MASTER EN SCIENCES INFORMATIQUES

Spécifications d'un agent de transfert de messages dans le cadre d'une messagerie électronique de type X400 : définition, conception et intégration de traitements parallèles en vue d'une implémentation sous Unix version 7.

Angelot, Marie-Elise; Delroisse, Thierry; Franssens, Christine

Award date:
1987

Awarding institution:
Universite de Namur

FACULTES
UNIVERSITAIRES
N.D. DE LA PAIX

NAMUR

INSTITUT D'INFORMATIQUE
RUE GRANDGAGNAGE, 21, B - 5000 NAMUR (BELGIUM)

Spécifications d'un agent de transfert de messages dans le cadre d'une messagerie électronique de type X400 ; définition, conception et intégration de traitements parallèles en vue d'une implémentation sous Unix version 7.

Mémoire présenté par

Marie-Elise ANGELOT
Thierry DELROISSE
Christine FRANSSENS

en vue de l'obtention

du titre de

Licencié et Maître en Informatique

Promoteur : Ph. van BASTELAER

Année académique 1986-1987

TOME I

Au terme de ce mémoire, nous tenons à adresser nos remerciements à Monsieur Pr. van Bastelaer pour avoir accepté la direction de ce mémoire et nous avoir proposé ce sujet intéressant.

Nous désirons assurer Serge Simonet de notre profonde gratitude. Ses conseils judicieux, sa grande disponibilité, sa patience à toute épreuve et son sourire ont été pour nous un soutien solide tout au long de ce travail.

Notre reconnaissance se porte également sur tous ceux qui, de près ou de loin, nous ont aidés à mener à bien ce mémoire, avec une intention toute particulière pour Isabelle, Pascal et Pierre. Sans leur bonne volonté et leur gentillesse, jamais ce rapport n'aurait pu voir le jour.

LISTE DES ABREVIATIONS.

ADMD : ADministration Management Domain
(domaine de gestion d'une administration)

AM : Association Manager
(gestionnaire des associations)

$\overline{AM}$: pipe allant de l'AM au GI-MTA-MTA

$\underline{AM}$: pipe allant du GI-MTA-MTA à l'AM

CCITT : Comité Consultatif pour la Télégraphie et la Téléphonie

GC : Gestionnaire Central

GI-MTA-MTA : Gestionnaire d'Interaction MTA-MTA

GI-UA-MTA : Gestionnaire d'Interaction UA-MTA

ISO : International Standards Organization
(Organisation pour les Standards Internationaux)

MD : soit Management Domain (domaine de gestion)
soit Message Dispatcher (gérant des messages)
(suivant le contexte)

$\overline{MD}$: pipe allant du Message Dispatcher vers le GI-UA-MTA

$\underline{MD}$: pipe allant du GI-UA-MTA vers le Message Dispatcher

MHS : Message Handling System
(système de traitement de messages)

MPDU : Message Protocol Data Unit
(unité de données de protocole pour la messagerie)

MTA : Message Transfer Agent
(Agent de transfert de messages)

MTAE : Message Transfer Agent Entity
(entité de l'agent de transfert de messages)

MTL : Message Transfer Layer
(niveau du transfert des messages)

MTS : Message Transfer System
(système de transfert de messages)

OSI : Open Systems Interconnection
(interconnexion de systèmes ouverts)

P1 : Protocole de transfert de messages

P2 : Protocole entre UA

P3 : Protocole d'envoi et de remise

PDU : Protocol Data Unit
(unité de données de protocole)

PRMD : PRivate Management Domain
(domaine de gestion privé)

RTS : Reliable Transfer Server
(agent de transfert fiable)

$\overline{\text{RTS}}$: pipe allant d'un RTS au GI-MTA-MTA

$\underline{\text{RTS}}$: pipe allant du GI-UA-MTA à un RTS

SDE : Submission and Delivery Entity
(entité d'envoi et de remise)

SMPDU : MPDU de service (sonde ou notification)

UA : User Agent
(agent de l'utilisateur)

UAE : User Agent Entity
(entité de l'agent utilisateur)

UAL : User Agent Layer
(niveau de l'agent utilisateur)

UAPDU : User Agent Protocol Data Unit
(unité de donnée de protocole de l'agent utilisateur)

UMPDU : MPDU d'usager

TABLE DES MATIERES.

XI. Conception des modules fonctionnels de l'architecture parallèle.

XI.1. Préliminaires.

XI.2. Le gestionnaire d'interaction UA-MTA (GI-UA-MTA).

11.2.1. Introduction.

11.2.2. Pourquoi un GI-UA-MTA ?

11.2.3. Elaboration progressive du GI-UA-MTA.

11.2.4. Messages transitant par le GI-UA-MTA.

11.2.5. En résumé.

XI.3. Le Message Dispatcher.

11.3.1. Introduction.

11.3.2. Le diagramme d'états.

11.3.3. L'algorithme.

11.3.4. Les procédures.

XI.4. Le gestionnaire d'interaction MTA-MTA (GI-MTA-MTA).

11.4.1. Introduction.

11.4.2. Modules interagissant avec le GI-MTA-MTA.

11.4.3. Construction progressive du gestionnaire d'interaction MTA-MTA.

XI.5. L'Association Manager.

11.5.1. Introduction.

11.5.2. Le diagramme d'états.

11.5.3. L'algorithme.

11.5.4. Les procédures.

XI.6. Le Reliable Transfer Server.

11.6.1. Introduction.

11.6.2. Le RTS, utilisateur des services de la couche session

11.6.3. Le RTS au sein de l'architecture parallèle.

11.6.4. Conclusion.

Chapitre I.

INTRODUCTION

Que ce soit via le système postal, le téléphone, le télex, la radio, la télévision ou le "bouche à oreille", chacun de nous se voit confronté à un flot d'informations surgissant de toutes parts.

Avec la prolifération des ordinateurs et le développement des réseaux, un nouveau moyen de communication a vu le jour : la messagerie électronique. Elle permet aux utilisateurs des moyens informatiques de correspondre entre eux, d'échanger leurs points de vue, de se transmettre des informations, d'aborder ensemble les sujets qui les intéressent, et même de collaborer, tout cela grâce à l'échange de messages de type quelconque, par le biais d'un équipement de télécommunication approprié.

Depuis une dizaine d'années, dans la plupart des pays, différents organismes ont ainsi conçu et réalisé des systèmes offrant des services de traitement de messages. N'étant soumis, au départ, à aucune réglementation, ces systèmes se caractérisent par des objectifs spécifiques et des avantages particuliers offerts à leurs utilisateurs. Ceci entraîne des différences fondamentales en ce qui concerne le type d'information traitée et la manière de la traiter.

Petit à petit, le succès toujours grandissant de ces services de transmission de messages, ainsi que l'impossibilité de communication entre deux systèmes différents, ont conduit à une évidence : il était nécessaire d'évoluer vers une normalisation qui permettrait l'établissement d'un système de messagerie mondial. Et en 1984, les travaux du CCITT (Comité Consultatif International pour la Téléphonie et la Télégraphie) ont donné naissance aux recommandations X400 qui, depuis, régissent les protocoles de communication entre les systèmes de traitement de messages.

Dans le modèle fonctionnel que décrivent les recommandations, notre étude s'est attachée à l'agent de transfert des messages (MTA) qui assure, comme son nom l'indique, les fonctionnalités de transfert de messages, à savoir le relais et la livraison de messages en passant par le choix du chemin à suivre à travers le réseau.

Après une année de recherche, ce mémoire présente une description détaillée et les spécifications complètes d'une entité MTA. Au-delà de ce travail d'analyse approfondie, nous proposons une approche personnelle du MTA.

En effet, la messagerie électronique s'inscrit à souhait dans la ligne des systèmes multi-utilisateurs : il est raisonnable de penser qu'à un moment donné, de nombreux utilisateurs désirent activer les services que leur offre un système de traitement de messages. Pour ce faire, une interaction s'établit entre chacun d'eux et un MTA, via un UA. Si bien qu'un MTA peut être amené à s'occuper simultanément du transfert et du relais de plusieurs messages, ce qui implique un grand nombre d'actions. Dès lors, pour éviter les attentes trop longues, l'idée nous est venue de proposer une extension du MTA. Elle repose sur l'intégration de traitements parallèles en vue de remplir les différentes fonctions prévues par X400.

Dans ce cadre, une architecture susceptible d'abriter la notion de parallélisme est construite et justifiée. Pour terminer, les spécifications élaborées pour les modules les plus importants de l'architecture parallèle conduisent à des algorithmes en pseudo-langage qui pourront servir de guide pour une implémentation future.

En tant que compte rendu de nos recherches, ce rapport s'articule comme suit : dans un premier temps les notions de base en télécommunication sont introduites au chapitre II. Ensuite vient la description du modèle fonctionnel présenté dans X400 (chapitre III), ainsi que les services qu'un MTA se doit d'offrir (chapitre IV). Le

reste de ce rapport reflète les étapes successives de notre travail, sur base de la méthodologie de développement de logiciels présentée au chapitre V. Ainsi le sixième chapitre contient la description d'un MTA en tant que "boîte noire" par le biais de spécifications qui utilisent plusieurs formalismes. Suit alors la composition de la sous-couche de la couche application qui renferme les MTA du système X400 (chapitre VII). Ceci permet la conception d'une architecture logique appropriée (chapitre VIII). On trouve ensuite les spécifications des modules fonctionnels qui se placent au coeur de l'architecture (chapitre IX). Celle-ci va évoluer vers une nouvelle formule, susceptible d'abriter la notion de parallélisme. Cette nouvelle architecture parallèle est justifiée au chapitre X. Tous les modules fonctionnels, ceux qui remplissent les tâches du MTA offertes par X400 aux utilisateurs ainsi que ceux qui apparaissent avec le parallélisme sont spécifiés avec soin grâce à des diagrammes d'états et sont matérialisés sous forme d'algorithmes en pseudo-langage (chapitre XI).

Pour terminer, la liste des choix et hypothèses posés en cours de conception est dressée au chapitre XII, tandis que le suivant (chapitre XIII) relève les problèmes et imprécisions apparus dans la norme. Et le tout est clôturé par une évaluation des démarches effectuées et des résultats obtenus dans le cadre de ce mémoire.

Chapitre II.

NOTIONS DE BASE EN TELEMATIQUE ET TELECOMMUNICATIONS

2.1. INTRODUCTION.

La télématique marque la rencontre de deux démarches : les télécommunications et l'informatique. En effet, l'objectif de la télématique est de transmettre à distance, par des voies de communication, des informations traitées au départ et à l'arrivée par l'informatique. La plupart des applications télématiques se déroulent selon le schéma suivant :

- le codage de l'information i.e. représentation de l'information sous une forme utilisable par l'ordinateur.
- l'adaptation aux techniques de télécommunications i.e. conversion des données en signaux utilisables par le réseau de télécommunications qui va les véhiculer.
- la propagation des signaux.
- la réadaptation des signaux en données informatiques.
- le traitement de l'information tel que affichage à l'écran du message reçu, exécution d'un programme utilisant le fichier envoyé,...

Ce mariage ordinateur et télécommunications au sein de vastes réseaux de transmission de données a profondément influencé l'organisation de systèmes informatiques. La salle " centre informatique " se voit remplacée par un réseau informatique i.e. un ensemble d'ordinateurs dispersés, connectés entre eux et bénéficiant d'une plus grande autonomie .

Certains besoins d'utilisation sont à dénoter comme facteurs d'évolution de la télématique :

- une entreprise possédant un grand nombre d'ordinateurs autonomes, disséminés dans le monde peut souhaiter les interconnecter afin de disposer à tout endroit et à tout moment, des informations générales concernant toutes les compagnies. Programmes et données peuvent être disponibles à chaque entreprise quelle que soit sa position géographique;

- dans certains domaines tels que banques, sociétés de courtage, bases militaires, les réseaux sont considérés comme une sécurité supplémentaire en cas de défaillance technique.

Il faut compter aussi avec le phénomène " microprocesseur " qui tend de plus en plus à élargir ses domaines d'application, notamment en raison de son rapport qualité/prix. Des systèmes distribués et indépendants sont adoptés.

2.2. LES RESEAUX

Après avoir abordé ces notions générales, il est temps de voir comment s'établit une liaison entre deux utilisateurs et comment l'information est transportée de l'un vers l'autre.

La solution la plus simple consisterait à établir une ligne reliant directement émetteur et récepteur. Une telle représentation peut s'intituler "réseau avec connectivité complète et directe", réseau signifiant un

ensemble de techniques et de moyens matériels mis en oeuvre pour relier entre eux des utilisateurs (cfr. fig. 2 . 1). Cette configuration est tout de suite rejetée vu le nombre de lignes à installer (pour n usagers, il faudrait (n * (n - 1)) / 2 lignes). Qui plus est, une ligne ne serait pas très rentable car deux utilisateurs seulement en feraient usage et si une ligne était coupée, les deux utilisateurs concernés ne sauraient plus communiquer. Pour de telles raisons, la topologie des réseaux se voit modifiée et un réseau dit "avec connectivité complète mais non directe" (fig. 2.2) est préféré et adopté au détriment de celui avec connectivité complète et directe (fig. 2.1).

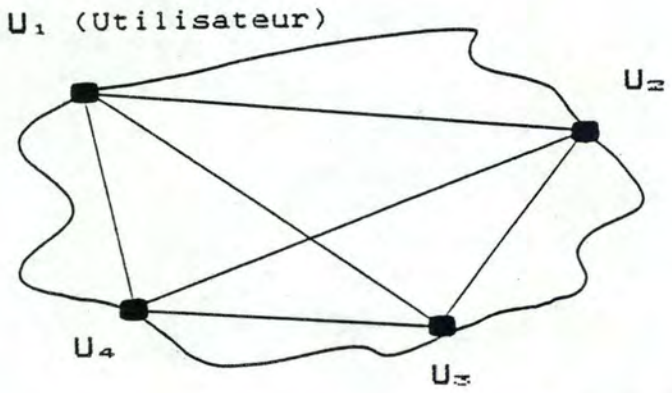


FIG. 2.1 : RESEAU AVEC CONNECTIVITE COMPLETE ET DIRECTE

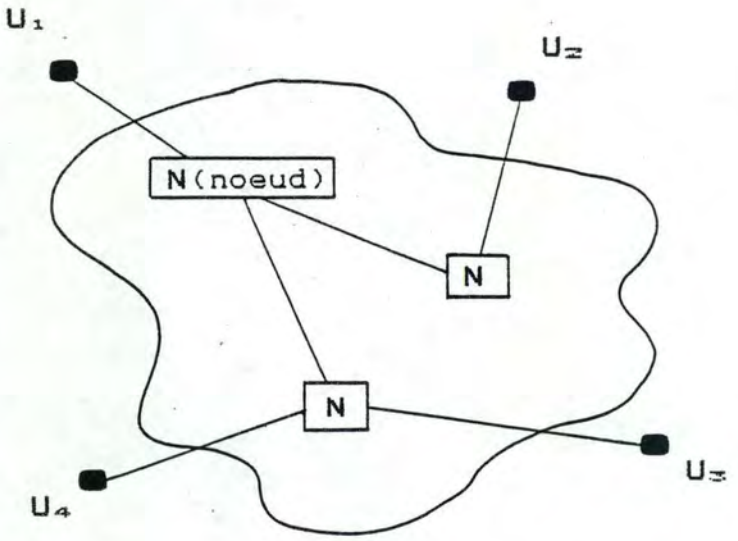


FIG. 2.2 : RESEAU AVEC CONNECTIVITE COMPLETE ET NON DIRECTE.

Un tel réseau public de communication est par exemple en Belgique, DCS (Data Communication System).

Selon la façon de router un message travers le réseau, une typologie de réseau peut être dressée [PVB 86]:

- " message switching "
- " circuit switching "
- " packet switching "

Un réseau est qualifié de " message switching " lorsque une fois l'adresse du destinataire inscrite dans le message, celui-ci est dirigé vers ce destinataire en passant de noeud en noeud. Pour ce faire, chacun de ceux-ci stocke le message, choisit le noeud suivant sur la route et dirige vers lui le message.

Un réseau " circuit switching " est fondamentalement différent du précédent. En effet,

l'utilisateur émetteur doit au préalable signaler qu'il veut communiquer avec un destinataire bien précis. Suite à cela le réseau réserve un chemin spécifique vers l'utilisateur récepteur. Une connexion est alors créée entre les deux utilisateurs qui peuvent en disposer comme ils l'entendent.

Le troisième type de réseau, dit " packet switching " est celui adopté par exemple par DCS. Il procède de la façon suivante. Avant l'envoi vers le réseau, chaque message est "découpé" en unités plus petites, de l'ordre de quelques centaines de bytes, nommées paquets. Chacune d'elles est considérée séparément par les noeuds du réseau qui vont décider la meilleure voie à suivre pour que le paquet atteigne sa destination. Puisque la charge et les problèmes rencontrés sur le réseau peuvent varier dans le temps, il se peut que tous les paquets n'empruntent pas le même chemin.

2.3. LA NORMALISATION

Comme le mentionne Y. Le Roux dans [YLR 86] : " Le développement de la téléinformatique dû aux progrès atteints d'une part dans le domaine du traitement de l'information et d'autre part dans le domaine de la transmission de données, a nécessité très tôt sa normalisation ".

Des organismes tels que CCITT (Comité Consultatif International pour la Télégraphie et la Téléphonie) et l'ISO (International Standards Organisation) impliqués dans l'établissement de normes, ont été institués au niveau

mondial. Ces organismes prennent en charge la standardisation moyennant concertation entre les différents membres. De plus, les télécommunications étant dans une majorité de pays un monopole de l'état, un organisme par pays normalise officiellement la transmission de données. Outre ceux-ci, les principaux utilisateurs en téléinformatique ont formé des associations s'intéressant aux normes.

Voici comment le vice président de ISO [YLR 86] définit la constitution et les fonctions de ISO et du CCITT.

> Le **CCITT** est constitué essentiellement des administrations des PTT et d'organisations privées de télécommunications officiellement reconnues. Il a pour tâche de traiter des problèmes liés aux télécommunications sur cables ou supports guidés.
>
> L'**ISO** regroupe des organismes nationaux de normalisation. Ses travaux s'étendent à tous les domaines de la standardisation, à l'exception des normes concernant la technologie électrique et électronique. L'ISO tente de concilier les intérêts des fabricants, des usagers, des gouvernements et des milieux scientifiques par l'élaboration de normes internationales. Ces normes sont des propositions, et par conséquent ne sont nullement obligatoires. Des réseaux privés tels que SNA, ARPANET, ... ne vérifient pas ces règles. Cependant, depuis que l'ISO spécifie un ensemble de standards de communication pour l'interconnexion de systèmes ouverts, la tendance actuelle de certaines industries est de s'écarter de plus en plus des solutions de communication uniques au profit de l'emploi de ces standards.

Le but des communautés internationales est de fournir un ensemble très riche de spécifications de telle sorte que des organisations variées puissent satisfaire leurs besoins en communication de données en respectant les protocoles standards. Suivant les exigences de la

communication, une organisation choisit un sous-ensemble des règles et des options offertes.

2.4. LE MODELE OSI DE L'ISO.

Le modèle OSI (Open Systems Interconnection) rassemble les standards pour l'interconnexion des systèmes ouverts.

Sans vouloir une fois de plus en faire la description, rappelons-en brièvement les grandes lignes.

Le modèle de référence OSI [PVB 86] [TAN 81] est une description abstraite de la communication entre systèmes. Il est à souligner qu'il s'intéresse à l'échange d'informations aux points d'interconnexion entre systèmes et pas à la description des opérations internes d'un seul système. Celles-ci étant locales, elles ne nécessitent par conséquent pas de standardisation. Le modèle est hiérarchisé par l'application de plusieurs techniques qui produisent une structure modulaire.

Voici une brève présentation des éléments constructeurs de la technique de modularisation. Pour que deux systèmes puissent communiquer, ils doivent partager un ensemble de règles (ou encore protocoles) pour générer et interpréter les messages qu'ils envoient et reçoivent. Dans le but de créer une approche structurée, OSI a divisé les règles pour l'interconnexion en une série de couches de fonctions, chacune d'elle étant limitée par une frontière bien précise. La subdivision étant purement hiérarchique,

la structure résultante est dite "en couches". Chaque système peut être vu comme étant logiquement composé d'une succession de sous-systèmes, chaque sous-système correspondant à l'interconnexion du système avec une couche (cfr.figure 2.3)

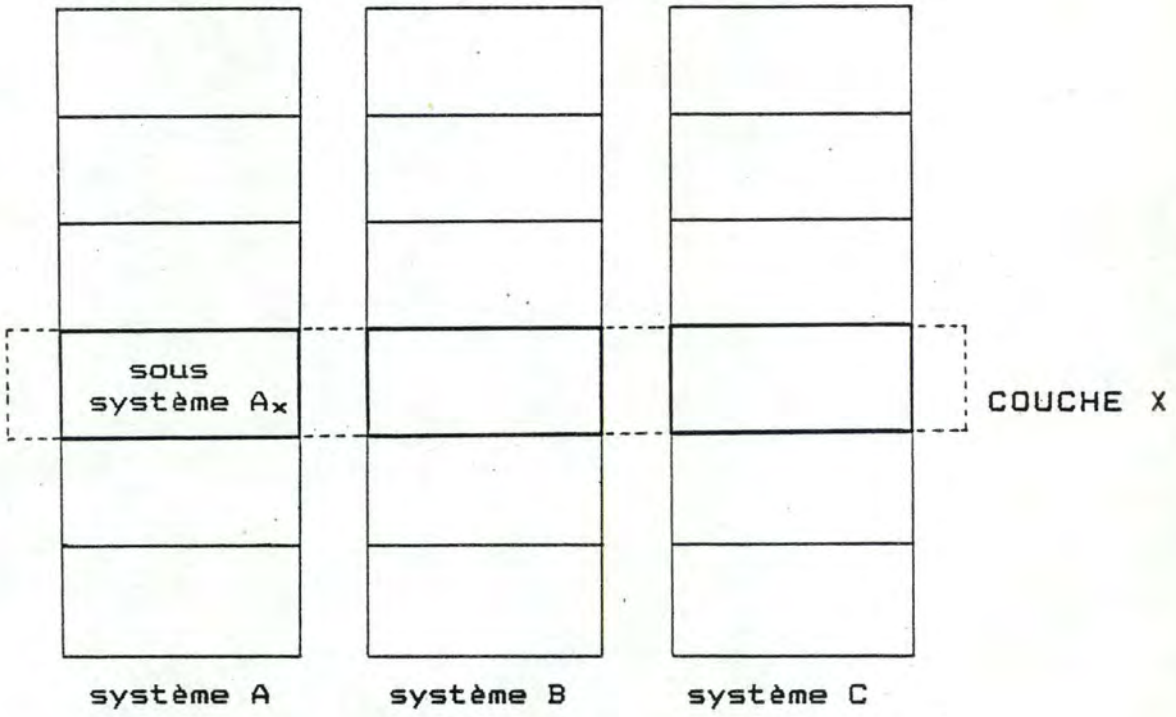


FIGURE 2.3 : DECOMPOSITION D'UN SYSTEME EN SOUS-SYSTEMES.

Une couche (ou encore un niveau) comprend plusieurs entités (ou sous-systèmes) distribuées parmi les systèmes ouverts interconnectés.

Une couche sera désignée comme étant la couche N, la couche suivante supérieure comme étant la couche N+1, et la couche suivante inférieure comme étant la couche N-1. Le concept de niveau repose sur deux principes fondamentaux.

* Le premier est d'assurer l'indépendance de chaque couche en définissant les services fournis par un niveau N (voir figure 2.4) et ignore tout ce qui

concerne la conception et la réalisation de son fournisseur (couche N).

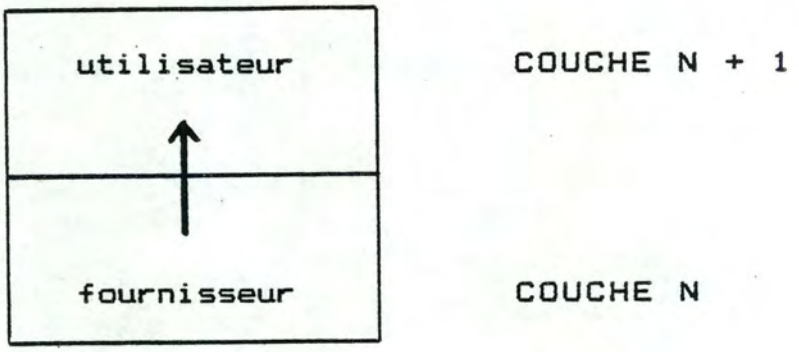


FIGURE 2.4 : SERVICES OFFERTS PAR LA COUCHE N A LA COUCHE N+1.

* Le deuxième principe réside dans le fait que chaque couche apporte une valeur ajoutée aux services fournis par les couches qui lui sont inférieures. De cette manière, le niveau le plus haut se voit offrir l'ensemble des services nécessaires pour exécuter des applications à distance.

Il est important de souligner qu'une telle découpe en niveaux a l'avantage de pouvoir réduire à un problème de taille traitable et ainsi de rendre la structure plus compréhensible. Chaque sous-problème peut être alors développé et maintenu indépendamment.

Reprenons plus en détails les notions de service et de protocole.

LE CONCEPT DE SERVICE.

En général, la définition de service établit une relation entre, d'une part, une entité jouant le rôle de fournisseur du service et, d'autre part, 1(es) autre(s) utilisateur(s) (cfr figure 2.5).

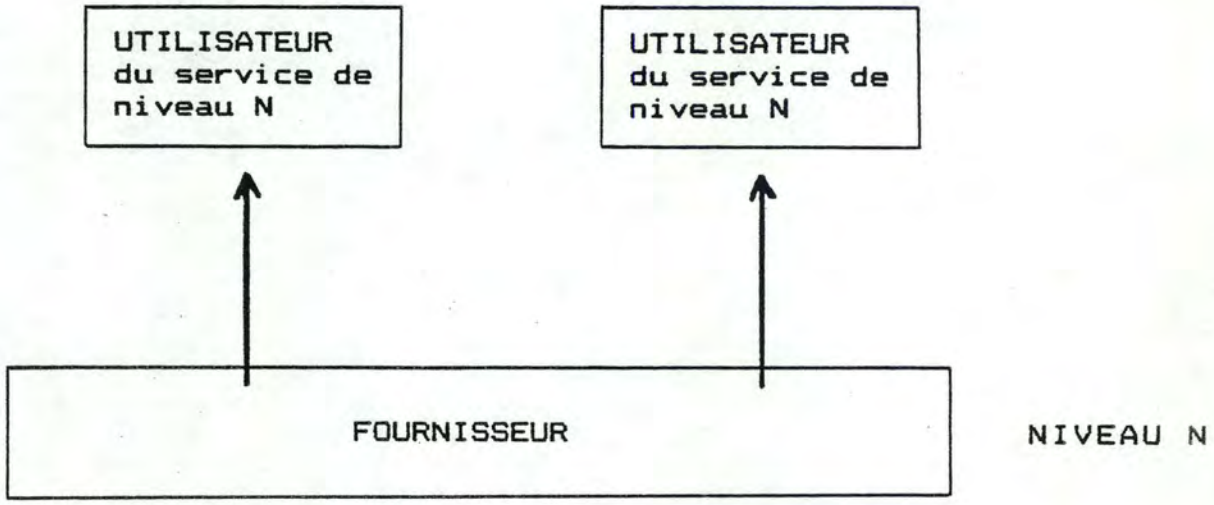


FIGURE 2.5 : SERVICE DE NIVEAU N.

L'utilisateur voit le fournisseur comme une machine abstraite fournissant des services spécifiques. Dans une telle structuration hiérarchique, on dira qu'une couche N+1 utilise la couche N si et seulement si le fonctionnement correct de la couche N+1 dépend de la disponibilité d'une version correcte de la couche N [AVL 87]. Ces services sont obtenus par l'emploi de primitives de service (i.e. interface entre deux couches). Ces primitives ne sont pas définies au bit près car elles sont locales au système et par conséquent n'interfèrent pas directement dans la communication à distance. Ce sont des conventions au sein d'un système et non des règles entre deux systèmes qui désirent se comprendre.

LE CONCEPT DE PROTOCOLE.

Une entité de niveau N "parle" toujours à une (des) autre(s) entité(s) de niveau N au moyen de services de niveau N-1. Pour se comprendre lorsqu'elles coopèrent, deux entités de même niveau (ou encore entités paires) ont besoin de règles, de conventions communément appelées protocoles.

Comme l'indique la figure 2.6,

l'entité de niveau N communique avec son entité paire conformément au protocole de niveau N

l'entité de niveau N-1 communique avec son entité paire conformément au protocole de niveau N-1

. . .

. . .

l'entité de niveau 1 communique avec son entité paire conformément au protocole de niveau 1.

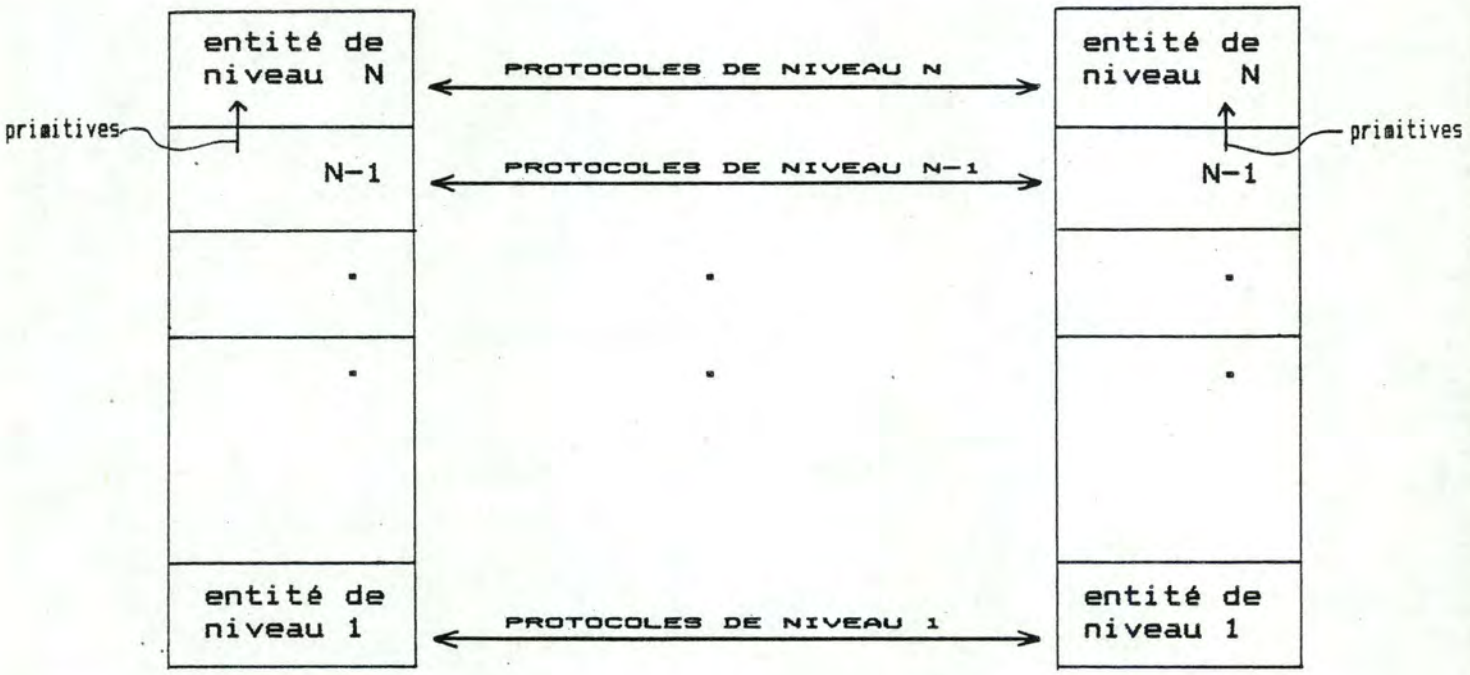


FIGURE 2.6 : PROTOCOLES ET PRIMITIVES DE SERVICE.

Les messages que deux entités paires s'échangent sont des PDU (Protocol Data Unit). En réalité, aucune donnée n'est directement transférée entre deux entités paires, excepté pour le niveau 1. Logiquement, deux entités pensent la communication comme horizontale. En fait, elle est verticale.

Ces protocoles sont définis au bit près. En effet, deux entités paires ont besoin de mêmes règles pour interpréter une unité de données de la même manière. La communication entre entités paires étant globale à un réseau, chaque système doit travailler en respectant les mêmes conventions.

Les éléments de base développés précédemment servent de blocs constructeurs dans l'élaboration du modèle de communication entre processus.

Deux systèmes quelconques désirant communiquer ne sont généralement jamais reliés physiquement par une seule connexion mais par une succession de connexions entre des noeuds dans un réseau. Chaque système est relié physiquement à un noeud du réseau appelé noeud d'entrée dans le réseau (cfr figure 2.7).

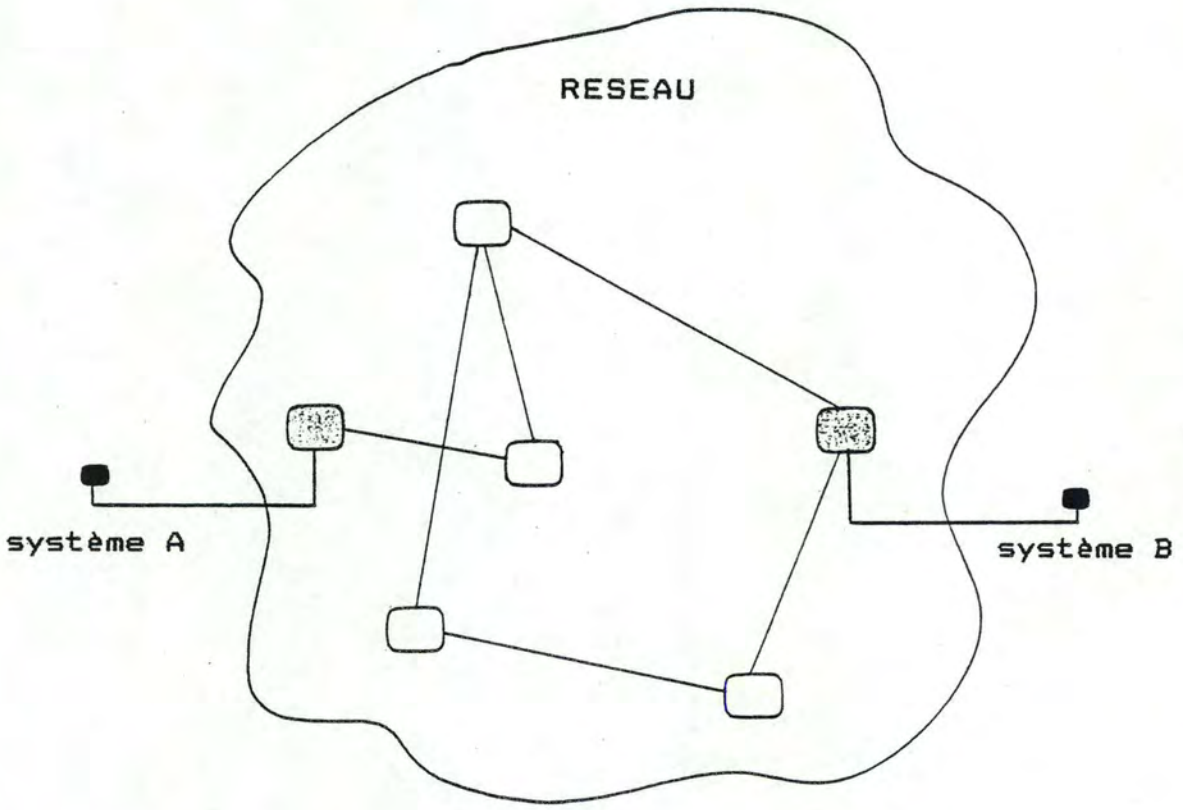


FIGURE 2.7 : SYSTEMES RELIES A UN RESEAU; NOEUD D'ENTREE.

Dans OSI, la communication entre systèmes est subdivisée en 7 niveaux.

Le niveau 1 est appelé couche physique; le deuxième, couche logique; le troisième, couche réseau; le quatrième, couche transport; le cinquième, couche session; le sixième, couche présentation; le septième, couche application.

Comme souligné dans la figure 2.8, les protocoles des niveaux 1,2 et 3 sont des règles entre système et noeud d'entrée dans le réseau, tandis que les protocoles des niveaux 4,5,6 et 7 sont établis entre les systèmes communicants origine/destination (on parle de protocoles de bout en bout).

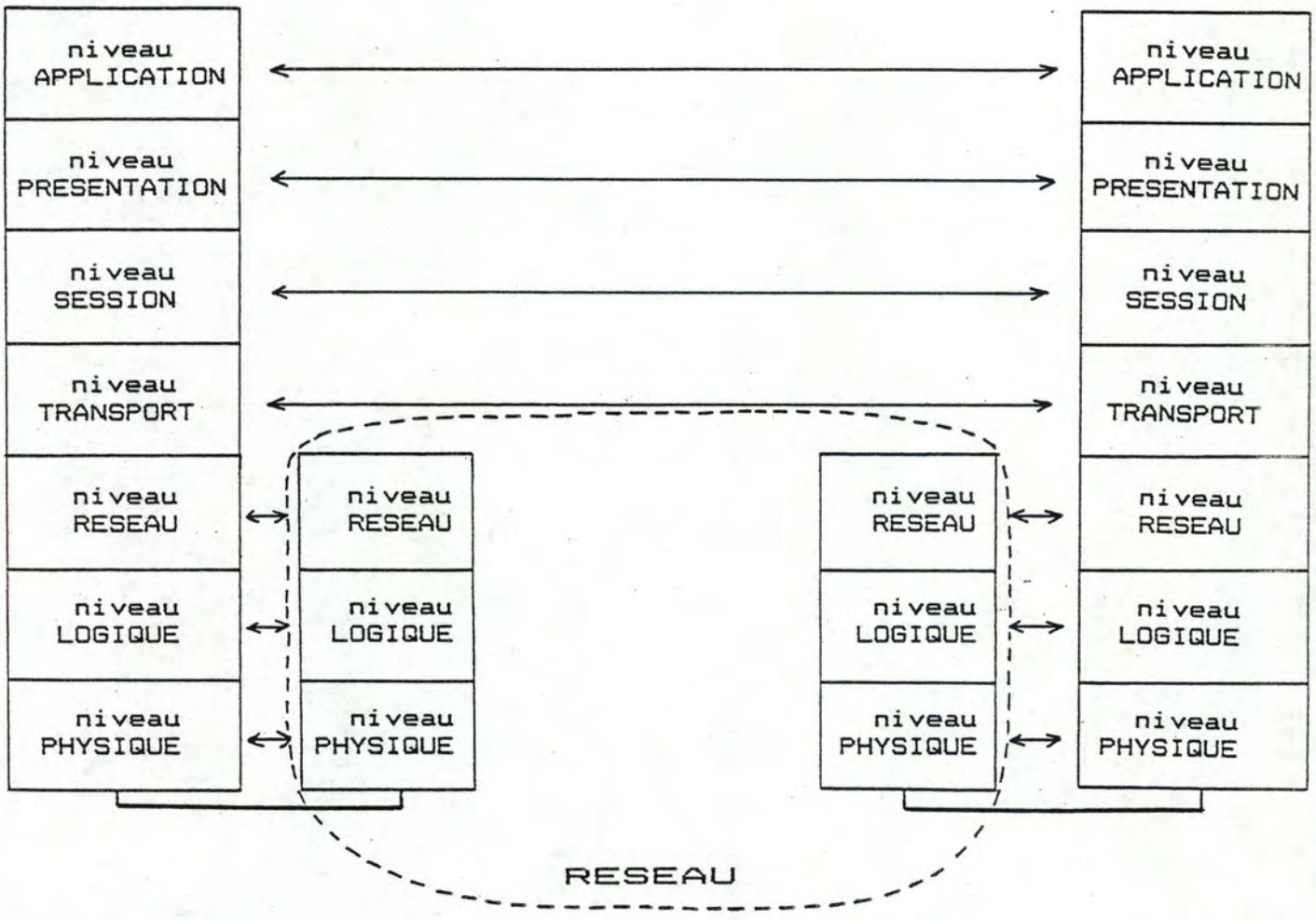


FIGURE 2.8 : LE MODELE OSI.

Ci-après suit une présentation succincte de chaque couche [TAN 81] [PVB 86].

COUCHE 1 : PHYSIQUE.

Le niveau 1 doit réaliser la transmission physique des bits via une voie de communication. Cette couche fournit des standards mécaniques, électriques, fonctionnels pour accéder au circuit physique. Elle doit activer et désactiver des connexions physiques vers d'autres entités et joue le rôle de mécanisme d'interface logique entre le moyen de transmission et la couche 2.

COUCHE 2 : LOGIQUE.

Partant des facilités de transmission offertes par le niveau 1, la couche logique a pour tâche d'offrir à la couche 3 un moyen de communication qui apparaît sans erreur de transmission. Elle gère les erreurs survenues, c'est-à-dire qu'elle les détecte et les corrige. Elle contrôle également le flux des données. Ainsi, si une des extrémités de la ligne est plus lente, elle ne sera pas submergée par les PDU lui arrivant.

COUCHE 3 : RESEAU.

La couche réseau masque à la couche 4 toutes les particularités du moyen de transfert réel. Elle permet à la couche 4 d'être indépendante de la technologie de transfert des données et des considérations d'adresse et de routage. C'est donc au niveau réseau que les notions d'adresse et de routage sont essentielles. Recevant un message, cette couche le découpe en morceaux (appelés paquets) et s'arrange pour qu'il atteigne sa destination suivant une certaine technique de routage. Il faut rappeler que les entités des sous-systèmes émetteur et récepteur ne parlent qu'avec le noeud d'accès. Le routage est donc dans le réseau. Deux types différents de service de réseau sont définis dans la norme, soient le service de réseau orienté connexion et le service de réseau orienté sans connexion.

Le service de réseau orienté connexion (encore appelé le service du circuit virtuel) est au téléphone ce que le service de réseau orienté sans connexion (ou encore service de datagramme) est à la poste. En effet, voici quelques différences fondamentales entre ces deux services :

d'une part, dans le cas du service de circuit virtuel, les deux utilisateurs ont l'impression qu'un cable leur est dédié. Ainsi, ils établissent la liaison. Une fois la communication établie, ils transmettent leur message. Ils terminent en fermant le circuit. Pour que l'illusion soit complète, il est nécessaire que le réseau assure les trois points suivants :

- l'ordre d'envoi des paquets est maintenu à la réception
- le traitement des erreurs est transparent pour les utilisateurs
- le contrôle du flux de bout en bout est effectué par le réseau

d'autre part, une couche réseau offrant le service de datagramme essaie simplement de délivrer chaque paquet et ceci de manière isolée. Cela impose de placer l'adresse du destinataire dans chaque paquet étant donné qu'aucun lien logique ne lie les deux "correspondants". Le réseau ne garantit ni l'arrivée ni l'ordre des paquets. Aucun contrôle de flux n'est effectué. Ce service peut conduire à l'obtention de paquets dupliqués, ce qui affaiblit la comparaison avec le système postal.

La sélection du noeud suivant (définie comme étant le routage) peut s'effectuer de trois manières différentes :

- soit il est basé sur des tables statiques représentant le cablage du réseau. Chaque noeud du réseau sélectionne le noeud suivant sur la route.

- soit l'usager émetteur réserve un chemin particulier au début de chaque conversation.

- soit le message est découpé au départ en plusieurs parties (paquets) et le routage est établi dynamiquement en fonction de la charge du réseau et ceci pour chaque paquet.

Il est possible de classer ces trois types de routage : le premier étant statique, le second semi-adaptatif et le dernier adaptatif.

COUCHE 4 : TRANSPORT.

La couche transport, contrairement aux couches mentionnées précédemment, est le premier niveau de protocole de bout en bout (origine-destination). Le but de la couche transport est d'offrir un transfert transparent de données entre les systèmes extrêmes, déchargeant les couches supérieures de toute notion de communication à distance. Au-dessus du niveau 4, tout va se passer comme si les machines étaient installées dans la même pièce. La couche transport permet le transfert de données avec une certaine qualité de service demandée par la couche 5. Par qualité de service, on entend, entre autres, détection et recouvrement d'erreurs et multiplexage. En fait, la couche transport comble le manque de qualité de service entre celle qui est demandée par la couche 5 et celle qui est offerte par le niveau réseau.

COUCHE 5 : SESSION.

La couche session offre un certain nombre de mécanismes de gestion du dialogue. Elle fournit des outils pour organiser et structurer les interactions entre les processus d'applications. Elle permet, par exemple, d'établir des points de synchronisation majeure et mineure, ou encore de définir des jetons afin de synchroniser l'échange. Il faut insister sur le fait que le niveau 5 offre des outils et ne synchronise ni n'organise lui-même le dialogue. En effet, le gérer signifie connaître sa "sémantique", et la couche session l'ignore.

COUCHE 6 : PRESENTATION.

La couche présentation a pour tâche de rendre les processus d'application indépendants des différences de syntaxe. En début d'interaction, les entités de la couche présentation négocient le langage commun à adopter pendant la communication. Des traductions doivent éventuellement être effectuées lors de l'échange. L'utilisateur de la couche présentation peut choisir le "contexte de présentation". Celui-ci peut être spécifique à une application, à un type de hardware ou à certaines représentations standards.

COUCHE 7 : APPLICATION.

La couche application est la plus haute couche du modèle OSI. Elle ne fournit donc pas de service à une quelconque autre couche. Le niveau 7 est propre à chaque type de problèmes. Suivant leurs besoins, les utilisateurs emploient le réseau pour des applications spécifiques ou générales telles que courrier électronique, copie de fichiers, travail à distance,... Couramment, seulement une partie de la couche application est réellement implémentée dans le système.

Suite à ces rappels sur les notions de base en télécommunications, le chapitre III va s'attarder à la couche application dans le cadre de la messagerie électronique.

Chapitre III.

LA MESSAGERIE ELECTRONIQUE DE TYPE X400

3.1 INTRODUCTION.

La messagerie électronique peut être vue comme un nouveau moyen de communication. Grâce à elle, les utilisateurs des outils informatiques peuvent dialoguer, car elle leur permet d'échanger des messages via un équipement de télécommunication approprié.

Avec le souci d'établir un système de messagerie mondial, divers organismes se sont dirigés vers des essais de normalisation. Dans ce cadre, le Comité Consultatif International pour la Téléphonie et la Télégraphie (CCITT) a publié en 1984 une liste de recommandations connues sous le nom de recommandations x400 [REC X400 84]. Elles se chargent de la standardisation des protocoles de communication entre les systèmes offrant des services de traitement de messages.

3.2 LE MODELE MHS.

3.2.1 ASPECTS FONCTIONNELS.

Les recommandations X400 décrivent le modèle MHS (message handling system) dont le principe essentiel est

la séparation des fonctions contrôlées par l'utilisateur de celles associées au transfert des messages. La figure 3.1 donne une vue d'ensemble du fonctionnement du modèle MHS.

FIGURE 3.1 : APERÇU DU FONCTIONNEMENT DU MODELE MHS.

La première classe de fonctions est fournie par une entité appelée "agent de l'utilisateur" (UA) qui assiste l'utilisateur dans la préparation, la conservation, l'envoi et la réception des messages.

La responsabilité du transfert des messages, formant la deuxième classe de fonctions, est prise en charge par l'objet fonctionnel désigné par "agent de transfert de messages" (MTA). L'ensemble de tous les MTA forme le "système de transfert de messages" (MTS).

L'UA assure donc des services locaux, liés au traitement des messages, tels que l'écriture, l'archivage,... De plus, il réalise des fonctions associées au transfert des messages, soient l'écriture de l'en-tête sous forme normalisée et l'appel au service de transmission de messages.

Cette entité peut n'être qu'un simple éditeur de lignes associé à un système de fichiers. Il peut encore être très sophistiqué. Les différences au niveau des possibilités et de l'organisation interne des UA ne posent pas de problème car elles ne forment aucune entrave à la communication et ne font donc l'objet d'aucune normalisation.

L'UA sert de lien entre l'utilisateur et le STM, lequel assure un service de transmission depuis l'UA envoyeur jusqu'à l'UA destinataire, basé sur le principe d'enregistrement et de retransmission des messages entre les MTA (système "store and forward").

Pour envoyer un message, l'utilisateur le prépare avec l'aide de son UA. Celui-ci émet, vers un MTA, une demande d'envoi contenant les instructions de transfert et le contenu du message. En s'appuyant sur les informations d'adressage reçues, le MTA choisit le MTA adjacent avce lequel il interagira pour relayer le message. Ce processus est répété jusqu'à ce que le message arrive à le MTA qui supporte l'UA du destinataire. Le dernier MTA remet alors le message à l'UA en question. L'utilisateur destinataire n'a plus qu'à entrer en relation avec son UA pour prendre connaissance du message. Le mécanisme entier est illustré par la figure 3.2.

FIGURE 3.2 : MECANISME DE TRANSFERT DES MESSAGES.

Il faut remarquer que les MTA fonctionnent pour relayer le message de son envoi à sa réception, si bien qu'il n'est nullement nécessaire de créer une connexion entre l'envoyeur et le destinataire pendant le transfert du message. Ceci représente une innovation intéressante vis à vis des services qui existent couramment (télex, vidéotex, ...), lesquels travaillent en mode connecté.

3.2.2 DIVERSES IMPLANTATIONS POSSIBLES.

En présentant le modèle MHS, La norme X400 n'impose aucune implémentation physique particulière. Certaines des configurations physiques possibles sont représentées à la figure 3.3.

Un UA et un MTA peuvent être installés dans un même système. Dans ce cas, l'UA accède aux éléments de service de transfert de messages par interaction directe avec le MTA.

L'UA peut aussi être implémenté dans un système physiquement distinct et communiquer avec le MTA au moyen des protocoles normalisés spécifiques au transfert des messages.

D'autre part le MTA peut être mis en oeuvre dans un système sans UA.

Les différentes implémentations physiques peuvent être reliées au moyen de connexions spécialisées ou de réseaux.

FIGURE 3.3 : EXEMPLES D'IMPLEMENTATION.

3.2.3 : LE MHS FACE A L'ORGANISATION.

Une partie du MHS sous le contrôle d'une organisation est appelée un domaine de gestion (MD). Chaque domaine de gestion appartient à l'une des deux catégories suivantes :

public (domaine de gestion d'administration : ADMD)

privé (domaine de gestion privé : PRMD)

Dans un tel domaine, l'organisation doit prendre part au service de transfert des messages et doit dès lors posséder au moins un MTA. Elle peut aussi offrir des services d'UA.

L'utilisateur peut donc accéder à l'UA du domaine de gestion soit par un terminal, soit par un terminal intelligent, ou bien posséder son propre UA et entrer en relation avec le MTA du MD. Ceci est illustré par les figures 3.4.A et 3.4.B.

FIGURE 3.4.A : UA FOURNIT PAR L'ADMINISTRATION.

FIGURE 3.4.B : UA PRIVE / MTA DE L'ADMINISTRATION.

La figure 3.5 donne un exemple possible d'organisation. Le MHS est supporté par deux pays différents, A et B.

Comme le montre la figure 3.5, un PRMD peut accéder à un ou plusieurs ADMD à l'intérieur d'un pays. Cependant, dans une interaction particulière (transfert d'un message, par exemple), le PRMD est associé à un seul ADMD. La fonction de relais est uniquement accordée aux MTA du ADMD. C'est pourquoi un PRMD ne peut servir de lien entre deux DGAD.

FIGURE 3.5 : DOMAINES DE GESTION D'ADMINISTRATION ET DOMAINES DE GESTION PRIVES.

3.2.4 : VISION EN COUCHES DU MODELE MHS SELON LA REPRESENTATION OSI.

Le modèle MHS peut être incorporé dans le schéma classique du modèle de référence OSI (Open System Interconnection), ainsi que le décrit la figure 3.6.

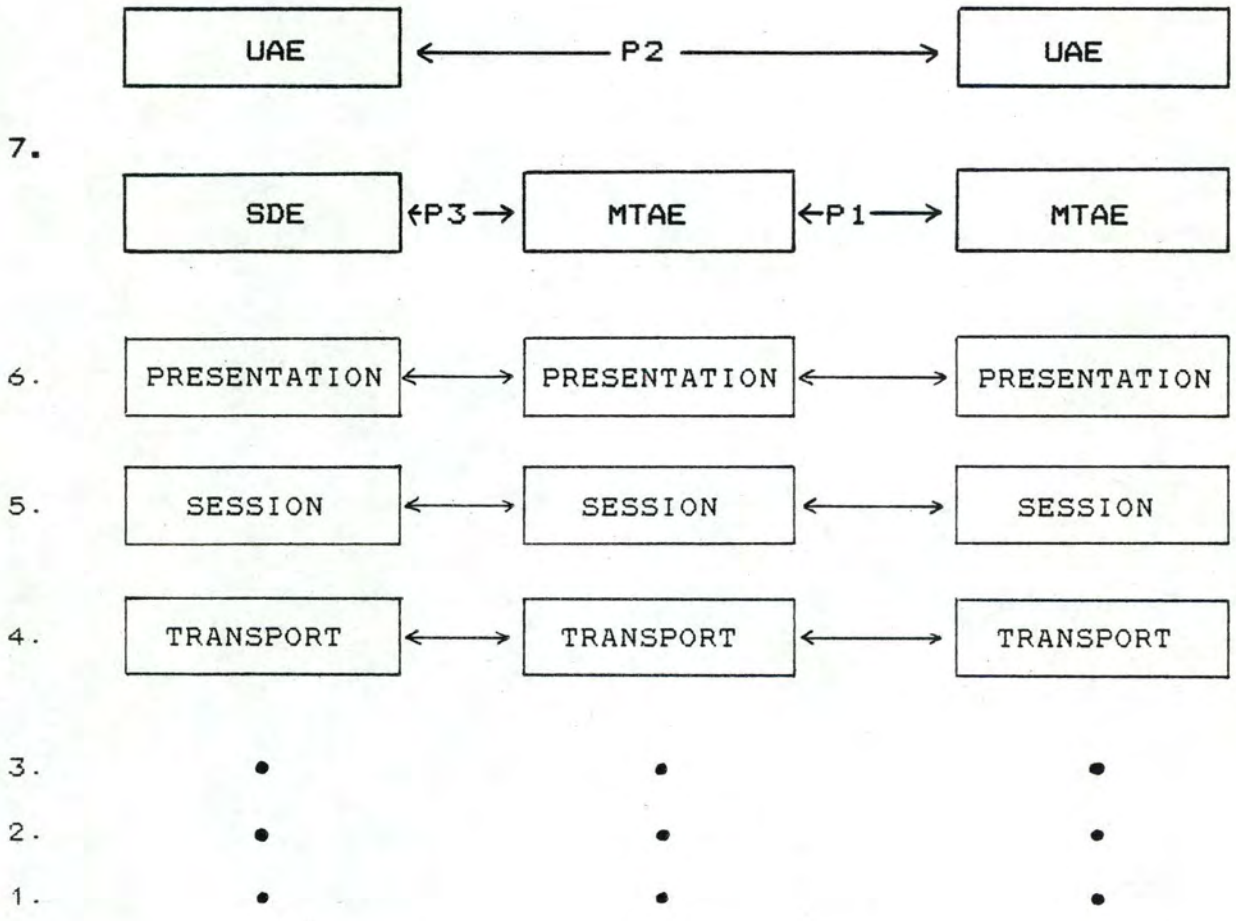


FIGURE 3.6 : REPRESENTATION OSI DU MODELE MHS.

Les trois premières couches (physique, logique et réseau) du modèle OSI décrivent les normes associées au réseau qui sert de moyen de communication entre les systèmes ouverts.

Les couches supérieures réalisent une communication de bout en bout entre les systèmes ouverts eux-mêmes. Cette communication est assurée par :

un transport fiable de données (couche 4)

un contrôle sur les sessions de communication (couche 5)

la négociation des syntaxes de présentation (couche 6)

Les entités actives du système MHS/x400 occupent la septième couche OSI. L'applicaton MHS est, à son tour, divisée en deux sous-couches (correspondant aux différentes classes de services) appelées :

UAL couche de l'agent de l'utilisateur

MTL couche chargée du transfert des messages

et dont les entités fonctionnelles sont

UAE entité d'agent utilisateur

MTAE entité d'agent du transfert des messages.

La figure 3.6 montre une autre entité : SDE, dont le rôle est d'assurer les protocoles de communication nécessaires à l'interaction d'un UA avec un MTA situé dans un autre système.

Les protocoles de communication sont également introduits :

P1 protocole de transfert de messages.

il gouverne le relais des messages entre MTA, tout au long du parcours depuis le MTA de l'envoyeur jusqu'à celui du destinataire

P3 protocole d'envoi et de remise

il permet l'accès à un MTA "éloigné" par une SDE associée à une entité UA

Pc protocoles entre UA

c'est une classe de protocoles dont les membres définissent la syntaxe et la sémantique pour un type particulier de messages à transférer entre UA.

Un de ces protocoles appelé P2 est spécialement utilisé pour la communication de personne à personne.

3.2.5 DESCRIPTION DES RECOMMANDATIONS.

L'ensemble de la norme X400 est matérialisée sous forme de huit recommandations. Celles-ci font plus qu'apporter une base pour les problèmes d'interconnexions de systèmes de messagerie électronique existant déjà : elles proposent un modèle à respecter lors de la création de nouveaux services traitant les messages.

Les recommandations X400 et X401 décrivent le modèle cité précédemment ainsi que les éléments de service accessibles via un système de traitement de messages (MHS) respectant la norme [REC X400 84] [REC X401 84].

Le coeur de toute la série est placé dans les recommandations X411 et X420. La première spécifie les services et protocoles en rapport avec le transfert des messages (soit P1), la seconde, elle, s'intéresse à ceux qui ont trait à la préparation et à la manipulation des messages par les utilisateurs dans le cadre d'une communication de personne à personne (soit P2) [REC X411 84] [REC X420 84].

Quant aux recommandations X408 et X409, elles définissent les techniques de notations utilisées pour spécifier les protocoles MHS et présentent les algorithmes de conversion entre types d'information codée différents [REC X408 84] [REC X409 84].

Les téléopérations et le serveur de transfert fiable (RTS) du système de traitement de messages sont détaillés au sein de la recommandation X410 [REC X410 84]. A cela s'ajoute la manière selon laquelle les applications de MHS s'appuient sur les services et protocoles normalisés pour l'interconnexion des systèmes ouverts (OSI). Finalement, la recommandation X430 explique comment les terminaux Télétex peuvent accéder au système MHS [REC X430 84] .

3.3 OBJECTIFS DU MEMOIRE.

Après la présentation générale du modèle MHS proposé par la norme X400, il est grand temps de préciser quels objectifs ce mémoire se fixe.

C'est une évidence, la norme X400 doit être l'objet principal des recherches effectuées. Or, les recommandations sont nombreuses, volumineuses, fastidieuses à lire, et souvent difficiles à comprendre. Pourtant, il est essentiel, si on veut concevoir un nouveau système de traitement de messages, d'approfondir cette norme et de la mettre en pratique, autant que possible, puisqu'elle semble représenter une direction d'avenir en ce qui concerne la messagerie électronique.

Pour cette raison, le travail de base à réaliser avant tout est une analyse détaillée des écrits X400. Le sujet se révélant très vaste - trop vaste , il est réduit, naturellement, à l'étude d'une des deux sous-couches décrites précédemment (cfr 3.3.4) : celle qui abrite l'agent responsable du transfert des messages, le MTA. Bien plus, une autre limitation intervient : le MTA envisagé ignore tout du protocole P3 et, dès lors, communique, via P1, avec ses UA, partageant tous avec lui un même site.

A cet objectif que l'on peut qualifier de bibliographique s'ajoute évidemment un aspect de recherche beaucoup plus personnelle. Celle-ci devrait, idéalement, se matérialiser par un essai de mise en oeuvre d'un système répondant aux fonctions dont x400 dote son entité MTA, et cela sur une des machines disponibles à l'Institut d'Informatique des Facultés Notre-Dame de la Paix de Namur : un PDP11 tournant sous la version 7 du système UNIX.

Cependant, la réalisation totale d'un MTA apparaît comme un travail de titan : outre l'analyse de la norme qui requiert, à elle seule, un laps de temps des plus importants, l'essai d'implémentation doit s'appuyer sur la conception d'une architecture susceptible de supporter un MTA. Dégager cette architecture exige, à son tour, une part de travail effrayante, même pour un groupe de trois personnes; d'autant plus que - et le démontrer est une des

tâches des chapitres suivants - l'architecture proposée ne s'impose pas à l'esprit d'une manière évidente, dès la première lecture des recommandations. Au contraire, puisque le sujet est d'une grande complexité, elle doit évoluer au fil des réflexions et recherches, qui amènent à une meilleure compréhension de X400, à la découverte de coins sombres dans la norme et à l'apport personnel de modifications à caractère exclusivement local. Ces dernières devraient - c'est un espoir qui exige critique ultérieure - engendrer des améliorations au profit des utilisateurs futurs du MTA, tout en restant, par l'essence même d'un essai de mise en oeuvre d'une norme, entièrement cohérentes avec les recommandations et à même de dialoguer avec des entités MTA et UA construites, par ailleurs, conformes à X400.

Au vu de toutes ces remarques concernant l'étendue du travail à réaliser, l'objectif du mémoire se voit progressivement remanié. Plus réaliste, il se dirige doucement vers une spécification complète du MTA, accompagnée de la construction de l'architecture du logociel. Le tout doit être suivi de l'implémentation, en ce compris le codage et les tests, d'un sous-ensemble choisi de fonctions représentatives, ce sous-ensemble pouvant être - si le temps imparti le permet- pas à pas élargi.

Pourtant, l'expérience a montré que ce but était encore très ambitieux, puisque peu à peu sont apparues de nombreuses difficultés, précédemment sous-estimées, ou tout simplement insoupçonnées.

Dès lors, malgré les efforts fournis, ce travail ne peut pas atteindre l'étape du codage et des tests. Il propose cependant un ensemble assez complet d'algorithmes, écrits en pseudo-langage, qui pourraient, par la suite, donner lieu à un essai sur machine, lequel permettrait d'analyser, sur base des performances obtenues, les

avantages engendrés par la touche toute personnelle apportée au MTA.

Chapitre IV.

LES SERVICES OFFERTS PAR UN MTA.

4.1. INTRODUCTION

Ce chapitre présente au lecteur un aperçu des services offerts par un MTA tels qu'ils sont définis dans la recommandation X400 du CCITT publiée en 1984.

La sélection de ces services permet à l'utilisateur de disposer d'un outil complet assurant la soumission de messages pour l'envoi à un ou plusieurs destinataires ainsi que la livraison de messages.

Les services que le MTA se doit de réaliser peuvent être classés en plusieurs catégories. La liste complète est donnée par les figures 4.1.a et 4.1.b.

La première catégorie regroupe les services de base, c'est-à-dire les services nécessaires pour permettre la soumission et la livraison des messages. Ces services de base concernent la gestion des accès entre UA et MTS et permettent à l'UA origine de spécifier certains paramètres en vue d'obtenir un envoi et une réception efficaces des messages vers le ou les UA destinataires (fig 4.1.a).

En plus des services de base, le MTA offre des services en option. Le choix de ceux-ci peut varier selon chaque message et la spécificité qu'on veut lui donner. Leur rôle est d'offrir à l'utilisateur la possibilité de bénéficier de services sélectionnés pour un message bien précis ou pour une période de temps fixée. Ces différents services sont divisés en quatre groupes:

services d'envoi et de remise des messages.

services de conversion d'informations.

services de sonde.

services complémentaires.

GROUPE DE SERVICE	ELEMENT DE SERVICE	SECTION
Service de Base	Gestion des accès entre UA et MTA	4.2.1.1.
	Spécification des types d'information encodée autorisés	4.2.1.2.
	Indication des types d'information encodée à l'origine	4.2.2.1.
	Indication du type de contenu	4.2.2.2.
	Identification des messages	4.2.3.1.
	Indication de conversion	4.2.3.2.
	Indication de l'heure de soumission	4.2.3.3.
	Indication de l'heure de livraison	4.2.3.4.
	Notification de non livraison	4.2.3.5.

FIGURE 4.1.A. : LES SERVICES DE BASE OFFERTS PAR UN MTA.

GROUPE DE SERVICE	ELEMENT DE SERVICE	SECTION
Services d'envoi et de remise de messages	Sélection du degré de priorité du message	4.3.1.
	Livraison multi-destination	4.3.2.
	Divulgation des autres destinataires	4.3.3.
	Permission de livraison au destinataire alternatif	4.3.4.
	Livraison différée	4.3.5.
	Notification de livraison	4.3.6.
	Interdiction de notification de non livraison	4.3.7.
	Retour de contenu	4.3.8.
	Annulation de la demande de livraison différée	4.3.9.
Services de conversion d'information	Conversion interdite	4.4.1.
	Conversion explicite	4.4.2.
	Conversion implicite	4.4.3.
Services de sonde	Sonde	4.5.1.
Services complémentaires	Désignation d'un destinataire alternatif	4.6.1.
	Stockage avant livraison	4.6.2.

FIGURE 4.1.B. : LES SERVICES A OPTION OFFERTS PAR UN MTA.

Le reste de ce chapitre présente en détail les services offerts aux utilisateurs qui, conjugués, fournissent une messagerie complète.

4.2. SERVICES DE BASE POUR LE TRANSFERT DE MESSAGES.

Cette section définit l'ensemble minimum des services qui sont offerts à l'utilisateur lorsque celui-ci désire utiliser le MTS pour envoyer un message vers un ou plusieurs UA destinataires.

Pour envoyer un message, l'UA origine le transmet à son MTA en signalant le nom ou l'adresse du ou des destinataires. Ce nom ou cette adresse doivent être conformes syntaxiquement au format qui est précisé par les normes.

4.2.1. PREMIER GROUPE.

le premier groupe parmi ces services de base comprend en fait les services qui ne peuvent être accomplis que par l'action des primitives mises à la disposition de l'UA par le MTA.

4.2.1.1. Gestion des accès entre UA et MTA.

Le mécanisme d'accès entre UA et MTA, nécessaire pour le transfert des messages, ainsi que la gestion de ces accès sont rendus possibles grâce au service "gestion des accès entre UA et MTA".

Ce service permet:

l'identification d'un UA par son MTA et vice-versa grâce à un système de mots de passe. Il est à noter que ce mécanisme de mots de passe est indépendant de celui qui permet à l'UA d'authentifier ses utilisateurs.

l'établissement des accès UA-MTA, ainsi que le traitement des informations liées à des accès (changement de mot de passe, adresse réseau fournie au MTA par l'UA, ...).

4.2.1.2. Spécification des types d'information encodée autorisés.

Grâce au service "spécification des types d'information encodée autorisés", l'UA a la possibilité de spécifier les types d'information encodée qu'il est prêt à recevoir du MTA. Ce service est nécessaire puisque, en théorie, le MHS/400 permet la soumission et la livraison de messages de tout type, alors que l'UA ne permet l'envoi et la réception que de certains types de messages.

4.2.2. DEUXIEME GROUPE.

Formant un deuxième groupe parmi les services de base, certains services sont activés par l'UA lorsque celui-ci donne des valeurs, autres que celles définies par défaut, aux paramètres des primitives qui sont mises à sa disposition par le MTA pour effectuer la soumission de messages. Les services de ce deuxième groupe sont décrits ci-dessous.

4.2.2.1. Indication des types d'information encodée à l'origine.

Lors de la soumission, l'UA peut fournir au MTA les types d'information encodée véhiculés par le message au moment de son envoi (par exemple: un texte de type ASCII). Ces types d'information accompagnent le message lors de sa livraison aux UA destinataires.

4.2.2.2. Indication de type de contenu.

L'UA d'origine peut indiquer le type de contenu du message qu'il veut envoyer c'est-à-dire le nom du protocole qu'il respecte pour dialoguer avec l'UA destinataire. Dans le cas de la messagerie interpersonnelle, le type de contenu du message est P2.

Le type de contenu ne doit pas être confondu avec le type d'information encodée qui est le type de l'information transmise par le message.

4.2.3. TROISIEME GROUPE.

Le dernier groupe comprend les services de base réalisés par le MTS. Par ces services, le MTS a la possibilité de donner des informations aux UA lors de la soumission ou la livraison de messages.

4.2.3.1. Identification de messages.

"Identification de messages" est un service qui permet à l'UA de disposer d'un identificateur unique pour chaque message soumis au MTS. Les UA et le MTS utilisent cet identificateur pour faire référence à un message envoyé précédemment lorsqu'une notification de livraison ou de non livraison est demandée.

4.2.3.2. Indication de conversion.

Lorqu'une conversion a été effectuée sur le message pour qu'il puisse être livré avec succès, le service "indication de conversion" permet au MTS d'indiquer aux UA destinataires qu'une conversion a été accomplie et de leur fournir les types d'information obtenus après cette conversion.

4.2.3.3. Indication de l'heure de soumission.

Grâce au service "indication de l'heure de soumission", le MTS peut renseigner les UA origine et destinataire sur la date et l'instant auquels la demande d'envoi du message a été prise en charge.

4.2.3.4. Indication de l'heure de livraison.

"Indication de l'heure de livraison" est un service qui permet au MTS d'informer l'UA destinataire de la date et du moment auquels le message a été livré à l'UA.

4.2.3.5. Notification de non livraison.

Le service "notification de non livraison" permet au MTS d'avertir l'UA d'origine dans le cas ou un des messages qu'il a envoyé n'a pas pu atteindre sa destination (ou l'une d'entre elles dans le cas d'un message multi-destination). La raison pour laquelle la livraison n'a pas pu être accomplie est un des paramètres dans la notification renvoyée par le MTS à l'UA origine.

4.3. SERVICES D'ENVOI ET DE REMISE DE MESSAGES.

Dans ce groupe de services concernant l'envoi et la remise de messages à un UA, le service "annulation de la demande de livraison différée" est activé par l'UA en utilisant une primitive offerte par le MTA. Les autres services sont mis à la disposition de l'utilisateur grâce aux différents paramètres dont il doit spécifier la valeur lors de la soumission du message pour l'envoi vers un ou plusieurs UA destinataires. Les services suivants peuvent être distingués.

4.3.1. SELECTION DU DEGRE DE PRIORITE DU MESSAGE.

L'UA origine peut choisir de transmettre un message à travers le MTS en lui attribuant une priorité en rapport avec l'importance de l'envoi. Celle-ci peut être non urgente, normale ou urgente.

4.3.2. LIVRAISON MULTI-DESTINATION.

On entend par "livraison multi-destination" la possibilité donnée à l'UA de demander que le message soit transmis et livré à plus d'un UA destinataire. Le nombre de destinataires pour un message est illimité.

4.3.3. DIVULGATION DES AUTRES DESTINATAIRES.

Une autre possibilité offerte à l'UA lors de la soumission du message est qu'il peut choisir de révéler les noms des destinataires à tous les récepteurs du message.

4.3.4. PERMISSION DE LIVRAISON AU DESTINATAIRE ALTERNATIF.

Lorsque l'UA origine fournit au MTS un nom de destinataire qui, correct syntaxiquement, permet de conduire le message jusqu'à l'intérieur d'un domaine de gestion, mais une fois dans ce domaine, ne suffit plus à le diriger avec précision vers un UA destinataire, le message peut être livré à une "boîte aux lettres" spécialement prévue pour la réception de tels messages par le domaine d'arrivée (à condition que le domaine de gestion ait prévu une telle "boîte aux lettres" grâce au service "désignation d'un destinataire alternatif").

Le MTA destinataire crée et envoie une notification de non livraison (si elle n'est pas interdite par l'origine) lorsque le message doit être livré à la "boîte aux lettres". Cette notification indique à l'émetteur que le message n'a pas pu atteindre la destination prévue et qu'il a été dirigé vers la "boîte aux lettres".

4.3.5. LIVRAISON DIFFEREE.

L'existence du service "livraison différée" permet à l'UA origine de demander au MTS que le message qu'il désire faire parvenir à ses destinataires ne leur soit pas livré avant un moment bien précis. Ce moment est défini à l'aide d'une date et d'une heure. La livraison du message

est effectuée aussitôt que possible après le moment fixé par l'UA origine. La date et l'heure précisées pour la livraison différée sont soumises à certaines restrictions déterminées par le domaine de gestion de l'origine.

4.3.6. NOTIFICATION DE LIVRAISON.

L'UA origine, pour s'assurer de la bonne livraison d'un message, peut demander qu'une notification explicite lui soit retournée par le MTA destinataire lorsque celui-ci a livré le message avec succès. Cette notification, outre le jour et l'heure de livraison, donne à l'UA d'origine l'identificateur du message concerné.

Dans le cas d'un message envoyé à plusieurs destinataires, une notification de livraison peut confirmer l'aboutissement de la transmission du message vers un ou plusieurs UA destinataires.

La notification de livraison n'implique nullement que l'UA destinataire ou son utilisateur ait pris connaissance du message.

4.3.7. INTERDICTION DE NOTIFICATION DE NON LIVRAISON.

Puisque "notification de non livraison" est un service de base, l'UA origine est automatiquement averti lorsqu'un message n'a pu être livré correctement à un de ses destinataires. Cependant, l'UA d'origine a la possibilité d'inhiber ce service en spécifiant au MTS lors de la soumission du message, qu'il ne désire aucune notification de non livraison.

4.3.8. RETOUR DE CONTENU.

Dans le cas où il n'interdit pas la réception de notification de non livraison, l'UA origine peut spécifier si les notifications éventuelles doivent renfermer le contenu du message envoyé. Ce service n'a de sens que si le contenu du message à livrer n'a subi aucune conversion.

Grâce à ce service, l'UA origine peut récupérer le contenu des messages qui n'ont pas été livrés à leurs destinataires et ainsi les renvoyer sans avoir dû en stocker le contenu.

4.3.9. ANNULATION DE LA DEMANDE DE LIVRAISON DIFFEREE.

Tout comme une livraison différée peut être demandée par l'UA origine, elle peut aussi être annulée. Pour ce faire, l'UA doit activer le service "annulation de la demande de livraison différée" mis à sa disposition dans ce but. Cette annulation peut échouer dans le cas où la livraison a déjà été accomplie ou lorsque le message a déjà été transmis au MTA suivant sur la route vers le ou les destinataires. Le MTA informe toujours l'UA des résultats de sa demande d'annulation. Dans le cas d'un échec, il en fournit alors la raison.

4.4. SERVICES DE CONVERSION D'INFORMATIONS.

Les conversions envisagées dans cette catégorie de services offerts s'appliquent aux informations que véhiculent les messages. Lorsqu'une conversion quelconque a été accomplie, l'UA destinataire est informé du type de l'information obtenu après cette conversion, ainsi que du type qu'avait le message à l'origine. A titre d'exemple, on peut citer les conversions des types d'information ASCII en téléfax, télex en télétex,....

Tous les services concernant les conversions sont activés grâce aux paramètres dont l'origine doit spécifier la valeur lors de la soumission du message.

le problème de toute conversion est qu'elle oblige le MTS à modifier le contenu du message et supprime ainsi la confidentialité de l'information véhiculée. Conceptuellement, les données contenues dans l'enveloppe du message devraient suffire au MTS pour qu'il puisse remplir son rôle. Ces services de conversion représentent cependant la seule possibilité laissée au MTS d'atteindre le contenu du message.

Dans ce groupe, on peut distinguer les éléments de conversion suivants.

4.4.1. CONVERSION INTERDITE.

L'UA origine peut interdire toute conversion du message qu'il transmet en l'indiquant au MTS lors de la soumission du message.

Lorsqu'une conversion est obligatoire pour pouvoir livrer le message à un UA destinataire, et que toute conversion est interdite par l'UA origine, le MTA destinataire crée et envoie une notification de non livraison à l'UA origine, pour autant que celle-ci n'ait pas été interdite.

4.4.2. CONVERSION EXPLICITE.

On définit par "conversion explicite" la possibilité qu'a le MTS d'accomplir, avant la livraison du message, une conversion d'informations particulière, demandée par l'UA origine.

Le choix du moment de la conversion est laissée à l'initiative du MTS. Il semble logique de penser que le MTS effectuera cette conversion au plus tôt si elle a pour conséquence de raccourcir la longueur du message à relayer.

4.4.3. CONVERSION IMPLICITE.

L'UA origine peut autoriser le MTS a effectuer toute conversion nécessaire sur le message envoyé. Lorsque l'UA destinataire permet la réception de plusieurs types d'information et qu'une conversion est nécessaire, une des conversions possibles est réalisée.

L'UA origine ne doit pas faire apparaître dans les paramètres instanciés lors de la soumission, qu'il désire le service "conversion implicite". Ce service est celui qui est choisi par défaut par le MTS lorsque l'UA origine ne stipule rien en ce qui concerne les conversions de messages.

4.5. SERVICES DE SONDE.

Ce groupe de services permet aux UA de se renseigner sur le contrôle et les opérations du MTS. Ces services sont activés par L'UA grâce à des primitives mises à sa disposition par le MTA. Cette catégorie ne contient qu'un seul élément de service.

4.5.1. LA SONDE.

Avant d'envoyer un message vers un ou plusieurs destinataires, l'UA origine peut utiliser le service "sonde". Par cela, il peut tester si un message futur, respectant certaines caractéritiques précises, pourra être livré sans encombre. L'UA donne ces caractéristiques (longueur estimée, type de contenu, type d'information) et la sonde permet le transfert d'un pseudo-message, de contenu vide, afin de tester la route et les divers paramètres. Le résultat de la sonde sera fonction des restrictions imposées par l'UA destinataire.

La sonde jouit des mêmes avantages que les messages de priorité urgente.

4.6. SERVICES COMPLEMENTAIRES.

Ces services se placent à un niveau différent des autres dans les relations entre l'UA et le MTA. Ces services sont à la disposition de l'UA lorsque celui-ci s'abonne à la messagerie électronique. Ces services complémentaires comprennent les points suivants.

4.6.1. DESIGNATION D'UN DESTINATAIRE ALTERNATIF.

"Désignation d'un destinataire alternatif" est un autre service offert par l'UA. Il permet à cet UA de recevoir certains messages ne pouvant être livrés à aucun UA du domaine à cause d'un problème d'adressage.

Cet UA aura une adresse formée de valeurs imposées comme le nom du pays, le nom du domaine de gestion, par exemple, mais laissera entièrement libre les autres paramètres de l'adresse. Un message pouvant être relayé jusqu'à l'intérieur de ce domaine, mais ne pouvant être livré à aucun UA à cause d'un problème dans l'adresse du destinataire sera transmis à cet UA "boîte aux lettres".

Ce service permet, entr'autres, une gestion manuelle de tous les messages aboutissant dans un domaine de gestion.

Pour que la livraison à la boîte aux lettres, au cas où elle devrait se produire, puisse s'accomplir en suivant la procédure habituelle, il est nécessaire que l'UA ait eu recours au service "permission de livraison au destinataire alternatif".

Dans le cas où le message ne peut être livré à son destinataire, une notification de non livraison est retournée à l'UA origine, si celui-ci n'a pas inhibé ce service accompli par défaut.

Lorsque le domaine ne dispose pas d'un UA "boîte aux lettres" ou que le service "permission de livraison au destinataire alternatif" n'a pas été sélectionné, le message est détruit.

4.6.2. STOCKAGE AVANT LIVRAISON.

"Stockage avant livraison" est le dernier service mis à la disposition des UA. Il permet aux UA destinataires de demander au MTA de conserver chez lui les messages ainsique les notifications qui lui sont adressées, ne pouvant être délivrés à l'UA destinataire à cause des restrictions que celui-ci a imposées sur les types d'information encodée, la longueur du contenu et la priorité.

Le MTA attend que l'UA soit à nouveau prêt à recevoir les messages et les notifications pour les lui livrer. A ce moment seulement, il envoie vers l'UA origine une notification de livraison si elle a été demandée. Le MTA transmet à l'UA origine une notification de non livraison dès que le temps d'attente de livraison maximum est écoulé.

Lorsqu'il le désire, l'UA destinataire peut se voir informé du nombre de messages retenus par le MTA, responsable de la gestion des messages en attente.

Chapitre V.

METHODOLOGIE

5.1. INTRODUCTION

Comme il a déjà été précisé précédemment, le but de ce travail est la mise en oeuvre d'un logiciel, en tout point conforme à la norme X400, remplissant les fonctionnalités imposées par cette même norme à l'objet fonctionnel appelé MTA.

La compréhension de X400, documentation volumineuse et souvent peu claire, représente à elle seule une difficulté non négligeable. La réalisation pratique, quant à elle, induit une quantité impressionnante de problèmes à régler, soit au niveau des choix d'implémentation, pour les aspects que la norme prescrit, soit au niveau des choix internes concernant les détails que la norme passe sous silence.

En conséquence, il apparaît clairement que le temps imparti à ce travail n'est pas suffisant pour mener à bien l'implémentation de logiciel dans son ensemble, même pour une équipe de trois personnes.

Dès lors, certaines restrictions et simplifications se révèlent indispensables. Et bien sûr, dans ce cadre, il est un objectif primordial à ne pas perdre de vue : le caractère portable et extensible du produit fini à présenter. A cela s'ajoutent les qualités que tout logiciel devrait posséder : fiabilité, robustesse, facilité de maintenance, documentation adéquate, performances,...

Puisque c'est à la construction d'un logiciel qu'il faut s'atteler, il convient de se référer à une méthodologie de développement de logiciels qui servira autant que possible de guide et de grille d'analyse pour

les problèmes divers à résoudre et qui dotera le résultat obtenu des propriétés qu'on souhaite le voir posséder.

5.2. PRINCIPES GENERAUX POUR LA CONCEPTION DE LOGICIELS

Lorsqu'il s'agit de prendre part à la conception d'un logiciel, l'Institut d'Informatique des Facultés Notre-Dame de la Paix de Namur prône une méthodologie spécifique et la présente de manière théorique à travers les cours dispensés [AVL 87].

Contrairement à la méthode dite "par essais et erreurs" qui, par le biais de tests de conformité par rapport aux spécifications globales du problème - globales, mais peut-être imprécises, développe un produit par ajustements successifs, la méthodologie défendue propose, dès après l'analyse des besoins et l'analyse fonctionnelle, la conception d'une architecture logicielle générale représentant le système à créer, vu dans son ensemble. Cette architecture est la charpente sur laquelle tout le logiciel s'appuie. Il s'agit d'en identifier les composants (ou modules), de les organiser en une structure solide, stable et extensible, et enfin de les spécifier de manière précise, i.e. exprimer clairement ce que l'utilisateur doit savoir pour se servir de ces composants, sans ajouter de renseignements non pertinents. Après cela, il faut encore concevoir en détail chacun des composants, et le matérialiser soit par un algorithme de traitement, soit par une représentation de structure de données adéquate.

Jusqu'à ce moment, tout se passe sur le plan logique, c'est à dire indépendamment de la machine et du langage à utiliser. Ceci aide sans aucun doute à doter le

logiciel de robustesse. De plus, ceci lui confère une part de portabilité, d'autant plus importante que la charpente logique se dégage de toute contrainte matérielle ou logicielle.

Après l'architecture logique vient l'architecture physique, qui est, dans le développement, une nouvelle étape par rapport à la précédente, prenant en compte les caractéristiques propres de la machine physique sur laquelle le logiciel doit être implémenté. C'est lors du passage d'une architecture à l'autre que la notion de "traçabilité", on l'espère, se dégage. Il ne reste alors qu'à coder les algorithmes dans un langage compréhensible par la machine.

La méthodologie prévoit également que l'architecture logique - et par voie de conséquence, l'architecture physique qui en découle - réponde à une structuration hiérarchisée du système. Et dans ce sens, il peut être opportun de rappeler que

une structure est hiérarchisée

ssi

∃ une relation R entre composants qui permet de définir des niveaux niv_i (i entier) tq

soient A,B,C des composants,

niv_0 = {A | ∃ B tq R(A,B)}

niv_1 = {A | ∃ B ∈ niv_{i-1} tq R(A,B)

et tq R(A,C) => C ∈ nivl où l <= i-1 } (i>0)

Diverses relations R peuvent être envisagées, mais il en est une qui présente un caractère particulièrement intéressant : la relation UTILISE. La définition suivante peut en être donnée :

A utilise B <=> le fonctionnement correct de A
dépend de la disponibilité
d'une version correcte de B.

En pratique, on constate que cette relation UTILISE permet une simplification importante des composants - que ce soit au niveau conception ou au niveau réalisation. Cette simplification vient du fait que la structuration du logiciel par la relation UTILISE organise le traitement global en divers modules, dont chacun règle soit un sous-problème particulier, soit un problème commun à plusieurs fonctions de niveau supérieur. Il en résulte une élimination des redondances fonctionnelles ainsi qu'une homogénéisation de conception et de réalisation qui induit une manipulation plus standardisée des traitements communs, et dès lors une simplification des composants qui les utilisent. De plus, cette relation facilite une factorisation intelligente du travail de développement au sein d'un groupe (spécifications, codage et validation). Elle assure également au produit résultant une maintenance simple puisqu'une modification éventuelle est rendue ponctuelle, enfermée dans un niveau, et ne se répercute en rien sur les autres niveaux.

Il est cependant important de signaler que cette relation UTILISE n'est pas la seule qui existe et qui est employée. Ainsi, la relation APPELLE apparaît fréquemment lors des décompositions en modules. Par ailleurs, la structuration hiérarchique du système se base très souvent sur son diagramme de flux. La méthodologie présentée ne postule pas que ces dernières relations sont sans valeur. Seulement, elle estime qu'elles sont plus faibles que la relation UTILISE, notamment parce qu'elles conduisent à des structurations constituées de modules très inter-reliés -

trop pour assurer une factorisation efficace du travail et trop pour que les modules apparaissent, à leurs utilisateurs, comme des boîtes noires remplissant, d'une manière personnelle et gardée cachée, les tâches pour lesquelles ils sont conçus.

Finalement, comme elles rappellent les qualités de tout bon logiciel, les caractéristiques issues de la relation UTILISE ne sont pas à dédaigner.

5.3 APPLICATION DE LA METHODOLOGIE AU CAS DU MTA

La méthodologie adoptée lors du développement du MTA est celle présentée au point précédent.

Dès lors, l'architecture logique, partiellement décrite par la normalisation X400, donne naissance à une architecture physique s'appuyant sur le fait que la machine physique visée par ce projet possède un système d'exploitation UNIX.

Mais avant tout, il est nécessaire que toute personne amenée, dans le futur, à manipuler le logiciel MTA, quel que soit son rôle (implémenteur, utilisateur,...), soit, dès à présent, guidée par une documentation complète et cohérente, sans pour cela devoir étudier en détails les parties de la norme X400 concernant typiquement le MTA. C'est pourquoi ce travail débute par des spécifications précises, expliquant les services réalisés par un MTA, ainsi que la manière de dialoguer avec lui. A ce niveau, le MTA reste une "boîte noire" pour l'utilisateur, lequel dispose, pour lui parler, de moyens soigneusement décrits.

La méthodologie, on l'a vu, demande la spécification détaillée de chaque module constituant l'architecture. Plusieurs modèles de spécifications existent dans la littérature. Mais bien vite, il apparaît évident qu'un travail de spécifications dans le langage naturel non structuré est épuisant, puisqu'il nécessite une lutte continuelle contre les imprécisions innées qui en découlent, ainsi que contre d'autres défauts bien connus tels que les bruit, silence, ambiguïté, surspécification,...

Les spécifications dites PRE-POST offrent un moyen efficace pour remédier à ce problème. Par cette méthode, chaque module - ou chaque fonction d'interface du module, si le module ne se limite pas à la réalisation d'une seule fonction, mais en regroupe plusieurs - se voit associer la liste de ses arguments (entrées), la liste de ses résultats (sorties) ainsi qu'une paire d'assertions :

* la précondition :

  la plus faible condition qui cactérise les propriétés des arguments, lesquelles doivent être satisfaites avant toute exécution du module, pour qu'il s'exécute correctement.

* la postcondition :

  la plus forte condition qui explicite les propriétés des résultats, lesquelles doivent être satisfaites après exécution, si le module s'est exécuté correctement.

Les condition pré et post peuvent, elles aussi, être exprimées en langage naturel. Mais leur structure rigide évite, à elle seule, les défauts habituels des spécifications par règles. Par ailleurs, il semble que les spécifications pré-post sous forme algébrique allient tous les aspects positifs, tout en restant, c'est vrai, plus complexes à écrire.

En théorie, cette méthode de spécification est extrêmement complète. Hélas, en pratique, elle peut se révéler insuffisante. C'est le cas lorsqu'il est impossible de prévoir l'enchaînement des différents modules (par exemple parce que leur déclenchement est causé par des événements en provenance de diverses sources indépendantes) et que cet enchaînement même fait partie de l'objet des spécifications. Le MTA se place bien dans ce cadre, puisqu'il sert de lien entre l'utilisateur du service de transfert de messages - l'UA - et la couche session du modèle OSI. Dès lors, les fonctions remplies par le MTA peuvent, à tout moment et de manière asynchrone, être déclenchées par chacun des UA qui lui est connecté, par la couche session sur laquelle il repose, ainsi que par un autre MTA avec lequel il dialogue par le biais du protocole adéquat, et toujours via la couche session. D'autre part, il arrive fréquemment qu'une relation de cause à effet unisse plusieurs actions réalisées au sein des diverses couches et sous-couches. En résumé, une "utilisation" du MTA dépend de l'historique des autres "utilisations".

Puisque - et cela malgré leur grande précision, leur facilité d'interprétation et leur non-ambiguïté - les spécifications PRE-POST ne suffisent plus lorsqu'intervient l'historique, au vu de leur caractère assez statique, elles se voient complétées par d'autres renseignements décrivant les enchaînements possibles des divers modules. C'est le formalisme des automates (ou machines) à états finis qui nous en fournit les moyens nécessaires.

Grâce à cette méthode, un système peut être décrit à l'aide de plusieurs concepts de base :

- les états possibles du système,
- les événements possibles,
- les actions à réaliser.

Dans ce "langage" de représentation et de spécification d'un système en dialogue fréquent avec son environnement, tout événement se voit définir comme un stimulus extérieur qui agit sur le système et engendre de sa part une réaction précise, notemment sous forme d'une suite d'actions. La notion d'état, quant à elle, présente parfois une certaine ambiguïté. En effet, il arrive que l'état d'un système soit considéré comme directement lié à l'instanciation de toutes les variables du système à un moment donné. Tandis que dans le formalisme adopté à des fins de spécification, l'état du système, plutôt qu'un état physique, matériel et déterminé par relevé des valeurs des variables, est une abstraction de l'historique du système. Cette abstraction reste totalement cohérente avec le rôle des spécifications : elles doivent, on le sait, exprimer tout ce qui est nécessaire pour utiliser le module, et rien de plus. Or une abstraction de l'historique se trouve être un renseignement nécessaire et suffisant.

A titre d'exemple d'historique exprimée en termes d'états, supposons que le système ait déjà réagi à une suite d'événements, soient les événements notés $évé_0, \ldots, évé_{i-1}$ (cfr figure 5.1), l'historique du système peut être résumée en un seul qualificatif, soit l'état A. L'arrivée de l'événement i implique alors un changement d'état - ou transition, et éventuellement l'une ou l'autre action. Pour être complet, il convient de définir un état 0, état particulier qui décrit l'état initial du système.

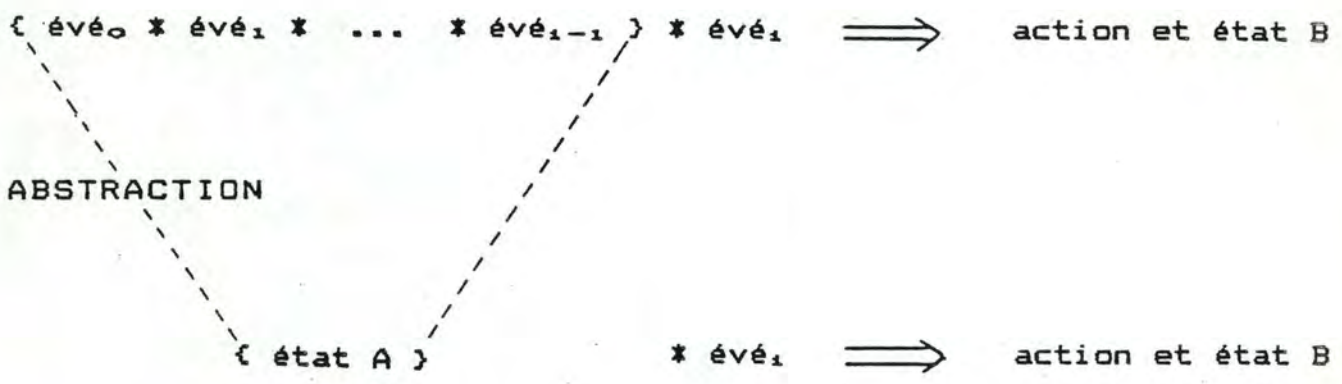


FIGURE 5.1 : NOTIONS D'ETAT, EVENEMENT ET ACTION.

Les états possibles sont supposés en nombre fini. Et dans le cas du MTA, les événements pris en compte sont des éléments de dialogue entre le MTA et les couches inférieure et supérieure, à savoir des primitives d'interface entre les couches.

Le principe de base du formalisme des automates à états finis est le suivant : connaissant l'état du système et l'événement qui survient, il doit être possible de déterminer de manière précise ce que le système doit faire, sans plus avoir à envisager l'histoire du système, puisqu'elle est présente en l'état.

Les notions fondamentales (événements, états, actions) peuvent donner naissance à deux représentations utilisables : les diagrammes d'états et les tables d'états.

Les tables d'états apparaissent comme des tables à deux entrées. L'événement caractérisé par l'en-tête de ligne, ayant lieu lorsque le système est dans l'état précisé par l'en-tête de la colonne, déclenche la séquence d'actions placée à l'intersection de la ligne et de la colonne, avant de passer, éventuellement, en un nouvel état, désigné à la suite des actions à effectuer (cfr figure 5.2).

	état 1	**état 2**
événement 1	action 1 action 2	action 2 état 1
événement 2	-	action 3 action 4

FIGURE 5.2 : CONFIGURATION D'UNE TABLE D'ETATS.

Les diagrammes d'états permettent de visualiser les enchaînements sous forme graphique, tout en s'exprimant cependant de manière moins complète en ce sens qu'ils ne permettent pas d'expliciter d'autre action que le changement d'état. Comme le montre la figure 5.3, l'événement1 survenant lorsque le système est en l'état1 a pour conséquence de le faire passer en l'état2, tandis que l'événement2 ne modifie pas son état.

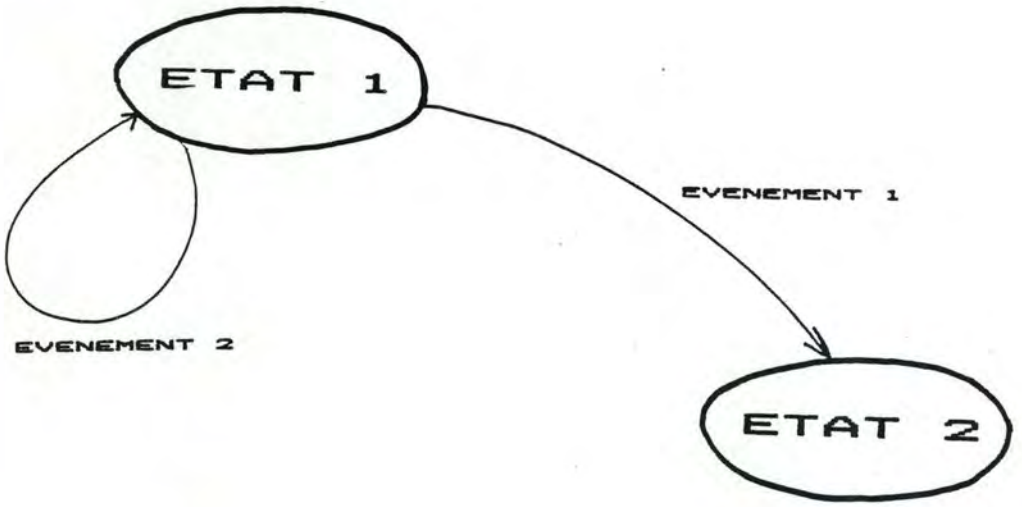


FIGURE 5.3 : CONFIGURATION D'UN DIAGRAMME D'ETATS.

Par ailleurs, les diagrammes d'états s'avèrent particulièrement bien adaptés à la modélisation de systèmes présentant la notion de parallélisme [GVB 83]. Pour ce faire, plusieurs machines abstraites cohabitent tout en conservant leur indépendance. Chacune d'entre elles reconnaît un certain nombre d'événements. Les états par lesquels elle peut passer forment un sous-ensemble de l'ensemble des états possibles du système global. Quoique menant leur vie propre, ces machines peuvent être amenées à se synchroniser sur des événements communs qui établissent entre elles un certain degré de communication.

Il est important de souligner le fait que les tables et les diagrammes d'états conjugués ne peuvent laisser subsister aucune ambiguïté quant aux enchaînements.

Dès lors, tout événement trouvant le système dans un autre état que ceux précisés lors des spécifications, conduit automatiquement à une erreur.

5.4 DEFINITION DE SOUS-SYSTEMES UTILES

La réalisation totale d'un MTA représente une entreprise de grande taille. De ce fait, la notion de sous-système utile devient prépondérante.

En effet, un sous-système utile est un sous-ensemble de composants de l'architecture logique, dont l'exécution est autonome, et qui remplit déjà des fonctionnalités non négligeables. En fait, c'est un petit noyau dont on peut faire une démonstration et qui assure que le projet global est en bonne voie. C'est pourquoi cette pratique permet d'aborder les difficultés de manière progressive, d'obtenir plus rapidement quelques résultats - et donc d'éviter le découragement - et peut servir de base au moment du plan de test.

Au niveau de l'implémentation du MTA, la définition de trois sous-systèmes utiles emboîtés peut être adoptée. Ils constituent par là des étapes importantes, chacune permettant d'étoffer et d'améliorer la précédente, dans le but d'atteindre finalement un logiciel complet, répondant à un cadre très général.

La première étape considère un MTA avec deux UA qui lui sont connectés. Tous trois partagent le même site, conformément à la figure 5.4.

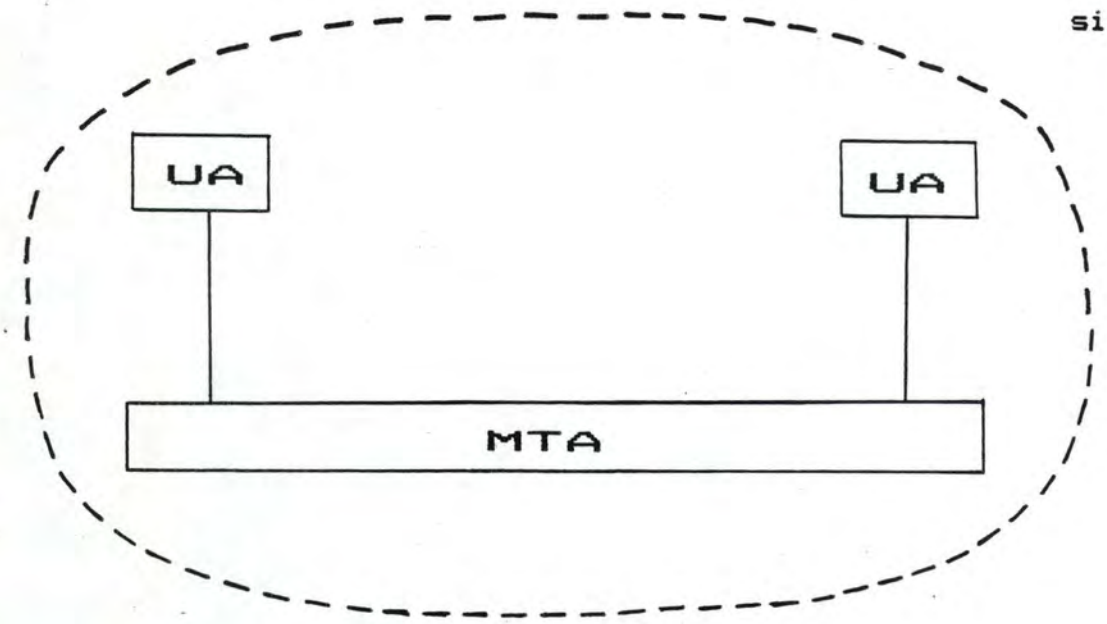


FIGURE 5.4 : PREMIER SOUS-SYSTEME UTILE.

Cette première phase permet à un UA d'envoyer des messages à un autre UA relié au même MTA. Cette hypothèse simplifie considérablement le travail à réaliser puisque l'existence d'un site unique supprime toute notion de transport. Il est possible de se contenter du seul interface MTA-UA, enveloppant l'intelligence du MTA capable de se rendre compte que l'UA destinataire est un UA local.

Vient alors la deuxième étape qui résoud le problème d'un UA désirant entrer en communication avec un UA éloigné, sachant que la relation unissant les deux MTA qui les servent est directe et qu'aucune autre n'existe (cfr figure 5.5).

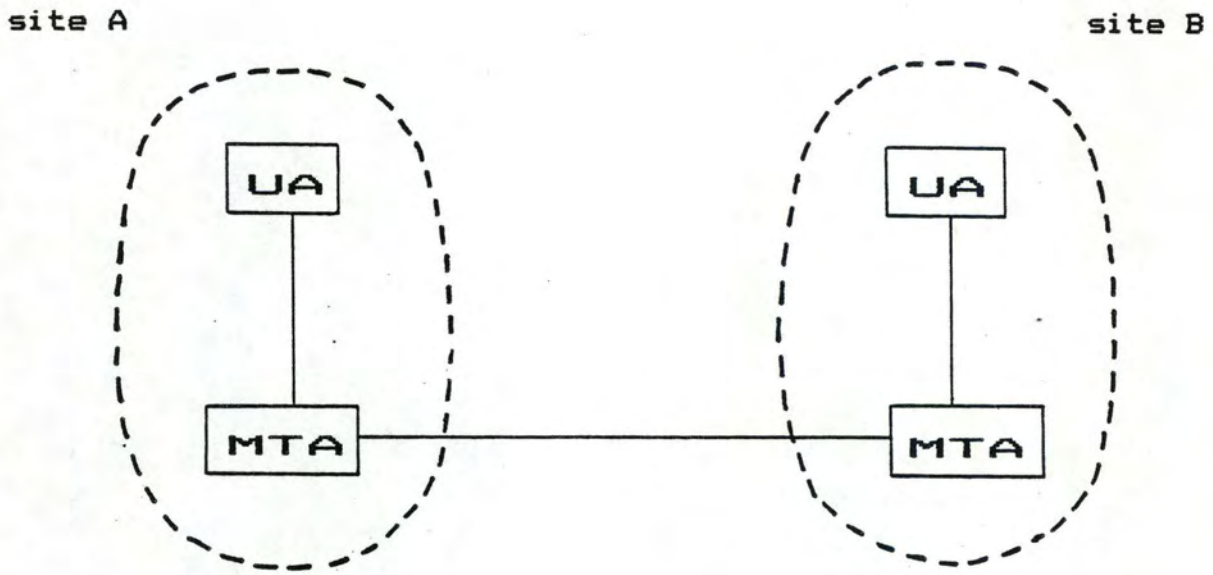


FIGURE 5.5 : DEUXIEME SOUS-SYSTEME UTILE.

La distance séparant les deux sites implique la résolution des difficultés liées à la télécommunication, par l'intermédiaire des diverses couches du modèle OSI. Cependant, le caractère trivial du routage en apporte une simplification sensible.

Enfin, le cas général, décrit à la figure 5.6, forme la troisième étape et règle le dernier problème, celui de l'algorithme complet de routage.

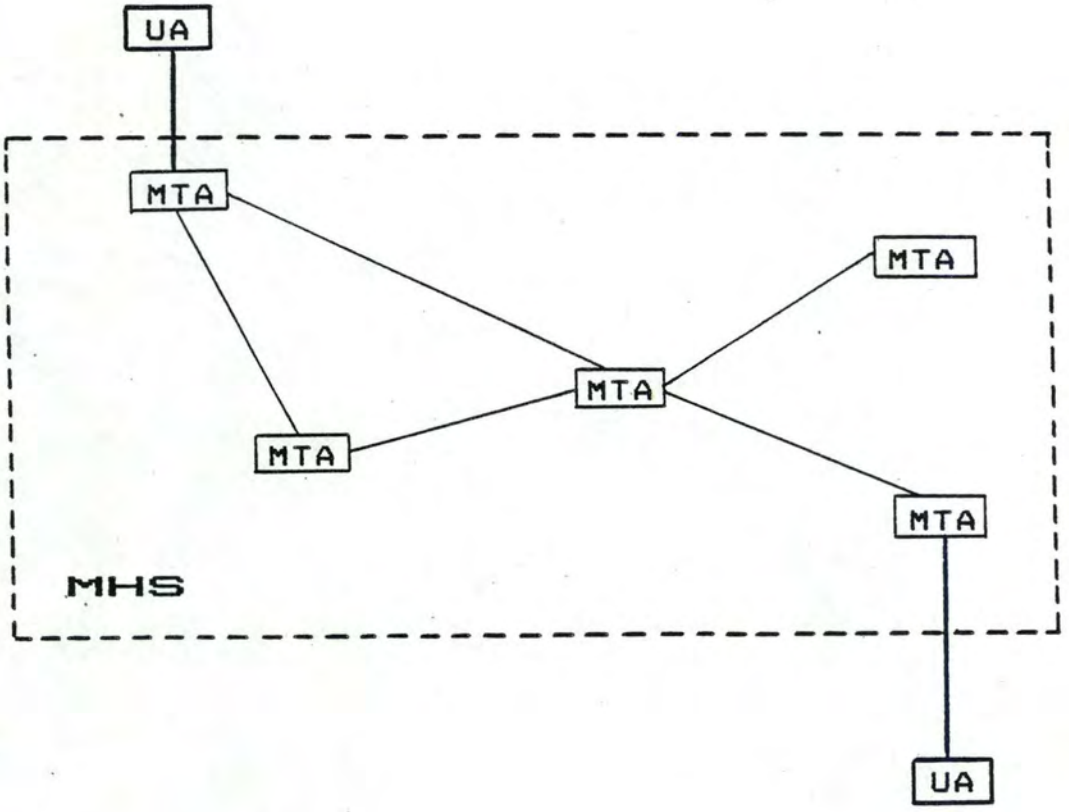


FIGURE 5.6 : TROISIEME SOUS-SYSTEME UTILE.

Parallèlement à ces trois étapes, d'autres simplifications peuvent réduire les sous-systèmes utiles. Elles consistent en l'omission de certains services lors d'une première implémentation. Ainsi, il n'est pas impensable de commencer par un MTA qui ignore le service "sonde" décrit au chapitre IV.

5.5 RESUME

Pour conclure, il convient peut-être de rassembler ici les grandes étapes que la méthodologie propose de suivre. Il s'agit:

- de la construction d'une architecture logique.

  structuration hiérarchique = découpe en niveaux

  structuration modulaire = identification des composants des niveaux

- des spécifications externes des modules

  i.e. ce qui est nécessaire et suffisant pour l'utilisation des modules.

- de la construction d'une architecture physique

  i.e. une architecture tenant compte des caractéristiques de la machine sur laquelle implanter le logiciel.

- de l'implémentation

  i.e. la conception des algorithmes répondant aux spécifications.

- du codage.

- des tests.

Il reste à souligner le fait que, le temps imparti au travail étant insuffisant pour tout mener à bien, le codage et les tests sont laissés à une continuation éventuelle de ce mémoire.

Chapitre VI.

SPECIFICATIONS EXTERNES D'UN MTA

6.1. INTRODUCTION.

Comme expliqué au chapitre précédent, la première étape dans la spécification d'un MTA est de considérer ce MTA comme une "boîte noire" à laquelle il n'est possible d'accéder que via un ensemble défini de primitives parfaitement spécifiées, qui réalisent l'interface entre l'UA et le MTA.

Ce chapitre va exposer les spécifications pré-post de ces primitives. Cependant, puisqu'il ne s'agit pas de spécifier des procédures statiques mais bien divers enchaînements possibles, ces spécifications pré-post sont accompagnées de tables ou de diagrammes d'états qui renseignent sur les conditions à remplir pour pouvoir utiliser les primitives.

Deux primitives traitant de l'établissement d'accès sont développées entièrement par la suite, l'une ayant pour origine l'UA, l'autre étant déclenchée par une initiative du MTA. Les spécifications ainsi que les tables et diagrammes d'états de toutes les autres primitives peuvent être trouvés dans l'annexe 1.

6.2. SPECIFICATIONS PRE-POST.

Afin que les types utilisés soient plus parlants, plus proches de leur signification, des noms de types spéciaux ont été employés pour les deux primitives suivantes :

NAME : string.

OR/NAME : string dont la structure est détaillée dans l'annexe 11.

ADRESS-ZONE : integer.

Ce type ADRESS-ZONE a été créé pour caractériser les types de données dont la longueur est inconnue ou variable. Dans ce cas, ADDRESS - ZONE contient la valeur de l'adresse (c'est-à-dire un pointeur) où sont mémorisées les données.

Ce choix conduit, bien entendu, à l'ajoût de certaines informations accompagnant les données stockées. Ces précisions supplémentaires concernent la longueur des données en mémoire ainsi que la découpe en champs quand ceux-ci existent.

Le type OR/NAME reprend en fait la formulation et la structure personnelle adoptées pour les noms O/R.

6.2.1. ETABLISSEMENT DE L'ACCES DEMANDE PAR L'UTILISATEUR.

SERVICE : ETABLISSEMENT DE L'ACCES DEMANDE PAR L'UTILISATEUR.

BUT:

L'UA utilise la primitive "logon" pour initialiser son interaction avec le MTA. Tous les services que le MTA offre à son UA - REGISTER, CONTROL, SUBMIT, PROBE, DELIVER, NOTIFY, CANCEL, CHANGE-PASSWORD - ne peuvent être demandés que si un accès a été établi entre l'UA et le MTA. Notons que cette liaison entre UA et MTA peut être désirée par l'UA (notre cas) ou par le MTA.

La demande d'établissement de liaison peut échouer si l'OR/NAME de l'UA ou son mot de passe sont incorrects. Si le MTA est surchargé, il refuse automatiquement la connexion quelle que soit la valeur de l'OR-Name et du password.

SPECIFICATIONS:

LOGON

Arguments

or-name ∈ O/R-NAME.

(nom de l'origine).

password ε STRING.

(mot de passe de l'UA).

préconditions

Résultats

Success-Indication ε (true, false).

(indication de succès ou de l'échec).

Failure-Reason ε (1,2,3).

Messages-waiting ε ADDRESS - ZONE.

postconditions

les résultats: Failure-Reason et Messages-Waiting sont obligatoires.

"Messages-Waiting" est l'adresse d'une zone qui contient le nombre de messages et le nombre total d'octets attendant d'être livrés, et ceci par priorité.

Ce résultat n'existe que si l'UA est abonné au service "Hold For Delivery".

Soient les conditions :

C1:O/R-Name incorrect (c'est-à-dire que le MTA ne connaît aucun UA qui possède ce nom).

C2:password incorrect (c'est-à-dire "Password" n'est pas le mot de passe de l'UA dont le nom public est O/R-Name).

C3:le MTA est surchargé.

C1	-	Y	N	N
C2	-	-	Y	N
C3	Y	N	N	N
Success-Indication = true				X
Success-Indication = false	X	X	X	
Failure-Reason = 1	X			
Failure-Reason = 2		X		
Failure-Reason = 3			X	

FIGURE 6.1 : TABLE DE DECISION POUR LA PRIMITIVE LOGON.

6.2.2. ETABLISSEMENT DE L'ACCES DEMANDE PAR LE MTA

SERVICE : MTL - LOGON.

BUT:

le MTA active ce service pour avertir un de ses UA qu'une connexion avec lui a été demandée et doit être établie.

SPECIFICATIONS:

MTL-LOGON-SIGN.

Arguments

MTA ∈ NAME.

(nom du MTA établissant l'accès).

Message-Waiting ∈ ADDRESS - ZONE.

Password ∈ STRING.

(chaîne de caractères qui identifie le MTA).

Préconditions

Message-Waiting est facultatif. Il n'est présent que si l'utilisateur a souscrit au service "Hold For Delivery". Les autres arguments sont obligatoires.

Message-Waiting est l'adresse d'une zone préalablement garnie par le MTA, contenant, pour chaque priorité, le nombre de messages et le nombre total d'octets qui attendent livraison.

Résultats

Postconditions

MTL-LOGON-RESP.

L'UA utilise cette primitive offerte par le MTA afin de répondre à un MTL-LOGON-SIGN reçu, après vérification des arguments de ce dernier.

Arguments

Success-Indication ∈ {true,false}.

{indication du succès ou de l'échec de la demande de connexion}.

Failure-Reason ∈ ADDRESS - ZONE.

{raisons de l'échec}.

préconditions

Failure-Reason est un argument facultatif. Sa présence dépend de la valeur de " Success-Indication" qui, lui, est obligatoire:

Success-Indication = true => Failure-Reason absent.

Success-Indication = false => Failure-Reason présent.

Soient les conditions suivantes :

C1:MTA, fourni par le MTL-LOGON-SIGN, est un nom inconnu.

C2:"Password", fourni par le MTL-LOGON-SIGN, est un mot de passe incorrect pour identifier le MTA.

C3:Message-Waiting fourni par le MTL-LOGON-SIGN, est une adresse inconnue.

C4:l'UA est trop occupé (impossible de prendre une nouvelle tâche en charge pour le moment):

Success-Indication = * (C1 $ C2 $ C3 $ C4)

c'est-à-dire

Success-Indication = true <=> µi 1 <?i <?4 Ci est fausse.

Success-Indication = false <=> £i 1 <? i <?4 Ci est vraie.

Failure-Reason est l'adresse d'une zone contenant un ou plusieurs entiers.

zone $i <=> Ci est vraie µi 1<? i<? 4

Résultats

postconditions

6.3. TABLE D'ETATS.

Comme déjà signalé au chapitre V.3, les tables et diagrammes d'états viennent compléter les spécifications pré-post, celles-ci étant insuffisantes pour exprimer l'aspect dynamique.

Suivent ici, les tables d'états pour l'établissement d'une connexion demandée par l'utilisateur ou le MTA.

6.3.1. ETABLISSEMENT DE L'ACCES DEMANDE PAR L'UTILISATEUR.

"UA LOGON"

	NON UA CONNECTE
LOGON	Si nom busy alors A1 si correct alors A2 UA CONNECTE sinon A3 sinon A3

TABLE 6.1. TABLE D'ETATS DE L'UA-LOGON

Ces tables contiennent des actions Ai qui sont détaillées ci-dessous :

A1:vérification des paramètres.

A2:remplir les paramètres de sortie de manière positive.

A3:remplir les paramètres de sortie de manière négative.

La liste des états possibles de l'UA repris dans ces tables sont :

NON UA CONNECTE :

le MTA n'est pas connecté avec l'UA qui veut ou avec qui il veut entrer en dialogue.

UA CONNECTE :

le MTA est connecté avec l'UA qui veut ou avec qui il veut entrer en dialogue.

6.3.2. ETABLISSEMENT DE L'ACCES DEMANDE PAR LE MTA.

Dans cette table, l'événement n'est plus l'arrivée d'un message venant de l'UA. Il faut donc exprimer le déclenchement des actions comme conséquence d'un autre événement ou d'une demande personnelle.

Les états suivants sont utilisés :

NON UA CONNECTE :

le MTA n'est pas connecté avec l'UA qui veut ou avec qui il veut entrer en dialogue.

UA CONNECTE :

le MTA est connecté avec l'UA qui veut ou avec qui il veut entrer en dialogue.

ATTENTE UA CONNECTE :

le MTA attend la réponse à une demande de connexion avec un de ses UA.

MTL-LOGON

	NON UA CONNECTE	ATTENTE UA CONNECTE
arrivée d'un message à transmettre à un UA non connecté ou décision du MTA de changer ses restrictions.	MTL-LOGON-SIGN ATTENTE UA CONNECTE	
MTL-LOGON-RESP		si réponse positive alors UA CONNECTE sinon NON UA CONNECTE

TABLE 6.2. TABLE D'ETATS DU MTL-LOGON.

Ceci termine les spécifications externes des primitives d'établissement de connexion entre UA et MTA. Les spécifications externes, dans leur entièreté, se trouvent dans l'annexe 1.

6.4. LES DIAGRAMMES D'ETATS.

Cette section va permettre de représenter les différents états que le MTA peut atteindre, ainsi que la manière dont il passe d'un état à un autre grâce au déclenchement d'un événement extérieur ou à l'exécution d'une opération interne.

Comme vu au chapitre III, les fonctionnalités du MHS sont nombreuses : permettre aux UA d'envoyer et recevoir des messages, assurer le relais de messages arrivés via les couches inférieures de OSI, offrir un service de livraison différée, de sonde, assurer les conversions nécessaires, la production et l'analyse de MPDU,... On peut facilement remarquer que ces fonctions peuvent être séparées en plusieurs classes :

l'interaction avec les UA

l'interaction avec les MTA voisins, et donc la couche OSI qui lui est inférieure

les fonctionnalités propres au MTA (routage, conversion, traduction selon P1, ...)

Ces trois classes de fonctions paraissent assez indépendantes les unes des autres. Or le formalisme des automates à états finis présenté brièvement au chapitre V.3 permet la représentation d'un système sur base de plusieurs machines abstraites qui cohabitent, tout en conservant leur indépendance. Dès lors, s'installe l'idée d'utiliser trois machines abstraites pour représenter un MTA. Chacune peut se charger d'une des classes de fonctions déterminées auparavant. Pour cela, elle reconnaît des événements précis. Les états accessibles pour elle forment un sous-ensemble des états admis pour le MTA entier.

Puisque chaque machine est autonome, bien vite on conclut qu'il est possible pour elles d'agir simultanément, sans se gêner. Et c'est de là que naît la notion de parallélisme : un MTA, formalisé grâce à trois machines abstraites, pourra gérer au même moment son dialogue avec un UA, son dialogue avec un MTA voisin et les fonctions qui lui sont propres.

6.4.1. LES TROIS MACHINES ABSTRAITES.

Les trois machines abstraites introduites, elles peuvent à présent être quelque peu détaillées.

La première permet de représenter l'interaction entre le MTA et un UA. Le diagramme d'états qui la formalise est proposé à la figure 6.2. Cette première machine est appelée M1 dans la suite.

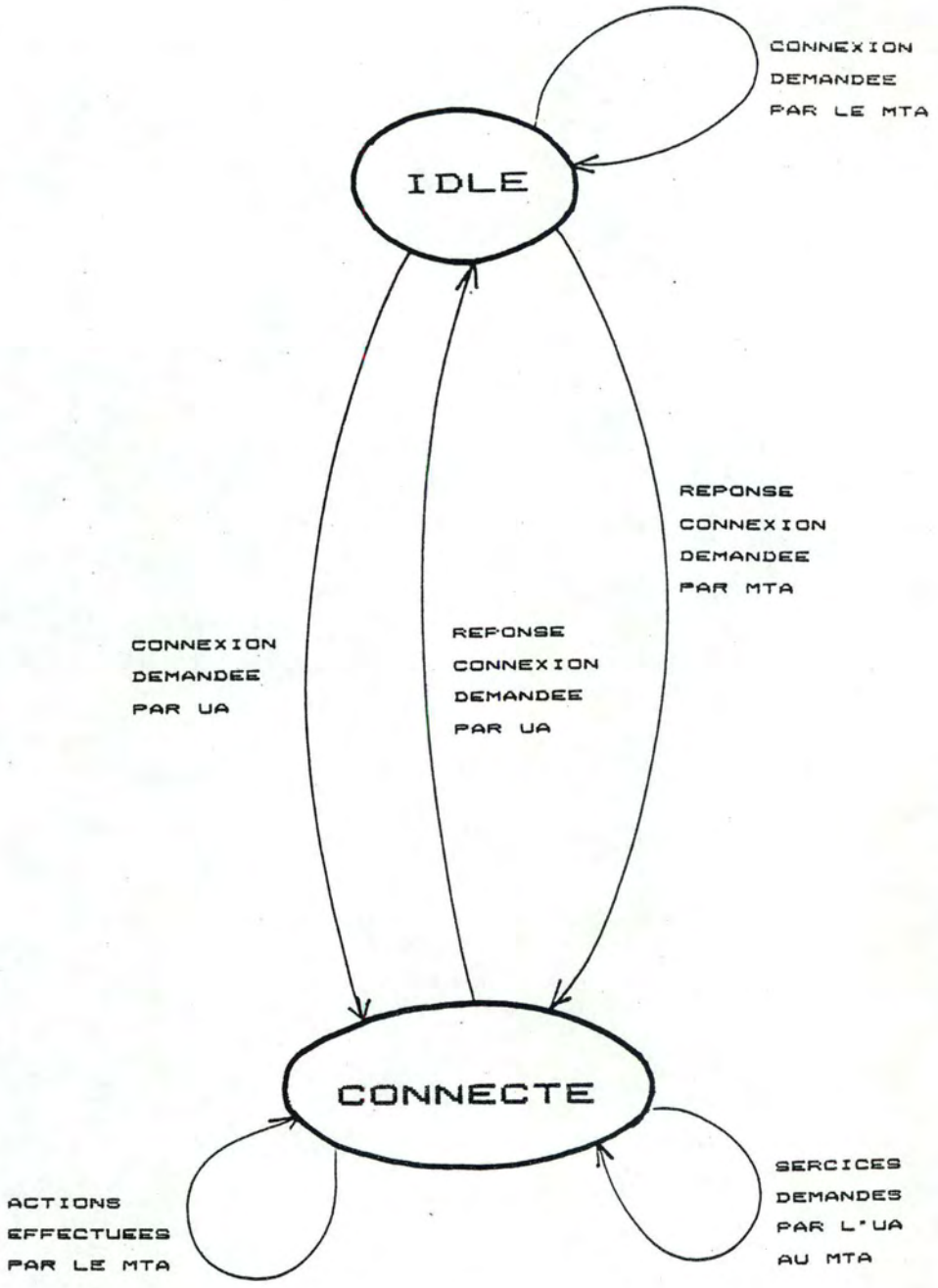


FIGURE 6.2. MACHINE ABSTRAITE REPRESENTANT L'INTERACTION UA-MTA.

La signification des états est la suivante :

IDLE : la machine est au repos

CONNECTE : le MTA est connecté à l'UA i.e. un lien a été établi entre eux

La deuxième machine abstraite nommée M2 s'occupe, d'une manière semblable, de l'interaction entre le MTA et un de ses voisins, conformément à la figure 6.3.

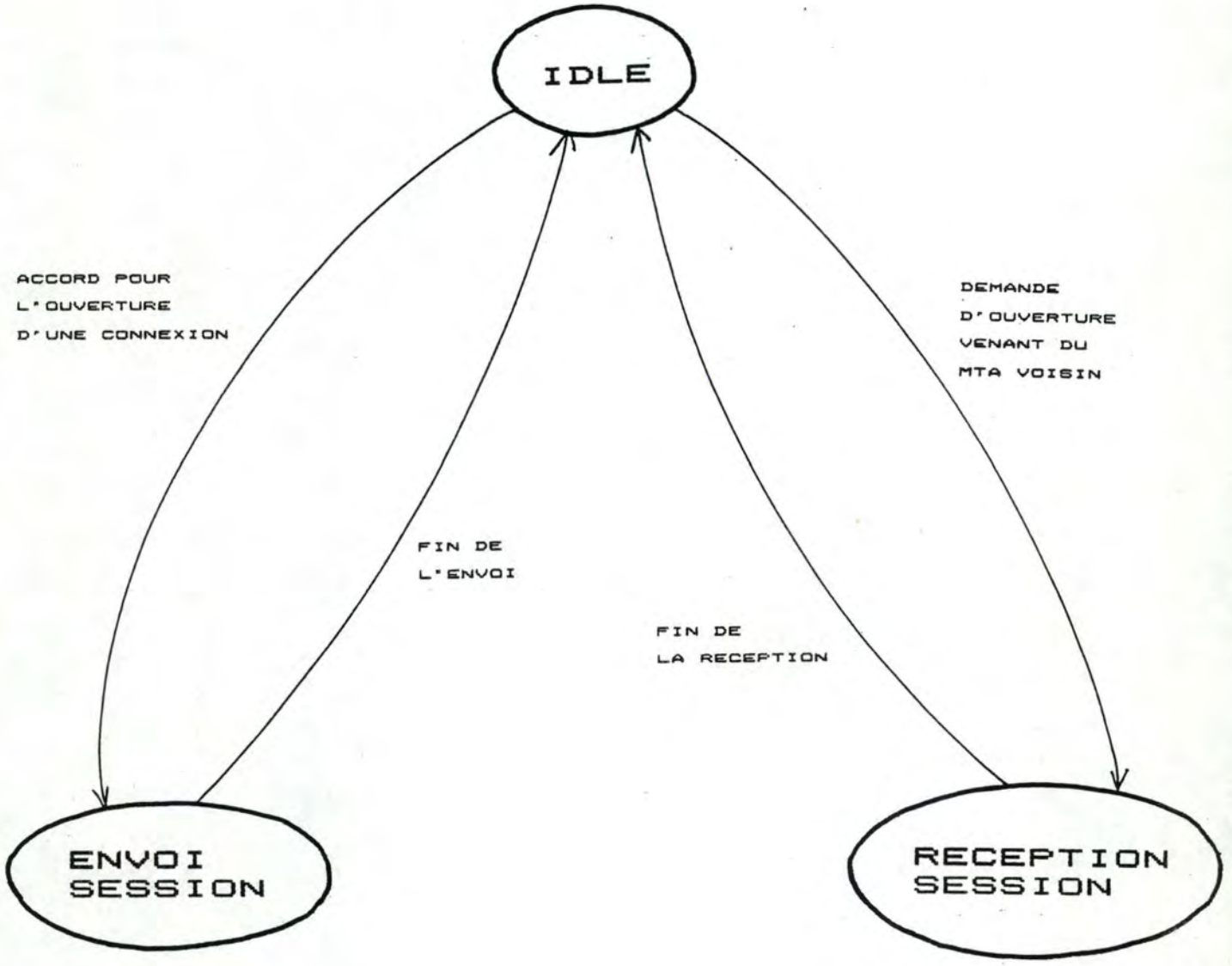


FIGURE 6.3. MACHINE ABSTRAITE REPRESENTANT L'INTERACTION MTA-COUCHE SESSION.

Les états sont repris ci-dessous :

IDLE : la machine est au repos

ENVOI SESSION : la machine doit envoyer un message au MTA voisin, via la couche session

RECEPTION SESSION : la machine traite un message qui arrive du MTA voisin par l'intermédiaire de la couche session.

La figure 6.4. illustre la dernière machine (M3), celle qui effectue toutes les tâches attribuées au MTA.

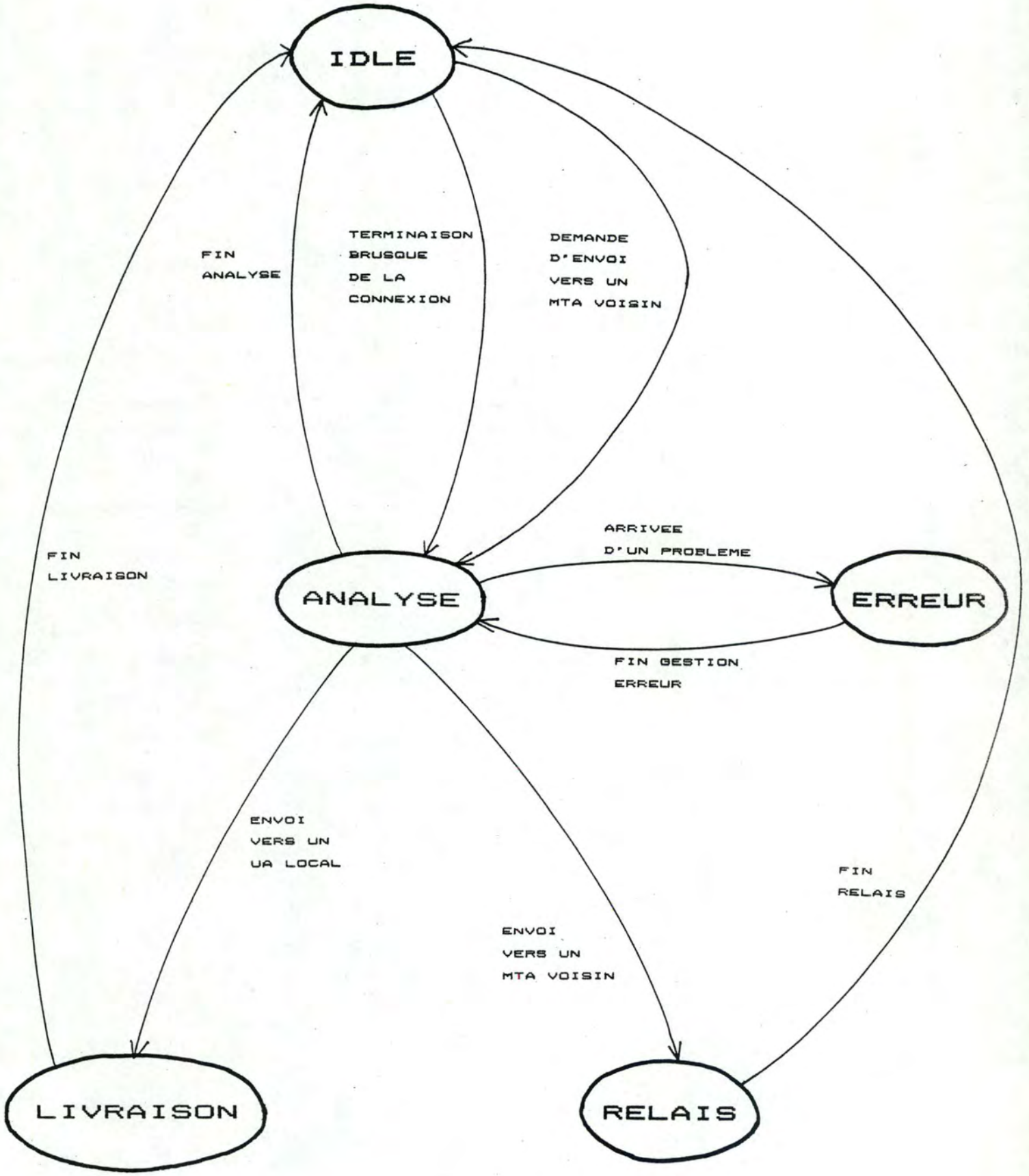


FIGURE 6.4. MACHINE ABSTRAITE REPRESENTANT LES FONCTIONS DU MTA.

La liste des états est la suivante :

- IDLE : la machine est au repos
- ANALYSE : la machine travaille afin de remplir une des fonctions propres du MTA (routage, vérification de conformité à P1, création de MPDU)
- LIVRAISON : la machine atteint cet état lorsqu'en l'état ANALYSE elle détermine qu'elle doit effectuer la livraison d'un message à un UA local
- RELAIS : la machine atteint cet état lorsqu'en l'état ANALYSE elle doit relayer un message, qu'il vienne de l'extérieur ou d'un UA local
- ERREUR : la machine atteint cet état lorsqu'en l'état ANALYSE elle détermine qu'elle détecte une erreur (problème au niveau du routage, bouclage, impossibilité de livrer à un UA local)

Cependant, il faut remarquer que, bien que fortement indépendantes, ces machines abstraites ont pour tâche d'effectuer toutes les actions d'un MTA. A elles trois, elles assurent les fonctionnalités d'une seule entité.

Il est donc naturel de devoir envisager une certaine synchronisation entre ces machines.

Lorsqu'un message doit être livré à un UA local, le MTA passe dans l'état LIVRAISON. Une fois dans cet état, il faut trouver une machine M1 pour assurer la livraison. On utilise, pour ce faire, le petit algorithme ci-dessous :

SI il existe une machine M1j travaillant avec l'UA destinataire

ALORS

l'utiliser (càd lui demander de faire la livraison d'un message ou d'une notification)

SINON

voir s'il existe un M1j libre

SI oui

ALORS

la sélectionner

lui faire faire un MTL-LOGON-IND

lui signaler que c'est pour faire la livraison d'un message ou d'une notification

SINON

on attend puis on recommence

Lorsqu'un message doit être relayé vers un MTA adjacent, le MTA arrive dans l'état RELAIS. Dans ce cas, il faut trouver une machine M3 en utilisant ce petit algorithme :

SI il existe une machine M3j travaillant avec le MTA visé

ALORS

attendre que cet M3j soit en IDLE

lui indiquer qu'il faut faire un envoi

```
SINON
  SI il existe un M3j quelconque disponible
  ALORS
    lui dire de faire un S-CONNECT-REQ
  SINON
    attendre puis recommencer
```

Ce chapitre nous a exposé les spécifications externes des modules fonctionnels et nous a mis sur la voie du parallélisme, parallélisme dont nous ferons le sujet principal des chapitres suivants.

Chapitre VII

DECOMPOSITION DE LA COUCHE APPLICATION.

7.1 INTRODUCTION.

Comme on l'a vu lors de la présentation générale du MHS (paragraphe III 2.4), la couche 7 du modèle OSI, dans le cadre de la messagerie électronique, peut être définie comme la superposition de deux sous-couches : le User Agent Layer (UAL) et le Message Transfer Layer (MTL).

Le but de ce chapitre est, dans un premier temps, de rappeler en quelques mots les protocoles qui sont définis à l'intérieur de ces sous-couches. La suite du chapitre détaille une entité MTA en expliquant les différents éléments qui la composent.

7.2 LES PROTOCOLES DE LA COUCHE APPLICATION.

7.2.1 DEFINITION DES PROTOCOLES.

Les protocoles définis à la couche application du modèle OSI, dans le cadre du MHS, sont au nombre de trois. Le protocole Pc, qui est en fait une classe de protocoles, se situe au niveau supérieur de la couche 7, c'est-à-dire entre les différentes entités du niveau UAL. Ces protocoles définissent la syntaxe et la sémantique des messages entre

les UA. A chaque catégorie de messages correspond un protocole de la classe Pc. Dans le cadre de la messagerie interpersonnelle, Pc est "instancié" à P2.

Les protocoles P1 et P3 définissent les règles permettant aux entités de la couche MTL de se comprendre. Le protocole P1 caractérise la syntaxe, la sémantique et les règles à respecter pour que se fasse le relais, à travers le MTL, de messages soumis et à destination d'UA.

P3, quant à lui, permet à un MTA de converser avec une Submission Delivery Entity (SDE) lorsqu'un message doit être reçu ou transmis à un UA non associé à un MTA. Cette entité SDE offre à l'UA tous les services que lui fournirait un MTA et donc remplace le MTA absent en jouant le rôle d'intermédiaire entre l'UA et un MTA que lui procure le domaine de gestion auquel il est appartient. Ce type d'interaction, faute de temps, est ignoré dans la suite du travail.

7.2.2 LES UNITES DE PROTOCOLES.

Lorsque deux entités d'une même couche doivent entrer en contact, elles doivent le faire en respectant le protocole défini pour la communication à ce niveau. Le dialogue est possible grâce à l'utilisation d'unités de données de protocole (PDU) qui transportent l'information.

Ces unités de données de protocole sont appelées UAPDU pour un transfert d'informations entre deux UA, et MPDU lorsqu'il s'agit de données entre deux entités communiquant grâce au protocle P1.

Comme l'illustre la figure 7.1, les éléments du protocole P1 peuvent être classés en deux groupes : les user MPDU (UMPDU) qui véhiculent entre deux MTA les messages soumis par un UA pour le relais et la livraison à un ou plusieurs UA destinataires, et les services MPDU (SMPDU) qui transportent des informations à propos des messages entre MTA. Ces SMPDU peuvent être des Probe MPDU ou des Delivery Report MPDU. Les Probe MPDU permettent de tester l'envoi ultérieur vers un UA d'un message ayant les mêmes caractéristiques que celles qui sont véhiculées dans le Probe MPDU. Les Delivery Report MPDU rapportent au MTA et à l'UA origines des notifications de livraison ou de non-livraison créées, suite à la réception d'un UMPDU ou un Probe MPDU, par le MTA destinataire.

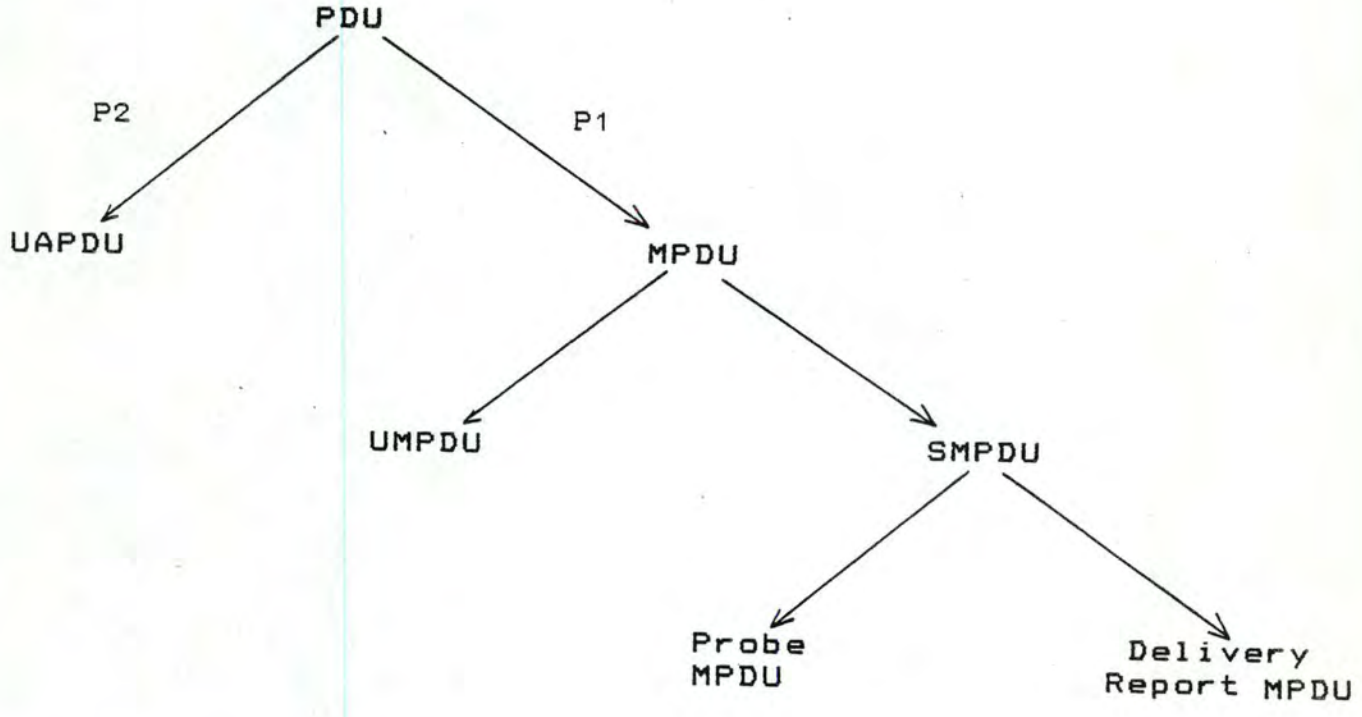


FIGURE 7.1 : LES UNITES DE DONNEES DE PROTOCOLES DE LA COUCHE APPLICATION.

7.3 DECOMPOSITION D'UN MTA.

7.3.1 VUE GLOBALE DU MTA.

Un MTA peut être modélisé comme ayant trois parties : le Message Dispatcher (MD), l'Association Manager (AM) et le Reliable Transfer Server (RTS). Ainsi, la figure 7.2 montre un MTA auquel deux UA sont connectés et qui est adjacent à deux autres MTA.

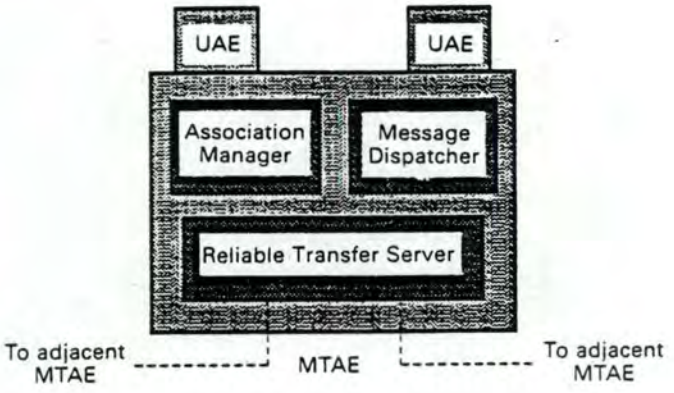


FIGURE 7.2 : MODELE D'UN MTA.

Le rôle du MD est d'effectuer les actions en rapport avec les éléments du protocole P1 indiquées dans le MPDU qu'il reçoit d'un MTA voisin ou qui résulte de messages soumis par un UA local. Le rôle de l'Association Manager est d'établir, de contrôler et de supprimer les interactions avec un MTA voisin, interactions appelées associations. Le RTS, quant à lui, utilise la couche qui lui est directement inférieure dans le modèle OSI pour offrir un support aux associations ainsi que pour assurer le transfert fiable des MPDU via ces associations. Il établit, pour ce faire, un lien, nommé connexion session,

avec le MTA voisin. Le dialogue entre les UA et le MD, l'AM et le RTS, ainsi qu'entre le RTS et la couche inférieure du modèle OSI se fait à l'aide de primitives offertes à leur utilisateur par le MD, le RTS et les couches inférieures du modèle OSI.

7.3.2 LE MESSAGE DISPATCHER.

Effectuant les fonctions importantes du MTA et assurant la liaison avec les UA , le Message Dispatcher (MD) peut être considéré comme le coeur de cette entité. Les tâches qu'il doit remplir sont énumérées sommairement ci-dessous.

Le MD doit :

- s'occuper du routage, c'est-à-dire du choix du MTA voisin sur la route que le message doit emprunter pour aller de l'UA origine à l'UA destinataire;
- vérifier si les MPDU qu'ils utilisent sont conformes au protocole P1;
- créer les rapports de livraison, les Probe MPDU et les UMPDU;
- créer les copies de messages lorsque ceux-ci doivent atteindre les UA destinataires par des chemins différents;
- convertir le contenu du message lorsque cette conversion est permise ou demandée et que l'UA destinataire exige un certain type de contenu pour les messages qu'il reçoit;
- vérifier si le message qu'il relaie n'est pas déjà passé par ce MTA (pour éviter les boucles lors du routage).

Lorsque le MD s'aperçoit que le MPDU reçu ou créé doit être routé vers un MTA voisin, il passe ce MPDU à l'AM qui s'occupe d'ouvrir une association vers le MTA voisin déterminé (si celle-ci n'existe pas encore).

7.3.3 L'ASSOCIATION MANAGER.

Le rôle principal de l'AM est d'établir, de gérer et de clôturer les associations en vue d'une communication avec un MTA voisin.

D'autre part, l'AM doit se conformer aux accords bilatéraux acceptés par les deux MTA qu'il veut mettre en relation. Ces accords portent notamment sur

- le nombre d'associations qui peuvent exister simultanément;
- le mode de dialogue utilisé : dans un sens ou dans les deux sens à l'alternat;
- le choix du MTA qui est responsable de l'établissement des associations;
- la durée d'existence des associations. Celles-ci peuvent être permanentes ou établies à la demande.

Dans le cadre de ce mémoire, certains choix ont été faits à propos de ces accords; le nombre d'associations entre deux MTA est fixé à 1, le mode de dialogue est dans les deux sens à l'alternat et les associations sont détruites à la demande. Le dialogue à l'alternat est géré logiquement au moyen de la notion de tour. Celui qui possède le tour est en mesure d'envoyer des informations et de fermer l'association. Celui qui n'a pas le tour ne peut que recevoir des données ou, s'il veut devenir émetteur,

activer une demande de tour. Pour mener à bien ces tâches, l'AM a la possibilité de

- s'occuper de l'établissement, du contrôle et de l'abandon de ces associations;
- demander le tour;
- commander un transfert en utilisant à cet effet les primitives de services qui lui sont offertes par le RTS.

7.3.4 LE RELIABLE TRANSFER SERVER.

Dans la couche application, le RTS est l'entité qui est responsable de la création physique et de l'entretien des associations entre deux MTA pairs. Le RTS doit aussi assurer le transfert fiable des MPDU via ces associations.

Le RTS offre ses services à l'AM grâce à un ensemble de primitives permettant à l'AM de lui signaler quand il veut établir ou détruire une association, demander le tour pour pouvoir envoyer un message vers le MTA voisin, ...

Pour mener à bien ces demandes, le RTS dispose de primitives offertes par la couche session (la couche présentation étant transparente dans le cas de l'application MHS).

7.4. CONCLUSION

Ce paragraphe termine la présentation de la couche MTL. Cette division en trois blocs ayant des fonctionnalités différentes va servir de base pour développer l'architecture logique présentée au chapitre suivant.

Chapitre VIII.

CONCEPTION D'UNE ARCHITECTURE LOGIQUE

Comme la méthodologie de développement de logiciels décrite auparavant le prescrit (cfr. chapitre V), il convient à présent de construire une architecture logique pour le logiciel à réaliser, soit le MTA.

Cette étape, on le sait, permet d'organiser le système en niveaux distincts et ordonnés de familles de composants, ou modules, de manière à réduire les relations entre composants et à obtenir une structure maîtrisable.

Il faut donc choisir une relation sur laquelle baser la hiérarchie. Et comme discuté précédemment, la relation UTILISE semble présenter des propriétés attrayantes.

8.1. STRUCTURATION HIERARCHIQUE DU SYSTEME

L'étape logique commence, en général, par la détermination des niveaux annoncés, dont les composants utilisent des composants de niveaux inférieurs.

La découpe logique obtenue dans le cadre du MTA présente sept niveaux qui sont, dans la suite, successivement cités et brièvement qualifiés.

Le **niveau 6**, souvent appelé "niveau des modules fonctionnels", abrite, comme son nom l'indique, des composants fonctionnels dérivés - directement ou après agrégation/décomposition - des fonctions définies au cours de l'analyse fonctionnelle.

En fait, la norme X400 assure, par elle-même, une part de cette recherche puisqu'elle propose déjà une découpe fonctionnelle du MTA. C'est pourquoi il n'est en rien étonnant de retrouver à ce niveau les blocs MD, AM et RTS.

Le **niveau 5**, quant à lui, forme un véritable noyau fonctionnel, regroupant toutes les fonctions de base utilisées par celles de plus haut niveau, à savoir tous les gérants de files d'attente et de tables, les conversions, les copies et destructions de messages, sans oublier l'algorithme de routage. Il s'agit là de modules réalisant des tâches importantes, certes, et liées à l'aspect fonctionnel, mais qui ne sont que des outils d'aide pour les véritables grandes fonctions du MTA.

Le **niveau 4** permet de passer au degré de raffinement à partir duquel les modules décrits ne sont plus détectables lors de l'analyse fonctionnelle. Ici se placent les analyseurs de données reçues (primitives, signaux,...), les vérificateurs de paramètres, les procédures chargées de formuler les réponses aux requêtes,... . En résumé, tout ce qui touche de près ou de loin aux communications entre les grandes fonctions ainsi qu'à la gestion des entrées/sorties du MTA.

Arrivent ensuite les outils : outils de base (**niveau 3**) et outils primitifs (**niveau 2**). Les premiers ont trait à la manipulation des messages à analyser par le MTA. Ils permettent, par exemple, de vérifier la conformité d'un message à P1, ou de modifier l'enveloppe d'un message,... . Les seconds outils sont utilisés par les précédents et offrent des services primitifs axés sur la vérification de conformité à P1, non plus pour un message entier, mais pour une partie : un champ.

Tous ces outils sont liés à l'application et s'efforcent de coder ou décoder les messages selon la syntaxe présentée dans la norme X409.

Le niveau le plus bas, le **niveau 1**, est le plus rapproché de la machine et rassemble les modules que le système d'exploitation UNIX propose, dans le cadre de la gestion des processus, par exemple, modules indispensables au bon fonctionnement de tous les autres.

Il reste alors à coiffer la hiérarchie construite par un sommet qui se charge d'activer les niveaux inférieurs.

Ainsi, c'est au **niveau 7** que la notion de choix entre plusieurs scénarii possibles intervient. En effet, le MTA ne peut pas être une séquence de procédures dont les enchaînements sont figés ! Il est, au contraire, divisé en modules qui entrent en jeu lorsqu'une demande de niveau supérieur l'exige. Ce niveau supérieur est matérialisé par l'UA. Par l'intermédiaire de ses requêtes et de la succession de celles-ci, il déclenche une chaîne de processus internes au MTA, qui aide à répondre au travail demandé. C'est ainsi que l'UA est placé au niveau 7 et joue, en quelque sorte, un rôle de coordinateur pour les modules du MTA - coordinateur au sens large puisqu'il n'est pas lui-même un module du logiciel dont il doit assurer la coordination.

8.2. STRUCTURATION MODULAIRE DU SYSTEME

Une fois que les degrés de la structure hiérarchique sont définis, il reste encore à déterminer de manière exacte quels sont les modules que chacun regroupe. Une représentation schématique en est donnée ci-dessous à la figure 8.1.

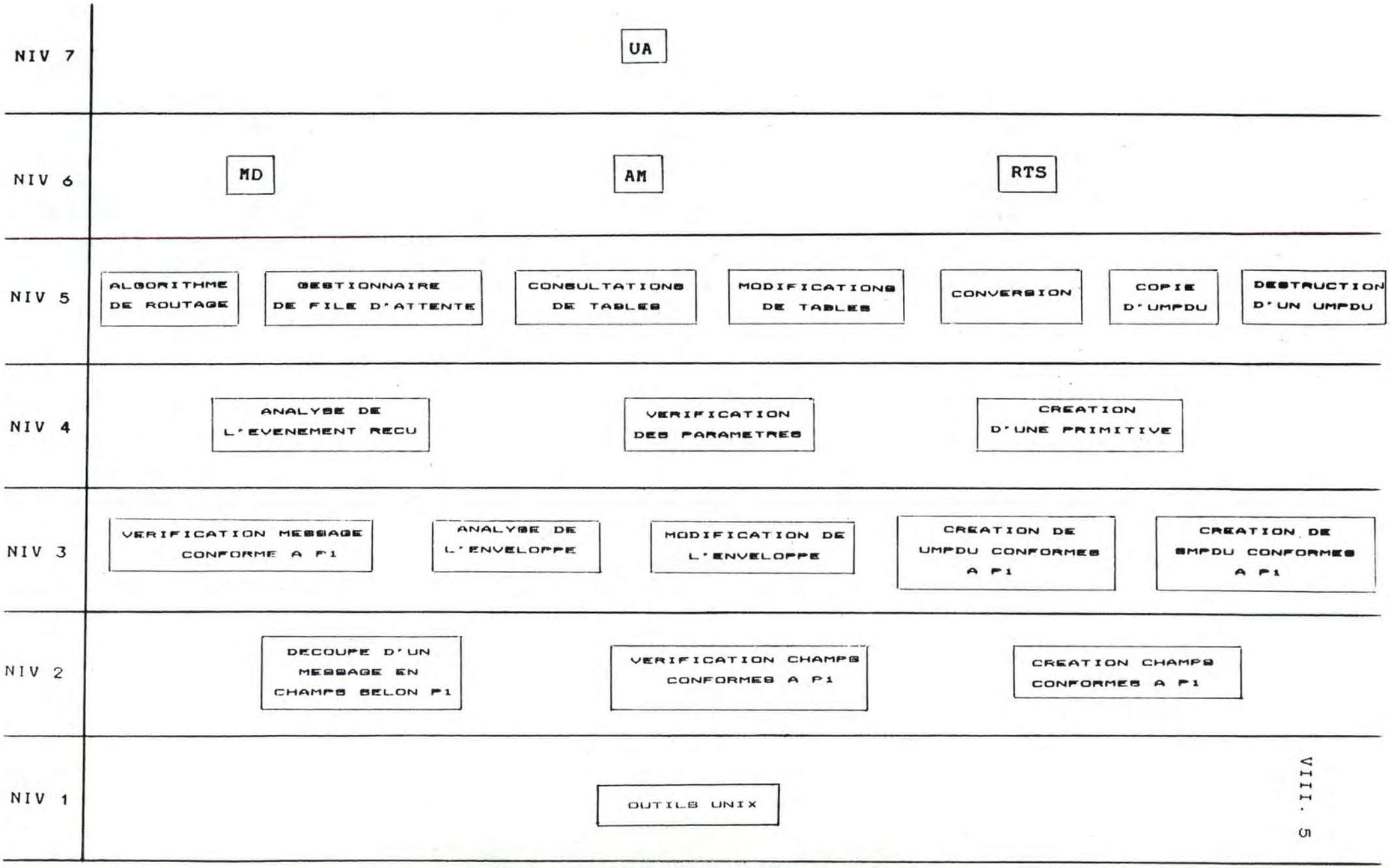


FIGURE 8.1. : STRUCTURE MODULAIRE DU MTA

Comme déjà précisé, la hiérarchie a été conçue grâce à la relation UTILISE, qui permet d'organiser les composants du système en une structure stable.

Il convient d'exprimer clairement les interactions entre les modules des différents niveaux. A cet effet, des tables à deux entrées, trés parlantes, sont employées. Ainsi, la figure 8.2 permet de décrire le fait que les modules M_1 et M_2 du niveau i sont liés par la relation UTILISE aux M_1*, M_2*, M_3* assemblés au niveau j : M_1 utilise M_1* et M_2*; M_2 utilise M_1* et M_3*.

UTILISE ↘ NIVEAU I NIVEAU J (J<I)	M_1	M_2
M_1*	*	*
M_2*	*	
M_3*		*

FIGURE 8.2 : EXEMPLE DE REPRESENTATION POUR LES RELATIONS INTER-NIVEAUX.

En suivant le même formalisme, il est simple d'énumérer toutes les interactions entre les sept niveaux de l'architecture logique du MTA. C'est là le but des figures 8.4 à 8.8.

NIVEAU 7

NIV 6	NIV 7	UA
MD		*
AM		*
RTS		*

FIGURE 8.3 : LA RELATION UTILISE AU NIVEAU 7.

NIVEAU 6

NIV 6 / NIV 5	MD	AM	RTS
algo routage	*		
gestion. file d'attente	*	*	*
consultations tables	*	*	*
modifications tables	*	*	*
conversion	*		
copies UMPDU	*		
destruction UMPDU	*		*
NIV 4			
analyse événement reçu	*	*	*
vérification paramètres	*	*	*
création d'une primitive	*	*	*
NIV 3			
vérif. mes. conforme P1	*		
analyse enveloppe	*		
modification enveloppe	*		
création UMPDU conformes P1	*		
création SMPDU conformes P1	*		
NIV1			
création pipes	*	*	*
gestion pipes	*	*	*
création processus	*	*	*
gestion processus	*	*	*

FIGURE 8.4 : LA RELATION UTILISE AU NIVEAU 6

NIVEAU 5

NIV 5 / NIV 3	algo routage	conversion	gestion file d'attente	consultations tables	modifications tables	copies UMPDU	destruction UMPDU
modif. enveloppe		*					
NIV 2							
découpe mess. champs P1		*					
NIV 1							
gestion processus	*	*	*	*	*	*	*
création processus	*	*	*	*	*	*	*

FIGURE 8.5 : LA RELATION UTILISE AU NIVEAU 5

NIVEAU 3

NIV 3 / NIV 2	vérif. mess. conformes P1	analyse enveloppe	modifications enveloppe	création UMPDU conformes P1	création SMPDU conformes P1
découpe mess. champs P1	*	*	*		
vérif. champs conformes P1	*				
création champs conformes P1			*	*	*
NIV 1					
création processus	*	*	*	*	*
gestion processus	*	*	*	*	*

FIGURE 8.6 : LA RELATION UTILISE AU NIVEAU 3

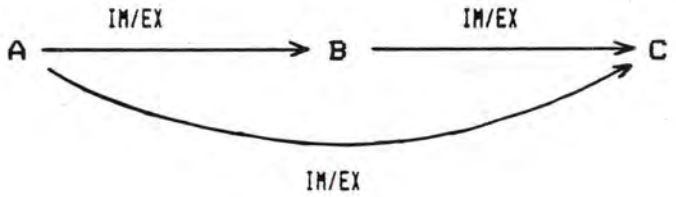
NIVEAU 2

NIV 1 \ NIV 2	découpe mess. champs P1	vérif. champs conformes P1	création champs conformes P1
création processus	*	*	*
gestion processus	*	*	*

FIGURE 8.7 : LA RELATION UTILISE AU NIVEAU 2

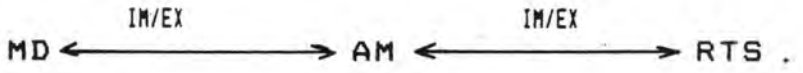
Il est important de remarquer que les modules d'un même niveau peuvent être unis eux aussi par des relations, non plus la relation UTILISE puisqu'elle impose une différence de niveaux, mais d'autres relations.

Ainsi, dans le cas du MTA, pour les trois modules du niveau fonctionnel, intervient la relation importe/exporte (notée Im/Ex). La définition en est illustrée ci-après : on écrit par exemple :



lorsque B, pour son exécution correcte, nécessite des informations fournies par l'exécution de A, avant de passer à son tour des données à C, lequel C dispose également de résultats en provenance de A.

Dès lors, au niveau 6, au vu des interactions décrites par la norme entre les blocs MD, AM et RTS, il apparaît que



Ceci achève la construction de l'architecture logique du MTA, sur laquelle va se greffer une architecture physique, plus adaptée aux caractéristiques physiques de la machine sur laquelle implanter le MTA.

8.3. CHOIX PERSONNEL AU NIVEAU LOGIQUE : LE PARALLELISME

Jusqu'à présent, l'architecture logique présentée est déduite de l'aspect fonctionnel proposé dans les recommandations X400.

Cependant, après lecture de cette norme, une idée assez personnelle s'est peu à peu dégagée. Du point de vue de l'utilisateur du MTA, soit l'UA, le MTA devrait être une boîte noire à laquelle il accède de manière simple et rapide. Or un MTA dessert plusieurs UA. C'est pourquoi, tout naturellement, la notion d'attente devant le MTA apparaît. Et puisque les actions du MTA suite à une requête d'un de ses UA ou d'un MTA voisin sont nombreuses, cette attente risque vite de prendre des proportions importantes.

Ce problème est, semble-t-il, assez proche de celui qui se pose dans le cadre d'un système d'exploitation gérant un gros système qui permet la multi-programmation et le time-sharing : chaque utilisateur se présente à un terminal au moment où il en éprouve le besoin, fait connaître sa présence et son désir de rentrer sur le système, et espère pouvoir être connecté et obtenir une réponse à ses demandes assez rapidement. Ainsi, de nombreuses personnes travaillent au même moment, soumettent leurs requêtes, de types très différents, et en attendent les résultats. Pourtant, vu de l'extérieur, il n'y a qu'un seul système d'exploitation qui gère l'ensemble. Il faudrait qu'il en soit de même pour le MTA : plusieurs UA qui interagissent, au même instant, avec un seul MTA.

Donc, pour éviter le désavantage certain de l'attente, il a été décidé qu'au niveau logique, le MTA conçu serait à même de traiter sur le champ - ou presque - toute demande de l'UA et d'un MTA adjacent. Pour ce faire, il est indispensable d'étendre le MTA de la norme à un MTA auquel s'ajoute l'idée de parallélisme. Ainsi, certaines parties du MTA doivent pouvoir tourner en parallèle afin de traiter plusieurs demandes simultanément.

Le nouvel aspect ne modifie que peu la conception logique de l'organisation du MTA. En effet, la hiérarchie construite précédemment est implicitement basée sur la découpe en fonctions du MTA, fonctions destinées à répondre aux requêtes émises par un UA générique; mais rien n'empêche ces fonctions de répondre à plusieurs UA et, pour cela, d'être actives au même moment ! Si bien que s'il existe plusieurs UA concurrents dans le système, l'architecture proposée auparavant doit encore être valable, moyennant - sans doute - l'ajoût de coordinateurs, coordinateurs au sens strict cette fois, afin de permettre l'agencement des diverses fonctions susceptibles de se dérouler au même moment.

L'introduction du parallélisme est une idée naturelle en soi. Cependant, il est important de souligner que, si le côté logique ne s'en trouve pas forcément énormément compliqué, la réalisation physique, quant à elle, se doit de résoudre de graves difficultés qui en découlent : gestion de diverses copies de plusieurs processus, communication entre ces copies, centralisation d'informations dans un environnement où les ressources partagées sont rarement admises, problème de synchronisation de processus,... Si bien que la conception d'un MTA parallèle représente une grande complication vis à vis du "simple" MTA prévu par la norme (où le mot "simple" est déjà tout relatif !). Ceci explique sans aucun doute l'aspect uniquement conceptuel de ce mémoire, et l'abandon - faute de temps - de tout codage et tests sur machine.

En conclusion, il reste à rappeler que dès à présent, on dispose d'une découpe en niveaux qui permettra, on l'espère, la construction d'un MTA structuré et jouissant des propriétés attribuées à un bon logiciel. De plus, l'idée d'un parallélisme ayant fait son chemin, le choix d'un MTA parallèle est définitivement adopté dans le cadre de ce travail. Et les modifications par rapport aux spécifications et à l'architecture déjà décrites vont être introduites progressivement dans les chapitres suivants.

Chapitre IX.

SPECIFICATIONS EXTERNES DES MODULES FONCTIONNELS.

9.1. INTRODUCTION.

Ce chapitre va fournir au lecteur les spécifications pré-post complètes des primitives externes des modules fonctionnels du MTA que sont le Message Dispatcher (MD), l'Association Manager (AM) ou le Reliable Transfer Server (RTS). Les spécifications des primitives offertes par la couche session étant fort techniques, le lecteur peut, pour une meilleure compréhension parcourir le paragraphe 9.4.

Ces spécifications vont être regroupées en trois parties :

les primitives offertes par le MD à l'UA

les primitives offertes par le RTS à l'AM

les primitives offertes par la couche session au RTS.

La deuxième partie de ce chapitre reprend les machines abstraites expliquées en 6.4.1. et replace, entre les états, les primitives dont on aura détaillé les spécifications.

9.2. PRIMITIVES OFFERTES PAR LE MD A L'UA.

Ces primitives sont celles qui sont activées par l'UA losqu'il veut établir une connexion avec le MTA ou bien envoyer un message vers un ou plusieurs UA destinataire(s).

Ces primitives sont en fait celles qui définissent la relation entre le MTA et l'extérieur. Leurs spécifications ont donc été détaillées dans le chapitre VI, traitant les spécifications externes du MTA.

9.3. PRIMITIVES OFFERTES PAR LE RTS A L'AM.

Comme pour les spécifications externes d'un MTA (chapitre VI), toutes les spécifications pré-post des primitives ne sont pas développées ici. Cette section se limite à la spécification de la primitive de transfert d'informations (TRANSFER) entre deux AM. Les autres primitives sont spécifiées dans l'annexe 2.

PROCEDURE DE TRANSFERT

BUT:

L'AM active cette procédure afin de demander le transfert fiable d'un MPDU sur une association préalablement établie avec l'AM d'un MTA voisin. Cela n'a de sens que si les RTS serveurs des deux AM sont reliés par une connexion session effective. C'est pourquoi, lorsque le RTS se voit demander un transfert, et dans le cas où la connexion session a été interrompue, il est amené à activer la procédure d'établissement de connexion session, avant de donner suite à la demande par la procédure de transmission des données (S-Data).

TRANSFER-REQ-SIGN

Argument

APDU ∈ ADDRESS - ZONE.

Transfer-Time ∈ TIME.

(période de temps pendant laquelle le RTS doit avoir assuré avec succès le transfert de l'APDU vers l'AM du MTA voisin).

préconditions

APDU est l'adresse d'une zone que contient l'APDU fabriqué par le MD.

Les deux paramètres sont obligatoires.

Résultats

postconditions

TRANSFER-IND.

Argument

APDU ∈ ADDRESS - ZONE.

préconditions

APDU est l'adresse d'une zone contenant l'APDU envoyé par le MTA origine.

Résultats

postconditions

9.4. PRIMITIVES OFFERTES PAR LA COUCHE SESSION AU RTS.

Les spécifications des primitives activées par le RTS se limitent, dans ce chapitre, à la primitive d'envoi de données via une connexion session (S-DATA). Les spécifications des autres procédures se trouvent dans l'annexe 2.

PROCEDURE S-DATA.

BUT :

Cette procédure permet au RTS de transférer, vers le RTS avec lequel il est en communication, un bloc de données dont la taille est au maximum égale à ce que peut envoyer un RTS entre deux points de synchronisation mineurs. Ce bloc de données transmis est appelé SSDU.

Pour effectuer un S-DATA, le RTS doit avoir les jetons. Après avoir envoyé un S-DATA, le RTS transmet un point de synchronisation mineure ou une fin d'activité.

SPECIFICATIONS:

S-DATA-REQ-SIGN

Argument

SS-User-Data ϵ ADDRESS - ZONE.

{cette adresse contient le bloc de données à transférer}.

préconditions

SS-User-DATA est l'adresse de la zone qui contient les données à transférer.

La taille du bloc de données ne peut pas excéder ce qui peut être envoyé entre deux points de synchronisation.

Résultats

postconditions

S-DATA-IND

Argument

SS-User-Data ϵ ADDRESS - ZONE.

{cette adresse contient le bloc de données à transférer}.

préconditions

SS-User-DATA est l'adresse de la zone qui contient les données à transférer.

La taille du bloc de données à transférer est inférieure à ce qui peut être envoyé entre deux points de synchronisation.

Résultats

postconditions

9.5. LES MACHINES ABSTRAITES.

Puisque toutes les primitives internes au MTA viennent d'être définies et spécifiées, les machines abstraites décrites en 6.4.1 peuvent être complétées grâce à l'ajoût des primitives citées ci-dessus. Celles-ci précisent grâce à la réception de quelle primitive une machine peut passer d'un état à un autre.

Pour connaître ce que cachent les états, ainsi que les méthodes de synchronisation de ces machines, le lecteur retournera à la section 6.4.1 où ont été définies dans le détail les machines abstraites.

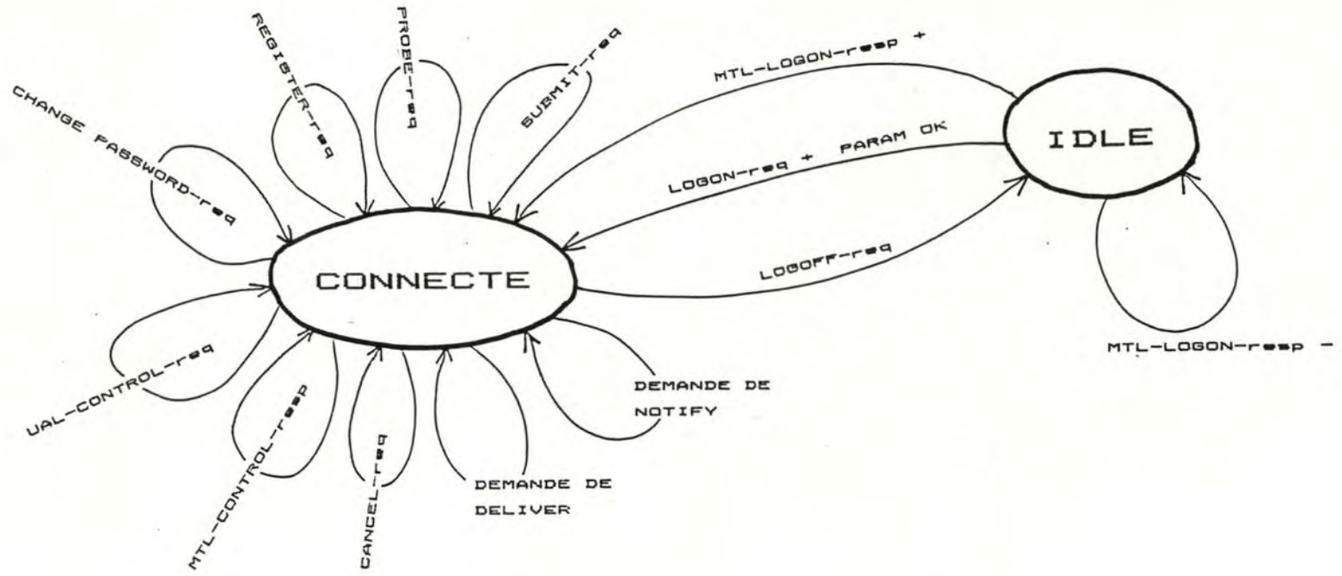
La figure 9.1 illustre la machine abstraite traitant l'interaction entre l'UA et le MTA. Cette machine reprend toutes les primitives que peut activer un UA.

La figure 9.2 complète la figure 6.2 en ajoutant toutes les primitives offertes par la couche session au RTS.

Pour terminer cette figure, il faudrait remplacer les états 'envoi session' et 'réception session' par les diagrammes complets présentés au chapitre VI.

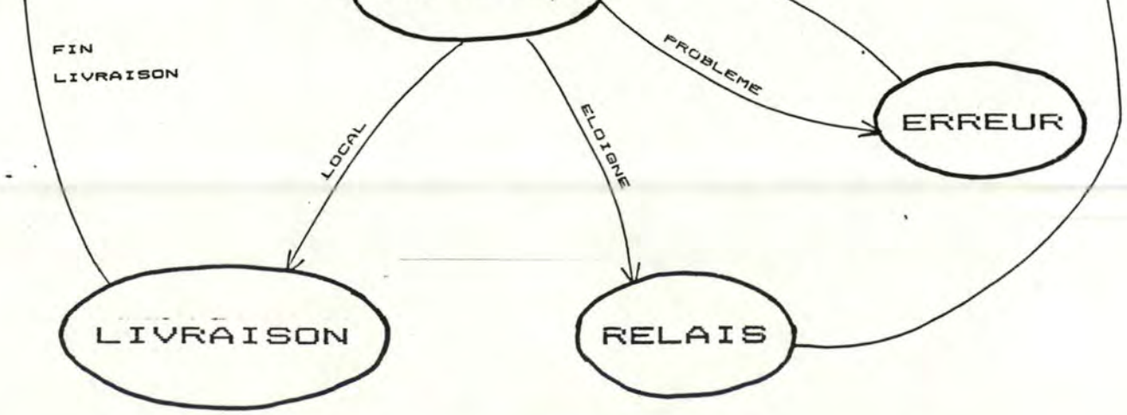
La figure 9.3 représente la machine abstraite commplète qui s'occupe de l'exécution des fonctionnalités principales du MTA.

Une fois toutes ces machines abstraites détaillées, nous pouvons passer à l'élaboration de l'architecture en parallèle. Ce sujet fait l'objet du chapitre X.



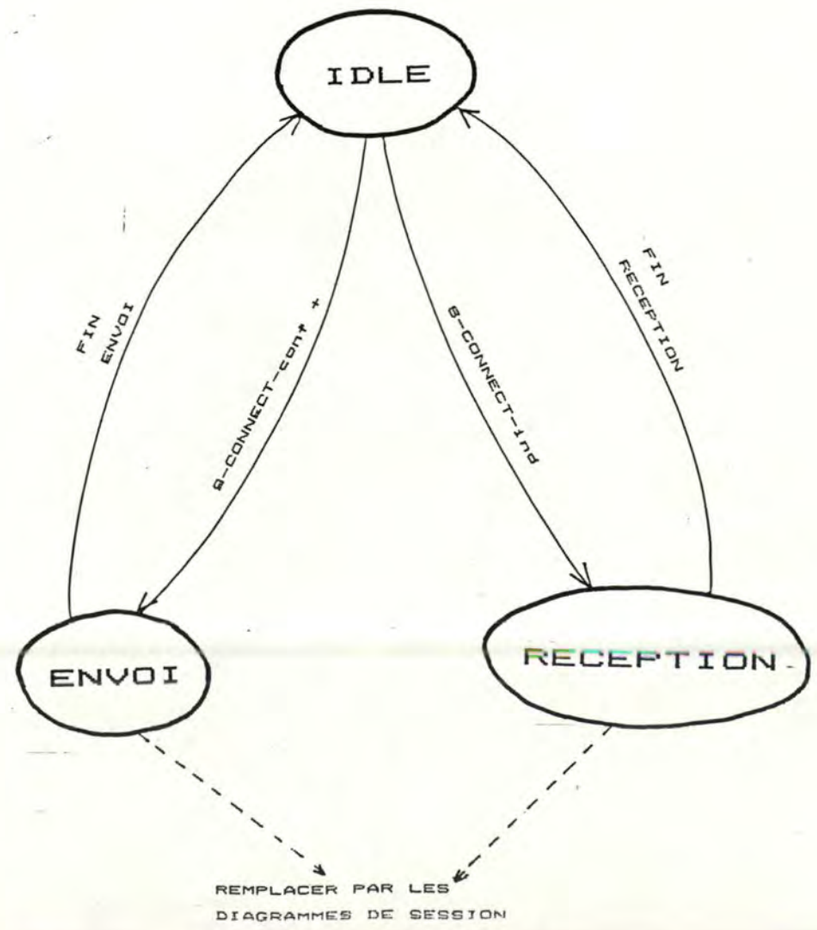
M.1.

FIGURE 7.1 : MACHINE ABSTRAITE REPRESENTANT L'INTERACTION ENTRE UN UA ET UN MTA (M1)



M.3.

FIGURE 7.3 : MACHINE ABSTRAITE PRESENTANT LES FONCTIONS DU MTA



M.2.

FIGURE 7.2 : MACHINE ABSTRAITE REPRESENTANT L'INTERACTION ENTRE LE MTA ET LA COUCHE SESSION (M2)

IX.6. LISTE DES PRIMITIVES.

Dans un souci de clarté, la liste ci-dessous regroupe toutes les primitives qui apparaissent dans la norme X400.

Un service s'effectue grâce à l'enchaînement de plusieurs primitives portant le même nom suivi d'un suffixe (req,ind,resp,conf pour request, indication, response, confirmation). Les tables fournissent les primitives requises pour chaque service.

PRIMITIVES MD

	REQ	IND	RESP	CONF
(UAL) LOGON	*			*
(MTL) LOGON		*	*	
LOGOFF	*			*
REGISTER	*			*
(UAL) CONTROL	*			*
(MTL) CONTROL		*	*	
SUBMIT	*			*
PROBE	*			*
DELIVER		*		
NOTIFY		*		
CANCEL	*			*
(UAL) CHANGE PASSWORD	*			*
(MTL) CHANGE PASSWORD		*		

PRIMITIVES AM

	REQ	IND	RESP	CONF
OPEN	*	*	*	*
CLOSE	*	*	*	*
TURN-PLEASE	*	*		
TURN-GIVE	*	*		
TRANSFER	*	*		
EXCEPTION		*		

PRIMITIVES RTS

	REQ	IND	RESP	CONF
S-CONNECT	*	*	*	*
S-ACTIVITY-START	*	*		
S-DATA	*	*		
S-SYNCH-MINOR	*	*	*	*
S-ACTIVITY-END	*	*	*	*
S-U-EXCEPTION-REPORT	*	*		
S-TOKEN-PLEASE	*	*		
S-ACTIVITY-RESUME	*	*		
S-CONTROL-GIVE	*	*		
S-RELEASE	*	*	*	*
S-U-ABORT	*	*		
S-P-ABORT		*		

Chapitre X.

NOUVELLE ARCHITECTURE PARALLELE

Un des choix les plus importants lors de la conception d'une architecture susceptible de servir de canevas à la réalisation d'un MTA est sans aucun doute l'introduction de la notion de parallélisme au niveau des tâches qu'il traite et des services qu'il rend, à un instant donné.

Ainsi, le rôle principal d'un MTA est de fournir à un ou plusieurs UA, rattachés à lui, la possibilité de soumettre et envoyer des messages, moyennant d'éventuels MTA relais. Ce travail implique de nombreuses analyses, vérifications syntaxiques, traductions selon le protocole P1, conversions indispensables pour permettre relais et livraisons, copies, routages, ouvertures et gestion de communications entre endroits éloignés, échanges d'informations entre entités via des unités de protocole ou l'activation de primitives standardisées,... . Autant d'actions à effectuer lors de la prise en charge d'un message amènent à penser qu'il serait intéressant de les classifier afin d'isoler celles qui pourraient se dérouler au même moment et de manière non concurrente.

La norme propose, on l'a vu, une séparation nette des fonctions du MTA puisqu'elles sont réalisées via l'interaction de trois blocs bien distincts :

- le MD, qui - véritable coeur du MTA- est chargé du relais, tout en dialoguant avec les UA, s'occupant du routage et exécutant les livraisons locales,

- l'AM, qui ouvre et gère les associations avec les MTA adjacents dans le réseau X400,

- le RTS, qui manipule les outils offerts par la couche session de OSI, afin d'établir et gouverner les

connexions servant de base aux associations et qui, par ce moyen, permet à un message répondant au protocole P1 d'atteindre le MTA voisin sur la route vers sa destination éloignée.

Cette découpe, en elle-même, guide, d'une manière toute naturelle, la recherche d'actions pour lesquelles le parallélisme prend un sens : puisque leurs fonctions sont distinctes et indépendantes, les trois blocs MD, AM et RTS semblent pouvoir travailler simultanément, sans se gêner. Ils peuvent dès lors être considérés comme des machines abstraites de trois types.

10.1. PARALLELISME AU NIVEAU DU MD

Suivant le modèle MHS/X400 présenté précédemment, un MTA peut desservir plusieurs UA. Chacun d'eux devrait être en mesure d'interagir avec le MTA en toute liberté, comme s'il était seul à converser avec lui. Dès lors, afin d'éviter une attente due aux services demandés au MTA par les autres UA, une solution se dégage : chaque UA dialoguant avec son MTA se voit aidé par un MD qui lui est dédié durant toute l'existence de l'interaction. Les blocs MD peuvent, sans se nuire, tourner en parallèle.

Cependant, s'il y a plusieurs copies du bloc MD, il n'y a qu'une seule entité MTA. C'est pourquoi le point de contact entre la sous-couche MTA et la sous-couche UA doit être unique. Les unités de dialogue entre le MTA et tous ses UA (i.e. les primitives de type SUBMIT, PROBE, REGISTER,...) doivent, de ce fait, être triées et envoyées vers l'interlocuteur adéquat : le "bon" MD ou le "bon" UA, suivant que l'un ou l'autre des deux prend la parole. Si

bien qu'un nouveau processus apparaît pour résoudre ces problèmes : le Gestionnaire d'Interaction UA-MTA (GI-UA-MTA).

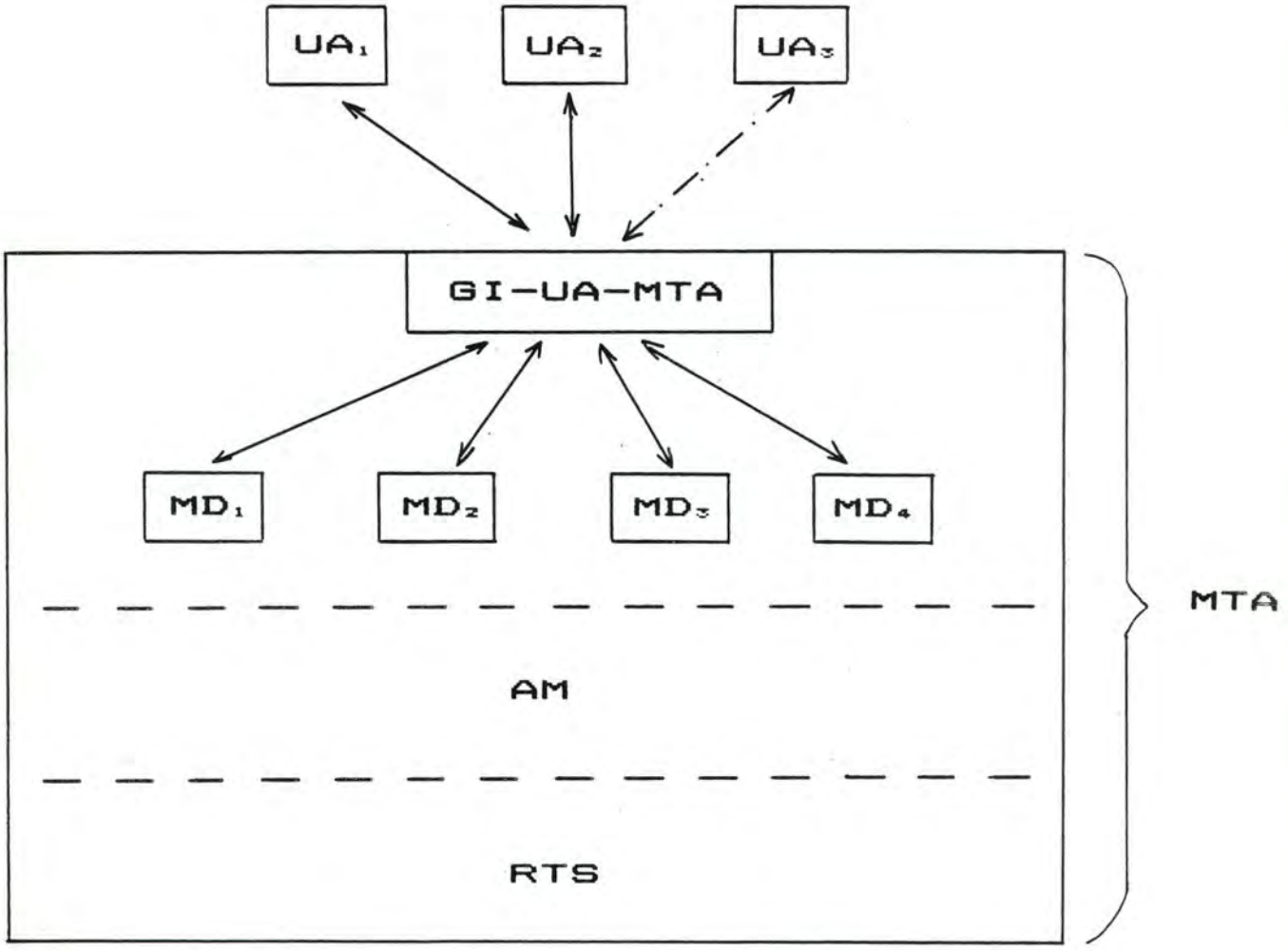


FIGURE 10.1 : PARALLELISME AU NIVEAU DU MD.

Comme le montre la figure 10.1, le MTA abrite plusieurs MD, mais pas forcément autant que d'UA abonnés aux services du MTA. En fait, idéalement, à un moment précis, il y a autant de MD que d'UA qui sont effectivement en relation avec le MTA. A ceux-ci s'ajoutent des MD servant à relayer des messages arrivant d'un MTA voisin, soit des MD travaillant à cet instant indépendamment de tout UA.

Toutefois, la réalité peut s'éloigner de cet idéal puisque finalement, il n'existe qu'un nombre limité de machines MD - nombre déterminé par les caractéristiques physiques du matériel et logiciel sur lesquels le MTA est implémenté (capacité de stockage, nombre de fichiers pouvant être ouverts simultanément,...).

C'est pourquoi la pénurie de machines abstraites MD disponibles doit être envisagée et traitée, aussi bien lors de la demande de connexion d'un UA (procédure de LOGON) que lors de la demande d'un MD afin de router un message venant de l'extérieur par l'intermédiaire des couches OSI inférieures.

10.2. PARALLELISME AU NIVEAU DE L'AM

Exactement suivant le même schéma, un MTA possède, dans le réseau MHS/X400, plusieurs MTA adjacents avec lesquels il peut communiquer dans le but d'envoyer et recevoir des messages,- exprimés selon le protocole P1 de manière à ce qu'ils soient relayés depuis l'UA origine jusqu'à l'UA destinataire.

L'interaction entre un MTA et un voisin (appelée association) étant totalement indépendante de celles établies entre le MTA et ses autres voisins, il peut exister plusieurs associations simultanées, pourvu que les MTA voisins visés soient distincts. Ceci suppose, comme dans le cas du MD, plusieurs copies de la machine AM, chacune chargée de la gestion d'une association.

Le fait de se restreindre à l'existence d'une association au plus entre deux MTA fixés à un instant donné est un choix personnel. Il prend, par rapport à la norme, une position qui semble naturelle. En effet, permettre

plusieurs associations entre deux mêmes MTA diminuerait sans doute l'attente devant l'AM. Mais puisqu'à ce niveau également, il n'existe qu'un nombre limité de machines, cela entraverait - en période de trafic dense - l'établissement d'une nouvelle association vers un MTA non encore atteint. C'est pourquoi la limitation personnelle imposée paraît raisonnable : un éventuel second message vers un MTA déjà en communication doit suivre le message courant, sur la même association.

Les problèmes d'aiguillage soulevés dans le cas de MD multiples apparaissent de même lorsque les AM se dédoublent. Dès lors, d'une façon toute semblable, un nouveau gestionnaire est introduit : le Gestionnaire d'Associations (GA), ainsi que le représente la figure 10.2.

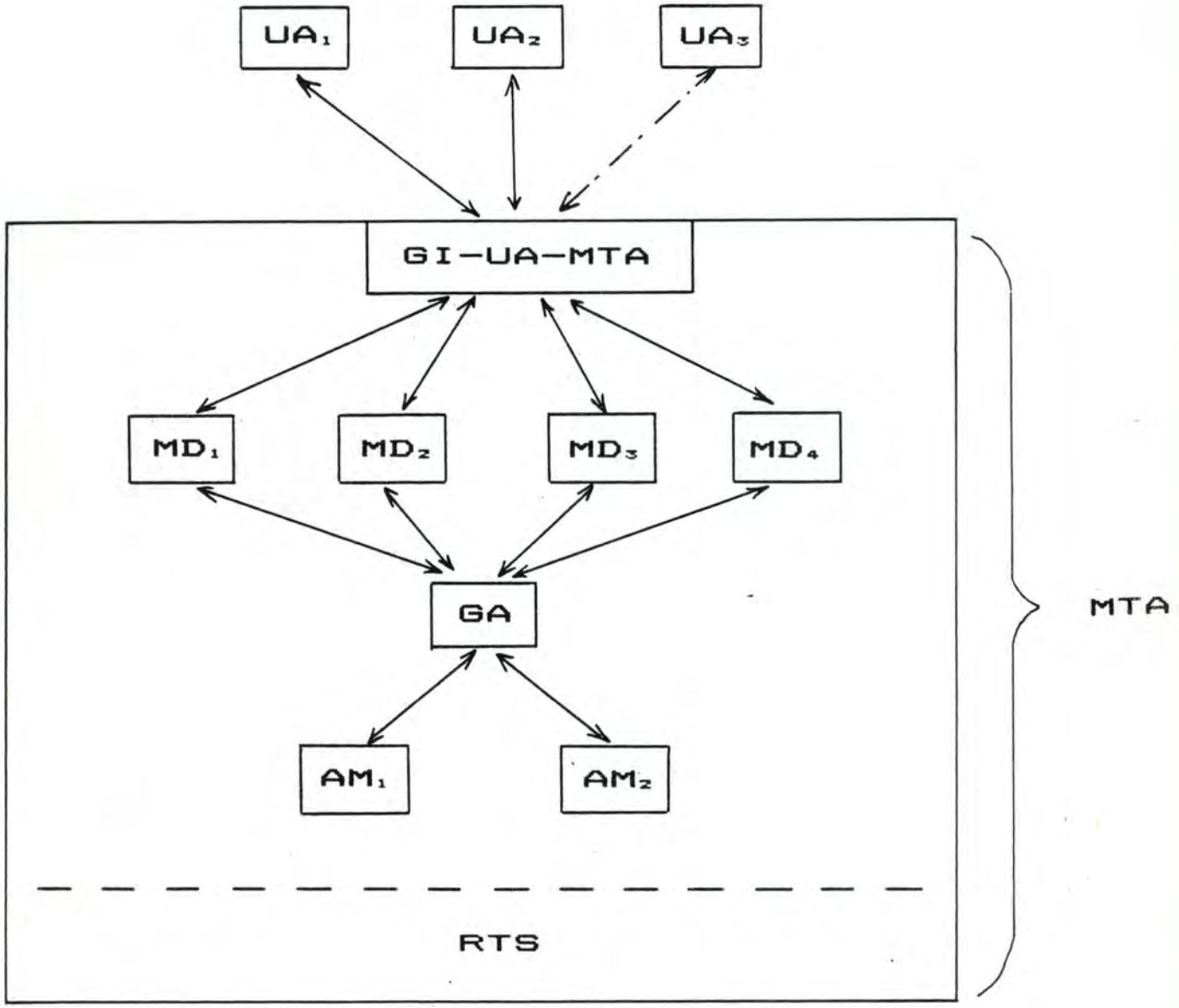


FIGURE 10.2 : PARALLELISME AU NIVEAU MD ET AM.

Il est à remarquer qu'il y a autant d'AM que d'associations en vigueur à un instant donné (moyennant le nombre maximal de machines AM disponibles) et que le nombre d'AM n'est pas une conséquence directe du nombre de MD. Ainsi, deux MD, communiquant avec leur UA respectif, peuvent, après routage, déterminer que les deux messages à relayer doivent prendre la même direction, emprunter la même association, et de ce fait, être traités par le même AM. De manière équivalente, deux messages en provenance de l'extérieur,

arrivant via deux associations différentes, peuvent se révéler être à destination d'un même UA local, si bien que les deux messages atteindront cet UA par l'intermédiaire du seul MD qui en est responsable. De plus, il ne faut pas oublier les messages émis par un UA à destination locale; ils sont traités par les MD associés respectivement aux UA origine et destination sans jamais atteindre un AM puisque celui-ci n'entre en jeu que pour permettre la sortie d'un message vers l'extérieur, pour relais vers un MTA voisin.

10.3. PARALLELISME AU NIVEAU DU RTS

Comme une association repose sur une connexion session établie par le RTS et comme plusieurs AM travaillent en parallèle, logiquement, il en va de même pour le bloc RTS : plusieurs RTS indépendants s'activent, chacun parlant au RTS d'un MTA adjacent, chacun lié à l'AM communiquant avec l'AM de ce même MTA adjacent. Face à eux, un gestionnaire répartit entre les divers RTS les messages et ordres en provenance des AM et des couches OSI inférieures : le Gestionnaire de Connexions (GC) (cfr figure 10.3).

Une fois de plus, le nombre de machines disponibles est limité.

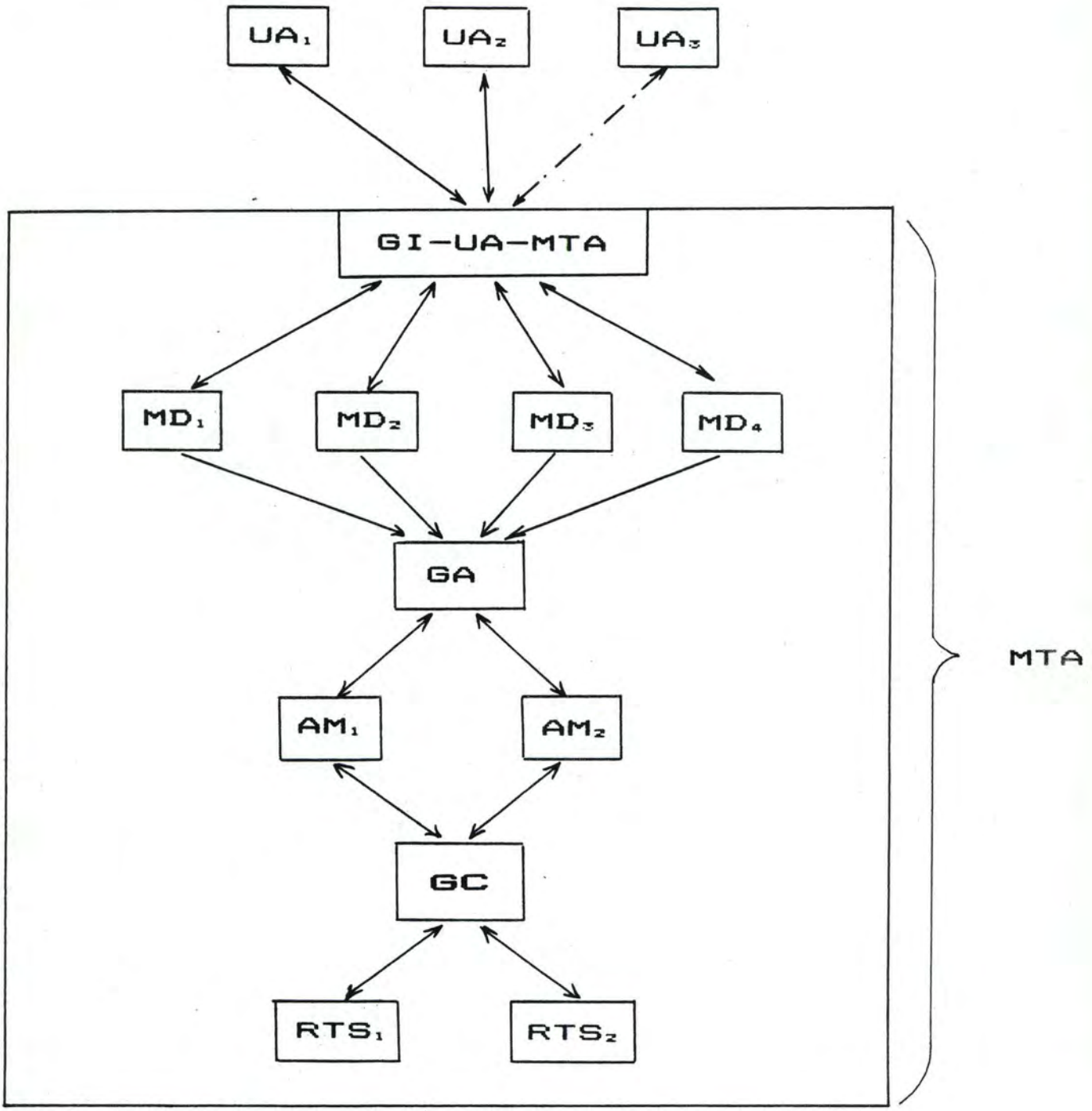


FIGURE 10.3 : PARALLELISME AU NIVEAU MD, AM ET RTS.

10.4. PREMIER MODELE COMPLET

Pour que le MTA soit à même de remplir son rôle de relayeur de messages, il doit être capable de dialoguer avec d'autres MTA. Ceci, on l'a vu, nécessite l'utilisation d'associations et de connexions. Ces dernières ont pour base les services offerts par les couches OSI situées sous la couche application.

Tout comme pour la communication vers le haut (i.e. avec les UA), l'interaction vers le bas (i.e. avec OSI) doit passer par un couloir unique et commun à tous les RTS. La raison en est que le MTA se veut, pour ses interlocuteurs, une boîte noire, avec une seule entrée standardisée, et qui ne laisse rien filtrer - au niveau des accès - des techniques de parallélisme qu'elle adopte. C'est pourquoi, la figure 10.4 montre l'ajoût d'un dernier gestionnaire, le Gestionnaire OSI (GO). Il se charge des échanges entre tous les RTS et la couche session du modèle OSI. Au même titre que les autres gestionnaires, il sert d'aiguilleur de messages.

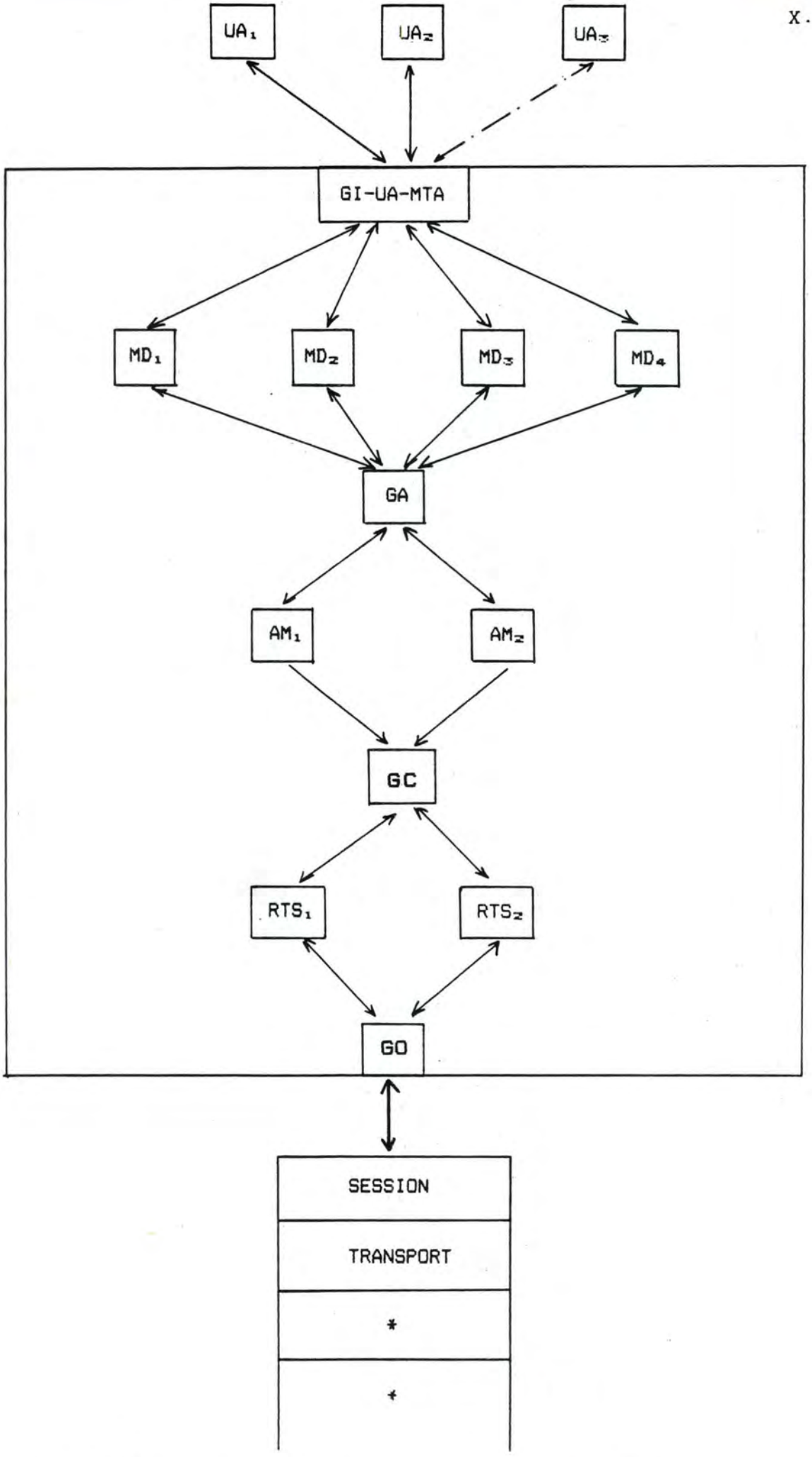


FIGURE 10.4. : DESCRIPTION COMPLETE DU MTA PARALLELE

10.5. EVOLUTION DU MODELE PROPOSE

10.5.1. LE PROBLEME DE L'AM

A la lecture des recommandations X400 s'installe un malaise certain quant à l'importance et la place exacte à accorder à l'AM.

D'une part, la norme propose une découpe en niveaux rappelée à la figure 10.5. L'AM et le MD y partagent un même niveau, qui domine celui du RTS. Les interactions standardisées - sous forme d'activation de primitives - concernent uniquement les dialogues entre UA et MD ainsi qu'entre AM et RTS.

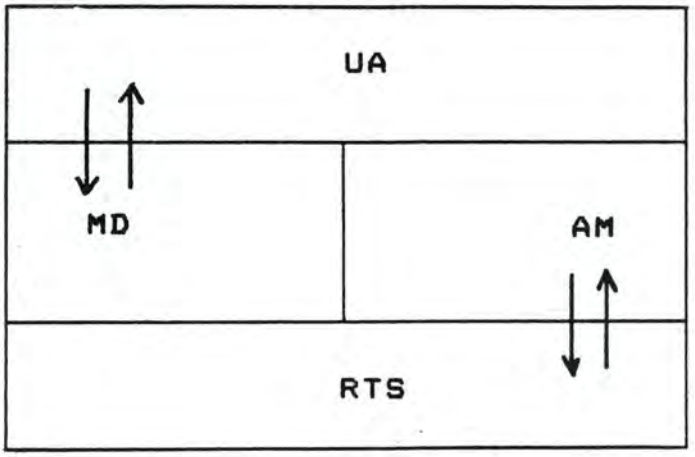


FIGURE 10.5 : RAPPEL DE LA DECOUPE DE LA COUCHE APPLICATION.

La norme reste fort discrète lorsqu'il s'agit de décrire l'interface entre MD et AM. Pourtant, le message doit, c'est une absolue nécessité, passer du MD au RTS. La seule voie possible semble être un transit par l'AM. Cependant, la réalisation n'en est nulle part prescrite. Quoiqu'il en soit, cette découpe a pour conséquence de donner à penser que MD et AM sont proches l'un de l'autre - puisqu'ils appartiennent au même niveau.

D'autre part, il apparaît aussi qu'un lien étroit unit une association avec la connexion session sur laquelle elle s'appuie, et, par là-même, un AM et un RTS.

Le fossé entre les deux tendances découle sans aucun doute du flou qui règne dans les recommandations dès que le chapitre de l'AM est abordé : il est là, avec un rôle qui peut paraître clairement décrit, mais jamais il n'est dit comment il entre en jeu. Ceci, on l'a dit, amène une sensation de malaise dont il est très difficile de se dégager et qui fut source de nombreux essais et revirements lors de la recherche en vue d'une simplification du modèle du MTA parallèle, présenté précédemment (cfr. figure 10.4). L'AM s'est vu tantôt rattaché au MD, tantôt totalement indépendant, tantôt quasi confondu avec le RTS.

Finalement, c'est le lien entre AM et RTS qui a semblé devoir être privilégié. En effet, l'établissement d'une association impose l'ouverture d'une connexion, si bien que la création d'un AM n'a de sens que si un RTS le seconde.

Par ailleurs, il peut arriver qu'une connexion session soit interrompue, pour motif grave, sans que l'association qu'elle dessert ne soit consciente des problèmes survenus plus bas. Plus tard, une nouvelle connexion, au sein de l'association qui continue d'exister, peut être ouverte, suite à une initiative interne au bloc RTS. Il s'agit là de la notion de recouvrement de connexion ou RECOVER.

Une première possibilité, dans ce cadre, serait de détruire un RTS dès que la connexion qu'il gère est coupée. Dès lors, un AM existerait sans qu'aucun RTS ne travaille pour lui et sans qu'il ne soit conscient de quoi que ce soit.

Cependant, cette solution risque de poser de graves problèmes en cas de RECOVER car celui-ci demande la création d'un nouvel RTS et son raccord à l'AM gérant de

l'association pour le compte de laquelle le RECOVER est activé. Ce raccord n'est pas trivial, vu le nombre d'AM existant et le nombre d'informations que les gestionnaires GO et GA doivent manipuler - complexité à laquelle se joint le problème de l'identification de la connexion à recouvrir: celle-ci s'effectue via un identificateur de connexion (donné lors de l'ouverture et rappelé lors du RECOVER) que le RTS seul conserve et qui, dès lors, disparaît avec lui, anéantissant tout espoir de reprise future.

C'est pourquoi une alternative intéressante est proposée : garder un RTS et un AM toujours unis jusqu'à la clôture d'une association, malgré les éventuelles coupures de connexion, pendant lesquelles le RTS existe mais est inactif, jusqu'à initialiser ou recevoir une demande de RECOVER.

En conséquence, les aiguillages des unités d'échange entre les divers AM et RTS sont identiques et les gestionnaires GO et GA se voient remplacés par un gestionnaire unique : le Gestionnaire d'Interaction MTA-MTA (GI-MTA-MTA). Il prend en charge toutes les communications entre des MTA voisins.

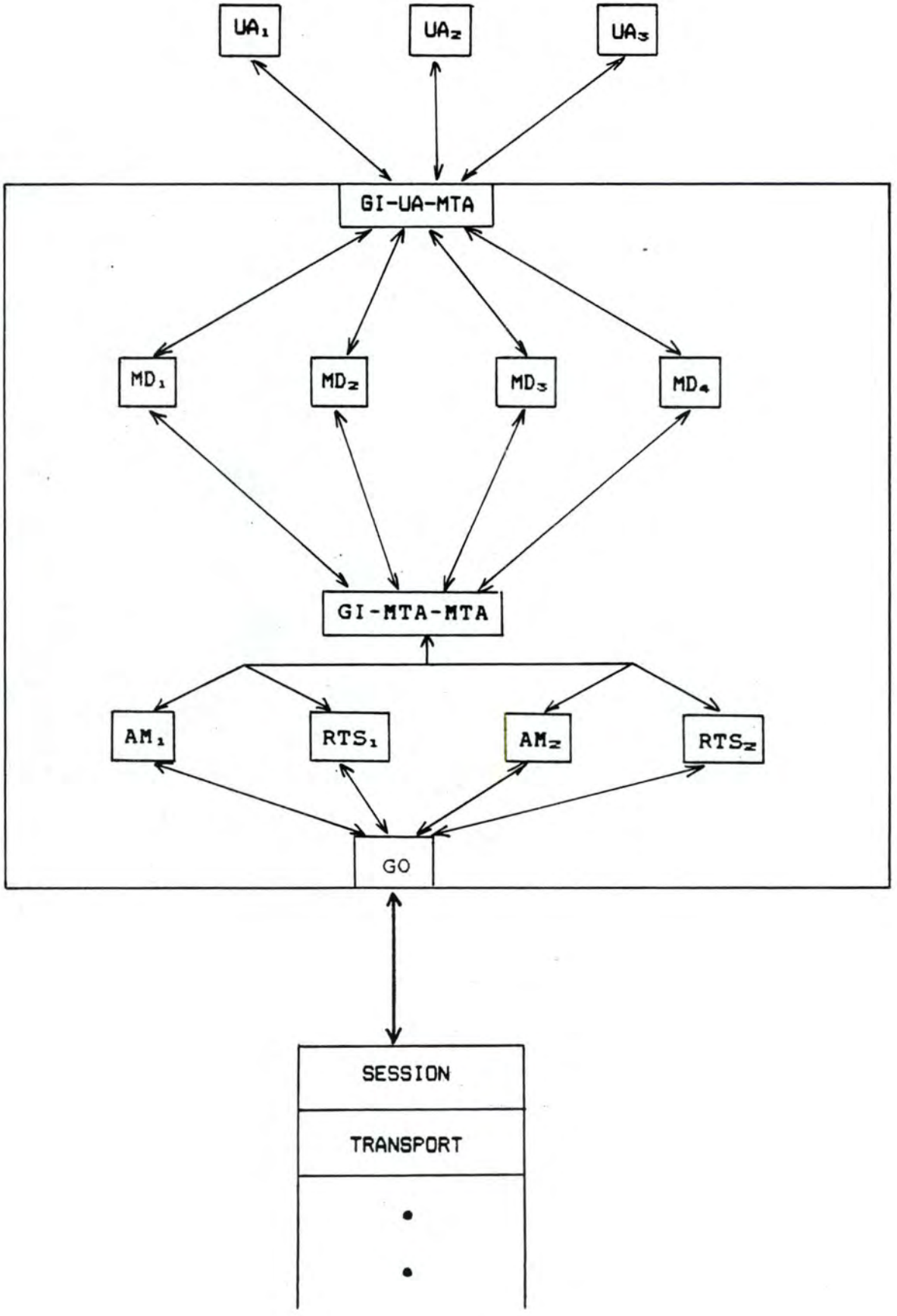


FIGURE 10.6 : REGROUPEMENT DE L'AM ET DU RTS.

La figure 10.6 présente une vision assez complète des divers constituants d'un MTA qui a été élargi par la notion de parallélisme. Ces divers constituants, pour remplir leurs tâches respectives, doivent s'échanger de nombreuses informations : le message à traiter - qui, depuis sa soumission par un UA, est véhiculé de l'un à l'autre, les demandes de service sous forme de primitives standardisées, des signaux qui ont une signification exclusivement locale,...

10.5.2. LES COMMUNICATIONS INTER-PROCESSUS

Puisque l'objectif de ce mémoire est la conception d'une architecture pour un MTA dans un cadre bien particulier, celui de UNIX version 7, les voies de communication entre les différentes parties du MTA doivent être réalisées au moyen des outils qu'offre UNIX, à savoir les pipes. Pour une présentation succincte des outils UNIX, le lecteur est renvoyé à l'annexe 3.

Les pipes imposent de fortes restrictions, puisqu'un pipe ne peut être établi qu'entre un processus fils et son père, et via le père, entre deux processus frères - i.e. fils d'un même père.

Or, il va sans dire que les gestionnaires GI-UA-MTA et GI-MTA-MTA doivent pouvoir dialoguer. En effet, un MD, physiquement, ne peut parler qu'au GI-UA-MTA. Lorsqu'il demande à avoir accès à une paire AM - RTS afin de relayer un message vers un MTA voisin, il doit le faire par l'intermédiare du GI-UA-MTA. Et seul le GI-MTA-MTA est en mesure d'allouer des machines AM et RTS disponibles, afin de répondre à la requête. Il en va de même dans le sens inverse lorsqu'un AM recevant, via son RTS, un message venant d'un site éloigné doit obtenir les services d'un MD afin d'exécuter l'algorithme de routage.

Si bien qu'il est indispensable que les GI-UA-MTA et GI-MTA-MTA soient capables de s'échanger des messages, et cela grâce à des pipes. Dès lors, il faut les rendre tous deux fils d'un même père.

La communication du GI-UA-MTA avec les UA et du GO avec la couche session pose un problème du même ordre. En effet, elle implique des échanges entre processus qui ne peuvent être père et fils puisque les UA et la couche session représentent l'extérieur de la boîte MTA, et sont dès lors totalement indépendants des processus composant le MTA. A ce niveau, le problème est nettement plus complexe que dans le cas des gestionnaires, car quelle que soit l'architecture adoptée, jamais un lien de parenté ne pourra unir UA et session avec des processus internes au MTA.

Finalement, une nouvelle organisation des divers constituants du MTA, présentée à la figure 10.7, permet d'apporter une solution cohérente aux difficultés soulevées par la communication inter-processus. Le père des GI-UA-MTA et GI-MTA-MTA est un gestionnaire, nommé Gestionnaire Central (GC). Il permet les échanges entre les deux autres et matérialise un point d'entrée unique dans le MTA, que ce soit en provenance des UA ou de OSI.

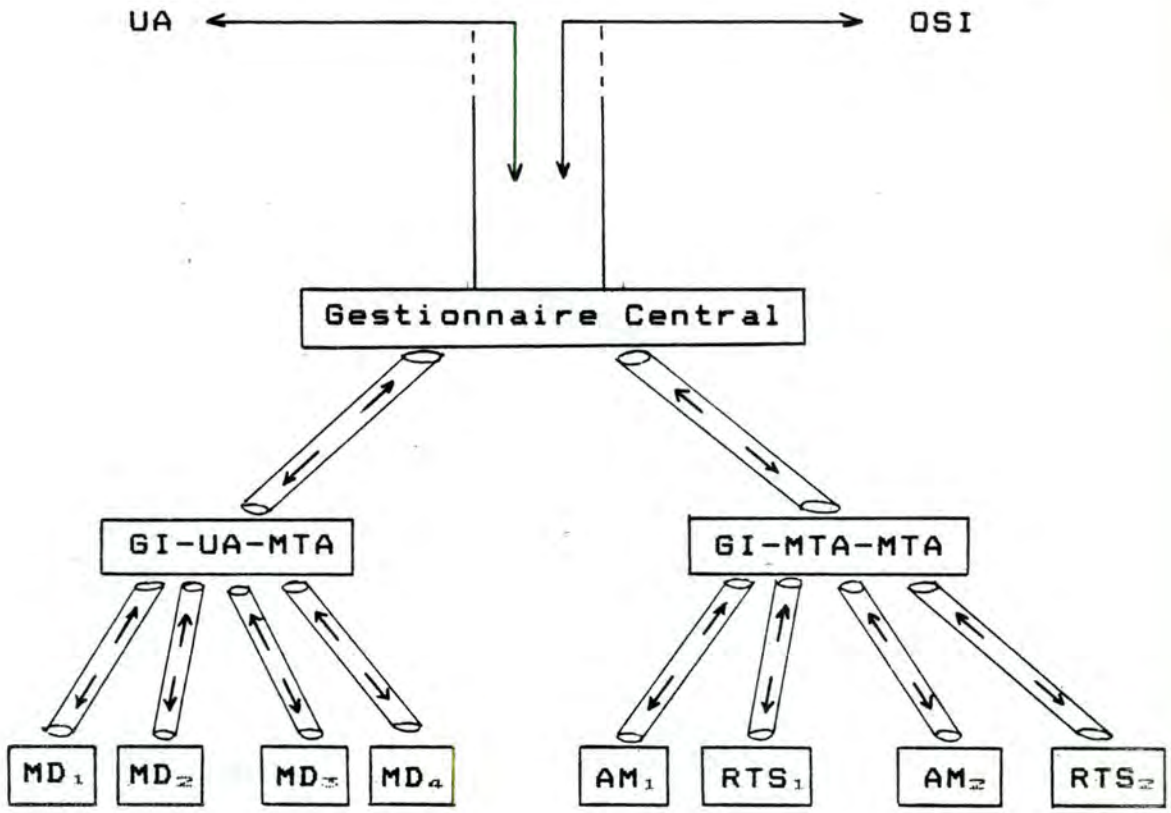


FIGURE 10.7 : ARCHITECTURE DEFINITIVE DU MTA PARALLELE.

A l'intérieur du MTA, les échanges entre deux frères passent à présent par leur père.

La communication inter-couches, soit entre les UA et le GC, soit entre la session et le GC, n'est pas, on l'a vu, aisée à réaliser.

Dans un cadre assez semblable à celui-ci, Béatrice Scoyer [SCO 86] a apporté une réponse satisfaisante lors de la conception et l'implémentation d'une maquette des trois premières couches OSI, sur la même machine que celle qui devrait servir de base au MTA parallèle. Les échanges inter-couches, quel que soit leur niveau dans OSI, présentent de grandes ressemblances. Il s'agit toujours d'activation de primitives déclenchant des traitements dans les différentes couches, de manière totalement asynchrone et indépendante. Et toujours s'ajoute le problème de la

filiation indispensable, sous UNIX, pour établir un conduit de communication.

En étudiant le mécanisme général des entrées/sorties dans un système UNIX [UNI 84], il apparaît qu'il s'appuie sur les appels systèmes standards d'ouverture et fermeture de fichiers, lecture et écriture d'un nombre fixé de caractères provenant ou à destination d'un fichier,... Le descripteur de fichier passé en argument indique au système d'exploitation qu'il s'agit d'une demande concernant un périphérique. Ceci a pour effet de faire basculer le système en mode "kernel", i.e. le mode privilégié de UNIX, dans lequel toutes les instructions de base sont accessibles - contrairement au mode utilisateur où certaines lui sont interdites. Le système lance alors la procédure appropriée pour traiter l'appel système. En l'occurrence, il s'agit du "driver" gérant le périphérique visé.

La relation inter-coucle suppose un dialogue entre un processus d'une couche et un processus d'une autre. Il faut un support à cette communication et bien sûr, des entrées/sorties pour réaliser l'accès. Dès lors, pour créer la relation inter-couches il semble avantageux de pouvoir basculer en mode kernel et user de toutes les instructions de base du système. Dans ce but, il est nécessaire d'intégrer le mécanisme de communication au sein même des modules du système d'exploitation. C'est par l'intermédiaire d'un appel système particulier qu'est accordé l'accès à de nouveaux modules, écrits dans le but de réaliser ces entrées/sorties très spéciales.

Puisque - et c'est une des caractéristiques des plus remarquables de UNIX - la plupart des objets de communication sont vus comme des fichiers (que ce soient de véritables fichiers utilisateurs, les répertoires, les pipes, les périphériques,...), l'entrée/sortie nécessaire pour relation inter-couches peut être, en fait, une lecture/écriture habituelle sur un fichier dont le descripteur fait référence à un périphérique particulier. Celui-ci, grâce à un nouveau "driver", dédicacé à la tâche

étudiée, entièrement conçu et programmé à cet effet, assure la gestion d'un espace disque exclusivement réservé à la communication inter-couches. En pratique, cet espace disque peut être, dans un environnement multi-utilisateurs, subdivisé en plusieurs parties distinctes, chacune correspondant à un utilisateur différent.

Le procédé décrit ci-avant, déjà expérimenté précédemment au niveau des couches OSI inférieures, correspond aux besoins du MTA. Il pourrait, de ce fait, servir de guide lors d'une implémentation réelle des échanges entre les UA et la couche session avec le MTA.

En conclusion, il est important de rappeler que l'architecture illustrée auparavant par la figure 10.7 atteint enfin le stade final de sa longue évolution. Elle peut être qualifiée d'"architecture parallèle" puisque, dès à présent, elle servira de toile de fond aux diagrammes d'états et algorithmes proposés pour chacun des blocs formant le MTA "parallèle".

10.6 DECOUPE EN NIVEAUX DU MTA INTEGRANT LE PARALLELISME

A présent que l'architecture type d'un MTA parallèle a été dégagée, il reste à faire évoluer la découpe en niveaux proposée - dans le cadre d'un MTA de base - au chapitre VIII.

Comme déjà annoncé auparavant, peu de changements, finalement, interviennent lors de l'introduction du parallélisme. Il faut cependant souligner l'addition des trois gestionnaires (GI-UA-MTA, GI-MTA-MTA et GC) qui sont placés au niveau 6, aux côtés des MD, AM et RTS puisqu'ils permettent eux-aussi de réaliser des fonctions

fondamentales pour un MTA parallèle. C'est la relation Importe/Exporte qui les unit aux autres modules de ce niveau, ainsi que l'illustre la figure 10.8.

Il ne faut pas l'oublier, la mise en oeuvre d'un MTA impose des communications inter-processus. Pour régler les problèmes qui en découle, le niveau 1, abritant les modules du système d'exploitation UNIX, contient plus précisément des éléments responsables de la gestion des processus et celle des pipes.

Une fois ajoutés les nouveaux modules cités ci-dessus, il ne reste plus qu'à déterminer quels sont les liens qui les unissent aux composants déjà présents dans l'architecture logique de base. C'est le but des figures 10.9 et 10.10 en ce qui concerne la relation UTILISE. La figure 10.11 apporte un complément en décrivant les interactions du type Importe/Exporte.

NIVEAU 7

NIV 7 / NIV 6	UA
MD	*
AM	*
RTS	*
GI-UA-MTA	*
GI-MTA-MTA	*
GC	*

FIGURE 10.7 : LA RELATION UTILISE AU NIVEAU 7

NIVEAU 6

NIV 6 / NIV 5	MD	AM	RTS	GI-UA-MTA	GI-MTA-MTA	GC
algo routage	*					
gestion. file d'attente	*	*	*	*	*	*
consultations tables	*	*	*	*	*	*
modifications tables	*	*	*	*	*	*
conversion	*					
copies UMPDU	*					
destruction UMPDU	*		*			
NIV 4						
analyse événement reçu	*	*	*	*	*	*
vérification paramètres	*	*	*	*	*	
création d'une primitive	*	*	*	*	*	
NIV 3						
vérif. mes. conforme P1	*					
analyse enveloppe	*					
modification enveloppe	*					
création UMPDU conformes P1	*					
création SMPDU conformes P1	*					
NIV1						
création pipes	*	*	*	*	*	*
gestion pipes	*	*	*	*	*	*
création processus	*	*	*	*	*	*
gestion processus	*	*	*	*	*	*

FIGURE 10.10 : LA RELATION UTILISE AU NIVEAU 6

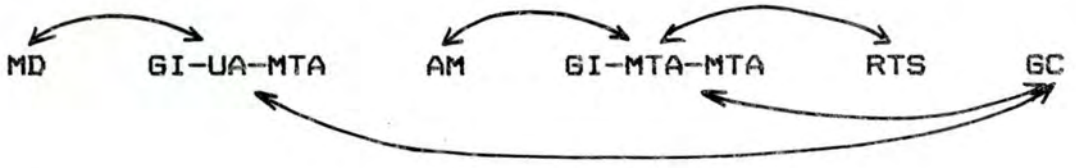


FIGURE 10.11 : INTERACTIONS IMPORT/EXPORTE DANS L'ARCHITECTURE PARALLELE.

10.7 NOTATIONS ADOPTEES

Dans la suite de ce rapport, les pipes vont être manipulés en de nombreuses occasions. Dès lors, il est intéressant de pouvoir les distinguer et les reconnaître facilement. C'est pourquoi, et la figure 10.12 l'illustre, il convient de leur donner un nom qui les identifie de manière si possible représentative.

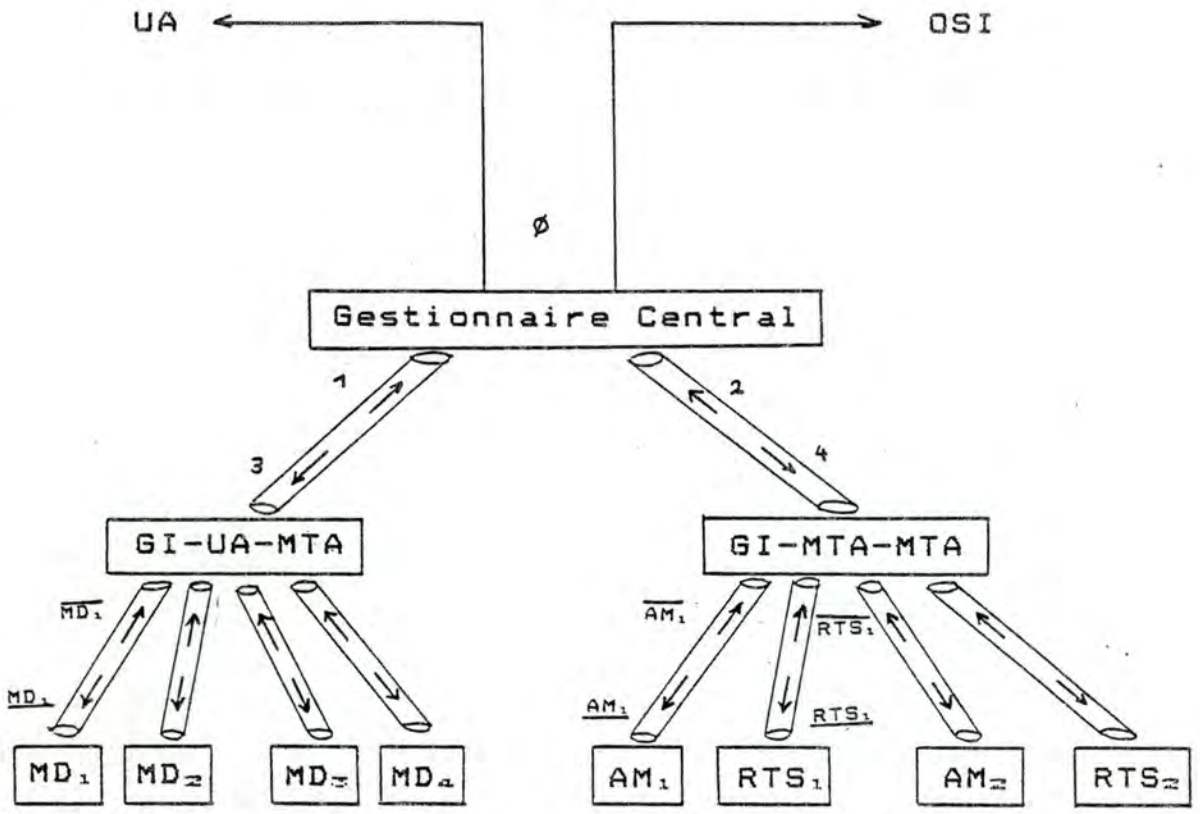


FIGURE 10.12 : NOTATIONS CONCERNANT LES PIPES.

D'ailleurs cette liaison entre nom et fonction prend toute son importance lorsqu'il s'agit des pipes unissant le GI-UA-MTA et les différents MD ainsi que le GI-MTA-MTA et ses AM et RTS. En effet, en raison de la multiplicité des machines MD, AM et RTS, il est indispensable de pouvoir lier le nom du pipe à la machine. Si bien que le nom du pipe contient le nom d'une des machines dédoublées (ex : MD) indicé par son numéro propre (ex : MDi). Un trait supérieur ou inférieur indique le sens que suivent les informations dans le pipe (ex : $\overline{\text{MDi}}$ si les données "montent" du ième MD vers le gestionnaire; $\underline{\text{MDi}}$ si elles "descendent" du gestionnaire vers le ième MD).

Ces notations seront utilisées couramment dans le chapitre suivant qui décrit de manière détaillée les modules fonctionnels de l'architecture parallèle adoptée.

FACULTES
UNIVERSITAIRES
N.D. DE LA PAIX
NAMUR

INSTITUT D'INFORMATIQUE
RUE GRANDGAGNAGE, 21, B - 5000 NAMUR (BELGIUM)

Spécifications d'un agent de transfert de messages dans le cadre d'une messagerie électronique de type X400 : définition, conception et intégration de traitements parallèles en vue d'une implémentation sous Unix version 7.

Mémoire présenté par

Marie-Elise ANGELOT
Thierry DELROISSE
Christine FRANSSENS

en vue de l'obtention

du titre de

Licencié et Maître en Informatique

Promoteur : Ph. van BASTELAER

Année académique 1986-1987

TOME II

Chapitre XI.

CONCEPTION DES MODULES FONCTIONNELS DE L'ARCHITECTURE PARALLELE.

XI.1. PRELIMINAIRES

Une nouvelle architecture a été dégagée au chapitre précédent. Elle peut dès à présent être qualifiée d'"architecture parallèle" puisqu'elle a été conçue en vue d'intégrer la notion de traitements parallèles à l'architecture de base déduite de la découpe fonctionnelle du MTA.

Les modules situés au niveau 6 de cette architecture parallèle sont détaillés dans la suite de ce chapitre. Une description soignée de chacun est suivie du diagramme d'états qui la complète en exprimant les enchaînements. Les algorithmes en pseudo-langage sont disponibles en annexe. En fin de chapitre, on peut trouver un complément d'informations sous forme du récapitulatif des tables permettant le stockage des renseignements nécessaires à la gestion du parallélisme.

Par ailleurs, il est important de rappeler que les divers modules fonctionnels, pour mener à bien leurs tâches, doivent échanger des informations. Dans la suite de ce chapitre, l'unité de dialogue entre ces modules est appelée "message". Il ne faut pas, pour autant, confondre ce terme "message" avec celui qui désigne les blocs d'informations que se passent deux utilisateurs via le MHS. Pour éviter toute confusion, on adopte les deux désignations suivantes :

* "message" lorsqu'il s'agit d'un échange entre deux modules

* "MPDU" lorsqu'il s'agit d'un échange entre deux utilisateurs

Une fois ces conventions posées, les modules fonctionnels peuvent être successivement abordés.

XI.2. LE GESTIONNAIRE D'INTERACTION UA-MTA

11.2.1. INTRODUCTION

Le GI-UA-MTA [Gestionnaire d'Interaction UA - MTA] présenté à la figure 10.7 entre le GC et les MDi a pour fonction de gérer les interactions entre les UA et le MTA. Ce chapitre commence par rappeler pourquoi l'architecture adoptée a nécessité l'introduction du GI-UA-MTA. Ensuite, une construction progressive du GI-UA-MTA est élaborée, en vue d'atteindre une solution appropriée faisant face aux obstacles rencontrés et répondant aux hypothèses posées. Cette partie est clôturée par un rappel des connaissances indispensables au GI-UA-MTA pour son bon fonctionnement.

11.2.2. POURQUOI UN GI-UA-MTA ?

Créer le GI-UA-MTA est apparu très vite indispensable suite à notre choix de construire un MTA parallèle. Le GI-UA-MTA, sorte de "MD" centralisateur qui lui seul, a la possiblité de gérer les connexions entre UA et MD, joue le rôle de noeud de branchement. Sachant quel UA est connecté, et avec quel MD, il aiguille les messages qu'ils s'échangent comme le suggère la figure 11.2.1, d'où son nom de gestionnaire d'interaction UA-MTA. Qui plus est, à cette fonction, s'ajoute celle de poursuivre ou non le relais de messages provenant des couches inférieures du modèle OSI via le RTS et l'AM.

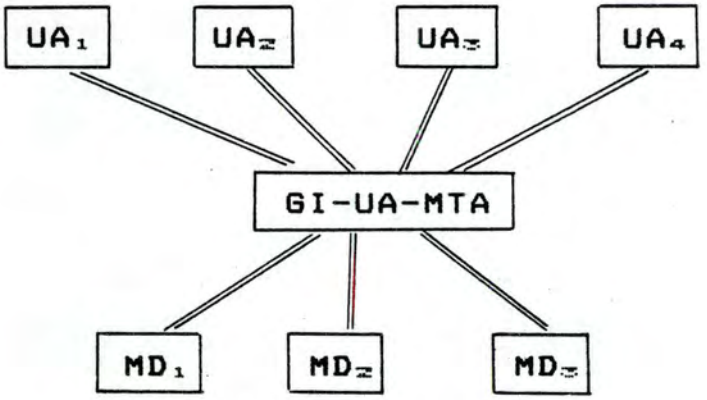


FIGURE 11.2.1 : INTRODUCTION DU GI-UA-MTA DANS L'ARCHITECTURE

Devant obligatoirement respecter les contraintes imposées par Unix version 7, cette ébauche de solution est complétée par l'introduction d'un GC (gestionnaire central) venant s'intercaler entre les UA et le GI-UA-MTA (cfr. figure 11.2.2). Ce nouveau gestionnaire dirige les communications avec les processus extérieurs au MTA et entre ses processus fils.

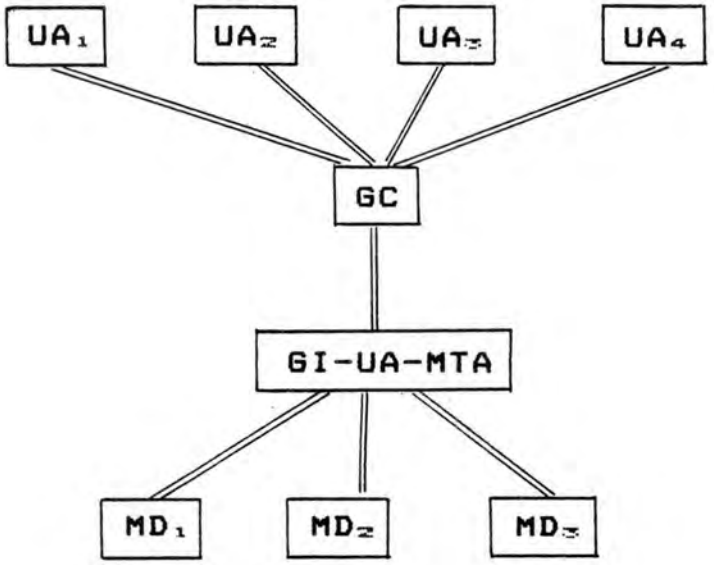


FIGURE 11.2.2 : INTRODUCTION DU GC DANS L'ARCHITECTURE

11.2.3. ELABORATION PROGRESSIVE DU GI-UA-MTA

La construction graduelle du GI-UA-MTA s'effectue en trois étapes. La première est limitée à l'interaction côté UA; la seconde, côté AM et pour terminer, la troisième complète les deux premières par les actions à prendre suite à des demandes spéciales dites "demandes de service".

Le problème étant complexe, une hypothèse simplificatrice est posée dès le départ afin de rendre l'approche du GI-UA-MTA plus compréhensible : lorsque le GI-UA-MTA reçoit un message, il peut connaître la provenance et la destination de ce message de même que le type du message (primitives venant de l'UA, relais, demande

de service,...). En fin de cette présentation, une liste exhaustive des messages soulève cette hypothèse.

PAS 1 : RAISONNEMENT LIMITE COTE INTERACTION UA/MD

Recevant un message d'un UA, le GI-UA-MTA a la responsabilité du routage de celui-ci vers un MD. Par conséquent, il doit absolument savoir

- si un MD est déjà attribué à cet UA
- si cet UA a établi une communication avec un MD (au moyen d'une primitive LOGON).

Devant contrôler si oui ou non l'UA a le droit de parler à un MD, il paraît raisonnable de lui attribuer la fonction de gestion de l'établissement d'interactions entre UA et MD ["LOGON"]. Cet établissement peut être invoqué soit par l'UA, soit par le MTA et donc via le GI-UA-MTA. Le GI-UA-MTA va mémoriser les renseignements nécessaires dans la table 3 dont voici une ligne

id-ua	ua OR-NAME	$\overline{\text{MDi}}$	$\underline{\text{MDi}}$	bit connex	verrou

Chaque UA attaché à un MD a une entrée dans cette table; id-UA est le numéro de l'UA. Celui-ci est attribué à chaque UA desservi par ce MTA. Ceci facilite la reconnaissance LOCALE des UA et évite ainsi la structure complexe des OR-NAME de X400.

La valeur de bit connex enregistre l'état connecté ou non de cet UA. Un 1 signale que l'UA est connecté, un 0 qu'il ne l'est pas. $\underline{\text{MDi}}$ et $\overline{\text{MDi}}$ sont les noms de pipes descendant et montant entre le GI-UA-MTA et le MD considéré. Ils représentent simplement les noms des chemins.

Possédant une telle table, le scénario du logon émanant d'un UA est le suivant :

1) le GI-UA-MTA sachant détecter que le message vient d'un UA identifié par id-UA, teste si id-UA est une entrée de la table 3.

SI OUI, aller en 2)

SINON, aller en 4)

2) id-UA est en relation avec un MD via $\underline{MDi}$ et $\overline{MDi}$.

SI bit connex = 1

ALORS

id-UA est connecté à ce MD et par conséquent du point de vue du gestionnaire, il a la permission de lui parler. Le gestionnaire fait donc transiter le message vers le MD par $\underline{MDi}$

SINON aller en 3)

3) un MD travaille pour l'UA mais ne lui est plus connecté. C'est le cas lorsque cet UA s'est déconnecté mais que le MD traite toujours pour lui des demandes différées. Le GI-UA-MTA bloque alors la voie d'accès entre l'UA et le MD et traite lui-même le message. Celui-ci ne peut être qu'une demande d'ouverture de transaction (LOGON). Le GI-UA-MTA prend en charge la vérification de la demande et suivant le résultat obtenu répond positivement ou négativement à l'UA. En cas de réussite, il met à jour le bit de connexion (bit connex = 1). Bit connex peut donc être interprété comme une porte d'accès.

SI bit connex = 1

ALORS

la porte est ouverte et le message passe directement

SINON

le message est stoppé et contrôlé au niveau du gestionnaire.

4) aucun MD n'est attribué à cet UA. A nouveau, le message est bloqué par le GI-UA-MTA qui l'analyse et y répond. Seul un LOGON est autorisé. En cas de réponse positive, un MD est créé et mis à disposition de l'UA avec bit connex = 1.

Cette première solution se justifie par le fait qu'un seul MD est autorisé par UA.

Pour la cohérence de l'implémentation, chaque MD ne peut comprendre (c-à-d pouvoir interpréter) la primitive LOGON et donc la traite comme cas d'erreur. Contrairement à ce que suggère X400, la primitive de LOGON est surveillée et traitée par le GI-UA-MTA.

Quelques tests sont à insérer dans cette suite d'opérations :

* test de sécurité, d'authentification

Le GI-UA-MTA doit vérifier les paramètres de la procédure d'ouverture d'interaction. Pour ce

faire, il consulte la table 4 renfermant toutes les caractéristiques des UA (mot de passe et contraintes) desservis par ce MTA.

id-ua	password	restrictions	register

* test de possibilité du système

Vouloir introduire le plus possible de parallélisme est une idée assez ambitieuse, cependant il ne faut pas perdre de vue les capacités limitées de la machine sur laquelle installer cet MTA. Prenant en considération cette restriction, un nombre "illimité" (i.e. aussi grand que nécessaire) de MD se voit remplacé par un nombre fixé de machines MD.

Par conséquent, si aucun MD n'est attribué à l'UA (phase 4), le scénario doit évoluer vers la solution suivante :

SI une machine MD est encore disponible

ALORS 4)

SINON

répondre négativement à la demande de LOGON pour cause "d'encombrement" ou de "surcharge".

Le table 5 va permettre au GI-UA-MTA de contrôler l'occupation des machines.

En conclusion, il faut retenir que trois rôles essentiels sont attribués au GI-UA-MTA :

traiter les demandes d'ouverture de connexion LOGON

gérer les machines MD et les attribuer

aiguiller les messages circulant entre MD et UA.

PAS 2 : RAISONNEMENT LIMITE COTE INTERACTION AM/MD

Il est possible d'oublier momentanément les messages provenant de l'UA et de se limiter au GI-UA-MTA servant d'intermédiaire entre MD et AM.

Comme précédemment expliqué, un des points de la norme reste obscur : par quel moyen MD et AM communiquent-ils ? Pour répondre à cette question, il est nécessaire d'inventer des messages entre un MD et un AM afin que chaque module comprenne la tâche demandée par l'autre. Par bonheur, cette imprécision de la norme n'est pas dramat'ique puisque cette communication est locale et ne fait par conséquent pas l'objet de protocoles.

Etant donné les fonctions du MD et de l'AM, voici le relevé des moments où une interaction s'impose :

1) MD --> AM :

le MD réalise que le destinataire n'est pas local. Après activation de son algorithme de routage, il demande à l'AM d'envoyer le MPDU vers le MTA voisin trouvé.

2) AM --> MD :

l'AM vient de recevoir un MPDU d'un MTA voisin et le passe au MD afin que celui-ci vérifie sa conformité à P1 et le rapproche de son destinataire par un relais ou une livraison.

3) AM --> MD :

bien que demandé par le MD, l'AM n'a pu transférer le MPDU vers un MTA voisin. Il doit alors prévenir le MD afin que celui-ci teste s'il y a lieu de générer une notification de non livraison.

Ces trois cas trouvés par un raisonnement dans le cadre du MTA non parallèle doivent être adaptés dans le contexte du MTA parallèle. L'architecture illustrée par la figure 11.2.3 impose, pour la commnunication dans le sens MD --> AM, que le GI-UA-MTA écrive simplement le message à destination du GI-MTA-MTA dans le pipe 3. (Un des rôles du GI-MTA-MTA est le choix du bon AM.)

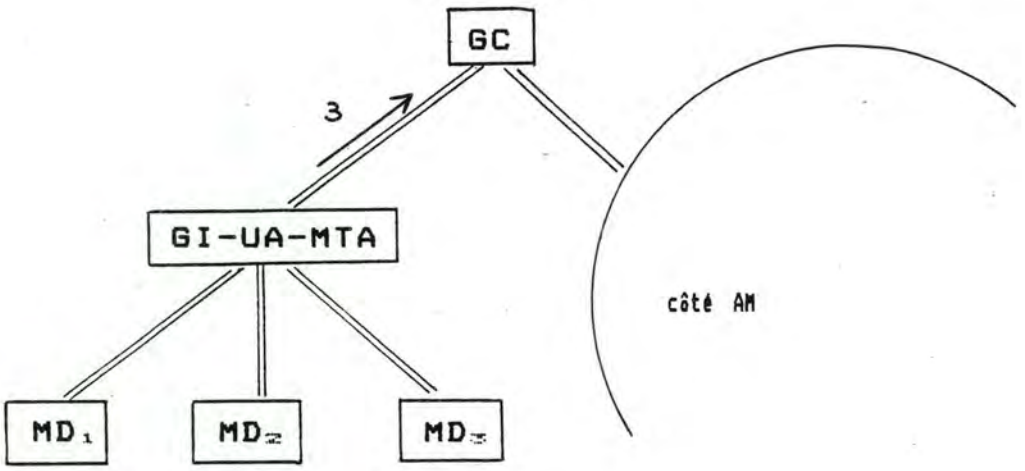


FIGURE 11.2.3 : POSITION DU GI-UA-MTA DANS L'ARCHITECTURE PARALLELE.

Tandis que pour la communication dans le sens inverse AM (et donc GI-MTA-MTA)-->MD, le GI-UA-MTA est responsable de la sélection du MD et suivant ce choix, de la distribution du message.

Gardant en mémoire le fait que le GI-UA-MTA est capable de savoir d'où provient le message, à qui il est

destiné et de quel type il est, voici exactement la politique à suivre (par le GI-UA-MTA) dans les trois interactions possibles soulevées :

1) MD --> AM :

la demande est simplement dirigée vers la voie de communication conduisant à l'AM (pipe 3)

2) AM --> MD (demande de relais) :

a)une nouvelle machine MD est attribuée pour chaque demande de relais.

Les arguments qui ont guidé ce choix sont les suivants:

* décharger le plus possible un MD alloué à un UA, celui-ci ne devant réaliser que des services de l'UA

* poursuivre l'idée d'introduire le maximum de parallélisme et donc mettre en route une machine MD par demande de relais.

* libérer le plus vite possible une machine pour pouvoir l'assigner à une autre tâche. Ceci est rendu possible par le fait qu'un relais ne risque pas de mobiliser trop longtemps une machine. Par contre un MD lié à un UA est attribué au moins jusqu'à ce que l'UA décide de terminer la connexion (LOGOFF).

b)si aucune machine n'est disponible alors le message ne peut continuer sa route pour cause d'engorgement ou surcharge. Suite à cela, il faut assurer qu'une notification de non livraison est générée si elle est demandée. (Ceci rejoint le cas 3.)

Un MD-relais est marqué comme suit dans la table 3

0	NULL	$\overline{MDi}$	MDi	0

3) AM --> MD :

(demande d'un examen du MPDU pour créer et envoyer une notification de non livraison si celle-ci a été exigée par l'émetteur.)

a) SI une machine est libre,

ALORS elle est activée pour effectuer cette demande

b) SINON plusieurs techniques sont envisageables :

- soit on choisit en premier lieu un MD non connecté (et ceci à nouveau dans le but de libérer les MD connectés aux UA)

- soit une variable comptabilise par machine le nombre de demandes d'examen de MPDU et les MD choisis d'abord sont ceux pour lesquels la valeur de la variable est minimale

- soit une machine est constamment réservée et destinée à ce type de demande

.....

Dès à présent, de gros problèmes de synchronisation entre processus sont soulevés et débouchent sur la notion de demande de service. En effet, dans le cas 3)b, le MD n'est pas créé explicitement pour réaliser la demande de création et envoi de notification de non livraison, si nécessaire (= DEMNOTNL). Au contraire, soit il est dédié à un UA, soit à un relais. L'hypothèse est posée que DEMNOTNL est une demande de service émanant du GI-UA-MTA et destinée à un MD. Dû à cela et au fait que le GI-UA-MTA et les MD travaillent en parallèle, lorsque le GI-UA-MTA envoie sa demande de service au MD, il est incapable de savoir dans quel état se trouve le MD concerné. Qui plus est, la notion d'attente ne doit pas être oubliée car une demande n'est pas traitée immédiatement. Ces deux exemples éclairent le problème.

Premier exemple :

Un MD destiné à satisfaire les demandes d'un UA peut vouloir se détruire si cet UA émet la primitive de LOGOFF. Le GI-UA-MTA ne peut déceler lui-même cet ordre de fin de connexion. Dès lors si, après l'ordre de LOGOFF, une DEMNOTNL a été confiée à ce MD, que va-t-il advenir de cette requête ?

Deuxième exemple :

De même, après avoir mené à bien le relais, le MD créé à cet effet peut désirer sa suppression afin de rendre la machine disponible. Que va-t-il se passer si ce MD demande à être détruit au moment même où le gestionnaire lui confie la DEMNOTNL ?

De ces exemples, certaines questions sont mises en évidence:

- quand faut-il détruire un MD (i.e. rendre la machine disponible) ?
- le MD peut-il toujours lire suffisamment vite la demande de service ? Ne risque-t-il pas de laisser tomber des demandes arrivées après un LOGOFF ou un relais ?
- inversément, le GI-UA-MTA est-il en mesure de comprendre assez tôt le nouvel état du MD que pour ne plus lui distribuer de demande ?

En résumé, le GI-UA-MTA doit assurer les trois tâches suivantes :

- l'aiguillage des messages du MD vers l'AM via le GC

- l'affectation d'une machine MD pour une demande de relais

- l'affectation d'une machine pour une demande de notification.

L'étape suivante a pour but de résoudre le problème soulevé dans ce pas 2 : la synchronisation entre MD et GI-UA-MTA.

PAS 3 : AJOUT DES DEMANDES DE SERVICE.

Essayons de préciser la notion de service introduite ci-dessus. Un service est une demande de travail donnée par ou via le GI-UA-MTA à un MD, travail autre que toute tâche répondant au critère suivant lequel le MD a été créé (dialogue avec UA ou relais). Cette notion de service étant intimement liée avec le problème de synchronisation inter-processus, les deux problèmes vont être résolus ensemble.

Une liste exhaustive de tous les services venant du GI-UA-MTA est placée en 11.10.

Considérant cet aspect service, le GI-UA-MTA peut être interprété comme un distributeur de services. Dès lors, lui seul sait quel MD est touché par ses demandes et donc lui seul peut juger si un MD a le droit ou non de se détruire.

Concrètement, deux faits sont à assurer :

- Le premier consiste à comptabiliser les services à effectuer par chaque MD et non encore rendus. Cette condition peut être établie matériellement par

l'emploi d'une nouvelle variable dans la table 3 nommée verrou, et par l'ajout d'un message standardisé véhiculé dans le sens MD --> AM signifiant "moi, MD, j'ai rendu le service que vous m'aviez demandé". Quant au GI-UA-MTA, il incrémente son verrou de 1 à chaque envoi de demande de service et le décrémente à chaque réponse de service rendu.

- La deuxième action à garantir réside dans le fait d'interdire la destruction du MD sans l'accord du GI-UA-MTA. Deux nouveaux messages sont ajoutés au répertoire du MD et GI-UA-MTA :

* la demande de destruction émanant du MD à l'intention du GI-UA-MTA "je n'ai plus rien à faire"

* l'ordre de destruction "destruction" émanant du GI-UA-MTA à l'intention du MD.

Tant que le MD n'a pas reçu cet ordre, il doit poursuivre son exécution et par conséquent vérifier si rien ne lui a été demandé.

Il faut garder en mémoire que le message "je n'ai plus rien à faire" peut être employé dans des circonstances différentes : soit après un LOGOFF, soit après un relais mais aussi après un service rendu si le MD n'en trouve pas d'autre.

En résumé, après avoir effectué le travail pour lequel il a été conçu, le MD envoie au GI-UA-MTA le message "je n'ai plus rien à faire" chaque fois qu'il ne trouve plus rien dans la liste des messages venant du GI-UA-MTA à l'exception de "destruction".

Joignant les deux actions (comptabiliser les demandes de service, demander au GI-UA-MTA la permission de se

détruire), le GI-UA-MTA ne permet la destruction d'un MD que si les deux conditions suivantes sont respectées :

-il reçoit un signal venant du MD affirmant "je n'ai plus rien à faire"

-le verrou correspondant à ce MD est à 0.

Ce problème réglé, un autre piège se dessine, celui-ci tournant à nouveau autour du fait que le GI-UA-MTA n'est pas conscient du contenu de certains messages qu'il fait transiter.

En effet, dû au temps nécessaire pour effectuer certaines actions et au travail simultané du MD et du GI-UA-MTA, le scénario suivant peut se dérouler. Un UA (id-UA) connecté à un MD est représenté dans la table 3 par la ligne

id-ua	ua or-name	$\overline{MDi}$	$\underline{MDi}$	bit connex	verrou

Cet UA décide de terminer sa connexion et émet la primitive de demande de LOGOFF. Si cet UA essaie d'envoyer une demande avant de recevoir la confirmation du LOGOFF, celle-ci risque de ne pas être bloquée par le GI-UA-MTA puisqu'il n'est pas encore au courant du LOGOFF. Cet abus ne peut être filtré par le GI-UA-MTA. Ce sera donc au MD de régler ce problème.

Avant de clôturer cette description du GI-UA-MTA, il est nécessaire d'expliquer la stratégie qu'applique le GI-UA-MTA face à un message de type "demande de MD connecté à un UA destinataire en vue de délivrer un message ou une notification". Brièvement, ceci se produit lorsqu'un MD (appelé MD demandeur) constate que l'UA destinataire est un UA local et que cet UA ne lui est pas connecté. Il produit alors un message spécial, analysé par le GI-UA-MTA, demandant un MD connecté à cet UA. Face à ce type de requête, voici la réaction du GI-UA-MTA :

1) si un MD est connecté à cet UA, alors il lui passe la demande et incrémente le verrou de ce MD de 1.

2) si un MD est lié mais non connecté à cet UA, alors il tente de connecter ce MD à l'UA. (Cas d'un LOGON initialisé par le MTA et donc le GI-UA-MTA)

S'il y réussit

ALORS

il envoie la demande de livraison à ce MD et incrémente son verrou de 1

SINON il va en 3)

3) le GI-UA-MTA compose la primitive de service : DEMNOTNL et la soumet au MD demandeur

4) SI aucun MD n'est lié à cet UA

ALORS

SI le MD demandeur n'est attaché à aucun UA

ALORS aller en 5)

SINON aller en 6)

5) le GI-UA-MTA essaie d'établir l'association entre ce MD et l'UA

SI le branchement réussit

ALORS il attribue le message à ce MD demandeur

SINON il retourne en 3)

6) SI une machine est libre et si le GI-UA-MTA parvient à la connecter à l'UA

ALORS

il l'active pour effectuer cette requête et

il augmente le verrou de ce MD de 1

SINON il retourne en 3)

En conclusion, pour résoudre le problème de la synchronisation entre processus et pour assurer que toutes les demandes de services (données par le GI-UA-MTA à un MD) soient traitées, le GI-UA-MTA doit comptabiliser par MD tous les services demandés et ne permettre la destruction du MD que lorsque celui-ci a demandé sa destruction et que tous les services ont été rendus.

11.2.4. MESSAGES TRANSITANT PAR LE GI-UA-MTA

Le GI-UA-MTA peut établir une typologie des messages en fonction

* de leur origine

* de la composition des messages.

D'une part, suivant la voie de communication d'où le message est lu (figure 11.2.4),

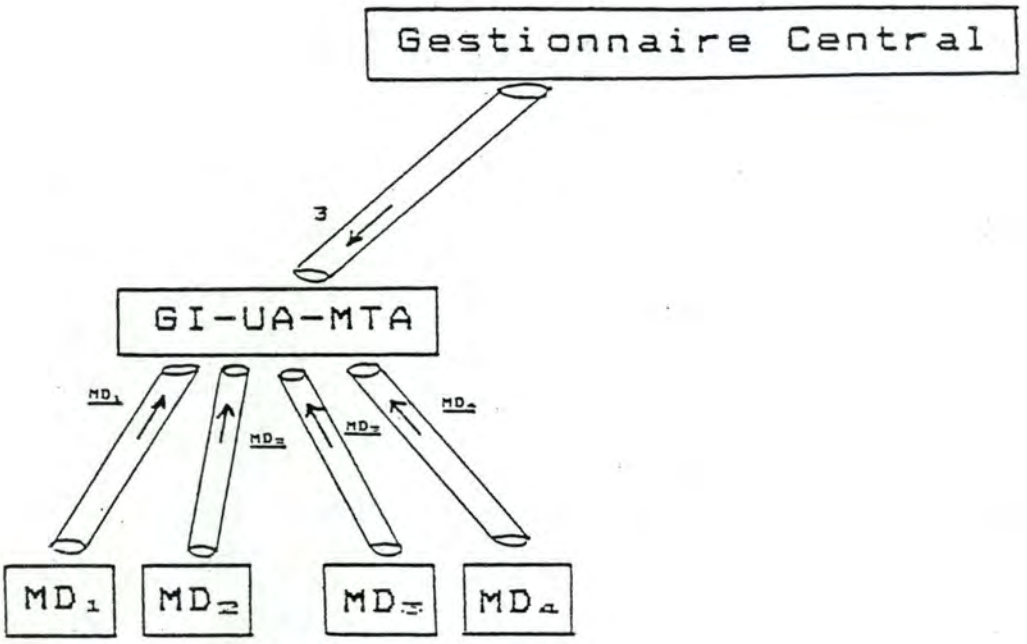


FIGURE 11.2.4 : MESSAGES ARRIVANT AU GI-UA-MTA.

d'autre part, en analysant les premiers champs des messages destinés à cet effet, le GI-UA-MTA est capable de déterminer à qui est destiné le message. La liste des messages se trouve en XI.8.

En fonction du message reçu, le GI-UA-MTA agit de deux manières différentes : soit il fait transiter directement le message vers un nouveau destinataire en modifiant en général le champ en-tête pour que celui-ci décode le message; soit il le place en fin de sa file d'attente car ce message requiert un certain travail. Il va faire précéder le message par le nom du pipe où celui-ci a été lu; ainsi il conserve toutes les données indispensables lors de l'exécution de ce travail.

11.2.5. EN RESUME

On peut conclure cette partie consacrée au GI-UA-MTA en rappelant son but : présenter comment celui-ci gère les ouvertures de connexions entre UA et MD, et aiguille les messages que ceux-ci s'échangent.

A cela, s'ajoute le fait qu'il garantit l'exécution de chaque demande.

Ce module MD fait l'objet du paragraphe suivant.

XI.3. LE MESSAGE DISPATCHER.

11.3.1. INTRODUCTION

Ce chapitre propose une présentation complète d'un message dispatcher tel qu'il est défini dans les normes MHS/X400 de 1984. Un des problèmes principaux qui s'est posé lors de la réalisation de cette description a été de trouver un formalisme adéquat pour exprimer tous les aspects du message dispatcher.

Le message dispatcher n'étant pas quelque chose de statique, il est bien entendu impossible d'utiliser à eux seuls les méthodes et moyens habituels de description. Pour mettre en vue les différents états que peut atteindre un MD, ainsi que les cas d'erreur lors de la réception de messages venant du GI-UA-MTA, l'idée est apparue d'utiliser un diagramme d'états. Malheureusement, ce choix de présentation est incomplet puisqu'il ne permet pas de préciser toutes les actions que le MD se doit d'effectuer dans chaque cas, ni la chronologie dans l'envoi et la réception des messages. L'algorithme du MD doit donc être vu comme un complément indispensable du diagramme d'états.

11.3.2. LE DIAGRAMME D'ETATS

Ce diagramme contient les différents états que peut atteindre un MD à un moment ou à un autre de son existence. En plus de ces états, le diagramme définit la façon dont le MD passe d'un état à un autre en envoyant ou en recevant des messages vers le GI-UA-MTA. Ces messages ainsi que

leurs conséquences sur les états et les actions du MD sont détaillés plus tard.

Le diagramme d'états du MD se trouve en fin de paragraphe XI.3.

11.3.2.1. LES ETATS

Parmi les états que renferme ce diagramme, on peut faire une distinction permettant de les classer en deux groupes. Tout d'abord, on peut isoler les états IDLE, CONNECTE et NON-CONNECTE qui peuvent être considérés comme stables. Un état stable est défini comme étant un état que peut atteindre un MD et dans lequel il peut rester inactif en attendant l'arrivée, dans sa file d'attente, d'une nouvelle demande à traiter.

Les trois états que l'on vient d'isoler sont opposés à tous les autres que l'on peut qualifier de transitoires. En effet, le MD qui atteint un de ces états transitoires suite à l'envoi ou à la réception d'un message ne reste jamais inactif lorsqu'il est dans cet état. Le MD aboutissant à un état transitoire, effectue un certain nombre d'actions et repasse alors, grâce à l'envoi, vers un autre état, stable ou transitoire. Le MD dans un état transitoire ne doit donc jamais attendre un message pour sortir de cet état. Puisque le MD ne stationne jamais dans un état transitoire, suite à la réception ou l'envoi d'un message, on peut alors définir un cycle à partir d'un état stable, comme étant la suite des actions effectuées et des états traversés par le MD, avant de rejoindre à nouveau un état stable.

Les états transitoires sont au nombre de huit : ATTENTE REPONSE, ANALYSE ROUTAGE, ANALYSE RAPPORT, ATTENTE PERMISSION DESTRUCTION, CREER UN SMPDU, TESTER P1, FIN TRAVAIL et TRAITEMENT DE DECONNEXION.

11.3.2.2. LES MESSAGES

La transition d'un état à un autre est définie à l'aide du concept de message. Un message, qu'il soit reçu ou à envoyer, est toujours composé de deux champs au moins. Le premier champ est une indication de la provenance du message dans le cas de la réception ou bien spécifie la destination que le message doit prendre. La deuxième partie du message en exprime la spécificité.

Tout comme l'AM et le RTS, le MD communique avec les autres entités de la couche application par l'intermédiaire des primitives proposées dans la norme. Cependant, pour permettre le parallélisme, il faut introduire de nouveaux types d'interaction (comme par exemple un dialogue entre deux MD via le GI-UA-MTA).

Les requêtes envoyées par une entité à une autre sont, tout comme les primitives de la norme, véhiculées entre deux parties du MTA grâce à des messages. Le MD, l'AM ou le RTS recevant une telle requête, la considère comme une nouvelle primitive. Ainsi, le nombre de primitives que chaque entité peut envoyer ou recevoir se voit élargi considérablement.

En ce qui concerne le MD, les nouvelles primitives créées dans le but de gérer le parallélisme sont au nombre de sept.

Lorsque le GI-UA-MTA veut signaler à un MD qu'il doit livrer le message de l'UA émetteur contenu dans un MPDU, à l'UA auquel il est connecté, il le fait à l'aide d'une nouvelle primitive 'deliver', interne au MTA.

D'autre part, quand le gestionnaire veut demander au MD si un éventuel message, conforme aux caractéristiques formulées dans la demande de PROBE, peut être transmis à l'UA connecté, il l'indique grâce à la primitive 'probe deliver' accompagnée d'un Probe MPDU.

Cependant, lorsque l'émetteur de la requête est un UA local, le message ne doit pas transiter par le réseau. Il est donc inutile de l'encapsuler dans un MPDU. C'est pour cette raison que les deux primitives citées précédemment ne conviennent plus et que les primitives 'local deliver' et 'local probe deliver', bien qu'ayant le même effet, ont été créées. Pour compenser l'absence du MPDU, ces primitives sont toujours accompagnées de paramètres donnés par l'UA origine lors de la soumission du message ou la demande de PROBE.

Pour permettre le dialogue entre l'AM et le MD, deux autres primitives sont ajoutées. Il s'agit de 'envoi' et 'relais'. Lorsque ces primitives sont véhiculées par des messages, elles sont accompagnées du MPDU. Ce MPDU existe puisqu'une interaction entre l'AM et le MD indique toujours un passage forcé par le réseau.

La dernière primitive permet de demander au MD de construire et envoyer une notification de non livraison. Cette primitive se nomme 'demnotnl' et est toujours accompagnée des données nécessaires à la création de la notification.

11.3.2.2.a. les messages reçus par le MD

Lorsque le GI-UA-MTA envoie un message au MD, il remplit le premier champ en indiquant l'origine de ce message.

En l'occurrence, le message peut avoir comme origine l'UA connecté au MD. Le GI-UA-MTA fait alors précéder de '0' le message qu'il transmet au MD.

Lorsque le message est une réponse de service demandé précédemment au GI-UA-MTA par le MD, le gestionnaire fait précéder le message retour de '1'.

Le dernier cas considère la réception par le MD d'un message provenant de l'AM. Une distinction peut être faite

à propos de ces messages, entre ceux qui véhiculent une demande de relais de la part de l'AM et ceux qui proviennent de l'AM et qui transportent une demande de notification de non livraison. Les premiers sont précédés de 'Null', les seconds commencent par '1'.

Les champs suivants viennent du message que le GI-UA-MTA reçoit du GC lorsque l'origine du message est l'AM ou l'UA. Dans le cas d'une réponse de service, le message est construit par le GI-UA-MTA lui-même.

Le détail de la composition du message aboutissant au MD est laissé pour plus tard, lors de l'explication de l'algorithme.

11.3.2.2.b. Les messages envoyés par le MD

Lorsqu'il envoie un message vers le GI-UA-MTA, le MD peut le faire précéder d'un champ '0' ou '1' indiquant au gestionnaire si le message qu'il vient de recevoir est à diriger vers le GC (le GI-UA-MTA est "transparent") ou si c'est une demande de service (il est destinataire).

Il est à noter que le deuxième champ est aussi une indication précieuse.

Pour les messages à destination du gestionnaire central (c'est-à-dire ceux dont le premier champ est '0'), le deuxième champ indique au GC la destination. Si le deuxième champ vaut '0', la destination est l'UA connecté au MD. Si c'est '1', la destination est l'AM et si c'est '2', le message reste au GC. Ce dernier cas tient compte de tous les messages destinés au MTA, c'est-à-dire les messages véhiculant les informations bilatérales ainsi que les données nécessaires au MTA, afin qu'il puisse calculer correctement les pourcentages de réussite de transmission de messages, d'utilisateur à utilisateur.

Pour les demandes de service, le deuxième champ du message indique la nature du service demandé au GI-UA-MTA par le MD. Ces demandes de service sont en fait des messages qui doivent être traités par le GI-UA-MTA. Parmi ces demandes de service, on trouve aussi des signaux envoyés par le MD, signaux qui n'appellent aucune réponse du gestionnaire. La présentation des messages dans le contexte de l'algorithme, va sans doute en permettre une meilleure compréhension.

11.3.3. L'ALGORITHME

L'algorithme du MD considère les trois états stables du diagramme d'état et envisage tous les messages que le MD peut recevoir dans chacun d'eux. Chaque réception d'un message déclenche une suite d'actions effectuées par le MD.

Ce sont ces actions qui sont expliquées et détaillées dans cet algorithme en pseudo-langage qui se trouve dans l'annexe 5. Chaque fois qu'une opération est effectuée, on explique le pourquoi de celle-ci ainsi que les répercussions qu'elle peut entraîner sur l'envoi de messages et le changement d'état du MD.

11.3.3.1. L'ETAT IDLE

Tant que la machine n'a pas été créée, c'est-à-dire tant que son bit d'existence dans la table 3 n'est pas à 1, elle se trouve dans l'état IDLE.

La machine quitte cet état IDLE et passe dans l'état NON CONNECTE dès que le GI-UA-MTA modifie le bit d'existence et lui fournit, par le pipe, le nom de la machine et le dernier numéro généré par ce MD, numéro qui représente le dernier identificateur de SUBMIT généré.

11.3.3.2. L'ETAT CONNECTE

Comme on l'a expliqué auparavant, le MD est relié au GI-UA-MTA par un pipe bidirectionnel. C'est dans ce pipe que vont se trouver les messages envoyés par le gestionnaire. Le premier travail du MD, une fois qu'il arrive dans l'état CONNECTE et qu'il se trouve au repos, est de vider le pipe et de mettre tous les messages qui s'y trouvent dans une file d'attente. Il existe une autre file d'attente qui contient les demandes de livraison différée. Le MD doit également consulter cette deuxième file et, s'il trouve une demande pour laquelle le temps d'attente est expiré, doit la placer à son tour dans la file d'attente principale du MD. Après cela, le MD va prendre la première demande se trouvant dans la file d'attente. Si aucune demande n'y figure, le MD retourne au début de l'algorithme gérant l'état CONNECTE pour vider à nouveau son pipe. Si par contre, il existe une demande en attente, il faut alors l'analyser.

La suite de l'algorithme envisage tous les messages que le MD peut trouver dans la file et suit, jusqu'au retour à un état stable, les répercussions de ces messages reçus par le MD. Tout autre message que ceux qui sont énoncés ci-dessus conduisent automatiquement à une erreur. La réception d'un message non repris dans cette liste implique directement le rejet de celui-ci. Voici les messages qui peuvent être reçus par le MD :

1) | O | Cancel |

A la lecture de ce message, le MD sait que l'UA désire la destruction d'une demande de livraison différée envoyée auparavant. Avant de retourner dans l'état CONNECTE et attendre une nouvelle demande, le MD doit effectuer la procédure Cancel dont les spécifications ont été détaillées lors des spécifications externes du MTA.

2) | O | UAL-Control-req |

| O | Register-req |

Lorsqu'il reçoit un de ces deux messages, le MD sait que l'UA désire changer les restrictions qu'il a imposées auparavant. Il faut signaler que les contraintes définies par la primitive CONTROL voient leur effet limité à la connexion courante, alors que celles qui sont imposées par une primitive REGISTER sont valables jusqu'au Register suivant.

Le travail du MD est de lancer la procédure Control ou la procédure Register toutes deux expliquées dans l'étape des spécifications externes du MTA. Toutes les contraintes imposées par L'UA sont mémorisées par le MD dans la table 4.

3) | O | Chge-password |

Ce message est donné au MD par l'UA lorsque celui-ci désire changer son mot de passe. Le MD va alors lancer la procédure UAL-Change-Password décrite dans les spécifications externes du MTA en vue de transformer la table 4.

4) | 0 | Probe-req |

Lorsqu'un tel message aboutit au MD, celui-ci déclenche la procédure de Probe, présentée lors des spécifications externes. Si le résultat de la vérification des paramètres est favorable, le MD exécute la procédure Probe-req spécifiée en 11.3.4.1. Sinon, le message véhiculant le probe est écarté.

5) | 0 | Submit-req |

Il s'agit d'un cas semblable à celui du Probe. Après avoir exécuté la procédure de Submit définie lors des spécifications externes du MTA, le MD déclenche la procédure Submit-req expliquée en 11.3.4.2., si la vérification des paramètres le permet. Si pas, la demande est écartée.

6) | 0 | MTL-Control-resp |

La réception de ce message confirme au MD que l'UA a bien reçu les restrictions imposées par le MTA en ce qui concerne les messages dont il accepte la transmission. Ces restrictions ont été initialisées par un MTL-Control-ind envoyé par le MTA.

7) | 0 | Local | probe deliver | paramètres probe | id-UA origine |

Comme pour tous les messages suivants, le premier champ à '0' n'est pas vraiment logique. Puisqu'il s'agit de conséquences immédiates de demandes de service, ces messages devraient commencer par '1'. Cependant, le GI-UA-MTA initialise le premier champ à '0' pour faire croire au MD que le message lui vient d'un UA.

Ce message est envoyé par le GI-UA-MTA à la suite d'une demande de service | 1 | 3 | id-UA | Local probe deliver | param probe | id-UA orig | venant d'un autre MD. Voulant transmettre un probe à un UA local qui ne lui est pas connecté, ce MD est obligé de faire une demande de service au gestionnaire pour que celui-ci puisse trouver le MD bien connecté. Lorsque le gestionnaire l'a déterminé, il lui envoie ce message | 0 | local probe deliver | param probe | id-UA orig | "en lui demandant" de contrôler si un message vérifiant les contraintes spécifiées dans les paramètres du Probe peut être transmis à son UA. Le message envoyé au MD par le gestionnaire contient l'identificateur de l'UA origine pour permettre à l'UA destinataire de savoir à qui renvoyer la notification du Probe. Cet identificateur est indispensable, puisqu'il s'agit ici d'un Probe entre deux UA locaux comme l'indique le quatrième champ du message reçu. Le MD ne reçoit donc pas la primitive PROBE encapsulé dans un MPDU, mais bien les paramètres du Probe tels qu'ils ont été donné au MTA par l'UA origine. L'UA destinataire n'a donc aucun moyen de connaître l'identificateur de l'UA origine si on ne lui fournit pas. Pour gérer la réception de ce message, le MD exécute la procédure Local-Probe-Deliver décrite en 11.3.4.3.

8) | 0 | Local deliver | parametres submit | id-UA origine |

Lorsque le GI-UA-MTA reçoit d'un MD une demande de livraison de message vers un UA local, il doit trouver le MD connecté à cet UA destinataire. Quand il l'a découvert, il lui envoie le message | 0 | Local deliver | param submit | id-UA or | lui indiquant qu'il s'agit d'une livraison de message vers son UA. Tout comme pour le local probe deliver, l'identificateur de l'UA origine est indispensable puisque, le message étant véhiculés entre deux UA locaux, il n'existe pas de MPDU. Après la réception de ce message, le MD va lancer l'exécution de la procédure Local-Deliver spécifiée en 11.3.4.4.

9) | O | Probe deliver | MPDU |

La réception de ce message par un MD indique une demande de Probe provenant d'un autre MTA à destination de l'UA connecté au MD considéré. Ce message est à distinguer de | O | local probe deliver | param probe | id-UA or | qui stipule lui-aussi une demande de probe, mais cette demande provient d'un UA local. Lorsque le MD reçoit le message Probe deliver, il exécute la procédure Probe-Deliver spécifiée en 11.3.4.5.

10) | O | Deliver | MPDU |

Lorsqu'un message vient de l'AM, le GI-UA-MTA crée un nouvel MD pour s'occuper de la réception de ce message. Il est bien évident, dès lors, que s'il s'avère que le message reçu doit être livré à un UA local, il est obligatoire de trouver un autre MD connecté à l'UA destinataire ou bien de connecter le MD créé à cet UA visé. Dans les deux cas, le gestionnaire envoie le message | O | deliver | MPDU | vers le MD choisi.

Le MD va alors lancer la procédure Deliver décrite en 11.3.4.6 pour qu'il puisse gérer la livraison de ce message.

11) | O | Notify |

La réception de cette demande indique au MD qu'il doit envoyer vers l'UA auquel il est connecté, la primitive NOTIFY reçue. Contrairement aux demande de livraison d'un message et demande de probe, on ne fait pas ici de distinction concernant l'origine du message, elle peut être locale ou pas. En effet, puisqu'en cas de problème, on ne renvoie pas de notification de non livraison, le MD destinataire n'a pas besoin des informations contenues dans le MPDU. On peut donc directement créer la primitive NOTIFY. L'arrivée de ce

message est gérée par le MD à l'aide de la procédure Notify expliquée en 11.3.4.7.

12) | O | Demnotnl | Message |

Lorsque le MD doit envoyer un message vers un UA local, il transmet, s'il n'est pas lui-même connecté au bon UA, une demande de service au GI-UA-MTA pour que celui-ci trouve un MD connecté à l'UA destinataire.

La réception de ce message | O | demnotnl | Message | indique au MD demandeur que la requête a échoué, c'est-à-dire que le GI-UA-MTA n'est pas parvenu à trouver un MD bien connecté et qu'il lui faut créer une notification de non livraison si elle n'est pas interdite. Le paramètre 'message' indique au MD dans quel cadre cette demande a échoué. Le MD pouvait avoir demandé au gestionnaire un autre MD en vue de livrer un probe ou un submit vers un UA local, de donner un notify ou de transmettre un probe, un submit ou un notify vers un autre MTA.

Pour gérer la réception d'un tel message, le MD lance alors l'exécution de la procédure Demnotnl expliquée en 11.3.4.8.

13) | O | Logoff |

Lorsque le MD reçoit ce message, il déclenche la procédure Logoff décrite lors des spécifications externes du MTA, suivie de l'exécution de la procédure Fermeture décrite en 11.3.4.9.

14) | -1 | OR | Demnotnl | MPDU |

Lorsque l'AM signale au GI-UA-MTA un problème, le gestionnaire envoie le message | -1 | AM ou GAM | Demnotnl | MPDU | avec le MPDU fourni par l'AM, vers un MD quelconque. Normalement, le GI-UA-MTA transmet ce message vers un MD non connecté à un UA pour ne pas interférer dans les relations entre un UA et son MD. lorsqu'aucun MD n'est plus disponible, le gestionnaire envoie la demande de notification vers un MD connecté. C'est le cas qui est géré ici.

Le paramètre OR indique un problème au niveau du GI-UA-MTA s'il vaut 'GAM'. Ce problème peut être un nombre insuffisant de connexions ou d'AM disponibles. Tandis que OR = AM indique un problème au niveau de la transmission du MPDU. Après cette réception, le MD exécute la procédure Problam décrite en 11.3.4.10.

11.3.3.3. L'ETAT NON CONNECTE.

Comme cela a été précisé en début de chapitre, l'état NON CONNECTE est le troisième état stable par lequel le MD peut passer. Le MD peut donc très bien se trouver dans un état NON CONNECTE et ne rien avoir à faire. Dans ce cas, le MD doit aller lire dans le pipe les messages qui lui viennent du GI-UA-MTA et les mettre en fin de file d'attente. Ensuite, il doit aller lire les demandes différées qui se trouvent dans la file des livraisons différées et, lorsqu'elle peuvent être traitées, les remettre elles-aussi dans sa file d'attente.

Une fois ce travail terminé, le MD traite la première demande se trouvant dans la file d'attente. Si toutefois, la file d'attente est vide, le MD envoie un message | 1 | 2 | num machine | dernier numéro généré | précisant qu'il n'a plus rien à faire. Après cet envoi, il retourne dans l'état NON CONNECTE.

Si par contre, le MD trouve un message à traiter dans sa file d'attente, il peut être de plusieurs types, à savoir :

1) | Null | relais |

Le MD reçoit ce relais lorsqu'il vient d'être créé en vue de recevoir un message venant de l'AM, message devant être relayé vers un autre MTA ou livré à un UA local. La réception de ce message déclenche l'exécution de la procédure Relais spécifiée en 11.3.4.11.

2) | 0 | deliver | MPDU |

| 0 | local deliver | param submit | id-UA or |

| 0 | notify |

| 0 | probe deliver | MPDU |

| 0 | local probe deliver | param probe | id-UA or |

Lorsque le GI-UA-MTA envoie un de ces messages à un MD pour lui indiquer qu'aucun MD ne peut traiter sa demande, il se peut qu'entre temps, le MD receveur ait été déconnecté sans que le gestionnaire n'en ait été averti.

Ainsi, un MD déconnecté peut recevoir de tels messages et les gérer. Le rôle du MD dans ce cas est de retransmettre le message reçu dans une demande de recherche de bon MD et de faire suivre ce message du signal | 1 | 4 | indiquant au GI-UA-MTA qu'il vient de traiter et terminer une réponse à un service.

3) | O | logon |

| O | submit-req |

| O | probe-req |

| O | cancel |

| O | register-req |

| O | UAL-control-req |

| O | chge-password |

| O | logoff |

La réception de ces messages est due à une erreur au niveau de l'UA : sans attendre la confirmation du logoff, il envoie une nouvelle demande vers le MD. Ces messages transitent à travers le gestionnaire vers le MD puisque ce gestionnaire n'a pas été averti suffisamment tôt du logoff. Lorsque le MD reçoit un tel message, il l'écarte et aucune suite n'est donnée à la demande.

4) | O | demnotnl | message |

Lorsque le MD a été créé pour recevoir un message de l'AM, il se peut qu'après la procédure recherche-chemin, l'UA destinataire soit un UA local. Dans ce cas, le MD envoie une demande de service au GI-UA-MTA pour que celui-ci aiguille le message vers le MD connecté à l'UA destinataire. Lorsque le gestionnaire ne parvient pas à répondre à cette demande, il renvoie un message | O | demnotnl | message | au MD. Une fois qu'il reçoit ce message, le MD déclenche la procédure Demande-notif spécifiée en 11.3.4.12.

5) | -1 | OR | demnotnl | MPDU |

Comme il a été expliqué auparavant, lorsque le GI-UA-MTA reçoit une demande de notification de non livraison depuis l'AM, il envoie | -1 | OR | demnotnl | MPDU | à un MD quelconque, mais de préférence à un MD non connecté. C'est le cas envisagé ici. Lorsque le MD découvre dans sa file d'attente un tel message, il exécute la procédure problam spécifiée en 11.3.4.10.

6) | 1 | destruction |

Quand il reçoit ce message, le MD est sûr que le GI-UA-MTA a reçu le signal | 1 | 1 | table 4 . Le MD passe donc en état IDLE, en attendant d'être réutilisé pour un autre UA.

7) | 1 | id-UA | table 4 |

Ce signal, reçu par le MD, lui donne l'identificateur de l'UA auquel il est nouvellement connecté, ainsi que les restrictions imposées précédemment par cet UA à l'aide d'une primitive register. Une fois ce signal reçu, le MD passe dans l'état CONNECTE.

11.3.4. LES PROCEDURES

11.3.4.1. PROBE-REQ

Cette procédure probe-req permet au MD de détecter les destinataires et d'analyser la route à suivre pour transmettre cette demande de sonde. Après avoir regroupé dans un même MPDU, les messages ayant une partie de route à faire en commun, le MD envisage les unes après les autres les directions obtenues par la procédure recherche-chemin spécifiée en 11.3.4.13. Trois cas peuvent alors se présenter :

1) L'UA destinataire est local au MTA et est en fait l'UA connecté au MD considéré.

Après vérification des restrictions imposées par l'UA à l'aide des primitives CONTROL et REGISTER, le MD envoie vers l'UA, si ça ne lui est pas interdit par le control, une notification indiquant à celui-ci s'il peut s'envoyer un message vérifiant les restrictions transmises dans le probe. Pour transmettre à son UA les résultats de la demande de probe, le MD envoie vers le GI-UA-MTA le message | O | O | id-UA | notify | . Comme on l'a précisé auparavant, le premier champ à 'O' indique que le message est à destination du GC. Le deuxième 'O' est à l'intention de ce GC, il lui indique que le message qu'il reçoit doit être dirigé vers l'UA dont l'identificateur est contenu dans le troisième champ du message. Le dernier champ contient la notification véhiculant les résultats de la demande de probe faite par l'UA.

2) L'UA destinataire est local au MTA mais n'est pas connecté au MD qui prend le probe en charge.

Dans ce cas, le MD demande au GI-UA-MTA de trouver un MD qui est connecté à l'UA local destinataire. Le message que le MD transmet alors au gestionnaire se

compose de plusieurs champs. Les deux premiers indiquent au gestionnaire qu'il s'agit d'une demande de service concernant la recherche d'un MD. Le champ suivant donne l'identificateur de l'UA pour lequel on recherche un MD. Le quatrième champ définit le type du message. Il s'agit ici d'un local probe deliver c'est-à-dire d'une demande de probe d'un UA vers un autre UA, tous deux utilisant les services du même MTA. Les paramètres nécessaires à la création de la notification se trouvent dans le cinquième champ, le dernier champ spécifiant l'identificateur de l'UA origine pour que la notification puisse être envoyée à qui de droit. Cet identificateur de l'UA origine est obligatoire dans le cas d'un message entre deux UA locaux contrairement au cas où un MPDU doit être créé pour assurer le relais, auquel cas les paramètres de l'enveloppe suffisent à retrouver l'émetteur du message. Le message complet se présente donc de cette manière :

1	3	local probe deliver	param probe	id-UA or

.

Lorsque le GI-UA-MTA a trouvé le MD recherché, il dispose de toutes les données pour lui transmettre la demande de probe à envoyer à l'UA.

3) L'UA destinataire n'est pas local.

Le MD doit alors transmettre la demande de probe au MTA voisin sur la route vers la destination. L'adresse de ce MTA voisin est donnée par la procédure recherche-chemin spécifiée ci-dessous en 11.3.4.13. Le MD doit donc créer le MPDU et envoyer le tout vers le GI-UA-MTA en lui signalant qu'il s'agit d'un message à faire passer vers un AM. Ce message à destination d'un AM est de la forme

0	1	AMdest	envoi	MPDU

. Les deux premiers champs indiquent aux gestionnaires que le message est destiné au MTA adjacent dont l'adresse de l'AM se trouve dans le troisième champ. Le quatrième champ signale qu'il s'agit d'un message véhiculant un envoi vers l'extérieur du MTA. Toutes les informations nécessaires à cet envoi, sont reprises dans le MPDU.

Après avoir passé en revue toutes les directions, le MD, avant de clôturer la gestion du probe, doit s'assurer que la procédure de recherche de chemins n'a pas décelé des identificateurs d'UA inconnus. Dans le cas où il existerait de tels UA, le MD doit envoyer vers son UA, une notification stipulant tous les UA non reconnus.

11.3.4.2. SUBMIT-REQ

Le principe est tout à fait le même que pour la procédure probe-req. Quelques petites différences doivent cependant être retenues en ce qui concerne l'envoi des notifications de livraison et de non livraison qui, dans le cas d'un submit, sont à la demande de l'UA.

Dans le cas où l'UA destinataire est local et connecté au MD considéré, on doit pouvoir permettre à l'UA d'interdire la réception de notifications de non livraison, notifications qui sont créées et envoyées par défaut. On doit aussi autoriser l'UA à demander l'envoi par l'UA destinataire, d'une notification de livraison. Ces deux paramètres sont contenus dans les paramètres du submit, et il convient d'en tenir compte pour la création des notifications. La livraison du message à l'UA ainsi que les notifications éventuelles sont envoyées vers le GI-UA-MTA grâce au message | 0 | 0 | id-UA | primitive | . Le paramètre 'primitive' du quatrième champ représentant selon le cas, un notify ou un deliver.

Dans le cas d'un UA destinataire local mais non connecté au MD considéré, le MD envoie au gestionnaire un message du même type que celui qu'il envoyait dans une telle situation pour un probe. Le message se présente ici de cette façon : | 1 | 3 | id-UA | local deliver | param submit | id-UA or | . Ceci permet au gestionnaire de transmettre au MD connecté à l'UA destinataire, toutes les informations nécessaires pour mener à bien la demande de livraison du message formulée par l'UA origine.

Le relais se fait de la même manière que pour le probe. Le MD n'a qu'à créer le MPDU et l'envoyer vers le GI-UA-MTA grâce au message | 0 | 1 | AMdest | envoi | MPDU | qui indique au gestionnaire que ce message est à destination de l'AM.

Le cas des UA non reconnus envisagé dans la procédure probe-req se traite ici de manière semblable. Il faut toutefois préciser que la notification de non livraison à renvoyer à l'UA émetteur ne peut se faire que si celui-ci n'a pas interdit toute notification de non livraison lorsqu'il a transmis la demande d'envoi à son MD.

11.3.4.3. LOCAL-PROBE-DELIVER

Cette procédure va être déclenchée par un MD connecté à un UA pour gérer une demande de probe venant d'un UA local et destiné à l'UA connecté au MD considéré.

Cette procédure, après comparaison des restrictions imposées par l'UA destinataire et des paramètres du probe transmis dans le message, va envoyer à l'UA origine dont l'adresse a également été donnée par le gestionnaire, une notification contenant les résultats du probe en transmettant au GI-UA-MTA, le message | 1 | 3 | id-UA | Notify |.

Cette procédure, terminant une réponse à un service demandé par un MD doit se clôturer par l'envoi d'un signal | 1 | 4 | vers le gestionnaire, permettant à celui-ci de décrémenter le verrou dont on a vu l'utilité au paragraphe XI.2.

11.3.4.4. LOCAL-DELIVER

Cette procédure est utilisée par le MD lorsqu'il a reçu du gestionnaire une demande lui indiquant de livrer à son UA le message envoyé par un autre UA local.

Mais avant de transmettre le message à l'UA auquel il est connecté, le MD doit s'assurer que les restrictions imposées par l'UA grâce aux primitives CONTROL et REGISTER sont respectées. De plus, le MD doit vérifier que les conversions nécessaires, si elles sont permises, ont bien été effectuées.

Lorsque toutes les vérifications sont positives, le MD envoie vers son UA (dont il trouve l'identificateur dans les paramètres du submit, transmis par le GI-UA-MTA) le message à livrer à l'aide du message | 0 | 0 | id-UA | deliver | . Si une notification de livraison a été demandée ou si une incompatibilité a été découverte lors des vérifications et que l'UA origine accepte les notifications de non livraison, le MD demande au GI-UA-MTA de transmettre à l'UA origine les notifications demandées grâce au message | 1 | 3 | id-UA | notify | .

Puisque tout ceci clôture une demande de service de la part du MD connecté à l'UA local, la procédure se termine par l'envoi d'un message | 1 | 4 | signalant la fin de la gestion du service demandé.

11.3.4.5. PROBE DELIVER

La grosse différence entre cette procédure et local-probe-deliver, c'est que celle-ci donne suite à un probe venant d'un autre MTA et donc travaille sur un MPDU. Pour s'assurer qu'un message vérifiant les critères spécifiés dans la demande de probe peut être transmis à son UA, le MD doit tout d'abord découper le MPDU. Après

comparaison de ce qui est spécifié dans le probe et de ce qui est autorisé par l'UA, le MD renvoie une notification.

Afin d'envoyer cette notification, il utilise la procédure recherche-chemin qui lui donne le premier MTA sur la route vers la destination de la notification.

Ce message reçu par le MD fait suite à une demande de service demandé au gestionnaire par un autre MD. Cette procédure gérant la réception de ce message doit alors se terminer par l'envoi vers le GI-UA-MTA du signal | 1 | 4 | .

11.3.4.6. DELIVER

Cette procédure va aider le MD à gérer la livraison à l'UA qui lui est connecté d'un message provenant d'un MTA voisin. Avant de pouvoir transmettre le message à l'UA, le MD doit découper le MPDU pour en extraire les données nécessaires, entre autres, à la vérification des contraintes imposées par l'UA et à l'accomplissement des conversions souhaitées.

Lorsque le MD ne trouve aucune trace d'erreur lors des vérifications, il peut envoyer la primitive deliver vers son UA grâce au message | 0 | 0 | id-UA | deliver transmis au GI-UA-MTA. Si, dans le MPDU, l'UA origine demande une notification de livraison, le MD doit la fabriquer et utiliser la procédure recherche-chemin afin de trouver vers quel MTA la transmettre.

Lorsque le MD ne parvient pas à livrer le message, il envoie toujours une notification de non livraison vers le MTA desservant l'UA origine. Cette notification n'a peut-être pas été demandée par l'UA origine, mais est nécessaire pour les contrôles (statistiques, de type gestion, ...) effectués par le MTA origine. Cette notification est envoyée dans le message | 0 | 1 | AMdest | envoi | MPDU | à destination du GI-UA-MTA.

Faisant suite à une demande de service, cette procédure se termine par l'envoi du signal | 1 | 4 | .

11.3.4.7. NOTIFY

Cette procédure permet au MD de gérer la livraison des notifications. Avant de transmettre à son UA la notification qu'il vient de recevoir, le MD doit s'assurer que cet UA n'a pas interdit toute réception de notifications.

Lorsqu'il est autorisé à le faire, le MD envoie la notification vers son UA grâce à l'envoi du message | 0 | 0 | id-UA | notify | vers le GI-UA-MTA. Quand la réception de notifications est interdite, celles-ci sont perdues et aucune suite n'est donnée, puisqu'une notification de non livraison d'une notification n'est pas permise. Dans ce cas, la notification est écartée.

11.3.4.8. DEMNOTNL

Cette procédure s'occupe de gérer la réception d'une demande de notification de non livraison suite à un problème venant du gestionnaire. Le MD peut devoir construire une notification suite à différentes demandes :

- Notify :

  Le MD n'a alors rien à faire puisque le gestionnaire lui signale qu'il n'arrive pas à transmettre la primitive notify vers le bon MD. Aucune suite n'est alors donnée à ce message puisqu'on n'envoie pas de notification de non livraison quand il s'agit d'une notification.

- Local deliver :

Le gestionnaire GI-UA-MTA indique au MD demandeur que la livraison du message ne peut pas être effectuée. Le MD recevant le message doit alors voir si l'UA origine permet la réception de notification de non livraison. Si oui, le MD crée et envoie une notification de non livraison vers son UA par l'intermédiaire du GI-UA-MTA en lui transmettant le message | O | O | id-UA | notify | .

- Local probe deliver :

La réception du message | O | demnotnl | local probe deliver | provoque la création et l'envoi d'une notification signalant la non livraison, vers l'UA connecté au MD à moins que cet UA n'ait interdit, par une primitive control, toute réception de notification de non livraison.

- Relais :

Le MD reçoit du GI-UA-MTA un signal précisant qu'il n'a pas pu , faute de machines disponibles, créer de nouvel MD pour assurer le relais. Comme le message véhiculé par le relais vient d'un MTA voisin, le MD dispose donc de le MPDU. Le MD va alors découper ce MPDU et voir s'il s'agit d'une notification ou pas. S'il s'agit d'une notification, le MD ne donne aucune suite. Si pas, le MD crée une notification de non livraison (toujours demandée par le MTA origine) en regardant dans le MPDU ce qui est demandé par l'UA origine. Après utilisation de la procédure recherche-chemin, le MD envoie le SRPDU vers l'UA origine à l'aide du message | O | 1 | AMdest | envoi | SRPDU | qu'il transmet au GI-UA-MTA.

11.3.4.9. FERMETURE

Cette procédure est déclenchée par le MD lorsque celui-ci a répondu à la demande de logoff venant de l'UA. Le MD envoie alors un signal | 1 | 1 | table 4 | au GI-UA-MTA pour lui signaler qu'il a reçu un logoff et pour lui repasser les dernières valeurs imposées par l'UA lors de l'envoi de la primitive REGISTER. Une fois ceci terminé, le MD passe dans l'état NON CONNECTE.

11.3.4.10. PROBLAM

Cette procédure permet au MD de gérer une demande de notification de non livraison venant de l'AM. Lorsque le MD reçoit un tel message, il doit tout d'abord commencer par découper le MPDU. Si le message pour lequel on demande une notification de non livraison est déjà une notification, on se contente d'envoyer un signal | 1 | 4 | pour indiquer qu'on a terminé la gestion d'une demande de service qui, en l'occurence, vient de l'AM.

Si c'est un User MPDU ou un Probe MPDU, on va regarder dans l'enveloppe et plus précisément dans Recipientinfo quelles sont les destinations dont le MTA est responsable pour pouvoir signaler à l'UA origine, quels destinataires on n'a pas pu atteindre. Une fois ceci terminé, le MD va déclencher la procédure Recherche-chemin vers l'UA origine. Si cet UA origine est local, le MD va regarder dans le MPDU si l'UA origine a demandé une notification de non livraison. Si c'est le cas, le MD va analyser l'identificateur de cet UA origine. Si le MD lui est directement connecté, alors, le MD envoie directement vers l'UA la primitive notify enfermée dans le message | 0 | 0 | id-UA | notify | . Si pas, le MD transmet au GI-UA-MTA une demande de service | 1 | 3 | id-UA | notify | pour faire passer la notification au MD connecté à l'UA origine.

Dans tous les cas où l'UA origine n'est pas local, le MD crée le rapport de non livraison et met ce SRMPDU dans un message qu'il envoie vers l'AM en vue du relais. Ceci se concrétise par l'envoi vers le gestionnaire du message | 0 | 1 | AMdest | envoi | SRMPDU | . Dans tous les cas, cette procédure répondant à un service de l'AM, est terminée par l'envoi du signal | 1 | 4 | .

11.3.4.11. RELAIS

Cette procédure gère la réception d'un message provenant de l'AM. Puisqu'il vient de l'AM, le message est formé de le MPDU. Le premier travail est donc de découper ce MPDU en champs permettant une analyse facile, et ensuite, de s'assurer que le MPDU reçu est bien conforme à P1. S'il ne l'est pas, on rejette le message reçu et on retourne à l'état NON CONNECTE.

Si le message est compréhensible par le décodeur P1, il faut alors utiliser la procédure Recherche-chemin permettant de déterminer les différentes directions à prendre en vue d'atteindre tous les UA destinataires.

Une fois le nombre de directions connu, deux cas peuvent se présenter :

Cas n°1 : Le MPDU est un User MPDU ou un Probe MPDU.

Dans ce cas, il faut fabriquer autant de copies que de directions décelées par la procédure Recherche-chemin. Pour chaque copie, on transforme le MPDU en changeant les informations de trace et les bits de responsabilité.

Lorsque ceci est terminé, le MD regarde dans les informations de trace, s'il n'y a pas eu de bouclage lors du routage. Si oui, il faut renvoyer une notification de non livraison indiquant le bouclage. Cette notification

est envoyée dans un MPDU à l'UA origine, après avoir déclenché la procédure Recherche-chemin pour obtenir vers quel MTA voisin envoyer le message [0 | 1 | AMdest | envoi | MPDU] .

Lorsque l'information de trace n'indique pas de bouclage, il faut voir, dans les résultats donnés par la procédure Recherche-chemin, si l'UA destinataire est local ou pas. S'il est local, on envoie une demande de service au GI-UA-MTA pour qu'il aiguille le message vers le MD connecté au bon UA. Le message envoyé au GI-UA-MTA peut être [1 | 3 | id-UA | deliver | MPDU] s'il s'agit d'un User MPDU ou [1 | 3 | id-UA | probe deliver | MPDU] s'il s'agit d'un Probe MPDU. Lorsque la direction précise qu'il ne s'agit pas d'un UA local, le message est dirigé vers le MTA suivant sur la route, MTA dont l'adresse est donnée par la procédure Recherche-chemin. Le MD envoie donc au gestionnaire un message [0 | 1 | AMdest | envoi | MPDU] véhiculant le MPDU.

Après avoir effectué toutes ces opérations pour chacune des directions, le MD doit vérifier que, lors de la détermination des directions, la procédure n'a pas décelé de direction inconnue. S'il en existe, le MD doit créer une notification de non livraison contenant toutes les identifications des UA que le message ne peut pas atteindre. Il faut ensuite utiliser la procédure Recherche-chemin pour savoir vers quel MTA voisin transmettre la notification contenue dans le MPDU du message [0 | 1 | AMdest | envoi | APDU] à envoyer au gestionnaire.

Cas n°2 : Le MPDU est un Delivery Report MPDU.

Dans ce cas, il n'y a qu'une seule copie pour laquelle la direction à prendre est donnée par la procédure Recherche-chemin. A moins que cette procédure ne renseigne une destination incompréhensible, auquel cas on laisse tomber le MPDU, différents cas peuvent se présenter. Si la direction suivante à prendre par le message n'est pas locale, il faut modifier le MPDU avec les nouvelles informations concernant la responsabilité du message, ainsi que la trace. S'il n'existe pas de bouclage, le message avec le MPDU transformé est envoyé

vers le GI-UA-MTA en lui indiquant que le message à véhiculer vers le MTA voisin (dont l'adresse est donnée par la procédure Recherche-chemin) est un Delivery Report MPDU. Si le MD décèle une boucle dans la trace, puisqu'il s'agit d'une notification, aucune suite n'est donnée à ce problème. Le message est abandonné.

Dans le cas où la procédure Recherche-chemin indique que la destination du MPDU reçu est locale, il faut aller voir dans le champ adéquat du MPDU si la notification est de livraison ou de non livraison. Le MD doit alors consulter la valeur de Recipientinfo pour savoir s'il faut envoyer une notification au MTA origine pour qu'il complète ses informations. Ce Recipientinfo permet aussi au MD de déterminer quelles informations il doit transmettre à l'UA. Si le MD constate qu'une notification doit être livrée à l'UA destinataire, il va envoyer au GI-UA-MTA, le message

1	3	id-UA	notify

pour que le gestionnaire transmette le tout au MD connecté au bon UA.

11.3.4.12. DEMANDE-NOTIF

Cette procédure permet au MD de gérer la réception d'un message lui indiquant que l'UA destinataire ne peut être atteint. La demande de notification peut être accompagnée de quatre types de messages, repris dans le paramètre ' message ' de

0	Demnotnl	message

- Notify :

Notify signale au MD que la notification qu'il voulait transmettre ne peut pas l'être. Dans ce cas, la notification est mise de côté.

- Deliver :

Deliver signifie que le GI-UA-MTA n'a pas trouvé le MD connecté à l'UA auquel on voulait livrer un message initialisé par un autre UA lors d'un Submit. Dans ce cas, on envoie vers l'UA origine, une notification de non livraison à l'aide du message | O | 1 | AMdest | envoi | MPDU | transmis au gestionnaire. Si la procédure Recherche-chemin signale une erreur dans l'identificateur de l'UA origine, tout est perdu.

- Probe deliver :

Probe deliver indique au MD que la demande de Probe n'a pas pu atteindre le MD connecté à l'UA destinataire. Dans ce cas, le MD crée et envoie une notification vers l'UA origine si la procédure Recherche-chemin n'a décelé aucune erreur dans l'identificateur de cet UA origine. Si tel est le cas, aucune suite n'est donnée à la réception de ce message.

- Relais :

Lorsque le MD reçoit 'relais' en paramètre du message demandant une notification de non livraison qu'il vient de lire dans sa file, il regarde si c'est un MPDU qui véhicule une notification. Si oui, il ne donne pas suite à ce message. Si ce n'est pas une notification, le MD crée et envoie une notification de non livraison vers l'UA origine dirigeant le MPDU vers le MTA voisin donné par la procédure Recherche-chemin.

11.3.4.13. RECHERCHE-CHEMIN

Cette procédure permet au MD d'obtenir, à partir des noms des UA destinataires, le nombre de directions différentes que le message doit emprunter vers ses destinations. Ceci indique au MD le nombre de copies à

créer. Un UA local est comptabilisé comme étant une direction, de même que tout MTA voisin sur la route vers le MTA destinataire.

Pour chaque direction, la procédure fournit les informations suivantes :

- la situation (local ou non local)
- l'adresse (c'est un UA ou un MTA voisin)
- quels destinataires sont concernés par cette direction.

La procédure indique aussi les noms des UA destinataires qu'elle n'a pas reconnu.

DIAGRAMME D'ETATS

MD1

MD 1

ANALYSE ROUTAGE

CREATION PDU

ANALYSE RAPPORT

ATTENTE REPONSE

CONNECTE

IDLE

TRAITEMENT DE DECONNEXION

NON CONNECTE

ATTENTE PERMISSION DESTRUCTION

FIN TRAVAIL

C1

C2

C3

C4

C5

C6

C7

C8

C10

0 1 AM DEST ENVOI APDU

LOCAL
0 3 ID-UA PROBE DELIVER PARAM-PROBE ID-UA OR

LOCAL
1 3 ID-UA DELIVER PARAM-SUBMIT ID-UA OR

1 3 ID-UA NOTIFY

1 + 0 SUBMIT-req

1 + 0 PROBE-req

0 ID-UA DELIVER

LOCAL
0 PROBE DELIVER (PARAM-PROBE) ID-UA OR

0 PROBE DELIVER APDU

1 3 ID-UA NOTIFY

0 0 ID-UA NOTIFY

-1 AM/GAM DEMNOTNL APDU

0 DELIVER (APDU)

0 DEMNOTNL MESSAGE

0 NOTIFY

0 LOCAL DELIVER (PARAM-SUBMIT) ID-UA OR

0 0 ID-UA MTL CONTROL-req

0 MTL-CONTROL-resp

0 SUBMIT-req

0 UAL-CONTROL-req

0 CHANGE PASSWORD

0 REGISTER-req

0 CANCEL

1 1 TABLE 4

0 LOGOFF

1 ID-UA TABLE 4

1 4

1 ID-UA TABLE 4

1 ID-UA TABLE 4

1 ID-UA TABLE 4

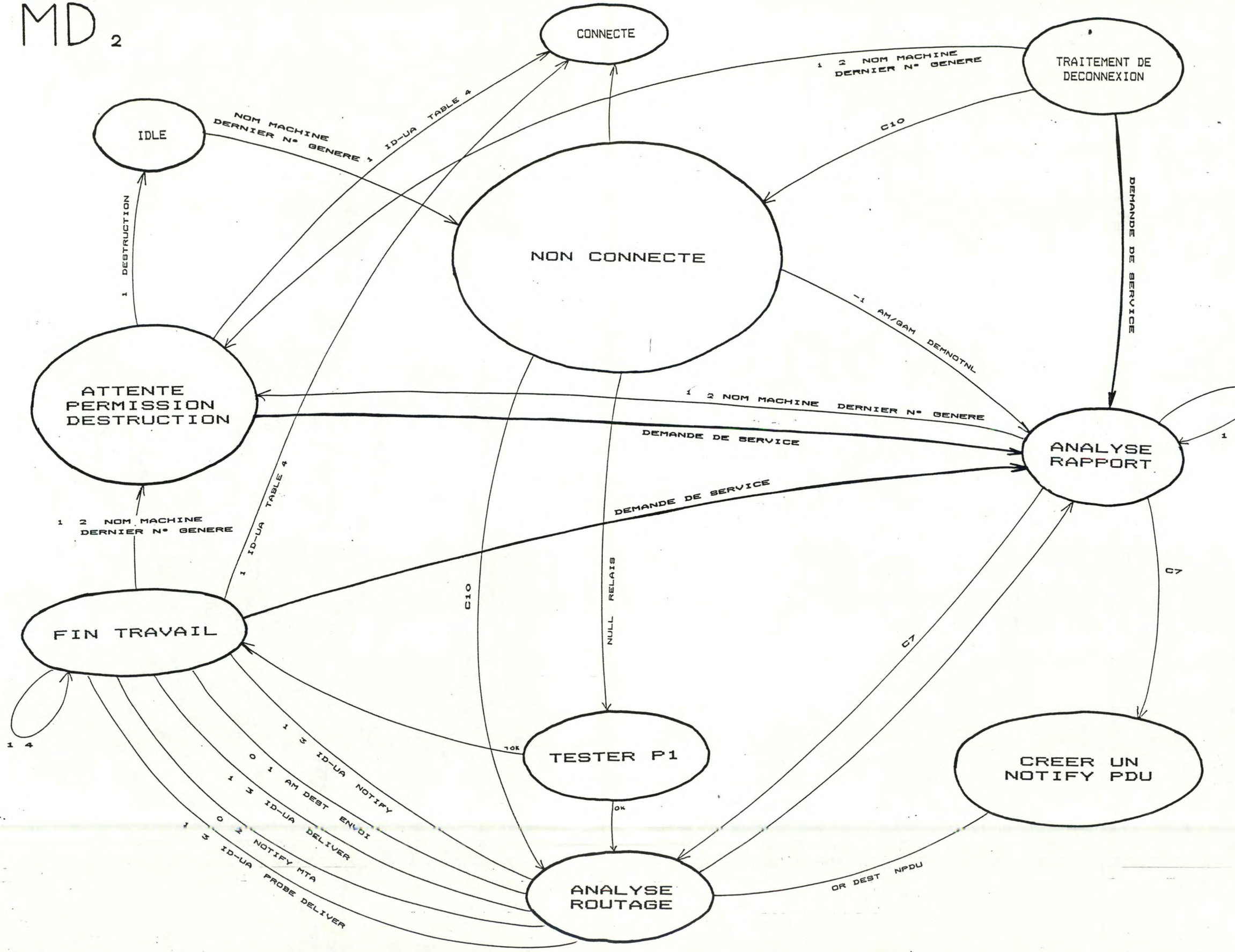
NOM MACHINE DERNIER NUMERO GENERE

Ci : condition voir page suivante

DIAGRAMME D'ETATS

MD2

MD 2


CONNECTE
TRAITEMENT DE DECONNEXION
1 2 NOM MACHINE DERNIER N° GENERE
IDLE
NOM MACHINE DERNIER N° GENERE 1 ID-UA TABLE 4
C10
1 DESTRUCTION
NON CONNECTE
DEMANDE DE SERVICE
-1 AM/GAM DEMNOTNL
ATTENTE PERMISSION DESTRUCTION
1 2 NOM MACHINE DERNIER N° GENERE
DEMANDE DE SERVICE
1 4
ANALYSE RAPPORT
1 ID-UA TABLE 4
1 2 NOM MACHINE DERNIER N° GENERE
DEMANDE DE SERVICE
C10
NULL RELAIS
C7
C7
FIN TRAVAIL
1 4
1 3 ID-UA NOTIFY
0 1 AM DEST ENVOI
1 3 ID-UA DELIVER
0 2 NOTIFY MTA
1 3 ID-UA PROBE DELIVER
TESTER P1
NOK
OK
CREER UN NOTIFY PDU
ANALYSE ROUTAGE
OR DEST NPDU

C1:

si Submit et UA destinataire non local (AM destinataire, paramètre Submit).

C2:

si Probe et UA destinataire non local (AM destinataire, paramètre Probe).

C3:

suite à un Probe, créer un Notify PDU (champ Probe PDU et raison refus).

C4:

si erreurs détectées au routage (soit paramètre Submit soit paramètre Probe plus raison échec.

C5:

si Submit et bon UA (en vue de faire un Notify L)

(paramètre Submit).

C6:

si Probe pour UA local => faire un Notify.

C7:

s'il faut faire une notification (L ou NL).

C8:

si pas de notification (L ou NL) à faire.

C9:

si différé.

C10:

si travail différé

C11:

si erreurs détectées au routage
(champs découpés + raison échec)

DEMANDE DE SERVICE :

```
 0  deliver
 0  local deliver
 0  probe deliver
 0  local probe deliver
 0  notify
 0  demnotnl
-1  AM/GAM  demnotnl
```

XI.4. LE GESTIONNAIRE D'INTERACTION MTA-MTA

11.4.1. INTRODUCTION

Le rôle du gèstionnaire d'interaction MTA-MTA (GI-MTA-MTA) doit être expliqué en repartant de l'architecture figure 10.7. Pour mieux discerner son rôle, il est indispensable d'expliquer la fonction des modules qui lui sont de près ou de loin rattachés. Ceci fait l'objet de la section suivante. Ensuite, par un raisonnement similaire à celui suivi pour la justification du gestionnaire d'interaction UA-MTA (GI-UA-MTA), le GI-MTA-MTA est construit pas à pas. En fin de chapitre (XI.8 et XI.9), une mise au point des tables et messages du GI-MTA-MTA est effectuée.

11.4.2. MODULES INTERAGISSANT AVEC LE GI-MTA-MTA

La figure 11.4.1 présente les modules interagissant avec le GI-MTA-MTA, modules dont les fonctions sont décrites ci-après.

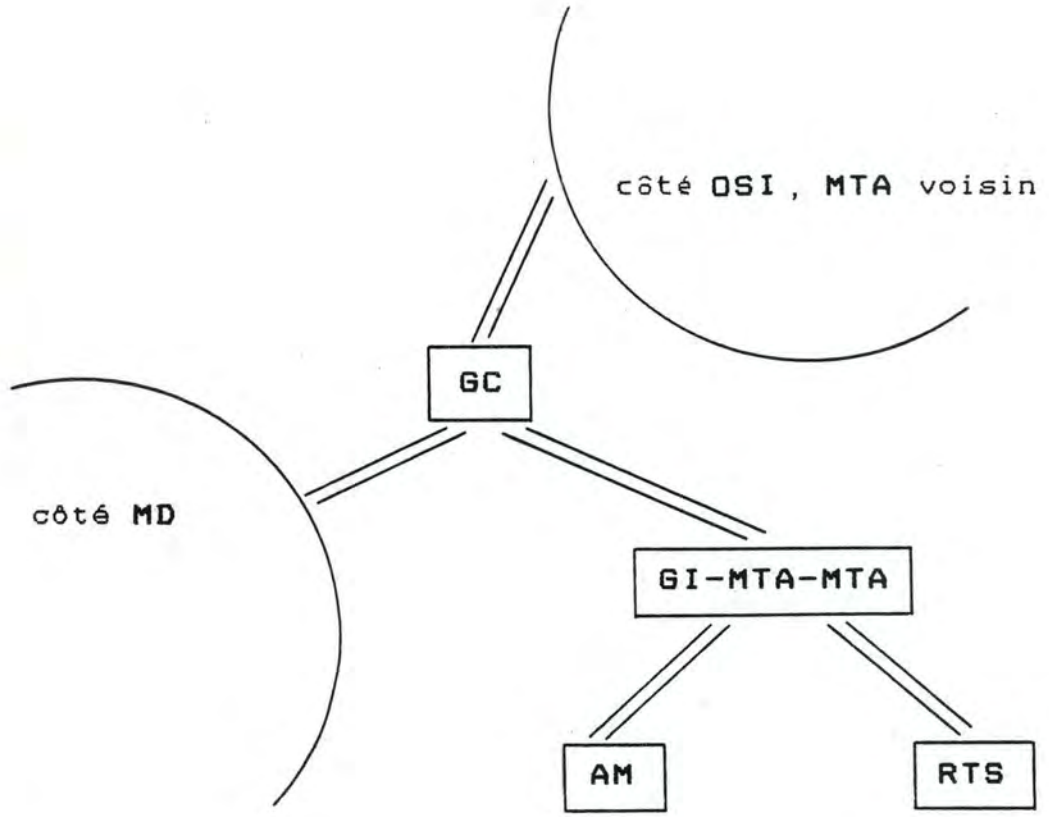


FIGURE 11.4.1 : MODULES INTERAGISSANT AVEC LE GI-MTA-MTA.

11.4.2.1. LE GESTIONNAIRE CENTRAL (GC) : COTE MESSAGE DISPATCHER (MD).

Le seul type de message émis par le MD et destiné à l'AM via le GI-MTA-MTA est une demande d'envoi d'un MPDU vers un MTA voisin. Plus de détails concernant cette interaction sont fournis dans les chapitres précédents concernant le GI-UA-MTA et le MD.

11.4.2.2. L'ASSOCIATION MANAGER (AM).

L'ASSOCIATION MANAGER (AM) peut être envisagé comme étant l'entité qui supervise l'envoi et la réception de MPDU, vers ou en provenance d'un MTA adjacent, via les couches 1 à 6 de OSI. Il dirige logiquement la

conversation d'un MTA avec l'autre sans se préoccuper des couches inférieures du modèle. Pour accomplir une telle tâche, des primitives élémentaires lui suffisent :

OPEN

L'AM ne peut émettre un message vers un MTA voisin que lorsqu'une association a été établie entre les deux AM pairs, i.e. acceptée par l'AM de chaque MTA.

TURN-PLEASE TURN-GIVE.

L'AM ne peut envoyer de message vers un MTA voisin que de manière synchronisée avec ce dernier (les deux savent qui possède le "tour"). Avant de pouvoir parler, l'AM récepteur est obligé de demander le tour et d'attendre la permission en provenance de l'AM émetteur.

TRANSFER-REQ.

L'AM possédant le tour envoie le MPDU vers l'AM correspondant.

TRANSFER-IND.

L'AM sans tour peut à tout moment recevoir un MPDU.

CLOSE.

L'AM qui a le tour a le droit de terminer l'association et l'autre AM ne peut refuser.

11.4.2.3. LE RTS

Le RTS exécute les ordres de l'AM en exploitant les outils offerts par la couche session permettant de synchroniser la conversation, de pouvoir la reprendre en cas d'interruption, etc. A chaque primitive activée par l'AM correspond une suite d'actions à effectuer par le RTS. L'établissement d'association provoque une ouverture de connexion session, une demande de tour-une demande de jetons, un don de tour- un don de jetons et une fin d'association-une terminaison de connexion session. Quant

au transfert et à la réception des données, ils sont exécutés par un algorithme prenant en charge la synchronisation de l'échange. Si le RTS ne parvient pas à effectuer le transfert, il prévient l' AM. Il faut souligner que le RTS peut activer des primitives session de manière interne et autonome (c'est-à-dire indépendamment de toute action de l'AM), mais destinées au bon fonctionnement du RTS lui-même. Ceci est très important pour la suite de ce développement car le RTS peut abandonner brutalement une connexion session alors que l'association reste maintenue. De même, il peut tenter de rétablir une connexion session pendant une association.

Pour bien comprendre le rapport qui peut exister entre l'établissement d'une association et l'établissement d'une connexion session, il faut noter les points suivants:

- la gestion d'une association se situe au niveau AM, tandis que la gestion d'une connexion session se situe au niveau RTS.
- une association ne peut être établie que par la création d'une connexion session.
- pendant une association, plusieurs connexions session peuvent se succéder.
- une association ne peut se terminer que par la terminaison d'une connexion session.

11.4.2.4. LE GESTIONNAIRE CENTRAL (GC) : COTE OSI

Le Gestionnaire Central (GC) joue le rôle de tampon entre le MTA et les processus extérieurs au MTA, en l'occurrence les UA et la couche session du modèle OSI. Ceux-ci ne sont pas unis par un lien de parenté (père-fils) aux processus composant le MTA. Ceci pose problème dans le cadre d'une communication inter-processus sous UNIX version 7. Par conséquent, une des tâches du GC est de permettre

les échanges entre ces différents processus. De plus, en fonction de l'origine et du type de message reçu, il doit assurer sa redistribution.

11.4.3. CONSTRUCTION PROGRESSIVE DU GESTIONNAIRE D'INTERACTION MTA-MTA (GI-MTA-MTA)

14.4.3.1. PRINCIPE GENERAL

Au centre des communications entre MD, AM, RTS et les couches inférieures du modèle OSI, le Gestionnaire d'Interaction MTA-MTA (GI-MTA-MTA) est une station de transit et de redistribution des messages et ce suivant des règles très spécifiques. Son principe est simple. Recevant un message, il détermine le type de celui-ci. En fonction du résultat obtenu et de l'état du MTA à cet instant, il décide des actions à prendre. Celles-ci peuvent être regroupées en deux catégories :

- d'une part, il dirige les messages vers une entité
- d'autre part, il exécute lui-même un travail.

11.4.3.2 HYPOTHESES DE TRAVAIL.

Afin de rendre la construction du GI-MTA-MTA plus compréhensible, dans un premier temps, une hypothèse simplificatrice est posée : le gestionnaire est supposé doté d'un mécanisme de découpe des messages lui fournissant l'origine, la destination et le type du message.

11.4.3.3. PRISE EN COMPTE ET REDISTRIBUTION DES MESSAGES.

Tenant compte de cette hypothèse, du fonctionnement de chaque module AM, RTS, du gestionnaire central (côté couches inférieures de l'OSI et côté MD), et de la position du GI-MTA-MTA au coeur de ceux-ci, il faut commencer par déterminer la discipline de redistribution des messages à travers le GI-MTA-MTA.

Avant toute chose, le flux des messages ne peut s'effectuer de manière arbitraire. Seuls ces trois directions, comme indiqué à la figure 11.4.2, sont autorisées :

MD <---> GI-MTA-MTA <---> AM

AM <---> GI-MTA-MTA <---> RTS

RTS <---> GI-MTA-MTA <---> Couches inférieures de OSI

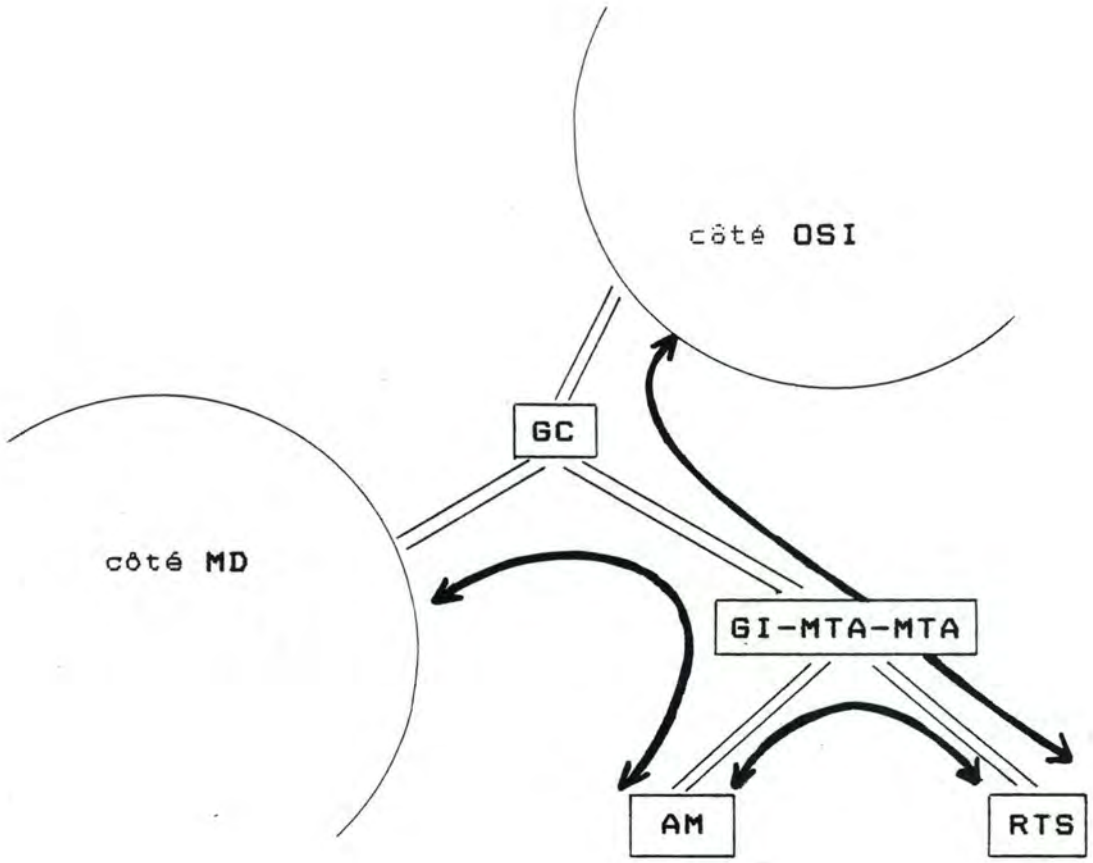


FIGURE 11.4.2 : CHEMINS PASSANT PAR GI-MTA-MTA.

Le GI-MTA-MTA peut dès lors être vu comme un carrefour d'où partent et arrivent des chemins, carrefour réglementé par des feux placés suivant le sens du trafic. Si le feu est vert, le message peut poursuivre son chemin dans la direction permise, sinon, il est contrôlé et traité par le GI-MTA-MTA. La méthode ci-dessous consiste à construire pas-à-pas le GI-MTA-MTA en fonction de tous les événements pouvant se présenter.

Pas 1 : message provenant d'un MD.

Un message provenant d'un MD est une demande d'envoi d'un MPDU vers un MTA voisin. Or l'unité capable de transférer un message vers l'extérieur est l'AM. Comme il a été mentionné précédemment, un transfert ne peut s'effectuer que si une association a été établie entre les deux AM concernés (un AM dans ce MTA et un AM dans le MTA voisin).

En supposant que le GI-MTA-MTA détienne les données lui disant s'il existe ou non une association, il va procéder de la manière suivante.

Si une telle association existe alors il dirige le message vers l'AM ayant ouvert cette association.

Sinon il essaie de créer une telle association. En cas de réussite, le message est dirigé vers l'AM choisi pour cette association. Dans le cas contraire, le GI-MTA-MTA n'étant pas parvenu à faire suivre le message, il le rend à un MD via le GI-UA-MTA en vue de créer une notification de non livraison si elle est nécessaire.

En résumé, le GI-MTA-MTA, recevant une demande d'envoi de MPDU d'un MD, regarde si une association existe entre un AM et un AM du MTA voisin destinataire.

- Si oui, il distribue le message à l'AM.
- Si non, il essaie d'établir une association. En cas d'échec, il renvoie le message vers un MD.

Pour réaliser cette tâche, il doit disposer de données le renseignant sur les associations existant entre un AM local et un AM d'un MTA voisin.

Le principe à appliquer repose donc sur l'existence et la création d'association. Ce concept est détaillé au pas suivant.

pas 2 : Ouverture d'association entre deux AM

A. Principe d'établissement d'association dans la norme X410.

Voici ce que X410 propose.

D'un côté, l'AM désirant entrer en communication avec l'AM voisin émet une demande d'ouverture d'association. Celle-ci n'est effectivement établie que lorsque la confirmation positive lui revient. Entre ces deux événements, l'AM attend.

De l'autre côté, l'AM reçoit une indication lui signalant la demande d'ouverture d'association. Il y répond positivement ou négativement, et retient l'association en cas de réussite.

Ainsi, une association peut se trouver dans trois états possibles:

- association en voie d'établissement du côté émetteur.
- association établie du côté émetteur.
- association établie du côté récepteur.

Le fait de ne pouvoir créer qu'une seule association entre deux MTA fixés (entre deux AM chacun dans un MTA) est posé comme un choix personnel. Cette décision est raisonnable car d'une part, une association repose sur une connexion session et le MTA

ne dispose que d'un nombre fixé de connexions session, et d'autre part, le GI-MTA-MTA n'a à sa disposition qu'un nombre fini de machines, où machine est un couple (AM,RTS), étroitement liés.

B. Données nécessaires au GI-MTA-MTA.

Pour remplir sa fonction, le GI-MTA-MTA doit donc disposer d'une table 10 répertoriant toutes les machines AM-RTS existant avec un bit d'occupation (ou d'existence) révélant l'état activé ou non de cette machine :

nom machine	$\overline{AMi}$	$\underline{AMi}$	$\overline{RTSi}$	$\underline{RTSi}$	dernier n° généré	bit d'occupation

Il dispose également d'une variable qui compte le nombre de connexions session réservées.

Une deuxième table (table 8), lui indiquant les associations déjà créées, est nécessaire puisqu'il doit mémoriser les associations existant avec un AM d'un MTA voisin.

MTAvois (AMdest)	AMorig	$\overline{AMorig}$	$\underline{AMorig}$	bit pipe

C. Choix du traitement de l'ouverture d'association.

Le GI-MTA-MTA doit être capable de réaliser l'établissement d'association. Face aux trois états possibles d'une association, le GI-MTA-MTA va les considérer comme étant un seul état. Ainsi, du point de vue du GI-MTA-MTA, une association établie ou en cours d'établissement est considérée comme étant établie. Cette décision est fondée sur le fait que, dans certaines situations, le GI-MTA-MTA est incapable de déceler le changement d'état de l'AM (suite à un CLOSE, c-à-d une demande de libération d'association).

Une demande d'ouverture d'association découle toujours d'une demande d'envoi d'un MPDU vers un MTA voisin pour lequel aucune communication n'a été installée ou initialisée.

Si une machine est libre, un AM et son RTS sont mis en oeuvre. L'AM commence par exécuter une procédure d'OPEN.

Le GI-MTA-MTA note dans la table 10 que la machine est à présent occupée et inscrit dans sa liste des associations (table 8) celle qui est en cours de création. Il ne faut pas omettre le fait que AM et RTS constituent à eux deux une seule machine. Par conséquent, le GI-MTA-MTA met à jour sa table 9 avec bit connex à 0,

MTAvois (RTSdest)	RTSorig	$\overline{\text{RTSorig}}$	$\underline{\text{RTSorig}}$	bit réservation	bit pipe

ce qui signifie pour lui que la connexion session n'a pas été créée. Cette table 9 est donc à ajouter aux données indispensables au GI-MTA-MTA. Si une nouvelle machine (AM-RTS) est créée, le GI-MTA-MTA distribue alors la demande d'envoi de MPDU (ayant nécessité cet établissement d'association) à cet AM. Sinon, ne pouvant assurer l'envoi de ce MPDU, le GI-MTA-MTA l'envoie à un MD en vue de créer une notification de non livraison.

Le GI-MTA-MTA a à présent terminé de traiter la demande d'envoi de MPDU et peut considérer une demande suivante éventuelle. Si celle-ci est une nouvelle demande d'envoi destinée au même MTA voisin, le GI-MTA-MTA agit comme si l'association était déjà établie et donc transmet le message à l'AM.

Si effectivement la confirmation d'ouverture d'association est positive, alors l'AM va être en mesure d'envoyer ce message.

Sinon, par un mécanisme décrit ci-après, ces requêtes vont être retournées au GI-MTA-MTA.

C'est au niveau du RTS que le caractère non confirmé des demandes d'établissement d'association est surveillé.

D. Déroulement d'une ouverture.

Pour une meilleure vision du déroulement de la procédure d'OPEN, il est bon de suivre l'enchaînement des primitives (cfr. figures 11.4.3 et 11.4.4).

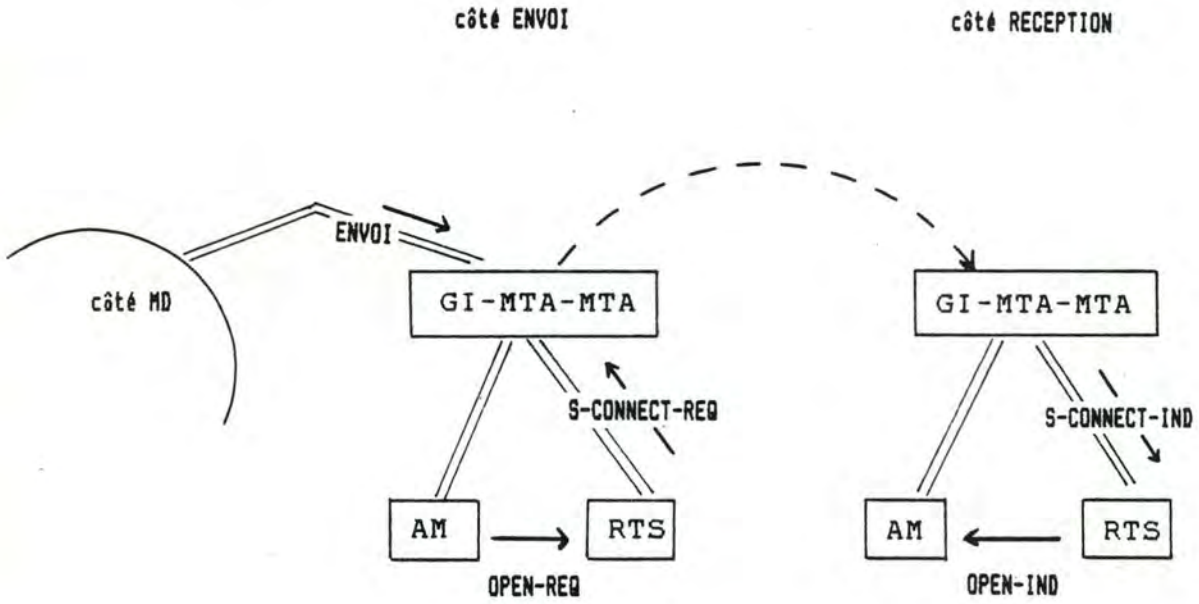


FIGURE 11.4.3 : DEMANDE D'OUVERTURE D'ASSOCIATION.

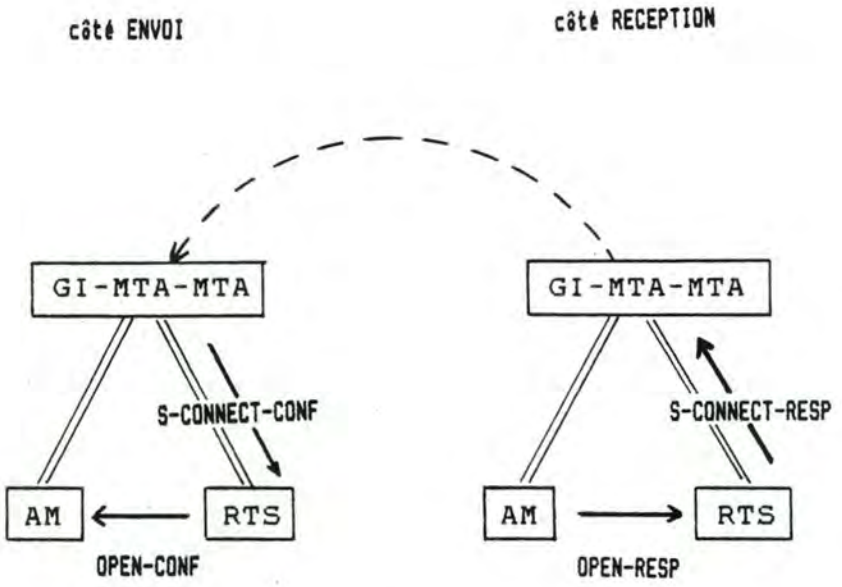


FIGURE 11.4.4 : REPONSE A LA DEMANDE D'OUVERTURE D'ASSOCIATION

En poursuivant cette demande d'ouverture à travers les modules, on constate côté ENVOI que l'AM a émis la primitive d'OPEN-req vers son RTS. Posant momentanément que celui-ci la reçoit, il crée la primitive de S-CONNECT-req et la dirige vers la couche session, via le GI-MTA-MTA.

A ce moment, l'AM ne peut qu'attendre la confirmation de sa demande d'ouverture. Du point de vue du GI-MTA-MTA, il ne se préoccupe plus ni de l'AM ni de l'ouverture d'association.

De l'autre côté à présent (côté RECEPTION), une ouverture d'association est engendrée par une demande provenant du MTA émetteur. Si les deux MTA travaillent correctement, le GI-MTA-MTA va recevoir une primitive S-CONNECT-ind sous la condition suivante :

sa table 9 ne comporte pas d'entrée au nom du MTA voisin émetteur.

Ainsi, il sait qu'une telle primitive est à analyser et ne peut être qu'un S-CONNECT-ind (sinon elle est écartée).

Avant de passer la demande à un RTS, plusieurs exigences doivent être remplies :

- une machine doit être inoccupée.

- une connexion session doit être libre.

S'il peut garantir ces deux conditions, le GI-MTA-MTA active une nouvelle machine, lui signale qu'elle est connectée au "MTA voisin" et relaie la primitive vers le RTS. Ces actions sont marquées dans ses tables de la façon suivante :

- table 10 : bit d'occupation à 1 : car la machine est à présent occupée.

- table 8 : jusqu'à preuve du contraire, l'association est présente.

- table 9 : le bit de connexion est mis à 0 tant que la confirmation positive n'est pas arrivée (cfr. plus loin dans le RTS).

Du côté récepteur comme du côté envoyeur, le GI-MTA-MTA considère donc que l'association existe dès que celle-ci a été demandée. Le problème de l'incertitude de l'établissement réel de cette association est repoussé vers le RTS.

E. Cas d'exception.

L'enchaînement des primitives décrit au point D peut être court-circuité par le GI-MTA-MTA dans certaines situations appelées cas d'exception.

Du côté ENVOI, le GI-MTA-MTA peut générer lui-même la réponse à la demande d'ouverture de connexion session. Il le fait dans le cas où aucune connexion session n'est attribuable à cet instant donné.

Du côté RECEPTION, si une des deux conditions :

- une machine est inoccupée
- une connexion session est libre

n'est pas satisfaite, le GI-MTA-MTA formule lui-même la primitive S-CONNECT-RESP à l'intention du RTS émetteur via la couche session. Il est évident que la demande est refusée pour cause de surcharge.

F. Conclusion.

En conclusion, pour gérer l'établissement d'association, le GI-MTA-MTA dispose de trois tables :

- la table 10 lui fournit le nom des machines libres.
- la table 8 le renseigne sur les associations établies entre un de ses AM et un AM d'un MTA voisin.
- la table 9 lui donne les machines RTS liées aux AM et pour chaque RTS, elle permet de savoir si la connexion session est créée ou non.

Possédant ces données, le GI-MTA-MTA procède de la façon suivante:

côté émetteur

Recevant une demande d'envoi vers un MTA voisin, il regarde si une association est établie entre un de ses AM et un AM de ce MTA voisin.

- Si oui, il passe la demande d'envoi à cet AM.
- si non, il essaie d'établir une association avec cet MTA voisin dans le cas où une machine est encore disponible. Pour ce faire, une nouvelle machine AM-RTS est mise en route. La première action que l'AM va exécuter est un OPEN-REQ. Du point de vue du GI-MTA-MTA, l'association est établie.

côté récepteur

Seule une indication signalant une demande de création de connexion session suite à une demande d'établissement d'association permet de créer une association du côté récepteur. Le GI-MTA-MTA n'autorise cette requête que sous les conditions suivantes :

- aucune association n'existe avec le MTA voisin origine.
- une machine est disponible.
- une connexion session est attribuable.

Dans le cas où ces trois points sont vérifiés, le GI-MTA-MTA considère l'association établie et passe le message au RTS.

Pas 3 : messages touchant à un AM

Le pas 3 complète le pas 2 en ce sens qu'il envisage non seulement les primitives d'établissement d'association mais aussi les autres messages arrivant ou partant de l'AM.

Voici, de manière complète, la stratégie adoptée par le GI-MTA-MTA vis-à-vis de l'AM :

* la procédure d'ouverture d'association est traitée comme décrit ci-avant au pas 2.

* étant supposé créé, les messages à destination ou en provenance de l'AM trouvent tous les feux verts sur leur chemin, c'est-à-dire le GI-MTA-MTA distribue simplement les messages suivant la destination demandée.

  - Un message venant du MD est dirigé vers l'AM relié au bon MTA voisin.

  - Un message de l'AM pour son RTS lui est directement transmis et vice-versa.

  - Un message de relais ou de demande de notification de non-livraison (DEMNOTNL) généré par l'AM est placé dans le pipe vers le GC en vue d'atteindre un MD.

Dans le cadre de cette association, le GI-MTA-MTA n'a plus par la suite qu'à considérer et remplir, comme il se doit, la demande de service émanant de son AM qui est une demande de permission de destruction.

Pas 4 : Création d'une connexion session entre deux RTS.

Un RTS peut effectuer des "transactions" avec un RTS adjacent à une seule condition : avoir ouvert auparavant une connexion session entre eux. Il peut déclencher une telle ouverture dans deux contextes différents : soit suite à un OPEN de l'AM (ce cas nommé OPEN est celui mentionné au pas 2), soit au sein d'une association existante, suite à une décision personnelle (ce cas, portant le nom de RECOVER, se produit lorsque le

RTS veut poursuivre l'envoi d'un message interrompu lors d'une précédente connexion session).

Le GI-MTA-MTA n'a pas à être averti de la différence entre ces deux cas, celle-ci étant traitée par le RTS.

En fonction du circuit des primitives S-CONNECT présenté précédemment aux figures 11.4.3 et 11.4.4, le GI-MTA-MTA doit réagir de la manière décrite ci-après.

Dans sa table 9

MTAvois	RTSorig	$\overline{\text{RTSorig}}$	RTSorig	bit reservation	bit pipe

le GI-MTA-MTA connaît l'existence ou non d'une connexion session et ceci au moyen du bit connex. Bit connex à 1 est le feu vert pour tous les messages se rapportant au RTS, c'est-à-dire que le GI-MTA-MTA distribue simplement les messages ayant pour origine ou pour destinataire le RTS. Par contre, lorsqu'il vaut 0, tous les messages sont stoppés et étudiés par le GI-MTA-MTA. Celui-ci ne tolère que les primitives S-CONNECT.

Suivre à nouveau le cheminement des primitives n'est peut-être pas inutile.

Du côté émetteur, le RTS voulant établir une connexion session avec un RTS destinataire, émet la primitive S-CONNECT-req. Le Bit connex étant à 0, celle-ci est décomposée par le GI-MTA-MTA. Si une connexion session est encore disponible, alors la primitive est dirigée vers les couches inférieures du modèle OSI et le bit de connexion est maintenu à 0 jusqu'à l'arrivée de la preuve de la création de cette connexion session. Si par contre, aucune connexion session n'est disponible, le GI-MTA-MTA met fin à cette demande au moyen d'une primitive créée à cet effet et stipulant qu'aucune connexion session n'est allouable actuellement (S-REFUSE-CONNEX-ind).

Le Bit connex restant à 0, les chemins venant du ou menant au RTS sont contrôlés. S-CONNECT-conf est la seule primitive acceptée. Avant de la passer au RTS, le GI-MTA-MTA met à jour bit connex en fonction des paramètres de cette réponse.

Du côté récepteur, le GI-MTA-MTA assure également la fonction de redistribution des messages si la connexion session est établie. De la même manière que du côté envoyeur, la présence dans la table 9 d'une entrée RTSvois et la valeur du bit indiquant l'existence d'une connexion, dévoilent au GI-MTA-MTA s'il doit examiner le message. Ainsi, le GI-MTA-MTA va analyser la primitive S-CONNECT-ind et l'envoyer vers le RTS.

Le GI-MTA-MTA attend la confirmation du RTS et ceci afin d'éviter de distribuer du travail à cette machine qui pourrait être détruite peu de temps après pour cause de réponse d'OPEN négatif. Ceci n'est pas contradictoire avec la justification décrite du côté envoyeur affirmant le groupement du cas association "en cours" ou "créée". En effet, du côté récepteur, l'exécution du S-CONNECT-ind est très rapide et reste locale, le temps d'attente devient donc négligeable.

Le gestionnaire va contrôler le champ du S-CONNECT-conf, révélant la réussite ou non de l'opération. Il est à noter que pour le cas OPEN du S-CONNECT-ind, il faut compléter ce pas 4 avec le raisonnement développé au pas 2.

Pas 5 : Messages touchant à un RTS.

Le pas 5 étend le pas 4 à tous les messages en provenance ou à destination du RTS.

Le GI-MTA-MTA va réagir de la façon décrite ci-après :

- la procédure d'ouverture de connexion session est celle considérée au pas 4.

- le RTS supposé créé et connecté, les messages à destination ou en provenance du RTS sont tous placés sur le chemin demandé et permis. Seules les demandes de service sont couvertes par le GI-MTA-MTA.

Le GI-MTA-MTA n'est pas en mesure de détecter les fins de connexion session. Celles-ci étant révélées au niveau du RTS, c'est lui qui a la responsabilité d'en prévenir le GI-MTA-MTA pour la mise à jour de ses tables.

11.4.3.4. RECUPERATION DES MESSAGES EN CAS DE DESYNCHRONISATION.

Les problèmes d'aiguillage à présent réglés, le GI-MTA-MTA ne peut être entièrement efficace que s'il récupère les messages transmis trop vite à un AM et ceci toujours pour cause de désynchronisation lors d'une fin d'association. Le fait de ne pas savoir quel est l'état exact de l'AM ne peut malheureusement pas non plus être évité de ce côté-ci de l'architecture. Voici un exemple de discordance temporelle entre AM et GI-MTA-MTA : au moment où le GI-MTA-MTA reçoit une demande d'envoi vers un MTA voisin et la place tout naturellement dans le pipe de l'AM associé à ce MTA voisin, cet AM décide de terminer son association. Ne pouvant revenir sur un tel ordre, la demande d'envoi ne peut plus être satisfaite.

L'exemple suivant renseigne un autre type de message devant être traité après un CLOSE : un AM possédant le tour et n'ayant plus de travail en attente prend la décision de stopper son association. Si on imagine que pendant ce temps, le RTS tente d'envoyer un message et que cet envoi échoue, le RTS doit prévenir l'AM de cet échec et lui rendre le MPDU en vue de demander la création d'une notification de non livraison. Cet ordre doit pouvoir être dirigé vers un MD malgré la fin de l'association.

Ce problème déjà très difficilement détecté pour le GI-UA-MTA, s'amplifie hélas pour le GI-MTA-MTA. En effet, il ne faut pas tomber dans le piège de vouloir réutiliser entièrement le travail déjà réalisé côté MD car ici une machine est un ensemble de deux modules. Ainsi une fois c'est le module AM qui détecte le premier la fin de l'association, une autre fois, c'est le RTS. Pour envenimer le tout, la communication entre AM - RTS, et celle entre RTS - les couches inférieures du modèle OSI ne sont pas toujours interrompues au même moment.

Pour régler de telles difficultés, deux décisions importantes sont prises. La première consiste à concentrer toute l'intelligence sur un module de la machine (AM - RTS), en l'occurrence le RTS. Ceci se défend d'autant mieux que dans le cas "OPEN récepteur", si le RTS refuse la connexion session, il est stupide d'en avertir l'AM. Le circuit normal des primitives est dès lors interrompu et repris par le RTS.

Par conséquent, le premier choix consiste à initialiser la procédure de destruction de machine par le RTS.

La deuxième décision peut s'intituler "mouvement aller-retour en quatre temps". Ces quatre mouvements de messages sont indispensables à la resynchronisation des modules et à la récupération totale des messages ne pouvant être traités par l'AM. L'information contenue dans les primitives S-RELEASE et CLOSE, informant respectivement RTS et AM, doit être communiquée au GI-MTA-MTA. En plus de signaler au GI-MTA-MTA la fin de l'association, elle le contraint à ne plus rien envoyer vers cette machine. C'est pourquoi ce signal se nomme [1 | boucher pipe] et les tables 8 et 9 contiennent par ligne, un champ intitulé bit pipe, fermant ou non les pipes vers la machine AM - RTS.

Dans un deuxième temps, en envoyant un message [1 pipe bouché] le GI-MTA-MTA avertit la machine qu'il a bien reçu ce signal. Du point de vue (AM - RTS), la réception de ce message signifie que plus rien ne suit.

D'où la troisième étape consiste à remonter chaque message trop vite envoyé à la machine AM vers le GI-MTA-MTA jusqu'à la réception du signal 1 pipe bouché indiquant la fin d'écriture dans le pipe AM. Pour que le GI-MTA-MTA soit en mesure de récupérer tous ces messages, la machine termine cette troisième étape en envoyant un signal 1 demande destruction signalant la fin de relecture.

En procédant de la sorte, aucun message n'est perdu . A ce moment, le quatrième temps coule de source. Le GI-MTA-MTA envoie un message permettant au RTS de se détruire et met à jour les données concernant cette machine.

En conclusion, cette construction progressive donne le squelette du GI-MTA-MTA.

L'algorithme placé en annexe 6 permet d'achever la présentation du GI-MTA-MTA.

11.4.3.5. REMISE EN COMPTE DE L'HYPOTHESE POSEE.

A présent, il ne reste plus qu'à lever l'hypothèse posée en début de chapitre affirmant que le GI-MTA-MTA connaît la provenance, la destination et la nature des messages. Ceci est immédiat si le GI-MTA-MTA a à sa disposition une procédure de découpe de messages et la liste de tous les messages pouvant lui arriver, liste fournie au point XI.9. Messages arrivant au GI-MTA-MTA

11.4.4. CONCLUSION.

Le GI-MTA-MTA est la station qui centralise et redistribue les messages entre, d'une part, les machines (AM-RTS) et d'autre part, soit un MD, soit les couches inférieures du modèle OSI. Pour réaliser cette fonction, il gère les établissements d'association et les ouvertures de connexion session. C'est également lui qui active et désactive les machines AM-RTS et assure la synchronisation entre tous ces modules.

La conception des modules AM et RTS fait l'objet des paragraphes suivants.

XI.5.L'ASSOCIATION MANAGER

11.5.1. INTRODUCTION.

La façon de procéder dans le cas de l'Association Manager (AM) va se rapprocher très fortement de ce qui a déjà été fait pour le MD.

Pour spécifier entièrement l'AM, il faut disposer de plusieurs outils dont le diagramme d'états présenté en fin de cette section XI.5. L'algorithme de l'AM est le complément indispensable pour exprimer les enchaînements. Il est placé dans l'annexe 7.

11.5.2. LE DIAGRAMME D'ETATS.

Ce diagramme d'états fournit au lecteur l'ensemble des états que peut atteindre un AM qui exécute les demandes venant du MD ou du RTS. Ces différents états sont accompagnés de messages permettant de définir entre autres le moment où l'AM passe d'un état à un autre (cfr fin de paragraphe). Les états définis pour l'AM sont au nombre de neuf : IDLE, ATTENTE DE PERMISSION DE DESTRUCTION, ATTENTE DE FIN D'ASSOCIATION, ETABLISSEMENT ASSOCIATION, ASSOCIATION EXISTE AVEC TOUR, ASSOCIATION EXISTE SANS TOUR, ATTENTE ASSOCIATION, GESTION DES ECHECS D'ETABLISSEMENT DES ASSOCIATIONS, FIN DU TRAVAIL.

Lorsque le message est reçu ou envoyé par l'AM, le premier champ désigne respectivement l'origine ou la destination de ce message.

Si le premier champ est 0, cela indique que le message vient du gestionnaire central ou qu'il lui est destiné.

Si le premier champ est 1, cela désigne une demande ou une réponse de service du GI-MTA-MTA.

Si le premier champ est 2, c'est que le message véhicule une primitive à destination ou en provenance du RTS créé de pair avec l'AM considéré.

11.5.3. L'ALGORITHME.

L'algorithme de l'AM ne fait pas de distinction entre les états stables et transitoires comme définis au paragraphe 11.3.2.1. Toutes les actions à effectuer dans chaque état, lorsqu'on reçoit un message, sont regroupées dans un morceau de l'algorithme.

11.5.3.1. IDLE.

Lorsque l'AM est dans l'état IDLE, il doit vider le pipe qui contient les messages venant du GI-MTA-MTA et mettre ces demandes dans une file d'attente. Lorsque cette file d'attente n'est plus vide, l'AM prend le premier message qui s'y trouve afin de le traiter.

Le premier message que l'AM doit recevoir lorsqu'il est dans l'état IDLE, c'est | 1 | MTA voisin | qui lui indique l'adresse du MTA pour lequel il gère dès à présent les associations.

Lorsque l'AM sait avec quel MTA il est en relation, il peut recevoir deux types de messages. Le premier est un message venant du MD à destination du MTA voisin. Ce message se présente sous la forme | 0 | AM dest | envoi APDU |. Le deuxième est un message | 2 | open-ind | venant du RTS et demandant l'ouverture d'une association avec un MTA voisin. Une fois ceci terminé, l'AM passe dans l'état ETABLISSEMENT-ASSOCIATION.

11.5.3.2. ETABLISSEMENT-ASSOCIATION.

Lorsque l'AM passe dans cet état, il doit tout d'abord se rendre compte s'il s'agit d'une demande de transfert transmise par le MD ou d'une demande d'association venant d'un MTA voisin.

Si c'est un envoi, l'AM doit envoyer une demande d'ouverture d'association vers le MTA voisin sur la route de l'UA destinataire. Cette demande est reçue par l'AM destinataire grâce au message | 2 | open-ind | en provenance du RTS . L'AM crée ensuite la primitive de réponse à l'open et l'envoie vers son RTS grâce au message | 2 | open-

resp}. Si cette réponse véhicule un refus, l'AM passe dans l'état FIN-TRAVAIL. Si, par contre, la réponse à l'open est positive, l'AM passe dans l'état ASSOCIATION-TOUR ou ASSOCIATION-SANS-TOUR selon que l'AM origine donne ou garde le tour.

11.5.3.3. ATTENTE-ASSOCIATION.

Lorsque l'AM aboutit à cet état, c'est parce qu'il est en attente d'une confirmation d'ouverture d'association. Dès lors, l'AM doit vider son pipe et placer ce qu'il contient en fin de file d'attente et ceci tant qu'un message n'est pas reçu. Lorsque la file d'attente contient au moins une demande, l'AM prend dans cette file la première demande qui a le RTS comme origine c'est-à-dire la première demande contenant un '2' dans le premier champ. Si on ne trouve pas une telle demande, l'AM retourne au début de l'algorithme gérant l'état ATTENTE-ASSOCIATION.

Dès que l'AM trouve un message provenant du RTS, si ce n'est pas | 2 | open-conf | , il s'agit d'une erreur, il l'écarte et continue sa recherche jusqu'à trouver la confirmation de l'open-req envoyé plus tôt.

Quand l'AM trouve enfin un | 2 | open-conf | dans sa file d'attente, il découpe cette primitive reçue pour l'analyser. Si l'ouverture de l'association est acceptée et si l'AM avait demandé de garder le tour, il passe dans l'état ASSOCIATION-TOUR. Si l'AM destinataire a le tour, l'AM origine retourne dans l'état ASSOCIATION-SANS-TOUR. Par contre, si le MTA voisin refuse l'association, l'AM passe dans l'état FIN-TRAVAIL.

11.5.3.4. ASSOCIATION-TOUR.

Quand il atteint cet état, l'AM doit vérifier si l'ouverture de l'association a été déclenchée pour un envoi. Si c'est le cas, l'AM crée la primitive et demande le transfert du message à l'aide de la demande | 2 | transfer-req | qu'il transmet au RTS.

Lorsqu'il arrive à cet état suite à une demande d'ouverture venant de l'AM pair ou lorsqu'il a terminé son transfert vers le MTA voisin, l'AM vide son pipe, place les demandes dans la file d'attente et commence à traiter la première si elle existe. Cette demande peut être d'un type parmi les suivants :

| 0 | AM dest | envoi | MPDU |

Cette demande signale à l'AM que le MD local souhaite envoyer un MPDU vers le MTA voisin. Ceci provoque la création et l'envoi vers le RTS local d'une primitive transfer. Après la gestion de cette demande, l'AM retourne vider son pipe et gérer ce qui se trouve dans sa file d'attente.

| 2 | turn-please-ind |

L'arrivée de ce message indique à l'AM que l'AM pair désire obtenir le tour. Ceci provoque l'exécution de la procédure DEM-TOUR expliquée en 11.5.4.1. Une fois terminée, cette procédure dirige l'AM vers l'état ASSOCIATION-SANS-TOUR si l'AM donne le tour ou vers la gestion du pipe et de la file d'attente s'il le garde.

| 2 | exception-ind |

Ce message signale à l'AM que le RTS ne peut pas transférer correctement l'MPDU que l'AM lui a

donné. Dans ce cas, l'AM envoie vers un MD un message | -1 | AM | Demnotnl | APDU | signalant qu'un problème s'est posé lors de la transmission.

Lorsqu'il a géré l'arrivée du message | 2 | exception-ind |, l'AM retourne à son pipe et sa file d'attente. Lorsqu'il vide son pipe, il place ce qui s'y trouve dans la file d'attente et s'il s'aperçoit que cette file d'attente est vide, il va vérifier si le tour n'a pas été demandé auparavant par son AM pair. Si c'est le cas, l'AM crée et envoie une primitive turn-give-req vers le RTS et retourne dans l'état ASSOCIATION-SANS-TOUR. Si, par contre, l'AM pair n'a pas demandé le tour, comme rien ne se trouve dans la file d'attente, l'AM va transmettre vers le gestionnaire un message | 2 | close-req | indiquant au RTS qu'il désire clôturer l'association. Après cet envoi, l'AM passe en ATTENTE-FIN-ASSOCIATION.

11.5.3.5. ASSOCIATION-SANS-TOUR.

L'AM peut atteindre cet état à la suite d'une demande d'envoi exigée par le MD. Lorsqu'il s'aperçoit qu'il doit envoyer un MPDU vers le MTA voisin, L'AM doit ouvrir une association. Si, dans la primitive open-req, l'AM donne le tour à l'AM destinataire, il se retrouve dans l'état ASSOCIATION-SANS-TOUR alors qu'il doit effectuer un transfert. Si c'est le cas, l'AM remet cette demande d'envoi à sa place, c'est-à-dire en tête de la file d'attente.

Lorsqu'il a remis cette demande dans la file d'attente ou lorsqu'il atteint cet état par la réception d'un open-ind ne lui donnant pas le tour, l'AM doit vider son pipe, mettre ce qui s'y trouve dans la file d'attente et prendre le premier message de la file. Ce message peut être d'un des types suivants :

| 0 | AM dest | envoi | MPDU | :

Ceci indique à l'AM que le MD voudrait bien envoyer un MPDU vers un MTA voisin. Si l'AM a déjà fait une demande de tour dont il n'a pas reçu réponse, il replace la demande à sa place dans la file d'attente. Une fois ceci terminé, l'AM vide son pipe à nouveau, place les demandes qui s'y trouvent en fin de file d'attente et commence à traiter la demande suivant l'envoi si cette demande existe. Si non, l'AM exécute les instructions de fin de gestion de cet état expliquées ci-dessous. Lorsque l'AM s'aperçoit qu'il n'a pas demandé le tour, il exécute la procédure TOUR décrite en 11.5.4.2. Une fois cette procédure terminée, l'AM retourne au pipe, le vide, met les demandes en fin de file d'attente et prend la demande suivant celle qu'il vient de gérer. Si une demande existe, l'AM la traite, si pas, il effectue les opérations de fin détaillées ci-dessous.

| 2 | transfer-ind | :

Ce message indique à l'AM qu'un MPDU lui est transmis par son AM pair par l'intermédiaire du RTS. L'AM envoie alors le message | 0 | null | relais | MPDU | à destination du MD pour que celui-ci puisse livrer l'MPDU à un UA local ou effectuer le relais. Une fois ceci terminé, l'AM vide son pipe, met les demandes en file d'attente et passe à la demande suivante dans la file. Si cette demande existe, l'AM la traite. Si pas, il passe à la fin de la gestion de cet état.

| 2 | exception-ind | :

La réception de ce message signale à l'AM que le RTS a trouvé un problème. L'AM envoie alors le message | 0 | -1 | AM | Demnotnl | MPDU | qui indique au MD un problème au niveau de la transmission de l'MPDU renvoyé en paramètre. Ceci terminé, l'AM retourne à son pipe, le vide en fin de la file d'attente et passe

à la demande suivante. Si elle existe, l'AM s'en occupe. Si on est en fin de file d'attente, l'AM exécute les opérations de fin de gestion.

| 2 | turn-give-ind | :

Ce message informe l'AM que son AM pair lui cède le tour. Dans ce cas, l'AM passe directement dans l'état ASSOCIATION-TOUR.

| 2 | close-ind | :

L'arrivée de cette demande indique à l'AM que son AM pair désire terminer l'association. Dans ce cas, l'AM crée et envoie la primitive close-resp dans le message 2 close-resp vers le RTS. Après cet envoi, l'AM passe dans l'état FIN-TRAVAIL.

Ceci termine la liste des messages que l'AM, dans l'état décrit, peut aller puiser dans sa file d'attente. Dans certains cas, l'AM doit effectuer quelques opérations de fin de gestion qui visent à demander le tour pour pouvoir envoyer des messages ou bien obtenir le tour pour envoyer la demande de fermeture de l'association. C'est ainsi qu'après un certain délai, l'AM arrivé en fin de file d'attente peut redemander le tour et à nouveau vider le pipe, remplir la file d'attente et analyser une nouvelle fois la première demande de cette file si elle existe.

11.5.3.6. ATTENTE-FIN-ASSOCIATION.

Lorsque l'AM arrive dans cet état, c'est qu'il a demandé la fermeture de l'association existante. L'AM doit alors vider son pipe, placer ce qui s'y trouve dans la file d'attente et prendre la première demande de cette file.

L'AM peut alors s'attendre à recevoir deux types de messages :

2	exception-ind

Ce message signale à l'AM qu'un problème est survenu lors de la transmission du message vers un MTA voisin. L'AM construit alors le message

0	-1	AM
Demnotnl	MPDU	

et le replace dans la file d'attente à la place de l'exception-ind pour pouvoir gérer par la suite ce problème de transmission. Ceci terminé, l'AM recommence à vider le pipe, mettre les demandes dans la file d'attente et passe à la demande suivante pour la traiter.

2	close-conf

La réception de ce message indique à l'AM que l'AM pair est conscient de la fermeture de l'association. L'association est donc clôturée proprement et l'AM passe alors dans l'état FIN-TRAVAIL.

Si l'AM trouve dans sa file d'attente un message venant du RTS autre qu'un des deux cités ci-dessus, il l'écarte pour cause d'erreur, vide son pipe, remplit la file d'attente et gère la demande suivante.

11.5.3.7. FIN-TRAVAIL.

Cet état est celui par lequel passe tout AM qui a vu son association terminée proprement.

Une fois dans cet état, l'AM vide son pipe, remplit la file d'attente et prend la première demande si elle

existe. Si cette demande est | 1 | pipe bouché | , l'AM peut alors envoyer la demande de destruction puisqu'il est sûr que plus rien ne va venir du gestionnaire. Il passe alors dans l'état ATTENTE-PERMISSION-DESTRUCTION. Si ce qu'il va lire dans la file d'attente est une autre demande, l'AM renvoie le message reçu du gestionnaire vers ce même gestionnaire mais précédé de deux nouveaux champs | 1 | 4 |. Ensuite, l'AM repasse dans l'état FIN-TRAVAIL.

Si, par contre, l'AM se trouve devant une file vide, il retourne dans l'état FIN-TRAVAIL en attendant un message.

11.5.3.8. ATTENTE-PERMISSION-DESTRUCTION.

Lorsque l'AM a envoyé son message | 1 | demande de destruction | , il attend la permission de destruction. Dans cet état, l'AM vide le pipe jusqu'à trouver le message 1 destruction, qui le fait passer dans l'état IDLE.

11.5.4. LES PROCEDURES.

11.5.4.1. DEM-TOUR.

Lorsqu'il reçoit une demande de tour, l'AM peut, en fonction de la priorité véhiculée dans la demande, décider de garder ou de donner le tour.

Si l'AM décide de céder le tour, il crée et envoie la primitive turn-give-req dans le message | 2 | turn-give-req | à destination du RTS. Si l'AM choisit de conserver le tour, il garde en mémoire le fait que son AM pair lui a demandé le tour, mais il continue son propre travail.

11.5.4.2. TOUR.

Cette procédure enregistre le fait que l'AM a demandé le tour. Ensuite, elle crée et envoie vers le RTS par l'intermédiaire du GI-MTA-MTA la demande de tour à l'autre AM grâce au message | 2 | turn-please-req |

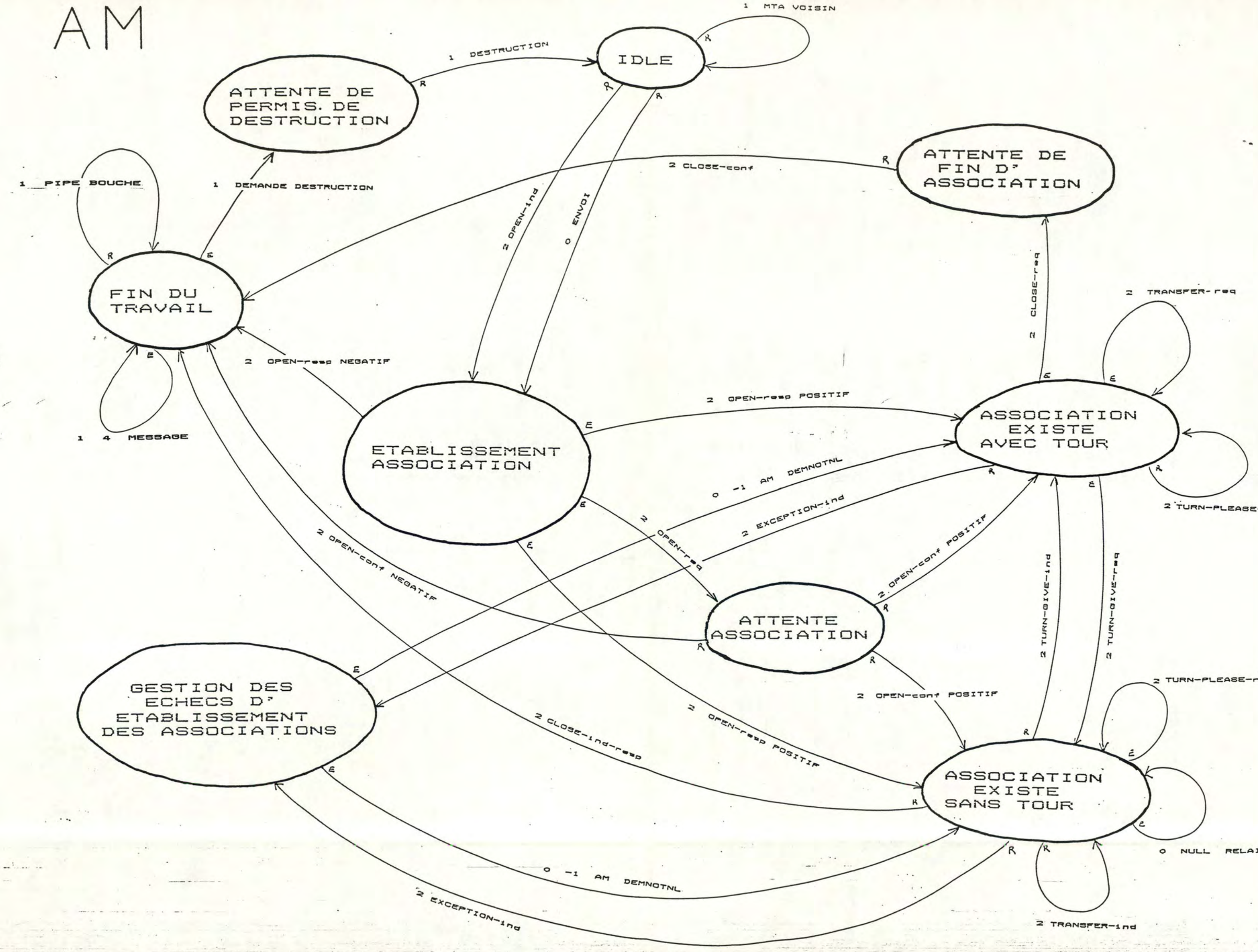
11.5.5. CONCLUSION

Maintenant que le fonctionnement de l'AM et du GI-MTA-MTA sont connus, il reste à terminer en décrivant le mécanisme du RTS. Une fois ceci achevé, les opérations de relais entre deux MTA pourront être mieux comprises.

DIAGRAMME D'ETATS

AM

AM


1 MTA VOISIN
IDLE
1 DESTRUCTION
ATTENTE DE PERMIS. DE DESTRUCTION
1 DEMANDE DESTRUCTION
1 PIPE BOUCHE
2 CLOSE-conf
ATTENTE DE FIN D' ASSOCIATION
2 OPEN-ind
0 ENVOI
FIN DU TRAVAIL
2 CLOSE-req
2 TRANSFER-req
2 OPEN-resp NEGATIF
2 OPEN-resp POSITIF
1 4 MESSAGE
ETABLISSEMENT ASSOCIATION
ASSOCIATION EXISTE AVEC TOUR
0 -1 AM DEMNOTNL
2 EXCEPTION-ind
2 OPEN-req
2 TURN-PLEASE-ind
2 OPEN-conf NEGATIF
2 OPEN-conf POSITIF
2 TURN-GIVE-ind
2 TURN-GIVE-req
ATTENTE ASSOCIATION
2 TURN-PLEASE-req
2 OPEN-conf POSITIF
GESTION DES ECHECS D' ETABLISSEMENT DES ASSOCIATIONS
2 CLOSE-ind-resp
2 OPEN-resp POSITIF
ASSOCIATION EXISTE SANS TOUR
0 NULL RELAIS
0 -1 AM DEMNOTNL
2 EXCEPTION-ind
2 TRANSFER-ind

XI.6. LE RELIABLE TRANSFER SERVER

11.6.1. INTRODUCTION

Le RTS (Reliable Transfer Server) comme présenté dans la norme X410 est le serveur assurant un transfert fiable entre deux MTA adjacents. Il assure cette tâche en utilisant les services des couches 6 et 5 du modèle OSI, ou encore la couche présentation et la couche session. En examinant plus en détail les recommandations concernant la messagerie électronique, il apparaît que le niveau application possède sa propre syntaxe grâce au choix de x409, rendant ainsi la couche présentation transparente. Par conséquent, l'objet de ce chapitre va consister en un premier temps à comprendre comment le RTS fait usage des services de la couche session.

Ensuite, le RTS étant placé dans un contexte bien particulier, en l'occurrence l'architecture parallèle, il est indispensable de le modifier ou plus exactement de le compléter en conséquence. Le relevé des messages arrivant au RTS, ainsi que de ses tables va clôturer ce chapitre consacré au RTS.

11.6.2 LE RTS, UTILISATEUR DES SERVICES DE LA COUCHE SESSION.

Sous-couche de la couche application, la plus basse dans le MTA, le RTS est le module utilisant les services des couches inférieures ou plus exactement de la couche session. Pour la clarté de ce rapport, il est souhaitable

de retracer dans les grandes lignes le fondement de la couche session [SES 84].

11.6.2.1. SERVICES OFFERTS PAR LA COUCHE SESSION.

Le principe de la couche session est de fournir un mécanisme de gestion du dialogue, c'est-à-dire un moyen pour transférer les données de façon organisée et synchronisée. Pour accomplir une telle tâche un ensemble de services est défini. Ceux-ci permettent à l'utilisateur de sélectionner le degré de contrôle et de synchronisation. Ces services sont définis sans savoir complètement les besoins et les désirs de l'utilisateur étant donné que cette gestion du dialogue est fonction de sa "sémantique". Ceci explique pourquoi la couche session peut être interprétée comme étant une boîte à outils. Son utilisateur, disposant de ces outils, organise son dialogue.

Il est nécessaire à présent de considérer successivement les services qu'offre la couche session.

1. Le mécanisme du jeton.

Un jeton est un attribut de la connexion session, assigné dynamiquement à un utilisateur du service session, à un instant donné, et ceci uniquement dans le cas où ce jeton est disponible. En effet, un jeton peut être disponible ou non. Ce fait est déterminé lors de l'établissement de la connexion session et ne peut être modifié pendant toute la durée de la connexion. Un jeton a pour effet de donner à celui qui le possède le droit d'utiliser certains services. Ainsi donc, dans le cas d'indisponibilité d'un jeton, les primitives que celui-ci permet d'invoquer ne peuvent être activées par les

utilisateurs. Plusieurs types de jetons sont définis au niveau session :

* le jeton de données

il permet à celui qui le possède de transférer des données

* le jeton de synchronisation mineure

accompagné du jeton de données, il autorise son propriétaire à poser des points de synchronisation mineure (cfr ci-après)

* le jeton de synchronisation majeure et activité

l'utilisateur qui le possède ainsi que le jeton de données et de synchronisation mineure a le droit de poser des points de synchronisation majeure (cfr ci-après)

* le jeton de terminaison

associé au jeton de données, il rend capable de terminer la connexion session

L'assignation du jeton à un utilisateur de la session autorise seulement celui-ci à employer les services correspondant à ce jeton. Seul le "transfert" du jeton vers l'autre utilisateur peut lui donner le droit d'activer à son tour ces services. Ce "transfert" aura lieu après une demande du jeton de la part de l'utilisateur ne le possédant pas, ou à la suite d'un don spontané de la part de l' utilisateur les possédant.

2. Le mécanisme du point de synchronisation.

Le concept du point de synchronisation est un moyen de régler la synchronisation des données à tranférer entre deux utilisateurs. Deux types de points de synchronisation peuvent être rencontrés : le point de synchronisation mineure et le point de synchronisation majeure. Le point de synchronisation majeure découpe la communication en unités de dialogue (cfr ci-après). Le point de synchronisation mineure structure l'échange de données au sein d'une unité de dialogue (cfr ci-après). Il est à noter que la sémantique de ces points n'est connue que de l'utilisateur et est transparente pour la couche session.

L'utilité de ces points est de permettre et faciliter la resynchronisation, i.e. de reprendre le dialogue à partir d'un point de synchronisation.

3. Le mécanisme d'unité du dialogue.

Une unité de dialogue est définie au moyen du service de point de synchronisation majeure. Ainsi, une unité de dialogue est constituée de tous les éléments de communication échangés entre deux points de synchronisation majeure. Ce concept garantit aux données contenues dans une unité de dialogue d'être isolées de toute communication précédent ou suivant cette unité.

4. Le concept d'activité.

Une activité, constituée d'une ou plusieurs unités de dialogue, représente une unité logique de travail. Grâce à ce concept, les utilisateurs ont la possibilité de

fractionner le dialogue en parties suivant une certaine logique.

Ces services ou outils sont regroupés logiquement en unités fonctionnelles. Plusieurs unités fonctionnelles existent avec un assortiment différents d'outils. Choisir une unité fonctionnelle, c'est choisir les outils la formant.

11.6.2.2. DEROULEMENT D'UNE CONNEXION SESSION.

Ces définitions posées, il est indispensable de suivre le déroulement d'une connexion session. Elle se passe en trois phases.

1. établissement de la connexion.

les utilisateurs désirant dialoguer doivent établir une connexion session. Ils veulent la créer soit pour commencer une nouvelle activité, soit pour reprendre une éventuelle activité interrompue. Au cours de cette ouverture, ils négocient le choix des unités fonctionnelles.

2. transfert.

sur base d'une connexion session créée, on entre dans la phase transfert de données. C'est ici qu'interviennent

éventuellement les concepts de jetons, de points de synchronisation, de resynchronisation, d'activité... .

3. clôture.

la phase trois consiste à libérer la connexion session. La clôture peut être brutale ou négociée. Dans le cas négocié (RELEASE), les deux utilisateurs coopèrent pour terminer la session sans perte de données. Dans l'autre cas, la connexion est subitement suspendue (ABORT).

11.6.2.3. EXEMPLE.

L'application sur un exemple permet de faciliter la compréhension de tous ces concepts :

un utilisateur de la couche session désire envoyer le contenu d'un livre à un autre utilisateur.

Dès lors, dans un premier temps, il demande l'établissement d'une connexion session. Au cours de la négociation qu'elle permet, les deux utilisateurs choisissent la (ou les) boîte(s) à outils. Cette négociation suit le principe : l'un propose (émetteur de la demande), l'autre dispose (celui répondant à cette demande).

Il est supposé pour la suite de l'exemple que la connexion session est établie et que la boîte à outils comprend tous les services décrits ci-avant.

Ensuite vient la phase de transfert. Elle peut s'organiser de la manière suivante :

le livre constitue l'unité logique de travail. Ainsi l'envoi du contenu fait l'objet d'une activité;

un chapitre définit une unité de dialogue. Donc, après l'envoi de chaque chapitre est posé un point de synchronisation majeure. Chaque chapitre est indépendant des autres;

un point de synchronisation mineure est placé en vue de séparer chaque page

Si l'utilisateur émetteur le livre constate un mauvais fonctionnement de l'envoi, il peut resynchroniser le dialogue, et cela à chaque marque posée (point de synchronisation).

En supposant l'envoi terminé, l'utilisateur va émettre une requête de demande de libération de la connexion session. Le récepteur voulant à son tour envoyer des données peut refuser cette demande.

11.6.2.4. LE RTS FACE A LA COUCHE SESSION.

Suite à ces généralités, chercher à savoir comment le RTS fait usage de ces outils s'impose. Premièrement, X400 propose l'utilisation des unités fonctionnelles et leurs services associés suivants :

Noyau

Connexion pour la session

Transfert de données normales

Libération normale

Interruption provoquée par l'usager

Interruption provoquée par le fournisseur

Impossibilités

Compte rendu d'impossibilité par l'usager

Compte rendu d'impossibilité par le fournisseur

Gestion de l'activité

Démarrage de l'activité

Reprise de l'activité

Fin de l'activité

Interruption de l'activité

Rejet de l'activité

Demande de jetons

Cession de jetons ou don de jetons

Commande de cession

Semi-duplex

Cession du jeton

Demande de jeton

Synchronisation mineure

Point de synchronisation mineure

Cession de jeton ou don de jeton

Demande de jeton

Le RTS n'emploie que des jetons de données, de synchronisation mineure et de synchronisation majeure et activités. De plus la norme exige l'assignation de tous les jetons au même RTS. La connexion est rejetée en cas de violation de cette règle. Dès lors, à tout moment, le RTS détenant les jetons est nommé RTS envoyeur, par opposition au RTS récepteur. Lors de l'établissement de la connexion session, les entités concernées doivent également régler le problème de la quantité maximum de données pouvant être envoyées entre deux points de synchronisation mineure, de même que le nombre de points de synchronisation mineure pouvant être posés sans attendre de confirmation.

Chaque MPDU à envoyer fait l'objet d'une activité. Celui-ci est découpé en "paquets" qui sont envoyés séparément, un point de synchronisation mineure étant chaque fois intercalé entr'eux. L'activité peut être interrompue pour cause de changement de tour, d'erreurs ou de problèmes détectés par l'utilisateur ou le fournisseur de la session. Par la suite, il est possible de reprendre cette activité dans le but de l'achever correctement i.e. de terminer l'envoi du MPDU. Les normes imposent l'existence unique d'une activité au cours d'une connexion session et la présence d'une seule activité interrompue attendant sa reprise. Il est à souligner que l'interruption d'une activité peut s'accompagner d'une rupture brusque de connexion session. Poursuivre l'activité interrompue ne peut, par conséquent, s'effectuer que sur une nouvelle connexion session.

La libération de la connexion peut se faire d'une manière brutale (connexion session suspendue par un utilisateur ou un fournisseur de la session) ou d'une

manière négociée (suite à une demande de libération de l'association entre les deux AM correspondant aux deux RTS). Négocier doit être pris dans le sens où le RTS émetteur prévient l'autre de la fin d'association et le RTS récepteur, obligé d'accepter, répond qu'il a reçu l'indication de fin de connexion session.

Dès à présent, on ressent la difficulté de construire un RTS. Il est regrettable à ce propos que la norme X410 soit la moins précise. Cependant ce flou, gênant pour celui qui désire construire un RTS, est introduit volontairement dans la norme et ceci afin de laisser à l'implémenteur plus de liberté suivant ses besoins. Il est ennuyeux cependant que certains concepts soient évoqués mais traités simplement superficiellement. Ainsi, on peut trouver des primitives sans le détail de leurs paramètres ou encore des noms de primitives dont le sens et la syntaxe ne sont jamais précisés.

11.6.3. LE RTS AU SEIN DE L'ARCHITECTURE PARALLELE.

11.6.3.1. INTRODUCTION.

Il est nécessaire de présenter l'élaboration graduelle du RTS placé dans l'architecture parallèle. Dans une première étape, l'utilisation de diagrammes d'états permet de présenter les différents aspects du RTS : tous les états qu'il peut atteindre et tous les événements qui le font changer d'état.

Les actions prises par le RTS sont présentées dans une deuxième étape. Ces deux étapes vont se détacher intentionnellement des messages représentant une requête ou un signal et dissimuler leur structure le plus longtemps

possible afin de permettre une explication plus souple du RTS et de ses stratégies. L'inventaire de tous les messages placés au point 11.9. va permettre de compléter la construction du RTS, de même que l'algorithme (annexe 8) permet de suivre pas à pas les actions du RTS suivant son état et les messages reçus.

11.6.3.2. Le diagramme d'états du RTS.

Les différents états que peut prendre le RTS et les événements déclenchants une transition d'un état à un autre sont contenus dans le diagramme se trouvant à la fin de cette section. Celui-ci peut être vu comme étant la réunion de trois diagrammes d'états.

Le RTS dans sa phase d'établissement ou de libération de connexion session (premier diagramme i.e. RTS1).

le RTS avec jetons (deuxième diagramme i.e. RTS2).

Le RTS sans jeton (troisième diagramme i.e. RTS3).

Tout comme pour le MD, les états peuvent être séparés en deux groupe:

le premier comprend les états stables, c'est-à-dire les états pendant lesquels le RTS est inactif car aucune réponse à un message qu'il a émis n'est attendue ou à traiter. Il reprendra son activité à la réception d'une requête de l'AM, du RTS pair ou suite à une initiative personnelle.

Le second regroupe les états transitoires, c'est-à-dire les états pendant lesquels le RTS attend une réponse ou exécute

une procédure d'analyse et de traitement de messages. Après cette attente ou cette analyse, le RTS change d'état.

A. Le RTS dans sa phase d'établissement ou de libération de connexion session.

Le RTS peut se trouver dans les états stables IDLE ou CONNECTE (avec ou sans jetons).

De l'état IDLE,

si le RTS émet une demande de création de connexion session, il attend la réponse dans l'état EN ATTENTE DE CONNEXION. Si celle-ci est positive, il passe en état CONNECTE. Sinon, il va en état ECHEC ou ASSOCIATION pour exécuter une procédure de désactivation de la machine qui la fera transiter par les états ATTENTE PIPE BOUCHE, ATTENTE DESTRUCTION.

si le RTS recoit une indication signalant une demande de création de connexion session, il analyse cette demande en l'état ANALYSE OUVERTURE CONNEXION. Si la demande réussit, il passe en l'état CONNECTE, sinon il atteint l'état ATTENTE DESTRUCTION avant de pouvoir désactiver la machine.

De l'état CONNECTE avec jetons,

le RTS peut négocier la libération de la connexion session et passe dans l'état ATTENTE FERMETURE en vue d'exécuter une procédure de désactivation de la machine.

De l'état CONNECTE sans jeton,

le RTS reçoit l'ordre de libérer la connexion session. Il quitte l'état CONNECTE pour aller en ATTENTE FERMETURE afin de déclencher une procédure de libération de la machine.

Voici la liste des états stables :

IDLE

CONNECTE AVEC JETONS

CONNECTE SANS JETON

Voici la liste des états transitoires:

ANALYSE OUVERTURE CONNEXION,

EN ATTENTE DE CONNEXION,

ECHEC DE L'ASSOCIATION,

ATTENTE FERMETURE,

ATTENTE PIPE BOUCHE,

ATTENTE DESTRUCTION.

B. Le RTS avec jetons.

Le RTS peut stationner dans deux états stables : CONNECTE avec JETONS et FIN TRAVAIL AVEC JETONS.

Les états stables CONNECTE sans jeton et FIN TRAVAIL SANS JETON dans le diagramme RTS3 afin d'établir la liaison avec le diagramme "le RTS sans jeton", soit RTS2.

De l'état CONNECTE avec JETONS :

Suite à une demande de transfert de l'AM, le RTS passe dans l'état ENVOI où il active une procédure d'envoi de messages en utilisant les primitives offertes par la couche session. Cet état ENVOI SESSION, vu de près, remplace à lui seul un diagramme d'états (diagramme RTS4). Il peut quitter l'état SESSION de deux façons :

soit il retourne dans l'état CONNECTE avec JETONS (fin de l'activité mais connexion session toujours créée).

soit il passe en l'état FIN TRAVAIL AVEC JETONS (fin de l'activité et libération brutale de la connexion session).

Tout en restant dans l'état CONNECTE AVEC JETONS, le RTS peut dialoguer avec son AM. Il lui rend un MPDU pour lequel il n'a pu assurer le transfert.

Le RTS transite de l'état CONNECTE AVEC JETONS vers l'état CONNECTE SANS JETON (et vice versa) suite à des requêtes don de tour/de jetons, demande de tour/de jetons.

De l'état FIN TRAVAIL AVEC JETONS :

Le RTS stationne dans l'état FIN TRAVAIL AVEC JETONS lorsqu'il a créé une connexion session au moins une fois dans son existence. Celle-ci s'est terminée brutalement (c'est-à-dire abandon de la connexion sans négociation). Au moment de la libération de la connexion, il possèdait les jetons.

soit il décide de réétablir une nouvelle connexion faisant suite à la précédente (brutalement coupée) et passe dans l'état ATTENTE CONNEXION AVEC JETONS. Si sa requête réussit, il va donc en CONNECTE AVEC JETONS ou en ENVOI SESSION car il a memorisé précédemment une demande de transfert (TRANSFER-REQ). En cas d'échec, il reste dans l'état FIN TRAVAILL AVEC JETONS.

soit le RTS (du MTA voisin) qui lui était connecté avant la rupture essaie d'ouvrir une nouvelle connexion faisant suite à la précédente (coupée). Il va en ANALYSE DEMANDE RECOVER et traite la demande. S'il y répond positivement, il passe en CONNECTE AVEC JETONS, sinon il retourne en l'état FIN TRAVAIL AVEC JETONS.

Voici la liste des états stables :

CONNECTE AVEC JETONS

FIN TRAVAIL AVEC JETONS

La liste des états transitoires est données ci-dessous:

ENVOI SESSION.

ATTENTE CONNEXION AVEC JETONS.

ANALYSE DEMANDE RECOVER.

C. Le RTS sans jeton.

Le diagramme du RTS sans jeton peut être calqué sur le diagramme du RTS avec jetons. Le raisonnement similaire au cas B. n'est par conséquent pas détaillé ici.

11.6.3.3. Actions prises par le RTS.

Les actions exécutées par le RTS peuvent être classées en deux catégories :

les actions entraînant un traitement "normal"

les actions entraînant un traitement d'exception

Ceci s'explique d'une part par le fait que certaines font suite à des commandes émanant de l'AM et d'autres sont initialisées suite à des décisions propres au RTS. D'autre part, l'AM ignore où le RTS se situe dans son code d'exécution. En plus de ces remarques, il faut se rappeler que les messages atteignent tous le RTS via le pipe unique RTS. Ainsi, le RTS attendant par exemple une réponse du RTS voisin, celle-ci faisant suite à une de ses requêtes, peut recevoir un ordre de son AM n'ayant aucun lien avec le travail en cours, ou même à la limite lui étant contradictoire. Il est par conséquent préférable de présenter, dans une première étape les enchaînements dits normaux, puis de greffer à ceux-ci les cas d'exception, conséquences de l'arrivée inattendue de messages.

A. Déroulement "normal" des actions

Les enchaînements normaux sont en fait ceux prescrits par X410.

Suite à la réception d'un ordre de l'AM, le RTS émet une requête à son RTS pair. Conformément à la chronologie des primitives en quatre temps (req, ind, resp, conf), le message suivant en provenance du RTS appelé doit être la confirmation de cette demande. Ou encore, lorsque le RTS active son algorithme réalisant un envoi, il dialogue avec le RTS auquel il est connecté et s'attend donc à ne recevoir que des primitives de son RTS pair.

Pour ces cas normaux, le lecteur est renvoyé à la description du RTS présentée en 11.4.2.3.

B. Déroulement "d'exception".

Parmi ces cas spéciaux, une gradation de situations peut être imaginée.

Au premier niveau se situent toutes les primitives d'interruption d'activité ou d'arrêt de connexions. Bien que citées dans la norme, certaines sont simplement survolées. Qui plus est, on peut déplorer le fait que rien n'indique la stratégie du RTS concernant le choix de ces primitives d'arrêt. Pour ne citer que quelques problèmes : pourquoi le RTS decide-t-il d'écarter une activité plutôt que de l'interrompre; étant donné qu'un seul MPDU peut être en attente de reprise, lequel sélectionner en cas de nouvelle interruption; quand le RTS prend-il en charge un recouvrement d'activité, le plus vite possible pour tenter de réussir l'envoi ou après les autres demandes d'envoi non encore traitées ?

Face à ces imprécisions ou plus exactement cette liberté introduite volontairement dans la norme, la politique du RTS est par consèquent laissée au choix de l'implémenteur. Dans le modèle d'implémentation personnel, nous n'avons pas pris position, faute d'expérience en le domaine. Un réglage en début d'exploitation pourrait permettre d'adopter une politique appropriée.

Le second niveau comprend les requêtes de transfert (TRANSFER) et de fermeture d'association (CLOSE) en provenance de l'AM. Ces primitives peuvent poser problèmes car elles peuvent se glisser dans le dialogue entre les deux RTS. En effet, tout d'abord l'AM communique avec son AM pair en utilisant les services du RTS. La manière dont s'exécute le RTS reste transparente pour cet AM. En plus de cela, la seule voie de communication vers le RTS étant

RTS, les messages provenant de l'AM et du RTS pair s'entremêlent.

Face à ce problème, le choix suivant est fait : si le RTS attend un message du RTS pair, il va passer outre les demandes de l'AM tout en les mémorisant afin d'y revenir dès qu'il n'attend plus rien en provenance des couches inférieures du modèle OSI.

Le dernier niveau est de même nature que le précédent. Il s'agit du cas où les primitives de demande de tour et de don de tour émises par l'AM s'intercalent au coeur du dialogue entre les RTS pairs. Quelle tactique adopter si au cours de l'exécution de sa procédure d'envoi d'MPDU, le RTS reçoit de son AM l'ordre de donner le tour ? Plusieurs réponses peuvent être avancées : soit le RTS obéit immédiatement à son AM et pour ce faire, il interrompt ou écarte l'émission en cours, soit il décide de poursuivre son travail et de repousser momentanément la requête de l'AM. Cependant, il est indispensable qu'il mémorise cette demande car lui seul est conscient de cette désynchronisation : AM sans tour, RTS avec jetons. La demande de l'AM n'étant pas perdue, dès que possible, le RTS l'effectue. A la limite, le fait de choisir parmi l'une des deux solutions proposées peut dépendre de la priorité de l'MPDU interrompu. L'implémentation personnelle envisage ces deux possibilités.

Toutes ces explications concernant le RTS permettent de comprendre qu'il est nécessaire de doter ce RTS d'une certaine intelligence.

Voici entre autres un cas où il est important que le RTS réagisse correctement. Soit le cas d'un MPDU correctement reçu par le RTS destinataire mais pour lequel la confirmation de la fin d'activité s'est perdue (par conséquent n'est pas reçue par le RTS émetteur). Dans de telles circonstances, le RTS émetteur est persuadé que le dernier bloc de données n'a pas atteint son destinataire. Si par la suite le RTS émetteur décide de reprendre cet envoi en vue de le terminer, le RTS récepteur doit être en

mesure de s'apercevoir de la confusion qui règne du côté origine et de feindre d'accepter la reprise.

Il ne faut pas l'oublier, le RTS doit appliquer les techniques consistant à prévenir son gestionnaire d'une fin de connexion session, à assurer l'arrêt d'une machine (AM-RTS) par la méthode en quatre temps et suite à cela, à réécrire les primitives vers le GI-MTA-MTA. Tout ceci n'apparaît plus ici puisqu'ayant fait l'objet du chapitre XIII, consacré au GI-MTA-MTA.

11.6.3.4. Conflit d'ouverture de connexion session.

Il est impossible de clôturer ce paragraphe sans poser le problème des demandes concurrentes d'ouverture de connexion session.

Si deux primitives de demande d'ouverture de connexion session sont émises simultanément (moyennant le temps de transport et de traitement des demandes) par les deux RTS, quelle entité peut arbitrer le conflit ?

Plusieurs solutions sont envisageables :

soit un système de priorités entre MTA est instauré. Celles-ci peuvent par exemple être enregistrées dans les informations bilatérales. Le MTA le plus prioritaire voit sa demande satisfaite, tandis que l'autre est écartée.

soit les deux demandes sont écartées et une procédure de time-out est mise en marche : après ce time-out, chaque RTS peut envoyer à nouveau sa requête. Il est évidemment souhaitable que les time-out soient différents. Il faut cependant noter qu'adopter une telle proposition semble insolite dans le contexte de la communication à distance.

soit il est possible d'imaginer d'insérer le temps d'envoi dans le paramètre ss-user-data. Par

comparaison des temps, le choix de la demande devient immédiat. Cette solution semble déboucher sur le problème de la synchronisation de machines éloignées, qui pose de réelles difficultés.

Il faut signaler que ce conflit d'ouverture de connexion session se pose seulement dans notre implémentation. Il est dû à la restriction : une seule association et une seule connexion session entre deux MTA.

L'algorithme du RTS (annexe 8) a été conçu sans prévoir ce cas d'ouverture simultanée.

11.6.4. Conclusion.

En résumé, pour assurer le transfert fiable des MPDU, le RTS utilise les primitives offertes par la couche session.

D'une part, des diagrammes d'états permettent de spécifier tous les états que peut atteindre le RTS et tous les événement déclenchant un changement d'état. D'autre part, la description des actions effectuées par le RTS, suite à l'occurence de certains événements, permet de justifier la conception de ce module.

L'algorithme (annexe 8) fournit le déroulemant chronologique des actions suite aux messages reçus et à différentes stratégies propres au module RTS.

Le paragraphe suivant consacré au GC permet de terminer la conception des modules fonctionnels de l'architecture parallèle.

DIAGRAMME D'ETATS

RTS1

RTS 1

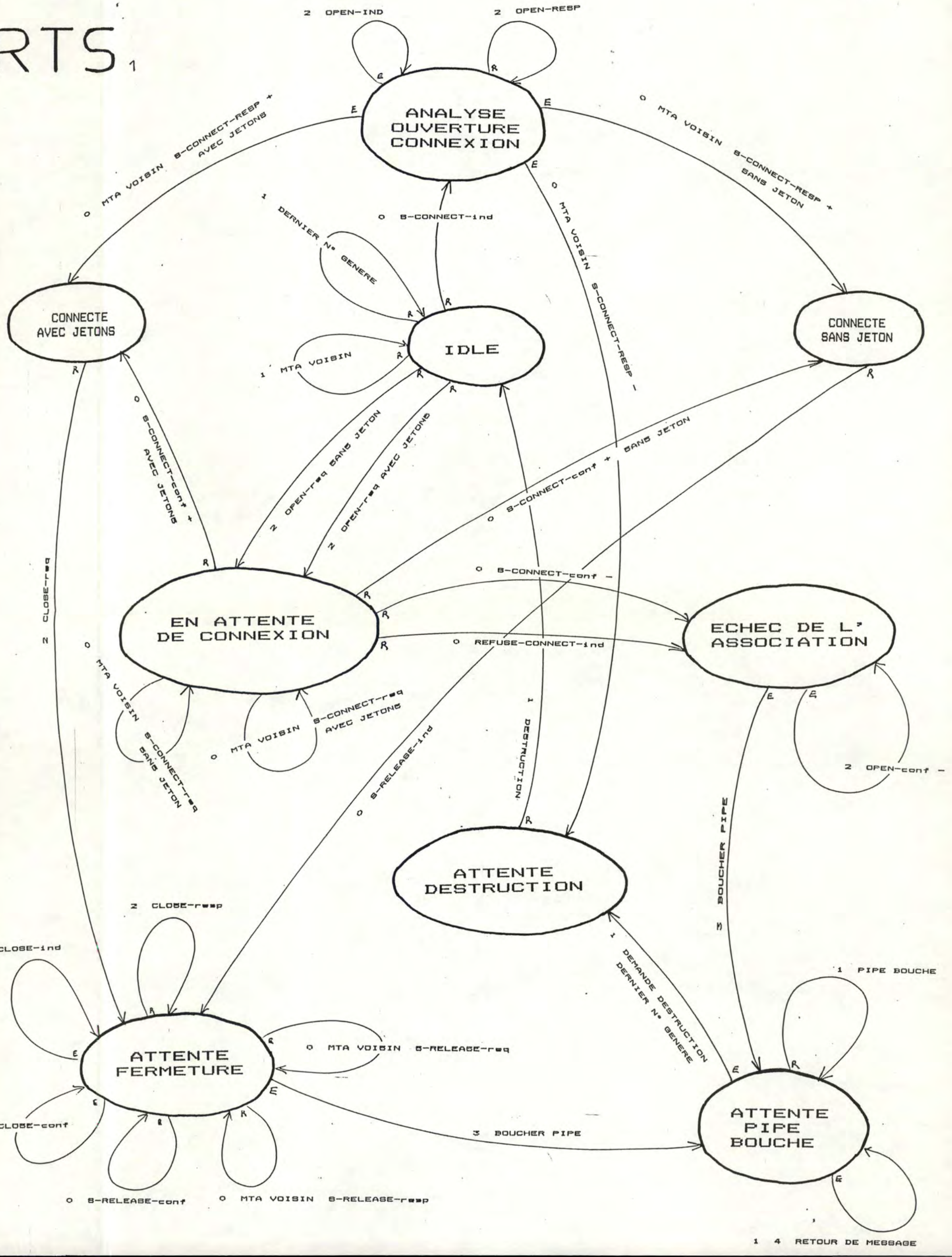
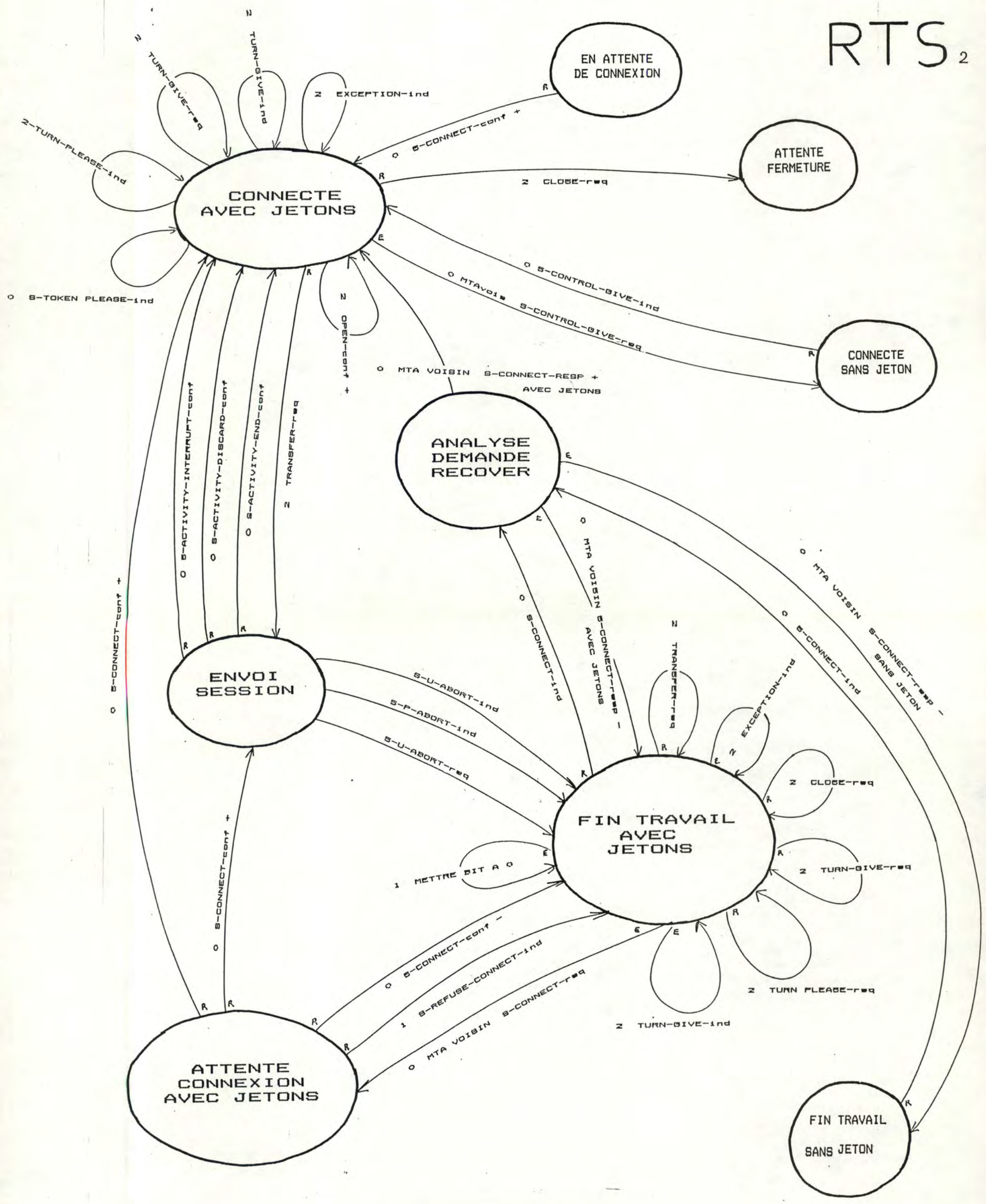

2 OPEN-IND
2 OPEN-REQP
ANALYSE OUVERTURE CONNEXION
0 MTA VOISIN S-CONNECT-REQP + AVEC JETONS
0 MTA VOISIN S-CONNECT-REQP + SANS JETON
1 DERNIER N° GENERE
0 S-CONNECT-ind
0 MTA VOISIN S-CONNECT-RESP -
CONNECTE AVEC JETONS
IDLE
CONNECTE SANS JETON
1 MTA VOISIN
0 S-CONNECT-conf + AVEC JETONS
2 OPEN-req SANS JETON
2 OPEN-req AVEC JETONS
0 S-CONNECT-conf + SANS JETON
2 CLOSE-req
0 S-CONNECT-conf -
EN ATTENTE DE CONNEXION
0 REFUSE-CONNECT-ind
ECHEC DE L' ASSOCIATION
0 MTA VOISIN S-CONNECT-req SANS JETON
0 MTA VOISIN S-CONNECT-req AVEC JETONS
0 S-RELEASE-ind
1 DESTRUCTION
2 OPEN-conf -
3 BOUCHER PIPE
ATTENTE DESTRUCTION
2 CLOSE-reqp
2 CLOSE-ind
1 DEMANDE DESTRUCTION DERNIER N° GENERE
1 PIPE BOUCHE
ATTENTE FERMETURE
0 MTA VOISIN S-RELEASE-req
ATTENTE PIPE BOUCHE
2 CLOSE-conf
3 BOUCHER PIPE
0 S-RELEASE-conf
0 MTA VOISIN S-RELEASE-reqp
1 4 RETOUR DE MESSAGE

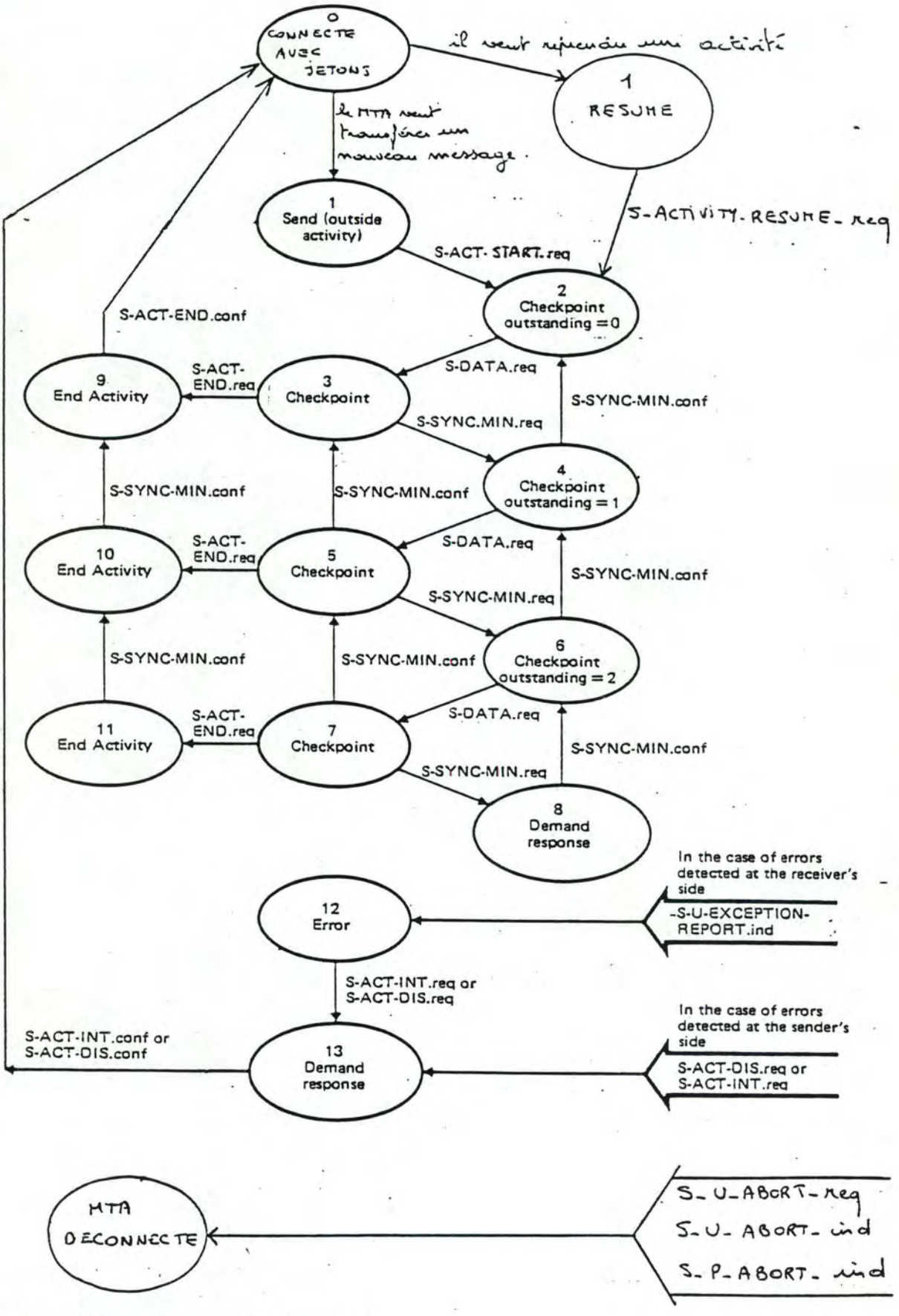
DIAGRAMME D'ETATS

RTS2

RTS 2


EN ATTENTE DE CONNEXION
ATTENTE FERMETURE
CONNECTE AVEC JETONS
CONNECTE SANS JETON
ANALYSE DEMANDE RECOVER
ENVOI SESSION
FIN TRAVAIL AVEC JETONS
ATTENTE CONNEXION AVEC JETONS
FIN TRAVAIL SANS JETON
N TURN-GIVE-req
N TURN-GIVE-ind
N EXCEPTION-ind
N TURN-PLEASE-ind
O S-CONNECT-conf +
N CLOSE-req
O S-TOKEN PLEASE-ind
O S-CONTROL-GIVE-ind
O MTA voisin S-CONTROL-GIVE-req
N OPEN-conf +
O MTA VOISIN S-CONNECT-RESP + AVEC JETONS
O S-ACTIVITY-INTERRUPT-conf +
O S-ACTIVITY-DISCARD-conf +
O S-ACTIVITY-END-conf +
N TRANSFER-req
O S-CONNECT-conf +
O S-CONNECT-ind
O MTA VOISIN S-CONNECT-resp - AVEC JETONS
N TRANSFER-req
N EXCEPTION-ind
O MTA VOISIN S-CONNECT-resp - SANS JETON
O S-CONNECT-ind
S-U-ABORT-ind
S-P-ABORT-ind
S-U-ABORT-req
N CLOSE-req
1 METTRE BIT A 0
N TURN-GIVE-req
O S-CONNECT-conf -
1 S-REFUSE-CONNECT-ind
O MTA VOISIN S-CONNECT-req
N TURN PLEASE-req
N TURN-GIVE-ind
R
E

ENVOI-SESSION



Note – This diagram assumes a window size of 3.

ENVOI-SESSION

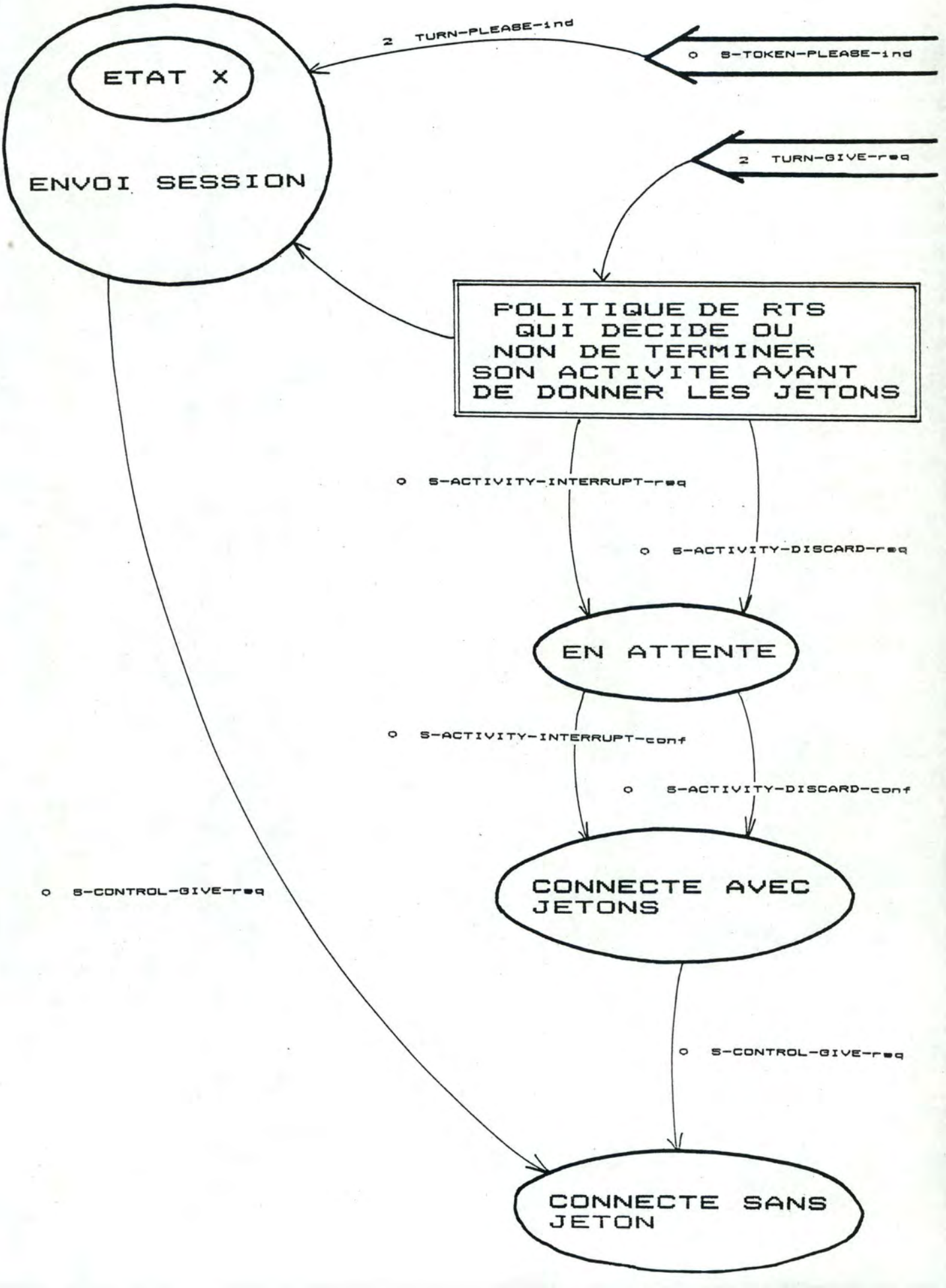
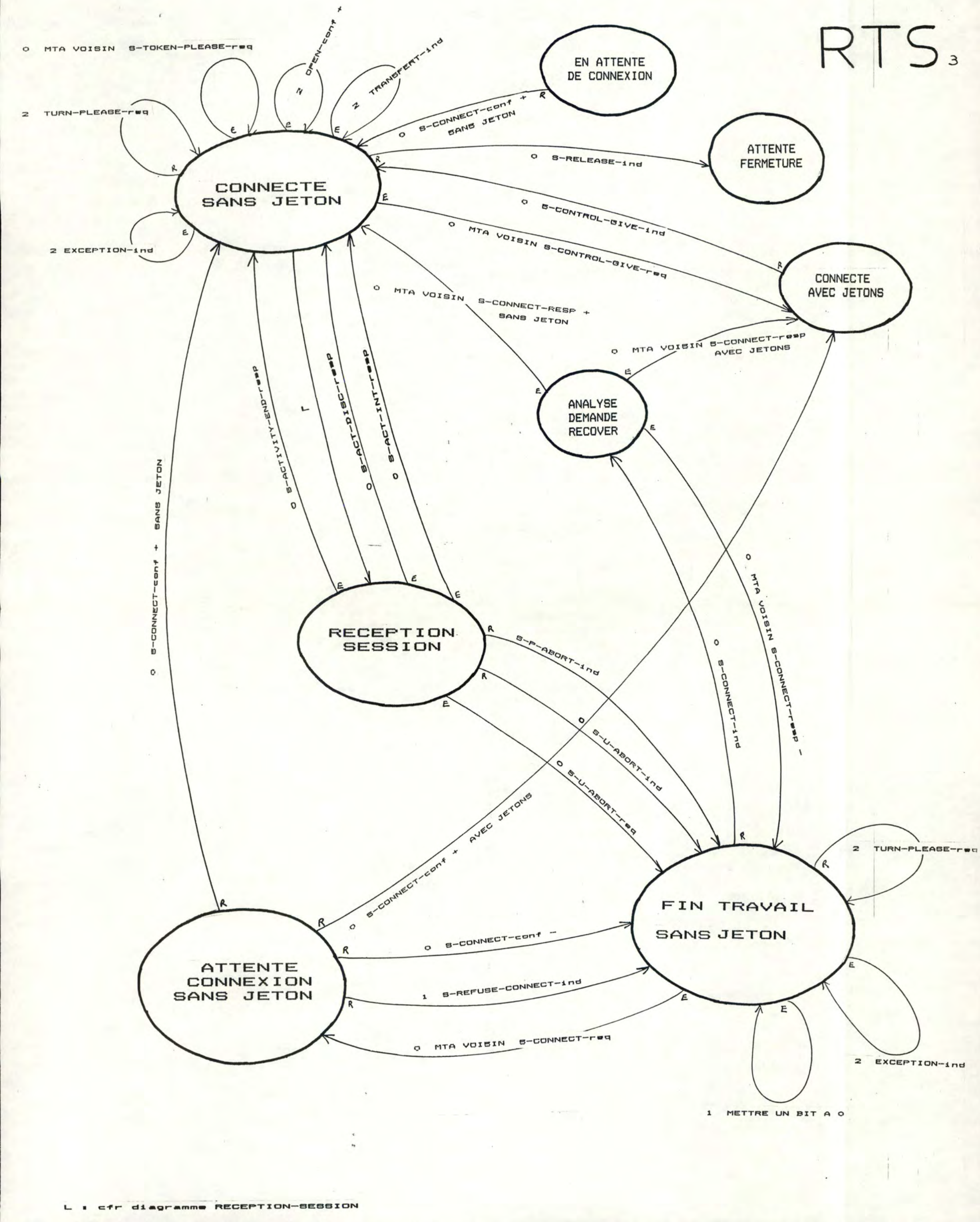


DIAGRAMME D'ETATS

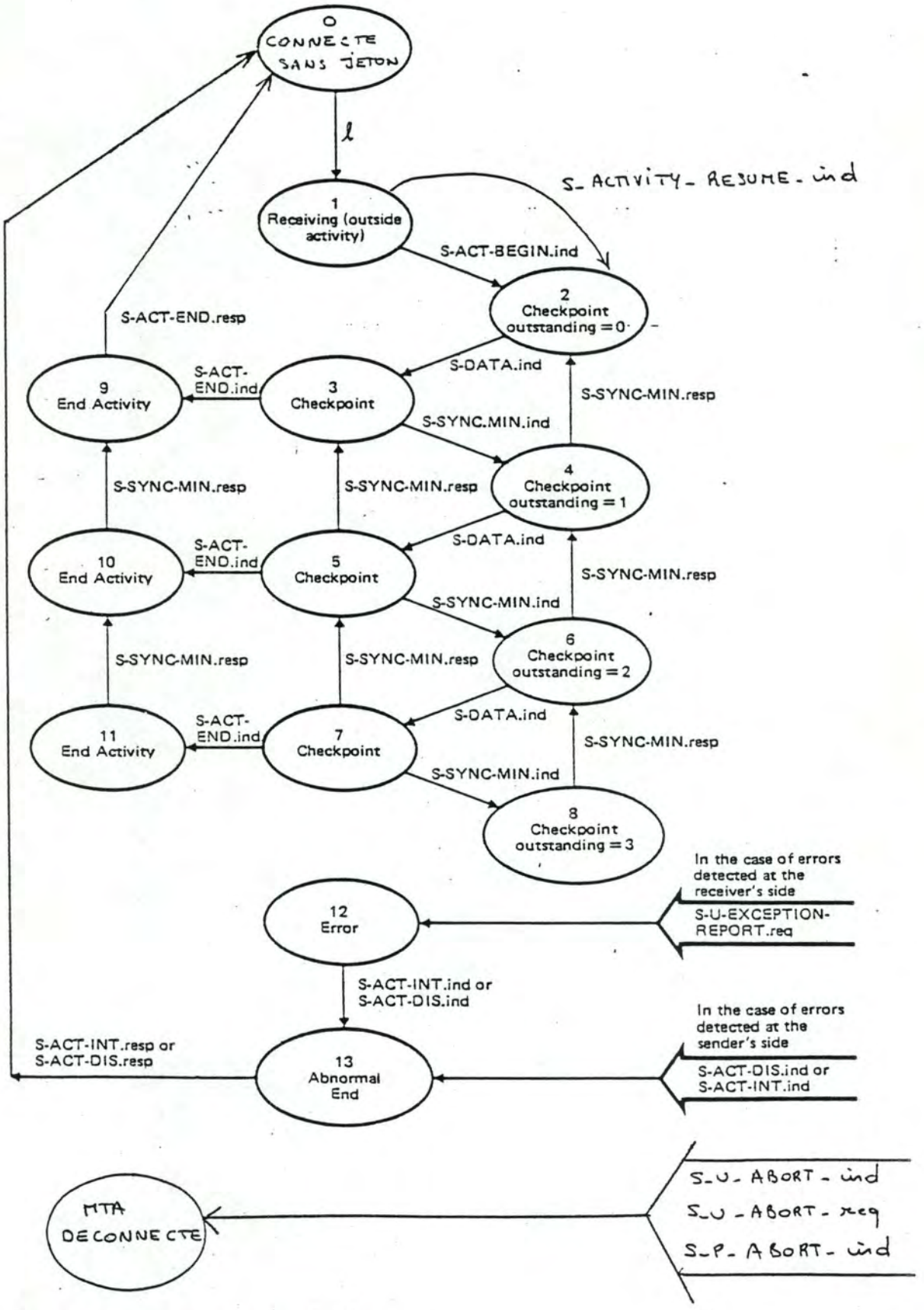
RTS3

RTS 3



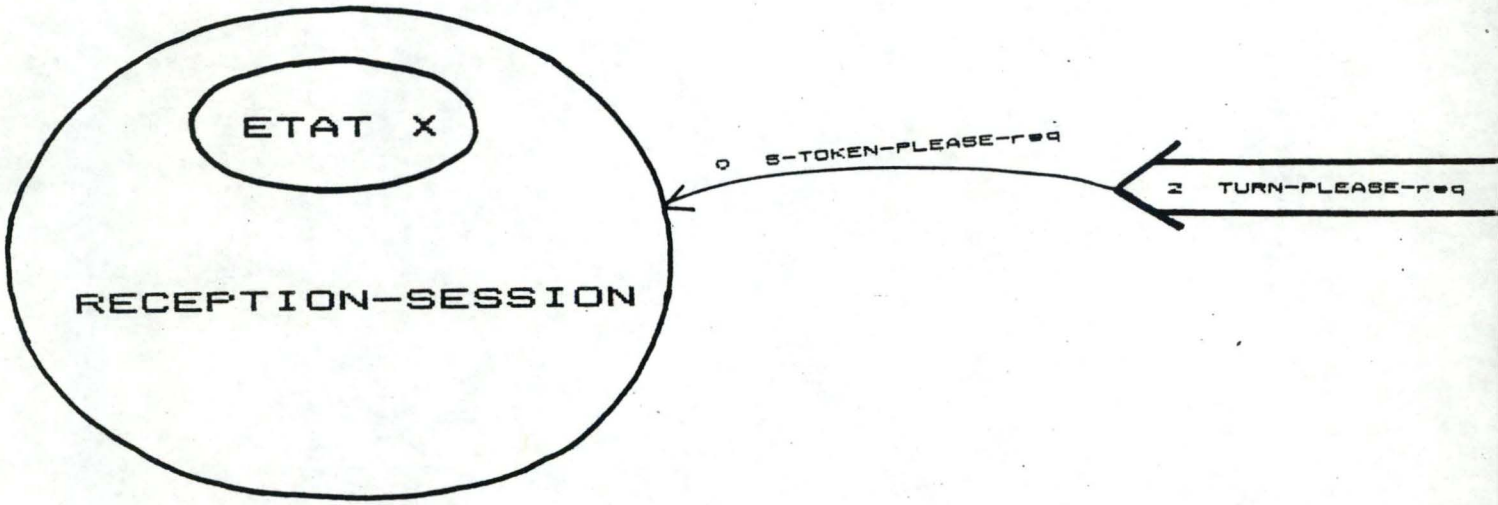
L : cfr diagramme RECEPTION-SESSION

RECEPTION-SESSION



Note – This diagram assumes a window size of 3.

RECEPTION-SESSION



XI.7. LE GESTIONNAIRE CENTRAL

Le Gestionnaire Central (GC) auquel les GI-UA-MTA et GI-MTA-MTA font très souvent référence, se doit d'effectuer deux types de tâches.

La première concerne l'aiguillage des informations venant de l'exérieur du MTA, que ce soit d'un UA ou de la couche session du modèle OSI.

Comme expliqué à la section 10.5.2, le GC reçoit les informations de l'extérieur par une source unique. Son rôle est alors de détecter l'origine des données et de les diriger correctement vers le GI-UA-MTA ou le GI-MTA-MTA.

Le GC peut aussi recevoir des messages des deux gestionnaires d'interaction. Son rôle est alors de diriger ces messages vers un UA, la couche session ou l'autre gestionnaire. Pour ce faire, il remplace, comme précisé en XI.9. les premiers champs des messages qui servent d'indication de direction.

Dans certains cas, les informations qu'il reçoit par l'intermédiaire d'un message venant du GI-UA-MTA peuvent être destinées au MTA pour sa propre gestion. Le GC prend alors ces données pour les placer dans une zone mémoire réservée à cet effet.

L'algorithme du GC peut être trouvée à l'annexe 9.

XI.8. LISTE DES TABLES

Ce paragraphe propose une liste exhaustive des tables nécessaires pour mémoriser les informations rendues nécessaires par l'introduction de la notion de parallélisme.

TABLE 1 : (GC)

id-UA	référence UA

TABLE 2 : (GC)

MTA vois	référence MTA vois

Ces deux tables sont indispensables au GC pour communiquer avec des processus n'ayant aucun lien de parenté. Le GC doit interagir avec les UA et les couches inférieures du modèle OSI.

TABLE 3 : (GI-UA-MTA)

id-UA	UA-OR-NAME	$\overline{\text{MDi}}$	MDi	bitconnex	verrou

Cette table contient la liste des UA (identifiés par OR-NAME de X400) attachés à un MD.

id-UA : identificateur local de l'UA

MDi, $\overline{\text{MDi}}$: noms des pipes descendant et montant

bit connex : 1 si l'UA est connecté au MD

0 sinon (* après logoff *)

verrou : entier, donne le nombre de services demandés au

MD et non encore réalisés

Elle contient aussi la liste des MD créés dans le but premier de gérer un relais où regarder s'il y a lieu de créer une notificatoin de non livraison

id-UA = 0

nom UA = null

MDI nom du pipe descendant

$\overline{\text{MDI}}$ nom du pipe remontant

bit connex = 0

verrou = nombre de services demandés à ce MD qui n'ont

pas été confirmés terminés

TABLE 4 : (GI-UA-MTA)

id-UA	password	restrictions register

Cette table contient la liste exhaustive des UA abonnés à cet MTA et de leurs caractéristiques

id-UA : pour faciliter la reconnaissance locale des UA et

éviter ainsi l'emploi des OR-NAME de X400

nom UA : OR-NAME

password : mot de passe devant être fourni par l'UA au

moment où celui-ci veut se connecter, dans le

but de s'identifier

table-register : paramètres concernant les possibilités

de réception de messages de l'UA

(paramètres de la primitive REGISTER)

TABLE 5 : (GI-UA-MTA) table des machines MD

nom machine MD	nom pipe $\overline{\text{MDi}}$	nom pipe $\underline{\text{MDi}}$	dernier n° généré	bit d'occupation

Cette table comprend la liste exhaustive des machines MD possibles

nom machine : MD

$\overline{\text{MDi}}$, $\underline{\text{MDi}}$: noms des pipes à utiliser lors de la création

de la machine

dernier n° généré :lors de l'envoi de messages (suite à

un submit) le MD attribue un identifi-

cateur de submit à cet envoi.

Cet identificateur doit être unique par soumission de message et par MTA.

Dû à notre parallèle, l'identificateur est marqué par le nom de la machine suivi d'un numéro qui croît strictement.

Lors de la création d'un MD, le GI-UA-MTA fournit au MD le dernier numéro. Celui-ci le gère (l'incrémente correctement) pendant toute son "existence". Avant sa destruction, le MD le restitue au GI-UA-MTA qui le conserve jusqu'à la prochaine création de la machine. Incontestablement, seul le GI-UA-MTA peut centraliser ces identificateurs étant donné que

les MD ont une durée de vie limitée

le GI-UA-MTA produit les machines et administre en quelque sorte les états du MD

bit d'occupation = 1 : la machine est créée

= 0 : la machine est non créée et donc disponible

TABLE 6 : (GI-UA-MTA)

MTA password	restrictions MTL-CONTROL

Cette table renferme certains paramètres concernant le MTA.

Chaque MD doit connaître son mot de passe s'il désire se connecter à un UA. Le troisième paramètre contient les restrictions du MD

(paramètres de la primitive MTL-CONTROL).

TABLE 7 : (GI-UA-MTA) tables de routage

7.1 nom pays liste des MTA adjacents, sur la route
7.2 nom domaine liste des MTA adjacents, sur la route
un par pays
7.3 nom de l'organisation liste des MTA adjacents, sur la route
un par domaine
7.4 adresse précise d'un UA liste des MTA adjacents, sur la route
un par organisation
7.5 adresse X121 liste des MTA adjacents, sur la route
7.6 identificateur numérique d'UA liste des MTA adjacents, sur la route

Cette table est utilisée par l'algorithme de routage. En fonction des champs de l'adresse, ces tables donnent la liste des MTA adjacents permettant de rapprocher le message de sa destination.

TABLE 8 : (GI-MTA-MTA) table des associations

MTA vois (AMdest)	AM orig	AM orig	AM orig	bit pipe

Cette table mémorise les associations établies.

MTA vois : nom du MTA voisin concerné par l'association

AM orig : nom de l'AM qui a établi une association avec

AMdest.

AM orig est dans MTA qui contient cette table, AMdest est dans MTA vois

AM orig, AM orig : noms des pipes unidirectionnels créés entre le GI-MTAM-MTA et AM orig. La communication dans AM orig s'effectue dans le sens AM orig --> GI-MTA-MTA. Celle qui se fait dans AM orig se fait dans le sens GI-MTA-MTA --> AM orig

bit pipe : il vaut 1 si le pipe Ami est bouché i.e. si le GI-MTA-MTA ne peut plus rien y écrire, il vaut 0 sinon

TABLE 9 : (GI-MTA-MTA) table des RTS créés

MTA vois bitpipe (RTSdest)	RTS orig	RTS orig	RTS orig	bit réservation

Cette table contient les données concernant les machines RTS créées.

MTA vois : nom du MTA voisin concerné par la connexion session

RTS orig : nom de la machine RTS qui a établi une connexion session avec un RTS de MTA vois

RTS orig, RTS orig : noms des pipes descendant et montant, pipes reliant le GI-MTA-MTA et le RTS orig

bit réservation : ce bit vaut 1 si la connexion session entre RTS orig et RTS dest est créée. Il vaut 0 sinon

bit pipe : ce bit est mis à 1 lorsque le pipe RTS orig est bouché i.e. quand le GI-MTA-MTA n'y écrit plus rien. Il est mis à 0 sinon

TABLE 10 : (GI-MTA-MTA) table des machines (AM-RTS)

nom machine	$\overline{AMi}$	$\underline{AMi}$	$\overline{RTSi}$	$\underline{RTSi}$	dernier n° généré	bit d'occupation

Cette table répertorie les noms des machines AM-RTS possibles.

nom machine : nom de la machine

$\overline{AMi}$, $\underline{AMi}$, $\overline{RTSi}$, $\underline{RTSi}$: noms des pipes permettant de communiquer avec les modules de cette machine

dernier n° généré : ce paramètre a la même fonction que le paramètre dernier n° généré de la table 5. Dernier n° généré est utilisé pour rendre uniques les identificateurs de connexion session

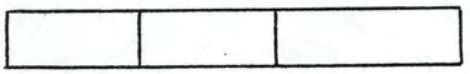
bit d'occupation : ce bit mis à 1 signifie que la machine est utilisée. Sinon il vaut 0 et la machine est disponible

XI.9. LISTE DES MESSAGES

11.9.1 MESSAGES ARRIVANT AU G.C

GI-UA-MTA ---> GC

A) interprétation du message reçu

message = 

[A] [B] [C]

si [A] = 0 : le message est destiné à un UA

[B] = id-UA

[C] = primitive

si [A] = 1 : le messages est destiné au GI-MTA-MTA

[B] = MTAvois

[C] = ENVOI(MPDU)

si [A] = 2 : le message est pour le GC

([B][C]) = .INFO BILAT(paramètres)

.NOTIFY MTA(paramètres)

B) modifications à apporter au message avant de l'envoyer vers la destination demandée

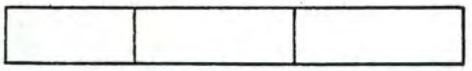
si [A] = 0 : [A] et [B] sont supprimés

si [A] = 1 : /

si [A] = 2 : les paramètres reçus sont stockés dans une Base de Données du GC

GI-MTA-MTA ---> GC

A) interprétation du message reçu

message = 

[A] [B] [C]

si [A] = 0 : le message vient du RTS;

il est destiné à [B] = MTAvois et doit donc être dirigé vers les couches inférieures du modèle OSI. [C] = primitive

si [A] = 1 : ([B][C]) vaut NULL ENVOI

ou -1 AM DEMNOTNL

ou -1 GAM DEMNOTNL

le message vient soit d'un AM, soit du GI-MTA-MTA et est destiné à un MD.

B) modifications à apporter au message avant de l'envoyer vers la destination demandée

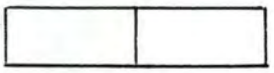
si [A] = 0 : [A] et [B] sont supprimés

si [A] = 1 : [A] est enlevé

11.9.2.MESSAGES ARRIVANT AU GI-UA-MTA

GC ---> GI-UA-MTA

A) interprétation du message reçu

message = 

[A] [B]

si [A] = id-UA : le message vient d'un UA local.

[B] = primitive

si [A] = NULL : le message vient de l'AM pour un relais

[B] = relais(APDU)

si [A] = -1 : le message vient de l'AM ou du GI-MTA-MTA pour une création de notification de non livraison si nécessaire

[B] = AM DEMNOTNL(MPDU)

ou GAM DEMNOTNL(MPDU)

Dans les trois cas, le message doit être distribué à un MD.

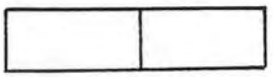
B) modifications à apporter au message avant de l'envoyer vers la destination demandée

si [A] = id-UA : [A] est remplacé par 0

sinon le message est passé tel que reçu

MDi ---> GI-UA-MTA

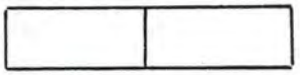
A) interprétation du message reçu

message = 

[A] [B]

si [A] = 0 : [B] est pour le GC

si [A] = 1 : il s'agit d'une demande de service

[B] = 

[C] [D]

si [C] = 1 : le MD signale au GI-UA-MTA qu'il a reçu un logoff

[D] = table register

si [C] = 2 : le MD signale qu'il pense qu'il n'a plus rien à faire [le MD doit être déconnecté]

[D] = nom machine dernier n° généré

si [C] = 3 : le MD demande un MD connecté à un UA local destinataire.

[D] =
id-UA PROBE DELIVER MPDU
ou
id-UA DELIVER MPDU
ou
id-UA NOTIFY param
ou
id-UA LOCAL DELIVER param id-UA-OR
ou
id-UA LOCAL PROBE DELIVER param id-UA-OR

si [C] = 4 : le MDi avertit le GI-UA-MTA qu'il a effectué la tâche (demande de service) demandée par le GI-UA-MTA

[D] = /

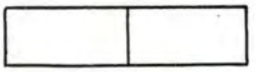
B) modifications à apporter au message avant de l'envoyer vers la destination demandée

si [A] = 0 : [A] est supprimé, sinon le message étant une demande de service, s'arrête au GI-UA-MTA

11.9.3. MESSAGES ARRIVANT AU MD

GI-UA-MTA ---> MD

Interprétation du message reçu

message = 

[A] [B]

si [A] = 0 :

le message [B] doit être interprété comme une primitive. Parmi ces primitives, on retrouve :

- les primitives existant entre UA et le MD
- deliver
- local deliver
- probe deliver
- local probe deliver
- notify
- demnotnl

si [A] = -1 :

le message vient du GI-MTA-MTA ou de l'AM et est une demande de création de notification de non livraison

[B] = AM/GAM DEMNOTNL

si [A] = NULL :

le message provient des couches inférieures de OSI via le RTS et l'AM. Il s'agit d'une demande de RELAIS = [B]

si [A] = 1 :

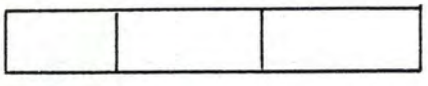
le message provient du GI-UA-MTA. Il peut être

id-ua table	message fournissant au MD toutes les caractéristiques de l'UA auquel il est connecté
destruction	message autorisant le MD à se détruire

11.9.4. MESSAGES ARRIVANT AU GI-MTA-MTA

GC ---> GI-MTA-MTA

A) interprétation du message reçu

message = 

[A] [B] [C]

si [A] = 0 : le message vient des couches inférieures du modèle OSI, pour un RTS

[B] = MTAvois

[C] = primitive

si [A] = 1 : le message, en provenance d'un MD via le GI-UTA-MTA, est destiné à un AM

[B] = MTAvois

[C] = ENVOI(MPDU)

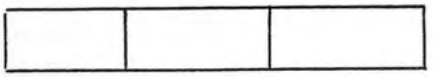
B) modifications à apporter au message avant de l'envoyer vers la destination demandée

si [A] = 0 : si le message peut être distribué à un RTS [cfr. XI.3], [B] est supprimé sinon le message reste au niveau du GI-MTA-MTA

si [A] = 1 : si le message peut être envoyé vers un AM [cfr.XI.3], [A] est mis à 0 et B est enlevé

AM ---> GI-MTA-MTA

A) interprétation du message reçu

message = 

[A] [B] [C]

si [A] = 0 : le message doit être envoyé au GC

([B][C]) = NULL RELAIS(MPDU)

ou

-1 AM DEMNOTNL(MPDU)

si [A] = 1 : le message est une demande de service

[B] = demande obstruction

[B] = 4

[C] est un message que l'AM ne peut traiter suite à une fin d'association.

si [A] = 2 : le message est destiné au RTS

B) modifications à apporter au message avant de l'envoyer vers la destination demandée

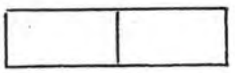
si [A] = 0 : A est mis à 1

si [A] = 1 : le message reste au niveau du GI-MTA-MTA

si [A] = 2 : /

RTS ---> GI-MTA-MTA

A) interprétation du message reçu

message = 

[A] [B]

si [A] = 0 : le message doit être donné au GC pour les couches inférieures du modèle OSI

[B] = MTAvois primitive

si [A] = 1 : le message est une demande de service

[B] = mettre bit à zéro

ou

demande de destruction dernier n° généré

ou

4 message : retour du message suite à une fin d'association.

si [A] = 2 : le message est destiné à l'AM

si [A] = 3 : le message est une demande de service devant être traitée immédiatement

[B] = boucher pipe

B) modifications à apporter au message avant de l'envoyer vers la destination demandée

si [A] = 0 : /

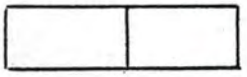
si [A] = 1 : le message reste au niveau du GI-MTA-MTA

si [A] = 2 : /

si [A] = 3 : le message est traité par le GI-MTA-MTA

11.9.5. MESSAGES ARRIVANT A L'AM

GI-MTA-MTA ---> AM

message = 

[A] [B]

si [A] = 0 : le message vient d'un MD via le GI-UA-MTA

[B] = ENVOI(MPDU)

si [A] = 1 : le message est .soit une réponse à un service demandé

[B] = destruction

ou

pipe bouché

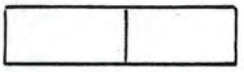
.soit l'adresse du MTA voisin avec lequel l'AM a établi une association

si [A] = 2 : le message vient du RTS

[B] = primitive

11.9.6. MESSAGES ARRIVANT AU RTS

GI-MTA-MTA ---> RTS

message = 

[A] [B]

si [A] = 0 : le message vient des couches inférieures de OSI

[B] = primitive

si [A] = 1 : le message = [B] provient du GI-MTA-MTA

et vaut * MTAvois (adresse du MTA voisin lié)

* dernier n° généré

(dernier n° généré par cette machine RTS pour identifier une connexion session)

* pipe bouché (signal avertissant le RTS que plus rien n'est écrit dans le pipe)

* s-refuse-connex-ind

(signal prévenant le RTS que plus aucune connexion session n'est attribuable)

* destruction (permission donnée au RTS de se détruire)

si [A] = 2 : le message vient de l'AM

[B] = primitive

XI.10. LISTE DES DEMANDES DE SERVICE

Voici la liste des demandes de service destinées au MD :

```
   0  deliver  MPDU
   0  local deliver   param  id-ua-or
   0  probe deliver  MPDU
   0  local probe deliver  param  id-ua-or
   0  notify  primitive
   0  demnotnl  MPDU
   0  demnotnl  param  id-ua-or
  -1  AM/GAM  demnotnl  MPDU
```

chapitre XII.

CHOIX PERSONNELS ET RESTRICTIONS ADOPTEES

Conformément à l'objectif fixé, dans le cadre de ce travail, une analyse approfondie de la norme X400 a débouché sur un essai de mise en oeuvre d'un MTA. Cependant, avec l'addition du parallélisme, les difficultés accrues ont fait en sorte de rendre obligatoires - plus encore que prévu - des simplifications et restrictions concernant certains services envisagés par la norme.

12.1. LE MTA EN TANT QUE "BOITE AUX LETTRES"

Ainsi, X400 propose que le MTA soit, en quelque sorte, en plus de son rôle de relayeur de messages ou agent de transfert, une boîte aux lettres pour les UA qui lui sont connectés. Ceci signifie que le MTA pourrait conserver, dans un espace contrôlé par lui, les messages, destinés à ses UA, qui ne peuvent leur être livrés. Les raisons de cette impossibilité de livraison pourraient dépendre d'une non disponibilité temporaire de l'UA [pas de réponse au MTL LOGON] ou encore de la non conformité du message aux restrictions imposées par l'UA [via un REGISTER ou un CONTROL], lesquelles, on l'espère, pourraient être ultérieurement modifiées par l'UA, dans un sens favorable pour les messages en attente.

Ce principe de boîte aux lettres, quoique très intéressant,n'a pas été envisagé dans ce travail, travail qui peut n'être considéré que comme un point de départ sur le long chemin à parcourir pour atteindre l'implémentation d'un MTA complet !

Cette simplification, doublée de la notion de parallélisme, demande, par voie de conséquence, l'ajout de nouvelles raisons d'échec à fournir à l'UA par l'intermédiaire du NOTIFY. En effet, la raison "busy" peut, dans le cadre particulier de ce travail, recouvrir plusieurs cas :

1) l'UA destinataire n'est pas connecté; le MTA qui le dessert tente d'établir une communication via un MTL-LOGON; l'UA répond qu'il est occupé. Le MTA n'offrant pas le service de boîte aux lettres, le message est perdu et une notification de non livraison est produite.

2) à un moment quelconque du transfert du message, une machine (un MD, ou un AM-RTS) est nécessaire pour continuer le traitement, mais aucune n'est disponible. Dans ce cas, le message est stoppé et l'origine est avertie par un NOTIFY négatif contenant, lui aussi, l'indication "busy".

3) un RTS sur la route, pour entamer un transfert vers un MTA voisin, ou pour tenter de recouvrir une ancienne activité s'occupant du transfert d'un message, génère un S-CONNECT-request. Celui-ci est immédiatement annulé par le GI-MTA-MTA faute de connexion session libre à ce moment-là. Dès lors, le message est perdu; c'est toujours la même raison "busy" qui est avancée dans le NOTIFY.

Il est important de remarquer que, dans un souci de plus grande vraisemblance dans le cadre d'une exploitation réelle, il serait bien plus intéressant de doter le MTA d'un système de réessai, moyennant time-out et compteur, évitant ainsi les stationnements infinis en attente d'un service dans les diverses situations présentées ci-avant. Ceci éviterait les NOTIFY un peu arbitraires.

12.2 NOMBRE D'ASSOCIATIONS SIMULTANEES ENTRE DEUX MTA ADJACENTS

Rappelons encore un des choix fondamentaux dans l'implémentation présentée ici : à un moment précis, une seule association entre deux MTA fixés est permise. Cette hypothèse pourrait être soulevée, mais sans elle s'ajoutent de lourds problèmes puisqu'il faut alors aiguiller des messages ayant même origine vers une machine AM-RTS parmi plusieurs, toutes liées au même MTA voisin. Encore faut-il être capable de savoir laquelle ! Cette difficulté n'est pas triviale puisque rien, dans une primitive véhiculée entre AM pairs ou entre RTS pairs, ne dit sur quelle association ou sur quelle connexion elle voyage. De plus, comme déjà expliqué auparavant, augmenter le nombre d'associations convergeant vers le même MTA diminue les chances d'ouvrir des associations avec des MTA non encore atteints, au vu du nombre maximal de machines disponibles.

12.3. REPONSE D'UN UA A SON MTA

Vis-à-vis de l'UA également, une certaine hypothèse a été posée : l'UA, lorsqu'il reçoit un MTL-LOGON, répond toujours (positivement ou négativement, mais dans tous les cas). Il ne peut pas rester muet. Si c'est le cas, cela signifie que l'UA est perdu pour le MTA. Et une intervention manuelle est conseillée, afin de récupérer le lien qui s'est coupé. Bien sûr, on pourrait imaginer, là aussi, un système de réessai.

12.4. TRAITEMENT DES MESSAGES EN ATTENTE DE LIVRAISON

Dans un tout autre domaine à présent, il faut aborder le problème du service de livraison différée que le MTA propose à ses UA. La norme X400 décrit des paramètres et primitives par l'intermédiaire desquels un UA peut demander ou annuler une livraison différée. La politique de stockage, de réveil,... n'est pas sujette à la normalisation, puisqu'interne au MTA. Dès lors, le message est - et c'est un choix - stocké dans le MTA desservant l'UA origine de la requête de livraison différée. Ceci impose l'existence d'une file d'attente dédicacée à cet effet et un réveil des messages dont le temps d'attente est expiré. Ce choix du MTA origine est guidé par le fait qu'il n'existe aucun élément de protocole P1 permettant d'exprimer une demande d'annulation de livraison différée (CANCEL) : impossible via P1, de rattraper un message déjà transféré plus loin sur sa route afin de le stopper. Et cela même si le délai exigé pour sa livraison est encore loin d'être écoulé. Il semble dès lors opportun de conserver le message à l'origine le plus longtemps possible afin de répondre positivement au CANCEL éventuel.

12.5.TRAITEMENT DES ERREURS PAR LE MTA

Il faut encore aborder le domaine des erreurs survenant en cours de traitement par le MTA. Ainsi, il a été décidé qu'un mauvais séquencement de primitives aurait pour conséquence la mise à l'écart des primitives fautives. Et en cela, c'est une continuation logique de l'optique adoptée dans les diagrammes d'états, où les enchaînements non prévus sont cas d'erreurs.

Pourtant il serait parfois profitable, en plus du rejet de la requête inattendue, d'avertir son émetteur. Après réflexion, ceci se révèle cependant quasi impossible. En effet, les erreurs viennent forcément de l'extérieur du MTA (forcément car on pose l'hypothèse que les modules internes réalisés selon les spécifications et algorithmes proposés agissent correctement). Elles proviennent donc de l'UA ou d'un MTA voisin. Dans le cas d'un MTA voisin, comment l'avertir ? Et surtout, qui avertir dans ce MTA dont on ne connaît que les entrées/sorties standardisées ? Si c'est l'UA qui guide mal le déroulement des opérations, il est impossible de le lui signaler sans ajouter de nouvelles primitives, de nouveaux types d'interactions entre MD et UA. C'est pourquoi c'est le rejet pur et simple qui a été retenu.

12.6. CONFORMITE DES MESSAGES A P1

Dans le même ordre d'idée, il faut insister sur le fait qu'un message arrivant de l'extérieur est toujours analysé en vue d'établir ou nier sa conformité par rapport à P1. Dans le cas où P1 n'est pas entièrement respecté, le message est oublié. Ceci représente une simplification dans le sens où peut-être le nom de l'origine pourrait-il quand même être compris, et dès lors, un rapport renvoyé jusqu'à l'UA origine.

12.7. GESTION DU "TOUR" POUR L'ENVOI DE MESSAGES

Si l'on aborde à présent des points plus techniques, il faut signaler que la politique d'attribution du tour choisie par l'AM lors de l'essai d'établissement d'une association est la suivante : X400 propose que celui qui ouvre l'association choisisse de garder ou de céder le tour. L'AM tel qu'il est envisagé dans l'architecture parallèle décrite ici, garde toujours le tour lorsqu'il active l'ouverture. Bien sûr, pour prévoir tous les cas, quand il passe du côté récepteur, notre AM prévoit la possibilité que l'autre lui donne le tour dès l'ouverture. L'option prise se base sur la discrétion de la norme quant à la description de situations réelles où un AM demande une association tout en se plaçant en position de récepteur. Sans doute en va-t-il ainsi dans un système de MTA où on introduit les notions de maître et esclave, l'esclave ne pouvant pas entamer la conversation mais seulement répondre au maître, via un système de polling.

12.8. ECHEC EN CAS DE RECOUVREMENT DE CONNEXION SESSION

La politique de non-réessai d'une requête qui échoue intervient de manière cruciale au niveau des demandes de transfert engagées par l'AM. Un AM, pour lequel est établie une association avec l'AM d'un MTA voisin, active, suite au désir d'un MD, la primitive TRANSFERT, qui ordonne au RTS d'assurer le transport fiable d'un message jusqu'au MTA visé. Le RTS, disposant d'une connexion session, s'exécute. Survient alors une rupture de connexion. Comme le prescrit la norme, à un moment qui

lui semble opportun [aucune précision supplémentaire n'apparaît dans X400], le RTS chargé du transfert tente de recouvrir l'activité et commence pour cela par un RECOVER stipulant qu'il veut garder les jetons. Il est très possible que cet essai échoue. Que faire alors du message interrompu et pour lequel une reprise semble impraticable, momentanément du moins ? L'algorithme construit montre que le RTS préposé au traitement du message bloqué avertit son AM, via un S-EXCEPTION-IND, des problèmes rencontrés et dès lors, détruit toute trace du message. Cette solution peut sembler un peu cavalière puisque le message est rayé de la liste alors que, remis en fin de file d'attente du RTS, il pourrait peut-être faire l'objet, plus tard, d'une nouvelle reprise, fructueuse cette fois. Cependant, comme toujours, des difficultés dues au déroulement non synchrone des AM et RTS apparaissent : si l'AM émetteur du TRANSFERT a terminé son interaction logique avec l'AM pair, il peut ordonner à son RTS de clôturer proprement l'association (CLOSE). Or, la file d'attente étant gérée selon une politique FIFO, s'il replace le message bloqué en fin de file d'attente, il se peut que le RTS soit amené à donner suite au CLOSE alors que des messages interrompus stationnent plus loin dans la file. Dès lors, l'association fraîchement abandonnée risque de laisser derrière elle des messages interrompus que seul un RTS peut comprendre et qui diparaîtront avec lui.

Finalement, pour la netteté du dialogue, il a paru plus sûr d'écarter immédiatement le message.

12.9. CHOIX DE L'ACTIVITE INTERROMPUE

Passant, de ce fait, au niveau du RTS, se présente alors tout l'aspect interruption-reprise. On le sait, la norme n'admet qu'une seule activité interrompue en attente de reprise. Si une nouvelle activité est à son tour interrompue, laquelle faut-il décider de conserver ? Au vu des renseignements dont il dispose, le RTS peut difficilement, pour choisir, appliquer une politique intelligente : l'une et l'autre demandes interrrompues se valent. La préférence a été portée, de manière assez arbitraire, sur la nouvelle requête interrompue, au détriment de celle déjà en attente. Bien sûr, il est possible d'imaginer des optiques différentes. Ainsi, il paraît intéressant de privilégier la demande qui est plus près du but (i.e. celle pour laquelle le volume de données en attente est le plus faible) paraît intéressant. D'autre part, si le RTS pouvait avoir accès aux priorités des messages, il semblerait tout naturel de tout miser sur le plus prioritaire. Hélas, rien dans l'enveloppe du message ou dans les paramètres de TRANSFER ne permet de tirer des conclusions à cet égard. Et c'est dans ce sens que l'on peut affirmer que l'optique choisie n'est pas plus mauvaise qu'une autre.

Comme expliqué ci-avant, la dernière activité interrompue est enregistrée et attend sa reprise, reprise laissée par la norme au bon gré du RTS. Encore faut-il fixer le moment où ce RTS va décider de remettre en jeu cette activité, en utilisant la primitive S-ACTIVITY-RESUME! En fait, c'est lorsqu'il reste sans tâche à remplir qu'il va tenter une reprise : lorsqu'il a achevé un transfert pour l'AM, et donc qu'il possède les jetons, s'il ne trouve plus, dans sa file d'attente, aucune demande émanant de son AM ou de son RTS pair, et s'il existe une demande en attente de reprise pour laquelle il était émetteur, il lance la procédure RESUME. De ce fait, finalement, le RTS privilégie le traitement des requêtes

nouvellement arrivées, vide ses pipes et sa file d'attente et évite l'engorgement de ceux-ci.

12.10. CONCLUSION

La liste des choix posés lors de la conception du MTA parallèle se termine ici; un bref rappel en est donné ci-dessous :

le MTA ne fait pas office de boîte aux lettres,

une seule association entre deux MTA voisins est permise à un instant donné,

un UA répond toujours à une demande de connexion initialisée par son MTA,

les messages touchés par une demande de livraison différée sont conservés le plus longtemps possible dans le MTA d'origine,

un mauvais séquencement de primitives entraîne la mise à l'écart des primitives malvenues,

un message non conforme à P1 est oublié, sans rapport de non livraison,

l'AM émetteur d'une demande d'établissement d'association conserve le tour,

en cas d'échec lors d'un essai de recouvrement de connexion session, le message pour la reprise duquel l'essai a été tenté est écarté, avec création d'un rapport de non livraison,

l'activité en attente de reprise conservée est toujours la dernière interrompue.

Ces choix sont souvent dus à un désir - et un grand besoin - de simplification. Au vu de la complexité du système à réaliser et puique ce travail n'est jamais qu'un point de départ pour l'implémentation réelle, ces choix sont sujets à la critique et à la modification.

Chapitre XIII.

IMPRECISIONS RELEVEES DANS LA NORME

13.1 INTRODUCTION

Bien que cela ait déjà été plusieurs fois mentionné au cours de ce rapport, il semble important de rassembler les points qui, dans la norme, ont posé problème, malgré les lectures répétées. Plutôt que d'erreurs - la norme, quoi qu'on en dise, est bien faite, quand on pense à la complexité du système qu'elle standardise - il s'agit d'imprécisions, de taille plus ou moins importante, qui parfois ne sont pas visibles du premier coup d'oeil, mais qui, après réflexion, font surgir des difficultés demeurées de temps en temps sans réponse. Une liste quelque peu détaillée, en est dressée ci-dessous.

13.2 CONVERSION ET TRANSPARENCE

Il convient avant tout de souligner un trait de caractère assez inattendu de la norme. X400 prévoit que le MTA se charge de conversions, demandées implicitement ou indispensables pour la réussite du transfert. Pour ce faire, il traduit évidemment le message. De cette façon, il accède au contenu du message sans plus se contenter de son enveloppe. Cette incursion semble nécessaire si le service de conversion est offert, mais ouvre tout de même le débat quant à la confidentialité.

13.3 CAS DES "INFORMATIONS BILATERALES"

Par ailleurs, outre l'effort à fournir lorsqu'il faut analyser le protocole P1 exprimé dans la norme selon la syntaxe X409, le caractère flou de certaines notions peut être dégagé. Ainsi, comme le lecteur peut le trouver au point 3.4.2.1 de la norme X411 décrivant l'enveloppe d'un User PDU, un champ très spécial apparaît : celui des informations bilatérales liées aux domaines. Il est stipulé qu'il s'agit là d'informations placées par le MD côté origine à l'intention d'un des MD sur le chemin à emprunter par le message. Elles peuvent être de tout type et adressées à des MD différents. Vu qu'aucun renseignement supplémentaire ne vient étoffer cette description, c'est sans trop savoir quelle est leur utilité qu'il faut permettre la réception et le stockage de telles données. Dans le cadre de l'architecture parallèle, cela signifie alors l'enregistrement de renseignements d'ordre général auxquels chacun des MD peut accéder. Il faut donc qu'un gestionnaire se charge de ces manipulations. c'est le GC qui a été choisi, cela simplement pour éviter la surcharge du GI-UA-MTA, dèjà bien sollicité alors que GC ne doit effectuer qu'un aiguillage automatique des messages. Bien sûr, plus d'éclaircissement à propos de ces informations bilatérales permettraient sans doute de mieux adapter notre choix aux traitements spécifiques qu'elles imposent.

13.4 NOTIFICATIONS DEMANDEES... PAR LE MTA !

Toujours au même endroit dans la norme, on peut trouver le champ "PerRecipientFlag" qui permet au MTA de déduire s'il doit construire une notification (c'est-à-dire un accusé de réception ou un rapport de non livraison)

respectivement pour le message qu'il vient de livrer ou d'arrêter sur sa route. Ce PerRecipientFlag présente, entr'autres, deux parties intitulées "Report Request" et "User Report Request". Elles semblent bien proches l'une de l'autre puisque chacune décrit quel service de notification l'origine a demandé. Pourtant, il faut trouver à chacune une spécifité propre. Après réflexion, il n'est pas impensable de conclure que le User Report Request garde trace de la requête de l'UA origine, véhiculée au départ grâce aux paramètres NDL Suppress, Content Return et Delivery Notice de la primitive SUBMIT. Le Report Request, quant à lui, représente les choix du MTA origine concernant les rapports de livraison. On apprend, dès lors, par cette petite note perdue dans le grand volume des recommandations, que le MTA demande des notifications. Plus même, il doit toujours être averti lorsqu'un message, dont il dessert l'UA origine, n'a pu être transmis (la cause en est que la valeur "00", dans le champ Report Request, ne peut être utilisée).C'est là une approche, un peu fugitive à notre goût, d'une démarche importante.

13.5 ABANDON D'ASSOCIATION

Si on délaisse le protocole P1 pour se tourner vers l'AM, certaines questions, là aussi, s'imposent. Il faut notamment insister sur le fait que la norme oblige l'initiateur d'un CLOSE à posséder le tour. Si bien que le récepteur, s'il désire stopper le dialogue, doit commencer par une demande de tour et l'attente de la cession de tour. Ceci vient du fait que la procédure de CLOSE, bien que confirmée, ne peut jamais être refusée. Dès lors, pour empêcher le récepteur de couper l'association alors que l'émetteur n'a pas terminé, il ne peut pas activer le CLOSE. Sans doute serait-il plus adapté de permettre à chacun des deux interlocuteurs d'émettre un CLOSE qui

devrait, par conséquent, pouvoir recevoir une réponse favorable ou défavorable.

La terminaison d'une association reste un aspect assez instable puisqu'il semble possible, a priori, de ne jamais pouvoir l'effectuer proprement. Ainsi, une association ne peut se clôturer que via une connexion session. Or, il peut y avoir coupure de connexion session. Par ce fait, pour répondre à la demande de CLOSE de l'AM, le RTS tente d'ouvrir une nouvelle connexion, et cela par l'intermédiaire d'un RECOVER. Dans le cas où il n'existe plus de connexion disponible, ou encore si les deux RTS ne réussissent pas à se mettre d'accord sur l'identificateur de l'ancienne session interrompue, le RECOVER est impossible et l'association reste en attente de clôture pendant un temps qui peut sans doute devenir important, voire illimité, puisque le RTS ne dispose d'aucun moyen pour avertir son AM de l'échec de sa requête.

13.6 PROBLEME DE TOUR

Déjà plusieurs fois auparavant, la référence aux procédures de demande et de cession de tour est apparue. La norme, dans les primitives qui les activent (TURN-PLEASE-REQ et TURN-GIVE-REQ), prévoit la notion de priorité: l'AM qui demande le tour ajoute une indication de la priorité estimée de sa demande. Rien dans la norme n'explique les renseignements qui lui permettent de décider de cette priorité. Peut-être est-elle fonction du nombre de messages qui sont en attente de transfert sur cette association, peut-être est-elle calquée sur la priorité des messages à envoyer. Mais l'AM ne connaît pas cette priorité car elle lui arrive codée, avec le message, sous forme d'un élément de donnée de protocole (PDU) de P1. Or, dans notre vision

de la découpe fonctionnelle du MTA, l'AM ne "comprend" pas P1: il reçoit un bloc d'information en provenance d'un MD et il utilise les services que lui offre le RTS pour tenter de le transférer vers un MTA voisin mais il n'analyse pas le bloc à transmettre; il le donne au RTS qui le découpe et le passe vers le site adjacent. Bien sûr, si on admet que l'AM est capable de consulter l'enveloppe, il peut plus facilement dégager la priorité à placer dans sa demande de tour.

Le problème est le même du côté de l'AM envoyeur lorsqu'il doit décider quand il convient de céder le tour, à la suite d'une demande de son AM pair. En fonction de la priorité qu'il attribue à son propre travail, il donne ou garde le tour. Il lui faut donc également pouvoir évaluer cette priorité.

Dans ce cadre, la norme laisse toute liberté à l'implémenteur. Ceci représente sans doute un souci d'adaptation de la solution aux besoins personnels, mais pour des gens inexpérimentés comme nous le sommes, cela pose de véritables problèmes de décision.

13.7 UTILITE DU USER-DATA DANS LA PRIMITIVE D'ETABLISSEMENT D'ASSOCIATION

Il est une autre imprécision qui peut être relevée au niveau de l'AM; lors de l'établissement de l'association, la primitive OPEN-REQ véhicule le paramètre User Data, dont le but et le contenu ne sont expliqués nulle part. Il est difficile dans ce cas et de le remplir et de l'analyser lorsqu'il est reçu.

13.8 CARENCES AU NIVEAU RTS

Si on passe à présent au niveau du RTS, on se trouve face à une partie de la norme extrêmement compliquée et parallèlement, hélas, très floue. Sans doute les imprécisions sont-elles souvent voulues, dans un désir d'éviter trop de rigueur dans un domaine local et finalement très personnalisé. Ainsi, un RTS qui est en difficulté au cours d'une activité, peut décider, selon sa logique propre, d'interrompre l'activité ou de l'abandonner définitivement. La norme ne propose pas de solution : pour elle, le RTS fait l'un ou l'autre. Et une fois de plus, le manque d'expérience pose véritablement un problème de jugement.

Mais en plus de ce "trop plein de liberté", on peut déceler aussi certaines absences à ce niveau. Ainsi, les primitives S-P-ABORT, S-U-ABORT, S-ACTIVITY-DISCARD, S-ACTIVITY-INTERRUPT sont employées dans de nombreuses occasions, sans que jamais elles soient détaillées ni même leurs paramètres seulement énumérés. Mieux encore, le paragraphe 4.6.2.1. de X410 contient, en remarque, une allusion à la primitive S-P-EXCEPTION-REPORT, sans rien expliquer à son propos. C'est là la seule fois où elle apparaît, nul ne peut d'ailleurs trop dire pourquoi.

Au moment de la clôture d'une connexion, le RTS utilise un S-RELEASE. Il semble que cette primitive contienne des paramètres, dont on ne dit rien sinon qu'ils existent, et dont il est difficile de cerner le rôle.

A un niveau plus logique, à présent, la technique du RECOVER paraît présenter plusieurs risques. En effet, un RTS, bien qu'il soit sans jeton, peut désirer reprendre une activité interrompue à un moment où il était détenteur des jetons. S'il y a eu,. depuis, coupure de connexion, il est indispensable de passer par l'intermédiaire d'un RECOVER pour rétablir une connexion. Mais lors du S-CONNECT, dans le cas d'un RECOVER, initialisé sans jeton, conformément au point 4.2.1. note 7 de X410, le RTS doit laisser le RTS pair décider de l'attribution des jetons. Si l'autre les garde, le RTS qui a activé le RECOVER ne pourra reprendre l'activité qu'à condition que son AM demande et obtienne le tour. De plus, si une nouvelle interruption survient, puisque seule la dernière activité interrompue est conservée, la première est perdue. Si bien que le RTS, malgré tous les efforts entrepris, ne pourra pas mener à bien le tranfert.

13.9 CONCLUSION

En guise de conclusion, il faut insister sur le fait que, malgré tous les "défauts" relevés dans la norme, il reste néanmoins certain que les recommandations peuvent être considérées comme une base de travail intéressante et extrêmement complète si on les compare à l'objet complexe qu'elles tentent de normaliser. Cependant il est vrai, elles sont souvent peu claires et soulèvent dès lors de nombreuses questions. Ces questions trouveraient sans doute plus aisément des réponses si nous possédions une plus grande maîtrise et une connaissance plus pratique du domaine particulier qu'est la communication à distance.

Chapitre XIV.

CRITIQUE DU TRAVAIL

14.1 EVALUATION GENERALE.

Une fois au terme de ce mémoire, arrive le moment décisif de l'évaluation du travail fourni au long de toute cette année.

Il s'agit, en fait, d'une mise au point, opérée d'un oeil critique, concernant les diverses démarches posées ainsi que les résultats obtenus; en cela, elle représente peut-être une étape presque aussi importante que la recherche elle-même.

A la lecture de ce rapport, il apparaît bien vite que l'objectif fixé au départ était un bel idéal à atteindre mais qu'il incluait un passage obligé par des chemins escarpés et des sols parfois mouvants. En effet, analyser une norme n'est pas une mince affaire. D'une part, les recommandations ne sont pas toujours limpides, inutile en cela de rappeler tout ce qui fait l'objet de développements dans les sections qui précèdent. Ainsi, X400 revêt une forme extrêmement technique et fort rébarbative pour qui n'a jamais abordé de tels écrits. D'autre part, malgré le volume imposant des textes à analyser, il est nécessaire de conserver une vue assez globale que pour en retirer l'essence qui permet de quitter la théorie et de proposer une application personnelle pratique. Ceci s'avère nécessaire, mais non pas suffisant puisqu'il faut, de par les exigences d'une normalisation, fouiller la norme jusque dans ses moindres recoins afin d'élaborer une solution extrêmement précise, de l'ordre du détail même, et respectant les spécifications du problème.

Dès lors, volume et complexité conjugués impliquent un travail éreintant et débouchent, malgré les efforts, sur

l'abandon progressif des objectifs finals et leur remplacement par un horizon plus à notre portée. C'est ainsi que, de l'implémentation de sous-systèmes emboîtés d'un MTA, le but de ce travail s'est doucement dirigé vers une conception très personnelle, mais non expérimentée sur machine. Cette évolution, à notre avis, n'est pas frustrante puisqu'elle nous a offert la possibilité d'affronter les difficultés qui surgissent dès qu'il faut quitter la théorie pour la pratique - pratique, même si elle reste de l'ordre du crayon et du papier - et puisqu'elle a permis le développement d'une solution personnelle assez complète.

Le caractère personnel de ce travail se situe, bien sûr, au niveau de l'introduction du parallélisme. Cependant, puisque codage et tests n'ont pu être entrepris, l'estimation de l'apport qu'il représente reste problématique. Toutefois, il est possible d'établir une liste des aspects qu'il semblerait intéressant d'améliorer. Cette liste répond d'ailleurs à celle des choix présentée au chapitre 13. En effet, les options adoptées en cours d'implémentation sont, en général, des simplifications par rapport aux services envisagés par la norme, simplifications qu'il conviendrait de dépasser dans le cadre d'une implémentation réelle et qui, dès lors, jouent le rôle d'autant de critiques négatives vis-à-vis du produit logiciel conçu.

14.2 AMELIORATIONS PROPOSEES.

Si l'on veut quitter le cadre d'un mémoire et produire un logiciel exploitable, il semble indispensable de lever certaines hypothèses posées en vue de simplifications. Ainsi, il paraît essentiel de doter un MTA d'un système de boîte aux lettres, lequel devrait occuper une place de choix dans un système de messagerie.

C'est là sans doute une des grandes lacunes de notre MTA. Néanmoins, dans l'architecture parallèle, ce système de stockage pose un problème certain. Qui peut prendre en charge la gestion de ces messages ? Est-ce le MD ? Si c'est le cas, un MD doit exister jusqu'à la livraison de tous les messages en attente dans la boîte aux lettres. Si bien que toutes les machines abstraites risquent vite d'être réservées, empêchant alors toute interaction avec un nouvel UA ou tout routage d'un message arrivant de l'extérieur. Dès lors, on pourrait imaginer que le MD, chargé de stocker un message pour son UA, le confie au GI-UA-MTA afin qu'il le place dans un endroit précis servant de boîte aux lettres. Le GI-UA-MTA, puisqu'il gère l'établissement des interactions UA-MTA, possède les renseignements nécessaires pour savoir à quel moment il faut reprendre les messages de la boîte aux lettres et les délivrer à l'UA auquel ils sont destinés. Cependant, ceci impose une charge supplémentaire à ce gestionnaire déjà fort sollicité par ailleurs. C'est pourquoi il semble préférable d'ajouter, pour remplir cette fonction, un nouveau module, un gestionnaire de boîte aux lettres qui serait chargé du stockage des messages pour tous les UA que le MTA dessert.

Il ne faut pas croire surtout que le logiciel que nous avons conçu écarte de nombreux services proposés par la norme. Cependant, il présente d'autres faiblesses. Ainsi, dans notre recherche, nous avons souvent choisi, dans un premier temps, une solution simple bien qu'un peu rapide, et cela volontairement, afin de permettre une approche progressive des difficultés.

C'est ainsi que, dans l'implémentation parallèle réalisée, dès que le MTA est surchargé, tous les messages qui lui arrivent sont perdus, après compte-rendu auprès de leur émetteur. C'est le cas lorsque chacune des machines abstraites MD est réquisitionnée par un UA ou pour le routage d'un message transmis via les couches inférieures OSI. Cette caractéristique rend le MTA peu crédible, mais peut assez facilement être dépassée. Dans un premier temps, la saturation des machines MD pourrait déclencher

une réponse négative automatique à toute demande d'interaction, que ce soit un LOGON d'un UA ou un OPEN de l'AM (car l'OPEN conduit naturellement à des transferts qui requièrent un MD pour routage). Cependant, cette amélioration n'est pas suffisante puisque de nouveaux messages à relayer peuvent parvenir via les associations existantes, sans pouvoir être traités faute de MD disponible. Il semble donc nécessaire d'ajouter une file spécialement prévue pour les messages en attente d'un MD pour routage. Si bien que, finalement, la meilleure solution semble être celle où une machine MD est dédicacée au traitement de tous les messages transmis par les MTA adjacents plutôt que la proposition faite lors de notre conception (réquisition d'un nouveau MD à chaque message demandant relais). Ce changement représente une amélioration certaine puisqu'il diminue considérablement le nombre de messages écartés et les notifications de non livraison trop rapides.

Le cas des messages à envoyer vers un MTA voisin via OSI est similaire: il faut disposer d'une association sur laquelle transmettre les messages, ou au moins d'un couple AM-RTS disponible et prêt à tenter d'établir une communication avec le MTA visé. Dès que tous les AM-RTS sont occupés, les messages qui doivent sortir sont arrêtés et une notification de non livraison est créée. Il serait, là aussi, préférable d'ajouter une file d'attente qui stockerait les messages jusqu'à la libération d'une machine.

Si l'on quitte à présent le domaine des défauts assez ponctuels, somme toute, on aborde alors le problème général du parallélisme.

14.3 LE PARALLELISME.

Conceptuellement, introduire le parallélisme paraît en soi une idée intéressante. C'est pourquoi, dans notre recherche, nous avons essayé de pousser le parallélisme au plus loin, à tous les niveaux possibles. Ceci implique, on l'a montré tout au long de ce travail, des dialogues fréquents entre les modules fonctionnels qui tournent de manière autonome. Ainsi, les échanges de messages et de tables se multiplient; si bien qu'une question cruciale se pose : du point de vue pratique, dans le cadre d'une exploitation réelle, le gain en temps et en disponibilité que le parallélisme offre aux utilisateurs compense-t-il l'effort à fournir pour la réalisation et le surplus de travail dû à l'aspect gestion ?

Dans la mesure où notre implémentation n'a pas été portée sur machine, il est bien difficile de répondre à cette question car le fossé entre la conception et la réalisation est grand. Ainsi, même du point de vue de l'utilisateur, là où il est évident qu'en théorie, le parallélisme se traduise par des avantages, il n'est pas certain que l'implémentation réelle réponde à toutes les attentes. En effet, plus le nombre de machines MD augmente, plus le GI-UA-MTA doit gérer des informations diverses, plus il y passe du temps et moins vite les UA sont mis en relation avec le MD qui leur est attribué. Et le problème se pose d'une manière toute semblable au niveau des interactions avec les MTA voisins. Dès lors, il serait extrêmement intéressant de prévoir des tests dont les résultats pourraient nous éclairer quant à l'apport pratique du parallélisme.

14.4 TACHES LAISSEES A L'IMPLEMENTEUR.

Avant d'en arriver aux tests, il est indispensable de passer par l'étape de codage. Et pour cela, les renseignements contenus dans ce rapport n'offrent pas une base suffisante puisque certains aspects assez techniques ont été écartés, et laissés au bon soin d'un éventuel implémenteur.

D'une part, il n'est plus besoin de rappeler que les modules fonctionnels sont réalisés grâce à l'activation de primitives. Celles-ci apparaissent dans la norme et voient leurs paramètres détaillés de manière précise. Mais il reste encore à trouver une représentation adéquate pour ces primitives, représentation qui devrait permettre au mieux leur création, leur transmission via les pipes et leur interprétation.

D'autre part, seul le niveau 7 de la structuration hiérarchique du système, soit le niveau des modules fonctionnels (cfr 8.2), a été traité en profondeur. Les spécifications, les tables et les diagrammes d'état, les algorithmes en pseudo-langage sont autant d'outils mis à la disposition de l'implémenteur. Faute de temps, les autres niveaux ne sont exposés que de manière très superficielle. Pourtant, il est important de préciser que pour la réalisation des niveaux 2 et 3, le mémoire de Philippe Brossel [PBR 87] représente une source intéressante puisqu'il propose un outil capable de produire et d'analyser des unités de données de protocole pour un protocole codé selon la syntaxe formelle X409, ce qui est le cas de P1. L'utilisation de cet outil permettrait de résoudre les problèmes de traduction selon X409.

14.5 EN RESUME.

Il reste à présent à rappeler que le travail que nous avons réalisé, malgré toutes les difficultés qu'impliquent la lecture et la compréhension d'une norme, malgré le manque d'expérience pratique dans le domaine, a conduit à une analyse fouillée de X400. A cela s'ajoute un essai de mise en oeuvre du MTA qui, bien qu'elle reste conceptuelle, n'en est pas moins intéressante puisqu'elle place le MTA dans un contexte nouveau et qui mérite, à notre avis, quelqu'attention : le parallélisme.

Chapitre XV.

CONCLUSION

La messagerie électronique, nouvel outil de communication alliant les moyens informatiques et les possibilités offertes par les réseaux, permet à ses utilisateurs d'échanger des messages de tout type. Or, à notre époque, l'échange d'informations peut être considéré comme vital. C'est pourquoi les systèmes proposant des services de traitement de messages prennent de plus en plus d'importance. Et dans ce cadre, puisqu'elle travaille à la standardisation avec le souci d'établir un système de messagerie mondial, la norme X400 publiée en 1984 par le CCITT semble digne d'intérêt.

Les recommandations X400 décrivent un modèle fonctionnel qui devrait aider à la normalisation des échanges, par l'intermédiaire de l'utilisation d'un protocole de communication défini de manière très précise.

Le modèle proposé par X400 se base sur les interactions de deux entités, l'agent de l'utilisateur (UA) et l'agent de transfert des messages (MTA), l'un facilitant les contacts entre l'utilisateur et le système X400, l'autre plus spécialement chargé de la manipulation des messages (i.e. l'envoi, le relais et la livraison). C'est à l'étude du MTA que notre travail s'est attaché. Il en résulte, après un an de recherche, une analyse fouillée de la norme X400 ainsi que des spécifications complètes de cette entité.

Puisque le système à décrire inclut un aspect "enchaînements d'actions" et un aspect "historique du système" qui font également l'objet des spécifications, plusieurs formalismes, à savoir les spécifications Pre-Post, les tables d'états et les diagrammes d'états, sont employés en vue d'aboutir à une description complète du fonctionnement de l'entité à construire.

Lors de l'étude des fonctions prescrites par la norme, une nouvelle vision du MTA s'est imposée. C'est ainsi qu'un MTA intégrant la notion de traitements parallèles a été conçu et spécifié. Pour ce faire, le formalisme des automates finis a permis la spécification du MTA "parallèle" par le biais de trois machines abstraites autonomes.

Par la suite, une nouvelle architecture a été construite. Elle permet l'introduction du parallélisme en s'appuyant sur les outils offerts par le système d'exploitation Unix version 7.

En vue d'une implémentation future, des algorithmes en pseudo-langage ont été écrits, dictant le séquencement des actions à réaliser par les modules fonctionnels de l'architecture parallèle adoptée.

Il reste à rappeler que le travail effectué n'inclut pas le codage. Cependant, il peut, à notre sens, faire office de guide pour un implémenteur éventuel.

On ne peut conclure une étude comme celle-ci sans mettre en évidence l'évolution rapide qui touche les techniques informatiques en général et les télécommunications en particulier. C'est ainsi que le CCITT poursuit sa recherche concernant la normalisation de la messagerie électronique. De nouvelles recommandations sont attendues pour 1988. Après avoir parcouru la deuxième version des documents provisoires, datant du mois de mai 1987, il semble que des changements substantiels doivent être signalés [DFT X400 87]. Ainsi, X400-88 propose l'ajout de nouvelles entités fonctionnelles :

* le "Message Store", qui fera office de boîte aux lettres, jouant le rôle de tampon entre le MTA et ses UA,

* les "Acces Units", qui seront des unités d'accès spécialisées permettant l'utilisation du modèle MHS depuis d'autres systèmes ou services de communication comme par exemple tous les services télématiques,

* la "Physical Delivery Acces Unit", qui sera une unité d'accès particulière autorisant un utilisateur à demander la livraison d'un message à un abonné d'un service de livraison physique comme le système postal.

Les recommandations à paraître feront aussi mention d'un nouveau service : le service de "directories" qui devrait aider à identifier de manière plus agréable un utilisateur quelconque du MHS ainsi que faciliter la création de listes de distribution.

Parallèlement à ces fonctions nouvelles, la norme 88 prévoit l'abandon de la découpe du MTA en sous-couches et leur remplacement par la notion d'éléments de services. Cependant cette vision différente du MTA requiert, pour sa compréhension, une analyse approfondie. Celle-ci peut constituer à elle-seule un travail de longue haleine et ouvre, dans le domaine de la messagerie électronique, un nouvel horizon.

RESPONSABILITE POUR LA REDACTION DES CHAPITRES.

Chapitre I : Marie-Elise Angelot
Thierry Delroisse
Christine Franssens

Chapitre II : Christine Franssens

Chapitre III : Marie-Elise Angelot

Chapitre IV : Thierry Delroisse

Chapitre V : Marie-Elise Angelot

Chapitre VI : Marie-Elise Angelot
Thierry Delroisse
Christine Franssens

Chapitre VII : Thierry Delroisse

Chapitre VIII : Marie-Elise Angelot

Chapitre IX : Thierry Delroisse

Chapitre X : Marie-Elise Angelot

Chapitre XI

XI.1 : Marie-Elise Angelot
Thierry Delroisse
Christine Franssens

XI.2 : Christine Franssens

XI.3 : Thierry Delroisse

XI.4 : Christine Franssens

XI.5 : Thierry Delroisse

XI.6 : Christine Franssens

Chapitre XII : Marie-Elise Angelot

Chapitre XIII : Marie-Elise Angelot

Chapitre XIV : Marie-Elise Angelot
Thierry Delroisse
Christine Franssens

Chapitre XV : Marie-Elise Angelot
Thierry Delroisse
Christine Franssens

Il est bien entendu que cette répartition ne concerne pas le travail d'analyse de la norme et de conception qui a été réalisé d'un commun accord tout au long de l'année.

LISTE DES ABREVIATIONS.

ADMD : ADministration Management Domain
(domaine de gestion d'une administration)

AM : Association Manager
(gestionnaire des associations)

$\overline{AM}$: pipe allant de l'AM au GI-MTA-MTA

$\underline{AM}$: pipe allant du GI-MTA-MTA à l'AM

CCITT : Comité Consultatif pour la Télégraphie et la Téléphonie

GC : Gestionnaire Central

GI-MTA-MTA : Gestionnaire d'Interaction MTA-MTA

GI-UA-MTA : Gestionnaire d'Interaction UA-MTA

ISO : International Standards Organization
(Organisation pour les Standards Internationaux)

MD : soit Management Domain (domaine de gestion)
soit Message Dispatcher (gérant des messages)
(suivant le contexte)

$\overline{MD}$: pipe allant du Message Dispatcher vers le GI-UA-MTA

$\underline{MD}$: pipe allant du GI-UA-MTA vers le Message Dispatcher

MHS : Message Handling System
(système de traitement de messages)

MPDU : Message Protocol Data Unit
(unité de données de protocole pour la messagerie)

MTA : Message Transfer Agent
(Agent de transfert de messages)

MTAE : Message Transfer Agent Entity
(entité de l'agent de transfert de messages)

MTL : Message Transfer Layer
(niveau du transfert des messages)

MTS : Message Transfer System
(système de transfert de messages)

OSI : Open Systems Interconnection
(interconnexion de systèmes ouverts)

P1 : Protocole de transfert de messages

P2 : Protocole entre UA

P3 : Protocole d'envoi et de remise

PDU : Protocol Data Unit
(unité de données de protocole)

PRMD : PRivate Management Domain
(domaine de gestion privé)

RTS : Reliable Transfer Server
(agent de transfert fiable)

$\overline{\text{RTS}}$: pipe allant d'un RTS au GI-MTA-MTA

$\underline{\text{RTS}}$: pipe allant du GI-UA-MTA à un RTS

SDE : Submission and Delivery Entity
(entité d'envoi et de remise)

SMPDU : MPDU de service (sonde ou notification)

UA : User Agent
(agent de l'utilisateur)

UAE : User Agent Entity
(entité de l'agent utilisateur)

UAL : User Agent Layer
(niveau de l'agent utilisateur)

UAPDU : User Agent Protocol Data Unit
(unité de donnée de protocole de l'agent utilisateur)

UMPDU : MPDU d'usager

BIBLIOGRAPHIE

[AVL 87] :

A. VAN LAMSWEERDE, Notes personnelles relatives au cours de méthodologie de développement de logiciels, Facultés Universitaires Notre-Dame de la Paix Namur, 1987.

[DFT X400 87] :

"CCITT Draft Recommendation X400, Message Handling System and Service Overview", CCITT Study Group I, Q13/I Rapporteur Group, Version 2, mai 1987.

[GVB 83] :

G. VON BOCHMANN, "Concepts for Distributed Systems Design", Springer - Verlag Berlin, Heidelberg, 1983.

[PBR 87] :

P. BROSSEL, "Contribution à la mise en oeuvre d'un logiciel X400 : utilisation des langages formels de spécification et réalisation d'un outil d'analyse et de production d'unités de données de protocole", Mémoire présenté pour l'obtention du titre de licencié et maître en informatique, Facultés Universitaires Notre-Dame de la Paix, Namur, septembre 87.

[PVB 86] :

P. VAN BASTELAER, "Notes pour un cours de téléinformatique", Facultés Universitaires Notre-Dame de la Paix, Namur, 1986.

[REC X420 84] :

Recommendation X420, Data Communication Networks Message Handling Systems : Interpersonnal Messaging User Agent Layer, Red Book VIIIth plenary assembly Malaga Torremolinos 8-19 october 1984, Genève 1985.

[REC X430 84] :

Recommendation X430, Data Communication Networks Message Handling Systems : Acces Protocol for Teletex Terminals, Red Book VIIIth plenary assembly Malaga Torremolinos 8-19 october 1984, Genève 1985.

[SCO 86] :

B. SCOYER, "Réalisation d'une maquette de réseau mettant en oeuvre la norme ISO X25 et la norme transport X214", rapport interne, Institut d'informatique, Facultés Universitaires Notre-Dame de la Paix, Namur, Août 1986.

[SES 84] :

"Information Processing Systems - Open System Interconnection - Basic Connection Oriented Session Server Definition", DIS 8326, ISO/TC97/SC16, 1984.

[TAN 81] :

A.S. TANENBAUM, "Computer Networks", Prentice-Hall, Englewood Cliffs, 1981.

[UNI 84] :

H. LUCAS, B. MARTIN, G. DE SOBLET, "UNIX, mécanismes de base, langage de commande, utilisation", Eyrolles, Paris, 1984.

[REC X400 84] :

Recommendation X400, Data Communication Networks Message Handling Systems, Red Book VIIIth plenary assembly Malaga Torremolinos 8-19 october 1984, Genève 1985.

[REC X408 84] :

Recommendation X408, Data Communication Networks Message Handling Systems : Encoded Information Types Conversion Rules, Red Book VIIIth plenary assembly Malaga Torremolinos 8-19 october 1984, Genève 1985.

[REC X409 84] :

Recommendation X409, Data Communication Networks Message Handling Systems : Presentation Transfer Syntax and Notation, Red Book VIIIth plenary assembly Malaga Torremolinos 8-19 october 1984, Genève 1985.

[REC X410 84] :

Recommendation X410, Data Communication Networks Message Handling Systems : Remote Operations and Reliable Transfer Server, Red Book VIIIth plenary assembly Malaga Torremolinos 8-19 october 1984, Genève 1985.

[REC X411 84] :

Recommendation X411, Data Communication Networks Message Handling Systems : Message Transfer Layer, Red Book VIIIth plenary assembly Malaga Torremolinos 8-19 october 1984, Genève 1985.

[YLR 86] :

Y. LE ROUX, "La normalisation dans le domaine de la téléinformatique", Actes du congrès : De nouvelles architectures pour les communications : impacts et enjeux de la normalisation, Paris, 1986.

FACULTES
UNIVERSITAIRES
N.D. DE LA PAIX

NAMUR

INSTITUT D'INFORMATIQUE
RUE GRANDGAGNAGE, 21, B - 5000 NAMUR (BELGIUM)

Spécifications d'un agent de transfert de messages dans le cadre d'une messagerie électronique de type X400 : définition, conception et intégration de traitements parallèles en vue d'une implémentation sous Unix version 7.

Mémoire présenté par

Marie-Elise ANGELOT
Thierry DELROISSE
Christine FRANSSENS

en vue de l'obtention

du titre de

Licencié et Maître en Informatique

Promoteur : Ph. van BASTELAER

Année académique 1986-1987

ANNEXE.

Annexe 1

SPECIFICATIONS EXTERNES D'UN MTA

1.1. INTRODUCTION

Cette annexe contient les spécifications ainsi que les tables et diagrammes d'états de toutes les primitives permettant la communication entre l'UA et le MTA, exception faite des primitives permettant l'établissement de la connexion qui se trouvent au chapitre VI.

Pour permettre une compréhension plus facile des types utilisés, des noms parlants ont été définis :

SUBMIT-EVENT-ID : integer.

PROBE-EVENT-ID : integer.

DELIVER-EVENT-ID : integer.

CONTENT-ID : integer.

ADDRESS-ZONE : integer.

PASSWORD : string.

NAME : string.

TIME : string dont la structure est "JJ/MM/AA (date) - HH/MM/SS(heure).

O/R-NAME : string dont la structure est détaillée en annexe.

ADRESSE-UA : string.

Le type O/R-NAME permet de respecter le format personnel qui a été défini pour les noms O/R.

Le type ADDRESS-ZONE permet de donner l'adresse de la zone où a été stockée la valeur du paramètre. Ce type a été créé principalement pour caractériser les paramètres de longueur variable.

Suivent ci-dessous les spécifications pré-post des primitives et les tables d'états.

1.2. LES SPECIFICATIONS PRE-POST

SERVICE : ACCES TERMINATION.

BUT :

permettre à l'UA de terminer une interaction avec le MTA.

SPECIFICATIONS :

LOGOFF.

arguments

préconditions

résultats

posrtconditions

SERVICE : REGISTRATION SERVICE.

BUT :

permettre de changer la valeur des paramètres d'enregitrement imposés au MTA. Ces changements restent effectifs indéfiniment à moins qu'ils ne soient modifiés par une nouvelle primitive Register.

SPECIFICATIONS :

REGISTER

Arguments :

Deliverable - Encoded - Information - Types ϵ ADDRESS - ZONE

(Deliverable - Encoded - Information - Types est l'adresse d'une zone contenant les types d'informations encodées acceptés par l'UA, y compris la valeur "undefined").

Maximum - Content - Lenght ϵ INTEGER

(Maximum - Content - Lenght est la longueur du plus long message qui sera accepté par l'UA. Cette valeur représente un nombre d'octets).

Default - Control - Settings ϵ ADDRESS - ZONE

{Default - Control - Settings est l'adresse d'une zone qui contient les valeurs par défaut des paramètres que l'on peut modifier grâce à une primitive Control. Ces valeurs prennent effet directement après le logon et restent valables jusqu'à ce qu'elles soient modifiées par une primitive Control ou MTL-Control-Sign}.

Address ϵ ADRESSE - UA

{address est l'adresse réseau ou transport que l'UA donne au MTA si celle-ci a changé}.

O/R - Name ϵ OR - NAME

{O/R - Name est le nom o/r de l'UA si celui-ci a changé}.

Précondition

Tous ces arguments sont facultatifs.

Résultats

Success - Indication ϵ {true, false}

{indication du succès ou de l'échec du changement des valeurs des paramètres d'enregistrement}.

Failure - Reason ϵ ADDRESS - ZONE

{Failure - Reason est l'adresse d'une zone contenant les raisons de l'échec}.

Postconditions

Failure - Reason est un argument facultatif. Sa présence dépend de la valeur de l'argument obligatoire "Success - Indication".

si Success - Indication = true => Failure Reason est absent.

si Success - Indication = false => Failure Reason est présent.

Soient les conditions suivantes:

C1: Maximum - Content - Lenght < la valeur par défaut du Control, exprimant la longueur maximale du contenu que l'UA accepte, valeur contenue dans la zone d'adresse Default - Control - Settings.

C2: les valeurs exprimant les valeurs par défaut du Control, des types que l'UA peut accepter (ces valeurs se trouvant dans une zone d'adresse Default - Control - Settings), ne sont pas toutes reprises dans la zone d'adresse Deliverable - Encoded - Information - Types.

C3: Deliverable - Encoded - Information - Types est une adresse inconnue.

C4: Maximum - Content -Lenght n'est pas un entier strictement positif.

C5: Default - Control - Settings est une adresse inconnue.

C6: les valeurs contenues dans la zone d'adresse Deliverable - Encoded - Information - Types ne sont pas des types d'informations encodées.

C7: les valeurs contenues dans la zone d'adresse Default - Control - Settings ne représentent pas des valeurs possibles des paramètres que l'on peut modifier grâce à une primitive Control.

Success - Indication = ¬ (C1 ∨ C2 ∨ C3 ∨ C4 ∨ C5 ∨ C6 ∨ C7)

c'est-à-dire

Success - Indication = true <=> ∀i i ∈ [1...7] Ci fausse.

Success - Indication = false <=> ∀ i i ∈ [1...7] Ci vraie.

Failure - Reason est l'adresse d'une zone contenant un ou plusieurs entiers:

zone ∋ 1 <=> C1 ∧ ¬C4 ∧ ¬C7 est vraie.

∋ 2 <=> C2 ∧ ¬C3 ∧ ¬C6 ∧ ¬C7 est vraie.

∋ i <=> Ci est vraie i ∈ {3,4,5}

∋ 6 <=> C6 ∧ ¬C3 est vraie.

∋ 7 <=> C6 ∧ ¬C5 est vraie.

SERVICE : HOLD FOR DELIVERY

BUT:

permettre de changer les restrictions qui contrôlent les messages que le MTL peut envoyer à l'UA. Les nouvelles valeurs remplacent les précédentes et restent valables jusqu'à un logoff, une autre primitive Control ou une primitive Register plus restrictive (c'est-à-dire portant sur ces arguments).

Ces nouvelles valeurs ne s'appliqueront qu'à des messages ayant passé sans problème les restrictions imposées par la primitive Register.

Les messages ne pouvant être livrés à un UA à cause des valeurs déterminées par la primitive Control seront retenus dans le MTA.

SPECIFICATIONS:

CONTROL

arguments :

Permissible - Encoded - Information - Types ϵ ADDRESS - ZONE

{Permissible - Encoded - Information - Types est l'adresse d'une zone contenant les types d'informations encodées possibles que le MTA peut envoyer à l'UA}.

Maximum - Content - Lenght ϵ INTEGER

(indication de la longueur du plus long message que l'UA peut accepter du MTA. Cette valeur représente un nombre d'octets).

Lowest - Priority ϵ ("urgent","non urgent","normal")

(Lowest - Priority indique la priorité du message le moins urgent que l'UA est prêt à recevoir).

Messages ϵ (true)

(indique que le MTL peut envoyer des messages à l'UA).

Notifications ϵ (true)

(indique que le MTL peut envoyer des notifications à l'UA).

Préconditions

Tous ces arguments sont facultatifs.

Résultats.

Success - Indication ϵ (true, false)

(indication du succès ou de l'échec du changement des valeurs des paramètres d'enregistrement provisoires).

Failure - Reason ϵ ADDRESS - ZONE

(Message - Hold est l'adresse d'une zone contenant les notifications et les messages retenus dans le MTA à cause des restrictions imposées par la primitive Control).

Postconditions

Messages - Hold et Failure - Reason sont des arguments facultatifs. Ils ne sont présents que si l'argument "Success - Indication" qui lui est obligatoire, a la valeur "false".

Soient les conditions suivantes:

C1: Permissible - Encoded - Information - Types est une adresse inconnue.

C2: Les valeurs contenues dans la zone d'adresse Permissible - Encoded - Information - Types ne sont pas des types d'informations encodées possibles.

C3: Les valeurs contenues dans la zone d'adresse Permissible - Encoded - Information - Types ne forment pas un sous-ensemble de l'ensemble des types permis contenus dans la zone d'adresse Delivrable - Encoded - Information - types donnée comme paramètre de la primitive Register.

C4: Maximum - Content - Lenght n'est pas un entier strictement positif.

C5: Maximum - Content - Lenght n'est pas plus petit ou égal à la valeur donnée comme argument de la primitive Register et elle aussi exprimant la longueur maximale du contenu que l'UA peut accepter.

Success - Indication = $\neg$ (C1 $\vee$ C2 $\vee$ C3 $\vee$ C4 $\vee$ C5)

c'est-à-dire

Success - Indication = true $\Leftrightarrow \forall i \; i \in [1...5]$ Ci fausse.

Success - Indication = false <=> ∀i i ∈ [1...5] Ci vraie.

Failure reason est l'adresse d'une zone contenant un ou plusieurs entiers

zone ∋ i <=> Ci est vraie i ∈ {1,2,3}.

∋ 3 <=> ¬C1 ∧ ¬C2 ∧ C3 est vraie.

∋ 5 <=> ¬C4 ∧ C5 est vraie.

Messages - Hold est l'adresse d'une zone contenant les notifications et leur longueur en octets ainsi que les messages avec leur priorité et leur longueur en octets qui attendent d'être livrés.

SERVICE : MTL - CONTROL

BUT:

Le MTA active ce service pour informer l'UA des messages qui seront acceptés par le MTA.

Ces indications remplacent les précédentes et restent valables jusqu'à un logoff ou une autre MTL-Control- Sign.

SPECIFICATIONS:

MTL-CONTROL-SIGN

Arguments

Maximum - Content - Lenght ϵ INTEGER.

(Maximum - Content - Lenght indique le nombre d'octets maximum des messages que l'UA pourra envoyer au MTA).

Lowest - Priority ϵ ("urgent", "non urgent", "normal").

(Lowest - priority indique la plus petite priorité que pourront avoir les messages acceptés par le MTA).

Probes ϵ (true)

(Probes indique si le MTA peut recevoir des probes envoyés par les UA).

Messages ϵ (true)

(Messages indique si l'UA peut envoyer des messages vers le MTA).

Préconditions

Tous ces arguments sont facultatifs.

Maximum - Content - Lenght est un entier strictement positif.

resultats

postconditions

MTL-CONTROL-RESP

Argument

Messages - Hold ϵ ADDRESS - ZONE

(Messages - Hold est l'adresse d'une zone qui contient les probes et les messages en attente dans l'UA dû aux restrictions imposées par le MTL - Control - Sign).

Préconditions

Cet argument est facultatif.

Message - Hold est l'adresse d'une zone connue.

Les éléments contenus dans la zone d'adresse Message - Hold seront des notifications avec leur longueur et des messages avec leur longueur et leur priorité.

Résultats

Postconditions

SERVICE : MESSAGE SUBMISSION

BUT:

L'UA demande à son MTA le transfert d'un message vers un ou plusieurs UA destinataires au moyen de la primitive SUBMIT. Une notification de non livraison lui sera remise par la

commande "notify" si le message ne peut arriver à son destinataire et s'il n'a pas refusé explicitement cette notification de non livraison. Un message peut ne pas arriver à son destinataire s'il y a un problème entre deux MTA (problème de communication), dans un MTA intermédiaire (circuit, incompatibilité de types,...), et dans le MTA destinataire (Control, Register,...). Une notification de livraison pourra être retournée à l'UA émetteur s'il l'a requise et si le message est livré à l'UA destinataire. Certains services, tels que "disclose recipient", "alternate recipient", sont à la disposition de l'UA envoyeur s'il y est abonné.

SPECIFICATIONS:

SUBMIT

Arguments

Récipient ∈ ADDRESS - ZONE.

{un ou plusieurs destinataires}.

Originator - O/R-Name ∈ O/R-Name.

{nom o/r de l'origine}.

Content ∈ ADDRESS - ZONE.

{longueur et contenu du message}.

Content - Type ∈ {"P2"}.

{type de contenu de message: P2 dans le cadre d'un message de personne à personne}.

Encoded - Information - Type ∈ ADDRESS -ZONE.

{type(s) d'information contenue dans le message à tester}

Ndn - Suppress ∈ ADDRESS -ZONE.

{demande qu'aucune notification de non livraison ne soit renvoyée}.

Priority ∈ {"urgent", "non urgent", "normal"}.

{sélectionne la priorité de la livraison}.

Deferred - Delivery - Time ∈ TIME.

{spécifie le moment au delà duquel le message ne doit plus être livré}.

Delivery - Notice ∈ ADDRESS -ZONE.

{demande qu'une notification de livraison soit renvoyée}.

Conversion - Prohibited ∈ {true}.

{demande qu'aucune conversion ne soit effectuée sur le message}.

Disclose - Recipients ∈ {true, false}.

{indique si les noms des destinataires doivent être révélés aux autres}.

Alternate - Recipient - Allowed ∈ {true}.

{indique que le message pourra être livré à un destinataire "poubelle", si l'adresse du destinataire n'est pas correcte au bit près}.

Content - Return ϵ {true}

{indique que le contenu du message doit être renvoyé dans chaque notification de non-livraison}.

UA - Content - Id ϵ CONTENT - ID.

{identificateur du contenu du message généré par l'UA}.

Explicit - Conversion ϵ ADDRESS - ZONE.

{indique qu'une conversion spécifiée doit être effectuée}.

Préconditions

Recipient, Originator - O/R-Name, Content, Content - Type, Priority, Disclose - Recipients sont des arguments obligatoires. Les autres sont facultatifs.

Recipient:

adresse d'une zone préalablement garnie par l'UA, Contenant la liste des noms O/R des destinataires du Submit. Un destinataire sera repéré par son ordre d'apparition dans cette liste (1,2,3,...).

Content:

adresse d'une zone préalablement garnie par l'UA, renfermant la longueur du contenu du message ou le message lui-même.

Encoded - Information - Type:

adresse d'une zone préalablement garnie par l'UA, contenant le(s) type(s) du contenu de message à transférer. Il est utile pour tester des incompatibilités de types.

Ndn - Suppress:

adresse d'une zone préalablement garnie par l'UA, contenant les numéros de référence des destinataires pour lesquels l'UA émetteur ne souhaite pas de notification de non livraison.

Delivery - Notice:

adresse d'une zone préalablement garnie par l'UA, contenant les numéros de référence des destinataires pour lesquels l'UA émetteur souhaite une notification de livraison.

Explicit - Conversion:

adresse d'une zone préalablement garnie par l'UA, contenant, pour certains destinataires précisés dans "Récipient", l'indication du fait qu'on demande pour eux le service "Explicit conversion". Cette zone contiendra la liste des numéros d'ordre des destinataires concernés par ce service, et pour chacun, le type dans lequel il faudra convertir.

Résultats

Success - Indication ∈ {true, false}.

Submission - Time ∈ TIME.

Submission - Event - Id ∈ SUBMIT - EVENT - ID.

Failure - Reason ∈ ADDRESS -ZONE.

UA - Content - Id ∈ CONTENT - ID.

Postconditions

soient les conditions suivantes:

C1:"Récipient" est une adresse inconnue.

C2:le nom O/R d'un destinataire est incorrect.

C3:le nom O/R de l'origine est inconnu [ce n'est pas celui de l'UA concerné].

C4:"Content" est une adresse inconnue.

C5:"Content - Type" n'a pas la bonne valeur.

C6:"Encoded - Information - Type" est une adresse inconnue.

C7:le type est inconnu.

C8:le paramètre Ndn - Suppress est incorrect [incompatibilité de types].

C9:le paramètre Priority est incorrect.

C10:le paramètre Deferred - Delivery - Time est incorrect.

C11:le paramètre Conversion - Prohibited est incorrect.

C12:"Disclose - Récipient" est une adresse inconnue.

C13:Le paramètre Alternate - Recipient - Allowed est incorrect.

C14:le paramètre Content - Return est incorrect.

C15:le paramètre UA - Content - Id est incorrect.

C16:"Explicit - Conversion" est une adresse inconnue.

C17:un des types précisés est inconnu.

C18:un des numéros d'ordre identifiant un destinataire est inconnu.

C19:la souscription au service "Alternate Recipient Allowed" n'a pas été demandée.

C20:la souscription au service "Explicit Conversion" n'a pas été demandée.

C21:violation de la longueur maximale de contenu imposée par le MTL.

C22:violation de la priorité minimale imposée par le MTL.

C23:le MTA ne permet pas l'envoi de messages ("Control").

C24:"Ndn - Suppress" et "Content - Return" sont tous les deux TRUE.

C25:"Delivery Notice" est une adresse inconnue.

C26:un des noms de référence identifiant un destinataire est inconnu.

C27:"Ndn - Suppress" est une adresse inconnue.

C28:un des noms de référence identifiant un destinataire est inconnu.

C29:"Delivery Notice" est une adresse inconnue.

C30:un des noms de référence identifiant un destinataire est inconnu.

C31:"Conversion Prohibited" et "explicit Conversion" ne peuvent exister ensemble.

Success - Indication = ¬(C1 ∨ C2 ∨ .....∨ C31).

C'est-à-dire:

Success - Indication = true <=> ∀i 1 <=i <=31: Ci est fausse.

Success - Indication = false <=> $\exists$ i 1 <=i <=31: Ci est vraie.

"Failure - Reason" est l'adresse d'une zone qui contient un ensemble d'entiers.

zone $\ni$ i <=> Ci est vraie $\forall$ i: 1<=i <=31

"Submission - Time" est l'heure à laquelle la demande de SUBMIT a été acceptée.

"Submission - Event - Id" est un identificateur unique, généré par le MTA, qui permettra, plus tard, de faire référence a l'événement Submit (pour les notifications, par exemple).

"Success - Indication" sera toujours présent comme résultat. Les autres, facultatifs, apparaissent suivant la valeur de Success - Indication.

Si Success - Indication = true -> "Submission - Time", "Submission - Event - Id", "UA - Content - Id", seront présents.

Si Success - Indication = false -> "Failure - Reason" sera présent.

"UA - Content - Id" est retransmis, inchangé à l'UA qui l'avait fourni au MTA.

SERVICE : PROBE.

BUT:

En utilisant le service fourni par le MTA, l'UA permet à un de ses utilisateurs de tester si un message, répondant à des caractéristiques précises, a des chances d'arriver à ses destinataires lors d'un réel envoi, c'est-à-dire lors d'une activation du service SUBMIT avec des caractéristiques équivalentes. L'échec sera signifié par une notification de non livraison.

SPECIFICATIONS:

PROBE

Arguments

Recipient ϵ ADDRESS - ZONE.

(un ou plusieurs destinataires).

Origin - O/R-NAME.

(nom - O/R de l'origine).

Content - Type ϵ ("P2").

(type de contenu du message. P2 dans le cadre d'un messsage de personne à personne).

Encoded - Information - Type ϵ ADDRESS -ZONE.

(type(s) d'information contenue dans le message à tester).

Conversion - Prohibited ϵ {true}.

{demande qu'aucune conversion ne soit effectuée sur le message à tester}.

Alternate - Recipient - Allowed ϵ {true}.

{indique que le probe pourra traiter la possibilité d'appel au service "Alternate Recipient Allowed"}.

Content - Lenght ϵ INTEGER.

{longueur estimée en octets du contenu du message à tester}.

UA - Content - Id ϵ CONTENT - ID.

{identificateur du contenu du message, généré par l'UA}.

Explicit - Conversion ϵ ADDRESS - ZONE.

{indique qu'une conversion spécifiée doit être effectuée}.

Préconditions

Recipient:

adresse d'une zone préalablement garnie par l'UA, contenant la liste des noms O/R des destinataires des probe. Un destinataire sera repéré par son numéro d'ordre d'apparition dans cette liste (1,2,3,...).

Encoded -Information - Type:

adresse de la liste des types d'information qui apparaîtraient dans le contenu du message à tester.

Explicit - Conversion:

adresse d'une zone préalablement garnie par l'UA, contenant pour certains destinataires précisés dans "Recipient", l'indication du fait qu'on demande pour eux le service Explicit Conversion. [Cette zone contiendra la liste des numéros d'ordre des destinataires concernés par ce service, et pour chacun, le type dans lequel il faudra convertir].

Recipient - Origin - O/R-Name et Content - Type sont des arguments obligatoires. Les autres sont facultatifs.

Résultats:

Success - Indication ∈ {true, false}.

{indication de succès ou d'échec}.

Probe - Time ∈ TIME.

Probe - Event - Id ∈ PROBE - EVENT - ID.

Failure - Reason ∈ ADDRESS - ZONE.

UA - Content - Id ∈ CONTENT - ID.

Postconditions

Soient les conditions suivantes:

C1:"Recipient" est une adresse inconne.

C2:le nom - O/R d'un destinataire est incorrect.

C3:le nom - O/R de l'origine est incorrect.

C4:"Encoded - Information - Type" est une adresse inconnue.

C5:un des types précisés est inconnu.

C6:le paramètre Content - Type est incorrect.

C7:le paramètre UA - Content - Type est incorrect.

C8:"Explicit - Conversion" est une adresse inconnue.

C9:un des types précisés dans Explicit Conversion est inconnu.

C10:un des numéros d'ordre est inconnu (=> destinataire inconnu).

C11:la souscription au service "Alternate - Recipient - Allowed" n'a pas été demandée.

C12:la souscription aux services de Conversion n'a pas été demandée.

C13:la valeur de "Content - Lenght" dépasse la longueur maximale imposée par le MTA.

C14:"Content - Lenght" est un entier inacceptable pour une longueur (négatif, par exemple).

C15:la souscription au service "Probe" n'a pas été demandée.

Success - Indication = $\neg$ (C1 $\vee$ C2 $\vee$ ..... $\vee$ C15).

c'est-à-dire

Success - Indication = true <=> $\forall$ i 1 <=i <=15 Ci est fausse.

Success - Indication = false <=> $\exists$ i 1 <=i <=15 Ci est vraie.

"Failure - Reason" est l'adresse d'une zone qui contient un ensemble d'entiers.

zone ∃i <=> Ci est vraie ∀i 1 <=i <=15.

"Probe - Time" est l'heure à laquelle la demande de PROBE a été acceptée.

"Probe - Event - Id" est un identificateur unique, généré par le MTA, qui permettra, plus tard, de faire référence à l'événement probe en cours (pour les notifications, par exemple).

"Success - Indication" sera toujours présent, comme résultat. Les autres, facultatifs, apparaissent suivant la valeur de Success -Indication.

Si Success - Indication = true -> "Probe - Time", "Probe - Event - Id", "UA - Content - Id" présents.

Si Success - Indication = false -> Failure - Reason présent.

"UA - Content - Id" est retransmis, inchangé à l'UA qui l'avait fourni au MTA.

SERVICE : MESSAGE DELIVERY.

BUT:

Le MTA utilise la primitive Deliver pour livrer un message à un UA Recipient. Cet UA ne peut pas refuser la livraison.

SPECIFICATIONS:

MTL - DELIVER - SIGN.

Arguments

Originator - O/R-Name ∈ OR-Name.

{Originator - O/R-Name est le nom de l'UA d'origine}.

This - Recipient - O/R-Name ∈ OR-Name.

{This - Recipient - O/R-Name est le nom du récipient auquel on donne le message}.

Other - Recipient - O/R-Names ∈ ADDRESS - ZONE.

{Other - Recipient - O/R-Names est l'adresse de la zone contenant tous les autres destinataires du message qu'on est occupé à livrer}.

Content ∈ ADDRESS - ZONE.

{Content est l'adresse d'une zone renfermant le contenu du message}.

Content - Type ∈ {"P2"}

Converted - Encoded - Information - Types ∈ ADDRESS - ZONE.

{Converted - Encoded - Information - Types est l'adresse d'une zone contenant les types d'informations encodées du message livré, si ces types sont différents de ceux d'origine}.

Original - Encoded - Information - Types ϵ ADDRESS - ZONE.

{Original - Encoded - Information - Types est l'adresse d'une zone contenant les types d'informations encodées du message à l'origine. Ces types sont présents, qu'une conversion ait été faite ou pas.

Delivery - Time ϵ TIME.

{Delivery - Time indique à quel moment le message a été livré}.

Submission - Time ϵ TIME

{Submission - Time indique à quel moment le message a été envoyé depuis l'origine}.

Priority ϵ {"urgent", "non urgent", "normal"}.

{indication de la priorité de la livraison}.

Intended - Recipient - O/R-Name ϵ OR-NAME.

{Intended - Recipient - O/R-Name est le nom du destinataire attendu, comme il a été spécifié par l'UA d'origine}.

Conversion - Prohibited ϵ {true}.

{indication qu'une interdiction de conversion a été imposée par l'UA d'origine}.

Deliver - Event - Id ϵ DELIVER - EVENT - ID.

{Deliver - Event - Id indique l'identificateur qui est attribué à cette primitive Deliver}.

Préconditions

Les arguments suivants sont facultatifs.

Other - Recipient - O/R-Name qui ne sera présent que lorsque l'UA d'origine a demandé "Disclosure of Other Recipients" lors du Submit.

Converted - Encoded - Information - Types qui existera s'il y a eu au moins une conversion de type d'informations encodées.

Original - Encoded - Information - Types.

Intented - Recipient - O/R-Name qui est présent uniquement si le message est livré à un Alternate - Recipient.

Conversion - Prohibited qui n'existe que si l'UA d'origine a interdit toute conversion.

La valeur de Originator - O/R-Name est celle qui est donnée dans l'argument de même nom lors du Submit.

This - Recipient - O/R-Name est bien un O/R-Name d'un UA appartenant au MTA.

Other - Recipient - O/R-Names est une adresse connue.

Other - Recipient - O/R-Names est l'adresse d'une zone contenant un ensemble d'O/R-Names.

Content est une adresse connue.

Content est l'adresse d'une zone contenant une information conforme à P2.

Converted - Encoded - Information - Type est une adresse connue.

converted - Encoded - Information - Types est l'adresse d'une zone contenant des types d'informations encodées possibles.

Original - Encoded - Information - Types est une adresse connue.

Original - Encoded - Information - Types est l'adresse contenant des types d'informatios encodées possibles.

Submission - Time < Delivery - Time.

Intented - Recipient - O/R-Name est le nom du destinataire donné dans le Submit.

Résultats

postconditions

SERVICE : MESSAGE NOTIFICATIONS

BUT:

Le MTA utilise la primitive Notify pour informer l'UA qu'un message envoyé précédemment a ou pas été livré pour convoyer le résultat d'un probe. L'envoi d'un message ou un probe vers plusieurs destinataires peut provoquer plusieurs occurences de cette primitive MTL - Notify - Sign.

SPECIFICATIONS:

MTL - NOTIFY - SIGN.

Arguments

Notification - Type ϵ {"livraison", "non-livraison"}.

{Notification - Type indique si la notification est une notification de livraison ou de non-livraison}.

Submit - Event - Id ϵ SUBMIT - EVENT -ID.

{Submit - Event -Id identifie le submit auquel la notification fait référence}.

Probe - Event - Id ϵ PROBE - EVENT - ID.

{Probe - Event - Id identifie le probe auquel la notification fait référence}.

Recipient - O/R-Names ϵ ADDRESS - ZONE.

{Recipient - O/R-Names est l'adresse d'une zone qui contient les noms des récipients d'origine vers lesquels la notification est envoyée}.

Non - Delivery - Reason ϵ ADDRESS - ZONE.

{indication de la raison de non-livraison}.

Delivery - Time ϵ ADDRESS - ZONE.

{indication du moment auquel le message a été livré}.

Converted - Encoded - Information - Types ϵ ADDRESS - ZONE.

{Converted - Encoded - Information - Types est l'adresse d'une zone contenant les types d'informations résultants de conversions}.

Intended - Recipient - O/R-Name ϵ ADDRESS - ZONE.

{Intended - Recipient - O/R-Name est l'adresse d'une zone contenant des noms UA d'origine tels que spécifiés par les UA d'origine eux-mêmes}.

Supplementary - Information ϵ ADDRESS - ZONE.

{Supplementary - Information est l'adresse d'une zone contenant une information supplémentaire pour une utilisation possible des services télématiques}.

Returned - Content ϵ ADDRESS - ZONE.

{Returned - Content est l'adresse d'une zone qui contient le contenu du message dont la livraison a échoué}.

Type - Of - UA ϵ ADDRESS - ZONE.

{indication des types des UA destinataires}.

UA - Content - Id ϵ CONTENT - ID.

{identification du contenu du message envoyé}.

Préconditions:

"Recipient - O/R-Names" et "Notification type" sont obligatoires.

"Probe - Event - Id" et "Submit - Event - Id" sont facultatifs mais un et un seul des deux doit être présent.

Recipient - O/R-Names est une adresse connue.

Recipient - O/R-Names est l'adresse d'une zone contenant un ensemble de noms OR.

Non - Delivery - Reason est une adresse connue.

Non - Delivery - Reason est présent si la valeur de Notification - Type est non livraison.

Non - Delivery - Reason est l'adresse d'une zone contenant autant de raisons de non livraison qu'il y a eu d'échecs.

Delivery - Time est présent seulement si la valeur de Notification Type est livraison.

Delivery - Time est une adresse connue.

Delivery - Time est l'adresse d'une zone contenant un ou plusieurs éléments de type Time. Le nombre d'éléments dans cette zone est fonction du nombre d'UA récipients auxquels on a livré le message ou le probe.

Converted - Encoded - Information - Types est une adresse connue.

Cet argument est présent uniquement s'il s'agit d'un Notify en réponse à un Submit et si la valeur de Notification - Type est livraison.

Converted - Encoded - Information - Types est l'adresse d'une zone contenant des types d'informations encodées.

Intended - Recipient - O/R-Name est présent dans le cas où on a livré le message à Alternate - Recipient ou dans le cas où on a essayé de livrer le message à Alternate - Recipient et que ça a échoué.

Intended - Recipient - O/R-Name est une adresse connue.

Intended - Recipient - O/R-Name est l'adresse d'une zone contenant des éléments de type OR-Name.

Supplementary - Information est une adresse connue.

Returned - Content est présent uniquement dans le cas d'un Notify qui répond à un Submit et si le contenu a été demandé en retour par l'UA d'origine.

Returned - Content est une adresse connue.

Type - Of - UA est une adresse connue.

Type - Of - UA est l'adresse d'un zone dont les éléments sont soit "privé", soit "administration". Il y aura autant d'éléments que d'UA destinataires qui ont reçu correctement le message ou le probe.

Résultats

postconditions

SERVICE : DEFERRED DELIVERY CANCELLATION.

BUT:

Essayer d'annuler le transfert ou la livraison d'un message précédemment envoyé avec demande de livraison différée.

SPECIFICATIONS:

CANCEL

Argument

identi ϵ SUBMIT - EVENT - ID.

{identifie l'événement Submit concernant l'envoi du message à annuler}.

Préconditions

Résultats

Success - Indication ϵ {true, false }.

{indication de succès ou d'échec}.

Failure - Reason ϵ {1,2,3}.

{raison de l'échec}.

postconditions

Le paramètre Failure - Reason est facultatif. Il est présent lorsque Success - Indication = false, c'est-à-dire lorsqu'il est impossible de réaliser le service demandé.

Table de décision.

identi inconnu	Y	N	N	N
message déjà livré	-	Y	-	N
message déjà transmis	-	-	Y	N
succes-ind = false	x	x	x	
succes-ind = true				x
failure reason = 1	x			
failure reason = 2		x		
failure reason = 3			x	
annulation de la tâche SUBMIT identifiée par identi				x

SERVICE : CHANGEMENT DE MOT DE PASSE.

BUT:

L'UA utilise la primitive "change password request" pour indiquer à son MTA que son mot de passe change [mot de passe que le MTA doit connaître pour identifier l'UA

lorsque celui-ci veut se connecter]. Il donne d'abord l'ancien mot de passe puis le nouveau.

Le MTA confirme cette modification.

Le changement de mot de passe échoue si l'ancien mot de passe fourni est incorrect. Si l'opération réussit, le MTA enregistrera comme mot de passe de l'UA le nouveau mot de passe , sinon, il ne modifiera rien.

SPECIFICATIONS:

UAL CHANGE PASSWORD.

Arguments

Old-password ∈ STRING.

New-password ∈ STRING.

Préconditions

Résultats

Success - Indication ∈ {true, false}

{indication du succès ou de l'échec}.

Failure - Reason ∈ {1}.

postconditions

old-password correct (i.e. old-password = password de l'UA enregistré dans le MTA)	Y	N
succes-indication = true	x	
succes-indication = false		x
failure-reason = 1		x

SERVICE : MTL - CHANGE - PASSWORD.

BUT:

Le MTA active ce service pour indiquer à un de ses UA que son mot de passe, pour lui, possèdera dorénavant une nouvelle valeur.

SPECIFICATIONS:

MTL - CHG - PASSW - SIGN.

arguments :

Old-passw ϵ PASSWORD

(valeur courante du mot de passe).

New-passw ϵ PASSWORD.

(nouvelle valeur du mot de passe).

1.3. LES TABLES D'ETATS

Ces tables d'états complètent les spécifications des primitives détaillées ci-dessus. Voici, avant de continuer, la définition des états et des actions figurant dans ces tables :

UA CONNECTE :

le MTA est connecté avec l'UA qui veut ou avec qui il veut enter en dialogue.

NON UA CONNECTE :

le MTA n'est pas connecté avec l'UA qui veut ou avec qui il veut entrer en dialogue.

ATTENTE UA CONNECTE :

le MTA attend la réponse à une demande de connexion avec un de ses UA.

NON LIVRAISON UA :

le MTA n'est pas en train de traiter un Deliver ou un Notify pour cet UA.

ATTENTE-CONTROLE-REPONSE :

le MTA attend la réponse à une demande de restriction pour un de ses UA.

TRANSFERT :

le MTA traite un message qui lui vient

soit d'un de ses UA pour un envoi.

soit des couches inférieures pour un transfert.

MTA EN ATTENTE :

le MTA attend la réponse à une demande de connexion avec un MTA adjacent.

MTA CONNECTE AVEC JETONS :

le MTA est connecté avec un MTA adjacent. Il détient les jetons, c'est-à-dire qu'il dirige le dialogue, pour envoyer ses données.

MTA CONNECTE SANS JETON :

le MTA est connecté avec un MTA adjacent. Il ne détient pas les jetons, c'est-à-dire qu'il est récepteur des données de l'autre.

IDLE MTA CONNECTE :

le MTA est connecté avec un MTA adjacent, mais se trouve au repos sur cette connexion session.

MTA CONNECTE IDLE SANS JETON :

(MTA CONNECTE SANS JETON) $\cup$ (IDLE et MTA CONNECTE)

MTA CONNECTE IDLE AVEC JETONS

(MTA CONNECTE AVEC JETONS) $\cup$ (IDLE et MTA CONNECTE)

A1 : vérification des paramètres.

A2 :remplir les paramètres de sortie de manière positive.

A3 :remplir les paramètres de sortie de manière négative.

A4 :renvoyer accusé de reception.

A5 :enregistrer les nouvelles valeurs de REGISTER.

A6 :enregistrer les nouvelles valeurs du CONTROL.

A7 :annulation de l'envoi différé.

A8 :mise à jour du mot de passe.

A9 :analyser des adresses.

A10 :déterminer si c'est un User MPDU ou un service MPDU.

A11 :déterminer le nombre et créer les copies.

A12 :analyse de l'enveloppe.

A13 :lecture des valeurs courantes.

A14 :destruction du message.

A15 :effectuer la conversion.

A16 :création de la notification de non livraison avec retour de contenu.

A17 :création de notification de non livraison sans retour de contenu.

A18 :création de la notification de livraison.

A19 :création de l'enveloppe.

A20 :activer l'algorithme de routage.

A21 :modification du message pour qu'il aille vers le récipient alternatif. On interdit toute notification.

A22 :analyse de la trace (et ajoûts si nécessaire).

A23 :suppression des informations dèjà reçues.

A24 :mémorisation des informations dèjà reçues en vue d'une reprise (selon politique du MTA).

FIGURE 3.1. LISTE DES ACTIONS.

LOGOFF

	UA CONNECTE
LOGOFF	A4 NON UA CONNECTE

REGISTER

	UA CONNECTE
REGISTER	A1 Si correct alors A5 A2 sinon A3

UA CONTROL

	UA CONNECTE
CONTROL	A1 Si correct (et NON-LIVRAISON UA) alors A6 A2 sinon A3

MTL - CONTROL

	UA CONNECTE	ATTENTE CONTROLE REPONSE
Décision du MTA de changer ses restrictions	MTL - CONTROL - SIGN ATTENTE CONTROLE REPONSE	
MTL - CONTROL - RESP		UA CONNECTE

SUBMIT

	UA CONNECTE
SUBMIT	Si MTL ressources disponibles alors A1 si non correct alors A3 sinon A2 A19 TRANSFERT sinon A3

PROBE

	UA CONNECTE
PROBE	A1 Si non correct alors A3 sinon A2 A19 TRANSFERT

DELIVER

	UA CONNECTE
Le MTA vient de recevoir un message à lirer à un UA local	DELIVER - SIGN UA CONNECTE

NOTIFY

	UA CONNECTE
Le MTA vient de recevoir une notification à donner à un UA local	NOTIFY UA CONNECTE

CANCEL

	UA CONNECTE
CANCEL	A1 Si (non correct) ou (message livré) ou (message transféré) alors A3 sinon A2 A7

PASSW

	UA CONNECTE
CHANGE - PASSWORD	A1 Si non correct alors A3 sinon A2 A8

MTL - PASSW

	UA CONNECTE
Le MTA décide de changer son mot de passe	MTL - CHG - PASSW - SIGN UA CONNECTE

1.4. LES DIAGRAMMES D'ETATS

Ce paragraphe explique, à l'aide de diagrammes, la façon dont le MTA gère les ouvertures et fermetures de connexion, ainsi que les soumissions de messages depuis un UA local.

Pour le Transfert, le diagramme d'état est impossible à construire vu la complexité de l'opération et les nombreux cas à distinguer. Ce diagramme est donc remplacé par un algorithme en pseudo langage dont la liste des actions est reprise à la figure 3.1.

CI SUIT L'ALGORITHME DE TRANSFERT

```
A9
A12
A11
Pour chaque copie,

SI (UA local)
ALORS

LABEL:

SI (UA CONNECTE)
ALORS
 A13
 SI (il s'agit d'une notification)
 ALORS
  SI (NOTIFY permis)
  ALORS
   SI longueur permise
   ALORS NOTIFY
```

```
  SINON A14
 SINON A14
 retour à l'état précédent le transfert.

SI (il s'agit d'un message)
ALORS
 SI (livraison différée)
 ALORS
  SI (file d'attente non complète)
  ALORS mise en file d'attente
  SINON
   SI (non Ndn - Suppress)
   ALORS
    SI (retour de contenu)
    ALORS
     A16
    SINON
     A17
    TRANSFERT
   SINON retour à l'état précédent le transfert
 SINON
 SI Deliver permis
 ALORS
  SI (conversion explicite demandée)
  ALORS
   SI (conversion explicite dèjà faite)
   ALORS
    SI (type résultant permis)
    ALORS type ok
    SINON
     SI (conversion implicite demandée)
     ALORS
      A15
      type OR
     SINON type non OR
   SINON
    SI (type explicite permis)
    ALORS
     A15
     type ok
    SINON
     SI (conversion implicite demandée)
     ALORS
      A15
      type ok
     SINON type non ok
   SINON
    SI (type courant permis)
    ALORS type OR
    SINON
     SI (conversion implicite demandée)
     ALORS
      A15
      type ok
     SINON type non ok
      SI (type ok)
      ALORS
       SI (longueur acceptable)
       ALORS
        SI (priorité acceptable)
```

```
          ALORS "DELIVER"
           SI (notification demandée)
           ALORS
            A18
            TRANSFERT
           SINON (retour à l'état précédant le transfert)
          SINON type non ok
        SINON type non ok
        SI (type non ok)
        ALORS
         SI (UA abonné à "Hold For Delivery")
         ALORS
          problème du Hold For Delivery
         SINON
          SI (non Ndn - Suppress)
          ALORS
           SI (retour de contenu)
           ALORS
            A16
           SINON
            A17
            TRANSFERT
          SINON (retour à l'état précédant le transfert).


SI (il s'agit d'un "probe")
ALORS
 SI (Register et contrôle ok)
 ALORS
  A18
  TRANSFERT
 SINON
  A17
  TRANSFERT


SI (NON UA CONNECTE)
ALORS
 "MTL - LOGON"
 SI (NON UA CONNECTE)
 ALORS (nouvel essai selon politique du MTA)
  Si (échec et non Ndn - Suppress et il s'agit d'un message)
  ALORS
   Si (retour du contenu demandé)
   ALORS
    A16
   SINON
    A17
   TRANSFERT
 SI (échec et il s'agit d'un "probe")
 ALORS
  A17
  TRANSFERT
 SI [(il s'agit d'une notification) ou ((il s'agit d'un
message) et Ndn - Suppress) et échec]
 ALORS (retour à l'état précédant le transfert)
 SINON goto LABEL
```

```
SI ( UA NON LOCAL )
ALORS
 A20
 SI ( Non UA Connecté )
 ALORS
  "OUVRIR CONNECTION"
  SI echec
  ALORS (nouvel essai selon politique MTA )
 SI ( NON MTA CONNECTE )
 ALORS
    Si (échec et non Ndn - Suppress et il s'agit d'un message)
  ALORS
   Si (retour du contenu demandé)
   ALORS
    A16
   SINON
    A17
   TRANSFERT
 SI (échec et il s'agit d'un "probe")
 ALORS
  A17
  TRANSFERT
 SI [(il s'agit d'une notification) ou ((il s'agit d'un
message) et Ndn - Suppress) et échec]
 ALORS (retour à l'état précédant le transfert)
 SINON goto LABEL
SINON
"ENVOI SESSION"
Si échec
ALORS
  Si (échec et non Ndn - Suppress et il s'agit d'un message)
  ALORS
   Si (retour du contenu demandé)
   ALORS
    A16
   SINON
    A17
   TRANSFERT
 SI (échec et il s'agit d'un "probe")
 ALORS
  A17
  TRANSFERT
 SI [(il s'agit d'une notification) ou ((il s'agit d'un
message) et Ndn - Suppress) et échec]
 ALORS (retour à l'état précédant le transfert)
 SINON goto LABEL
SINON [éventuellement FERMER CONNEXION selon politique MTA]


SI (routage impossible)
ALORS
 SI (il s'agit d'une notification)
 ALORS (retour à l'état précédant le transfert)
 SINON
  SI (Alternate Récipient Allowed)
  ALORS
   A21
   TRANSFERT
   A17
   TRANSFERT
```

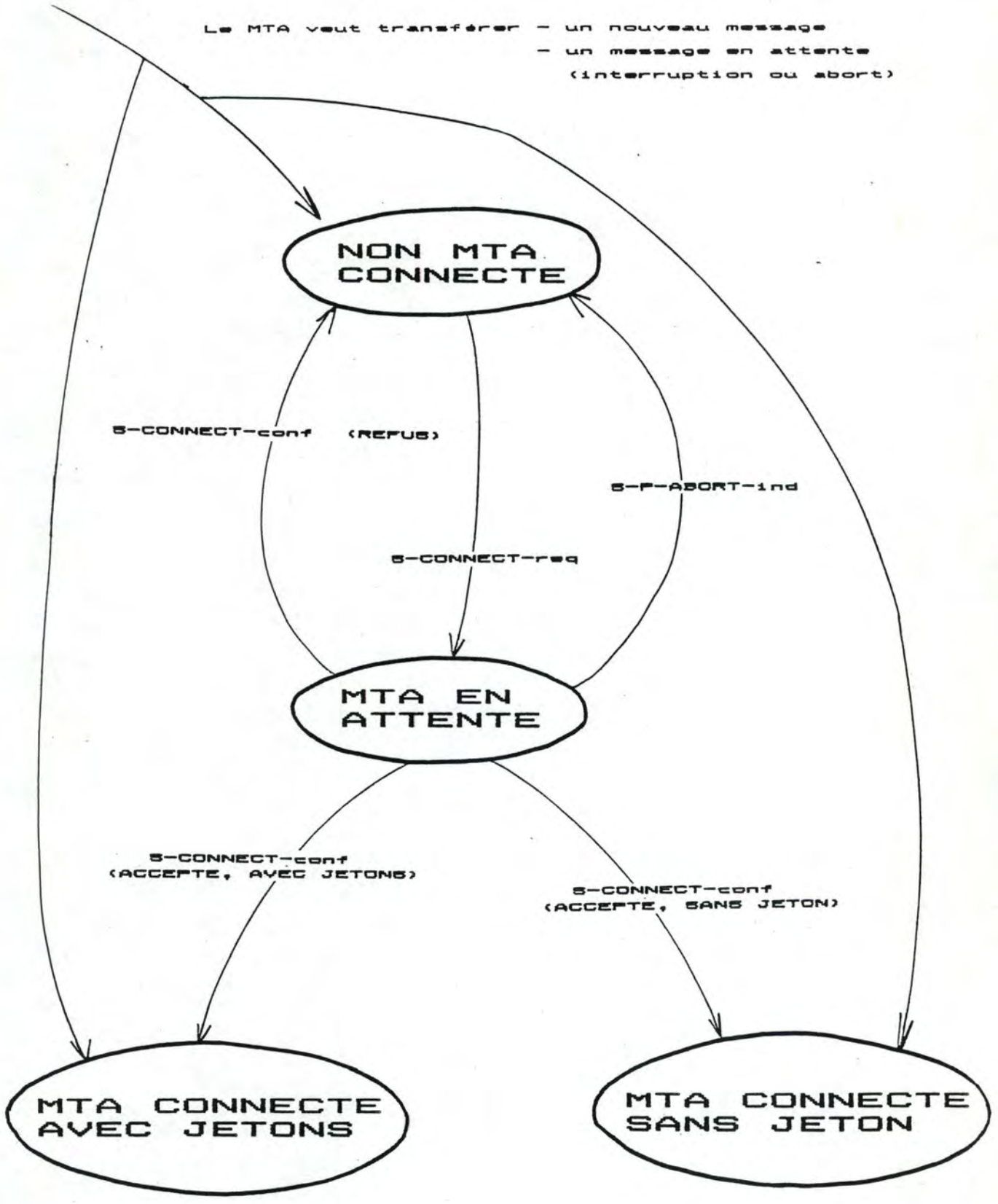
```
  SINON
    Si (échec et non Ndn - Suppress et il s'agit d'un message)
  ALORS
   Si (retour du contenu demandé)
   ALORS
    A16
   SINON
    A17
   TRANSFERT
 SI (échec et il s'agit d'un "probe")
 ALORS
  A17
  TRANSFERT
 SI [(il s'agit d'une notification) ou ((il s'agit d'un
message) et Ndn - Suppress) et échec]
 ALORS (retour à l'état précédant le transfert)
 SINON goto LABEL
```

MTA - CONNECTE

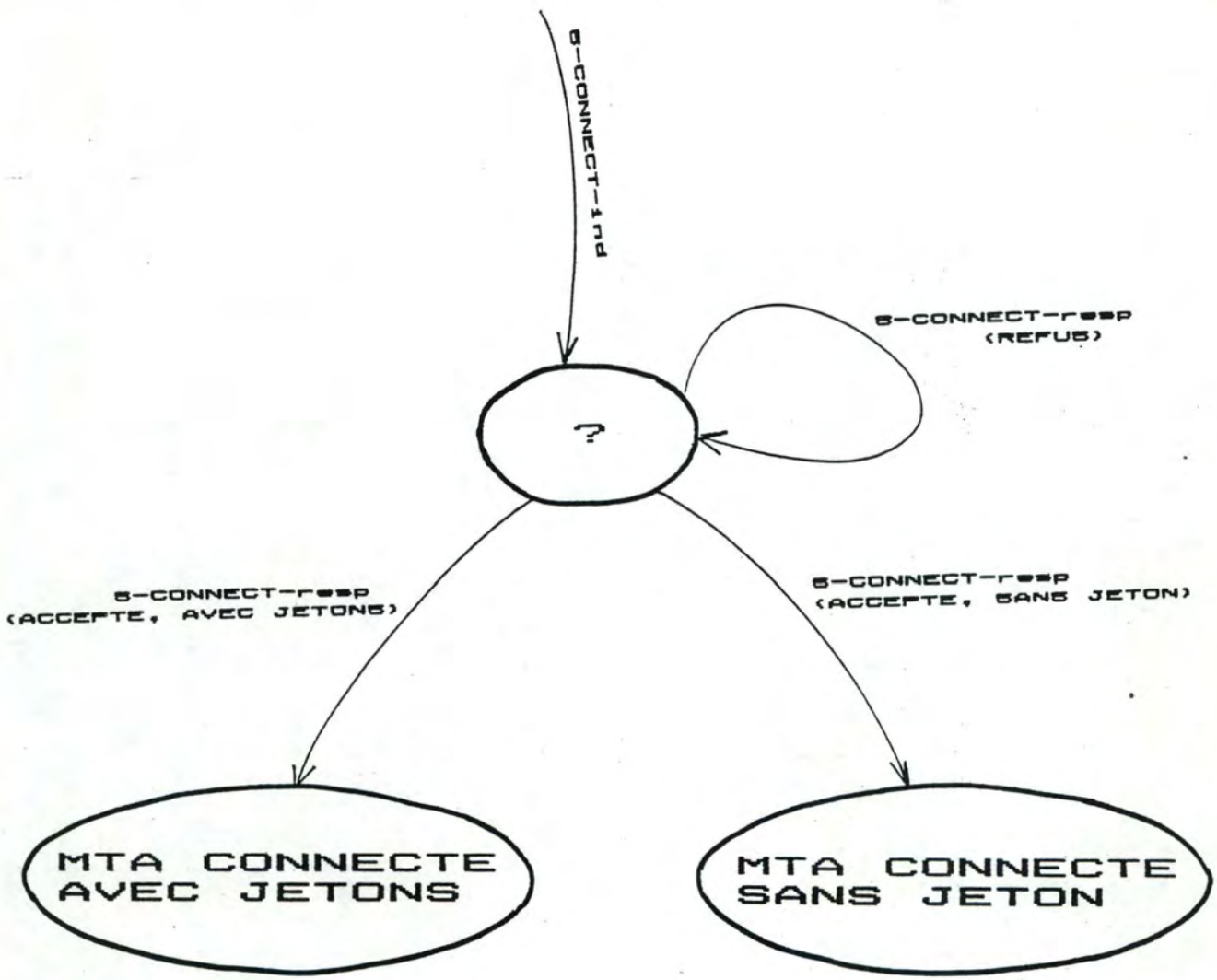
```
"RECEPTION SESSION"

SI (réception correcte)
ALORS
 A22
 SI circuit
 ALORS
  Si (échec et non Ndn - Suppress et il s'agit d'un message)
  ALORS
   Si (retour du contenu demandé)
   ALORS
    A16
   SINON
    A17
   TRANSFERT
 SI (échec et il s'agit d'un "probe")
 ALORS
  A17
  TRANSFERT
 SI [(il s'agit d'une notification) ou ((il s'agit d'un
message) et Ndn - Suppress) et échec]
 ALORS (retour à l'état précédant le transfert)
 SINON goto LABEL
SINON
  TRANSFERT
SI (Discard)
ALORS
 A23
SI (interruption ou abort)
ALORS
 A24
```

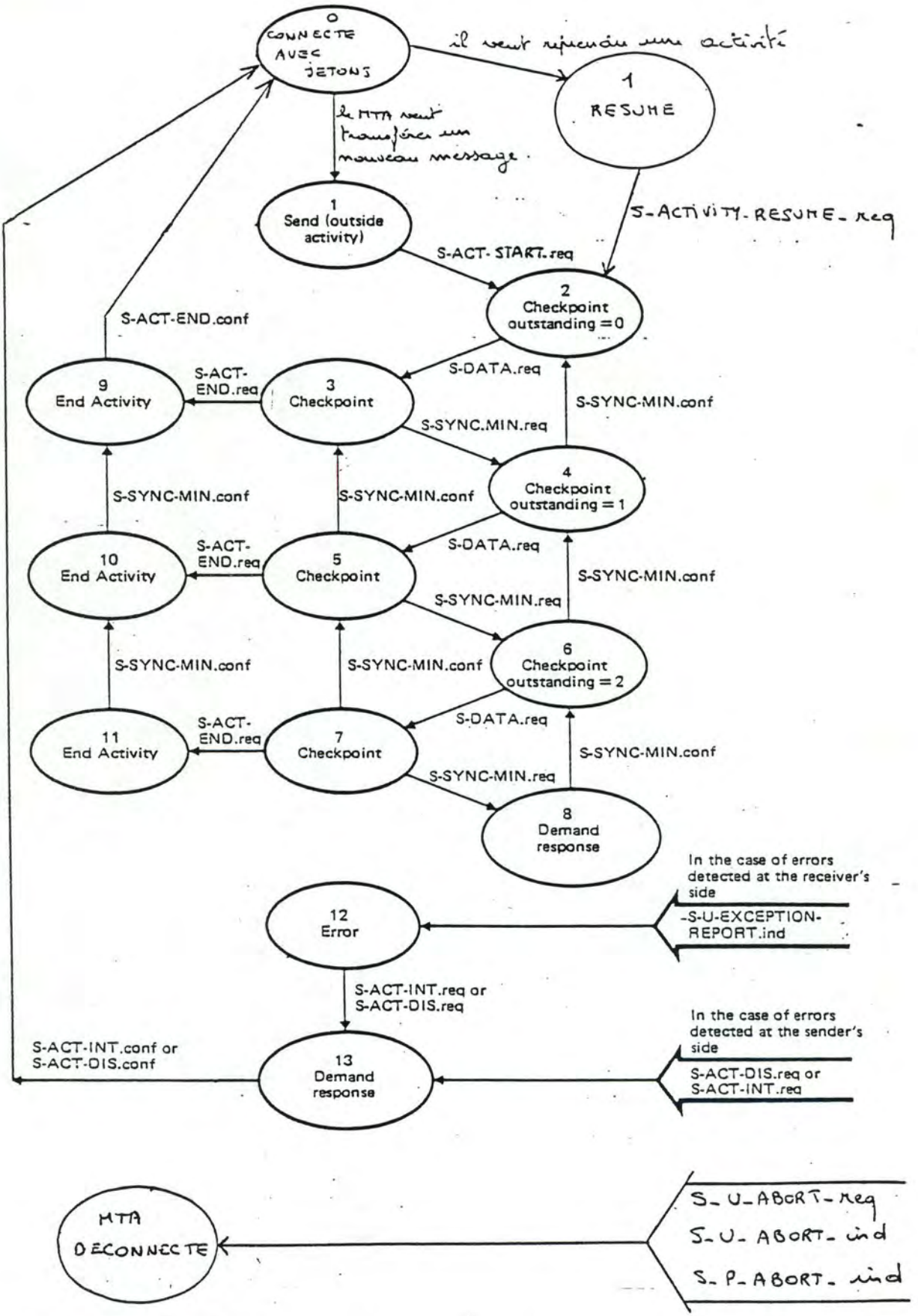
OUVRIR CONNEXION (point de vue émetteur)



OUVRIR CONNEXION (point de vue récepteur)

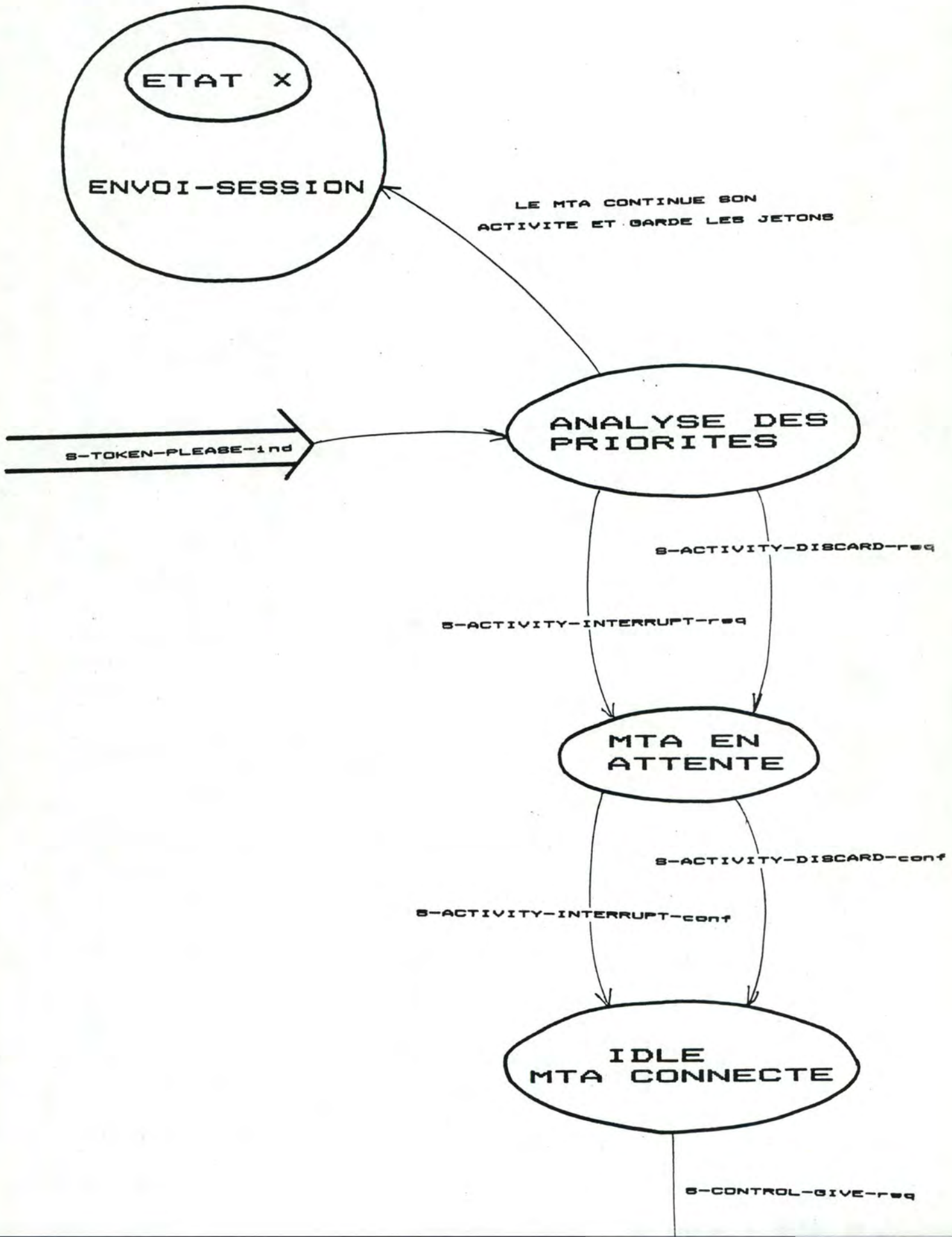


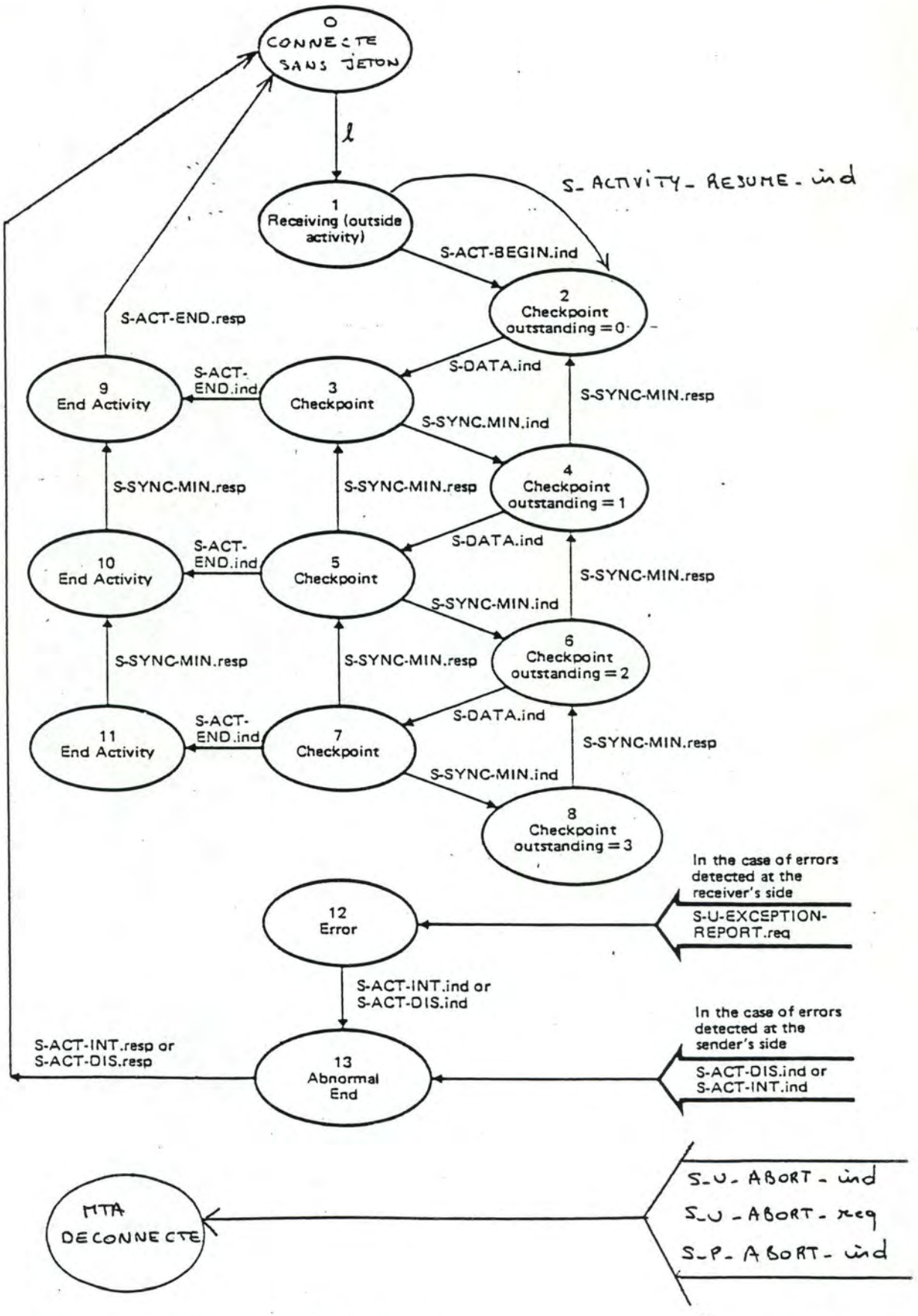
ENVOI-SESSION



Note – This diagram assumes a window size of 3.

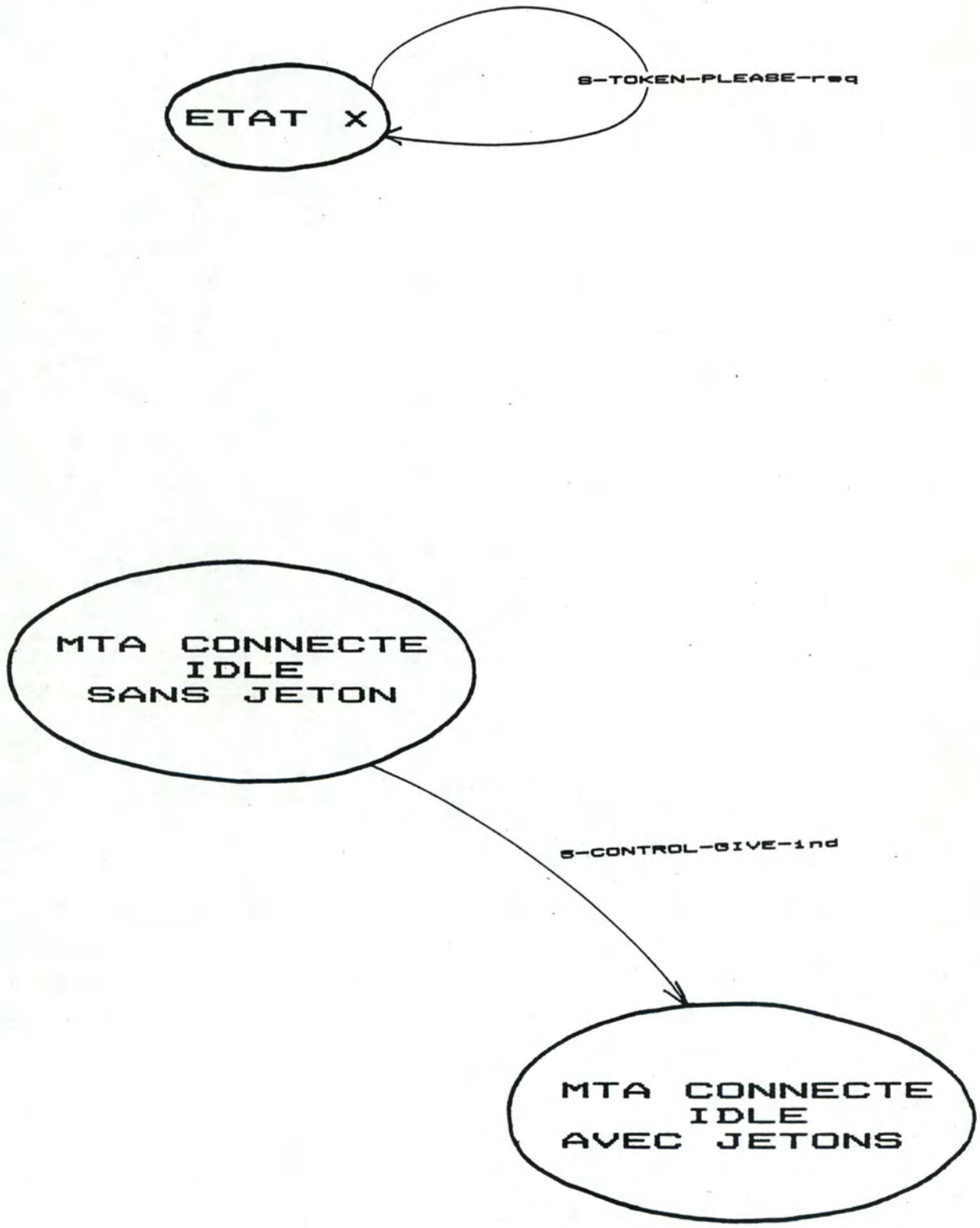
ENVOI SESSION



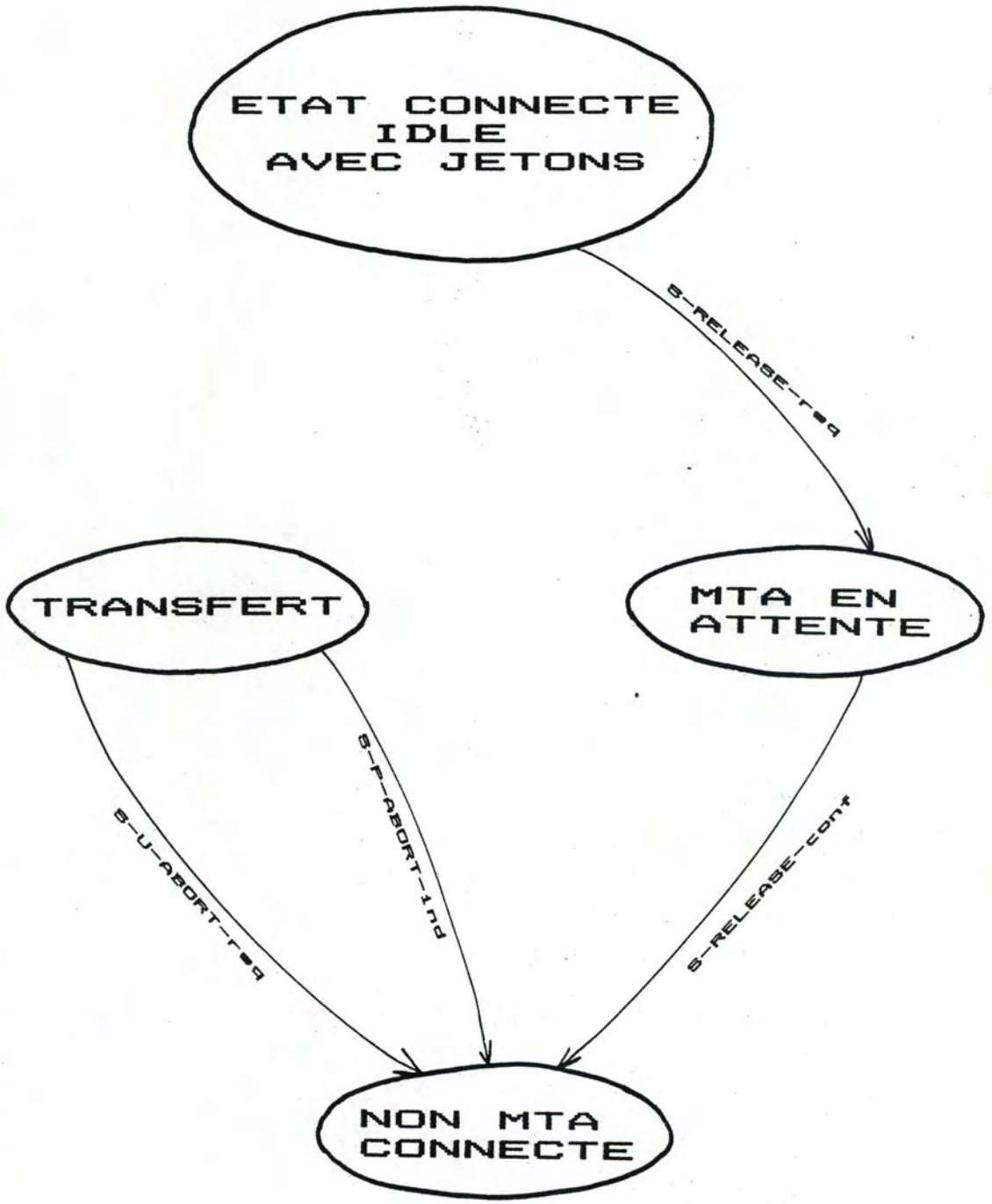


Note – This diagram assumes a window size of 3.

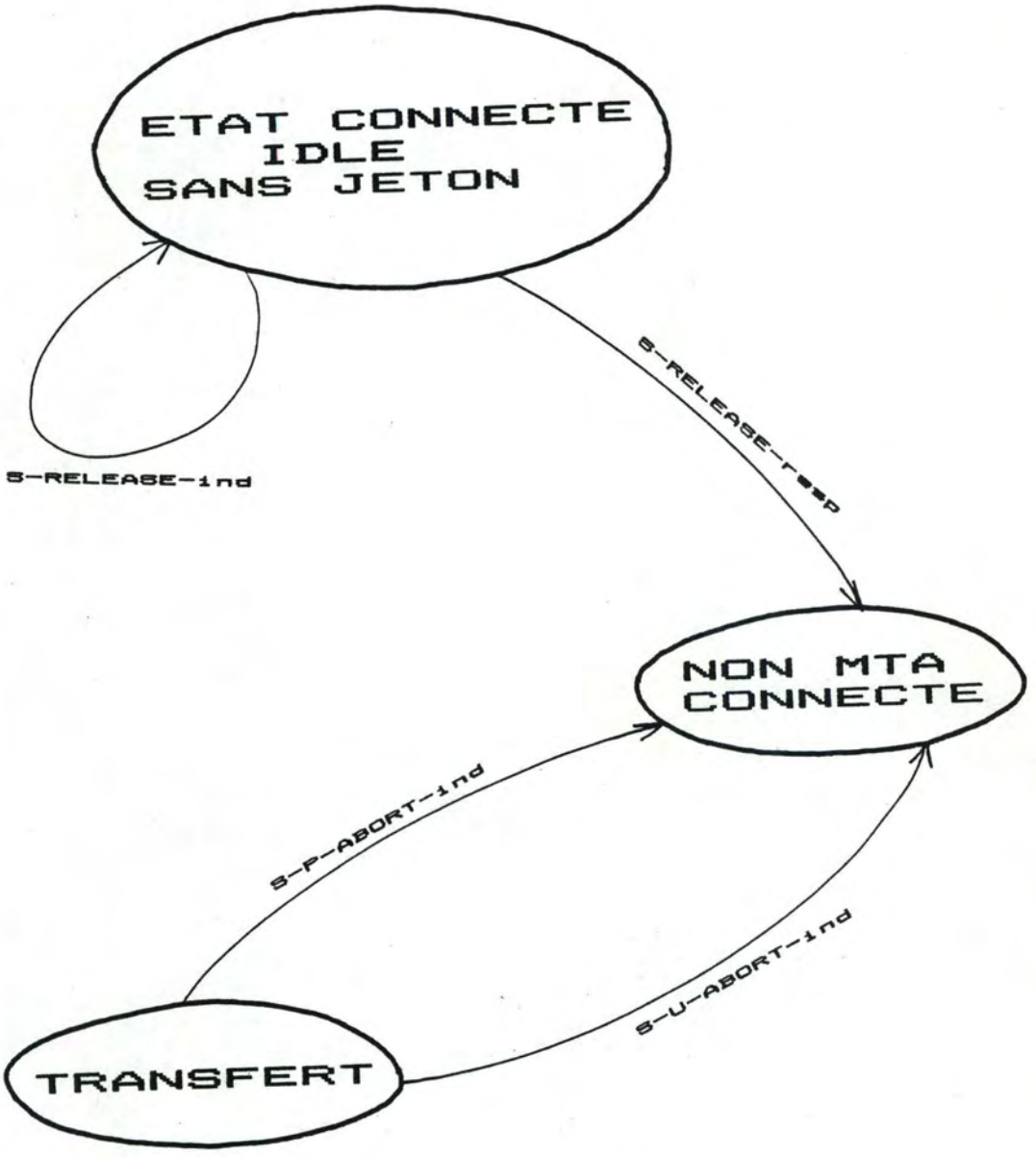
RECEPTION SESSION



FERMER CONNEXION (point de vue émetteur)



FERMER CONNEXION (point de vue récepteur)



Annexe 2.

SPECIFICATIONS EXTERNES DES MODULES FONCTIONNELS.

2.1. INTRODUCTION.

Cette annexe va reprendre dans le détail les spécifications pré-post de toutes les primitives des modules fonctionnels, à l'exception des primitives permettant l'envoi de messages que sont TRANSFER et S-DATA déjà présentées au chapitre IX.

Il faut remarquer aussi que pour les spécifications des primitives d'interaction entre le MTA et un UA, le lecteur est renvoyé au chapitre VI.

2.2. LES SPECIFICATIONS PRE-POST

2.2.1. LES PRIMITIVES OFFERTES PAR LE RTS.

LES TYPES

PRIORITY :

contient toutes les priorités que l'AM peut véhiculer lorsqu'il demande le tour. Ces priorités sont fonction

des actions qui peuvent provoquer la demande de tour et les valeurs qu'elles peuvent prendre sont fixées par le protocole application.

PROCEDURE D'OPEN. ETABLISSEMENT D'UNE ASSOCIATION.

BUT:

L'AM active cette procédure afin de demander l'ouverture d'une association entre lui et l'AM d'un MTA adjacent. Ceci s'appuie sur une connexion session (ou plusieurs, en séquence) établie(s) par le RTS comme conséquence d'un OPEN.

SPECIFICATIONS:

Lorsque l'AM est créé par le GI-MTA-MTA, il est branché automatiquement sur cette procédure d'OPEN.

OPEN-REQ-SIGN.

Arguments

Responder-Address ϵ ADDRESS.

(adresse session du RTS associé à l'AM avec lequel l'association doit être établie).

Dialogue-Mode ∈ {1,2}.

{type de l'association à établir

1 <=> monologue.

2 <=> communication dans les deux sens, à l'alternat}.

Initial-Turn ∈ {1,2}.

{désignation de l'AM qui aura le tour initial

1 <=> celui qui ouvre l'association.

2 <=> celui qui répond à la demande}.

Application-Protocol ∈ {"P1", "P3"}.

{choix du protocole de niveau application qui régira la communication utilisant l'association.

"P1" <=> l'UA est directement relié à un MTA.

"P3" <=> l'UA est lié à un MTA via une entité SDE}.

User-Data ∈ STRING

préconditions

User-Data est un argument facultatif.

OPEN-IND-RESP

(ceci suit une primitive OPEN-REQ-SIGN du côté du AM demandeur).

Arguments

Initiator-Address ε ADDRESS.

(adresse session du RTS associé à l'AM qui a déclenché la primitive OPEN-REQ-SIGN).

Dialogue-Mode ε (1,2).

(type de l'association à établir

1 <=> monologue.

2 <=> communication dans les deux sens à l'alternat).

Initial-Turn ε (1,2).

(désignation de l'AM qui aura le tour initial

1 <=> celui qui ouvre l'association.

2 <=> celui qui répond à la demande).

Application-Protocol ε ("P1", "P3").

(choix du protocole du niveau application qui régira la communication utilisant l'association

"P1" <=> l'UA est directement relié à un MTA.

"P3" <=> l'UA est lié à un MTA via une entité).

User-Data ε STRING.

préconditions

User-Data est un paramètre facultatif.

Dialogue-Mode, Initial-Turn, Application-Protocol,User-Data doivent contenir les mêmes valeurs que leurs homologues dans la primitive OPEN-REQ-SIGN qui précède.

Résultats

Disposition ∈ {1,2,3}.

{raison pour laquelle l'association est refusée}.

postconditions

User-Data et Refusal-Reason sont des arguments facultatifs.

User-Data présent <=> disposition = 1.

{association acceptée}.

Refusal-Reason <=> disposition = 2.

{association refusée}.

Refusal-Reason = 1 <=> le mode de dialogue proposé par le demandeur est non acceptable par le répondeur.

Refusal-Reason = 2 <=> il y a eu erreur lors de l'authentification du demandeur de l'ouverture de l'association.

Refusal-Reason = 3 <=> le MTA voisin est trop occupé pour accepter une nouvelle association.

exemple: le nombre de processus AM occupés est maximum.

OPEN-CONFIRM

(cette procédure est activée par le RTS lorsqu'il a reçu du MTA voisin la réponse à la demande de connexion qu'il a déclenchée suite à l'OPEN-REQ-SIGN de l'AM).

Les arguments sont identiques aux paramètres Résultats de la procédure OPEN-IND-RESP. Il ont même valeur.

PROCEDURE CLOSE

BUT:

cette procédure permet à l'AM possèdant le tour d'abandonner ou de clôturer une association avec un AM voisin.

SPECIFICATIONS:

CLOSE-REQ-SIGN

Arguments

préconditions

Résultats

postconditions

CLOSE-IND-RESP

Arguments

préconditions

Résultats

postconditions

CLOSE-CONF

Arguments

préconditions

Résultats

postconditions

PROCEDURE TURN-PLEASE

BUT:

Cette procédure permet à l'AM ne possèdant pas le tour, d'obtenir la permission de transfert d'un APDU ou de

clôture d'association grâce à la réception de ce tour. Cette demande de tour est accompagnée de la priorité des actions à effectuer de sorte que l'AM récepteur peut choisir de donner ou de garder le tour.

SPECIFICATIONS:

TURN-PLEASE-REQ-SIGN.

Arguments

priority ∈ PRIORITE.

{priorité maximale des actions ayant entraîné la demande de tour}.

préconditions

priority est la valeur donnée en fonction de l'action (à effectuer par l'AM) qui a provoqué la demande de tour.

Résultats

postconditions

TURN-PLEASE-IND.

Arguments

priority ∈ PRIORITE.

préconditions

priority est exactement la valeur contenue dans le paramètre priority de la procédure de demande de tour envoyée par l'AM origine.

Résultats

postconditions

PROCEDURE EXCEPTION

BUT:

Le RTS utilise cette primitive pour signaler à l'AM qu'il ne peut pas assurer le transfert de l'APDU, demandé précédemment par l'AM.

SPECIFICATIONS:

EXCEPTION-IND

Arguments

APDU ∈ ADDRESS - ZONE.

(c'est l'APDU qui ne peut pas être transféré dans le temps imparti et dont l'échec du transfert provoque l'envoi, par le RTS, de l'exception-ind).

préconditions

APDU est l'adresse de l'APDU présent dans la primitive TRANSFER qui a échoué.

Résultats

postconditions

2.2.2. LES PRIMITIVES OFFERTES PAR LA COUCHE SESSION.

PROCEDURE S-CONNECT.

LES TYPES

ADDRESS-TRANSP :

type qui renferme toutes des adresses transport. Toutes les adresses de ce type sont représentées par un string de caractères T.61.

QUALITY :

type reprenant les valeurs "Extended Control" ou "Optimized Dialogue Transfer" que peut demander le RTS ouvrant la connexion session en ce qui concerne la qualité de son service.

UNITS :

type reprenant toutes les unités fonctionnelles qui peuvent être choisies comme valeurs du paramètre Session Requirements lors d'une demande d'ouverture de connexion session.

UNITS = (Kernel, Exceptions, Activity Management, Half Duplex, Minor Synchronize).

CAUSE :

Type renfermant toutes les raisons d'échec qui peuvent expliquer l'envoi par le RTS récepteur, d'une primitive S-U-EXCEPTION-REPORT. Les raisons possibles sont:

Recuring ability jeopardized.

Local SS-User error.

Sequence error.

Unrecoverable procédure error.

Non specific error.

PRIORITY :

Ce type contient toutes les priorités que l'AM peut véhiculer lorsqu'il demande le tour. Ces priorités sont fonctions des actions qui peuvent provoquer la demande de tour et des valeurs qu'elles peuvent prendre sont fixées par le protocole application.

BUT:

Cette procédure permet au RTS d'obtenir l'ouverture d'une connexion session suite à un OPEN activé par l'AM ou suite à une initiative personnelle du RTS qui a besoin

d'utiliser une connexion session pour entrer en communication avec un MTA voisin.

SPECIFICATIONS:

S-CONNECT-REQ-SIGN.

Arguments

Session Connection Identifier ϵ INTEGER.

{c'est l'identificateur de la connexion que l'on veut créer}.

Calling SSAP address ϵ ADDRESS-TRANSP.

{adresse transport de l'entité vers laquelle on ouvre une connexion session}.

Quality of Service (QOS) ϵ QUALITY.

{qualité du service demandé par le RTS initiateur de la demande d'ouverture}.

Session Requirements ϵ ADDRESS - ZONE.

{ce paramètre reprend les unités fonctionnelles choisies par le RTS demandeur}.

Synchronisation Point Serial Number ϵ {0}.

{signale au RTS reception qu'il faut initialiser le contenu des points de synchronisation}.

Initial Data Token Assignment ϵ (Sending, Receiving, Acceptor Chooses).

(indique si c'est le RTS initiateur ou destinataire qui aura les jetons pour l'envoi des données).

Initial Minor Synch Token Assignment ϵ (Sending, Receiving, Acceptor chooses).

(indique à quel RTS va être attribué le jeton pour la synchronisation mineure).

Initial Major/Act Token Assignment ϵ (Sending, Receiving, Acceptor chooses).

(indique quel RTS va posséder le jeton pour la synchronisation majeure).

SS-User-Data ϵ ADDRESS - ZONE.

(ce paramètre contient la taille entre deux points de synchronisation, la taille de la fenêtre, le mode de dialogue, le protocole d'application et l'indication de la cause de la connexion c'est-à-dire OPEN ou RECOVER).

préconditions

Session Connection Identifier contient une valeur différente de toutes celles qui ont été attribuées précédemment au même paramètre pour d'autres S-CONNECT-REQ-SIGN.

Calling SSAP Address contient l'adresse transport du RTS origine.

Called SSAP Address contient l'adresse transport du RTS destinataire.

Session Requirements est l'adresse d'une zone contenant les unités fonctionnelles choisies par le RTS émetteur. Chaque unité fonctionnelle reprise à cette adresse doit être du type UNITS.

la valeur du paramètre Initial Data Token Assignment, dans le cas d'un S-CONNECT-REQ-SIGN déclenché par un OPEN, doit être la même que la valeur de Initial-Turn dans OPEN-REQ-SIGN.

les jetons de données, de synchronisation majeure et mineure sont tous fournis au même RTS.

SS-User-Data est l'adresse de la zone qui contient:

la taille entre deux points de synchronisation (qui définit le volume de données entre deux points de synchronisation) ϵ INTEGER.

la taille de la fenêtre (qui définit le nombre de points de synchronisation mineure avant que l'envoi ne soit interrompu) ϵ INTEGER.

le mode de dialogue ϵ {monologue, dialogue à l'alternat}.

le protocole d'application ϵ {P1 ou P3}.

l'indication de raison de connexion ϵ {OPEN, RECOVER}.

Résultats

postconditions

S-CONNECT-IND-RESP

Arguments

Session connection Identifier ∈ INTEGER.

(identificateur de la connexion ouverte par le RTS origine).

Calling SSAP Address ∈ ADDRESS-TRANSP.

(adresse transport du RTS émetteur).

Called SSAP Address ∈ ADDRESS-TRANSP.

(adresse transport du RTS destinataire).

Quality of Service ∈ QUALITY.

(qualité de service, demandée par l'émetteur, pour la connexion qu'il veut ouvrir).

Session Requirements ∈ ADDRESS - ZONE.

(contient les unités fonctionnelles choisies par le RTS appelant).

Synchronisation Point Serial Number ∈ (0).

(ce paramètre indique qu'il faut initialiser à 0 le compteur des points de synchronisation).

Initial Data Token Assignment ∈ (Sending, Receiving, Acceptor chooses).

(valeur indiquant au RTS receveur s'il obtient le jeton de données, s'il le donne ou s'il peut choisir de le garder ou de le donner).

Initial Minor Synch Token Assignment ϵ {Sending, Receiving, Acceptor chooses}.

{contient l'indication de celui qui possèdera le jeton de synchronisation mineure. Par ce paramètre, le RTS receveur peut garder le jeton, le donner ou choisir de le garder ou pas}.

Initial Major/Act Token Assignment ϵ {Sending, Receiving, Acceptor chooses}.

{indique au RTS receveur s'il obtient ou pas le jeton de synchronisation majeure ou s'il a le choix de le garder ou pas}.

SS-User-Data ϵ ADDRESS - ZONE.

{indique au RTS receveur la taille de la fenêtre, la taille entre deux points de synchronisation, le mode de dialogue, le protocole d'application fourni par le RTS émetteur ainsi que l'indication OPEN ou RECOVER}.

préconditions

Session Connection Identifier possède la même valeur que le paramètre Session Connection Identifier de la primitive S-CONNECT-REQ-SIGN envoyée par le RTS origine.

Calling SSAP Address a la même valeur que celle contenue dans le paramètre du même nom de la primitive S-CONNECT-REQ-SIGN émise par le RTS demandeur.

Called SSAP Address contient la même valeur que le paramètre du même nom initialisé dans la primitive S-CONNECT-REQ-SIGN.

Quality of Service contient la même valeur que celle qui est contenue dans le paramètre du même nom initialisé par le RTS émetteur dans la primitive S-CONNECT-REQ-SIGN.

Session Requirements contient exactement les mêmes unités fonctionnelles que celles qui ont été données par le RTS initiateur de la demande S-CONNECT-REQ-SIGN.

la valeur de Initial Data Token Assignment est la même que celle qui est donnée pour ce paramètre dans la primitive S-CONNECT-REQ-SIGN.

Initial Minor Synch Token Assignment possède la même valeur que dans la primitive S-CONNECT-REQ-SIGN.

Initial Major/Act Token Assignment a exactement la même valeur que celle qu'il avait lors de l'envoi de la primitive S-CONNECT-REQ-SIGN.

les trois paramètres cités ci-dessus doivent avoir la même valeur.

SS-User-Data est la même adresse que celle qui a été envoyée par le RTS envoyeur dans la primitive S-CONNECT-REQ-SIGN.

Called SSAP Address est l'adresse transport du RTS récepteur.

Résultats

Session Connection Identifier ∈ INTEGER.

(contient l'identificateur de la connexion session ouverte).

Called SSAP Address ∈ ADDRESS-TRANSP.

(adresse transport du RTS récepteur de la demande d'ouverture de connexion session).

Result ϵ {Accept, Reject}.

{résultat de la demande d'ouverture de connexion}.

Session Requirements ϵ ADDRESS - ZONE.

{contient les unités fonctionnelles dont on disposera pour gérer la connexion session}.

Synchronisation Point Serial number ϵ {0}.

{indique au RTS émetteur qu'il doit initialiser à 0 le compteur des points de synchronisation}.

Initial Data Token Assignment ϵ {Sending, Receiving}.

{indication par le RTS récepteur de la demande d'ouverture, de qui possèdera le jeton des données}.

Initial Minor Synch Token Assignment ϵ {Sending, Receiving}.

{indique au RTS initiateur de l'ouverture, qui du RTS émetteur ou récepteur possède le jeton de synchronisation mineure}.

Initial Major/ACT Token Assignment ϵ {Sending, Receiving}.

{procure au RTS émetteur de la demande d'ouverture l'indication du RTS qui possède le jeton de synchronisation majeure}.

SS-User-Data.

{contient les informations en rapport avec le résultat fourni par le RTS récepteur}.

postconditions

Called SSAP Address est l'adresse transport du RTS émetteur de la primitive S-CONNECT-IND-RESP.

Initial Data Token Assignment possède une valeur compatible avec ce qui a été indiqué en entrée dans ce paramètre.

Initial Minor Synch Token Assignment possède une valeur concordant avec la valeur de ce paramètre en entrée.

Initial Major/Act Token Assignment contient une valeur compatible avec ce que ce paramètre indique en entrée de la procédure.

Les valeurs contenues dans les trois paramètres précédents sont toutes identiques.

SS-User-Data contient un Paccept ou un Prefuse selon la valeur du paramètre résultat.

Paccept = taille entre deux points de synchronisation, taille de la fenêtre, l'indication OPEN ou RECOVER.

Prefuse = raison d'échec.

Taille point synchronisation ϵ INTEGER.

Taille de la fenêtre ϵ INTEGER.

Indication connection data ϵ {OPEN, RECOVER}.

Raison d'échec ϵ {RTSbusy, Cannotrecover, Validationfailure, unacceptable Dialogue Mode}.

S-CONNECT-CONF.

Arguments

Session Connection Identifier ϵ INTEGER.

(contient l'identificateur de la connexion session qui a été donnée par le RTS émetteur dans la primitive S-CONNECT-REQ-SIGN).

Called SSAP Address ϵ ADDRESS-TRANSP.

(contient l'adresse transport du RTS avec lequel on a voulu ouvrir une connexion session).

Result ϵ (Accept, Reject).

(contient le résultat de la demande d'ouverture avec le RTS voisin).

Quality of Service ϵ QUALITY.

(véhicule la qualité de service confirmée par le RTS récepteur de la demande d'ouverture).

Session Requirements ϵ ADDRESS - ZONE.

(contient les unités fonctionnelles sur lesquelles on peut se baser pour utiliser la connexion session).

Synchronisation Point Serial Number ϵ (0).

(sert à l'initialisation du numéro de point de synchronisation).

Initial Data Token Assignment ϵ (Sending, Receiving).

(indique au RTS émetteur s'il possède le jeton des données ou pas).

initial Minor Synch Token Assignment ϵ {Sending, Receiving}.

{indique au RTS initiateur de la demande d'ouverture de connexion s'il possède le jeton de synchronisation mineure ou pas}.

Initial Major/Act Token Assignment ϵ {Sending, Receiving}.

{signale au RTS s'il possède ou pas le jeton de synchronisation majeure}.

SS-User-Data ϵ ADDRESS - ZONE.

{contient des informations venant depuis le RTS récepteur de la demande d'ouverture}.

préconditions

Tous les paramètres de cette procédure contiennent exactement les mêmes valeurs que celles qu'avaient les paramètres du même nom dans la primitive S-CONNECT-IND-RESP.

Called SSAP Address est l'adresse du RTS avec lequel on a essayé d'ouvrir une connexion session.

Result indique si oui ou non, la connexion session est établie.

Quality of Service retourne la réponse choisie par le récepteur de la demande d'ouverture.

Initial Data Token Assignment, Initial Minor Synch Token Assignment et Initial Major/Act Token Assignment contiennent tous trois la même valeur.

SS-User-Data contient un Paccept ou un Prefuse selon la valeur du paramètre Result.

Un Paccept est formé de la taille entre deux points de synchronisation, de la taille de la fenêtre et de l'indication OPEN ou RECOVER.

Un Prefuse contient la raison d'échec.

les valeurs que peuvent prendre ces informations sont :

taille entre deux points de synchronisation ϵ INTEGER.

taille de la fenêtre ϵ INTEGER.

indication ϵ {OPEN, RECOVER}.

raison d'échec ϵ {OPEN, RECOVER}.

raison d'échec ϵ {RTSbusy, Cannotrecover, Validationfailure, unacceptable Dialogue Mode}.

Résultats

postconditions

PROCEDURE S-ACTIVITY-START

BUT :

Cette procédure permet de commencer une nouvelle activité pour envoyer un MPDU vers un MTA voisin. Pour pouvoir entamer une nouvelle activité, le RTS doit posséder les jetons. Après avoir envoyé la demande de création d'activité, le RTS émetteur peut transmettre les premières données. Une activité existe le temps d'un transfert d'un MPDU.

SPECIFICATIONS:

S-ACTIVITY-START-REQ-SIGN.

Arguments

Activity Identifier ϵ INTEGER.

{permet d'identifier l'activité que le RTS veut entamer}.

préconditions

Cet argument ne peut pas contenir une valeur qui a déjà été choisie pour identifier une autre activité sur la même connexion session.

Résultats

postconditions

S-ACTIVITY-START-IND.

Arguments

Activity Identifier ϵ INTEGER.

{c'est la valeur permettant d'identifier l'activité demandée par le RTS appelant}.

préconditions

La valeur de cet argument est la même que celle qui est contenue dans le paramètre S-ACTIVITY-START-REQ-SIGN.

Résultats

postconditions

PROCEDURE S-SYNCH-MINOR

BUT :

La pose de point de synchronisation sert à la vérification de la bonne réception des données véhiculées entre deux RTS. Lorsqu'il reçoit un point de synchronisation suivant un envoi de données, le RTS destinataire renvoie à l'émetteur une confirmation qui lui permet de savoir que les données transmises avant ce point de synchronisation sont bien arrivées.

La pose de point de synchronisation ne peut être effectuée que si Checkpointsize est différent de zéro et si le RTS émetteur possède les jetons.

SPECIFICATIONS:

S-SYNCH-MINOR-REQ-SIGN.

Arguments

Type ∈ {explicit confirmation expected}.

{indique le type de confirmation attendu. Le seul type existant est la demande explicite de confirmation de tous les points de synchronisation}.

Synchronisation Point Serial Number ∈ INTEGER.

{contient le numéro de série du point de synchronisation mineure émis par le RTS envoyeur, durant l'activité courante}.

préconditions

Synchronisation Point Serial Number est un nombre entier non encore choisi pour désigner un point de synchronisation mineure.

Résultats

postconditions

S-SYNCH-MINOR-IND-RESP.

Arguments

Type ∈ (Explicit Confirmation Expected).

(cet argument contient l'indication de la confirmation demandée par le RTS émetteur).

Synchronisation Point Serial Number ∈ INTEGER.

(contient le numéro du point de synchronisation qui suit les données dernièrement envoyées).

préconditions

La valeur de l'argument Type est la même que celle contenue dans la primitive S-SYNCH-MINOR-REQ-SIGN.

L'argument Synchronisation Point Serial Number a la même valeur que celle qui est contenue dans l'argument du même nom dans la primitive S-SYNCH-MINOR-REQ-SIGN.

La valeur de Synchronisation Point Serial Number est le numéro de point de synchronisation envoyé juste après le dernier bloc de données émis.

Résultats

Synchronisation Point Serial Number ∈ INTEGER.

(contient le numéro du point de synchronisation confirmé par le RTS destinataire).

postconditions

L'argument Synchronisation Point Serial Number contient le numéro du point de synchronisation suivant le bloc de données bien reçu par le RTS destinataire.

S-SYNCH-MINOR-CONF.

Arguments

Synchronisation Point Serial Number ∈ INTEGER.

{numéro du point de synchronisation mineure confirmé par le RTS destinataire}.

préconditions

Cet argument a exactement la valeur qu'il avait en sortie de la primitive S-SYNCH-MINOR-IND-RESP.

Résultats

/

postconditions

/

PROCEDURE S-ACTIVITY-END.

BUT :

Cette procédure a pour but de terminer proprement l'envoi d'un MPDU. Une fin d'activité est en fait un point de synchronisation majeure. Une fois confirmé, il indique aux deux RTS que la transmission du MPDU est terminée. Le MPDU peut donc être effacé du RTS envoyeur. La procédure S-ACTIVITY-END ne peut être déclenchée que par le RTS qui possède les jetons.

SPECIFICATIONS:

S-ACTIVITY-END-REQ-SIGN.

Arguments

Synchronisation Point Serial Number ∈ INTEGER.

(contient le numéro de point de synchronisation majeure).

préconditions

Synchronisation Point Serial Number est un numéro alloué par la couche session. Ce numéro doit être différent des autres numéros choisis de points de synchronisation majeure pour identifier les activités sur la connexion session.

Résultats

postconditions

S-ACTIVITY-END-IND-RESP.

Arguments

Synchronisation Point Serial Number ϵ INTEGER.

{indique le numéro du point de synchronisation majeure}.

préconditions

Cet argument contient exactement la même valeur que dans la primitive S-ACTIVITY-END-REQ-SIGN.

Résultats

Synchronisation Point Serial Number ϵ INTEGER.

{indique le numéro du point de synchronisation majeure}.

postconditions

La valeur de Synchronisation Point Serial Number indique au RTS émetteur le numéro du point de synchronisation majeure spécifiant l'activité terminée. Ce numéro est celui qui a été envoyé par le RTS appelant après l'activité que le RTS récepteur veut confirmer.

S-ACTIVITY-END-CONF.

Arguments

Synchronisation Point Serial Number ∈ INTEGER.

(indique le numéro du point de synchronisation majeure).

préconditions

La valeur de cet argument est identique à celle qui est contenue par cet argument en sortie de la primitive S-ACTIVITY-END-IND-RESP.

Résultats

/

postconditions

/

PROCEDURE S-U-EXCEPTION-REPORT.

BUT :

Cette procédure permet au RTS destinataire de signaler au RTS émetteur un problème de faible importance, c'est-à-dire un problème que le RTS émetteur pourrait solutionner.

SPECIFICATIONS:

S-U-EXCEPTION-REPORT-REQ-SIGN

Arguments

Reason ∈ CAUSE

(indique la raison pour laquelle le RTS destinataire utilise une telle procédure).

préconditions

/

Résultats

/

postconditions

/

S-U-EXCEPTION-REPORT-IND.

Arguments

Reason ∈ CAUSE

(contient la raison expliquant l'échec).

préconditions

Cet argument a la même valeur que dans la primitive S-U-EXCEPTION-REPORT-REQ-SIGN.

Résultats

postconditions

PROCEDURE S-TOKEN-PLEASE.

BUT :

Lorsque l'AM demande le tour, le RTS doit demander les jetons. Cette procédure est donc la conséquence d'une procédure TURN-PLEASE-REQ-SIGN envoyée par l'AM. L'envoi de cette procédure de demande de jetons peut être effectuée, uniquement par un RTS sans jeton, à l'intérieur ou à l'extérieur d'une activité. Lorsqu'il reçoit la demande de jeton, le RTS destinataire le fait savoir à son AM par l'intermédiaire de la procédure TURN-PLEASE-IND.

SPECIFICATIONS:

S-TOKEN-PLEASE-REQ-SIGN.

Arguments

Tokens ∈ {Data}.

{spécifie le type de jeton demandé par le RTS émetteur}.

SS-User-Data ϵ PRIORITY.

(indique la priorité de la demande de tour).

préconditions

Tokens doit toujours être mis à "Data" car, de toutes façons, pour la messagerie électronique, les trois jetons sont toujours ensemble.

SS-User-Data est de la valeur du paramètre PRIORITY de la primitive TURN-PLEASE-REQ-SIGN qui a initialisé la demande de jetons.

Résultats

postconditions

S-TOKEN-PLEASE-IND.

Arguments

Tokens ϵ (Data).

(spécifie au RTS destinataire le jeton demandé).

SS-User-Data ϵ PRIORITY.

(indique au RTS destinataire la priorité donnée par l'AM à la demande).

préconditions

Ces arguments ont les mêmes valeurs que dans la procédure S-TOKEN-PLEASE-REQ-SIGN.

Lorsqu'il envoie une telle primitive, le RTS donne toujours les trois jetons ensemble.

Résultats

postconditions

PROCEDURE S-ACTIVITY-RESUME.

BUT :

Cette procédure permet à un RTS de reprendre une activité (c'est-à-dire un transfert de MPDU) qui a été interrompue précédemment.

SPECIFICATIONS:

S-ACTIVITY-RESUME-REQ-SIGN.

Arguments

Activity Identifier ϵ INTEGER.

(numéro courant de l'activité qui va être recréée pour faire le "resume").

Old Activity Identifier ϵ INTEGER.

(numéro de l'activité interrompue à reprendre).

Synchronisation Point Serial Number ϵ INTEGER.

(numéro du dernier point de synchronisation mineure confirmé).

Old Session Connection Identifier ϵ INTEGER.

(numéro identifiant la connexion durant laquelle il y a eu interruption d'activité).

préconditions

Old Activity Identifier est le numéro d'une activité interrompue.

Activity Identifier est un numéro non encore attribué à une activité.

Old Session Connection Identifier est l'identificateur du connexion session ayant existé.

Résultats

postconditions

S-ACTIVITY-RESUME-IND.

Arguments

Activity Identifier ϵ INTEGER

(identificateur de l'activité que l'on crée pour reprendre celle qui a été interrompue).

Old Activity Identifier ϵ INTEGER.

(identificateur de l'activité interrompue que l'on désire reprendre).

Synchronisation Point Serial Number ϵ INTEGER.

(numéro du dernier point de synchronisation mineure confirmé lors de l'activité interrompue).

Old Session Connection Identifier ϵ INTEGER.

(identificateur de la connexion session utilisée pour véhiculer l'activité interrompue).

préconditions

les valeurs de ces arguments sont identiques à celles qui se trouvent dans la primitive S-ACTIVITY-RESUME-REQ-SIGN.

Résultats

/

postconditions

/

PROCEDURE S-CONTROL-GIVE.

BUT :

Cette procédure, utilisée par un RTS possédant les jetons, lui permet de les passer au RTS avec lequel il est en relation. Cette procédure suit toujours un don du tour par l'AM.

SPECIFICATIONS:

S-CONTROL-GIVE-REQ-SIGN.

Arguments

/

préconditions

/

Résultats

/

postconditions

/

S-CONTROL-GIVE-REQ-IND.

Arguments

préconditions

Résultats

postconditions

PROCEDURE S-RELEASE.

BUT :

Le RTS qui possède les jetons peut, grâce à cette procédure, terminer proprement la connexion session. Cette procédure est une réponse, par le RTS, au CLOSE déclenché par l'AM.

SPECIFICATIONS:

S-RELEASE-REQ-SIGN.

Arguments

préconditions

Résultats

postconditions

S-RELEASE-IND-RESP.

Arguments

Result ϵ {" "}.

{décrit le résultat de la demande de terminaison de connexion session}.

préconditions

Résultats

Result ϵ {accept}.

{indique au RTS émetteur le résultat de la demande d'abandon de la connexion. Le RTS récepteur doit toujours accepter}.

postconditions

S-RELEASE-CONF.

Arguments

Result ∈ {Accept}.

{donne au RTS émetteur la réponse de la demande de fermeture de la connexion session fournie par le RTS récepteur}.

préconditions

Result contient la même valeur qu'en sortie de la procédure S-RELEASE-IND-RESP.

Résultats

postconditions

PROCEDURE S-U-ABORT.

BUT :

Cette procédure permet au RTS de couper la connexion, s'il rencontre un problème grave à l'intérieur ou à l'extérieur d'une activité.

SPECIFICATIONS:

S-U-ABORT-REQ-SIGN.

Arguments

User Data ϵ ADDRESS - ZONE.

(informe le RTS à propos de la coupure).

préconditions

User Data est l'adresse qui contient la raison de l'Abort qui peut être :

Local System Problem.

Invalid Parameter

Unrecognized Activity.

Temporary Problem.

Protocol Error.

Lorsqu'il s'agit d'un paramètre invalide, ce paramètre est également retourné dans cette primitive.

Résultats

postconditions

S-U-ABORT-IND.

Arguments

User Data ∈ ADDRESS - ZONE.

(renseigne le RTS récepteur sur la raison de l'Abort).

préconditions

User Data contient la même adresse que dans la primitive S-U-ABORT-REQ-SIGN.

Résultats

postconditions

PROCEDURE S-P-ABORT.

BUT :

Cette procédure peut être reçue lorsque la couche Session veut annoncer qu'un problème a surgi à son niveau.

SPECIFICATIONS:

S-P-ABORT-IND.

Arguments

Reason ϵ (Transport Disconnect, Protocol Error, Undefined).

préconditions

Résultats

postconditions

Annexe 3 .

LES OUTILS UNIX

Sans aucun doute d'un grand intérêt, UNIX est un système d'exploitation connu et apprécié pour les caractéristiques qu'il présente.

Ainsi, son système de fichiers prend l'apparence d'une structure hiérarchique. Côté utilisateur, il est vu comme une arborescence qui, en guise de noeuds, abrite des répertoires (ou directories), lesquels conduisent pas à pas aux feuilles de l'arbre : les fichiers ordinaires. Les répertoires sont des fichiers particuliers. Il assurent la correspondance entre les fichiers et les noms par lesquels on les désigne. Parcourir une partie de la hiérarchie, partant du sommet (ou racine) jusqu'à une feuille, c'est en fait parcourir un chemin d'accès vers le fichier-feuille. Ce chemin, à lui seul, sert de moyen de désignation pour le fichier.

Outre le fait d'offrir une structure organisée et stable, le système de fichiers UNIX est important puisque le fichier est un concept fondamental dans ce système d'exploitation. Le fichier est en effet utilisé pour représenter de manière uniforme tous les objets de communication présents dans le système : les fichiers proprement dits, les répertoires, les périphériques, les supports de communication inter-processus,... Dès lors l'utilisateur et ses programmes d'application ne manipule qu'un même type d'objet à l'aide des mêmes appels système. Ceci permet une vision uniforme des entrées/sorties : que l'on demande une écriture sur un fichier utilisateur, un disque, une imprimante, ou à l'écran, la requête à formuler ne varie guère. Il s'agit chaque fois d'une instruction d'écriture (WRITE) sur un fichier désigné par un identificateur, lequel permet d'avertir le système du type de support choisi pour l'écriture; le système peut alors en déduire la procédure à suivre.

D'autre part, UNIX présente un attrait essentiel pour ses utilisateurs, il vient du fait qu'il s'agit d'un système très ouvert, très modularisé, du type " boîte à outils ", qui offre la possibilité d'étendre l'ensemble des commandes disponibles dans le système. L'utilisateur peut lui-même concevoir et implanter de nouveaux utilitaires et ensuite les intégrer au système d'exploitation. Sans doute y trouve-t-il grand avantage puisqu'il peut, s'il le désire, se tailler un système mieux adapté à ses besoins.

Suivant l'habitude en ce domaine, UNIX travaille en s'appuyant sur la notion de processus. Une de ses caractéristiques, est qu'il permet à un programme de lancer l'exécution de processus asynchrones, via des appels système appropriés.

Une dernière particularité importante est l'absence de mémoire partagée entre les différents processus. Ceci n'empêche cependant pas un certain type de communication inter-processus : il faut, pour la circonstance, utiliser le mécanisme de " pipe " (ou "tube") qui établit un conduit entre deux processus, suivant le schéma producteur/consommateur.

En complément, on peut établir la liste des appels systèmes fondamentaux de UNIX :

FORK :

établit une copie conforme du processus appelant, surnommé 'processus père'. La copie, appelée 'processus fils', jouit d'une existence propre.

EXEC :

remplace le processus appelant par le processus que spécifie le nom du programme donné en argument.

WAIT :

place le processus appelant en attente d'un événement, que ce soit un signal à recevoir ou la fin du déroulement d'un de ses enfants.

PIPE :

il crée un tuyau de communication entre deux processus sous forme d'un fichier géré de manière FIFO. L'un des deux processus joue le rôle de lecteur et lit les informations que contient le pipe dans l'ordre où l'autre processus, appelé écrivain, les y aura écrites.

SIGNAL :

accorde la possibilité de définir l'action à entreprendre à la réception d'un signal particulier; il peut encore demander d'ignorer le signal ou d'effectuer une action définie par défaut.

Par l'intermédiaire de ces appels systèmes et des instructions de lecture/écriture sur un fichier, UNIX fournit des outils qui permettent, par exemple, à un processus de lancer, au sein de son exécution, le déroulement parallèle d'un autre processus avec lequel il peut, via un pipe, échanger des informations.

Cependant, ces pipes, si importants puisque la coopération inter-processus est souvent essentielle et qu'elle repose sur l'échange d'information, imposent de lourdes restrictions : ils ne peuvent voir le jour qu'entre un processus père et un fils, ou par l'intermédiaire du père, entre deux de ses fils, conformément aux figures An 3.1 et An 3.2, où les flèches peuvent être selon les besoins, renversées.

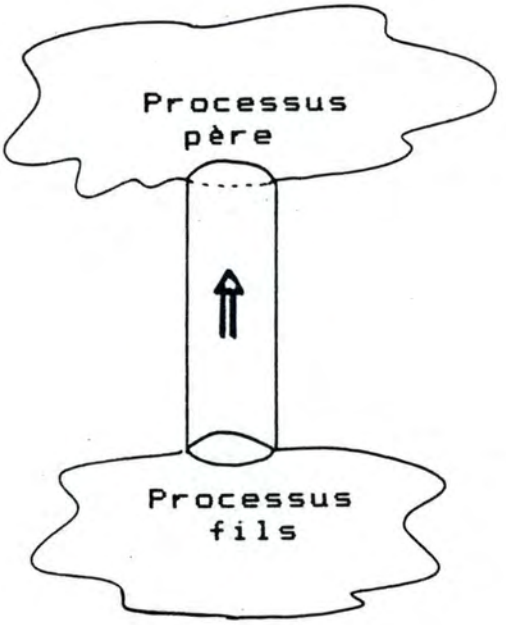


FIGURE AN 3.1 : COMMUNICATION ENTRE PERE ET FILS

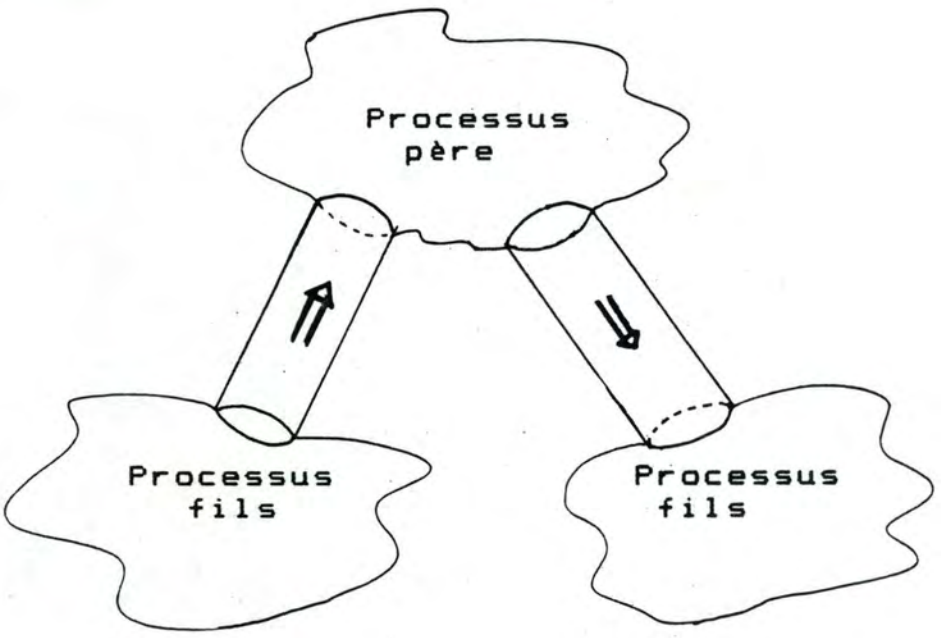


FIGURE AN 3.2 : COMMUNICATION ENTRE DEUX FILS

Il est à noter que la restriction père-fils provient du fait que les tubes - ou pipes - ne sont que des fichiers particuliers. Or, dans le système UNIX, qui évite

les accès partagés, un processus ne peut travailler que sur les fichiers qu'il a ouverts, ou sur ceux qu'a ouverts son père et dont il a hérité lors de sa création par FORK/EXEC. Le pipe étant un fichier parmi les autres, seuls père et fils sont autorisés à l'utiliser; et donc la génération d'un pipe entre deux processus exige un lien de filiation entre eux.

Cependant, souvent, dans les systèmes d'exploitation-mêmes, apparaît le besoin d'assurer une communication entre processus utilisateurs et des services assurés par des processus serveurs permanents, sans filiation proche commune. Il s'agit, par exemple, d'un gestionnaire d'imprimante gouvernant l'impression des fichiers pour tous les utilisateurs du système. Dans ce cadre, UNIX version 7 dispose de fichiers disque partageables entre tous les processus du système. Si bien que, dans le système lui-même, en mode Kernel, mode privilégié, propre au système, une communication inter-processus dans un sens plus large que père/fils peut être établie.

Annexe 4.

ALGORITHME DU GI-UA-MTA.

```
DEBUT :

POUR id-pipe de MD1 à MDder
      (* regarder dans table 17 les machines créées càd bit à 1
*)
FAIRE
 POUR première demande jusque dernière demande
 FAIRE
  SI [A]=0
  ALORS
   supprimer [A]
   écrire le message dans le pipe 1
  SINON
   supprimer [A]
   écrire  le message  en fin  de file  d'attente précédé du n°
   MDi
  FINSI
 FINFAIRE
FINFAIRE

POUR pipe 3
FAIRE
 POUR première demande jusque dernière demande
 FAIRE
  écrire le message en fin de file d'attente précédé du n°3
 FINFAIRE
FINFAIRE


lire DEM qui est la première demande de la file d'attente

SI DEM.pipe.or=3
ALORS
 SI  DEM.[B]=null
 ALORS  (* le  message vient  du GIMTAMTA  et  il  s'agit  d'un
 relais *)
  call CREER MD (bool,MDi,MDi)
  SI bool
  ALORS
   maj table 16                MDi  0 null  relais
   écrire DEM dans pipe n° MDI (* goto DEBUT *)
  SINON
```

```
 envoyer  0  DEMNOTNL  RELAIS(APDU)  à un MDj
    (*  rem pour choix du MD : ne pas donner toujours au même
 --> var. globale, priorité aux MD non connectés *)
 incrémenter le verrou de 1 dans la table 16 d'entrée MDj
FINSI
SINON
 SI DEM.[B]=-1 (* vient du GIMTAMTA *) (* Demnotnl *)
 ALORS
  call CREER MD (bool,MDi,MDi)
  SI bool
  ALORS
   maj table 16                          MDi  0
   écrire  DEM   -1     DEMNOTNL  dans pipe  MDI (* goto DEBUT
   *)
  SINON
   envoyer  DEM  -1     DEMNOTNL   à un MDj
   incrémenter le verrou de 1 dans la table 16 d'entrée MDj
  FINSI
 SINON (* la demande vient de l'UA via GC *)
  lire id-UA=DEM.[B]
  SI id-UA est une entrée dans table 16
  ALORS (* un MD est déjà attribué à cet UA *)
   SI bit-connexion=1
   ALORS (* l'UA est déjà connecté *)
    enlever le [B]
    écrire DEM dans pipe n° donné par table 16 précédé de 0
   SINON (* il est logout *)
    enlever le [B]
    SI ce n'est pas un logon
    ALORS
     traitement d'erreurs
     envoyer  0  id-UA  "login please"
    SINON
     vérification des paramètres du logon (* table 2 *)
     SI OK
     ALORS
      émettre un LOGON-CONF+
      écrire dans pipe 1  0  id-UA  LOGON-CONF
      mettre le bit à 1 dans table 16
     SINON
      émettre un LOGON-CONF-
      écrire dans pipe 1  0  id-UA  LOGON-CONF
```

```
  SINON (* l'UA n'a pas de MD *)
   SI id-UA est dans table 2
   ALORS (* id-UA abonné au MTA *)
    SI LOGON reason echec=' '
    ALORS
     call EXIST MD (bool)
     SI bool (* il existe encore un MD disponible *)
     ALORS
      vérification des paramètres (* table 2 *)
      SI OK
      ALORS
       call CREER MD (bool,MDi,MDi)
        ajouter une ligne dans table 16  nom UA  id-UA  MDi
1  0
       écrire dans pipe 1  0  id-UA  LOGON-CONF+
       écrire dans pipe MDi  1  id-UA  table 4
      SINON
       reason echec=' paramètres incorrects '
      FINSI
     SINON
      reason echec=' busy '
     FINSI
     SI reason echec n'est pas ' '
     ALORS
      écrire dans pipe 1  0  id-UA  LOGON-CONF-
     FINSI
    SINON
     traitement d'erreurs
     écrire dans pipe 1  0  id-UA  "login please"
    FINSI
   SINON
    traitement d'erreurs
    envoyer dans pipep 1  0  ib-UA  " Qui êtes-vous ?"
   FINSI
  FINSI
 FINSI
SINON  (* la  demande vient d'un MD et ce n'est pas pour le
GC *)
      (* le message est  pipe-or=MDi  demande  *)

  CASE [C]=1  :   (* MDi  1  table 4 *) (* signal de logoff
*)
```

```
 SI bit =0 dans table 16 d'entrée MDi
 ALORS traitement d'erreurs : envoyer dans pipe 1
     0  0  id-UA  "déjà logoff"
 SINON
  mettre le bit à 0 dans table 16 d'entrée MDi
  maj table 4 (* register *)
 FINSI

CASE [C]=2 :  MDi  2  n° machine  dernier n° généré
   (* je n'ai plus rien à faire *)
 SI bit de connexion dans table 16 d'entrée MDi =0
 ALORS
  SI verrou dans table 16 de même entrée =0
  ALORS
   maj table 17
   changer bit d'occupation
   changer n° généré
   envoyer dans MDi  1  pipe bouché
   maj table 16 d'entrée MDi : effacer la ligne
  FINSI
 FINSI

CASE [C]=3 :  (* demande de MD connecté à un UA dest *)
   MDi  3  id-UA  message
 SI id-UA dans table 16
 ALORS
  SI bit de connexion=1
  ALORS
     envoyer dans pipe MDj (* d'entrée id-UA dans table 16
*) 0  message
     incrémenter  le verrou  de 1  dans table  16 de  même
entrée
   SINON
    créer un MTL-LOGON-SIGN
    l'envoyer dans pipe 1  0  id-UA  MTL-LOGON-SIGN
    attendre la réponse MTL-LOGON-RESPONSE
    call DECOUPE LOGON (primitive,success indication)
    SI SUCCESS-INDICATION
    ALORS
        maj table  16 d'entrée  id-UA (* verrou ++1, bit de
connexion à 1 *)
     envoyer dans MDj (* d'entrée id-UA ( table 16 )*)
```

```
      1  id-UA  table 4
      envoyer dans MDj  0  message
     SINON
      envoyer dans MDi  0  DEMNOTNL  message
        incrémenter le verrou de 1 dans la table 16 d'entrée
MDi
     FINSI
    FINSI
   SINON (* id-UA n'a pas de MD associé *)
     SI id-UA=null  dans table  16 d'entrée  MDi (*  le  MD
demandeur n'est pas connecté *)
    ALORS
     créer un MTL-LOGON-SIGN
     envoyer dans pipe 1  0  id-UA  MTL-LOGON-SIGN
     attendre le MTL-LOGON-RESPONSE
     call DECOUPE LOGON (primitive,success indication)
     SI SUCCESS-INDICATION
     ALORS
        maj table 16 d'entrée MDi (* verrou ++1, id-ua, bit
de connexion =1 *)
      envoyer dans MDi  1  id-UA  table 4
      envoyer dans MDi  0  message
     SINON
      envoyer dans MDi  0  DEMNOTNL  message
      incrémenter le verrou de 1 dans table 16 d'entrée MDi
     FINSI
      SINON (*  le MD demandeur est déjà connecté à un autre
id-UA *)
     call EXIST-MD (bool)
     SI bool
     ALORS
      créer un MTL-LOGON-SIGN
      envoyer dans pipe 1  0  id-UA  MTL-LOGON-SIGN
      attendre le MTL-LOGON-RESPONSE
      call DECOUPE LOGON (primitive,success indication)
      SI SUCCESS-INDICATION
      ALORS
       call CREER MD (bool,MDj,MDj)
          maj  table 16 (* créer une ligne *)  nom UA  id-UA
MDj  MDj  1  1
       envoyer dans pipe MDJ  1  id-UA  table 4
       envoyer dans pipe MDj  0  message
```

```
        SINON
         envoyer dans pipe MDi  O  DEMNOTNL  message
            incrémenter  le verrou de 1 dans table 16 d'entrée
    MDi
         FINSI
        SINON
         envoyer dans pipe MDi  O  DEMNOTNL  message
         incrémenter le verrou de 1 dans table 16 d'entrée MDi
       FINSI
      FINSI
     FINSI;
    FINCASE
 FINSI;
goto DEBUT;
END.
```

Annexe 5.

ALGORITHME DU MD.

5.1. ALGORITHME.

IDLE :

```
attendre la première demande
 (* ce sera le nom de la machine et le dernier n° connecté *)
mémoriser ce nom et ce dernier n° généré
goto NON CONNECTE
```

CONNECTE :

```
lire dans son pipe et dans la file différée
SI il existe demande différée pouvant être traitée
ALORS
 enlever de la file différée et placer dans la file d'attente
 du super MD
FINSI
mettre ce qu'il a trouvé dans la file d'attente
SI la file est vide
ALORS
 goto CONNECTE
SINON
 CASE première demande de la file est

   0  cancel   call CANCEL

   0  UAL-CONTROL-REQ   call CONTROL

   0  REGISTER-REQ   call REGISTER

   0  chge password   call UAL-CHANGE-PASSWORD

   -1  *  Demnotnl   call ANALYSE-RAPPORT-1

   0  Demnotnl  message   call ANALYSE-RAPPORT-4

   0  probe-req   call PROBE-REQ
     SI +
```

```
    ALORS
     PROBE-REQ
    FINSI
    SI -
    ALORS
     pas de suite à donner
    FINSI

  O  submit-req   call SUBMIT-REQ
    SI +
    ALORS
     SUBMIT-REQ
    FINSI
    SI -
    ALORS
     rien
    FINSI

   O   local-probe-deliver   param-probe   id-UA  orig    call
LOCAL-PROBE-DELIVER

   O   local-deliver   param-submit   id-UA orig   call LOCAL-
DELIVER

  O  MTL-control-resp   rien à faire

  O  notify   call ANALYSE-RAPPORT-6

  O  deliver-APDU   call DELIVER

  O  probe-deliver  APDU   call PROBE-DELIVER

  O  logoff   call LOGOFF


DEFAULT :

 ' Primitive reçue non acceptée  =>  on écarte '

ENDCASE
goto CONNECTE
```

NON-CONNECTE :

```
logonreçu  := false  ; lire  dans les pipes et mettre dans file
d'attente
```

```
SI file d'attente normale et file différée sont vides
ALORS
 envoyer  1  2  num machine  dernier n° généré
 goto NON-CONNECTE
SINON
 CASE première demande dans la file est

  NULL  RELAIS   call RELAIS

  -1  Demnotnl   call ANALYSE-RAPPORT-1 ( -1, APDU )

  0  Demnotnl   call ANALYSE-RAPPORT-7

  1  destruction   goto IDLE

  0   deliver   envoyer  1  3  id-UA  deliver  APDU   ; envoyer
  1  4

  0   local-deliver   envoyer   1   3    id-UA    local-deliver
  primitive   ; envoyer  1  4

  0   notify   envoyer   1   3   id-UA   notify   primitive   ;
  envoyer  1  4

  0   probe-deliver   envoyer  1  3  id-UA  probe-deliver  APDU
  ; envoyer  1  4

  0   local-probe-deliver   envoyer   1  3  id-UA  local-probe-
  deliver  primitive   ; envoyer  1  4

  0  logon  écarter demande

  0  submit-req  écarter demande

  0  probe-req  écarter demande

  0  cancel  écarter demande

  0  UAL-control-req  écarter demande

  0  register-req  écarter demande

  0  chge password  écarter demande

  0  logoff  écarter demande

  1  id-UA  table 4   goto CONNECTE
```

5.2. PROCEDURES.

ANALYSE ROUTAGE1(IDUA,ADR,champs APDU)

```
call ALGO-ROUTAGE(ADR,LOCAL,NUMDEST)
 (* si non local => numdest = andest, sinon => numdest =
 identif d'UA *)
IF NOT LOCAL
THEN
 envoyer 0 1 AMDEST envoi(APDU)
ELSE
 IF IDUA = NUMDEST
 THEN
  call CREER-PRIM-NOTIFY(param APDU utiles,long,primitive)
  envoyer 0 0 idUA primitive
 ELSE
  (* c'est local => pas de notify MTA *)
  aller voir dans recipient info les bits 4 et 5
  (* pour la notif à UA *)
  SI c'est <> 00
  (* c-à-d le UA veut une notif *)
  ALORS
   call ANALYSE-RAPPORT5(NOMDEST,champs APDU)
  FINSI
 ELSE
  aller voir dans recipient info les bits 4 et 5
```

ANALYSE RAPPORT4(message,decoupe)

```
(* Ce qui est déclenché par la réception de 0 demnotnil
mess . Lui-même déclenché si le super MD reçoit qqch à livrer
mais n'arrive pas à trouver un MD connecté au bon UA *)
call DECOUPE-MESSAGE(message,type,nbrechamps,(champs)&)
CASE type OF
```

```
notify : FAIRE FINFAIRE
local deliver : FAIRE
 SI NDN suppress n'est pas un paramètre du submit
 ALORS
  SI  le paramètre  notifications  de  control  dans  table  4
  n'existe pas
  ALORS
   mettre la suite
  SINON
   écarter la demande
  FINSI
   non delivery reason = busy ou MTL-logon-rate
   call      CREER-NOTIFY-SN(param     submit,non     delivery
   reason,long,primitive)
   (*  on est dans le bon MD lié à l'UA auquel il faut envoyer
   la notif *)
   envoyer  O  O  idUA  primitive
  FINSI
 FINFAIRE
 local prob deliver : FAIRE
 SI  le  paramètre  notifications  de  control  dans  table  4
 n'existe pas
 ALORS
  mettre la suite
 SINON
  écarter la demande
 FINSI
  non delivery reason = busy ou MTL-logon-rate
  call  CREER-NOTIFY-PN(param prob,non  delivery reason, long,
  primitive)
  envoyer  O  O  idUA  primitive
FINFAIRE
relais : FAIRE (* on a un APDU *)
SI c'est un delivery report PDU
ALORS
SINON
 regarder  dans le recipientinfo les bits 2,3,4,5 et prendre le
 max pour ne creer qu'une notif.
 call   CREER-NOTIFY-APDU(champs  du  APDU  necessaires,  long,
 DRMPDU, busy)
 call           RECHERCHE-CHEMIN-COPIE(originator-or-name,nbre-
 direct,(1,dest,info)&,nbre UA inconnus,(nom UA)&)
```

```
 SI nbre UA inconnus = 0
 ALORS
  envoyer 0  1  dest  ENVOI  DRMPDU
 FINSI
FINFAIRE
FINCASE
envoyer  1  5
return.
```

PROBE-REQ

```
call   DECOUPE-PRIMITIVE(string,nbrechamps,champs 1,...,champs
n)
call  RECHERCHE-CHEMIN-COPIE(champs 1,nbre  direct,  (1,  dest,
info)&, nbre UA inconnus,(nom UA)&)
POUR chaque direction
FAIRE
 SI c'est local
 ALORS
  SI c'est le bon UA
  ALORS
   call  VERIFICATION-REGISTER  (bool1,  bool2,  bool3,  bool4,
   bool5, DEIT°, MCL°,DCS°,A°,ORAD°,result)
   non delivery reason = register
   SI result
   ALORS
    SI  le paramètre  notifications de  control  dans  table  4
    n'existe pas
    ALORS
     mettre la suite
    SINON
     écarter la demande
    FINSI
    call CREER-NOTIFY-PP(param du probe,long,primitive)
    envoyer  0  0  idUA  primitive
   SINON
    SI  le paramètre  notifications de  control  dans  table  4
    n'existe pas
    ALORS
     mettre la suite
    SINON
     écarter la demande
```

```
    FINSI
    call  CREER-NOTIFY-PN(param probe,non  delivery reason,
    long, primitive)
    envoyer  0  0  idUA  primitive
    SINON (* pas bon UA *)
      envoyer  1  3  id-UAdest  local-probe-deliver  param
    probe  id-UA orig
    SINON  (* probe à relayer *)
      call CREER-PROBE-PDU(param-probe,recipientinfo,dest,probe
    PDU)
     envoyer  0  1  dest  ENVOI  probe PDU
   FINFAIRE
   SI nbre UA non reconnus <> 0
   ALORS
    SI  le paramètre  notifications de  control  dans  table  4
    n'existe pas
    ALORS
     mettre la suite
    SINON
     écarter la demande
    FINSI
       call  CREER-NOTIFY-SN(param-probe,UA  non  recognized  +
    liste ,long,primitive)
    envoyer  0  0  idUA  primitive
FINSI
```

SUBMIT-REQ

```
call DECOUPE-PRIMITIVE(string,nbrechamps,(champs)&)
SI deffered delivery time existe
ALORS
 mettre la demande en file d'attente différée
SINON
 call  RECHERCHE-CHEMIN-COPIE(champs  1,nbre  direct,  (1,dest,
 info)&, nbre UA non reconnus,(nom UA)&)
 POUR chaque direction
 FAIRE
  SI c'est local
  ALORS
   SI c'est le bon UA
   ALORS
    call  VERIFICATION-REGISTER (bool1,  bool2,  bool3,  bool4,
    bool5, DEIT°,MCL°,DCS°,A°,ORAD°,result)
```

```
non delivery reason = register
SI result
 ALORS
 call CREER-LOCAL-DELIVER(param-submit,long,primitive)
 envoyer  0  0  idUA  primitive
 SI delivery notice dans paramètres du submit
 ALORS
   SI  le paramètre  notifications de  control dans table 4
n'existe pas
  ALORS
   mettre la suite
  SINON
   écarter la demande
  FINSI
  call CREER-NOTIFY-SP(param submit,long,primitive)
  envoyer  0  0  idUA  primitive
 FINSI
SINON
 SI NDN suppress n'est pas un param du submit
 ALORS
   SI  le paramètre  notifications de  control dans table 4
n'existe pas
  ALORS
   mettre la suite
   SINON
    écarter la demande
   FINSI
     call CREER-NOTIFY-SN(param submit,non delivery reason,
long, primitive)
   envoyer  0  0  idUA  primitive
  FINSI
 SINON (* pas bon UA *)
   envoyer  1  3  id-UA  local-deliver  param submit  id-UA
orig
SINON  (* c'est un relais *)
  call CREER-USER-PDU(param  submit,recipientinfo,dest,user
PDU)
 envoyer  0  1  dest  ENVOI  user PDU
FINFAIRE
SI nbre UA non reconnus <> 0
ALORS
 SI NDN suppress n'est pas un param du submit
```

```
    ALORS
      SI le paramètre notifications de control dans table 4
    n'existe pas
     ALORS
      mettre la suite
     SINON
      écarter la demande
     FINSI
    call  CREER-NOTIFY-SN(param submit,UA non recognized +
    liste ,long,primitive)
    envoyer 0 0 idUA primitive
   FINSI
 FINSI
```

LOCAL-PROBE-DELIVER

```
call VERIFICATION-REGISTER (bool1, bool2, bool3, bool4, bool5,
DEIT°, MCL°,DCS°,A°,ORAD°,result)
non delivery reason = register
SI result
ALORS
 call CREER-NOTIFY-PP(param probe,long,primitive)
 envoyer 1 3 idUA primitive
SINON
 call CREER-NOTIFY-PN(param probe,non delivery reason, long,
 primitive)
 envoyer 1 3 idUA primitive
FINSI
envoyer 1 5
```

LOCAL-DELIVER

```
call VERIFICATION-REGISTER (bool1, bool2, bool3, bool4, bool5,
DEIT°,MCL°,DCS°,A°,ORAD°,result)
non delivery reason = register
SI result
ALORS
 call CREER-LOCAL-DELIVER(param submit,long,primitive)
 envoyer 0 0 idUA primitive
 SI delivery notice est dans param du submit
 ALORS
  call CREER-NOTIFY-SP(param submit,long,primitive)
  envoyer 1 3 idUA primitive
```

```
 FINSI
SINON
 SI NDN suppress n'est pas un param du submit
 ALORS
  call  CREER-NOTIFY-SN(param submit,non delivery reason, long,
  primitive)
  envoyer 1  3  idUA  primitive
 FINSI
FINSI
envoyer  1  5
```

ANALYSE RAPORT5(numdest,champs APDU)

```
call  CREER-PRIMITIVE-NOTIFY(champs  du  APDU  utilisés,  long,
primitive)
envoyer  1  3  numdest  primitive
```

ANALYSE RAPPORT6(primitive)

```
SI notifications de control sans table 4 n'existent pas
ALORS
 envoyer  0  0  idUA  primitive
SINON
 écarter la demande d'envoi de notify
FINSI
envoyer  1  5
```

DELIVER(APDU)

```
call DECOUPE-MESS-P1
call  VERIFICATION-REGISTER (bool1, bool2, bool3, bool4, bool5,
DEIT°, MCL°,DCS°,A°,ORAD°,result)
reason echec = ' '
SI result
ALORS
 call CREER-DELIVER(champs APDU,long,primitive)
 envoyer  0  0  idUA  primitive
 regarder dans recipientinfo
 SI il faut une notif de livraison
 ALORS
  call  CREER-NOTIFY-APDU(champs  du  APDU  ,long,DRMPDU,raison
  echec)
```

```
  call          RECHERCHE-CHEMIN-COPIE(originator-or-name,nbre-
  direct,(1,dest,info)&,nbre UA inconnus,(nom UA)&)
  SI nbre UA inconnus = 0
  ALORS
   envoyer 0  1  dest  ENVOI  DRMPDU
  FINSI
 FINSI
SINON
 reason echec = register
 call  CREER-NOTIFY-APDU(champs du APDU necessaires, long,
 DRMPDU, busy)
 call   RECHERCHE-CHEMIN-COPIE(originator-or-name,nbre-direct,
 (1,dest,info)&, nbre UA inconnus,(nom UA)&)
 SI nbre UA inconnus = 0
 ALORS
  envoyer 0  1  dest  ENVOI  DRMPDU
 FINSI
 FINSI
envoyer  1  5
```

PROBE-DELIVER(APDU)

```
call DECOUPE-MESS-P1(APDU,bool,type,nbrechamps,(champs)&)
call  VERIFICATION-REGISTER (bool1, bool2, bool3, bool4, bool5,
DEIT°, MCL°,DCS°,A°,ORAD°,result)
SI result
ALORS
 reason echec = ' '
SINON
 reason echec = register
FINSI
call  CREER-NOTIFY-APDU(champs de  l'APDU,long,DRMPDU,raison
echec)
call    RECHERCHE-CHEMIN-COPIE(originator-or-name,nbre-direct,
(1,dest,info)&, nbre UA inconnus,(nom UA)&)
SI nbre UA inconnus = 0
ALORS
 envoyer 0  1  dest  ENVOI  DRMPDU
FINSI
envoyer  1  5
```

LOGOFF

```
envoyer  1  1
goto NON-CONNECTE
```

RELAIS(APDU)

```
call DECOUPE-MESS-P1(APDU,bool,type,nbrechamps,(champs)&)
SI not bool
ALORS
 on jette l'ADPU
 goto NON-CONNECTE
SINON
 call   RECHERCHE-CHEMIN-COPIE(or-name,nbre-direct,  (1,  dest,
 info)&, nbre UA inconnus,(nom UA)&)
 SI type = PROBEPDU ou type user PDU
 ALORS
  créer autant de copies que de nombre de directions
  POUR chaque copie
  FAIRE
   chger le flag dans recipientinfo
   aller écrire des infos dans infotrace => on a un NOUVAPDU
   SI il existe un bouclage
   ALORS
    reason echec = bouclage
    call  CREER-NOTIFY-APDU(champs APDU utilisés, long, DRMPDU,
    raison echec)
    call  RECHERCHE-CHEMIN-COPIE(or-name,nbre-direct, (1, dest,
    info)&, nbre UA inconnus,(nom UA)&)
    envoyer 0  1  dest  ENVOI  DRMPDU
   SINON
    SI 1 = non local
    ALORS
     envoyer  0  1  dest  ENVOI  NOUVAPDU
    SINON  (* 1 = local *)
     SI type = user PDU
     ALORS
      envoyer  1  3  dest  deliver  NOUVAPDU
     SINON  (* c'est un probe PDU *)
      envoyer  1  3  dest  probe deliver  NOUVAPDU
  FINFAIRE
SI nbre UA non connus <> 0
ALORS
```

```
 call   CREER-NOTIFY-APDU(champs  APDU  utilisés,long,DRMPDU,UA
 non connus + liste des UA)
 call   RECHERCHE-CHEMIN-COPIE(or-name,nbre-direct,  (1,  dest,
 info)&,nbre UA inconnus,(nom UA)&)
 envoyer 0  1  dest  ENVOI  DRMPDU
FINSI
SINON  (* c'est un notify *)
 SI nbre dest inconnus = 0
 ALORS
  SI 1 = non local
  ALORS
   aller  écrire  l'information  dans  infotrace  =>  on  a  un
   NOUVAPDU
   SI bouclage
   ALORS
    abandonner ce notifyAPDU
   SINON
    envoyer  0  1  dest  ENVOI  NOUVAPDU
  SINON  (* c'est local *)
   aller voir dans delivery report PDU
   SI report = [0]  (* positif *)
   ALORS
    aller voir dans recipientinfo pour qui est la notif
    call CREER-PRIM-NOTIFY(champs APDU,long,primitive)
    SI bits 2 et 3 = 10 ou 11
    ALORS
     envoyer  0  2  NOTIFYMTA  primitive
    FINSI
    aller voir dans recipient info (* pour UA *)
    SI bits 4 et 5 = 10
    ALORS
     call CREER-PRIM-NOTIFY(champs APDU,long,primitive)
     envoyer  1  3  dest  NOTIFY  primitive
    FINSI
   SINON  (* report = [1], c'est une notif de non livraison *)
    aller voir les bits 2 et 3 dans recipient info
    call CREER-PRIM-NOTIFY(champs APDU,long,primitive)
    envoyer  0  2  NOTIFYMTA  primitive
    aller voir dans recipient info
    SI bits 4 et 5 <> 00
    ALORS
     call CREER-PRIM-NOTIFY(champs APDU,long,primitive)
```

```
     envoyer  1  3  dest  NOTIFY  primitive
    FINSI
  FINSI

ANALYSE RAPPORT1(idUA,APDU,*)

(* ce qui est déclenché par la reception de
     -1  *  demnotnil  APDU
!  * = AM  ou  * = GAM  valeur donnée dans
     -1  *  demnotnil  APDU  *)
call DECOUPE-MESS-P1(APDU,bool,type,nbrechamps,(champs)&)
SI delivery-report-MPDU
ALORS
 envoyer  1  5
SINON  (* c'est un userMPDU ou probe MPDU *)
 POUR boucle sur la suite des recipientsinfo
 FAIRE
  regarder dans perrecipientflag, le responsabilityflag
  SI c'est 1
  ALORS
   conserver la demande maximale
  FINSI
 FINFAIRE
 call          RECHERCHE-CHEMIN-COPIE(originator-or-name,nbre-
 direct,(1,dest,info)&,nbre UA inconnus,(nom UA)&)
 SI nbre UA inconnus = 0
 ALORS
  SI 1 = local
  ALORS
   aller voir dans recipient info le 4ième et 5ième bit
   (* c'est local => rien envoyer comme notify MTA *)
   SI c'est <> 00
   ALORS
    call     CREER-PRIM-NON-LIVRAISON(champs    APDU,     long,
    primitive, raison d'echec)
    (* raison d'echec = busy si * = GAM
                  = pas transférable si * = AM *)
    SI dest = idUA
    ALORS
     envoyer  0  0  idUA  primitive
    SINON  (* c-à-d dest local mais MD pas bien connecté *)
     envoyer  1  3  dest  NOTIFY  primitive
    FINSI
```

```
   SINON  (* pas local *)
    call  CREER-NOTIFY-APDU(champs APDU  utiles, long,  DRMPDU,
    raison non livraison)
    (* raison non livraison = busy si * = GAM
                            = pas transférable si * = AM
      Il en faut tjrs au moins une pour le MTA *)
    envoyer 0  1  dest  ENVOI  DRMPDU
   FINSI
   envoyer  1  5
   return
```

ANALYSE-RAPPORT7(message)

```
call DECOUPE-MESSAGE(message,type,nbrechamps,(champs)&)
CASE type OF
 notify : FAIRE FINFAIRE
 deliver : FAIRE
  boucler sur tous les recipientinfo pour voir le max demandé
  (*  recipientinfo dont  le recipientflag = 1 . C'est tjrs non
  local  car c'est  0  demnotnil  et  pas  -1  demnotnil  =>
  cela a été bloqué au niveau de super MD avec un APDU *)
  call  CREER-NOTIFY-APDU(champs APDU  utilisés, long,  DRMPDU,
  busy)
  call   RECHERCHE-CHEMIN-COPIE(originator-or-name,nbre-direct,
  (1, dest,info)&,nbre UA inconnus,(nom UA)&)
  SI nbre UA inconnus = 0
  ALORS
   envoyer 0  1  dest  ENVOI  DRMPDU
  FINSI
 FINFAIRE
 probe  deliver :  (* idem  DELIVER excepté  que les champs ADU
 peuvent être <> pour créer la notify APDU *)
 relais : FAIRE  (* on a les morceaux de l'APDU découpé *)
  SI c'est un delivery report PDU
  ALORS
  SINON  (* idem deliver *)
 FINFAIRE
ENDCASE
envoyer  1  5
return
```

Annexe 6.

ALGORITHME DU GI-MTA-MTA.

6.1. ALGORITHME.

```
DEBUT :

POUR id-pipe de AM1 à AMder
    ( regarder dans table 18 les machines créées càd bit à 1 )
FAIRE
 POUR première demande jusque dernière demande
 FAIRE
  SI  [A]=0 (  càd ça  indique au  GIMTAMTA que ce qui vient de
  l'AM doit passer par GC )
  ALORS
   supprimer [A]
   écrire  le message  précédé de  1 dans  le pipe  n°2       1
   message
  SINON
   SI [A]=2   ( message pour le RTS )
   ALORS
    écrire le message dans le pipe RTS   2  message
          (  quand le message est précédé de 2, le RTS comprend
    que ça vient de l'AM )
   SINON  ( càd [A]=1, càd demande de source )
    supprimer [A]
    écrire  le message  en fin de file d'attente précédé de id-
    pipe ( =AMi )
   FINSI        AMi   message    (  c'est une demande de source
   venant de l'AM )
  FINSI
 FINFAIRE
FINFAIRE

POUR  id-pipe de  RTS1 à  RTSder (  regarder dans  table 18 les
machines créées càd bit=1 )
FAIRE
 POUR première demande jusque dernière demande
 FAIRE
```

```
SI [A]=0  ( le RTS envoie un message destiné à ISO )
ALORS
 SI bit de réservation=1 dans table 20 d'entrée RTSi
 ALORS
  supprimer [A]
  aller  voir dans  table 20 quel est le n° de connexion dans
  la table d'entrée RTSi
  écrire  le message  dans le  pipe 2  précédé de  0 et n° de
  connexion lu  0  n° connex  message
 SINON
  mettre  en fin  de file  d'attente précédé de RTSi  RTSi  0
  message
SINON
 SI [A]=2  ( c'est pour l'AM )
 ALORS
  écrire le message dans le pipe AMi  2  message
       ( quand message précédé de 2, l'AM sait qu'il vient du
  RTS)
 SINON
  SI [A]=1  ( cad c'est une demande de source )
  ALORS
   écrire le message en fin de file d'attente
```

6.2 PROCEDURES.

RECEPTION-GIUAMTA (AMdest, ENVOI (...))

```
regarder dans table 19 si il existe une entrée AMdest
SI oui
ALORS
 regarder  dans la  table 19  si le  pipe AMi  trouvé n'est pas
 bouché
 SI pas bouché
 ALORS
  envoyer dans le pipe AMi  0  AMdest  ENVOI (...)
 SINON
  remettre  1  AMdest  ENVOI (...)  en fin de file d'attente
```

```
 FINSI
SINON (* il n'y a pas d'association entre ces 2 MTA *)
 regarder  dans table  18 si  il existe  encore une machine càd
 bit = 0
 SI oui
 ALORS
  prendre  cette ligne  et mettre  bit à  1 (*  cette table  18
  comprend les n° de pipes *)
  call  CREER-AM-RTS-PIPES (  pipe AMD,  pipe AMM,  pipe  RTSD,
  pipe RTSM, AMdest, identconnex )
   (* brancher l'AM sur procédure d'open *)
  envoyer dans pipe AMD  0  AMdest  message ENVOI (...)
 SINON
  envoyer  1  -1  GAM  Demnotnl  APDU
 FINSI
FINSI
```

RECEPTION-ISO (MTAvoisin, message)

```
regarder dans table 20 s'il existe une entrée avec RTSdest
SI oui
ALORS
 regarder dans table 20 d'entrée RTSdest le bit de réservation
 SI bit = 1
 ALORS (* le pipe n'est pas bouché dans ce cas *)
  envoyer dans le pipe RTSi  0  MTAvois  message
     (*obtenus ds table 20*)
 SINON (* bit =0 => analyse *)
  SI  pipes bouchés  (* càd  bitpipe =  0 dans  table 20 entrée
  AMdest *)
  ALORS
   mettre  le message  en fin  de file  d'attente  0  MTAvoisin
   message
  SINON
   SI c'est un s-connect-ind  (* c'est un recover *)
   ALORS
    SI numconnex <> 0
    ALORS
     diminuer  numconnex de 1
     envoyer dans le pipe RTSi  0  MTAvoisin  message
     attendre la réponse du s-connect-ind
     SI s-connect-resp est +
     ALORS
```

```
    mettre bit dans table 20 à 1
   FINSI
     envoyer  dans pipe 2  0  MTAvoisin  message = s-connect-
  resp +ou-
   SINON
    failure reason := RTS busy
     envoyer  dans pipe 2  0  MTAvoisin  message = s-connect-
  resp -
   FINSI
  SINON
   SI c'est  un s-connect-confirm
   ALORS
    SI s-connect-confirm est +
    ALORS
     mettre dans table 20 le bit de réservation à 1
    FINSI
     envoyer dans pipe RTSi  0  MTAvoisin  message
    SINON (* erreur *)
     écarter la demande
    FINSI
   FINSI
  FINSI
 FINSI
SINON
 SI  message =  s-connect-ind (*  c'est un  open car il n'y a
 rien dans table 20 *)
 ALORS
  regarder  dans la  table 18  s'il existe encore une machine
  libre càd bit=0
  SI oui
  ALORS
   SI numconnex <> 0
   ALORS
     mettre bit à 1 dans table 18 (* cette table comprend les
  n° de pipes *)
    diminuer numconnex de 1
     call CREER-AM-RTS-PIPES ( pipe AMD, pipe AMM, pipe RTSD,
  pipe RTSM, MTAvoisin, identconnex )
    envoyer dans le pipe RTSD  0  MTAvoisin  message
    attendre la réponse du s-connect-ind
    SI s-connect-resp est +
    ALORS
```

```
        mettre dans table 20 le bitde réservation à 1
       SINON
        mettre bit à 0 dans table 18 (* càd détruire machine *)
          envoyer l'ordre  de destruction  dans les pipes AMD et
     RTSD
       FINSI
      envoyer dans pipe 2  0  MTAvoisin  message
      SINON
       failure reason := busy
       créer le s-connect-resp-
       envoyer dans pipe 2  0  MTAvoisin  s-connect-resp -
      FINSI
     SINON
      failure reason = busy
      créer le s-connect-resp-
      envoyer dans pipe 2  0  MTAvoisin  s-connect-resp -
     FINSI
    SINON (* erreur *)
     rejeter la demande
    FINSI
   FINSI
FINSI
```

Annexe 7.

ALGORITHME DE L'AM

7.1. ALGORITHME.

IDLE :

```
vider pipe
tourdemande := false
placer en fin de file d'attente
SI file est vide
ALORS
 goto IDLE
SINON
 prendre la première demande
 CASE   1   MTAvoisin   (* ce message est tjs le 1° reçu car le
 gestion. est bien conçu *)
  mtavois := MTA voisin
  goto IDLE
 CASE  0  AMdest  envoi  APDU
  premdem := 'ENVOI'
  goto ETABLISS-ASSOC
 CASE  2  open-ind
  premdem := 'OPEN-IND'
  goto ETABLISS-ASSOC
 ENDCASE
FINSI
```

ETABLISS-ASSOC :

```
SI premdem = 'envoi'
ALORS
 call CREER-OPEN-REQ ( mtavois, primitive )
 envoyer  2  primitive
 turn := initial turn
 goto ATTENTE-ASSOC
SINON ( premdem = OPEN-IND *)
 call VERIF-PARAM-OPEN ( open-ind, reason )
 call CREER-OPEN-RESP ( opren-ind, reason, primitive )
```

```
envoyer  2  primitve
turn := initial turn
SI reason = ' '
ALORS
 SI initial-turn = initiator
 ALORS
  goto ASSOC-TOUR
 SINON
  goto ASSOC-SS-TOUR
 FINSI
SINON
 goto FIN-TRAVAIL
FINSI
FINSI
```

ATTENTE-ASSOC :

```
vider pipe
placer en fin de file d'attente
SI file vide
ALORS
 goto ATTENTE-ASSOC
SINON
 lire  dans la file jusqu'à avoir une demande commençant par  2
 et si on ne trouve pas : revider pipe
 SI ce n'est pas  2  open-conf
 ALORS
  écarter la demande et continuer la recherche
 SINON
  call DECOUPE-PRIM ( open-conf, nbrechps, (n°chp,val)* )
  SI disposition = Accepted
  ALORS
   SI turn = initiator
   ALORS
    goto ASSOC-TOUR
   SINON
    goto ASSOC-SS-TOUR
   FINSI
  SINON
   goto FIN-TRAVAIL
  FINSI
 FINSI
```

```
FINSI
```

ASSOC-TOUR :

```
tourattendu := false
SI premdem = 'ENVOI'
ALORS
 call CREER-TRANSF-REQ ( APDU, primitive )
 envoyer  2 primitive
FINSI
premdem := nul

file :

vider pipe
placer en fin de file d'attente
SI file pas vide
ALORS
 prendre la première demqnde
 CASE  0  AMdest ENVOI
  call CREER-TRANSF-REQ ( APDU, primitive )
  envoyer  2  primitive
  goto file
 CASE  2  turn-please-ind
  call DEM-TOUR ( turn-please-ind, turn, état )
 CASE état OF
  AST : goto ASSOC-SS-TOUR
  AT : goto file
 ENDCASE
 CASE  2  exception-ind
  call EXCEPTION ( exception-ind )
  goto file
 ENDCASE
SINON
 SI tourdemandé
 ALORS
  call  CREER-TURN-GIVE-REQ (  primitive )  (* on décide de
  donner  le tour  dès qu'on  reçoit la demande ou quand on n'a
  plus  rien à faire.  Mais on aurait pu vérifier la priorité à
  chaque nouvelle demande à traiter *)
  envoyer  2  primitive
  goto ASSOC-SS-TOUR
 SINON
```

```
  call CREER-CLOSE-REQ ( primitive )
  envoyer  2 primitive
  goto ATTENTE-FIN-ASSOC
 FINSI
FINSI
```

ASSOC-SS-TOUR :

```
tourdemandé := false
SI premdem = 'ENVOI'
ALORS
 remettre la demande contenue dans premdem dans la file
 d'attente  ( en début )
FINSI
premdem := nul
vider pipe
mettre dans la file d'attente
prendre la première demqnde
CASE  0  AMdest ENVOI
 SI tourattendu
 ALORS
  remettre la demande à sa place dans le file
  vider pipe
  mettre en fin de file d'attente
  passer à la demande suivant l'envoi
  SI demande existe
  ALORS
   goto CASE
  SINON
   goto fin
  FINSI
 SINON
  call TOUR
  vider pipe
  mettre en fin de file d'attente
  passer à la demande suivant l'envoi
  remettre la demande à sa place dans la file
  SI demande existe
  ALORS
   goto CASE
  SINON
   goto fin
```

```
  FINSI
 FINSI
CASE  2  transfer-ind
 envoyer  0  NULL  RELAIS  APDU
 vider pipe
 mettre en fin de file d'attente
 passer à la demande suivante
 SI demande existe
 ALORS
  goto CASE
 SINON
  goto fin
 FINSI
CASE  2  exception-ind
 envoyer  0  -1  AM  Demnotnl  APDU
 vider pipe
 mettre en fin de file d'attente
 passer à la demande suivante
 SI demande existe
 ALORS
  goto CASE
 SINON
  goto fin
 FINSI
CASE  2  turn-give-ind
 goto ASSOC-TOUR
CASE  2  close-ind
 call CREER-CLOSE-RESP ( primitive )
 envoyer  2 primitive
 goto FIN-TRAVAIL
ENDCASE

fin  :  (* on est passé au-dessus de bcp de damandes ss
recevoir  le tour.  Il reste des envois à faire => on attend et
on  redemande le  tour.  Soit il n'y a plus rien à faire, alors
on redemande le tour pour faire un CLOSE *)

après un certain temps faire
call TOUR
vider pipe
mettre en fin de file d'attente
passer à la demande suivante
goto CASE
```

ATTENTE-FIN-ASSOC :

```
vider pipe
placer demande en fin de file d'attente
prendre première demande
CASE  2  exception-ind
 mettre   0   -1   AM  Demnotnl  APDU  dans la file d'attente à
 la place de l'exception
 vider pipe
 mettre en fin de file d'attente
 passer à la demande suivante
 goto CASE
CASE  2  close-conf
 goto FIN-TRAVAIL
ELSECASE  (* cas du ENVOI *)
 vider pipe
 mettre en fin de file d'attente
 passer à la demande suivante
 goto CASE
```

FIN-TRAVAIL :

```
vider le pipe AMi
mettre les demandes en fin de file d'attente
prendre la première demande si elle existe
SI c'est  1  pipe bouché
ALORS
 envoyer  1  demande destruction
 goto ATTENTE-PERMISS-DESTRUCT
SINON
 faire précéder la demande de  1  4
 mettre dans le pipe AMi
 goto FIN-TRAVAIL
FINSI
SI existe pas de demande dans file
ALORS
 goto FIN-TRAVAIL
FINSI
```

ATTENTE-PERMISS-DESTRUCT :

vider le pipe jusqu'à trouver 1 destruction (* on ne peut trouver que cela *)
goto IDLE

7.2. PROCEDURES

CREER-OPEN-REQ (mtavois, primitive)

VERIF-PARAM-OPEN (open-ind, reason)

reason := ' ' => la vérif est OK

CREER-OPEN-RESP (open-ind, reason, primitive)

CREER-TRANSFER-REQ (APDU, primitive)

DEM-TOUR (turn-please-ind, turn, état)

soit il le donne
call CREER-TURN-GIVE-REQ (primitive)
envoyer 2 primitive
turn := responder
état := AST

soit il ne le donne pas
tourdemandé = true
état := AT

EXCEPTION (exception-ind)

envoyer 0 -1 AM Demnotnl APDU

TOUR

tourattendu := true
call CREER-TURN-PLEASE-REQ (primitive)

envoyer 2 primitive

CREER-CLOSE-RESP (primitive)

CREER-TURN-GIVE-REQ (primitive)

CREER-TURN-PLEASE-REQ (primitive)

Annexe 8

ALGORITHME DU RTS

8.1. ALGORITHME.

IDLE :

```
deridactsec := null
deridconnexsec := null
idactcourrec := null
idactintrec := null
idactcourenv := null
idactintenv := null
deridconnexint := null
deridconnexcour := null
origine  := null  (* c-à-d  pas d'activité  interrompue pour le
moment *)
demsuivante
On reçoit  1  MTA voisin
demsuivante
On reçoit  1  dernier n° généré
demsuivante
Mémoriser  le dernier numéro généré (DERNUMGE) et le MTA voisin
(MTAVOIS)
CASE on reçoit  2  open-req
 call ESSAI-OUVERT-ASSOC
 goto ATTENTE-DE-CONNEXION
CASE on reçoit  0  S-connect-ind
 call   ESSAI-RECEPTION-ASSOC  (*  on  est  en  idle  =>  c'est
 forcément un open qui a provoqué la S-connect *)
 CASE état OF
  AD : goto ATTENTE-DESTRUCTION
  CJ : goto CONNECTE-AVEC-JETONS
  CSJ : goto CONNECTE-SS-JETON
 ENDCASE
ENDCASE
```

ATTENTE-DE-CONNEXION :

```
demsuivante
CASE on reçoit  0  S-connect-conf
 call CONFIRM-ASSOC
 CASE étaf OF
  CJ : goto CONNECTE-AVEC-JETONS
  CSJ : goto CONNECTE-SS-JETON
  EA : goto ECHEC-ASSOCIATION
 ENDCASE
CASE on reçoit  1  S-refuse-connexion-ind
 call REFUSE-CONNEXION
 goto ECHEC-ASSOCIATION
ENDCASE
```

ECHEC-ASSOCIATION :

```
demsuivante
envoyer  3  boucher pipe
goto ATTENTE-PIPE-BOUCHE
```

ATTENTE-PIPE-BOUCHE :

```
BOUCLE sur demsuivante
 SI demande =  1  pipe bouché
 ALORS
  envoyer  1  demande destruction  dernumge
  goto ATTENTE-DESTRUCTION
 SINON
  envoyer  1  4  demande
 FINSI
FINBOUCLE
```

ATTENTE-DESTRUCTION :

```
demsuivante  (* c'est  1  destruction  *)
goto IDLE
```

CONNECTE-AVEC-JETONS :

```
AMtourdemandé := false
SI AMtourdonné
ALORS
 call CREER-S-CONTROL-GIVE-REQ(primitive)
 envoyer  0  mtavois  primitive
 AMtourdonné := false
 goto CONNECTE-SS-JETON
 donnéesreçues := false
FINSI
prioritétour := -1
jeton := true
lire dans le pipe
SI demande existe
ALORS
 mettre en fin de file d'attente
 prendre la 1ère demande
 CASE on reçoit  0  S-taken-please-ind
  call DEMANDE-JETONS
 CASE on reçoit  2  turn-give-req
  call DON-TOUR
 CASE on reçoit  2  transfer-req
  état := ES
  demandeenvoi := transfer-req
 CASE on reçoit  2  close-req
  call DEMANDE-FERM-ASSOC
 ENDCASE
 CASE état OF
  CJ : goto CONNECTE-AVEC-JETONS
  CSJ : goto CONNECTE-SS-JETON
  ES : goto ENVOI-SESSION
  AF : goto ATTENTE-FERMETURE
 ENDCASE
SINON
 SI  origine  =  envoyeur      (*  car  seul  envoyeur  reprend
 l'activité interrompue *)
 ALORS
  demandeenvoi := 'resume'
  goto ENVOI-SESSION
```

```
 SINON
  goto CONNECTE-AVEC-JETONS
 FINSI
FINSI
```

CONNECTE-SS-JETON :

```
AMtourdonné := false
SI AMtourdemandé
ALORS
 call CREER-S-TOKEN-PLEASE-REQ(param,prioritétour,primitive)
 envoyer  0  mtavois  primitive
 AMtourdemandé := false
 prioritétour := -1
FINSI
jeton := false
demsuivante
CASE on reçoit  0  S-activity-resume
 demanderecept := resume-ind
 goto RECEPTION-SESSION
CASE on reçoit  2  turn-please-req
 call DEMANDE-TOUR
CASE on reçoit  0  S-control-give-ind
 call RECEPTION-JETONS
CASE on reçoit  0  S-release-ind
 call RECEPTION-FERMETURE
CASE on reçoit  0  S-activity-start-ind
 demanderecept := S-activity-start
 goto RECEPTION-SESSION
ENDCASE
CASE état OF
 CJ : goto CONNECTE-AVEC-JETONS
 CSJ : goto CONNECTE-SS-JETON
 AF : goto ATTENTE-FERMETURE
ENDCASE
```

ATTENTE-FERMETURE :

```
demsuivante
SI jetons
ALORS
 on reçoit  0  S-release-conf
 call CONFIRM-FERMETURE
SINON
 on reçoit  2  close-req
 call REPONSE-FERMETURE
FINSI
envoyer  3  boucher pipe
goto ATTENTE-PIPE-BOUCHE
```

FIN-TRAVAIL-PAS-JETONS

```
lire dans le pipe
SI demande existe
ALORS
 mettre en fin de file d'attente
 prendre la 1ère demande
 CASE on reçoit  2  turn-please-req
  call DEMANDE-TOUR-RECOVER
 CASE on reçoit  0  S-connect-ind
  call ESSAI-CONNEXION-RECOVER-SJ
 ENDCASE
 CASE état OF
  FTSJ : goto FIN-TRAVAIL-PAS-JETON
  CJ : goto CONNECTE-AVEC-JETONS
  CSJ : goto CONNECTE-SS-JETON
 ENDCASE
SINON
 (* faut-il essayer de faire un recover en espérant se
 retrouver dans un état connecté avec jetons pour résumer une
 activité qui aété interrompue de son côté? Ceci en prenant
 le risque de se retrouver dans état connecté sana jeton et en
 étant pas sûr que l'on demandera le tour,... *)
FINSI
```

FIN-TRAVAIL-AVEC-JETONS :

```
SI finassoc
 (* AM a fait close => ne peut plus rien demander
ALORS
 lire dans le pipe
 SI demande existe
 ALORS
  mettre en fin de file d'attente
  prendre la 1ère demande
   (*  Ce ne  peut venir  que d'ISO.  Théoriquement, ce ne peut
   être qu'un S-connect-ind *)
  call DECOUPE-PRIM(S-connect-ind, nbrechps,(n°chp,val)&)
  call     VERIFICATION     -     PARAM     -     COMPATIBILITE
  (deridconnexint,reason,état,nbrechps,(n°chp,val)')
  call     CREER-    S    -CONNECT    -    RESP    -    RECOVER
  (nbrechps,(n°chp,val)&,reason,primitive)
  SI reason = ' '
  ALORS
   call CREER-S-RELEASE-REQ(primitive)
   envoyer  0  mtavois  primitive
   goto ATTENTE-FERMETURE
  SINON
   goto FIN-TRAVAIL-AVEC-JETONS
  FINSI
  lire dans le pipe
  SI demande existe
  ALORS
   mettre en fin de file d'attente
   prendre la 1ère demande
   CASE on reçoit  2  transfert-req
    call DEMAN-ENVOI-RECOVER
   CASE on reçoit  2  turn-give-req
    call DON-TOUR-RECOVER
   CASE on reçoit  2  turn-please-req
    call DEMAN-TOUR-RECOVER
   CASE on reçoit  2  close-req
    call DEMAN-FERMET-RECOVER
   CASE on reçoit  2  S-connect-ind
    call ESSAI-CONNEXION-RECOVER-AJ
   ENDCASE
```

```
    CASE état OF
     CJ : goto CONNECTE-AVEC-JETONS
     FTJ : goto FIN-TRAVAIL-AVEC-JETONS
     ES : goto ENVOI-SESSION
     AF : goto ATTENTE-FERMETURE
    ENDCASE
   SINON
    SI origine = envoyeur    (* C-à-d qu'il n'existe aucune
    demande venant de l'extérieur mais il sait qu'il existe un
    message pour lequel il était envoyeur qui a été interrompu
    => il fera un recover puis resume pour le récupérer *)
    ALORS
     call RECUP-RECOVER
    SINON
     goto FIN-TRAVAIL-AVEC-JETONS
    FINSI
    CASE état OF
     FTJ : goto FIN-TRAVAIL-AVEC-JETONS
     ES : goto ENVOI-SESSION
    ENDCASE
   FINSI
  FINSI
FINSI
```

ENVOI-SESSION :

```
call ENVOI
CASE état OF
 CJ : goto CONNECTE-AVEC-JETONS
 FTJ : goto FIN-TRAVAIL-AVEC-JETONS
ENDCASE
```

RECEPTION-SESSION

```
call RECEPTION
CASE état OF
 CSJ : goto CONNECTE-SS-JETONS
 FTSJ : goto FIN-TRAVAIL-SS-JETONS
ENDCASE
```

8.2. PROCEDURES.

DECOUPE-PRIM(primitive,nbrechamps,(n°champs,valeur)&)

Cette procédure obtient en entrée une primitive qcq reçue par le RTS (venant de l'AM ou de ISO) et découpe cette primitive en donnant les champs et leur valeur.

ESSAI-OUVERT-ASSOC(open-req,mtavois,jeton,dernumge)

```
call DECOUPE-PRIM(open-req,nbrechamps,(n°champs,valeur)&)
SI 3ième champs = INITIATOR
ALORS
 JETON := true
SINON
 JETON := false
FINSI
callCREER-S-CONNECT-REQ(nbrechamps,
(n°champs,valeur)&,mtavois,dernumge, primitive)
```

ESSAI-RECEPTION-ASSOC(S-connect-ind, mtavois, état, idactcourenv,idactcourrec,deridconnexcour,finassoc)

```
call DECOUPE-PRIM(S-connect-ind,nbrechamps,(n°champs,valeur)&)
SI dans le 11ième chp on trouve recover
ALORS
 refuse reason = cannotrecover
 call              CREER-S-CONNECT-RESP              (nbrechamps,
 (n°champs,valeur)&,refuse reason, primitive)
 envoyer  0  mtavois  primitive
 état := AD
SINON
 call VERIF-PARAM(nbrchamps,valchps,bool)
 SI not bool
 ALORS
  refuse reason = validation failure
  call   CREER-S-CONNECT-REQ  (nbrechamps,  (n°champs,valeur)&,
```

```
 mtavois, refuse reason, primitive)
 envoyer  0  mtavois  primitive
 état := AD
SINON
 (* Les paramètres sont bons *)
 call CREER-OPEN--IND(chps necessaires,primitive)
 envoyer  2  open-ind
 demsuivante  (* c'est un open-resp)
 call DECOUPE-PRIM(open-resp,nbrechamps,(n°champs,valeur)&)
 SI disposition = 'accepted'
 ALORS
  failure reason =' '
 SINON
  failure reason = refusal reason
 FINSI
 call  CREER-S-CONNECT-RESP  (nbrechamps,  (n°champs,valeur)&,
 failure reason, primitive)
 envoyer  0  mtavois  primitive
 SI failure reason = ' '
 ALORS
  SI jeton
  ALORS
   état := CJ
  SINON
   état := CSJ
  FINSI
  finassoc := false
  idactcourenv := idactcourrec := 0
  deridconnexcour  := 1er  param de  S-connect-ind (*  session
  connexion identifier *)
 SINON
  état := AD
 FINSI
FINSI
FINSI
```

CONFIRM-ASSOC(S-connect-conf, jeton, état, finassoc, deridconnexcour)

call DECOUPE-PRIM(S-connect-conf, nbrechamps, (n°champs, valeur)&)

SI 4ième champs = accept

```
ALORS
 SI jeton
 ALORS
  état := CJ
 SINON
  état := CSJ
 FINSI
 finassoc := false
 deridconnexcour := 1er param de S-connect-conf (* session
 connexion identifier *)
SINON
 état := EA
FINSI
call CREER-OPEN-CONF(param du S-connect-conf,primitive)
envoyer 2 primitive
```

REFUSE-CONNEXION(S-refuse-connexion-ind,état)

```
disposition := refuse
refusal reason := busy
call CREER-OPEN-CONF(param ci-dessus,primitive)
envoyer 2 primitive
état := EA
```

DEMANDE-JETONS(S-token-please-ind,état)

```
call DECOUPE-PRIM(S-token-please-ind, nbrechamps, (n°champs,
valeur)&)
call CREER-TURN-PLEASE-IND(paramètres nécessaires,primitive)
envoyer 2 primitive
état := CJ
```

DON-TOUR(turn-give-req,mtavois,état,données reçues)

```
call CREER-S-CONTROL-GIVE-REQ(primitive)
envoyer 0 mtavois primitive
état := CSJ
donnéesreçues := false
```

DEMANDE-FERM-ASSOC(close-req,mtavois,état,finassoc)

call CREER-S-RELEASE-REQ(primitive)
envoyer 0 mtavois primitive
état := AF
finassoc := true

DEMANDE-TOUR(turn-please-req, mtavois, AMtourdemandé, prioritétour, état)

call DECOUPE-PRIM(turn-please-req, nbrechamps, (n°champs, valeur)&)
call CREER-S-TOKEN-PLEASE-REQ(paramètres nécessaires, primitive)
envoyer 0 mtavois primitive
état := CSJ
AMtourdemandé := true
prioritétour := param se trouvant dans turn-please-req

RECEPTION-JETON(S-control-give-ind,état)

call CREER-TURN-GIVE-IND(primitive)
envoyer 2 primitive
état := CJ

RECEPTION-FERMETURE(S-release-ind,état)

call DECOUPE-PRIM(S-release-ind,nbrechamps,(n°champs,valeur)&)
call CREER-CLOSE-IND(paramètres nécessaires du S-release, primitive)
envoyer 2 primitive
état := AF

CONFIRM-FERMETURE(S-release-conf)

call DECOUPE-PRIM(S-release-conf, nbrechamps, (n°champs, valeur)&)
call CREER-CLOSE-CONF(paramètres nécessaires du S-release, primitive)
envoyer 2 primitive

REPONSE-FERMETURE(close-resp,mtavois)

```
call CREER-S-RELEASE-RESP(primitive)
envoyer  0  mtavois  primitive
```

DEMANDE-TOUR-RECOVER(turn-please-req, mtavois, dernumge, donnéesreçues,deridconnexint,AMtourdemandé,prioritétour,jetons,état)

```
call   CREER-RECOVER-REQ  (dernumge,  deridconnexint,  mtavois,
donnéesreçues, jetons, primitive)
envoyer  0  mtavois  primitive
SI c'est  1  refuse-connex-ind
ALORS
 état := FTSJ
SINON  (* on reçoit un  0  S-connect-confirm *)
 SI result = reject
 ALORS
  état := FTSJ
 SINON
  regarder dans initial-token-assignment
  SI receiving
  ALORS
   état := CSJ
  SINON  (* c'est sending *)
   état := CJ
  FINSI
 FINSI
FINSI
AMtourdemandé := true
prioritétour := param se trouvant dans turn-please-req
```

ESSAI-CONNEXION-RECOVER-SJ(S-connect-ind, deridconnexint, idactcourenv,idactcourrec,état,mtavois)

```
call DECOUPE-PRIM(S-connect-ind,nbrechamps,(n°champs,valeur)&)
CALL  VERIFICATION-COMPATIBILITE (deridconnexint, idactcourenv,
idactcourrec,état,mtavois)
call   CREER-S-CONNECT-RESP-RECOVER   (nbrechamps,   (n°champs,
valeur)&, reason, primitive)
envoyer  0  mtavois  primitive
SI reason = ' '
```

```
ALORS
 idactcourenv := idactcourec := 0
FINSI
```

DEMAN-ENVOI-RECOVER(transfert-req, dernumge, deridconnexint, mtavois,donnéesreçues,demandeenvoi,jetons,état)

```
call DECOUPE-PRIM(transfert-req,nbrechamps,(n°champs,valeur)&)
demandeenvoi := transfert-req
call  CREER-RECOVER-REQ (dernumge, deridconnexint, mtavois,
donnéesreçues,jetons,primitive)
 (* il a les jetons => après le recover il les garde *)
envoyer 0 mtavois primitive
lire dans le pipe
mettre en fin de file d'attente
prendre la 1ère demande qui commence par 0 ou 1 si elle existe
si cette demande n'esiste pas => retourner lire dans le pipe
SI on reçoit 0 S-refuse-connex-ind
ALORS
 on envoie 2 exception-ind APDU
 état := FTJ
SINON
 SI on reçoit 0 S-connect-conf
 ALORS
  call  DECOUPE-PRIM(S-connect-conf, nbrechamps, (n°champs,
  valeur)&)
  SI result = reject
  ALORS
   on envoie 2 exception-ind ADPU
   état := FTJ
  SINON (* le S-connect est + *)
   état := ES
   demandeenvoi := transfert-req
  FINSI
 SINON
  (* rien du tout car pour recevoir un S-P-abort, il faut une
  connexion *)
 FINSI
FINSI
```

DON-TOUR-RECOVER(turn-give-req, AMtourdonné, dernumge, deridconnexint, mtavois,donnéesreçues,jetons,état)

```
AMtourdonné := true
call  CREER-RECOVER-REQ (dernumge, deridconnexint, mtavois,
donnéesreçues,jetons,primitive)
envoyer 0 mtavois primitive
SI on reçoit 1 S-refuse-connex-ind
ALORS
 état := FTJ
SINON
 (* on reçoit 0 S-connect-conf *)
 SI result = reject
 ALORS
  état := FTJ
 SINON
  état := CJ
 FINSI
FINSI
```

DEMAN-TOUR-RECOVER(turn-please-req,AMtourdonné,état)

```
envoyer 2 turngive-ind
AMtourdonné := false
état := FTJ
```

DEMAN-FERMET-RECOVER(close-req, mtavois, dernumge, deridconnexint, donnéesreçues,jetons,état,finassoc)

```
finassoc := true
call  CREER-RECOVER-REQ (dernumge, deridconnexint, mtavois,
donnéesreçues,jetons,primitive)
envoyer 0 mtavois primitive
SI on reçoit 1 S-refuse-connex-ind
ALORS
 état := FTJ
SINON
 (* on reçoit 0 S-connect-conf *)
 SI result = reject
 ALORS
  état := FTJ
```

```
 SINON
  call CREER-S-RELEASE-REQ(primitive)
  envoyer  0  mtavois  primitive
  état := AF
 FINSI
FINSI
```

**ESSAI-CONNEXION-RECOVER-AJ(S-connect-ind,
deridconnexint,
idactcourenv,idactcourrec,état,mtavois)**

```
call DECOUPE-PRIM(S-connect-ind,nbrechamps,(n°champs,valeur)&)
CALL     VERIFIC-PARAM-COMPATIB-AVECJETONS     (deridconnexint,
idactcourenv, idactcourrec,état,mtavois)
call   CREER-S-CONNECT-RESP-RECOVER   (nbrechamps,   (n°champs,
valeur)&, reason, primitive)
envoyer  0  mtavois  primitive
SI reason = ' '
ALORS
 idactcourenv := idactcourec := 0
FINSI
```

**RECUP-RECOVER (dernumge, deridconnexint, mtavois,
donnéesreçues, demandeenvoi,jetons,état)**

```
call   CREER-RECOVER-REQ  (dernumge,  deridconnexint,  mtavois,
donnéesreçues, jetons,primitive)
envoyer  0  mtavois  primitive
SI on reçoit  1  S-refuse-connex-ind
ALORS
 état := FTJ
SINON
 (* on reçoit  0  S-connect-conf *)
 SI result = reject
 ALORS
  état := FTJ
 SINON
  état := ES
  demandeenvoi := 'resume'
 FINSI
FINSI
```

ENVOI(demandeenvoi,AMtourdonné,origine,transfertime,idactcourenv,checkpointsize,windowsize,mtavois,état,jetons)

```
SI demandeenvoi = transfert-req
ALORS
 SI transfertime écoulé
 ALORS
  temps écoulé
  goto fin
 SINON
  call CREER-ACTIV-START(idactcourenv,primitive)
  envoyer  0  mtavois  primitive
  call    DECOUPE-ADPU-EN-SSDU    (ADPU,    nbressdu,    (ssdu)&,
  checkpointsize)
  taillebuffer := urndorsize + 1
  pteurdebut := 1
  pteurfin := 0
  BOUCLE pour i = 1 jusque(nbressdu-1)
  TANT QUE (pteurfin + 2) mod taillebuffer <> pteurdebut
  FAIRE
   SI transfertime écoulé
   ALORS
    temps écoulé
    goto fin
   SINON
    call CREER-S-DATA-REQpteurfin,SSDU n°i,primitive)
    envoyer  0  mtavois  primitive
    call CREER-SYNCHRO-REQ(pteurfin,taillebuffer,primitive)
    envoyer  0  mtavois  primitive
    vider le pipe
    mettre en fin de file d'attente
    POUR  1ère demande  jusqu'à la  dernière  [(venant  du  RTS
    voisin) ou (étant  2  turn-give-req  )]
    FAIRE
     CASE  0    S-synch-min-conf
     call GESTION-SYNCHRO(CS,FTJ)
     IF état = CI ou FTJ
     THEN
      goto fin
      CASE  0  S-token-please-ind
      call GESTION-JETONS
```

```
    CASE  2  turn-give-req
    call GESTION-TOUR
    IF état = CS
    THEN
     goto fin
     CASE  0  S-U-abort-ind
      call GESTION-ABORT
      goto fin
     CASE  0  S-P-abort-ind
      call GESTION-ABORT
      goto fin
     CASE  0  S-U-exception-report-ind
      call GESTION-EXCEPTION
      goto fin
     ENDCASE
  ENDFAIRE
FINFAIRE
call CREER-S-DATA-REQ(pteurfin,dernier SSDU,primitive)
envoyer  0  mtavois  primitive
call CREER-S-ACTIVITY-END(primitive)
envoyer  0  mtavois  primitive
TANT QUE pteurdebut <> pteurfin
FAIRE
 vider le pipe
 mettre en fin de file d'attente
 POUR  1ère demande  jusqu'à  la  dernière  [(venant  du  RTS
 voisin) ou (étant  2  turn-give-req  )]
 FAIRE
  CASE  0  S-synch-min-conf
   call GESTION -SYNCHRO(CS,FTJ)
   IF état = CI ou FTJ
   THEN
    goto fin
  CASE  0  S-token-please-ind
   call GESTION-JETONS
  CASE  2  turn-give-req
   call GESTION-TOUR
   IF état = CS
   THEN
    goto fin
```

```
   CASE  O  S-U-abort-ind
    call GESTION-ABORT
    goto fin
   CASE  O  S-P-abort-ind
    call GESTION-ABORT
    goto fin
   CASE  O  S-U-exception-report-ind
    call GESTION-EXCEPTION
    goto fin
   ENDCASE
  ENDFAIRE
 FINFAIRE
 call CREER-S-ACTIVITY-END(primitive)
 envoyer  O  mtavois  primitive
 vider le pipe
 mettre  en fin  de file  d'attente jusqu'à  recevoir une  des
 quatre  demandes  suivantes  (* ce qu'il peut recevoir
 d'autre.  Ce  sont le  S-token-please-ind  et  des  messages
 venant de l'AM *)
 on peut recevoir
 CASE  O  S-act-end-conf
  call EFFACER-APDU
  état := CJ
  goto fin
 CASE  O  S-U-abort-ind
  call GESTION-ABORT
  goto fin
 CASE  O  S-P-abort-ind
  call GESTION-ABORT
  goto fin
 CASE  O  S-U-exception-report-ind
  call GESTION-EXCEPTION
  goto fin
 CASE  O  S-token-please-ind
  call GESTION-JETONS
 CASE  2  turn-give-req
  AMtourdonné := true
   (* c-à-d on veut finir avant de donner des jetons *)
 ENDCASE
SINON
 (* c'est un resume *)
 origine := nul
```

```
SI transfertime écoulé
ALORS
 temps écoulé
 goto fin
SINON
 call     CREER-RESUME    (idactcourenv,    dercheckpointint,
 messageint, pointcurreprise,mtavois)
 restedaptu := APDU commençant à pointeurreprise
 call  DECOUPE-ADPU-EN-SSDU (restedapdu,  nbressdu,  (SSDU)&,
 checkpointsize)
 goto fin
FINSI
fin : return
```

RECEPTION(demanderecept,windowsize,jetons,mtavois,état)

```
déjàreçu := false
SI demanderecept := param activity identifier de demanderecept
dercheckpoint := 0
donnéesreçues := true
donnéesenattente := 0
taillebuffer := windowsize + 1
état := recept
messagecour := ' '
(*  rem : on lit dans le pipe puis dans la file d'attente => on
ne  lit qu'une  seule demande  à la  fois  => plus  besoin  de
fenêtre *)
file:
vider le pipe
mettre en file d'attente
SI la file d'attente est vide
ALORS
 goto file
SINON
 prendre la 1ère demande
 CASE  0  S-data-ind
  call GESTION-DONNEES(donnéesenattente,ssdu,état)
  IF état = recept
  THEN
   goto file
 CASE  0  S-synch-min-ind
  call GESTION-PTSYNCHRO
```

```
IF état = recept
THEN
 goto file
 CASE  0  S-activity-end-ind
  call GESTION-FIN
 CASE  0  S-U-abort-ind
  call GESTION-ABORT
 CASE  0  S-P-abort-ind
  call GESTION-ABORT
 CASE  0  S-activity-interrupt-ind
  call GESTION-INTERRUPT
  call CREER-S-ACT-INTER-RESP
 CASE  0  S-activity-discard-ind
  call EFFACER-APDU(messagecour)
  call CREER-S-ACT-DISC-RESP
 CASE  2  turn-please-req
  AMtourdemandé := true
  call CREER-S-TOKEN-PLEASE-REQ
  goto file
 ENDCASE
FINSI
SINON  (* c'est un S-act-resume *)
 call DECOUPE-PRIM
 SI  (2ième paramètre  (c-à-d oldactidentifier) = deridactsee)
 et (4ième param (c-à-d oldconnexident) = deridconnexsee)
 ALORS
  déjàreçu := true
 FINSI
 SI  (2ième paramètre  (c-à-d oldactidentifier) = idactintree)
 et  (3ième param (synchroptserialnber) = dercheckpointint) et
 (4ième param (c-à-d oldconnexident) = deridconnexint)
 ALORS
  idactcourec := activity identifier
  dercheckpoint := 0
  donnéesreçues := true
  donnéesenattente := 0
  taillebuffer := windowsize + 1
  état := recept
  messagecour := messageint
  goto file
```

```
  SINON
   call CREER-ABORT
  FINSI
 FINSI
FINSI
```

CREER-S-CONNECT-REQ (nmbrechamps, (n°champ,valeur)*, mtavois, dernumge, idactcourenv, idactcourrec, primitive)

CREER-S-CONNECT-RESP (nbrechp, (n°chp,valeur)*, refuse reason, primitive)

CREER-OPEN-IND (champs nécessaires, primitive)

CREER-OPEN-CONF (param nécess, primitive)

CREER-TURN-PLEASE-IND (param nécess, primitive)

CREER-S-CONTROL-GIVE-REQ (primitive)

CREER-S-RELEASE-REQ (primitive)

CREER-S-TOKEN-PLEASE-REQ (param nécess, primitive)

CREER-TURN-GIVE-IND (primitive)

CREER-CLOSE-IND (param nécess, primitive)

CREER-CLOSE-CONF (param nécess, primitive)

CREER-S-RELEASE-RESP (primitive)

VERIFICATION-PARAM+COMPATIBILITE (deridconnexint, reason, état, nbrechps, (n°chp,val)*)

```
regarder si les param à vérifier sont vérifiés
SI pas vérifiés
ALORS
 etat := FTSJ
 reason := validation failure
```

```
SINON
 regarder si on retrouve l'ancienne connexion
     (* à l'aide de deridconnexint *)
 SI non
 ALORS
  reason := cannotrecover
  état := FTSJ
 SINON
  réajuster les tailles de fenêtres en fct de ses contraintes
       (* tout est OK *)
  regarder le param Initial Token Assignment  (* les 3 *)
  SI c'est Sending (* envoyeur garde les jetons *)
  ALORS
   état := CSJ
  SINON
   SI  c'est Receiving  (* le  récepteur du S-CONNECT-IND prend
   les jetons *)
   ALORS
    état := CJ
   SINON (* c'est Acceptor chooses *)
    état := CSJ
   FINSI
  FINSI
 FINSI
FINSI
```

CREER-S-CONNECT-RESP-RECOVER (nbrechps, (n°chp,val)*, reason, primitive)

```
SI reason = ' '
ALORS
 CONNECT-RESP-+
```

VERIFIC-PARAM-COMPATIB-AVECJETONS (deridconnexint, reason, état, nbrechps, (n°chp,valeur)*)

```
regarder si les param à vérifier sont OK
SI non
ALORS
 reason := validation failure
 état := FTJ
```

```
SINON
 regarder si on retrouve l'ancienne connexion
      (* à l'aide de deridconnexint *)
 SI non
 ALORS
  reason := cannotrecover
  état := FTJ
 SINON
  réajuster les fenêtres en fct des contraintes
          (* les  jetons seront  tjs pour  le destinataire  car
  l'envoyeur  soit les  lui donne, soit met acceptor chooses et
  le récepteur les prend *)
  état := CJ
 FINSI
FINSI
```

CREER-RECOVER-REQ (dernumge, deridconnexint, mtavois, donnéesreçues, jetons, idactcourenv, idactcourrec, primitive)

```
SI jeton
ALORS
 initial token assignment := sending
SINON
 SI données reçues = false (* sûr que l'autre a les jetons *)
 ALORS
  initial tokens assignment := receiving
 SINON  (* l'envoyeur  a reçu  des données  --> il pense que le
 récepteur a les jetons *)
  initial tokens assignment := acceptor chooses
 FINSI
FINSI
idactcourenv := 0
idactcourrec := 0
```

CREER-ACTIV-START (idactcourenv, primitive)

DECOUPE-APDU-EN-SSDU (APDU, nbre SSDU, (SSDU)*, deckpointsize)

CREER-S-DATA-REQ (pteur fin, SSDU, primitive)

CREER-SYNCHRO-REQ (pteur fin, taillebuffer, primitive)

GESTION-SYNCHRO (s-synchro-conf, pteur début, taillebuffer, mtavois, état)

```
call DECOUPE-PRIM ( s-synchro-conf, nbrechps, (n°chp,val)* )
SI champ n°2 = pteur début ; état := ES
ALORS
 pteur début := ( pteur début +1 ) modulo taillebuffer
SINON
 suivant la politique on fait
 call CREER-DISCARD ; GESTION DISCARD
 call CREER-INTERRUPT ; GESTION INTERRUPT
 call CREER-ABORT ; GESTION ABORT
```

TEMPS ECOULE (transfertune, mtavois, état, APDU)

```
call CREER-DISCARD ( n° activité, primitive ) ;
GESTION DISCARD
```

CREER-EXCEPTION (APDU, primitive)

EFFACER-APDU (APDU)

GESTION-INTERRUPT (origine, taillebuffer, pteur début, APDU, état, jeton)

```
SI   origine   =   envoyeur   (*   càd   il   existe   déjà   une   act
interrompue *)
ALORS
 call CREER-EXCEPTION
 envoyer  2  primitive
FINSI
dercheckpointint := ( pteur début - 1 ) modulo taillebuffer
SI jetons
ALORS
 idactintenv := idactcourenv
SINON
 idactintrec := idactcourrec
 deridconnexint := deridconnexcour
 messageint := APDU
```

```
pointeur reprise := valeur de [ ( pteur début - 1 ) modulo
taillebuffer ]
SI jetons
ALORS
 état := CJ
 origine := envoyeur
SINON
 état :=CSJ
 origine := réception
```

GESTION-DISCARD (n° activité, primitive, état)

```
call CREER-EXCEPTION
envoyer 2 primitive
call EFFACER-APDU ( APDU )
état := CJ
```

GESTION-ABORT (pteur début, taillebuffer, origine, état, APDU, jetons)

```
SI origine = envoyeur
ALORS
 call CREER-EXCEPTION
 envoyer 2 primitive
FINSI
dercheckpointint := ( pteur début - 1 ) modulo taillebuffer
SI jetons
ALORS
 idactintenv := idactcourenv
SINON
 idactintrec := idactcourrec
 deridconnexint := deridconnexcour
 messageint := APDU
 pointeur reprise := valeur de [ ( pteur début - 1 ) modulo
 taillebuffer ]
 SI jetons
 ALORS
  état := FTJ
  origine := envoyeur
 SINON
  état :=FTSJ
  origine := récepteur
  envoyer 1 mettre bit à 0
```

GESTION-JETONS (s-token-please-ind, état)

call DECOUPE-PRIM (s-token-please-ind, nbrechps, (n°chp,val)*
)
call CREER-TURN-PLEASE-IND (param nécessaires, primitive)
envoyer 2 primitive
état := ES

GESTION-TOUR (état)

SI il donne tout de suite le tour
ALORS
 suivant politique faire
 call CREER-INTERRUPT
 call CREER-DISCARD
 AMtourdonné := true
SINON
 AMtourdonné := true
 état := ES
FINSI

CREER-DISCARD (état)

CREER-INTERRUPT (état)

CREER-ABORT

GESTION-EXCEPTION (état)

selon la politique, le RTS décide de faire
CREER-INTERRUPT ; GESTION-INTERRUPT
CREER-DISCARD ; GESTION-DISCARD
état := CJ

CREER-S-ACTIVITY-END

CREER-RESUME (idactcourenv, dercheckpointint, messageint, pointeurreprise, mtavois)

```
augmenter idactcourenv de 1
lire idactintenv
lire deridconnexint
lire dercheckpointint
lire messageint
lire pointeurreprise
fabriquer Resume : s-activity-rescame-req et l'envoyer
```

GESTION-S-U-EXCEPTION-REPORT-REQ (mtavois, AMtourdemandé, état)

```
file :
vider pipe
mettre dans file d'attente
SI file attente vide
ALORS
 goto file
SINON
 prendre la première demande
 CASE  2  turn-please-req
  call CREER-S-TOKEN-PLEASE-REQ ( param, primitive )
  envoyer  0  mtavois  primitive
  AMtourdemandé := true
  goto file
 CASE  0  s-activity-interrupt-ind
  call GESTION-INTERRUPT
  call CREER-S-ACT-INTER-RESP
  goto fin
 CASE  0  s-activity-discard-ind
  call EFFACER-APDU
  call CREER-ACT-DISC-RESP
  goto fin
 CASE  0  s-u-abort-ind
  call GESTION-ABORT
  goto fin
 CASE  0  s-p-abort-ind
  call GESTION-ABORT
  goto fin

   ( CASE  0  s-control-give-ind )
```

```
 ELSECASE
  écarter la primitive
  goto file
 ENDCASE
finsi
fin : return
```

GESTION-DONNEES (donnéesenattente, ssder, état)

```
donnéesenattente := donnéesenattente + 1
SI donnéesenattente > 1
ALORS
 call TRAIT-ERROR-RECEPT
FINSI
```

GESTION-PTSYNCHRO (taillebuffer, dercheckpoint, mtavois, état, déjàreçu)

```
call DECOUPE-PRIM ( s-synch-min-ind, ..... )
SI 2° param = ( dercheckpoint + 1 ) modulo taillebuffer
ALORS
 dercheckpoint := 2° param
 SI donnéesenattente = 1
 ALORS
  call CREER-S-SYNCH-MIN-RESP
  donnéesenattente := donnéesenattente - 1
  SI not déjàreçu
  ALORS
   messagecour := messagecour +  SSDU
  FINSI
 SINON
  call TRAIT-ERROR-RECEPT
SINON
 call TRAIT-ERROR-RECEPT
FINSI
```

TRAIT-ERROR-RECEPT (état)

```
suivant la gravité de l'erreur, on fait
call CREER S-U-EXCEPTION-REPORT-REQ
call CREER-ABORT ; call GESTION-ABORT
```

```
CREER-S-ACT-INTER-RESP

CREER-S-SYNCH-MIN-RESP

GESTION-FIN ( donnéesenattente, SSDU, état,
deridactsec, idactcourrec, deridconnexsec,
deridconnexcour, déjàreçu )

SI donnéesenattente = 1
ALORS
 call CREER-S-ACT-END-RESP
 SI not déjàreçu
 ALORS
  messagecour := messagecour + SSDU
  call CREER-TRANSFER-IND ( messagecour ... )
 FINSI
 état  := CSJ  ;deridactsec := idactcourrec ; deridconnexsec :=
 deridconnexcour
SINON
 call TRAIT-ERROR-RECEPT
FINSI

CREER-S-ACT-END-RESP

CREER-TRANSFER-IND

CREER-S-ACT-DISC-RESP
```

Annexe 9.

ALGORITHME DU GESTIONNAIRE CENTRAL

```
DEBUT :

POUR pipe 0
FAIRE
 POUR première demande --> dernière demande
 FAIRE
  SI demande vient d'un UA
  ALORS
   écrire dans pipe 3
  SINON
   écrire dans pipe 4
  FINSI
 FINFAIRE
FINFAIRE
POUR pipe 1
FAIRE
 POUR première demande --> dernière demande
 FAIRE
  SI [A] = 0
  ALORS
   enlever [A] et [B]
   écrire dans pipe 0 pour UA
  SINON
   SI A = 1
   ALORS
    enlever [A]
    écrire dans pipe 4
   SINON (* A = 2 *)
    écrire les informations dans la BD
 FINFAIRE
FINFAIRE
POUR pipe 2
FAIRE
 POUR première demande --> dernière demande
 FAIRE
  SI [A] = 0
  ALORS
   enlever [A] et [B]
   écrire dans pipe 0 pour couches inférieures OSI
  SINON
   enlever [A]
```

```
    écrire dans pipe 3
  FINSI
 FINFAIRE
FINFAIRE
goto DEBUT
```

Annexe 10.

LE PROBLEME DES ADRESSES

10.1. INTRODUCTION

Dès que l'on aborde le problème de la réalisation physique d'une communication à l'intérieur d'un réseau, les notions de nom, adresse et routage deviennent très importantes, mais hélas souvent ambiguës.

Le **nom** d'un hôte, d'une boîte aux lettres ou autre ressource est le mot employé par l'usager pour indiquer la ressource désirée.

L'**adresse** spécifie l'emplacement de cette ressource au software du réseau.

Le **routage** est utilisé par le software du réseau pour déterminer comment atteindre cette ressource.

Adresse et nom sont relatifs aux protocoles entre réseaux.

X400 ne fait pas clairement la distinction entre nom et adresse. Une adresse X400 est un nom décrivant et caractérisant l'UA en vue d'aider le MTS à situer le point d'attache de l'UA. Chaque adresse X400 est un nom X400, mais l'inverse n'est pas toujours vrai (?!)

10.2. LA STRUCTURE D'ADRESSE PROPOSEE.

Pour identifier les utilisateurs et leurs UA, la norme X400 propose une structure descriptive plutôt que l'assignation d'un nom primitif par une autorité d'application.

exemple : nom primitif: numéro de l'employé.

autorité d'appellation: organisation qui emploie.

nom descriptif: chef de département.

On parle de nom descriptif lorsque la description correspond à une et une seule entité.

Pour ce faire, on spécifie un ou plusieurs attributs et leur valeur.

Le nom descriptif qui fait référence aux usagers et leur UA est l'O/R-NAME (Origine - Recipient Name).

Une adresse - O/R est le nom descriptif d'un UA. Elle possède certaines caractéristiques qui aident le MTS à situer le point de connexion de l'UA.

X400 précise un ensemble d'attributs normalisés :

a) d'ordre personnel : nom personnel.

[nom de famille, prénom, initiales, indicateur généalogique (ex: JR)].

b) d'ordre géographique : nom du pays.

nom de la région.

c) relatifs à l'organisation : nom de l'organisation.

unité dans l'organisation.

d) d'ordre architecturaux : adresse X121.

identificateur unique d'UA.

nom du domaine de gestion

de l'administration.

nom du domaine de gestion

privé.

e) définis par le domaine (non normalisés, valables uniquement pendant une période intérimaire).

Un ensemble d'attributs de base est égal à l'ensemble minimal d'attributs dont les valeurs identifient sans ambiguïté un domaine de gestion particulier.

Deux ensembles d'attributs de base liés à l'architecture et au terminal ont été choisis :

A. Forme 1 : Architecture

variante 1 : nom des pays.

nom du domaine de l'administration.

au moins un nom du domaine privé.

nom personnel.

nom de l'organisation.

nom de l'unité administrative.

attributs définis par le domaine (facultatif).

variante 2 : nom du pays.

nom du domaine de l'administration.

identificateur numérique d'UA.

attributs définis par le domaine (facultatif).

variante 3 : nom du pays.

nom du domaine de l'administration.

adresse X121.

attributs définis par le domaine (facultatif).

B. Forme 2 : Terminal.

adresse X121

identificateur de terminal télématique.

La forme la plus adaptée pour la messagerie électronique interpersonnelle semble être la forme 1 variante 1.

exemple : nom du pays: "Belgique".

nom de l'administration :

"service d'enseignement belge".

nom du domaine privé: "FNDP".

nom de l'organisation: "Institut d'informatique".

nom de l'unité administrative: "domaine télécom".

nom personnel :(nom de famille) "Simonet"

S'il n'y avait qu'une seule forme d'adresse, il serait tentant d'imposer un ordre, et de séparer chaque champ d'attribut par un sigle spécial :

*Simonet * domaine télécom * Institut d'info * FNDP * Belgique.

Cependant, il est impossible de se limiter à une seule forme d'adresse. Toutes doivent être admises. Dès lors, différentes possibilités peuvent être envisagées :

A. Première solution.

numéroter les quatre variantes d'O/R-Name.

imposer un ordre à l'intérieur de chaque variante.

le type d'attribut est déterminé par la place occupée par la valeur au sein de la suite d'attributs.

chaque champ de valeur d'attribut est séparé par un sigle spécial (par exemple "*").

exemple :

1) 1 * Simonet * * * * domaine télécom * Institut d'informatique * FNDP * * Belgique.

"1" signifie "première variante, première forme".

2) 4 * 20623334 *

"4" signifie "deuxième forme".

Cependant, il n'est pas facile de retenir l'ordre des attributs. De plus, si un type d'attribut n'apparaît pas dans l'O/R-Name, il faut quand même laisser un champ vide.

exemple :

le champ "nom personnel" se compose de quatre parties (nom, prénom, initiales, indicateur généalogique). Pour rester cohérent, il faut laisser un champ vide pour les trois dernières valeurs.

D'autre part, dans le cas d'une modification de la norme, les changements pourraient se révéler conséquents.

B. Deuxième solution.

représenter chaque attribut par l'association de son type et de sa valeur.

séparer chaque association type-valeur par un sigle spécial (exemple: "*").

Cette solution a l'avantage de n'imposer aucun ordre sur les attributs, mais présente le désavantage d'allonger l'O/R-Name.

Type d'attribut	Représentatif
nom du pays	1
nom du domaine de l'administration	2
nom du domaine privé	3
nom de l'organisation	4
nom de l'unité administrative	5
nom	6
prénom	7
initiales	8
indicateur généalogique	9
adresse X121	10
identificateur du terminal télématique	11
identificateur numérique d'UA	12

exemple : * 6: Simonet * 5: domaine télécom * 4: Institut d'informatique * 2 : service d'enseignement belge * 3: FNDP * 1: Belgique.

Cette structure a l'avantage d'être très souple et de nécessiter peu de changements en cas de modification.

D'autre part, si dans l'avenir, il était possible de disposer d'un système d'annuaire qui permettrait de trouver l'entièreté d'un O/R-Name à partir de quelques attributs, cette solution semble beaucoup mieux adaptée. En effet, on peut imaginer qu'en fournissant le nom d'une personne, l'annuaire suffirait à reconstituer le reste de l'O/R-Name. Dans ce cas, il paraît plus simple d'écrire

* 6: Simonet *

selon la deuxième solution, que

1 * Simonet ********

selon la première.

En conséquence, malgré la longueur et la nécessité de retenir les identificateurs de types d'attributs, c'est la deuxième solution qui est proposée, principalement pour la liberté qu'elle laisse à l'utilisateur.