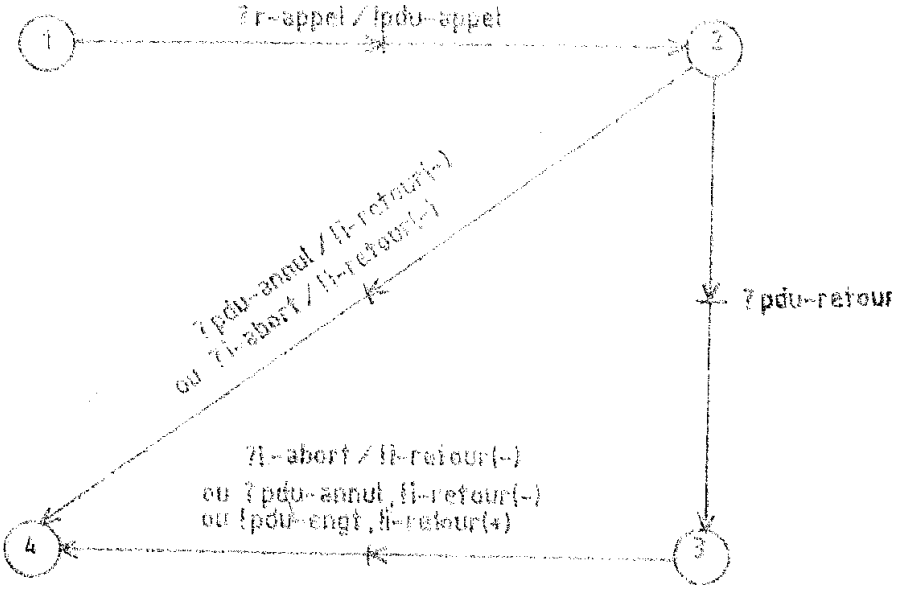

Fig. II-14 Modèle du stub client

L'élément supplémentaire par rapport au modèle de la figure II-5 consiste en la réception par le stub client d'une indication d'abort (?i-abort) émise par le medium de communication pour signaler la panne du site serveur ou la perte de messages. Cette indication engendre une indication de retour négatif transmise par le stub client à destination du client.

3. Modèle du stub serveur

Le modèle réseau de Petri étiqueté du stub serveur est donné par la figure II-15.

Par rapport au modèle de la figure II-6, le stub serveur peut recevoir une indication d'abort (?i-abort) envoyée par le medium de communication afin de signaler la panne du site client ou la perte de messages. Si le serveur a été déjà activé, l'événement ?i-abort engendre une indication d'annulation (!i-annul) envoyée par le stub serveur pour arrêter l'exécution en cours.

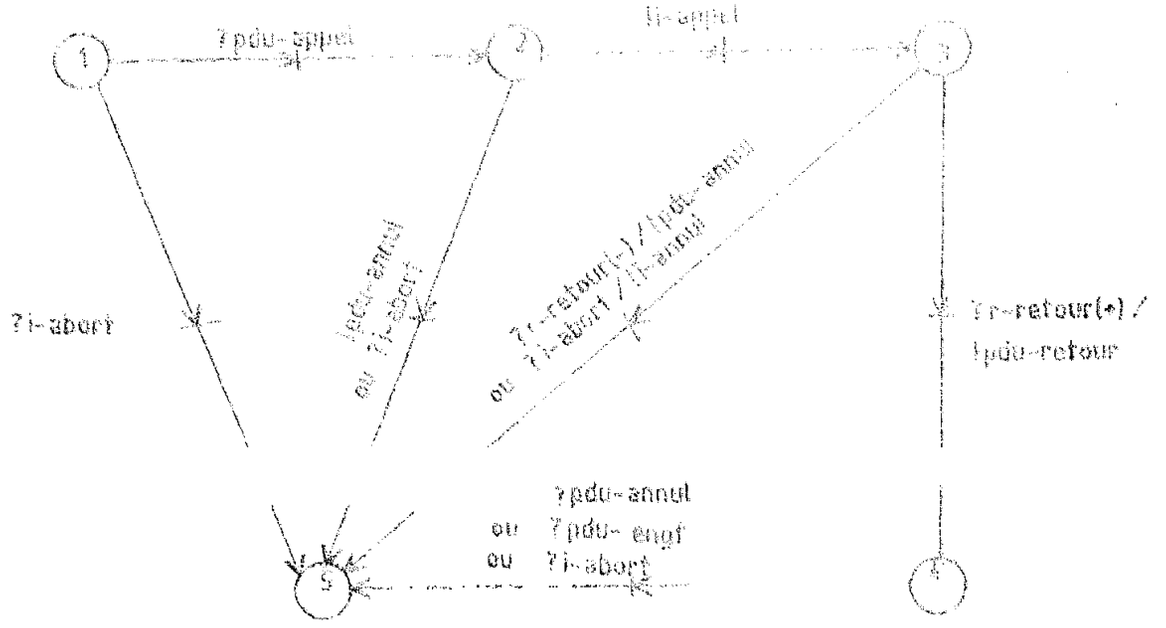

Fig. II-15 Modèle du stub serveur

4. Analyse globale

4.1 Propriétés générales

L'automate possède 38 états et 69 transitions. L'analyse à l'aide de PIPN a permis de vérifier qu'il n'y a pas de blocage du protocole et de l'utilisateur.

4.2 Propriétés spécifiques

1. Projection avec équivalence langage

La projection de cet automate avec equivalence langage donne un automate de 8 états et de 14 transitions (fig. II-16).

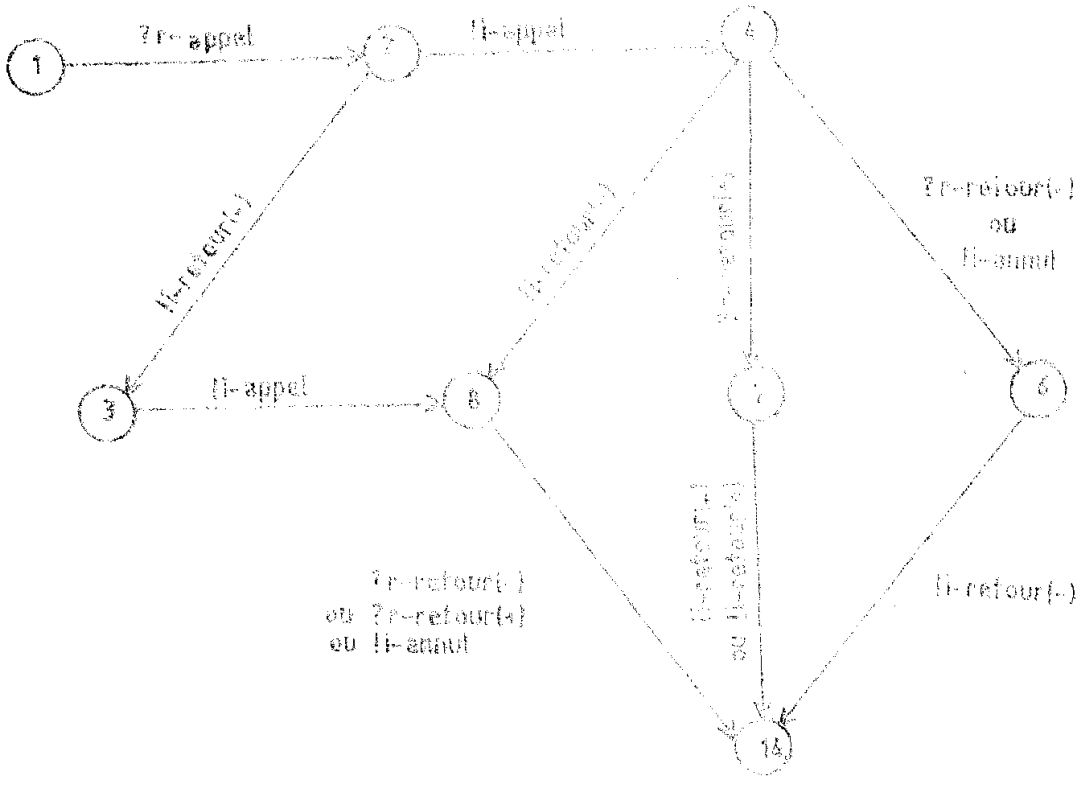

fig. II-16 Automate du service (projection langage)

Cet automate permet de mettre en évidence plusieurs séquences décrivant le comportement du service global fourni. Parmi ces séquences, nous citons:

- La séquence 1, 2, 4, 7, 14 qui décrit un comportement normal du service. En effet, une requête d'appel r-appel est suivie d'une indication d'appel puis d'une requête de retour positif et enfin d'une indication de retour (négatif ou positif).

- La séquence 1, 2, 4, 8, 14 représente un comportement anormal du service. En effet, une requête d'appel est suivie d'une indication d'appel puis d'une indication de retour négatif vers le client.

- Dans la séquence 1, 2, 3, 8, 1e, une requête d'appel est suivie immédiatement après par une indication de retour négatif. Cette séquence représente aussi un comportement anormal du service.

3. Projection avec équivalence observationnelle

L'automate projeté avec équivalence observationnelle va nous permettre d'expliquer les différents comportements anormaux du service. Cet automate est constitué de 12 états et de 25 transitions dont 11 sont inobservables (fig. II 17).

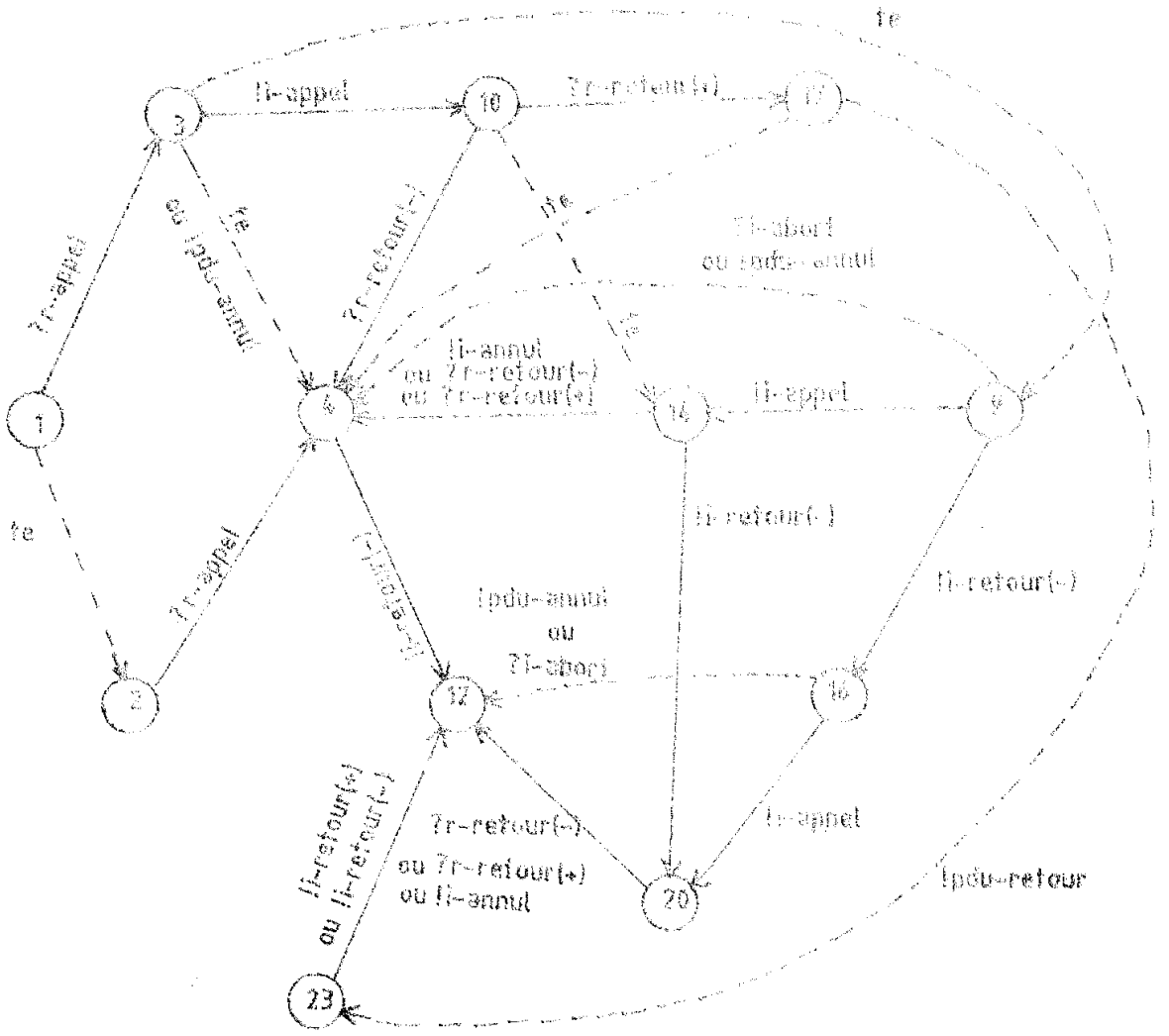

Fig. II-17 Automate du service (équivalence observationnelle)

Les séquences anormales précédemment citées sont expliquées de la manière suivante:

Les sequences anormales précédemment citees sont expliquées de la manière suivante:

- la sequence 1, 2, 4, 8, 14 de l'automate précédent (fig. 11.16) correspond à la sequence 1, 3, 10, 14, 20, 12 de cet automate (fig. 11-17). L'événement inobservable ayant provoqué cette sequence est l'arrivée d'une erreur représentée par la transition te: arc (10,14).

- la séquence 1, 2, 3, 8, 14 de l'automate projeté avec Bquivalence langage correspond quant a elle, à la sequence de tarnsitions 1, 3, 9, 16, 20, 12. La transition te (arc (3,9)) a provoquée ce comportement anormal du service.

L'automate projeté avec bquivalence observationnelle montre également qu'une requête d'appel peut être suivie d'une indication de retour négatif dans le cas de l'envoi d'un pdu-annul par le stub serveur (arc (3,4)).

11.5 Conclusion

Dans ce chapitre, nous avons dbcrit la verification du service d'appel de procedure à distance en utilisant un outil complet et performant (PIPN).

Cette phase de vérification, nous a permis de corriger les erreurs de conception et de verifier la coherence des interactions entre entités distantes. Elle a en outre prouvé que l'implantation du service d'appel de procedure à distance est possible.

CHAPITRE III

Implantation du service d'appel de procedure à distance

III.1 Description de l'environnement

Le service d'appel de procédure à distance est implanté sur un réseau compose de trois machines (noeuds ou sites) VAX et du gestionnaire de réseau DECnet. Les nueuds adjacents sont connectés par un cable coaxial de marque ETHERNET (fig. III-1). Le réseau ainsi obtenu est UP, réseau local homogène. Les sites sont identifies par des numéros logiques distincts. Par la suite, on assimilera le numéro d'un site à SOP, adresse.

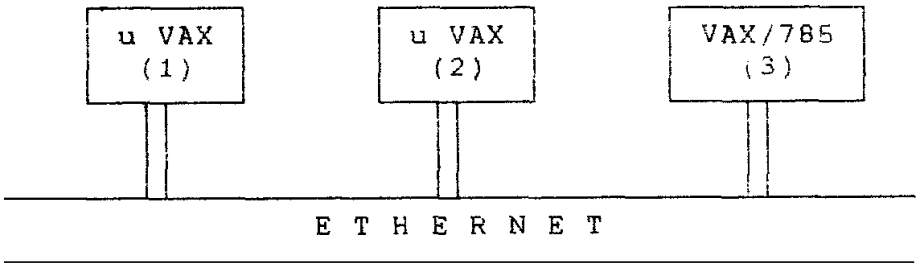

fig. III-1 Environnement de travail

Les produits DECnet sont bases sur la conception en couches spécifiée dans l'architecture réseau de DIGITAL (Digital Network Architecture: DNA) [Dig 62, Dig 84]. Les fonctions de DECnet, les différentes couches de DNA à partir desquelles elles ont été initiées et les protocoles DNA avec lesquels ces fcnctiuns ont été implantées sont illustrés dans la figure III-2.

Les messages envoyés à travers ETHERNET sont appelés datagrammes car il n'y a aucune garantie qu'un message va être reçu par le destinataire. Une connexion fiable peut être fournie en interposant un protocole entre l'utilisateur et ie service datagramme d'ETHERNET. Dans DNA, ce circuit virtuel est fourni par le protocole du service reseau dans la couche communication.

L'implantation d'un service de contrôle basé sur 1'appei de procedure à distance sur ce réseau permet d'utiliser les services offerts par l'architecture DNA sans avoir à les redéfinir. En effet, la fonction de communication est réalisé par ies services offerts par les quatre couches basses dir l'architecture DNA et la synchronisation des échanges est assurée par les services de la couche session. Les stubs (client et serveur) sont quant à eux, implantés au niveau de la couche application.

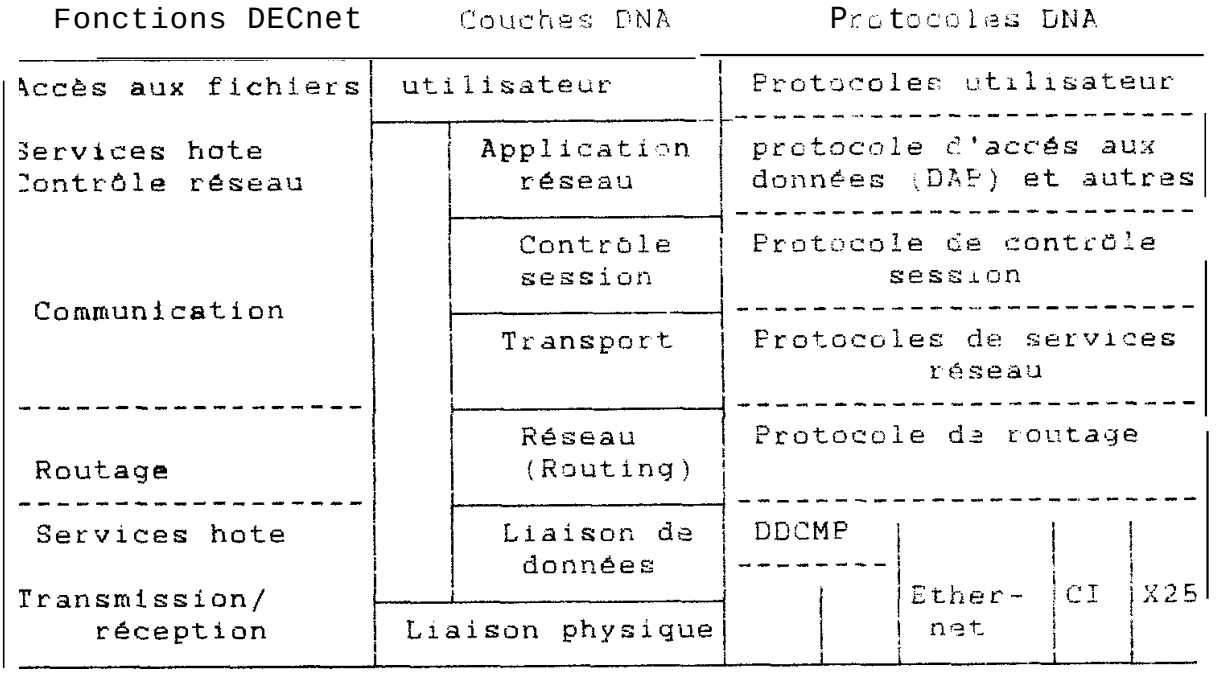

fig. 111-2 Architecture DNA

Remarque :

Le réseau utilisé étant un réseau homogène, les problèmes de representation des données dans ies messages n'ont pas été évoqués. Si des functions de codage et de décodage s'avèrent nécessaires (en cas d'hétérogenéité des langages ou conncxion d'un autre type de machinej, elles seront rajoutees dans le service de contrôle.

III.2 Primitives de service

L'accès d'une entité (N) au service (N-1) se fait suivant le type de l'entité et du service. Nous distinguons trois cas:

- L'entité de traitement client accede au stub client (service de contrôle coté client) par un appel local.

- Le stub client et le stub serveur [service de contrôle coté serveur) accèdent aux services des couches inférieures (session, transport, etc..) par des ruutines systèmes uffertes par le gestionnaire de réseau DECnet [Dig 84]. Ainsi ie **stub** client établit un lien **logique** avet le site distant en utrlisant la routine système SYS$ASSIGN(CANAL,DEVNAME) où DEVNAME csntient le nom du stub serveur et l'adresse du site distant et **CANAL** est un numéro retourné par cette routine. Ce numéro permet de designer de façon unique un lien logique pour un appel donne. I1 joue le rôle de l'identificateur unique d'un appel de procedure à distance utilisé par Birrell et Nelson [Brr 84]. **Lea** envois et les receptions de messages se feront à l'aide de la routine

SYS$QIO en indiquant le numéro du canal ainsi que d'autres informations telles que l'adresse du buffer contenant le **message**, la taille du message, etc..

- Le serveur accède au stub serveur en effectuant un retour local.

Nous distinguons, également deux types d'indications émises par les entités N-1 vers les entités N, à savoir:
- un retour local du stub client vers le client,
- un appel local du stub serveur vers le serveur

III.3 Réalisations spécifiques

Avant de décrire l'implantation du service d'appel de procédure à distance, nous signalons d'abord que l'environnement de travail (réseau DECnet-vax) présente les caractéristiques suivantes:
- pas de déséquencement, de perte ou d'altération de messages,
- probabilité de panne très faible,
- durée de la panne d'un site n'est pas connue à priori.

Nous avons vu dans le premier chapitre que pour invoquer une procédure à distance, il faut au préalable qu'elle soit exportée. A la fin de cette phase d'exportation, un stub serveur est généré sur son site de résidence. De même, avant d'invoquer une procédure à distance, un client doit d'abord importer cette procédure. A l'issue de cette importation un stub client est généré du coté de l'appelant.

L'implantation du service de procédure à distance nécessite donc la mise en oeuvre de deux fonctions importantes à savoir l'exportation et l'importation. Ces deux fonctions font appel au générateur de stubs pour produire respectivement le stub serveur et le stub client.

III.3.1 Exportation

Exporter une procédure revient à diffuser un message contenant, entre autres son nom et l'adresse du site de résidence. Dans le cas où, cette procédure n'existe sur aucun site (création pour la première fois), l'utilisateur doit fournir sa description (déclaration d'interface) afin de pouvoir générer le stub serveur correspondant. Cette description est à son tour diffusée.

Si la **procedure** existe déjà, le stub serveur est généré en utilisant la description existante. Il s'agit en fait d'une instantiation de la procedure.

A la reception du message d'exportation, chaque site *met* **à jour** sun catalogue local en rajoutant le nom de la **procedure** et l'adresse du site si **la** procedure est exportée pour **la** première

fois ou uniquement l'adresse du nouveau site en cas d'instantiation de la procédure. Dans les deux cas, le champ ETAT correspondant au site d'exportation est mis à ACTIF.

Afin de résoudre les problèmes de conflit et d'incohérence des différentes copies du catalogue, un algorithme de maintien de la cohérence est mis en oeuvre [Bou 88].

Principe de l'algorithme

Localement, le problème de conflit et d'incohérence est résolu par un verrouillage du catalogue au début de l'exécution de la recherche du nom de la procédure et un déverrouillage après la mise à jour. Cependant si la mise à jour du catalogue est effectuée immédiatement, ce nom risque d'être retiré si sur un autre site le résultat de la recherche est défavorable. Si la mise à jour ne s'effectuait qu'après réception des accords de tous les sites, les catalogues seraient trop longtemps monopolisés par des verrouillages et aucune consultation ne serait autorisée jusqu'à ce que la mise à jour soit reportée sur toutes les copies (cohérence forte). Le temps d'exécution serait alors très important même pour une simple consultation.

Par conséquent, l'algorithme que nous allons adopter repose sur le principe de la cohérence faible, à savoir que toutes les copies deviennent identiques au bout d'un temps fini [Bou 88]. Les mises à jour sont donc différées et l'accès à une copie n'est pas interdit lors de la mise à jour d'une autre copie.

Pour éviter la monopolisation des catalogues, l'algorithme utilise en outre, un catalogue intermédiaire (sur chaque site), appelé table des demandes qui contient à tout moment toutes les demandes d'exportation locales ou distantes en cours. Les demandes sont estampillées par la valeur de l'heure logique du site exportateur de la procédure. C'est un estampillage au sens de Lamport [Lam 78], effectué automatiquement. Nous définissons une relation de priorité entre deux demandes comme suit:

Soient deux sites de numéros i et j respectivement. Le site i émet la demande dem_i d'estampille $stamp_i$. Le site j émet la demande dem_j d'estampille $stamp_j$. Dans le cas où les estampilles sont différentes on a:

si $stamp_i < stamp_j$ alors dem_i est prioritaire,
si $stamp_j < stamp_i$ alors dem_j est prioritaire.

Dans le cas où les estampilles sont égales, on aura:

si $i > j$ alors dem_i est prioritaire,
sinon dem_j est prioritaire.

Le principe de l'algorithme est le suivant: lorsqu'il y a une demande locale d'exportation d'une procédure, la table des demandes est consultée:

- si aucune procédure portant le même nom n'y figure alors la demande est insérée sinon elle est refusée sans consultation des autres catalogues et une erreur est transmise au demandeur local.
- si la demande est insérée dans la table des demandes, elle est ensuite diffusée vers les autres sites. Lorsqu'un site i reçoit une demande d'un site j, deux cas se présentent:

1) i a émis une demande dem_i semblable à dem_j:
- si dem_i est prioritaire alors une réponse négative est envoyée au demandeur j,
- si dem_j est prioritaire alors l'exportation est refusée au demandeur local et une réponse favorable est envoyée au demandeur j.

2) Il n'existe aucune demande semblable à dem_j (ni locale ni de la part des autres sites), une reponse favorable est envoyée au demandeur j et la demande est insérée dans la table des demandes.

La demande reste dans cette table jusqu'à la mise à jour des catalogues.

Du point de vue du demandeur deux cas se présentent:
- s'il reçoit une réponse favorable de tous les sites, il diffuse un message de mise à jour à tous les sites,
- sinon il signale à l'utilisateur l'existence de cette procédure.

Lorsqu'un site reçoit une derande de mise à jour, il met à jour son catalogue et supprime la demande de la table des demandes. Cette table est donc nécessaire puisqu'elle permet de repeter toutes les demandes identiques.

III.3.2 Importation

Lorsqu'un utilisateur désire invoquer une procédure distante, il doit d'abord l'importer. Importer une procédure revient à générer son stub client en utilisant la description de la procédure transmise lors de l'exportation.

La génération d'un stub client est effectuée si les conditions suivantes sont vérifiées:
- la procédure distante a déja été exportée,
- le site de résidence de cette procédure est actif (ETAT = ACTIF),
- la procédure n'a encore été importée par aucun utilisateur du même site. Si la procédure a déja été importée sur ce site, le stub client correspondant exite dans la bibliothèque LIBSRPC.

Nous disposons donc, au niveau du catalogue, d'un champ NBRE-IMPORT qui contient le nombre de demandes d'importation d'une procédure donnée. L'algorithme suivant illustre les différentes actions effectuées lors de l'importation :

```
Debut
  (* Donner le nom de la procédure à importer *)
  lire(proc);
  si recherche(proc) = vrai et état = actif
   alors début
          si nbre-import = 0
          alors générer le stub client
          fsi;
          nbre-import := nbre-import + 1
         fin
   sinon écrire('la procédure n'a pas été exportée')
  fsi;
fin;
```

III.3.3 Déexportation

Dans le cas où un utilisateur ne désire plus que sa procédure soit accessible à distance, temporairement ou définitivement, il doit effectuer une déexportation de cette procédure. Pour cela, il diffuse un message de déexportation à tous les sites. La réception d'un message de déexportation définitive provoque la suppression du nom de la procédure du catalogue et la suppression de sa description s'il n'existe qu'une seule instance de cette procédure. S'il en existe plusieurs, seul le site ayant demandé la déexportation est supprimé.

Dans le cas où la déexportation est temporaire, la valeur du champ ETAT du catalogue est changée (ETAT = ABS-temp). Le catalogue local est modifié de la même façon.

III.3.4 Le générateur de stubs

Etant donné la nature homogène du réseau sur lequel notre service est implanté, il ne sera pas tenu compte de l'hétérogénéité des machines. Le générateur de stubs n'aura donc en entrée que la description de la procédure et la spécification du langage dans lequel le stub sera généré.

III.3.4.1 Spécification de langage

L'utilisation de langages différents pour l'appelant et l'appelé peut engendrer les problèmes suivants:

- les langages peuvent exprimer une structure de données similaire utilisant une syntaxe différente avec une sémantique légèrement différente,

- les langages peuvent supporter des structures de données et des interfaces différentes (procédures et fonctions par exemple).

Pour construire les spécifications des différents langages, il faut donc:

- sélectionner un ensemble commun de types de données. Cet ensemble contient dans notre cas les types de données suivants: booleen, entier, reel, caractère, chaine de caractères, tableau (array), enregistrement (record) et pointeur. Tous les langages doivent soit avoir ou pouvoir construire chaque type de donnée de cet ensemble qui peut par la suite être enrichi. Ceci ne pose aucun problème car la plupart des langages ont des structures de données similaires.

- sélectionner un ensemble d'interfaces et la sémantique associée à chacune. Dans notre cas, les interfaces sont les procédures et les fonctions et les modes de passage de paramètres sont : par valeur (entrée), par résultat (sortie) et par valeur/résultat (entrée/sortie).

- adopter une syntaxe pour définir les types et déclarer des procédures distantes dans un langages donné.

La spécification d'un langage est composée de plusieurs lignes. Chaque ligne comporte une commande interprétable par le générateur de stubs et un texte. Il existe quatre commandes qui sont les suivantes:

> ecrire : le texte suivant cette commande est reproduit intégralement par le générateur de stubs.

> ecrirevar : permet au générateur de stubs de reproduire le texte tout en remplaçant les mots préfixés par '$' par les variables correspondantes indiquées dans la déclaration d'interface.

> boucle : indique que le texte de la ligne suivante doit être reproduit autant de fois qu'il y a de paramètres.

> séparateur : indique la fin de la boucle et donne le séparateur de fin de ligne.

Exemple:

Pour déclarer l'entête d'une procédure stub client, la spécification du langage Pascal devra contenir les lignes suivantes:

ecrire (*Déclaration de l'entete de la procédure stub client*)

ecrirevar $nom-proc(

boucle

ecrirevar $nom-param : $type-param;

séparateur);

Si un utilisateur veut générer le stub client d'une procédure appelée PROC ayant deux paramètres A et B de type entier et dont le mode de passage est par valeur (mode 'e'), la déclaration d'interface contient la ligne suivante:

procedure PROC 2 A integer e B integer e.

Utilisant cette déclaration et la spécification du langage Pascal, le générateur de stubs produit les lignes suivantes:

(*Déclaration de l'entête de la procédure stub client*)

procedure PROC(A,B:integer);

ainsi que le code permettant de réaliser les différentes fonctions du stub client.

III.3.4.2 Contenu des messages d'appel et de retour

Un message d'appel contient les paramètres d'appel de la procédure distante. Il n'est pas nécessaire de spécifier le nom de cette procédure car les échanges de messages entre les deux entités distantes se feront en utilisant le numéro de canal qui résulte de l'établissement du lien. Puisque chaque procédure possède son propre stub serveur l'invocation du stub serveur implique l'invocation de cette procédure.

Un message retour ne contient que les arguments dont les valeurs ont changé; c'est-à-dire ceux dont le mode de passage est par résultat ou valeur/résultat.

Le réseau étant homogène, il n'est pas nécessaire de convertir la valeur des paramètres d'appel et de retour dans une représentation externe. Un paramètre d'appel ou de retour est donc représenté dans un message en donnant uniquement son type et sa valeur.

I.I.4 Conclusion

Le générateur de stubs est entièrement écrit en langage Pascal. Les deux spécifications décrites jusqu'à présent sont celles du langages C et Pascal. Ces deux langages présentent sensiblement le même ensemble de types de données.

La connexion de nouveaux types de machines nécessitera la mise en oeuvre d'une représentation externe et l'écriture des spécifications de machines. En outre, des commandes de codage et de décodage doivent être rajoutées aux spécifications de langages.

CONCLUSION

Le problème majeur lors de la conception d'applications réparties est de rendre transparents tous les problèmes inhérents à la répartition. Les différentes études menées jusqu'à ce jour, ont montré que l'appel de procédure à distance est un outil puissant et efficace pour la mise en oeuvre d'applications réparties. En effet, ce procédé fournit un degré de transparence tel que le programmeur ignore tout des détails de la répartition.

La contribution de cette étude dans ce domaine a consisté principalement en:

> la mise en évidence des différents aspects des systèmes répartis ainsi que les différents problèmes inhérents à la répartition afin de déduire les fonctions nécessaires à l'exécution cohérente d'un traitement réparti, à savoir le contrôle du traitement, la synchronisation des échanges et la communication. Cela a conduit à la définition d'une architecture fonctionnelle en couches, proche de celle de l'ISO.

> la conception d'un service assurant le contrôle du traitement réparti. Ce service offre l'abstraction d'un service procédural. Les différents aspects de la répartition sont pris en charge par les stubs qui sont invoqués lors d'un appel et retour. Ces stubs sont produits par un générateur de stubs, conçu indépendamment des langages et des types de machines. La génération d'un stub nécessite la donnée de la spécification du langage dans lequel il sera généré, la spécification de sa machine de résidence ainsi que la description de la procédure concernée par l'appel distant.

> la validation du service conçu. Cette phase est nécessaire avant l'étape d'implantation. En effet, l'application de techniques formelles pour vérifier le service, a permis de détecter les erreurs de conception et a prouvé que l'implantation est possible. Cette vérification a été effectuée au moyen de l'outil PIPN qui permet l'analyse des systèmes modélisés à l'aide des réseaux de Petri étiquetés.

> l'implantation du service d'appel de procédure à distance, à titre expérimental, dans un environnement homogène composé de machines VAX, tout en lui laissant de grandes possibilités d'extension.

En résultats de cette étude, nous tenons à souligner l'importance de la méthodologie de conception. Cette méthodologie consiste en la définition d'une architecture selon

une démarche descendante (traitement, contrôle, communication), définition du protocole à réaliser à partir du service à offrir et du service utilisé, vérification des propriétés du protocole et du service et enfin implantation du service.

Par rapport à l'existant, l'apport de cette étude consiste principalement en la spécification formelle et la vérification du protocole d'appel de procédure à distance et du service fourni à l'aide d'un outil formel d'analyse. En outre, le service d'appel de procédure à distance décrit n'est pas spécifique à un langage donné ou à un type de machine. En effet, il peut être adapté à n'importe quel environnement. D'autre part, la mise en œuvre d'un générateur de stubs permettant de produire automatiquement le stub client et le stub serveur à la demande, confère au service un degré de transparence non négligeable.

Afin d'enrichir ce service, plusieurs extensions peuvent être envisagées, nous citons :

> spécification formelle, vérification et implantation d'un service prenant en compte les appels parallèles,

> implantation de ce service sur un réseau hétérogène par la mise en œuvre d'une représentation externe, le retrait des fonctions de codage et de décodage dans les spécifications de langage et l'écriture de la spécification de chaque type de machine,

> intégration du service d'appel de procédure à distance dans l'éditeur de liens afin de le rendre encore plus transparent. En effet, cela évitera aux utilisateurs de demander explicitement la génération de stubs.

BIBLIOGRAPHIE

[Alg 82] B.Algayres, "Sur la Modélisation, la Validation et l'Implantation d'un Protocole de Transport", Thèse de Docteur Ingénieur, Université Paul Sabatier, Toulouse. 1982

[Alm 85] G.T.Almes, A.P Black, E.D.Lazowska, and J.D.Noe, The Eden System: Technicale View", in IEEE Trans. on Sof. Eng., vol. SE-11, pp. 13-59 Janvier 1985

[78] T.Anderson, P.A Lee, S K.Shrivastava, "Model of Recoverability in Multilevel Syatems", IEEE Trans. on Soft. Eng. vol. SE-4, no 6. November 1978

[85] J M Ayache, J.P.Courtiat, M.Diaz, C.Juanole, "Utilisation des Réseaux de Petri pour la Modélisation et Validation de Protocole", TSI, vol. 4, no. 1, 1985

[Ayo 87] M.Ayoub, "Sur la Représentation des Données dans les Réseaux Hétérogènes", Laboratoire d'Automatique et d'Analyse des Systèmes, Rapport L.A.A.S no. 87106, Mars 1987

[Aze 88] P.Azema, B.Berthomieu, J.P.Courtiat, M.Diaz, G.Juanole B.Pradin-Chezalviel, "Spécification formelle et Conception des Syatèmes Informatiques Distribues", Laboratoire d'Automatique et d'Analyse des Systèmes du C.N.R.S. 7, avenue du Colonel Roche 31077 Toulouse Cedex-France, 1988

[Bal 85] E.Balkovich, S.Lerman, R.P.Parmelee, "Computing High Education: The Athena Experience", Com. Of the ACM, vol. 28, no. 11, November 1985

[Bar 85] A.Barak, A.Litman, "MOS: A Computer Distributed Operating System", *Software-Practice and Experience*, vol. 15(8), August 1985

[Ber 87] B.N.Bershad, D.T.Ching, E.D.Lazowska, J.Sanislo, M.Schwarz, "A Remote Procedure Call Facility for Interconnecting Heterogeneous Computer Systems", IEEE Trans. on Soft. Eng., vol. SE-13, no. 8, August 1987

[Bir 82] A.D.Birrell et al., "Grapevine: an exercise in ditributed computing", Communications of the ACM, vol. 25, no. 4, April 1982

[Bir 84] A.D.Birrell, B.J.Nelson, "Implementing Remote Procedure Calls" in ACM Transaction on Computer Systems, vol. 2, no. 1, Fevrier 1984

[Bir 85] A.D.Birrell, "Secure Communicating Using Remote Procedure Calls" ACM Trans. on Computer Systems, vol. 3, no. 1, Février 1985

[Boc 78] G.V.Bochmann, "Finite State of Communication Protocols", in Computer Networks, vol. 2, 1978

[Bou 88] S.Bouallag, N.Semaoune, N.Taboudjemat, "SAFIR: Système d'Accès aux Fichiers Répartis, Conception et Réalisation", Thèse d'Ingéniorat, Université des Sciences et de la Technologie Houari Boumédienne, Institut d'Informatique, Juin 1988

[Bra 83] G.W.Brams, "Réseaux de Petri: théorie et pratique", Masson, 1983

[Car 84] B.E.Carpenter, R.Cailliau, "Experience with Remote Procedure Calls in a Real-Time Control System", Software-practice and Experience, vol. 14(3), September 1984

L.M.Casey, "Comments on the Design of Efficient Reliable Remote Procedure Call Mechanism", IEEE Trans. on Computers, vol. C-35, no. 3, March 1986

[Cle 88] J.G.Cleaveland, "Building Application Generators", IEEE Software, July 1988

[Cor 84] Cornafion (Nom collectif), "Systèmes Informatiques Répartiss: concepts et techniques", Dunod informatique, Août 1984

[Cou 84] J.P.Courtiat, J.M.Ayache, S.Algayres, " Petri Nets Are Good For protocols", Laboratoire d'Automatique et d'Analyse des Systèmes du C.N.R.S. 7, avenue du Colonel Roche 31077 Toulouse-France. 1984

[Dig 82] Digital, "VAX: Technical Summary", 1982

[Dig 84] Digital, "VAX/VMS: Record Management Services Manual", vol. 5B, September 1984

[Dia 85] M.Diaz, P.Mondain-Monval, "Appel de Procédure à Distance dans les Réseaux Hétérogènes", Acte du colloque C3 Angouleme, Septembre 1985

[Ger 80] S.L.Gerhart et al, An Overview of : A Specification and Verification System", Proc of the IFIP Congress, 1980

[Gib 87] P.B.Gibbons, "A Stub Generator for Multilanguage RPC in Heterogeneous Environments", IEEE Trans. on Soft Eng., vol. SE-13, no. 13, Janvier 1987

[Gif 88] D.K.Gifford and N.Glasser, "Remote Procedures for Efficient Distributed Communication", ACM Trans. on Computer Systems, vol. 6, no. 3, August 1988

[Her 82] M.Herlihy and B.Liskov, "A Value Transmission Method for Abstract Data Types", IEEE Trans. on Prog. Lang. and Systems, vol. 4, no. 4, October 1982

[Hoa 78] C.A.R.Hoare, "Communication Sequential Processes" Communications of the ACM, August 1978

[Jua 84] G.Juanole, B.Algayres, J.Dufau, "On Communication Protocole Modelling and Design", in Advances in Petri Nets 1984, Lecture Notes in Computer Sciences, no.188, Springer-Verlag

[Kel 76] R.M.Keller, "Formal Verification of Parallel Programs", Com. ACM, vol. 19, no. 7, Juillet 1976

[Kra 87a S.Krakowiac, "Systèmes d'Exploitation Répartis", TSI vol. 6, no. 2, 1987

[Kra 87b] S.Krakowiak, "Introduction aux Systèmes Informatiques Répartis", Université Scientifique, Technologie et Médicale de Grenoble, Institut National Polytechnique de Grenoble 1987

[Lad 88] R.Ladin, B.Liskov, L.Shrira, "A Technique for Contructing Highly Available Services", Algorithmica, vol. 3, 1988

[Lam 81] B.W.Lampson, "Remote Procedure Calls in Distributed Systems, Architecture and Implementing", Lecture Notes in Computer Science, vol. 105, Springer-Verlag, 1981

[Lam 78] L.Lamport, "Time, Clocks and Ordering of Events in Distributed Systems", ACM, vol. 27, no. 7, Juillet 1987

[Lam 84] L.Lamport, "Using Time Instead Timeout for Fault Tolerant Distributed Systems", ACM Trans. on Prog. Lang. and Systems, vol. 6, no. 2, April 1984

[Leg 88a] J.Legatheaux Martins, Y.Berbers, "La Désignation dans les Systèmes d'Exploitation Répartis", TSI vol. 7, no. 4, 1988

[Leg 88b] J.Legateaux Martins, "La Désignation et l'Edition de Liens dans les Systèmes d'Exploitation Répartis", Thèse de Doctorat, Université de Rennes I, Juillet 1984

[Lin 85] K.J.Lin, J.D.Gannon, "Atomic Remote Procedure Call", IEEE Trans. on Software Engineering, vol. SE-11, no. 10, October 1985

[Lis 82] B.Liskov, "On Linguistic Support for Distributed Programs", IEEE Trans. Software Eng., vol. SE-8, pp. 203-210, Mai 1982

[Mak 89] M.M.Makpangou, "Protocoles de Communication et Programmation par Objets: l'exemple de SOS", Thèse de Doctorat, Université de Paris-VI, Février 89

[Mon 87] P.Mondain-Monval, "Conception, et Vérification d'un Service d'Appel de Procédure à Distance dans les Réseaux Hétérogènes", Thèse de Doctorat de 3ème Cycle Université Paul Sabatier, Toulouse, Novembre 1987

[Nel 81] B.J.Nelson, "Remote Procedure Call", Thèse de Ph.D, Carnegie Mellon University, CMU-CS-81-119, 1981

[Opp 83] D.Oppen, Y.K.Dalal, "The Clearinghouse: A Decentralized Agent for Locating Named Objects in a Distributed Environment", ACM Trans.on Office Information Systems, vol. 1, no.3, July 1983

[Pap 88] M.Papageorgiou, "Le Système de Gestion de Fichiers Répartis dans Chorus", TSI, vol. 7, no. 4, 1988

[Raz 80] R.M.Razouk, G.Estrin, "Modelling and Verification of Communication Protocols in SARA: the X-21 interface", IEEE Trans. on Computers, vol. C-29, no. 12, December 1980

[Rou 85] J.L.Roux, "Modélisation et Analyse des Systèmes Distribués par les Réseaux de Petri Temporels", Thèse de Docteur Ingénieur, INSA, Toulouse Décembre 1985.

[Sat 86] M.Satyanarayanan, E.H.Siegel, "MULTIRPC, a Parallel Remote Procedure Call Mechanism", Carnegie Mellon University, CMU-CS-86-139 Technical Report, August 1986

[Sha 84] M.Shapiro, "Le Service d'Appel de Procédure Distante Girolle", TSI, vol. 3, no. 6, 1984

[Shr 81] S.K.Shrivastava, "Structuring Distributed Systems for Recoverability and Crash Resistance", IEEE Trans. on Soft. Eng., vol. SE-7, no. 4, July 1981

[Shr 82] S.K.Shrivastava, F.Panzieri, "The Design of Reliable Remote Procedure Call Mechanism", IEEE Trans. on Computers, vol. C-31, no. 7, July 1982

[Spe 82] A.Z.Spector, "Performing Remote Operations Efficiently on a Local Computer Network", CACM vol. 25, no. 4, Avril 1982

[Sta 82] J.A.Stankovic, "Software Communication Mechanisms Procedure Calls Versus Messages", IEEE Trans. on Computers, Avril 82

[Sun 86] RPC programming reference manual, revision B of February 1986

[Sun 86] External Data Representation Protocol Specification reference manual, revision B of February 1986

[Svo 84] L.Svobodova, "Resilient Distributed Computing", IEEE Trans. on Soft. Eng., vol. SE-10, no. 3, May 1984

[Tad 89] M.Tadao "Petri Nets: Properties, Analysis and Applications", Proceedings of the IEEE, vol. 77, no. 4 Avril 1989

[Tan 85] A.S.Tanenbaum and R.Van-renesse, "Distributed Operating Systems", Computing Surveys, vol. 17, no. 4, December 1985

[Wh- 86] K.White, "An Implementation of a Remote Procedure Call Protocol in the Berkeley Unix Kernel", Report no. UCB/CSD86/249, Progress Report no. 86/3, Computer Science Division (EECS) University of California Berkley, June 1986