MASTER EN SCIENCES INFORMATIQUES

Étude du logiciel de courrier électronique EAN

Rutten, Xavier; Vanthournout, Joël

Award date:
1989

Awarding institution:
Universite de Namur

Année académique 1988-1989

Facultés Universitaires Notre-Dame de la Paix
Institut d'Informatique
B-5000 Namur, Belgique

Etude du logiciel de courrier électronique EAN

Xavier Rutten
Joël Vanthournout

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

Facultés Universitaires Notre-Dame de la Paix
Institut d'Informatique
Rue Grandgagnage, 21, B - 5000 NAMUR (Belgium)

Etude du logiciel de courrier électronique EAN

Xavier Rutten
Joël Vanthournout

Résumé.

Ce travail étudie le logiciel de messagerie électronique EAN. Il rappelle d'abord les principaux concepts des recommandations X.400, sur lesquelles est basé EAN. Il analyse ensuite, dans un premier temps, l'architecture du logiciel et les interactions de ses différentes composantes. Dans un deuxième temps, ses sources en langage C sont abordées. Un chapitre est également consacré à l'installation du logiciel aux Facultés Universitaires Notre-Dame de la Paix et aux améliorations que nous avons tenté d'y apporter.

Abstract.

This work is a study about the EAN distributed message system. It first recalls the main concepts in recommendations X.400, on which EAN is based. Then, it analyzes the software architecture and the relations between its components. Thereafter, it studies its sources in C language. One chapter also deals with the EAN installation at the Facultés Universitaires Notre-Dame de la Paix and with the improvements we have tried to effect in it.

Mémoire de Licence et Maîtrise en Informatique
Août 1989
Directeur : Philippe Van Bastelaer

Nous tenons ici à exprimer toute notre gratitude à Monsieur le Professeur Philippe Van Bastelaer qui a accepté d'assurer la direction de ce mémoire et qui, grâce à ses conseils judicieux, nous a guidés et nous a permis de mener à bien cette étude.

Nous remercions également son assistant Monsieur Bernard Detrembleur pour la disponibilité dont il a fait preuve chaque fois qu'un problème technique ou pratique survenait.

Table des matières

Introduction

Le nombre d'ordinateurs dans le monde est en croissance perpétuelle. Beaucoup d'utilisateurs souhaitent interconnecter leurs ordinateurs entre eux. Une application pratique pouvant en résulter est le **courrier électronique**, appelé aussi **messagerie électronique**.

Il s'agit essentiellement de permettre à chaque utilisateur d'envoyer un message quelconque à un ou plusieurs autres utilisateurs, même s'ils ne sont pas connectés à ce moment. Le message est déposé dans une "boîte aux lettres" électronique au sein de l'ordinateur utilisé par le destinataire du message. Le destinataire peut alors, lorsqu'il se connecte à son ordinateur, examiner le contenu de sa boîte aux lettres, lire les lettres s'y trouvant et y répondre.

Ce service de messagerie électronique ne devient cependant réellement intéressant que dans la mesure où le nombre de personnes interconnectées (c.-à-d. de personnes possédant une "boîte aux lettres") est relativement élevé. Il est donc très important de favoriser l'accès à ce service au plus grand nombre de personnes possible, quelles que soient les caractéristiques de leurs systèmes informatiques. C'est pourquoi l'organisme de standardisation CCITT a défini une norme décrivant les fonctionnalités et l'architecture d'un standard d'un tel système de messagerie électronique : il s'agit de l'avis **CCITT X.400**[1] (anciennement **X.MHS**).

Un des premiers logiciels implémentant une bonne partie des fonctionnalités prévues par X.400 est le logiciel EAN ("Electronic Access Network"), dont la première version a été réalisée en 1984 par la University of British Columbia (UBC). Il est écrit en langage C et est disponible sous BSD Unix et sous DEC/VMS.

L'objet de ce travail est d'étudier la version BSD Unix de EAN, installée aux FUNDP, et cela essentiellement en ce qui concerne son fonctionnement dans la couche application.

Le chapitre 1 rappelle les concepts principaux de X.400. De nombreuses références y seront faites ultérieurement.

Le chapitre 2 décrit l'architecture globale de EAN, sans aborder les sources des programmes.

Le chapitre 3 décrit la façon dont le routage est réalisé par EAN.

Le chapitre 4 décrit les différents programmes qui composent EAN et leurs interactions pour offrir le service de messagerie électronique.

Le chapitre 5 décrit l'implémentation de EAN aux FUNDP et les améliorations que nous avons tenté d'y apporter.

---

1. Plus précisément: les avis CCITT X.400 à X.430.

Le chapitre 6 décrit la syntaxe de transfert utilisée par EAN (X.409).

Le chapitre 7 décrit trois structures de données très employées dans les sources de EAN. Cette description est nécessaire à la compréhension du chapitre 8.

Le chapitre 8 décrit le parcours d'un message dans EAN, en examinant les fonctions C successivement rencontrées par celui-ci.

L'annexe A reprend la répartition des tâches accomplies par les deux auteurs.

L'annexe B reprend la localisation dans les fichiers sources des fonctions C considérées dans ce travail.

Remarque :

Tout au long de ce mémoire, nous utiliserons les abréviations et les termes anglais. Nous parlerons ainsi par exemple de MTA (Message Transfer Agent) et non d'ATM (Agent de Transfert de Message). Ce choix est justifié par le fait que les termes anglais sont beaucoup plus répandus - et donc connus - que leurs homologues français. De plus, ils sont en général plus concis et plus précis.

Chapitre 1 : L'avis X.400

Ce chapitre décrit la norme X.400 définie en 1984. Il est largement inspiré de [X400], [X401], [X410], [X411] et [X420]. [MHS1], [MHS2], [MHS3], [PB] (chapitre 2) et [RT] (chapitre 14) sont d'autres références également intéressantes, mais moins complètes.

La dernière section de ce chapitre (1.10.) décrit les modifications apportées dans la version 1988 de X.400.

1.1. L'architecture générale

1.1.1. Le principe général

Le principe général du système de messagerie électronique X.400 est représenté à la figure 1.1.

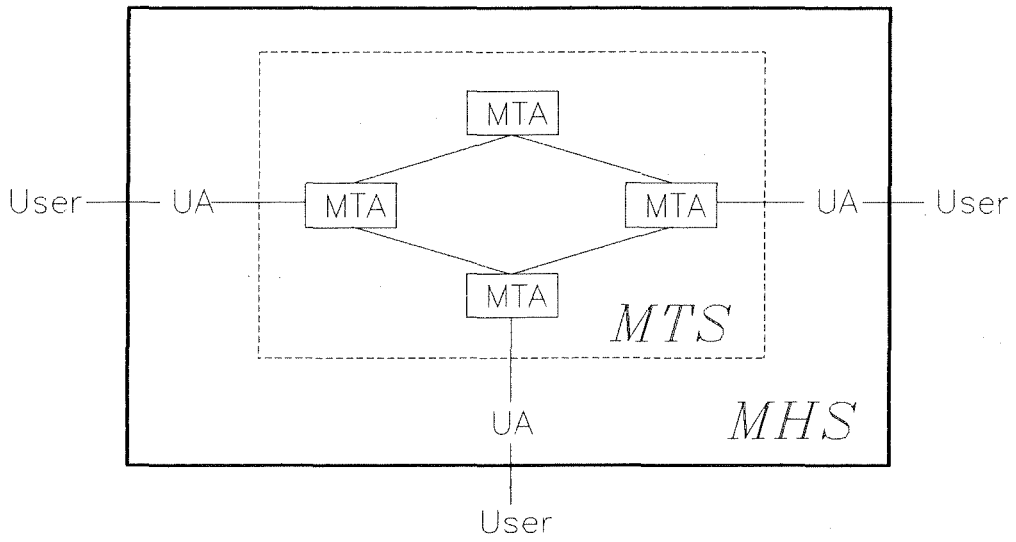


Figure 1.1 : le MHS

Cette figure repose sur les définitions suivantes:

Définitions:

- Un **UA (User Agent)** est un dispositif permettant à l'**utilisateur** ("**user**") de préparer, d'envoyer et de recevoir des messages.
- Un **MTA (Message Transfer Agent)** est un dispositif qui reçoit un message en provenance d'un UA ou d'un autre MTA et le réexpédie vers un autre UA ou un autre MTA[1]. C'est donc l'agent actif qui s'occupe du transfert des messages d'un endroit à l'autre.

1. Dans certains cas, le MTA réexpédie le message vers plusieurs autres UAs ou MTAs (plusieurs destinataires).

- Le **MTS (Message Transfer System)** est l'ensemble de tous les MTAs interconnectés. C'est donc le réseau proprement dit où circulent les messages.
- Le **MHS (Message Handling System)** est constitué du MTS et de tous les UAs qui y sont connectés.

Un utilisateur peut être une personne humaine ou une application informatique quelconque.

Chaque utilisateur est associé à un et un seul UA. Chaque UA est associé à un MTA[1].

Un utilisateur expéditeur d'un message est appelé **originator**, un utilisateur destinataire d'un message est appelé **recipient**.

Fonctionnement:

- l'utilisateur originator prépare son message sur son UA et le **soumet** ("**submission**") au MTA associé à son UA;
- le MTA examine l'adresse du destinataire du message et en déduit à qui il doit envoyer le message :
  - soit à l'UA du destinataire (si cet UA est connecté directement au MTA)
  - soit à un autre MTA qu'il sait "plus proche" de l'UA du destinataire. Dans ce dernier cas, le nouveau MTA répète le même processus : il examine l'adresse du destinataire et envoie le message à l'UA du destinataire ou à un troisième MTA...
- et ainsi de suite jusqu'à ce que le message soit **(dé)livré** ("**delivery**") à l'UA du destinataire. Le destinataire peut alors lire sur son UA le message qui lui a été envoyé.

Les messages sont donc envoyés selon un principe de **store and forward** : chaque MTA intermédiaire recevant un message le sauve temporairement ("**store**") en attendant de le renvoyer ("**forward**") au MTA ou à l'UA suivant. Le message arrivera de proche en proche à destination.

Remarques:

1°) Le système comprend des sécurités pour éviter le bouclage de messages (c.-à-d. qu'un message ne revienne à un MTA par lequel il est déjà passé): voir 1.6.2.A.

2°) Les MTAs peuvent transporter n'importe quels types de messages, même des messages binaires quelconques (voir [X400], section 2.2.2.2). Ils ne modifient jamais le contenu des messages[2] (voir [X411], section 3.4.2.2).

1.1.2. Les différents types d'UAs et l'IPM

Tous les UAs ne manipulent pas le même genre de messages et n'offrent pas les mêmes services à leurs utilisateurs (humains ou informatiques).

On a donc groupé les UAs selon le type du contenu des messages qu'ils manipulent. Des UAs de même type sont appelés **cooperating UAs** : de tels UAs s'échangent des messages de même type et **coopèrent** pour offrir aux utilisateurs un certain nombre de services bien définis. Pour offrir ces services, un protocole bien précis doit être respecté entre les UAs coopérants (voir section 1.4.).

Remarquons que lorsqu'un UA soumet un message à un MTA, il peut spécifier sa classe et donc le type des UAs avec qui il peut communiquer (voir section 1.5.1.).

---

1. A notre connaissance, un UA n'est jamais associé en pratique qu'à un seul MTA. Il nous semble cependant très possible qu'un UA soit associé à plusieurs MTAs et qu'il ait donc plusieurs "points d'accès" au MTS.

2. Sauf quand un UA demande une conversion de données.

Les UAs relatifs au système de messagerie électronique forment une classe particulière d'UAs appelée **IPM UAs (Interpersonal UAs)**. L'**IPMS (Interpersonal Messaging System)** est constitué du sous-ensemble du MHS où l'on ne reprend que les UAs qui sont des IPM UAs[1]. Les services offerts par ces UAs sont décrits dans la section 1.5.2.. Le protocole utilisé est appelé protocole **P2** et est décrit dans la norme X.420.

1.1.3. Les trois configurations élémentaires

Le réseau des UA et des MTA peut être structuré de n'importe quelle manière, pourvu que chaque UA soit relié à un MTA. Cependant, il est possible de distinguer 3 configurations élémentaires :

- **S1** : système ne fournissant que des fonctions d'UA (un UA par utilisateur);
- **S2** : système ne fournissant que des fonctions de MTA;
- **S3** : système fournissant à la fois des fonctions d'UA et de MTA.

Quelques exemples d'architectures souvent rencontrées sont donnés aux figures 1.2, 1.3 et 1.4. Dans la figure 1.2, les UAs et le MTA associé sont co-résidents (ils résident dans le même système). La figure 1.3 illustre à la fois le cas d'un UA et d'un MTA co-résidents (S3) et la cas d'un UA et d'un MTA séparés (S1 et S2 : S2 sert d'intermédiaire entre S1 et le MTS). La figure 1.4 comprend une autre combinaison d'UAs et de MTAs co-résidents et séparés.

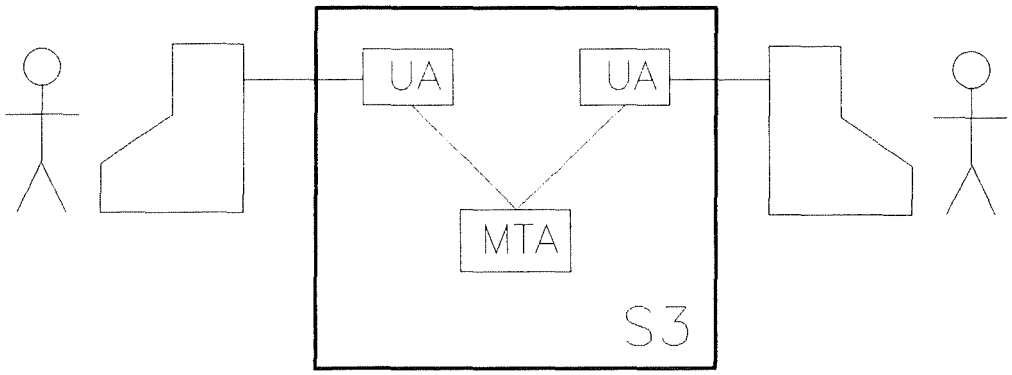


Figure 1.2 : UA et MTA co-résidents

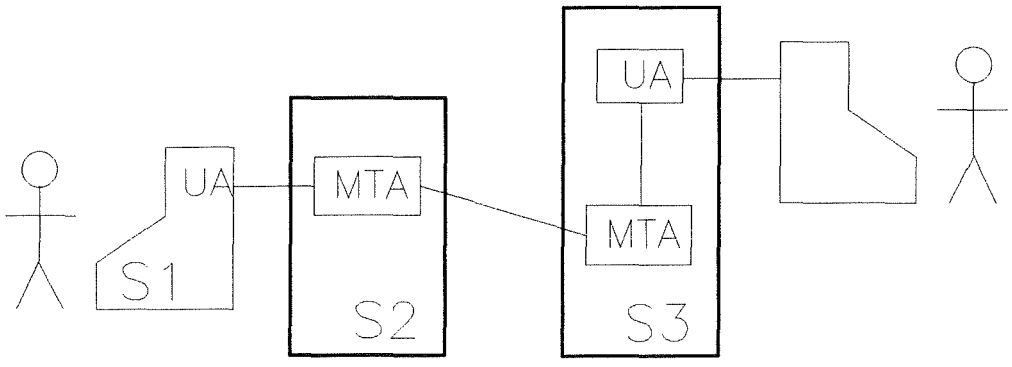


Figure 1.3 : UA et MTA séparés

1. Pour être précis, signalons que l'IPMS comprend également des accès aux services du télex et aux services télématiques. En 1984, seule une partie de ceux-ci était normalisée (à savoir: le protocole d'accès aux terminaux télétex, normalisé dans X.430 (voir [X430])).

Remarque: on désigne par **service télématique** l'ensemble des services publics d'information à usages domestiques et bureautiques, tels que le **télétex** (service public de transfert de textes entre machines de traitement de textes) - et le **vidéotexte**. Notons qu'il ne faut pas confondre le **télétex** avec le **télétexte**, ce dernier étant un service de vidéotexte particulier (**vidéotexte diffusé**, utilisant le réseau de diffusion de la télévision). Le lecteur trouvera plus de renseignements à se sujet dans [MG], pages 472-473 et 496-497, et dans [AT], pages 532 et 574-576.

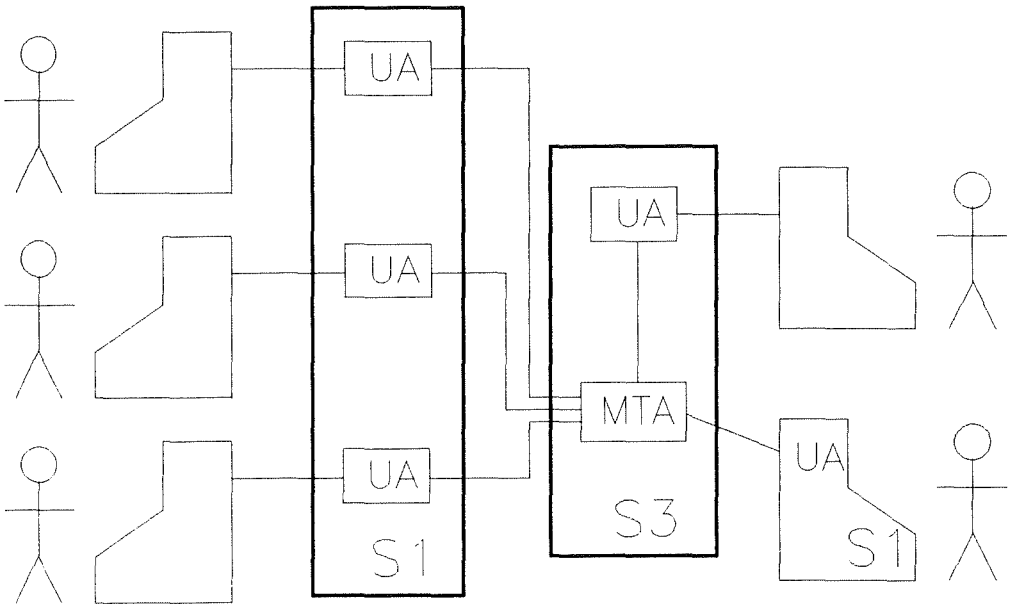


Figure 1.4 : combinaison d'UAs et MTAs co-résidents et séparés

1.1.4. Les Domaines de Gestion

Pour faciliter la gestion et l'administration du réseau, l'on regroupe les MTAs et UAs en **Domaines de Gestion** (ou **Management Domains : MD**) : un MD est une partie du MHS contrôlée par un organisme unique; cette partie doit nécessairement comprendre au moins un MTA (mais elle peut comprendre un nombre quelconque d'UAs: 0, 1 ou plusieurs).

Si l'organisme gérant le MD est une administration publique, alors on appelle le MD **ADministration Management Domain (ADMD)**. S'il est privé, on l'appelle **PRivate Management Domain (PRMD)**.

En outre, la découpe en MD fait les hypothèses suivantes (voir [X400], section 2.3.2.4):

1°) un MD doit toujours être contenu entièrement dans un seul pays;
2°) un PRMD ne peut jamais servir d'intermédiaire entre deux ADMDs;
3°) un PRMD ne peut être relié directement à un ADMD d'un autre pays : il doit passer par l'intermédiaire d'un ADMD du pays du PRMD.

Remarques :

1°) L'interconnexion entre 2 PRMDs (même de pays différents) n'est pas interdite, mais elle ne fait pas l'objet des recommandations X.400.

2°) Quand un ADMD interagit avec un PRMD, c'est l'ADMD qui est responsable des messages échangés entre les deux MDs, aussi bien dans un sens que dans l'autre (la responsabilité comprend les concepts de comptabilisation, logging, sécurité...).

3°) Ces hypothèses sont aujourd'hui remises en question (voir section 1.10.6.).

1.2. L'adressage

1.2.1. Les O/R names et les O/R address

Chaque utilisateur est identifié par un nom appelé **O/R name** (**Originator/Recipient name**). Comme chaque utilisateur est associé à un et un seul UA, chaque O/R name identifie aussi bien un utilisateur que son UA.

Actuellement, un O/R name n'est rien d'autre qu'une liste d'un ou plusieurs **attributs** (un attribut est une caractéristique quelconque de l'UA).

Lorsque les attributs de l'O/R name spécifient **où** se trouve l'UA de l'utilisateur, on dit que cet O/R name est une **O/R address**. Une O/R address est donc un O/R name particulier qui permet de **localiser** l'utilisateur. L'information contenue dans l'O/R address du recipient d'un message sera utilisée pour déterminer le chemin à suivre par ce message (voir section 1.3.). Un O/R name qui n'est pas une O/R address doit donc d'abord être converti en une O/R address pour déterminer la route du message. Ceci nécessite l'utilisation d'un service d'annuaire ("**directory service**") contenant pour chaque utilisateur la liste de ses attributs, et en particulier une O/R address. Ce directory service n'était pas normalisé lors de l'élaboration de la version 1984 de X.400. Depuis lors, des travaux ont conduit à l'élaboration de la norme X.500 (ISO Draft Proposal 9594-1, en 1988).

1.2.2. Les attributs possibles

X.400 reconnaît un certain nombre d'attributs standardisés (**standard attributes**) et des attributs non standardisés, définis uniquement dans le domaine de l'utilisateur (**domain-defined attributes**, i.e. **DDA**s).

A) Les attributs standardisés

Les attributs standardisés sont actuellement classés en 4 catégories. Ces catégories sont données ci-dessous, avec des exemples d'attributs possibles dans chaque catégorie. D'autres attributs et catégories peuvent être standardisés par la suite.

i) Attributs personnels

Exemples: nom personnel (comprenant le nom de famille ("name"), le(s) prénom(s) ("given name(s)"), les initiales, l'indication de la génération (Ex.: "Jr."))

ii) Attributs géographiques

Exemples: nom de rue et numéro, ville, région, pays

iii) Attributs organisationnels

Exemples: nom de l'organisation, nom de l'unité organisationnelle, position ou rôle

iv) Attributs architecturaux

Exemples: adresse X.121[1], identifiant unique de l'UA, nom de l'ADMD, nom du PRMD

1. X.121 est un avis CCITT définissant le plan de numérotage international pour les réseaux publics de données (cf. [MG], pages 351 et 842).

B) Les attributs définis dans le domaine de l'utilisateur

Les attributs définis dans le domaine de l'utilisateur (DDAs) ont été introduits essentiellement pour permettre l'interconnexion X.400 des systèmes existants qui utilisaient déjà certains attributs non standardisés ou d'autres méthodes d'adressage (voir par exemple l'adressage RFC822 dans la version UBC1 de EAN, section 2.3.3.).

A long terme, les systèmes devraient utiliser uniquement des attributs standardisés, et les DDAs devraient donc disparaître.

1.2.3. L'ensemble d'attributs de base

On appelle **ensemble d'attributs de base (base attribute set)** un ensemble d'attributs permettant d'identifier un MD particulier.

Remarque: cette identification peut ou non nécessiter l'utilisation d'un directory service.

1.2.4. Le choix initial d'O/R names

La recommandation X.400 1984 prévoit qu'initialement, il est difficile de supporter tous les attributs standardisés et toutes les formes possibles d'O/R names. Dès lors, elle prévoit un certain nombre de formes standardisées d'O/R names (d'autres formes pourront être ajoutées par la suite).

Initialement, tous les O/R names devraient comprendre un ensemble de base d'attributs. En effet, comme un directory service global (connaissant tous les attributs de tous les UAs du monde) n'était pas standardisé initialement en 1984, l'ensemble d'attributs de base permet de déterminer immédiatement (sans directory service) vers quel MD envoyer un message. Lorsque le message est arrivé dans le MD destinataire, un éventuel directory service local (propre au MD destinataire) donnera, si nécessaire, plus d'informations sur les attributs de l'UA destinataire (attributs standardisés, mais aussi DDAs).

Initialement, un seul ensemble d'attributs de base doit être reconnu [1]:

- si l'UA appartient à un ADMD:
  nom du pays + nom de l'ADMD de l'UA

- si l'UA appartient à un PRMD:
  nom du pays + nom de l'ADMD associé au PRMD de l'UA

Plusieurs formes et variantes d'O/R names sont reconnues initialement. La principale est la suivante :

ensemble d'attributs de base:
  nom du pays
  nom de l'ADMD

1. plus un autre ensemble d'attributs de base utilisé dans le contexte des services télématiques : il s'agit d'une adresse X.121, d'une adresse télex ou d'un identifiant d'un terminal d'un service télématique. Il permet d'identifier directement l'UA d'un utilisateur d'un service télématique et donc à fortiori le MD auquel il appartient.

attributs identifiant l'utilisateur dans son MD:

[nom du PRMD]
[nom personnel de l'utilisateur]
[nom de l'organisation]
[nom de l'unité organisationnelle]
[DDAs]

Remarques:

1°) Les attributs entre [] sont facultatifs (mais il doit y avoir suffisamment d'attributs pour identifier un et un seul UA). De plus, au moins un des 4 premiers attributs doit être présent.

2°) Toutes les formes et variantes d'O/R name reconnues initialement sont des O/R address.

1.3. Le routage

1.3.1. Définition

La **route** est le chemin suivi par un message depuis l'UA originator jusqu'à l'UA recipient.

Remarque :

- l'avis X400 considère uniquement le routage entre le MD originator (c.-à-d. le MD auquel appartient l'UA originator) et le MD recipient (c.-à-d. le MD auquel appartient l'UA recipient)
- une fois que le message est arrivé dans le MD recipient, le chemin qu'il doit suivre pour arriver au MTA recipient n'est pas précisé par l'avis X400: il est déterminé par le MD.

1.3.2. Principe du routage

Chaque MD le long de la route détermine le MTA dans le MD suivant où il faut réexpédier le message, mais il ne s'occupe pas de la suite du chemin à suivre. Remarquons en outre que les critères de sélection du prochain MD ne sont pas précisés par l'avis : ils peuvent être techniques, politiques, économiques, ... Dans tous les cas, ils sont évidemment basés sur les attributs de base de l'O/R address.

1.3.3. Fonctionnement

Lorsque un message arrive dans le MTA d'un MD, les opérations suivantes sont effectuées :

- le MTA examine les attributs de base de l'O/R address du message;
- il détermine ainsi si l'UA recipient est dans le même MD (c.-à-d. si le MD est le MD recipient);
- si oui : il détermine le chemin depuis le MTA jusqu'à l'UA recipient (sur base des attributs utilisateurs);

  si non : il réexpédie le message au MD suivant, déterminé par les attributs de base.

Remarquons que ce mode de fonctionnement a pour conséquence que les attributs utilisateurs ne peuvent être testés que lorsque le message est arrivé dans le MD recipient, tandis que les attributs de base peuvent être testés dès la soumission du message à un quelconque MTA.

Remarque :

Si le MD recipient est privé (PRMD), alors les attributs de base déterminent en fait l'ADMD relié au PRMD et non le PRMD. Les attributs utilisateurs déterminent alors le PRMD proprement dit et l'UA recipient dans le PRMD (voir section 1.2.4.).

1.4. La représentation en couches du modèle MHS

Le modèle MHS s'inscrit dans la 7e couche du modèle OSI. Cette 7e couche MHS est divisée en deux sous-couches appelées **UAL (UA Layer)** et **MTL (Message Transfer Layer)**. L'UAL comprend essentiellement les fonctions associées au contenu des messages, la MTL les fonctions associées au transfert des messages.

Les entités et les protocoles de ce modèle sont représentés dans la figure 1.5.

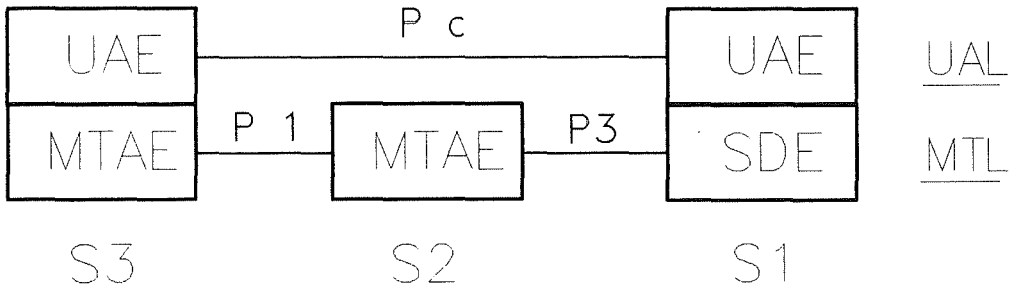


Figure 1.5 : entités et protocoles du modèle MHS

On distingue les 3 types de systèmes déjà vus précédemment:

- S1 : système ne fournissant que des fonctions d'UA;
- S2 : système ne fournissant que des fonctions de MTA;
- S3 : système mixte (fournissant des fonctions d'UA et de MTA).

L'**UAE (UA Entity)** et la **MTAE (MTA Entity)** sont les entités fonctionnelles correspondant à l'UA et au MTA. La **SDE (Submission and Delivery Entity)** a pour rôle de servir d'intermédiaire entre l'UA isolé et le MTS plus éloigné. Autrement dit, la SDE permet d'offrir à la couche UAL du S1 les services de la couche MTL, comme si le MTA était co- résident avec l'UA.

Les protocoles définis sont les suivants :

- **P1** (**"Message Transfer Protocol"**) : définit la façon dont un message est réexpédié depuis un MTA vers un autre MTA; de façon plus générale : il spécifie les interactions entre deux MTAs.
- **P3** (**"Submission and Delivery Protocol"**) : définit comment un UA isolé (donc appartenant à un système S1) peut disposer des mêmes services de la couche MTL qu'un UA relié à un MTA co-résident (système S3). Pour cela, il définit les interactions entre une SDE et une MTAE.
- **Pc** : il s'agit d'une classe de protocoles, chacun d'eux définissant la syntaxe et la sémantique d'un type de message particulier à transférer entre des UAEs. On distingue par exemple :

  - **P2** : protocole de courrier électronique
  - **P10** : protocole de transfert de fichiers

- **P11** : protocole de conférence
- **P12** : protocole de soumission de travaux
- ...

Parmi tous les protocoles Pc, ce mémoire ne considère que le protocole P2, appelé aussi **Interpersonal Messaging Protocol (IPM Protocol)**. Dans ce cas, les UAs communiquant entre eux forment une classe particulière et sont appelés **IPM UAs** et le service offert par l'UAL est appelé **IPM Service.**

Remarque :

- P1 et P3 sont définis dans l'avis X.411;
- P2 est défini dans l'avis X.420.

1.5. Les éléments de service

1.5.1. Les services offerts par le MTL

Les services offerts par le MTL comprennent essentiellement l'envoi d'un message d'un UA vers un autre UA ou vers plusieurs autres UAs, avec de nombreuses options telles que:

- la gestion d'accusés de **livraison** ou de **non-livraison** à l'UA recipient;
- la spécification des types d'informations encodées dans le message, avec conversions éventuelles de types[1];
- la spécification par un UA au MTS du type des messages que l'UA reconnaît et peut se voir délivrer;
- la possibilité de livraison d'un message à telle heure/date;
- l'identification des messages par un numéro;
- le test que l'UA recipient existe et peut être atteint (par un message test (sonde) appelé "**probe message**");
- la protection de l'accès au MTA et à l'UA par mots de passe[2];
- la conservation dans le message véhiculé dans le MTS d'une trace des endroits par où il est passé (voir le contenu des messages échangés entre MTAs, section 1.6.2.A.)
- ...

La liste complète et précise de ces services est reprise dans [X400], section 4.1 : "Message transfer service".

1.5.2. Les services offerts par l'IPM UAL

Les services offerts par l'IPM UAL sont appelés "**IPM Services**" (**IPMS**). On distingue d'une part les services nécessitant la coordination et la coopération de plusieurs utilisateurs et de leurs IPM UAs (exemple: accusé de réception: voir ci-dessous), d'autre part les services ne nécessitant pas une telle coordination (exemple: traitement de texte évolué pour composer les messages, facilités d'archivage et de

---

1. Les différents types d'information encodées et les règles de conversion entre ces types sont décrites dans la norme X.408: voir [X408].

2. Lorsqu'une interaction est initialisée entre une UAE et une MTAE, la demande peut provenir soit de l'UAE (exemple: soumission de message), soit de la MTAE (exemple: livraison de message).

- Si la demande provient de l'UAE: l'UAE doit spécifier son O/R name et le mot de passe associé à l'UAE (ce mot de passe "valide" l'UAE auprès de la MTAE et autorise donc l'interaction).
- Si la demande provient de la MTAE: la MTAE doit spécifier son nom et le mot de passe associé à la MTAE (ce mot de passe "valide" la MTAE auprès de l'UAE et autorise donc l'interaction).

Ces règles sont décrites par les primitives **LOGON.Request/Confirmation** et **LOGON.Indication/Response**, dans [X411], sections 2.2.1. et 2.2.2.. Les mots de passe peuvent être modifiés à l'aide des primitives **CHANGE-PASSWORD.Request/Confirmation** et **CHANGE-PASSWORD.Indication** (voir [X411], sections 2.2.12. et 2.2.13.).

consultation des messages). Les premiers services doivent être standardisés et sont définis dans la Recommandation X.400, les seconds ne doivent pas être standardisés (ils sont appelés **local IPM UA services**) et ne sont donc pas repris dans X.400.

Voici quelques exemples d'IPM services standardisés et définis dans X.400:

- tous les services offerts par la couche MTL;
- la numérotation des IP-messages (voir section 1.6.1.A. pour la définition d'un IP-message);
- des indications diverses sur le message (sujet, date limite de validité, importance, références à d'autres messages...);
- des accusés de **réception** ou de **non réception** par l'utilisateur recipient (la **réception** d'un message signifie non seulement que le message est arrivé à l'UA recipient, mais également qu'il a été lu par l'utilisateur recipient)[1];
- ...

La liste complète et précise des IPM services standardisés est reprise dans [X400], section 4.2 : "Interpersonal messaging service".

1.5.3. La classification des différents services

Certains des services vus ci-dessus sont inhérents au MHS, c.-à-d. doivent obligatoirement être supportés pour que le transfert de message s'effectue correctement (exemple: soumission d'un message). On les appelle **basic services.**

Les autres services, appelés **optional user facilities,** ne sont pas indispensables au bon fonctionnement du MHS (exemple: signaler le sujet du message). Ils peuvent être sélectionnés soit pour chaque message, soit pour une période de temps donnée. Cependant, pour fonctionner correctement, ces services doivent être supportés par tous les MTAs sur la route du message. On a donc établi une liste des services qui doivent obligatoirement être fournis par toutes les administrations publiques (en plus des basic services). Ces services sont appelés **essential optional user facilities,** tandis que les autres sont appelés **additional optional user facilities.**

Si les services proprement dits sont décrits dans la norme X.400 (voir [X400]), la classification des différents services en **basic services, essential optional user facilities** et **additional optional user facilities** est décrite dans la norme X.401 (voir [X401]).

1.6. Les messages échangés entre UAs et entre MTAs

Les UAEs s'échangent des unités de données appelées **UAPDUs (User Agent Protocol Data Units),** en respectant un protocole **Pc (P2** dans le cas d'IPM UAs). Les MTAEs s'échangent des unités de données appelées **MPDUs (Message Protocol Data Units),** en respectant le protocole **P1.**

Lorsqu'un utilisateur désire envoyer un message à un autre utilisateur, son message est d'abord placé par son UA dans un UAPDU. Cet UAPDU est alors passé au MTA (par la primitive de service SUBMIT.Request) qui le considérera comme un nouveau message à communiquer et l'enveloppera dans un MPDU.

1. Note: aucune importance juridique ne peut être attachée à ces accusés de (non)-réception.

La section 1.6.1. décrit brièvement la syntaxe des UAPDUs échangés entre IPM UAEs; la section 1.6.2. décrit brièvement celle des MPDUs échangés entre MTAEs. La syntaxe précise et complète de ces UAPDUs et de ces MPDUs est reprise respectivement dans [X420], section 3 et dans [X411], section 3.4.

1.6.1. Les UAPDUs

Dans le cadre de l'IPMS, deux types d'UAPDUs sont définis: les **IP-messages (IM-UAPDUs)** et les **IPM-status-reports (SR-UAPDUs)**.

A) Les IP-messages (IM-UAPDUs)

Un **IP-message** est un UAPDU échangé entre IPM UAs et contenant l'information que l'utilisateur désire communiquer. Il comprend:

- L'**en-tête** ("**heading**") : les informations diverses au sujet du message à communiquer. Celles-ci comprennent notamment:
  - un numéro identifiant l'IP-message;
  - l'O/R name de l'originator et du (des) recipient(s) (classés en primary recipients, copy recipients et blind copy recipients);
  - le sujet du message;
  - l'importance du message;
  - ...

  La liste complète des informations comprises dans l'en-tête se trouve dans [X420], section 3.2.1.

- Le **corps** ("**body**") : le message proprement dit à communiquer. Le corps d'un IP-message peut comprendre plusieurs parties (appelées chacune "**body part**"). Chacune de ces parties contient des données d'un seul type. La liste complète et précise des types de données possibles se trouve dans [X420], section 3.2.2. On y trouve par exemple: du texte, des fac-similés, de la voix, des messages encryptés (suites quelconques de bits) et même un type de donnée tout à fait quelconque qui doit être défini et reconnu au niveau national.

B) Les IPM-status-reports (SR-UAPDUs)

Un **IPM-status-report** est un UAPDU créé par un IPM UA et envoyé à un autre IPM UA, soit pour être utilisé directement par cet IPM UA, soit pour être communiqué à l'utilisateur de cet IPM UA (exemple: notification de réception). Les IPM-status-reports sont définis de façon plus précise dans [X420], section 3.3.

1.6.2. Les MPDUs

Deux types de MPDUs sont définis: les **User-MPDUs (UMPDUs)** et les **Service-MPDUs (SMPDUs)**. Les User-MPDUs contiennent les messages à communiquer, les Service-MPDUs contiennent des renseignements au sujet de ces messages.

A) Les User-MPDUs

Un **User-MPDU** est un MPDU contenant l'information à échanger entre MTAs. Il comprend:

- L'**enveloppe** ("**envelope**") : un certain nombre d'informations relatives au message à communiquer. Celles-ci comprennent notamment:
  - un numéro identifiant le MPDU;
  - l'O/R name de l'originator et du (des) recipient(s) (classés en primary recipients, copy recipients et blind copy recipients);

- le type du contenu du message;
- des informations concernant les MDs par où est passé le message et les décisions prises par ces MDs ("Trace informations")[1];
- ...

La liste complète des informations comprises dans l'enveloppe se trouve dans [X411], section 3.4.2.1. Remarquons que cette enveloppe est construite lorsque le message est soumis au MTA; elle peut être modifiée au fur et à mesure que le message parcourt le MTS (exemple: les "Trace informations" doivent être modifiées). Remarquons également que l'enveloppe comprend le nom de l'expéditeur et du destinataire, tout comme l'enveloppe d'un message postal (d'où l'appellation "enveloppe").

- Le contenu ("**content**") : le message proprement dit à communiquer entre MTAs. Ce message peut être tout à fait quelconque (voir [X411], section 3.4.2.2). Il s'agit en fait d'un UAPDU passé par l'UA au MTA (Dans notre cas (IPMS): IP-message ou IPM-status-report). Rappelons que le contenu d'un User-MPDU ne peut jamais être modifié par un MTA[2].

B) Les Service-MPDUs

Les **Service-MPDUs** contiennent des informations au sujet des messages échangés entre MTAs. Deux types de Service-MPDUs sont définis: les **Delivery-Report-MPDUs** et les **Probe-MPDUs.**

Les **Delivery-Report-MPDUs** contiennent des informations au sujet de la livraison ou de la non-livraison d'un message. Ils sont destinés à l'originator du message.

Les **Probe-MPDUs** contiennent une requête de vérification qu'un UA existe et peut être atteint (c.-à-d. que, si on lui envoyait un message, il lui serait livré). Ils sont utilisés par la MTL pour offrir le service "probe message" (voir 1.5.1.).

Les Delivery-Report-MPDUs sont constitués d'une enveloppe et d'une partie contenant l'information relative à la (non)-livraison. Les Probe-MPDUs ne contiennent qu'une enveloppe. L'enveloppe des Delivery-Report-MPDUs et celle des Probe-MPDUs est similaire à celle des User-MPDUs. Le contenu exact d'un Delivery-Report-MPDU est donné dans [X411], section 3.4.3.; celui d'un Probe-MPDU est donné dans [X411], section 3.4.4.

1.6.3. Le contenu global d'un User-MPDU contenant un IM-UAPDU

La figure 1.6 représente les principales parties d'un User-MPDU contenant un IM-UAPDU (c.-à-d. d'un message utilisateur enveloppé dans un UAPDU et dans un MPDU).

1. Ces informations permettent par exemple de vérifier que le message ne boucle pas.
2. Sauf quand un UA demande une conversion de données.

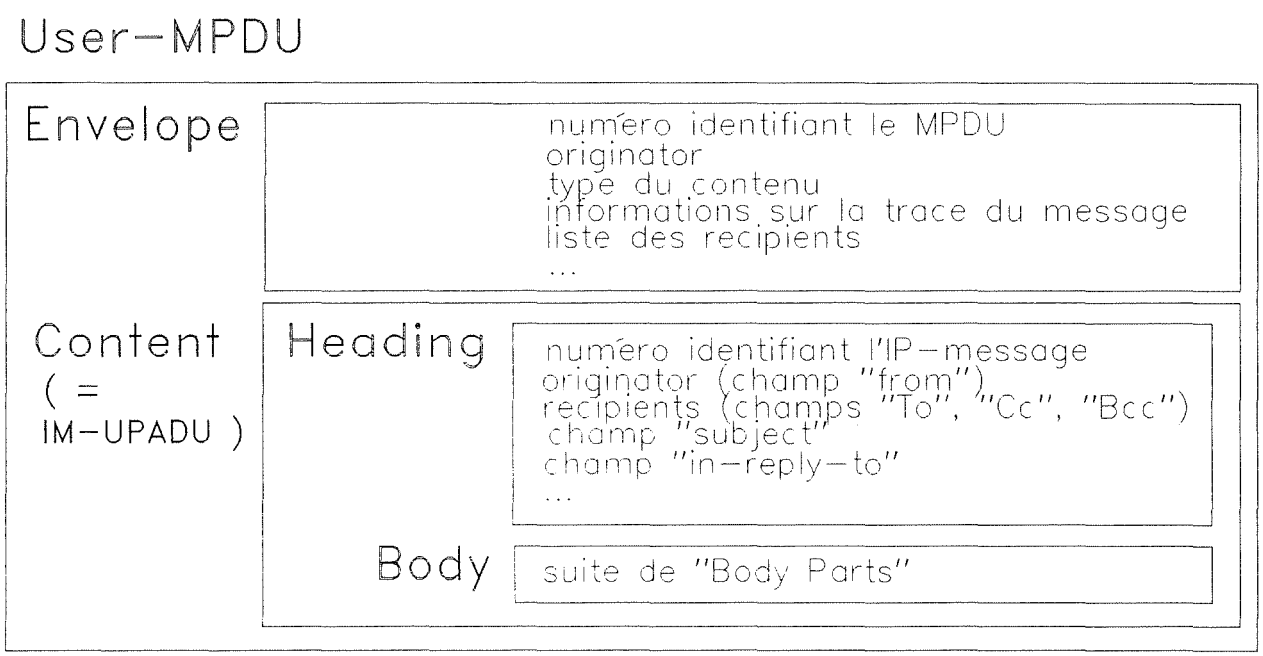


Figure 1.6 : User-MPDU contenant un IM-UAPDU

1.7. Le contenu des MTAEs

La section 1.4. décrit la découpe de la couche 7 en deux sous-couches: l'UAL et la MTL. Nous allons voir dans cette section que chaque MTAE se décompose à son tour en plusieurs parties.

Nous supposerons dans un premier temps que les MTAEs ne doivent communiquer qu'avec d'autres MTAEs (protocole P1) et ne doivent donc pas communiquer avec des SDEs (protocole P3). Les modifications apportées par l'introduction de ces derniers seront décrites dans la section 1.8..

1.7.1. La structure générale d'une MTAE

Une MTAE comprend trois parties: le **Message Dispatcher (MD[1])**, l'**Association Manager (AM)** et le **Reliable Transfer Service (RTS)**. Les rôles de chacune de ces parties sont décrits ci-dessous.

1.7.2. Le Message Dispatcher (MD)

Le **Message Dispatcher** est la partie du MTA qui a pour but d'exécuter les actions prévues par le protocole P1 lorsque le MTA reçoit un MPDU d'un autre MTA ou lorsqu'un de ses UAs lui soumet un message. Il réalise donc toutes les fonctions actives propres aux MTAs, sauf celles relatives au transfert proprement dit de messages de et vers d'autres MTAs, pour lesquelles il utilise les services de l'AM et du RTS.

Donc, si un MTA reçoit un message d'un de ses UAs et que ce message a pour destinataire un UA relié au même MTA, alors seule la partie Message Dispatcher du MTA est utilisée.

Les actions effectuées par le Message Dispatcher sont décrites dans [X411], section 3.3.

1. A ne pas confondre avec l'abréviation MD = Management Domain !

1.7.3. L'Association Manager (AM)

L'**Association Manager** est la partie du MTA qui a pour but d'établir, de contrôler et de fermer les associations fournies par le RTS.

De façon plus précise, c'est l'Association Manager qui:

- décide quand et quelle(s) association(s) ouvrir et fermer;
- assigne une priorité aux différents MPDUs à envoyer;
- choisit quelles associations utiliser pour le transfert de MPDUs (en fonction de leur priorité);
- gère qui possède le "tour", c.-à-d. qui peut envoyer ses APDUs (dans le cas d'associations permettant la communication alternativement dans les deux sens)[1].

L'Association Manager est décrit dans [X411], section 3.5.

1.7.4. Le Reliable Transfer Service (RTS)

Le **Reliable Transfer Service (RTS)** est la partie du MTA qui a pour but de fournir des associations permettant le transfert fiable de MPDUs avec d'autres MTAs. Le RTS est décrit dans [X410], section 3.

Remarques:

Le RTS constitue la couche inférieure de la couche 7. Il utilise les services offerts par la couche 5 (session), qui lui sont fournis via la couche 6 (présentation). La partie de la couche 7 utilisant le RTS est appelée **RTS-user**. La façon dont le RTS utilise les services des couches 6 et 5 est décrite dans [X410], section 4.

1.8. Le protocole P3 et les SDEs

1.8.1. Introduction

Nous avons vu que, lorsque l'UAE est éloignée de sa MTAE, elle communique avec cette UAE par l'intermédiaire d'une SDE (voir 1.4.). La SDE est donc une entité intermédiaire permettant à une UAE d'avoir accès aux services de la MTL, comme si l'UAE était reliée directement à une MTAE.

Le protocole respecté entre SDE et MTAE est appelé **protocole P3**. Il permet à chacune de ces deux entités de commander des opérations à distance sur l'autre entité ("**remote operations**"). L'entité demanderesse envoie un message à l'entité exécutrice comprenant la description de l'opération à effectuer et ses paramètres. L'entité exécutrice exécute alors l'opération et retourne à l'entité demanderesse un message comprenant les résultats de l'opération (si celle-ci a réussi) ou un message d'erreur (si l'opération a échoué).

Les unités de données échangées entre MTAE et SDE (selon le protocole P3) sont appelées **OPDUs (Operation Protocol Data Units)**. Elles sont classées en **Invoke OPDU** (requiert l'exécution d'une opération), **ReturnResult OPDU** (signale qu'une opération a été correctement effectuée et retourne les résultats de l'opération) et **ReturnError OPDU** (signale qu'une opération a échoué).

---

1. Une association peut être de deux types:
- **monologue**: les APDUs sont transférés dans une seule direction;
- **two-way alternate** (alternativement dans les deux sens): l'entité association ayant le droit de transférer des APDUs est celle possédant le "tour"; le "tour" peut être passé d'une entité à l'autre.

Par exemple, lorsque l'UAE éloignée désire soumettre un message au MTS, elle envoie sa requête à la SDE. Celle-ci enverra alors un "Invoke OPDU" à la MTAE comprenant la description de l'opération à effectuer (la soumission d'un message) et ses paramètres (le message proprement dit: enveloppe et contenu). Si tout se passe bien, la MTAE renverra alors un "ReturnResult OPDU" à la SDE pour signaler que l'opération a été correctement effectuée.

Le protocole P3 est décrit de façon précise dans [X411], section 4. La syntaxe générale des OPDUs est décrite dans [X410], sections 2.2. et 2.3.[1].

1.8.2. Opérations possibles

Rappelons qu'une SDE doit permettre à une UAE d'avoir accès au MTS comme si l'UAE était reliée directement à la MTAE. Par conséquent, une SDE doit offrir les mêmes services qu'une MTAE (ceux-ci sont résumés à la section 1.9.1.).

Chacun de ces services[2] est donc associé à une opération "remote" à effectuer par le MTAE ou par la SDAE (et donc à un OPDU). Par exemple, au service de soumission de message correspond une opération "remote" que la SDE ordonne à la MTAE. De même, au service de livraison de message correspond une opération "remote" que la MTAE ordonne à la SDE.

La liste des opérations possibles (donc des OPDUs possibles) et des services offerts par ces opérations est donnée dans le tableau 1.1.

Opérations	Primitives de service associées
Register	REGISTER.Req/Conf
Control	CONTROL.Req/Conf/Resp/Indic
Change-Password	CHANGE_PASSWORD.Req/Conf/Ind
Submit	SUBMIT.Req/Conf
Probe	PROBE.Req/Conf
Cancel	CANCEL.Req/Conf
Deliver	DELIVER.Ind
Notify	NOTIFY.Ind

Tableau 1.1 : opérations possibles en P3 et primitives de service associées

Remarquons qu'aucune opération ne correspond aux primitives de service LOGON et LOGOFF. Celles-ci sont en faits associées directement aux primitives de service OPEN et CLOSE offertes par le RTS (voir section 1.9.2.).

1.8.3. Implications sur la structure des MTAEs et des SDEs

Chaque MTAE comprend, en plus du MD, de l'AM et du RTS, une partie supplémentaire chargée de la gestion des Remote Operations: elle est appelée **Remote Operation Service.**

1. Les OPDUs peuvent être utilisés par des protocoles autres que P3, mais basés eux aussi sur le concept de Remote Operations.
2. Sauf LOGON et LOGOFF: voir plus loin.

Une SDE, quant à elle, est une mini-MTAE comprenant toutes les parties classiques de la MTAE, sauf le MD (celui-ci n'est pas nécessaire à la SDE puisque les opérations normalement effectuées par le MD ne sont pas effectuées dans la SDE mais renvoyées à une autre MTAE).

En résumé, une MTAE comprend: MD, AM, RTS, ROS. Une SDE comprend: AM, RTS, ROS.

1.9. Les primitives de service

Cette section reprend un résumé des primitives de service offertes d'une part par le MTA à l'UA, d'autre part par le RTS au RTS-User.

Les primitives de service sont groupées par fonctionnalités, en indiquant la chronologie des événements: **Req** = Request, **Ind** = Indication, **Resp** = Response, **Conf** = Confirm.

1.9.1. Les primitives de service offertes par le MTA à l'UA

Les primitives de service offertes par le MTA à l'UA sont données dans le tableau 1.2 (voir [X411], section 2.2 pour plus de précisions).

Nom	Initiateur	Evénements	Fonctionnalité
LOGON	UA	Req/Conf	l'UA veut initialiser l'interaction avec le MTA
LOGON	MTA	Ind/Resp	le MTA veut initialiser l'interaction avec l'UA
LOGOFF	UA	Req/Conf	l'UA veut terminer l'interaction avec le MTA
REGISTER	UA	Req/Conf	l'UA veut modifier certains de ses paramètres, utilisés par le MTA
CONTROL	UA	Req/Conf	l'UA veut modifier les restrictions concernant le contrôle des messages que la MTL peut lui envoyer
CONTROL	MTA	Ind/Resp	le MTA indique à l'UA quels messages de l'UA seront acceptés par la MTL
SUBMIT	UA	Req/Conf	l'UA demande l'envoi d'un message
PROBE	UA	Req/Conf	l'UA demande l'envoi d'une sonde (probe message)
DELIVER	MTA	Ind	le MTA livre un message à l'UA
NOTIFY	MTA	Ind	le MTA livre à l'UA un accusé de (non)-livraison
CANCEL	UA	Req/Conf	l'UA demande l'annulation de la livraison d'un message précédemment posté avec l'option "Deferred Delivery" (livraison après telle date/heure)
CHANGE-PASSWORD	UA	Req/Conf	l'UA indique au MTA le nouveau mot de passe de l'UA
CHANGE-PASSWORD	MTA	Ind	le MTA indique au MTA le nouveau mot de passe du MTA

Tableau 1.2 : primitives de service offertes par le MTA à l'UA

1.9.2. Les primitives de service offertes par le RTS au RTS-User

Les primitives de service offertes par le RTS au RTS-User sont données dans le tableau 1.3 (voir [X410], section 3.2 pour plus de précisions).

Nom	Initiateur	Evénements	Fonctionnalité
OPEN	RTS-User	Req/Ind/Resp/Conf	Le RTS-User veut établir une nouvelle association avec un autre RTS-User
CLOSE	RTS-User	Req/Ind/Resp/Conf	Le RTS-User ayant l'initiative veut fermer une association
TURN-PLEASE	RTS-User	Req/Ind	Un RTS-User n'ayant pas l'initiative demande pour l'avoir
TURN-GIVE	RTS-User	Req/Ind	Le RTS-User ayant l'initiative la cède à l'autre RTS-User
TRANSFER	RTS-User	Req/Ind	Le RTS-User demande le transfert d'un APDU (le transfert doit être effectué endéans un laps de temps donné)
EXCEPTION	RTS	Ind	Le RTS signale au RTS-User que le transfert d'un APDU n'a pas pu se faire comme demandé

Tableau 1.3 : primitives de service offertes par le RTS au RTS-User

1.10. La nouvelle recommandation X.400 1988

Nous allons ici résumer les principales modifications apportées dans la version 1988 de X.400 par rapport à la version originale de 1984. Cette section est essentiellement basée sur les articles [VF] et [JC].

1.10.1. Les services abstraits

La version 1988 précise les services offerts par les différentes sous-couches du niveau 7, grâce à l'introduction du concept de **service abstrait** (dans la nouvelle recommandation **X.407**). Les services abstraits sont définis de façon beaucoup plus précise que ne l'étaient les services dans la version 1984 de X.400: ils sont définis formellement en ASN.1[1]. De plus, un service abstrait global peut être décomposé en plusieurs parties, chacune réalisant un "sous-service" abstrait. Chacune de ces parties peut à son tour être redécomposée en d'autres parties plus petites...

Lorsque des entités fonctionnelles résident dans le même système, l'interface entre elles dépend de l'implémentation; elle doit simplement respecter les services abstraits définis par la norme. En revanche, lorsque des entités fonctionnelles sont séparées, la définition formelle du service abstrait entre ces deux entités permet de définir très facilement le protocole à utiliser pour assurer ce service.

1.10.2. Le Message Store et les Access Units

Deux nouvelles entités fonctionnelles sont introduites dans X.400 1988: le **Message Store** ("réserve de messages") et les **Access Units** ("unités d'accès").

1. ASN.1 est le nouveau nom donné à la syntaxe X.409.

A) Le Message Store

La réserve de messages (Message Store, défini dans la nouvelle recommandation **X.413**) permet à un UA isolé d'accéder à son MTA, mais avec beaucoup plus de contrôle sur les interactions UA <-> MTA que ne le permet le protocole P3.

Généralement, la "réserve de messages" sera située sur le même système que le MTA. Le protocole entre "réserve de messages" et UA est appelé **protocole P7**.

Exemples:

Avec P3, l'utilisateur est forcé d'accepter les messages lorsqu'il a ouvert une connexion avec le MTA; avec P7 l'utilisateur peut contrôler lui-même la livraison des messages qui lui sont destinés. Ainsi, un des inconvénients de P3 est que, si un utilisateur ne sait pas accéder à son MTA pendant un certain temps[1], certains messages qui lui sont destinés risquent d'arriver à expiration et de ne jamais être livrés. Par contre, en P7, l'utilisateur peut demander que ses messages soient placés temporairement dans sa "réserve de messages", qu'il lira beaucoup plus tard.

La "réserve de messages" est une véritable base de données des messages et des rapports de (non)-livraison. Des données peuvent en être extraites par l'utilisateur, d'après des critères de sélection très souples.

B) Les Access Units

Deux types d'"unités d'accès" (Access Units) sont définis en 1988: les unités d'accès pour services télématiques[2] (**Access Units for Telematic Services**), et les unités d'accès pour livraison physique[3] (**Physical Delivery Access Unit**). Des extensions au protocole P1 ont été introduites pour tenir compte de ces nouvelles entités fonctionnelles.

1.10.3. Le mécanisme d'extension

La version 1988 de X.400 introduit un mécanisme permettant de définir à volonté des extensions aux protocoles de base.

Des extensions peuvent donc être ajoutées aux protocoles P1 et P2. Cependant, les extensions particulières au protocole P1 comprennent chacune un drapeau indiquant si elles sont critiques ou non. Si un message P1 comprend des extensions non critiques, n'importe quel MTA peut le "relayer". En revanche, si le message comprend des extensions critiques, chaque MTA sur la route du message doit comprendre la sémantique de ces extensions (sinon le message ne sera pas livré, et un rapport de non-livraison sera renvoyé à l'expéditeur).

La plupart des extensions apportées aux protocoles P1 et P2 dans la version 1988 de X.400 (voir sections 1.10.4. et 1.10.5.) ont été définies par ce mécanisme.

1.10.4. Les extensions au protocole P1

La version 1988 de X.400 apporte les extensions suivantes au protocole P1.

---

1. par exemple dans le cas d'un UA situé sur un PC portable
2. par exemple: télex et télétex
3. elles permettent d'utiliser des adresses de recipients qui recevront leurs messages sur un support physique (par exemple du papier)

A) Les listes de distribution

Une **liste de distribution** (**Distribution List**) est un boîte aux lettres spéciale où des utilisateurs peuvent envoyer des messages. Tout le courrier arrivant dans cette boîte aux lettres sera redistribué à une liste précise d'utilisateurs. L'adresse de cette boîte aux lettres est appelée **Distribution List Expansion Point**. On dit que le (destinataire du) message est "**développé**" ("**expanded**") par la liste de distribution.

Un même message peut être développé consécutivement par plusieurs listes de distribution. X.400 1988 prévoit qu'un destinataire final doit savoir pourquoi il reçoit le message, c.-à-d. non seulement quel est l'expéditeur initial mais également quelles sont les listes de distribution par où est passé le message et quels sont les développements effectués par ces listes de distribution.

De plus, X.400 1988 prévoit que les rapports de (non)-livraison puissent parcourir en arrière toute la chaîne des listes de distribution rencontrées par le message original, afin que chaque liste de distribution ajoute ses propres informations dans le rapport. Ceci nécessite une modification de tous les MTAs puisque, dans la version 1984 de X.400, un rapport est toujours renvoyé directement à l'expéditeur de départ.

Enfin, il est possible à l'utilisateur de signaler qu'il refuse tout développement d'un message qu'il envoie (par exemple pour des raisons de confidentialité).

B) Les O/R names, les O/R address et l'utilisation du Directory

Un **O/R name (1984)** a été renommé **O/R address (1988)**; le nouveau terme **O/R name (1988)** consiste en, soit une **O/R address (1988)**, soit un **Directory name (1988)**. Un **Directory name (1988)** est simplement une entrée possible du Directory, qui peut être convertie grâce à celui-ci en **O/R address (1988)**.

Le Directory peut être utilisé par les utilisateurs pour chercher eux-mêmes certains renseignements. Cette utilisation n'est pas normalisée. Le Directory peut cependant aussi être utilisé par un MTA qui désire convertir un O/R name (qui n'est pas déjà une O/R address) en une O/R address. Cette seconde utilisation a été normalisée dans X.400 1988.

De nouveaux attributs ont également été introduits dans les O/R address.

C) La redirection de messages

Un expéditeur d'un message peut spécifier, pour chaque destinataire, un second destinataire à qui le message sera envoyé s'il ne peut être livré au premier destinataire.

De plus, un utilisateur peut demander que le MTA redirige tous les messages à destination de son UA vers un autre UA (c'est l'équivalent au niveau P1 du service "auto-forward" au niveau P2).

D) La sécurité

Plusieurs extensions ont été apportées dans la version 1988 de X.400, relatives à la sécurité du système de messagerie électronique, par exemple l'authentification de l'origine d'un message, d'un rapport ou d'une sonde ("probe"), la preuve de soumission ou de livraison d'un message.

1.10.5. Les extensions au protocole P2

Deux nouveaux champs ont été introduits dans la partie "heading" d'un message P2: **Language Indication** et **Incomplete Copy**.

De plus, certaines parties du corps du message (body part) peuvent avoir une syntaxe et une sémantique définie par l'utilisateur lui-même. Cette syntaxe et cette sémantique peuvent être envoyées dans le message , et le destinataire peut donc les lire pour savoir comment décoder et interpréter le message. Ce mécanisme très puissant permet donc de transférer des messages dont le format est défini dans d'autres standards (par exemple ISO 8613 ODA : Office Document Architecture, ou EDI : Electronic Data Interchange).

1.10.6. La norme X.400 1988 versus la norme ISO 10021

La version 1984 de X.400 a été élaborée par le CCITT. Par après, le CCITT a collaboré avec ISO pour l'étude des améliorations à apporter à X.400 1984. Malheureusement, le CCITT et ISO ne sont jamais parvenus à se mettre d'accord sur certains détails. Ceci a débouché sur l'élaboration de deux normes: la norme CCITT 1988 et la norme ISO 10021.

Bien que les différences soient mineures, il est probable et souhaitable que dans l'avenir, le CCITT et l'ISO se mettront d'accord sur une seule normalisation du système de messagerie électronique.

Une des différences est que le CCITT continue à ne considérer que l'interconnexion d'ADMDs entre eux et de PRMDs avec des ADMDs. En revanche, l'ISO autorise la connexion directe d'un PRMD avec un autre PRMD, et ceci, même via un troisième PRMD ou même en traversant les frontières d'un pays.

Chapitre 2 : L'architecture de EAN

Ce chapitre décrit l'architecture du logiciel EAN. Il est essentiellement basé sur [EAN1], [EAN2], [EANMAN], [EANUBC] et [EANDFN].

2.1. L'architecture générale

EAN est un système de messagerie basé sur l'avis X.400 (1984). Il comprend 3 grandes parties:

- le **User Service (US)**;
- le **Message Transfer Service (MTS)**;
- le **Directory Service (DS)**.

Chacune de ces parties réalise les principales fonctionnalités X.400 offertes respectivement par les UAs, les MTAs et le service d'annuaire (directory).

Avant d'étudier plus en détail chacune de ces 3 parties, nous allons d'abord voir quelles sont les différentes versions de EAN disponibles (section 2.2.) et comment référencer un utilisateur du réseau (section 2.3.).

2.2. Les différentes versions de EAN

Il existe actuellement (en 1988) essentiellement trois versions de EAN.

La première version a été développée entre 1981 et 1983 par l'Université de British Columbia (UBC), au Canada. Cette version utilise le format d'adressage RFC822 décrit dans la section 2.3.3.. Nous désignerons cette version par "**EAN UBC Version 1**" ou par "**EAN UBC1**".

Malheureusement, cette première version ne respecte pas l'adressage imposé par la norme X.400 (adressage par attributs standards). C'est pourquoi UBC a retravaillé sa première version de EAN pour y ajouter la possibilité d'utiliser des adresses pouvant comprendre à la fois une partie RFC822 (pour la compatibilité avec la première version) et des attributs standards (pour la conformité à X.400). Ce travail a débouché en 1987 sur une deuxième version de EAN, que nous désignerons par "**EAN UBC Version 2**" ou par "**EAN UBC2**".

Cette deuxième version a cependant l'inconvénient de mélanger dans la même adresse l'adressage RFC822 et les attributs standards. Ceci augmente considérablement la complexité de l'algorithme de routage. Une table de routage supplémentaire (la "**Route-Rule Table**") doit être construite par le gestionnaire de système, décrivant de façon précise un certain nombre de règles, les conditions d'applications de celles-ci, et leur enchaînement (voir [EANUBC], pages 9 à 13).

Une troisième version a été élaborée par le groupe FOKUS (GMD Berlin, Europe), sous la direction du Deutsches ForschungsNetz (groupe allemand de recherche sur les réseaux). Il s'agit de la version "**EAN DFN**". Tout comme la version UBC2, la version DFN permet à l'utilisateur d'employer des adresses RFC822 ou des adresses comprenant des attributs standards. Cependant, les deux formats ne sont jamais mélangés dans la même adresse: l'utilisateur spécifie une adresse soit en RFC822, soit par attributs standards. De plus, l'algorithme de routage est fort simplifié par rapport à la version UBC2, puisqu'il ne nécessite qu'une bonne description du MHS en terme d'attributs C,A,P,O, et OU (il n'y a plus de Route-Rule Table): voir section 3.2..

Nous n'étudierons dans les chapitres 2 et 3 que les versions UBC1 et DFN.

2.3. L'adressage

Un utilisateur est identifié soit par son **NUN (Network User Name)**, soit par son **NUA (Network User Address)**.

2.3.1. Le NUN

Le NUN est composé d'un simple nom d'utilisateur suivi d'un ':' et du nom de l'organisation à laquelle l'utilisateur appartient (exemple: "Robin Smith : University of British Columbia")[1].

Malheureusement, EAN ne sait pas identifier directement un utilisateur par son NUN, mais uniquement par son NUA. EAN doit donc d'abord convertir le NUN en NUA. C'est le principal rôle du Directory Service (voir section 2.8.). Nous ne parlerons donc plus ici que de NUAs.

2.3.2. Le NUA

Le NUA identifie un utilisateur en **localisant où** se trouve son UA. Les versions UBC1 et DFN de EAN utilisent une technique d'adressage différente, donc des NUAs ayant une forme différente. La forme des NUAs utilisés et leur représentation dans EAN UBC1 et dans EAN DFN sont décrites respectivement dans la section 2.3.3. et dans la section 2.3.4..

Remarquons cependant que ces versions de EAN représentent toutes deux un NUA par un O/R Name. La différence réside dans le fait que la version UBC1 n'utilise que les DDAs et l'attribut PRMD (voir section 2.3.3.), tandis que la version DFN utilise des combinaisons possibles des attributs standards prévus dans la norme X.400 (voir section 2.3.4.).

Remarquons également que, dans les deux versions, le NUA peut être accompagné d'un commentaire expliquant de façon plus claire qui est la personne désignée par le NUA. La syntaxe est la suivante : **commentaire < NUA >** .

Exemple: Xavier Rutten <S=xru;OU=ts;O=info;P=fundp;A=rtt;C=be[2]>

1. Dans la version UBC1 de EAN, l'organisation est simplement un sous-ensemble du domaine de l'utilisateur; dans la version DFN, l'organisation correspond généralement à l'attribut standard Organization.

2. Voir l'adressage DFN pour la signification de ce NUA.

Le commentaire est utilisé à la place du NUA lorsque l'on liste le résumé des messages dans sa boîte aux lettres. De plus, lorsque l'on envoie un message à un autre utilisateur, EAN ajoute automatiquement comme commentaire dans le champ "From" le nom de l'utilisateur expéditeur (tel qu'il est enregistré dans son profile[1]).

Enfin, chaque utilisateur peut également définir des synonymes servant d'abréviations pour les NUAs de ses correspondants. La syntaxe est la suivante:

set alias abréviation NUA
ou **set alias** abréviation commentaire < NUA >.

Exemple:

la commande:

set alias xruj Xavier Rutten <S=xru;OU=ts;O=info;P=fundp;A=rtt;C=be>

définit "xruj" comme étant synonyme du NUA "S=xru;OU=ts;O=info;P=fundp;A=rtt;C=be", avec "Xavier Rutten" comme commentaire.

2.3.3. L'adressage UBC1 (représentation RFC822)

A) Forme d'un NUA

De façon générale, un NUA dans EAN UBC1 respecte la syntaxe suivante:

domain-specific-description.domain

où

domain est le domaine de l'utilisateur;
domain-specific-description est une chaîne de caractères identifiant l'utilisateur dans son domaine.

En particulier, si le NUA correspond à une adresse d'un site EAN X.400, **domain-specific-description** s'écrit:

mailbox@subdomain-list

où

mailbox est le nom de la boîte aux lettres de l'utilisateur;
subdomain-list est une liste ordonnée de sous-domaines imbriqués comprenant tous l'UA de l'utilisateur (le plus petit sous-domaine (contenu dans tous les autres) est le plus à gauche dans la liste, le plus grand sous-domaine (contenant tous les autres) est le plus à droite dans la liste)[2].

Un NUA EAN se représente donc comme suit:

mailbox@subdomain-list.domain

On appelle ce dernier type de représentation la **représentation RFC822.**

---

1. Le **profile** est un fichier texte associé à chaque utilisateur et contenant une série d'informations propres à l'utilisateur (nom, adresse, téléphone, abréviations pour les NUAs,...). Il est modifiable par l'utilisateur. Lorsque l'utilisateur commence une session UA, EAN lit automatiquement son fichier profile pour établir certaines options propres à l'utilisateur.

2. Cette liste doit comprendre au moins un sous-domaine.

Par exemple, dans le NUA "xru@ts.info.fundp":

- "xru@ts.info" identifie l'utilisateur dans son domaine "fundp";
- "xru" est le nom de la boîte aux lettres de l'utilisateur;
- "ts" et "info" sont les noms de deux sous-domaines contenant l'UA de l'utilisateur "xru" ("ts" étant inclus dans "info").

Autres exemples (FUNDP):

"bdt@ts.info.fundp"
"jva@rlab.info.fundp"
"xru@zeus.info.fundp"

Dans ces 3 exemples, "**fundp**" désigne le domaine (les Facultés Universitaires Notre-Dame de la Paix); "**info**" désigne un sous-domaine particulier (la faculté d'informatique) comprenant lui-même les trois sous-domaines "**ts**", "**rlab**" et "**zeus**" (qui correspondent respectivement à l'ordinateur **Junon**, **Alma** et **Zeus** de la faculté d'informatique); **bdt**, **jva**, **xru** désignent le nom de la boîte aux lettres de l'utilisateur.

B) Représentation interne d'un NUA

Un NUA "**domain-specific-description.domain**" est représenté de façon interne dans EAN par un O/R name standardisé (comme défini dans la section 1.2.4.). Le mapping entre le NUA et l'O/R name est le suivant:

- **domain** est représenté par l'attribut standard PRMD;
- **domain-specific-description** est représenté par une suite de DDAs.

En particulier, un NUA "mailbox@subdomain-list.domain" est représenté comme suit:

- **domain** est représenté par l'attribut standard PRMD;
- **mailbox** est représenté par un premier DDA;
- **subdomain-list** est représenté par un second DDA.

Exemple:

Le NUA "xru@ts.info.fundp" est représenté par l'O/R Name suivant:
"DD.=xru;DD.=ts.info;PRMD=fundp".

Remarquons que les attributs de base "ADMD" et "Country" de l'O/R Name ne sont pas utilisés. Par conséquent, les PRMDs des autres ADMDs et des autres pays seront assimilés à de simples PRMDs particuliers, sans que leur ADMD ou que leur pays soit mentionné.

2.3.4. L'adressage DFN (représentation par attributs standards X.400)

A) Introduction

Le NUA de l'utilisateur de EAN DFN est un O/R Name X.400. Cette représentation est expliquée ci-dessous.

Remarques:

1°) La version DFN de EAN supporte également l'adressage RFC822. De façon plus précise:
- l'utilisateur peut spécifier ses adresses soit en X.400, soit en RFC822;
- EAN représente de façon interne les adresses selon la représentation X.400[1].

2°) Donc, EAN comprend une fonction de conversion entre représentation X.400 et représentation RFC822, et vice-versa. Ces conversions sont exposées à la section 2.3.5..

B) Représentation par attributs standards

Un O/R Name X.400 respecte la syntaxe suivante:

```
O/R Name X.400 : = = list-of-elements
list-of-elements : = = element ¦ list-of-elements
element  : = =      attribute-keyword = { value }
          ¦         DD.type = { value }
attribute-keyword : = =
              Country              ; Nom du pays
          ¦   Admd ¦ ADMInd        ; Nom de l'ADMD
          ¦   Prmd ¦ PRIvate       ; Nom du PRMD
          ¦   X121address          ; Adresse X.121
          ¦   T-ID                 ; IDentificateur de terminal
          ¦   Organization         ; Nom de l'Organisation
          ¦   OUnit ¦ ORGUnit      ; Nom de l'Unité Organisationnelle
          ¦   UA-ID                ; Identificateur d'un UA unique
          ¦   GQ                   ; Nom personnel : qualificatif de génération
          ¦   Surname              ; Nom personnel : nom de famille
          ¦   Initials             ; Nom personnel : initiales
          ¦   Givenname            ; Nom personnel : prénom
```

Remarques:

- les lettres majuscules soulignées peuvent être utilisées comme abréviations du terme correspondant;
- les attributs X.121, T-ID, UA-ID sont mentionnés dans la documentation de EAN; ils ne sont cependant pas utilisables sur les versions de EAN installées aux FUNDP;

1. Sauf dans les tables du MTA où EAN utilise encore la représentation RFC822.

- si la valeur d'un même attribut est spécifiée plusieurs fois, seule la dernière valeur est utilisée (sauf pour les attributs OU et DD.type, dont toutes les valeurs spécifiées sont utilisées).

Une des formes les plus rencontrées est le cas particulier suivant:

S = <Surname>;
OU = <Smallest Organization Unit>;
OU = <Larger Organization Unit>;
...
OU = <Largest Organization Unit>;
O = <Organization>;
P = <Prmd>;
A = <Admd>;
C = <Country>

2.3.5. Le mapping adresse X.400 <--> adresse RFC822

A) Introduction

Le mapping entre une adresse X.400 et une adresse RFC822 est une fonction utilisée par la version DFN de EAN. La conversion entre les deux types de représentation respecte les spécifications **RFC987** et est réalisée dans EAN par un module appelé "**RFC987 gateway**".

Toutes les adresses RFC822 peuvent être converties en adresses X.400. Nous noterons ici la forme générale d'une adresse RFC822 comme suit:

localpart@subd(n).subd(n-1). ... .subd(2).subd(1).domain

où $n \geq 1$.

En revanche, seules les adresses X.400 ne comprenant que des attributs I,G,S,OU,O,P,A et C peuvent être converties en adresses RFC822. Leur forme générale est donc la suivante:

G = ...;I = ...;S = ...;OU = ...;OU = ...; ... ;OU = ...;O = ...;P = ...;A = ...;C = ...

où G, I et S forment le **Personal Name** (S est un attribut obligatoire).

Normalement, un mapping par défaut est utilisé. Cependant, l'administrateur du système peut spécifier un certain nombre de règles particulières à appliquer dans un certain nombre de cas: si une règle particulière est applicable, alors on l'applique, sinon, on applique le mapping par défaut.

Le mapping par défaut est décrit dans la section 2.3.5.B., le mapping par règles particulières (appelées "**tables de conversion**") est décrit dans la section 2.3.5.C.. Toutes ces considérations au sujet de conversions d'adresse sont extraites de [EANDFN], pages 6 et 20 à 24.

B) Mapping par défaut

Par défaut, le mapping suivant est appliqué:

X.400	RFC822
Personal Name	localpart
OU	subd(n)
OU	subd(n-1)
...	...
OU	subd(4)
O	subd(3)
P	subd(2)
A	subd(1)
C	domain

avec les conventions suivantes:

- mapping X.400 -> RFC822:
  si un attribut manque en X.400, on n'écrit pas de vide à sa place (il n'y a pas deux points consécutifs parce que l'attribut est manquant)

- mapping RFC822 -> X.400:
  si n < 4, on utilise d'abord les attributs X.400 les plus à droite (les plus généraux), c.-à-d.:
  - si n = 1: localpart@subd(1).domain
    -> A = subd(1);C = domain
  - si n = 2: localpart@subd(2).subd(1).domain
    -> P = subd(2);A = subd(1);C = domain
  - si n = 3: localpart@subd(3).subd(2).subd(1).domain
    -> O = subd(3);P = subd(2);A = subd(1);C = domain

Remarques:

- le mapping Personal Name <-> localpart est expliqué ci-dessous;
- l'ordre des OUs en X.400 et des sous-domaines en RFC822 doit être respecté;
- le domaine RFC822 devient l'attribut C(ountry) en X.400; ce mapping est différent du mapping effectué par la première version UBC1 de EAN (le domaine devenait le PRMD, les attributs ADMD et Country n'étaient pas utilisés: voir section 2.3.3.B.);
- certaines adresses X.400 sont converties en les mêmes adresses RFC822 (voir 3e et 4e exemple ci-dessous).

Exemples:

	X.400	RFC822
1.	S = meier;OU = fokus;OU = berlin;P = gmd; A = dbp;C = de	meier@fokus.berlin.gmd.dbp.de
2.	S = postmaster;A = ptt;C = at	postmaster@ptt.at
3.	S = postmaster;OU = education;A = ptt;C = at	postmaster@education.ptt.at
4.	S = postmaster;P = education;A = ptt;C = at	postmaster@education.ptt.at

Conversion Personal Name (G,I,S en X.400) <-> localpart (RFC822):

L'attribut **I** peut comprendre au maximum trois lettres : I1, I2 et I3.

* Le mapping X.400 -> RFC822 est le suivant (voir exemples 1 à 7 ci-dessous):

localpart = G.I1.I2.I3.S

où les attributs manquants ne sont pas écrits (pas de "..").

* Le mapping RFC822 -> X.400 est légèrement plus compliqué: il résulte de l'application de l'algorithme suivant:

1°) L'algorithme examine **localpart** de gauche à droite, en cherchant un point. S'il n'en trouve pas, alors **S : = localpart** et l'algorithme se termine (voir exemple 1). Si l'algorithme en trouve un, alors il exécute l'étape 2°).

2°) Si la chaîne **ch** précédant le point trouvé ne comprend qu'une et une seule lettre, alors **I : = ch** (voir exemple 3), sinon **G: = ch** (voir exemples 2 et 4 à 7). Dans les deux cas, aller en 3°).

3°) L'algorithme cherche le point suivant. S'il n'y en a pas, alors **S : = la chaîne restant (à droite du 1er point)** et l'algorithme se termine: voir exemples 2 et 3. Sinon, aller en 4°).

4°) Si la chaîne trouvée avant le nouveau point et après l'ancien point a une longueur différente de 1, alors **S : = la chaîne restant** et l'algorithme se termine: voir exemple 7. Sinon, l'algorithme ajoute à **I** l'unique lettre constituant la chaîne et l'algorithme reprend au point 3°): voir exemples 4 à 6.

Exemples:

	Personal Name X.400	localpart RFC822
1.	S = Rose	Rose
2.	G = Marshall;S = Rose	Marshall.Rose
3.	I = M;S = Rose	M.Rose
4.	G = Marshall;I = M;S = Rose	Marshall.M.Rose
5.	G = Marshall;I = MT;S = Rose	Marshall.M.T.Rose
6.	G = Marshall;I = MT;S = Rose.Bs	Marshall.M.T.Rose.Bs
7.	G = Marshall;S = MT.Rose.Bs	Marshall.MT.Rose.Bs

C) Mapping guidé par les tables de conversion

i) Introduction

Dans certains cas, la conversion effectuée par le mapping par défaut est erronée.

Exemple 1:

L'adresse RFC822 "**trickle@banufs11.bitnet**" est convertie en "**S = trickle;A = banufs11;C = bitnet**". Or "bitnet" n'est pas un nom de pays et "banufs11" n'est pas un nom d'ADMD. L'adresse X.400 correspondante devrait plutôt être "**S = trickle;O = banufs11;P = bitnet;A = rtt;C = be**" (en supposant que dans l'ADMD "A = rtt" du pays "C = be", il y ait une passerelle vers le réseau Bitnet assimilée à un PRMD "bitnet").

De plus, la véritable adresse X.400 "**S=trickle;O=banufs11;P=bitnet;A=rtt;C=be**" serait convertie en une mauvaise adresse RFC822: "**trickle@banufs11.bitnet.rtt.be**".

Exemple 2:

Le mapping RFC822 -> X.400 essaie d'abord d'utiliser les attributs X.400 les plus à droites (les plus généraux). Or certaines adresses ne comprennent par exemple pas d'attribut "O" mais bien un attribut "OU". C'est ainsi que EAN convertit l'adresse RFC822 "**grimm@darmstadt.gmd.dbp.de**" en l'adresse X.400 "**S=grimm;O=darmstadt;P=gmd;A=dbp;C=de**", alors que l'adresse X.400 correcte est "**S=grimm;OU=darmstadt;P=gmd;A=dbp;C=de**".

En conclusion, il s'avère que le mapping par défaut ne fonctionne pas toujours correctement et que l'utilisateur peut souhaiter vouloir imposer certaines règles particulières de conversion.

L'administrateur de EAN doit écrire ces règles particulières de conversion dans un fichier particulier ("**.rfc987.x**" pour les conversions X.400 -> RFC822, "**.rfc987.r**" pour les conversions RFC822 -> X.400). Chacun de ces fichiers comprend 2 colonnes, séparées par le caractère '#'. Chaque ligne correspond à une règle. La colonne de gauche de la ligne correspond à une condition d'application de la règle, la colonne de droite est la conversion à appliquer si la règle est appliquée.

ii) Conversion X.400 -> RFC822

Algorithme:

- chercher les lignes du fichier ".rfc987.x" dont la colonne de gauche décrit une partie de l'adresse X.400 à convertir;
- si aucune ligne ne convient, appliquer le mapping par défaut et terminer;
- si plusieurs lignes sont applicables, ne retenir que la ligne comprenant dans sa colonne de gauche le plus d'attributs X.400 d'"ordre supérieur" (d'après la relation d'ordre: C > A > P > O > OU > S);
- utiliser comme "partie supérieure" de l'adresse RFC822 la chaîne de caractères contenue dans la colonne de droite de l'unique ligne retenue; utiliser comme "partie inférieure"[1] la juxtaposition des éventuels attributs présents dans l'adresse X.400 et non présents dans la colonne de gauche de la ligne retenue;
- enfin, convertir le Personal Name en localpart comme vu dans le mapping par défaut.

Exemples:

Supposons que le fichier ".rfc987.x" contienne :

```
A=dbp;C=de                                          #
O=;A=dbp;C=de                                       # dbp.de
P=mpg;A=dbp;C=de                                    # mpg.dbp.de
P=fhg;A=dbp;C=de                                    # fhg.dbp.de
OU=rocquencourt;O=inria;P=aristote;A=ptt;C=fr       # multics.inria.fr
```

1°) Soit l'adresse X.400:
"S=frank;OU=pw;P=mpg;A=dbp;C=de"

1. localpart non compris

- les lignes applicables sont les lignes n° 1, 2 et 3;
- parmi les lignes n° 1, 2 et 3, la ligne contenant dans sa colonne de gauche le plus d'attributs X.400 d'"ordre supérieur" est la ligne n° 3 (la ligne 1 ne comprend que 2 attributs; les lignes 2 et 3 en comprennent chacune 3, mais les attributs de la ligne 3 sont d'"ordre plus grand" que ceux de la ligne 2);
- donc la ligne n° 3 est appliquée:
    - la "partie supérieure" de l'adresse RFC822 sera "mpg.dbp.de" (voir colonne de droite);
    - la "partie inférieure" de l'adresse RFC822 sera "pw";
    - le Personal Name "S=frank" est converti en localpart = "frank".

Donc, l'adresse RFC822 correspondante sera "frank@pw.mpg.dbp.de".

2°) Soit l'adresse X.400:
"S=tintin;OU=rocquencourt;O=inria;P=aristote;A=ptt;C=fr"

- seule la ligne n° 5 est applicable; elle est appliquée:
    - la "partie supérieure" de l'adresse RFC822 sera "multics.inria.fr" (voir colonne de droite);
    - la "partie inférieure" de l'adresse RFC822 sera "";
    - le Personal Name "S=tintin" est converti en localpart = "tintin".

Donc, l'adresse RFC822 correspondante sera "tintin@multics.inria.fr".

iii) Conversion RFC822 -> X.400

Algorithme:

- chercher les lignes du fichier ".rfc987.r" dont la colonne de gauche décrit complètement une partie à droite de l'adresse RFC822 à convertir;
- si aucune ligne ne convient, appliquer le mapping par défaut et terminer;
- si plusieurs lignes sont applicables, ne retenir que la ligne comprenant dans sa colonne de gauche le plus grand nombre de sous-chaînes de caractères (c.-à-d. le plus grand nombre de points);
- utiliser comme "partie supérieure" de l'adresse X.400 la chaîne de caractères contenue dans la colonne de droite de l'unique ligne retenue; pour obtenir la "partie inférieure"[1], distribuer les sous-chaînes de caractères restant aux attributs restants, en commençant par les attributs libres les plus à droite (selon le même principe que dans le mapping par défaut);
- enfin, convertir localpart en Personal comme vu dans le mapping par défaut.

Exemples:

Supposons que le fichier ".rfc987.r" contienne :

```
de              # O=;A=;C=de
dbp.de          # O=;A=dbp;C=de
mpg.dbp.de      # P=mpg;A=dbp;C=de
```

1°) Soit l'adresse RFC822:
"dittrich@fokus.berlin.gmd.dbp.de"

1. Personal Name non compris

- les lignes applicables sont les lignes n° 1 et 2;
- parmi les lignes n° 1 et 2, la ligne contenant dans sa colonne de gauche le plus de sous-chaînes de caractères est la ligne n° 2 (la ligne 1 ne comprend qu'1 seule sous-chaîne; la ligne 2 en comprend 2);
- donc la ligne n° 2 est appliquée:
  - la "partie supérieure" de l'adresse X.400 sera "O=;A=dbp;C=de" (voir colonne de droite);
  - la "partie inférieure" de l'adresse X.400 sera "OU=fokus;OU=berlin;P=gmd";
  - localpart "dittrich" est converti en Personal Name "S=dittrich".

Donc, l'adresse X.400 correspondante sera "S=dittrich;OU=fokus;OU=berlin;P=gmd;A=dbp;C=de".

2°) Soit l'adresse RFC822:
"tintin@institut.mpg.dbp.de"

- toutes les lignes sont applicables;
- parmi les lignes n° 1, 2 et 3, la ligne contenant dans sa colonne de gauche le plus de sous-chaînes de caractères est la ligne n° 3 (la ligne 1 ne comprend qu'1 seule sous-chaîne; la ligne 2 en comprend 2; la ligne 3 en comprend 3);
- donc la ligne n° 3 est appliquée:
  - la "partie supérieure" de l'adresse X.400 sera "P=mpg;A=dbp;C=de" (voir colonne de droite);
  - la "partie inférieure" de l'adresse X.400 sera "O=institut";
  - localpart "tintin" est converti en Personal Name "S=tintin".

Donc, l'adresse X.400 correspondante sera "S=tintin;O=institut;P=mpg;A=dbp;C=de".

2.4. Le User Service

Le User Service (US) est constitué d'un ensemble d'UAs (rappel : un UA est un programme interface entre un utilisateur et son MTA). EAN distingue les UAs interagissant avec un utilisateur humain et ceux interagissant avec un utilisateur non humain (applications informatiques). Dans le premier cas, les UAs conservent leur appellation d'UA, dans le second cas on leur donne un nom plus spécifique.

2.4.1. L'UA

A) But

L'UA effectue les fonctions suivantes :

- composition, édition, envoi de messages vers d'autres utilisateurs;
- réception, sauvegarde, organisation et consultation de messages reçus d'autres utilisateurs;
- utilisation facilitée du Directory Service;
- gestion du profile.

Les commandes EAN exécutant ces fonctions sont configurables en anglais ou en français (extensible à d'autres langues).

Par rapport aux spécifications X.400, on peut noter que:

- Le User Service de EAN n'offre pas la possibilité de tester des adresses (fonction "**probe**").

- La fonction Blind Copy Carbon existe mais ne fonctionne pas correctement.
- Dans EAN, un UA est uniquement actif lorsque l'utilisateur de l'UA lance son programme **ean** (voir section 4.3.3.). La recommandation X.400 suppose l'existence permanente des UAs et MTAs.
- L'UA a toujours l'initiative de l'établissement du dialogue avec le MTA (si le MTA pouvait en avoir l'initiative et si l'UA n'était pas actif à ce moment, alors le MTA serait malgré tout obligé d'attendre l'activation de l'UA; il vaut donc mieux que ce soit l'UA qui ait l'initiative).

B) Les messages

- les messages satisfont le protocole P2;
- actuellement, les messages ne peuvent contenir que du texte ASCII (bien que la recommandation X.400 prévoie d'autres types de données);
- les champs des messages sont affichés sous forme de texte mais stockés selon leur représentation X.409[1];
- l'utilisateur peut définir dans son user profile les champs pour lesquels EAN doit automatiquement demander une valeur, lors du remplissage du message (commande **set compose**)[2].

C) L'arrivée d'un message

Si l'utilisateur n'est pas en cours de session de travail avec son UA, alors il ne sera averti de l'arrivée d'un nouveau message que quand il commencera une nouvelle session UA.

Si, au contraire, l'utilisateur est en cours de session UA, alors il est averti immédiatement (de façon asynchrone) de l'arrivée du nouveau message. Ceci est réalisé par un programme qui est lancé à chaque réception d'un nouveau message (**work program**).

Lorsqu'un message est accepté par l'UA, il est sauvé dans un "**folder**" particulier. Ce folder est :

- soit un folder par défaut (appelé **inbox**);
- soit un folder déterminé automatiquement par l'UA selon les contenus de certains champs du message (les règles précises permettant d'effectuer le choix sont spécifiées par l'utilisateur dans son user profile: commande **set folder**).

Les messages sont ordonnés chronologiquement dans un même folder.

Un message n'est stocké qu'une seule fois, même s'il doit appartenir à plusieurs folders : une table permet de faire la conversion entre un numéro de message dans un folder et un numéro identifiant le message et son emplacement en mémoire.

---

1. Les champs possibles sont: To:, Cc:, Bcc:, From:, Authorizing-Users:, In-Reply-To:, Reply-By:, Reply-To:, Subject:, Message-Id:, Obsoletes:, References:, Importance:, Sensitivity:, Expiry-Date:, Auto-forward:. Leur signification est donnée dans [EANMAN], chapitre 5.

2. Remarques:
- le champ "To:" est toujours demandé;
- certains champs sont automatiquement remplis par EAN (par exemple, le champ "From:" est automatiquement rempli par le NUA de l'utilisateur, accompagné du nom de celui-ci comme commentaire).

2.4.2. Les autres UAs

Nous n'en verrons qu'un exemple ici : le "**Distribution List User Agent**". Son rôle est de dévier tous les messages arrivant à un NUA vers un ou plusieurs autre(s) NUA(s).

Fonctionnement :

- le work program du NUA à dévier est remplacé par un programme spécial appelé **dist program**;
- quand un message arrive au NUA à dévier, le dist program est lancé; celui-ci crée alors un nouveau message en remplaçant dans l'enveloppe P1 du premier message le NUA par le(s) nouveau(x) NUA(s) de destination[1]; ce nouveau message est ensuite envoyé.

Les Distribution Lists peuvent être utilisées par exemple pour créer des groupes de discussion sur des sujets bien précis. On crée alors une boîte aux lettres spéciale par groupe de discussion. Les membres du groupe envoient leurs contributions dans cette boîte aux lettres. Le work program associé à cette boîte aux lettres se chargera de réexpédier les contributions à tous les membres du groupe.

Il s'agit donc d'une implémentation simplifiée des listes de distribution introduites dans la version 1988 de X.400 (voir 1.10.4..A.).

2.5. Le Message Transfer Service

Le Message Transfer Service (MTS) est constitué d'un ensemble de MTAs répartis dans tout le réseau.

Le rôle des MTAs est d'une part d'assurer l'interface entre l'utilisateur et le réseau, d'autre part de permettre l'acheminement des messages de proche en proche (de MTA en MTA).

2.5.1. Services

Les services offerts par les MTAs aux UAs sont :

- établissement et terminaison de la connexion;
- enregistrement d'un nouvel utilisateur;
- changement de mots de passe;
- soumission de message[2];
- livraison de message[3].

Dans EAN, le MTA et l'UA sont co-résidents (situation S3 dans l'avis X.400); aucun protocole P3 n'est dès lors nécessaire, et il n'existe pas de SDEs. Cependant, l'interface dans EAN entre MTA et UA est fort similaire au protocole P3 (voir les fonctions P3, section 8.4.).

2.5.2. Structure des données

Un MTA reçoit un message soit d'un UA local (i.e. qui lui est directement connecté), soit d'un autre MTA. Son rôle est de transmettre ce message dans la boîte aux lettres d'un autre UA local ("**local delivery**") ou d'un autre MTA ("**forwarding**"). Pour réaliser ce travail, chaque MTA possède les données suivantes:

---

1. Les NUAs de destination doivent avoir été écrits au préalable dans un fichier texte particulier.

2. y compris les possibilités suivantes: destinataires multiples, priorité, livraison après une certaine date et heure, notifications de livraison et de non-livraison.

3. y compris les possibilités suivantes: indication de l'O/R name de l'originator et du recipient, indication du moment de soumission et du moment de livraison.

A) Les "message files" et les "queue files"

Le MTA comprend :

- pour chaque nouveau message non encore traité : un fichier comprenant le message proprement dit : le **message file**;
- pour chaque UA et pour chaque MTA directement relié au MTA : un fichier comprenant la liste des messages attendant d'être envoyés vers cet UA ou ce MTA : le **queue file**.

Remarquons que le contenu d'un message n'est stocké que dans son message file. Les queue files ne contiennent que des références à des messages. Donc, si le MTA doit envoyer un message à plusieurs UAs ou MTAs, son contenu n'est stocké qu'une seule fois. Cette situation est illustrée à la figure 2.1.

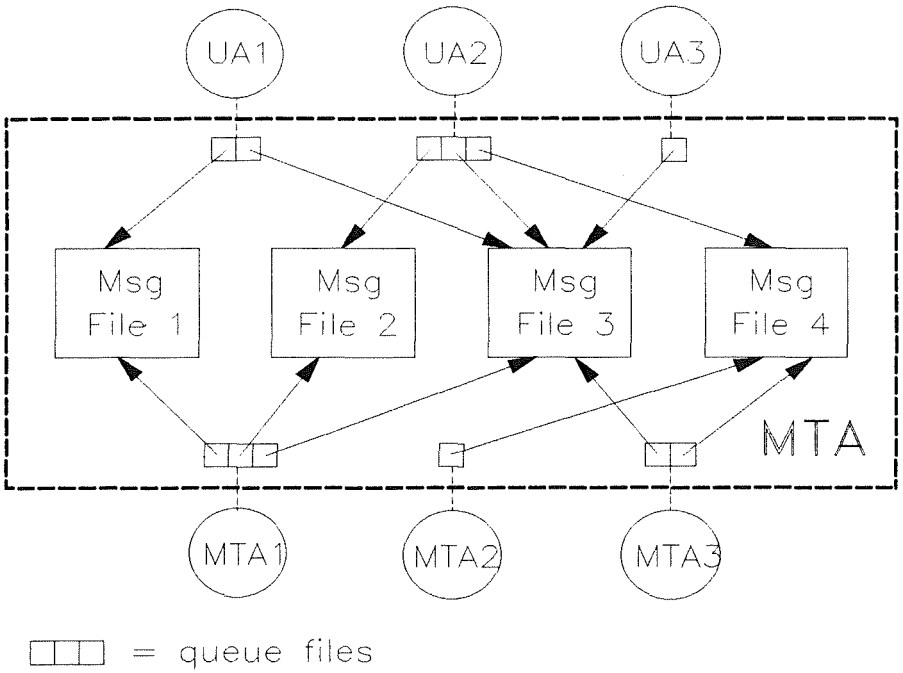


Figure 2.1 : Message Files et Queue Files.

B) Les statuts de responsabilité

i) A chaque message file (c.-à-d. à chaque message) est associé une liste de statuts de responsabilité : il y a un statut de responsabilité par UA destinataire du message. Ce statut vaut '**r**' ("**responsible**") si le message doit être envoyé à l'UA destinataire ou à un MTA intermédiaire, mais ne l'a pas encore été; il vaut '**n**' ("**non responsible**") sinon.

Les statuts de responsabilité permettent de déterminer si, à un moment donné, un message donné:

- a déjà été envoyé vers[1] un destinataire donné;
- a déjà été envoyé vers tous ses destinataires (dans ce cas, son message file peut être supprimé).

(initialement, lorsqu'un message arrive dans le MTA, tous ses statuts de responsabilité pour chaque destinataire sont initialisés à "responsible")

ii) Une liste de statuts de responsabilité est également associée à chaque entrée (chaque message) de chaque queue file. De nouveau, cette liste comprend un statut de responsabilité pour chaque destinataire (recipient) du message.

Le statut de responsabilité d'une entrée d'une queue file vaut '**r**' si le message doit encore être envoyé à l'UA ou au MTA associé à la queue file. Sinon, il vaut '**n**'.

Donc si la queue file est associée à un UA, alors seul le statut de responsabilité associé à l'UA est initialisé à 'r'. Si la queue file est associée à un MTA (c.-à-d. si le message doit être envoyé à un ou plusieurs UA(s) via ce MTA), alors les statuts de responsabilité initialisés à 'r' seront ceux associés aux UA destinataires nécessitant le passage du message par le MTA associé à la queue file.

A tout moment, la liste des statuts de responsabilité d'un message file est donc la réunion de tous les statuts de responsabilité des entrées de queue files correspondant à ce message.

C) Les tables

Le MTA utilise également des tables pour déterminer la route à suivre par les messages. Ces tables et l'algorithme de routage sont décrits dans le chapitre 3.

2.6. La soumission et la livraison de messages

L'interaction entre un UA et son MTA comprend 3 phases:

- l'établissement de la session;
- la soumission d'un message;
- la livraison du message.

2.6.1. L'UA originator établit l'accès à son MTA

L'accès par l'UA au MTA est réalisé en deux temps, en respectant les primitives de service **LOGON.Request** et **LOGON.Confirmation** (voir section 1.9.1. de ce mémoire et [X411], section 2.2.1):

(1) l'UA demande un accès au MTA en lui donnant son NUA et son mot de passe (primitive **LOGON.Request**);

(2) le MTA vérifie que l'UA est local et que le mot de passe fourni est correct (c.-à-d. est le même que celui mémorisé dans sa User Agent Table); il informe l'UA si oui ou non il autorise l'accès (primitive **LOGON.Confirmation**).

---

1. L'expression "**envoyer un message vers un destinataire**" signifie ici:
- "envoyer le message à l'UA du destinataire" si celui-ci est relié directement au MTA;
- "envoyer le message à un autre MTA qui le prendra en charge" si l'UA destinataire n'est pas directement connecté au MTA.

Remarques:

1°)EAN attribue automatiquement des valeurs pour les mots de passe (sauf les mots de passe des UAs de la version UBC1 de EAN, qui sont modifiables par l'utilisateur). L'utilisateur n'a donc pas à se soucier des mots de passe.

2°) Lorsque l'UA ne doit plus rien envoyer au MTA ou bien ne plus rien en recevoir, il demande au MTA d'arrêter la conversation à l'aide de la primitive **LOGOFF.Request**; le MTA lui répond à l'aide de la primitive **LOGOFF.Confirmation** (voir section 1.9.1. de ce mémoire et [X411], section 2.2.3).

2.6.2. L'UA originator soumet le message à son MTA

La soumission d'un message par l'UA au MTA est réalisée en 2 temps, en respectant les primitives de service **SUBMIT.Request** et **SUBMIT.Confirmation** (voir section 1.9.1. de ce mémoire et [X411], section 2.2.7):

(1) l'UA communique au MTA le message à envoyer (primitive **SUBMIT.Request**);

(2) le MTA détermine si oui on non il accepte la responsabilité du transfert du message; il communique la réponse à l'UA (primitive **SUBMIT.Confirmation**)[1].

Lorsqu'il a accepté un message, le MTA effectue les opérations suivantes:

(1) le MTA crée un fichier contenant le message : un message file;

(2) le MTA détermine pour chaque destinataire s'il est local ou non; s'il n'est pas local, il détermine vers quel MTA envoyer le message (tout ceci se fait selon l'algorithme de routage exposé au chapitre 3; on obtient ainsi une liste des queue files qui devront recevoir le message;
remarque: si l'algorithme de routage rapporte une ou des erreurs, des messages de non-livraison sont renvoyés en conséquence.

(3) pour chaque queue file déterminé en (2) :
le MTA crée une entrée dans ce queue file référençant le message file et ayant les statuts de responsabilité initialisés comme suit :
- si le queue file correspond à un UA local: seul le statut de responsabilité de cet UA vaut 'r'; les autres valent 'n';
- si le queue file correspond à un autre MTA: les statuts de responsabilité valant 'r' sont ceux correspondant à un UA destinataire nécessitant le passage par cet autre MTA; les autres valent 'n';

(4) la liste des statuts de responsabilité du message file est initialisée en faisant le OU logique de tous les statuts de responsabilité des entrées de queue files crées en (3).

---

1. Remarques:

1°) Une confirmation positive signifie donc simplement que le MTA accepte de traiter le message. Elle ne signifie nullement que le transfert et/ou la livraison a été effectué ou est possible.

2°) La confirmation peut être négative par exemple dans le cas d'une mauvaise configuration des tables du MTA (voir le chapitre 3 pour les tables du MTA et le chapitre 8 pour l'implémentation précise en C de ces primitives de service).

2.6.3. Le MTA recipient livre le message à l'UA recipient

Pour chaque queue file:

- Si le queue file correspond à un UA local: le work program de cet UA est exécuté[1]. Dans la plupart des cas, ce programme effectue la livraison de messages à l'UA. Il effectue donc les opérations suivantes (pour chaque entrée dans le queue file):
  - le message file est envoyé à l'UA (primitive **DELIVER.Indication**[2]: voir section 1.9.1. de ce mémoire et [X411], section 2.2.9);
  - l'entrée dans le queue file est supprimée et le statut de responsabilité correspondant à l'UA dans le message file est mis à 'n';
  - si tous les statuts de responsabilité du message file valent 'n', alors le message file est supprimé.

- Si le queue file correspond à un autre MTA: le message est transporté à ce MTA, selon un principe similaire au cas ci- dessus. Les principales différences sont qu'ici:
  - le RTS est utilisé pour relier les 2 MTAs;
  - le MTA recevant le message ne doit considérer que les destinataires dont le statut dans la liste de statuts de responsabilité du queue file vaut 'r' (c.-à-d. les destinataires dont il est responsable).

2.7. Les protocoles utilisés par EAN

Les protocoles mis en oeuvre dans les différentes couches du modèle OSI sont illustrés à la figure 2.2.

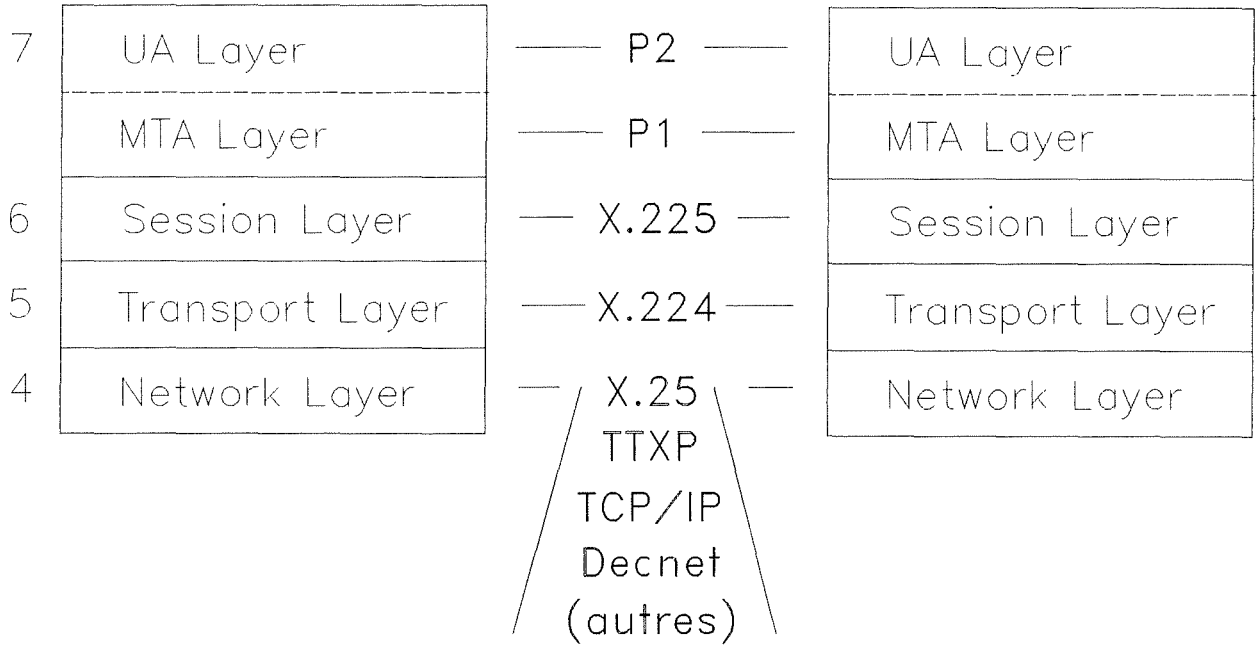


Figure 2.2 : protocoles utilisés dans EAN

2.7.1. Couche application (niveau 7)

La couche application est subdivisée en:

- **UA Layer** (appelée aussi **Interpersonal Messaging UA Layer** dans le cas de l'IPMS): le protocole P2 spécifie le format des PDUs échangés entre IPM UAs (IPM

1. Le work program d'un UA est spécifié dans la User Agent Table.
2. L'UA ne peut refuser une livraison de message.

UAPDUs);
- **MT Layer** : le protocole P1 spécifie le format des PDUs échangés entre MTAs (MPDUs).

Remarquons que dans EAN, l'UAE et la MTAE sont toujours co-résidents; par conséquent il n'existe ni SDE, ni protocole P3 entre SDE et MTAE. Il n'y a donc pas non plus de sous-couche ROS dans la MTL. Un MTA ne comprend donc que le Message Dispatcher, l'Association Manager et le Reliable Transfer Service (voir section 1.7. pour la définition de ces termes).

2.7.2. Couche présentation (niveau 6)

La couche présentation est inexistante, car les messages, adresses, ... sont codés dans EAN sous forme standardisée X.409.

2.7.3. Couche session (niveau 5)

La couche session implémente le Basic Activity Subset du protocole X.225, à l'exception de l'interruption et la reprise d'activités et de la demande de jeton (voir [EAN2] page 15). Le protocole de session est utilisé pour transférer un message comme suit:

- une session est établie entre les 2 MTAs;
- le MTA appelant envoie ses messages au MTA appelé;
- le MTA appelant donne le contrôle de la session au MTA appelé, qui peut alors à son tour envoyer au premier MTA les éventuels messages qu'il aurait pour lui;
- lorsque tous les messages sont échangés entre les deux MTAs, la session est terminée.

2.7.4. Couche transport (niveau 4)

La couche transport implémente le protocole X.224, avec la classe 0 (la plus simple).

Remarque : si nécessaire (c.-à-d. si le service offert par le niveau 3 n'est pas assez fiable), il est possible d'implémenter la classe 4 (mais cela n'est pas fait dans la version standard de EAN).

2.7.5. Couche réseau (niveau 3)

Plusieurs protocoles sont proposés pour la couche réseau:

- X.25 (accès direct (c.-à-d. sans nécessiter de PAD) à X.25);
- TTXP (accès X.25 par PAD);
- TCP/IP[1] (exemple: pour réseaux Ethernet);
- Decnet (pour les machines VAX utilisant VMS).

1. TCP/IP est considéré par EAN comme un protocole de niveau 3.

2.8. Le Directory Service

2.8.1. Description

Le but principal du Directory Service (DS) est de permettre la conversion d'un NUN en un NUA. Il permet également d'obtenir d'autres informations diverses (adresse, téléphone, description) au sujet d'un utilisateur. Un utilisateur y accède par son UA.

Chaque entrée du DS correspond à un utilisateur et contient:

- son NUN (son nom + le nom de son organisation[1]);
- les synonymes du nom de l'utilisateur ("**Alternates**");
- son NUA;
- son adresse (optionnel);
- son téléphone (optionnel);
- sa description (optionnel).

Il est possible d'ajouter, de supprimer, de mettre à jour et de retrouver des entrées (voir respectivement les commandes **register, drop, install, find**, par exemple dans [EANMAN], respectivement pages 76, 58, 70, 63). Remarquons que l'enregistrement d'un utilisateur dans le DS n'est pas obligatoire (mais il est vivement conseillé !).

2.8.2. Structure et fonctionnement

Le DS n'est pas centralisé: il est réparti dans plusieurs **Central Nameservers** gérant chacun une partie de la base de données du DS (typiquement, il y a un Central Nameserver par PRMD). Pour accéder à ces Central Nameservers, l'utilisateur doit passer par l'intermédiaire d'un **Relay Nameserver** (typiquement, il y a un Relay Nameserver par organisation, appelé **Organizational Nameserver**).

Les commandes de l'utilisateur sont envoyées depuis son UA au Relay Nameserver de son organisation (sous forme de message). Ce dernier ajoute au message le nom de l'organisation[2] et envoie le nouveau message au Central Nameserver du PRMD de cette organisation.

Ce dernier examine alors la requête et tente de la satisfaire. Eventuellement, s'il ne sait pas la satisfaire localement, alors il peut l'envoyer à un autre Central Nameserver... Les réponses aux requêtes sont toujours renvoyées directement à l'UA.

2.8.3. Enregistrement d'un utilisateur

Pour que les données concernant un utilisateur W soient accessibles par les autres utilisateurs XYZ, il faut que ces données aient été envoyées au DS par W. Il utilise pour cela la commande "**register**"[3]. Cette commande déclenche les opérations suivantes:

---

1. Le terme "organisation" utilisé ici est spécifié dans la section 2.3.1.

2. Le nom de l'organisation utilisé ici est introduit par le gestionnaire de système lors de la création du Relay/Organizational Nameserver. Il est donc inconnu du MTA de l'utilisateur. Un MTA ne connaît pas le nom de l'organisation à laquelle il appartient (même dans la version DFN, ce nom ne correspond pas nécessairement à l'attribut standard X.400 "O"). Un MTA ne connaît que l'adresse de son Relay/Organizational Nameserver (cf. [EANUBC], page 14). C'est pourquoi le nom de l'organisation ne peut être spécifié par le MTA et doit être ajouté par le Relay/Organizational Nameserver.

3. Lors de la première exécution de EAN par un nouvel utilisateur, on lui demande s'il désire s'enregistrer au Directory Service. Si oui, la commande "register" est alors automatiquement exécutée. Si non, l'utilisateur peut toujours l'exécuter ultérieurement.

1°) Le programme EAN va lire dans le profile de l'utilisateur son nom (**Name**), les synonymes de son nom (**Alternate**), son adresse (**Address**), ses numéros de téléphone (**Phone**), sa description (**Description**) ainsi que le nom de sa boîte aux lettres (**Mailbox**). Si certains de ces renseignements n'ont pas été entrés dans le profile, alors ils sont demandés à ce moment à l'utilisateur.

Exemple (visualisable par la commande "show profile"):

```
Name = Xavier Rutten
Alternate = Rutten,X.Rutten
Address = 18,place des Tilleuls
          B-5220 Andenne
          Belgium
Phone = [0]85/84.23.58
Description = student
Mailbox = xru
```

2°) EAN envoie ces renseignements sous forme de message au Relay Nameserver.

3°) Le Relay Nameserver ajoute le nom de l'organisation et renvoie le message au Central Nameserver.

4°) Le Central Nameserver se rend compte qu'une requête d'enregistrement est arrivée. Il en extrait alors les diverses informations et les stocke dans sa base de données.

Remarque:

A partir du nom de l'utilisateur (donnée provenant du profile) et du nom de son organisation (donnée ajoutée par le Relay/Organizational Nameserver), le DS en déduit le NUN. Le NUA, quant à lui, peut se trouver dans la zone "From" du message correspondant à la requête initiale de l'utilisateur. Donc, le DS possède les données suffisantes pour déterminer le NUN et le NUA à stocker dans sa base de données.

5°) Le Central Nameserver renvoie alors un message de confirmation directement à l'utilisateur qui voulait s'enregistrer.

2.8.4. Remarque

Le DS de la version 1984 de EAN est une fonctionnalité temporaire. La norme X.500 n'existait pas à l'époque. Les versions futures de EAN suivront très certainement la nouvelle norme X.500.

Chapitre 3 : Les tables du MTA et le routage

Ce chapitre décrit la façon dont est réalisé le routage dans un MTA, et cela pour la version UBC1 et la version DFN de EAN. Il est essentiellement basé sur [EAN2], pages 11, 12 et 14, ainsi que sur [EANDFN], pages 6 à 13, 15 à 19 et 24 à 28.

3.1. La version UBC1

Le MTA a à sa disposition 5 tables:

- la **User (Agent) Table** : contient des informations sur les UAs locaux;
- la **Connection Table** : contient des informations sur les MTAs directement reliés au MTA;
- la **Local Identification Table** : contient des informations sur le MTA lui-même;
- la **Subdomain Table** : contient des informations sur les sous-domaines appartenant au même PRMD que le MTA;
- la **Private Domain Table** : contient des informations sur les PRMDs.

Nous nous baserons dans cette section 3.1. sur l'exemple fictif du PRMD "**cdn**" illustré à la figure 3.1. Les exemples de tables de MTAs seront les tables que pourrait posséder le MTA du sous-domaine "ean" dans ce PRMD.

Remarques relatives à la figure 3.1:

1°) Chaque MTA représente un et un seul sous-domaine, mais certains sous-domaines ne sont représentés par aucun MTA (exemple: ubc et carleton)[1].

2°) Les MTAs ont ici pour nom la juxtaposition des noms des sous-domaines auxquels ils appartiennent (séparés par des points). Ce n'est pas obligatoire, mais cela clarifie les tables de MTAs.

3°) Les autres PRMDs[2] sont accessibles via des MTAs spéciaux appelés **gateways** ("portes").

1. De tels sous-domaines sont donc inaccessibles (seuls sont accessibles les sous-domaines de ces sous-domaines). Ils ne peuvent donc comprendre aucun UA (sauf les UAs de leurs sous-domaines). Puisqu'ils ne comprennent ni UA, ni MTA, ils n'ont donc qu'un rôle passif, théorique.

2. qu'ils soient ou non X.400.

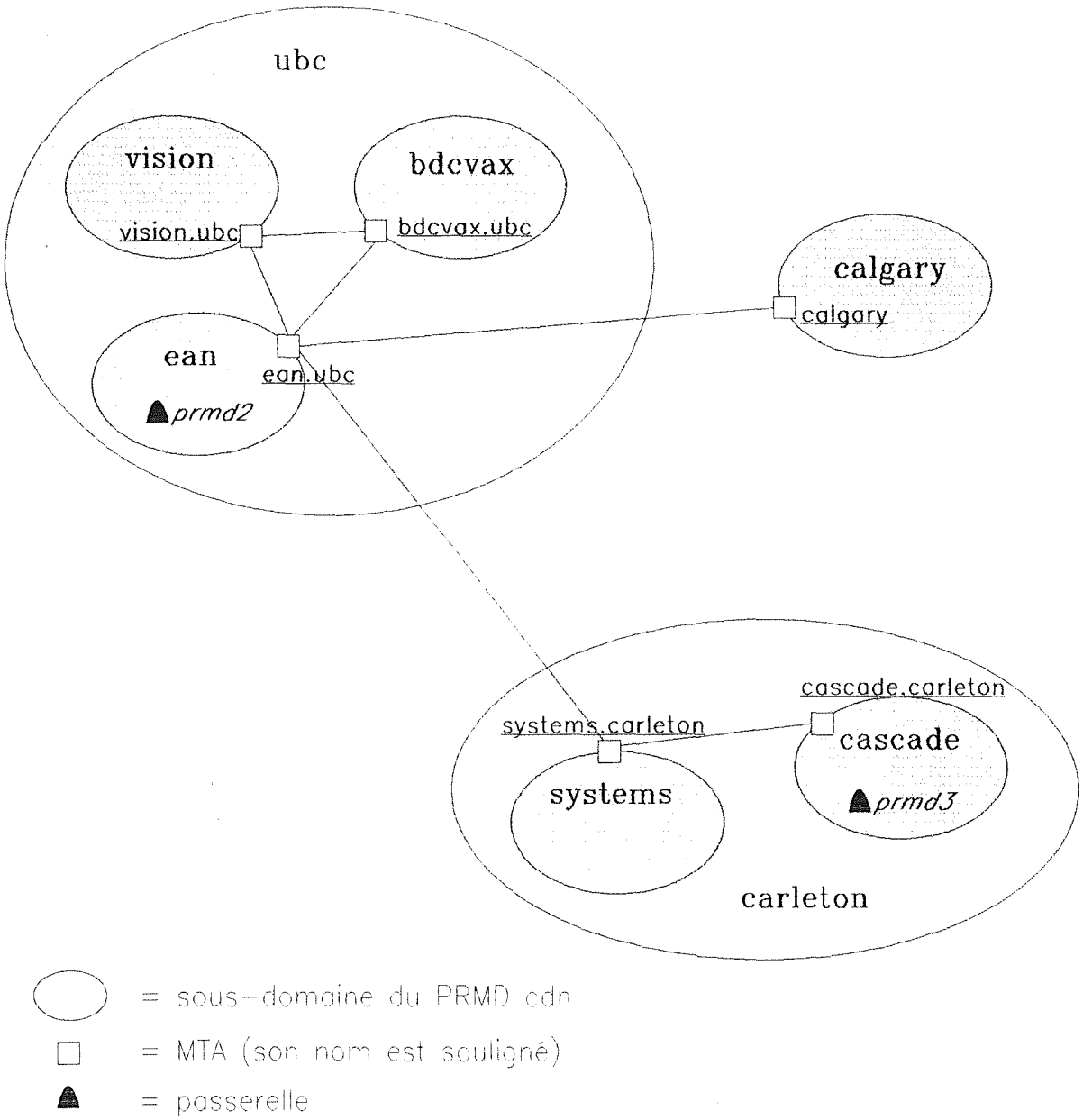


Figure 3.1 : exemple fictif pour EAN version UBC1: PRMD "**cdn**"

3.1.1. La User (Agent) Table

La **User (Agent) Table** comprend pour chaque utilisateur enregistré sur le MTA:

- le nom de son UA (=son "userid");
- le nom de sa queue file (généralement le même que son userid);
- le nom du work program à exécuter quand du courrier arrive pour cet utilisateur;
- le mot de passe autorisant l'accès par cet utilisateur.

Exemple:

UA	Queue file	Work Program	Password
xru	xru	auto	dedekdke
jva	jva	auto	uhfcnfz
forum	forum	dist dist_file forum@ean.ubc pwd	enzifeoa

Remarque: **auto** = le work program standard; **dist** = le work program renvoyant le message vers une ou plusieurs autres adresses (les caractères qui suivent sont les paramètres du programme **dist**).

3.1.2. La Connection Table

La **Connection Table** comprend pour chaque MTA directement connecté au MTA:

- le nom de ce MTA (généralement la juxtaposition des sous-domaines imbriqués contenant le MTA);
- le nom de sa queue file (généralement une partie du nom du MTA);
- des informations diverses utilisées par le RTS pour effectuer la connexion avec le MTA (voir plus loin), par exemple le protocole de réseau à utiliser.

Exemple:

Connection	Queue file	Network Information
vision.ubc	_vision	inet,ubc-vision,ean
bdcvax.ubc	_bdcvax	ttxp,ubc-bdcvax,ean
systems.carleton	_carleton	x25,20409999,,ean1
calgary	_calgary	x25,62601234,,ean1
.local	_local	nsg,a822,sendmail

La dernière ligne de ce tableau correspond au gateway (MTA spécial) local vers l'autre PRMD "prmd2". Les autres lignes correspondent aux autres MTAs directement accessibles par le MTA "ean" à la figure 3.1 (c.-à-d. "vision" et "bdcvax" dans "ubc", "systems" dans "carleton" et "calgary").

Dans la colonne "Network Information", "inet", "ttxp", "x25", "nsg" signifient que la connexion s'effectue respectivement par TCP/IP, TTXP, X.25 ou par un gateway. Les informations qui suivent sont les paramètres de la connexion (par exemple l'adresse X.25 dans le cas d'une connexion X.25). La liste complète de ces paramètres et leur syntaxe précise est donnée par exemple dans [EANUBC], sections 4.4 et 4.5.

3.1.3. La Local Identification Table

La **Local Identification Table** ne comprend que le nom du MTA et le nom du PRMD auquel il appartient.

Exemple:

Local MTA name	PRMD	admd	country
ean.ubc	cdn		

Les colonnes "admd" et "country" sont prévues pour des extensions futures et ne sont pas utilisées.

3.1.4. La Subdomain Table

La Subdomain Table comprend une entrée par sous-domaine accessible directement par le MTA et une entrée par sous-domaine "global" dans l'imbrication des sous-domaines du PRMD local (voir exemple).

Chacune de ces entrées comprend d'une part le nom du sous-domaine (complété par les noms des sous-domaines dans lesquels il est compris) et le nom du MTA qui permet d'atteindre ce sous-domaine.

Exemple:

Subdomain	Connection
ubc	
ean.ubc	@local
vision.ubc	vision.ubc
bdcvax.ubc	bdcvax.ubc
carleton	systems.carleton
calgary	calgary

Les sous-domaines directement accessibles par le MTA sont:

- "ean", "vision" et "bdcvax" dans "ubc":
  - "ean" est le sous-domaine du MTA; on indique donc par convention la chaîne de caractères "**@local**" dans la colonne "Connection";
  - "vision" et "bdcvax" sont accessibles respectivement par le MTA "vision.ubc" et le MTA "bdcvax.ubc";
- "carleton": est accessible par le MTA "systems.carleton";
- "calgary": est accessible par le MTA "calgary".

Les sous-domaines "globaux" sont "ubc", "calgary" et "carleton". "calgary" et "carleton" ont déjà une entrée dans la table. Il reste donc à ajouter une entrée pour "ubc". Cette entrée ne comprend pas de nom de MTA dans la colonne "Connection" car le sous-domaine "ubc" n'est représenté par aucun MTA.

3.1.5. La Private Domain Table

La **Private Domain Table** comprend une entrée par PRMD accessible (directement ou non) par le MTA. Chaque entrée contient le nom du PRMD et le nom du MTA à utiliser pour atteindre ce PRMD.

Exemple:

Private Domain	Connection
cdn	
prmd2	.local
prmd3	systems.carleton

Les PRMDs accessibles par le MTA sont "cdn", "prmd2" et "prmd3".

- "cdn" est le PRMD-même du MTA: on ne place donc rien dans la colonne "Connection" puisqu'il est directement accessible;
- "prmd2" est un deuxième PRMD, accessible directement depuis le MTA (le gateway se trouve dans "ean"): on place donc "**.local**" dans la colonne "Connection" ("**.local**" désigne le MTA spécial gateway vers le PRMD "prmd2": voir la Connection Table)
- "prmd3" est un troisième PRMD, accessible par le MTA "systems.carleton". On place donc "systems.carleton" dans la colonne "Connection".

3.1.6. Le routage

Le MTA effectue les opérations décrites ci-dessous, pour chaque destinataire d'un message reçu. Nous illustrerons ces opérations à l'aide de la configuration de la figure 3.1 et des exemples de destinataires de message suivants:

Exemple N°	Destinataire du message
1.	"user1@abc.def.prmdx"
2.	"user2@ghi.jkl.prmd3"
3.	"user3@cascade.carleton.cdn"
4.	"user4@mno.carleton.cdn"
5.	"user5@vision.ubc.cdn"

1°) Le MTA détermine si le destinataire du message est dans le même PRMD. Si oui, on passe directement au point 2°). Si non, le MTA cherche dans la Private Domain Table le MTA servant d'intermédiaire pour l'autre PRMD[1]. Le message sera placé dans la queue file de ce MTA.

Exemple 1: Le destinataire appartient à un PRMD inconnu => stop (accusé de non-livraison).

Exemple 2: Le destinataire appartient à un autre PRMD; le message doit être placé dans la queue file du MTA "systems.carleton" (en vue de se rapprocher du gateway dans "cascade.carleton").

Exemples 3,4,5: Le destinataire appartient au même PRMD => passage au point 2°).

2°) Routage à l'intérieur du domaine:

Le MTA examine d'abord le dernier élément de la liste des sous-domaines du NUA recipient et cherche dans la Subdomain Table un sous-domaine correspondant. S'il n'en trouve pas, l'adresse du destinataire est inconnue et l'algorithme s'arrête sur un échec.

S'il en trouve un, le MTA examine alors la juxtaposition des deux derniers sous-domaines du NUA recipient. De nouveau, il cherche une entrée correspondante dans la Subdomain Table. L'algorithme continue ainsi en examinant ensuite successivement les trois, quatre, cinq... derniers sous-domaines du NUA recipient et en cherchant chaque fois une entrée correspondante dans la Subdomain Table.

Le processus s'arrête soit lorsqu'on arrive au bout de la liste des sous-domaines du NUA recipient, soit lorsqu'aucune entrée de la Subdomain Table ne correspond à l'ensemble des sous-domaines en cours d'examen. L'algorithme examine alors la dernière entrée qui a été cherchée et trouvée avec succès dans la Subdomain Table et en extrait le nom du MTA correspondant. Il s'agit du MTA permettant d'atteindre le destinataire. Il ne reste alors plus qu'à placer le message dans la queue file de ce MTA.

---

1. Si ce MTA n'est pas trouvé, c'est que le PRMD du destinataire du message est inconnu. L'algorithme de routage s'arrête alors ici sur un échec.

Remarque: dans tous les cas d'échec, un accusé de non-livraison est renvoyé à l'expéditeur.

Remarques:
- Si le nom ce MTA est vide, alors le sous-domaine ne comprend pas de MTA et est donc inaccessible. L'algorithme s'arrête donc ici sur un échec.
- Si ce nom de MTA est "**@local**", alors le MTA est le MTA recipient et il suffit de placer le message dans la queue file du destinataire (après vérification dans la User (Agent) Table qu'il existe).

Exemple 3: L'algorithme examine successivement "carleton" puis "cascade.carleton". Il trouve une entrée dans la Subdomain Table pour "carleton", mais pas pour "cascade.carleton" = > il utilise l'entrée pour "carleton". Le message est donc placé dans la queue file du MTA correspondant: "systems.carleton".

Exemple 4: L'algorithme examine successivement "carleton" puis "mno.carleton". Il trouve une entrée dans la Subdomain Table pour "carleton", mais pas pour "mno.carleton" = > il utilise l'entrée pour "carleton". Le message est donc placé dans la queue file du MTA correspondant: "systems.carleton". Remarquons que le MTA ne se rend pas compte lui-même que le sous-domaine "mno" de "carleton" n'existe pas.

Exemple 5: L'algorithme examine successivement "ubc" puis "vision.ubc". Il ne trouve pas d'autre sous-domaine dans le NUA recipient = > il utilise la dernière entrée trouvée (celle correspondant à "vision.ubc"). Le message est donc placé dans la queue file du MTA correspondant: "vision.ubc".

3.2. La version DFN

Le MTA a à sa disposition 5 tables :

- la **User Agent Table** : contient des informations sur les UA locaux;
- l'**Organization Table** : contient des informations sur les MTAs à utiliser pour accéder aux différentes organisations et unités organisationnelles;
- la **Domain Table** : contient des informations sur les MTAs à utiliser pour accéder aux différents domaines;
- la **MTA Table** : contient des informations sur les MTA spécifiés dans l'Organization Table et dans la Domain Table;
- la **Local Identification Table** : contient des informations sur le MTA lui-même;

Par rapport à la version UBC1, on peut donc voir essentiellement les différences suivantes:

- la User Agent Table et la Connection Table sont conservées (la User Agent Table ne comprend plus de mot de passe; la Connection Table s'appelle maintenant MTA Table);
- la Local Identity Table est conservée (elle s'appelle maintenant Local Identification Table);
- la Subdomain Table et la Private Domain Table sont remplacées par l'Organization Table et la Domain Table.

Nous nous baserons dans cette section 3.2. sur l'exemple fictif illustré à la figure 3.2. Les exemples de tables de MTAs seront les tables que pourrait posséder le MTA "info.fundp.rtt".

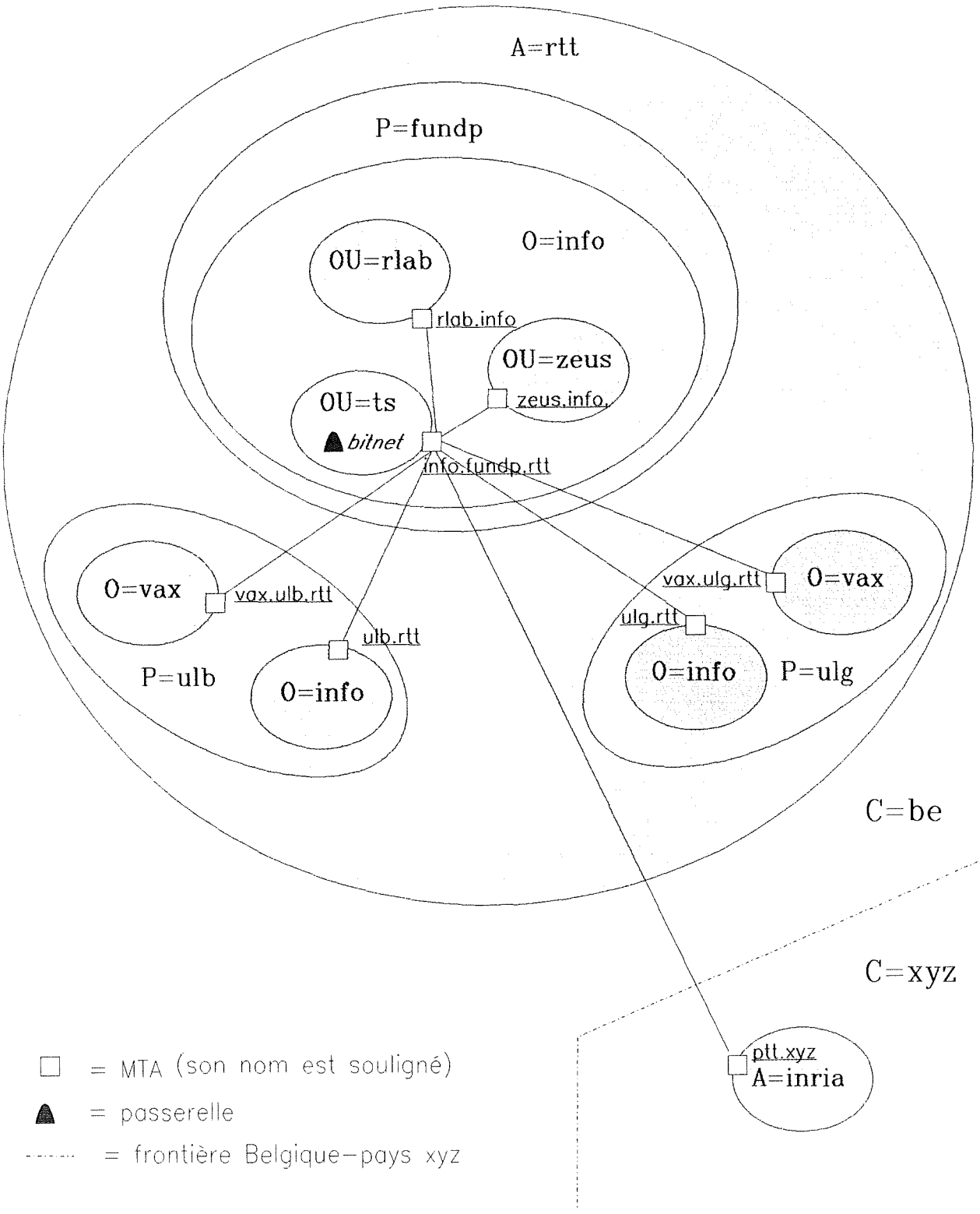


Figure 3.2 : exemple fictif pour EAN version DFN

Les remarques suivantes sont relatives à la figure 3.2:

1°) "Bitnet" est une passerelle vers le réseau Bitnet.

2°) Le MTA possède 2 accès au PRMD "P=ulb;A=rtt;C=be": l'accès doit se faire normalement par le MTA "ulb.rtt", sauf si on désire accéder directement à l'organisation "vax" du domaine "P=ulb;A=rtt;C=be", l'accès se faisant dans ce cas par le MTA "vax.ulb.rtt".

Il en est de même pour les MTAs "ulg.rtt" et "vax.ulg.rtt".

3°) L'accès à tous les PRMDs et ADMDs du pays "C=xyz" se fait par le MTA "ptt.xyz"

3.2.1. La User Agent Table

La **User Agent Table** comprend pour chaque utilisateur enregistré sur le MTA:

- le nom de son UA (=son "userid");
- le nom de sa queue file (généralement le même que son userid);
- le nom du work program à exécuter quand du courrier arrive pour cet utilisateur;

Exemple:

UA	Queue file	Work Program
xru	xru	/usr/local/lib/ean/auto xru
jva	jva	/usr/local/lib/ean/auto jva
bdt	bdt	/usr/local/lib/ean/auto bdt
postmast	postmast	/usr/local/lib/ean/auto postmast

3.2.2. La MTA Table

La **MTA Table** comprend pour chaque MTA spécifié dans l'Organization Table ou dans la Domain Table:

- le nom de ce MTA;
- le nom de sa queue file;
- un drapeau indiquant si oui ou non le MTA utilise des DDAs dans les O/R names (si le MTA fait partie d'une ancienne version de EAN (version UBC1), alors cette zone vaut 'y' (yes), sinon (EAN UBC2 ou DFN), elle est laissée vide);
- des informations diverses utilisées par le RTS pour effectuer la connexion avec le MTA (même syntaxe que pour la version UBC1).

Exemple:

MTA	Queue file	DDAs	Network Information
rlab.info	_rlab.info		ttxp,rlab.info,ean
zeus.info	_zeus.info		inet,zeus.info,ean
ulb.rtt	_ulb.rtt	y	x25,02284681140535,,ean1
vax.ulb.rtt	_vax.ulb.rtt		x25,02829839839739,,ean1
ulg.rtt	_ulg.rtt	y	x25,02120929029202,,ean1
vax.ulg.rtt	_vax.ulg.rtt		x25,02192093083038,,ean1
ptt.xyz	_ptt.xyz		x25,020802902292092,,ean1
.local	_local		nsg,a822,sendmail

Ce tableau reprend tous les MTA spécifiés dans l'Organization Table et la Domain Table décrites ci-dessous.

3.2.3. La Local Identification Table

La **Local Identification Table** comprend le nom du MTA et le nom de l'organisation, du Prmd, de l'Admd et du pays auquel il appartient.

Remarque:

Le nom de l'organisation comprend obligatoirement l'ensemble des attributs "OU" et l'attribut "O". Il est de la forme "OU1.OU2. ... .OUn.O"[1].

Cependant, si le MTA appartient à un site n'utilisant que des "OUs" et pas de "O", alors le nom de l'organisation comprend un "O" vide et est donc de la forme "OU1.OU2. ... .OUn." (notez le point terminal !).

Exemple:

Local MTA name	Local Organization	Prmd	Admd	Country
info.fundp.rtt	ts.info	fundp	rtt	be

3.2.4. L'Organization Table

L'**Organization Table** comprend pour chaque organisation du domaine du MTA et accessible (directement ou non) par le MTA:

- le nom de cette organisation (voir remarque à la section 3.2.3.);
- le nom du MTA à utiliser pour joindre cette organisation (si l'organisation est directement accessible, il s'agit d'un MTA de cette organisation; sinon, il s'agit d'un MTA "relais").

Il est également possible d'ajouter des entrées concernant des organisations d'autres domaines (pourvu que ces domaines soient dans la Domain Table) afin de spécifier un routage direct particulier entre deux organisations bien précises de deux domaines différents.

Remarques:

1°) Le nom de MTA "@local" signifie qu'il s'agit du MTA local (et que le message est arrivé au MTA destinataire).

Le nom de MTA peut être vide si l'organisation n'est accessible par aucun MTA (et ne possède donc aucun UA).

2°) Si une organisation dont le nom est de la forme "OU1.OU2.OU3 ...OUn-1..OUn.O" ($n \geq 0$) possède une entrée dans la table, alors la table doit également comprendre les entrées suivantes:

1. A proprement parler, le nom de l'organisation est donné par l'attribut "O" et les attributs "OUs" donnent le nom des unités organisationnelles. Le manuel EAN utilise ici le terme "nom d'organisation" pour désigner l'ensemble des "OUs" plus "O".

- "OU2.OU3. .... OUn-1.OUn.O"
- "OU3. .... .OUn-1.OUn.O"
- ...
- "OUn-1.OUn.O"
- "OUn.O"
- "O"

Si une organisation dont le nom est de la forme "OU1.OU2.OU3 ...OUn-1. .OUn." ($n \geq 1$) possède une entrée dans la table, alors la table doit également comprendre les entrées suivantes:

- "OU2.OU3. .... OUn-1.OUn."
- "OU3. .... .OUn-1.OUn."
- ...
- "OUn-1.OUn."
- "OUn."
- "" (entrée vide, correspondant à "O = <vide>")

On appelle **entrée d'ordre n** une entrée de l'Organization Table dont le nom comprend soit n attributs OUs, soit n-1 attributs OUs et 1 attribut O.

On voit ici que chaque organisation doit au moins posséder une entrée d'ordre 1, c.-à-d. de type "O" ou "OUn." (mais le nom de MTA correspondant peut être vide si cette organisation ne correspond à aucun MTA).

3°) Il est possible qu'un même nom d'organisation soit utilisé dans plusieurs PRMDs différents. Pour cela, tous les noms d'organisations n'appartenant pas au PRMD local sont suivis du caractère "&", qui précède le nom du PRMD.

Exemple:

Organization	MTA
info	
ts.info	@local
rlab.info	rlab.info
zeus.info	zeus.info
vax&ulb	vax.ulb.rtt
vax&ulg	vax.ulg.rtt

Les organisations du même PRMD que le MTA sont "info", "ts.info", "rlab.info" et "zeus.info":

- "ts.info", "rlab.info" et "zeus.info" sont trois organisations comprises dans l'organisation englobante "info";
- "ts.info" correspond à l'organisation locale du MTA;
- "rlab.info" est accessible par le MTA "rlab.info";
- "zeus.info" est accessible par le MTA "zeus.info";
- "info" n'est pas accessible (cette entrée est néanmoins nécessaire à cause de la présence des entrées "ts.info", "rlab.info" et "zeus.info", mais le nom de MTA correspondant est une chaîne de caractères vide)

En outre, le MTA peut accéder directement à l'organisation "vax" du PRMD "ulb" et à l'organisation "vax" du PRMD "ulg" [1]. Le MTA utilise pour cela les MTAs "vax.ulb.rtt" et "vax.ulg.rtt".

3.2.5. La Domain Table

La **Domain Table** comprend pour chaque domaine accessible (directement ou non) par le MTA:

- le nom de ce domaine (juxtaposition des attributs PRMD, ADMD et Country, séparés par des points);
- le nom du MTA à utiliser pour joindre ce domaine.

Remarque:

On peut utiliser " *" comme nom de PRMD, d'ADMD ou de pays (Country). Le PRMD, ADMD ou pays correspondant est alors quelconque (lors du routage, on cherchera d'abord si possible une entrée sans " *").

Exemple:

Domain	MTA
fundp.rtt.be	
bitnet.rtt.be	.local
ulb.rtt.be	ulb.rtt
ulg.rtt.be	ulg.rtt
xyz	ptt.xyz

Les domaines accessibles sont "fundp.rtt.be", "bitnet.rtt.be", "ulb.rtt.be", "ulg.rtt.be" et tous les domaines du pays "xyz".

- "fundp.rtt.be" n'est pas accessible (pas de MTA ni d'UAs);
- "bitnet.rtt.be" est un PRMD accessible par une passerelle qui se trouve dans le MTA local;
- les PRMDs "ulb.rtt.be" et "ulg.rtt.be" sont accessibles par les MTAs "ulb.rtt" et "ulg.rtt";
- tous les PRMDs et ADMDs du pays "xyz" sont accessibles par le MTA "ptt.xyz".

3.2.6. L'algorithme de routage

Etant donné les tables du MTA, l'algorithme de routage va de soi:

1°) Il examine d'abord l'Organization Table, afin d'essayer d'effectuer un "**routage au niveau des organisations**" (l'Organization Table spécifie à quels MTAs envoyer les messages pour certaines organisations).

1. "directement" signifie ici: sans passer par les MTAs relais prévus normalement pour l'accès aux PRMDs "ulb" et "ulg" (à savoir: les MTAs "ulb.rtt" et "ulg.rtt").

2°) Si aucune entrée dans l'Organization Table ne convient, alors l'algorithme examine la Domain Table, afin d'essayer d'effectuer un "**routage au niveau des domaines**" (la Domain Table spécifie à quel MTA envoyer les messages pour les PRMDs externes[1]).

3°) Si aucune entrée ne convient dans la Domain Table, alors l'algorithme s'arrête sur un échec.

Exemples:

1°) "S=user1;OU=rlab;O=info;P=fundp;A=rtt;C=be":

Le routage utilise l'entrée "rlab.info" de l'Organization Table; le MTA correspondant a pour nom "rlab.info".

2°) "S=user2;O=vax;P=ulb;A=rtt;C=be":

Le routage utilise l'entrée "vax&ulb" de l'Organization Table; le MTA correspondant a pour nom "vax.ulb.rtt".

3°) "S=user3;O=info;P=ulb;A=rtt;C=be":

Aucune entrée adéquate n'est trouvée dans l'Organization Table. La Domain Table est alors examinée et le routage utilise l'entrée "ulb.rtt.be"; le MTA correspondant a pour nom "ulb.rtt".

4°) "S=user4;O=sorbonne;P=archimede;A=egh;C=xyz":

Aucune entrée adéquate n'est trouvée dans l'Organization Table. La Domain Table est alors examinée et le routage utilise l'entrée "xyz"; le MTA correspondant a pour nom "ptt.xyz".

3.2.7. Le contrôle et la comptabilisation du routage

Signalons pour être complet que dans la version DFN de EAN, il est possible à l'administrateur du système de spécifier quels messages peuvent être relayés par le MTA, et quels messages ne peuvent pas l'être (par exemple: il est possible de spécifier que, par défaut, les utilisateurs n'ont accès à aucun pays étranger, sauf tel et tel utilisateur...).

Il est également possible de comptabiliser les messages traités par le MTA.

Ces fonctionnalités n'ont pas été testées et n'ont pas été implémentées aux Facultés Universitaires Notre-Dame de la Paix. Le lecteur intéressé trouvera plus d'informations à leur sujet dans [EANDFN], pages 24 à 43.

1. si l'Organization Table ne spécifie pas un autre MTA.

Chapitre 4 : Les répertoires et les fichiers de EAN

Ce chapitre décrit les principaux répertoires de EAN, ainsi que certains de leurs fichiers. Il est basé essentiellement sur [EANUBC], chapitre 3, et sur l'étude personnelle de EAN.

4.1. Les différents répertoires de EAN

Les principaux répertoires utilisés par EAN sont les suivants:

/usr/local et **/usr/local/lib/ean** : contiennent les programmes exécutables (décrits dans la section 4.3.).

/usr/spool/ean/help : contient les fichiers d'aide (d'extension "**.hep**" et "**.hpi**") utilisés par certains programmes.

/usr/spool/ean/log : contient les fichiers "**log**", c.-à-d. des fichiers contenant des informations sur l'exécution des différents programmes de EAN.

/usr/spool/ean/sys : contient des fichiers nécessaires à EAN et propres au système sur lequel il est installé (par exemple les tables de routage, les messages attendant d'être livrés, ...).

/usr/ean[1] : contient un ensemble de sous-répertoires, dont:

src : contient les sources de EAN[2];

misc : contient des programmes divers utiles à EAN (par exemple, un éditeur Emacs pour l'édition des messages, une interface à X.25).

De plus, chaque utilisateur de EAN possède un répertoire spécifique où se trouvent les données EAN propres à cet utilisateur (profile, messages et folders). Ce répertoire est appelé "**ean**" et se trouve localisé juste sous le répertoire de base de l'utilisateur. Nous le désignerons par le terme "**répertoire EAN de l'utilisateur**". Son contenu est décrit dans 4.2..

4.2. Le répertoire EAN de l'utilisateur

Le répertoire EAN de l'utilisateur comprend les fichiers suivants (voir [EANUBC], section 3.1 (page 16)):

- **profile@** : le fichier "profile" de l'utilisateur (fichier texte, modifiable soit par le programme **ean**, soit par un éditeur de texte ordinaire).

---

1. Remarque: l'administrateur de EAN peut utiliser un autre répertoire que **/usr/ean** (exemple: **/usr/users/ean**). Dans ce cas, ses sous-répertoires sont modifiés en conséquence (exemple: **/usr/ean/src** devient **/usr/users/ean/src**).

2. - Ce répertoire comprend plusieurs sous-répertoires; les sous-répertoires "**host**" contiennent des fichiers qui peuvent être modifiés selon le système sur lequel EAN est installé.
- EAN est écrit en langage C; le lecteur non familié avec ce langage peut consulter [KR].

- **.notify** : ce fichier ne comprend aucune donnée. La seule information associée à ce fichier est le moment de sa dernière réécriture, qui correspond toujours au dernier moment où l'utilisateur a lu le contenu de sa boîte aux lettres (car, lorsqu'il ouvre sa boîte aux lettres, son fichier **.notify** est réécrit).

  Par conséquent, le fichier **.notify** permet de déterminer si l'on a lu ou non les nouveaux messages arrivés: si le moment de la dernière réécriture de ce fichier est postérieur au moment de la dernière réécriture du fichier contenant les messages (**content@**), alors tous les messages ont été lus; sinon de nouveaux messages sont arrivés et n'ont pas encore été lus.

- **content@** : ce fichier contient les messages (chaque message n'est sauvé qu'une seule fois). L'espace de ce fichier est décomposé en "segments".

- **content@.map** : ce fichier contient la lite des segments libres et occupés par les messages dans le fichier **content@**.

- **headers@** : ce fichier contient, pour chaque message contenu dans **content@**, une référence à ce message ainsi qu'une copie de son header. Les fichiers référençant les messages feront en réalité référence à une entrée de ce fichier, et non directement à une entrée du fichier **content@**. Les headers sont ici sauvés sous une forme beaucoup plus efficace à utiliser par les programmes de EAN que dans le fichier **content@**.

- **direct@.dir** et **direct@.pag** : ces fichiers contiennent des informations sur les différents folders de l'utilisateur.

- **map@.dir** et **map@.pag** : ces fichiers contiennent les informations nécessaires pour convertir un numéro identifiant un message en un nom de folder et un emplacement dans ce folder.

- les folders de l'utilisateur, dont "**inbox**". Ces folders ne comprennent pas les messages mais des références à ceux-ci (plus précisément, des références aux entrées du fichier **header@**).

4.3. Les différents programmes exécutables de EAN

Cette section décrit brièvement les fonctionnalités et les buts des principaux programmes exécutables de EAN. La plupart de ces programmes se retrouvent dans toutes les versions de EAN.

Le lecteur peut se référer par exemple à [EANUBC] pages 18 à 29 pour plus de précisions quant à la syntaxe des paramètres de ces programmes. Certains paramètres sont cependant communs à plusieurs de ces programmes:

- Le paramètre **-t<nombre>** permet de spécifier la quantité d'informations affichées à l'écran lors du déroulement du programme. Celle-ci va de <nombre> = 5 (très peu d'informations: uniquement les erreurs graves) à <nombre> = 10 (beaucoup d'informations).

- Le paramètre **-p<nombre>** est similaire au paramètre **-t<nombre>**, mais les informations sont écrites non pas à l'écran mais dans un fichier du répertoire **/usr/spool/ean/log**.

4.3.1. Auto

Lorsqu'un message arrive dans la queue file d'un UA, le work program de cet UA est exécuté. Généralement, le work program est le programme **auto**. Celui-ci effectue la livraison à l'UA:

1°) Il va lire dans le profile de l'utilisateur le nom de sa boîte aux lettres (ou de la boîte aux lettres remplaçant celle de l'utilisateur si celui-ci utilise l'option "auto-forward"[1]).
2°) Il va ensuite placer le message dans cette boîte aux lettres.
3°) Si l'utilisateur a exécuté la commande "biff y", il avertit celui-ci.
4°) Si l'utilisateur utilise l'option "auto-reply"[2], il renvoie automatiquement la réponse spécifiée dans le paramètre de "auto-reply".

Remarque:
- le programme **auto** appelle le programme **autoua** pour exécuter les commandes "auto-forward" et "auto-reply";
- le programme auto n'est normalement exécuté que par le MTA.

4.3.2. Dist

Le programme **dist** est le work program ayant pour but de rediriger tout le courrier arrivant dans une boîte aux lettres vers une série d'autres boîtes aux lettres, spécifiées dans un fichier. Il n'est normalement exécuté que par le MTA.

4.3.3. Ean

Le programme **ean** est le principal programme utilisé par l'utilisateur normal. C'est le programme interface avec l'utilisateur qui permet à ce dernier de composer, d'envoyer, de recevoir et de répondre à des messages. Une aide "on-line" est disponible (commande "help").

4.3.4. Eancheck

Le programme **eancheck** permet à un utilisateur de vérifier s'il a reçu de nouveaux messages. Pour cela, il compare la date et l'heure de la dernière modification du fichier "**.notify**" et celle du fichier "**content@**" (dans le répertoire EAN de l'utilisateur). Si la date/heure du fichier "**.notify**" précède celle du fichier "**content@**", cela signifie que du nouveau courrier est arrivé et n'a pas été lu.

4.3.5. Eanrebuild

Le programme **eanrebuild** permet à un utilisateur de compacter tous ses folders.

Lorsque l'utilisateur exécute la commande "**delete**" pour effacer un message, ce dernier n'est pas physiquement effacé: il est simplement marqué "**deleted**". L'utilisateur a d'ailleurs la possibilité de récupérer ce message à l'aide de la commande "**undelete**".

Le programme eanrebuild permet de supprimer physiquement tous les messages marqués "**deleted**". Il permet en outre de vérifier la cohérence des fichiers se trouvant dans le répertoire EAN de l'utilisateur.

---

1. L'option "auto-forward" est utilisée pour que tous les messages arrivant dans une boîte aux lettres soient redirigés vers une autre boîte aux lettres (cela équivaut au changement d'adresse dans le système postal traditionnel).

2. L'option "auto-reply" est utilisée pour renvoyer automatiquement une même réponse donnée aux expéditeurs des messages arrivant dans une même boîte aux lettres.

4.3.6. Gweansm

Gweansm est un programme utilisé par les passerelles vers des réseaux non-X.400, mais supportant l'adressage RFC822 (voir [EANUBC], section 4.5). Il remplace dans ce cas le programme **txp**.

4.3.7. Helpcompile

Les messages d'aide à l'utilisateur se trouvent dans des fichiers texte d'extension "**.hep**", normalement dans le répertoire **/usr/spool/ean/help**. EAN ne sait pas utiliser directement ces fichiers, il a besoin en plus de fichiers d'index ayant l'extension "**.hpi**".

Le programme **helpcompile** permet de créer un fichier d'index "**.hpi**" à partir d'un fichier texte d'aide "**.hep**".

Il est donc possible à l'utilisateur de modifier lui-même l'aide "on-line", par exemple pour clarifier des points obscurs dans cette aide ou pour la traduire dans une autre langue. Il lui suffit d'éditer les fichiers texte "**.hep**"[1] puis de créer les nouveaux fichiers index "**.hpi**" à l'aide du programme **helpcompile**.

4.3.8. Mailer

Le programme **mailer** lit depuis son entrée standard (stdin) un message encodé selon le format ARPA RFC822; il le convertit au format CCITT P2 et l'envoie ensuite à une liste de recipients X.400. Cette liste peut être soit la liste des recipients présents dans la partie "Heading" du message P2, soit une liste donnée comme paramètre.

4.3.9. Mtamaint

Le programme **mtamaint** est le principal programme utilisé par le gestionnaire de EAN. Il permet de construire les tables du MTA, de contrôler l'exécution du programme **mtamngr** (voir 4.3.10.) et de vérifier le contenu et la cohérence des "message files" et des "queue files". Une aide "on-line" est disponible, expliquant les nombreuses commandes de ce programme (commande "help").

4.3.10. Mtamngr

Le programme **mtamngr** est le programme permettant de créer le processus permanent "**MTA Manager**". Ce processus est averti par IPC ("InterProcess Communication") chaque fois qu'un message doit être envoyé à un autre MTA. Il lance alors le programme **txp**[2] pour établir la connexion avec ce MTA et permettre ainsi l'échange de messages.

Lorsque le MTA Manager est lancé, il s'initialise comme suit:

1°) Il lit la liste des MTAs connus (dans la table des MTAs).

2°) Il lit le fichier **/usr/spool/ean/sys/mta/mmcfg.txt**. Ce fichier contient des informations relatives aux connexions avec les différents MTAs[3], par exemple:

---

1. La syntaxe de ces fichiers texte est donnée par exemple dans [EANUBC], section 3.3.6.
2. sauf spécification contraire
3. Si ce fichier n'existe pas, le MTA Manager utilise des paramètres par défaut.

```
@configure
        maxcalls = 3
        txpfile = /usr/local/lib/ean/txp
        txpargs = txp -l
        expiry = 21 d
        retry =     1 m, 2 m, 5 m, 10 m, 30 m,
                    1 h, 2 h, 5 h, 10 h, 1 d

.local
        retry = 7 m
        txpfile = /usr/local/lib/ean/gweansm
        txpargs=gweansm -ilp12 %s

ts.info
        poll = 12 h
        expiry = 29 d 15 h 10 m 4 s
```

Les renseignements dans la partie "**@configure**" s'appliquent à tous les MTAs. Les renseignements dans la partie "**ts.info**" s'appliquent uniquement au MTA "**ts.info**", ceux dans la partie "**.local**" s'appliquent au MTA "**.local**". Tous ces renseignements sont indiqués comme suit:

maxcalls = nombre maximal de processus que le MTA Manager peut lancer à la fois;
txpfile = processus lancé pour atteindre le MTA;
txpargs = arguments passés lors du lancement du processus **txpfile**;
expiry = intervalle de temps après lequel EAN peut supprimer les messages à destination du MTA, si celui-ci est toujours inaccessible[1];
retry = intervalles de temps séparant les essais successifs de connexion avec le MTA (en cas d'échecs);
poll = intervalle de temps après lequel le MTA est appelé, qu'il y ait ou non des messages pour lui (dans le cas d'un MTA à appeler à intervalles réguliers).

3°) Il lit le fichier **/usr/spool/ean/sys/mmstats.key** (ce fichier comprend des informations sur les statistiques d'essais de connexions).

4°) Il examine l'état des queue files des MTAs. Si certains messages sont à envoyer, il lance pour chaque MTA destinataire un processus **txp** (voir sections 4.3.15. et 4.4.).

Remarques:

1°) **Mtamngr** ne doit être exécuté qu'une seule fois, lors du démarrage du système (à spécifier dans le fichier **/etc/rc.local**).

2°) Si l'on modifie des informations dans la table des MTAs, il est nécessaire de redémarrer le MTA Manager pour qu'il en tienne compte:

1. Cette option n'est disponible que dans la version EAN DFN. Dans les précédentes versions, cet intervalle de temps est toujours égal à 1 semaine.

% mtamaint
MTA > manager
MTAmngr > set restart
MTAmngr > exit
MTA > exit

4.3.11. Nameserver

Le programme **nameserver** est le work program utilisé par l'UA **"nameserver"**.

Lorsqu'un utilisateur exécute une commande drop, install, find ou register pour accéder au Directory Service, un message est envoyé à l'UA **"nameserver"** de son système, contenant la requête de l'utilisateur. Le work program de cet UA est normalement le programme **nameserver**, qui traite la requête de l'utilisateur et lui répond.

Le programme **nameserver** est en réalité un des deux programmes suivants (qui a été renommé en "**nameserver**"): **nscentral** ou **nsrelay**.

La base de données de l'organisation est centralisée sur un seul système. Dans le cas où l'on se trouve sur ce système, le work program **nameserver** est **nscentral**: c'est lui qui gère la base de données et accède directement à ses informations pour répondre aux différentes requêtes. En revanche, si l'on se trouve sur un autre système, le work program **nameserver** est **nsrelay**. Ce dernier a simplement pour but de renvoyer toute requête dans la boîte aux lettres **"nameserver"** du système comprenant réellement la base de données (la boîte aux lettres dont le work program est **nscentral**).

4.3.12. Netserve

Le programme **netserve** permet de surveiller les appels EAN arrivant au système par TCP/IP et par X.25. Lorsqu'un tel appel survient, le programme **netserve** appelle le programme **txp** (en mode "slave": voir section 4.3.15.) avec les paramètres appropriés pour établir la connexion avec le MTA appelant et permettre ainsi l'échange de messages. Puis, le programme **netserve** recommence à guetter les appels arrivant.

Tout comme **mtamngr**, le programme **netserve** est normalement résident et ne doit être exécuté qu'une seule fois, lors du démarrage du système (à spécifier dans le fichier **/etc/rc.local**).

4.3.13. Nscentral et nsrelay

Voir le programme **nameserver**, section 4.3.11..

4.3.14. Transin et transout

Le programme **transin** permet de lire les nouveaux messages se trouvant dans une boîte aux lettres données et de les écrire tels quels dans sa sortie standard (stdout). Eventuellement, un programme donné peut être exécuté pour chacun de ces messages.

Le programme **transout** est le programme inverse de **transin**: il lit des messages à partir de son entrée standard (stdin) et place tel quel ce qu'il a lu dans une boîte aux lettres donnée.

Les programmes **transin** et **transout** agissent donc comme des "pipes" permettant de lire et écrire des messages dans des boîtes aux lettres, sans jamais interpréter ou modifier le contenu de ces messages.

4.3.15. Txp

Le programme **txp** permet d'établir une connexion avec un autre MTA et d'échanger des données avec ce dernier. Lorsque la connexion est établie, tous les messages à destination de ce MTA sont envoyés, puis tous les messages que cet autre MTA voudrait envoyer sont reçus.

Txp peut être lancé soit mode "**maître**" ("**master**"), pour avoir l'initiative d'envoyer des messages, soit en mode "**esclave**" ("**slave**"), pour commencer d'abord à recevoir des messages. Lorsque tous les messages ont été communiqués dans un sens, chacun des deux programmes **txp** change alors de mode et les messages sont envoyés dans l'autre sens. La syntaxe du mode "master" est la suivante: **txp <nom_de_mta>**; celle du mode "slave" est: **txp -s <type_de_réseau>** (où **type_de_réseau** est le réseau par où survient l'appel, par exemple **inet** ou **x25**).

Le programme **txp** est normalement exécuté par le MTA Manager, mais il est possible de l'exécuter manuellement, par exemple pour détecter les causes des erreurs survenant lors d'établissements de connexions ou de transferts de données. Ainsi, si la connexion avec le MTA "zeus.info" ne fonctionne pas correctement, on peut étudier ce qui se passe en lançant:

txp -t10 zeus.info

4.3.16. X9scan

Le programme **x9scan** lit un fichier respectant la syntaxe X.409 et en affiche le contenu sous une forme lisible par l'utilisateur. Il est capable d'interpréter les protocoles P1 et P2. Une aide "on-line" est disponible (commande "help").

4.4. L'enchaînement des programmes utilisés pour l'envoi et la réception de messages

4.4.1. L'envoi d'un message

Lorsqu'un message est créé par un processus **ean**, il est placé dans un message file et les queue files sont mis à jour en fonction des destinataires du message. Si le message est destiné à un UA local, le work program de cet UA est simplement lancé. Si, en revanche, le message doit être envoyé à des MTAs éloignés, alors le processus **ean** utilise les IPC pour communiquer au **MTA Manager** à quels MTAs il faut envoyer des messages.

Pour chacun de ces MTAs, le **MTA Manager** lance alors un processus **txp**, en mode "master", avec le nom du MTA à appeler.

Chacun de ces processus **txp** effectue les opérations suivantes:

- il établit une session avec l'autre MTA (d'après les informations de connexion se trouvant dans la table des MTAs);
- il envoie tous les messages qu'il a pour ce MTA;
- il passe en mode "slave" (le processus **txp** correspondant sur l'autre MTA passe en mode "master");
- il reçoit tous les messages que l'autre MTA avait à lui envoyer;
- il traite chacun de ces nouveaux messages (lancement d'un work program ou appel du MTA Manager).

4.4.2. La réception d'un message

La réception d'un message est détectée différemment selon que le message arrive par TTXP ou par TCP/IP ou X.25.

Lorsqu'un appel arrive par TCP/IP ou par X.25, il est détecté par le processus permanent **netserve**. Celui-ci lance alors un processus **txp** en mode "slave", avec comme paramètre le type du réseau sur lequel l'appel est survenu (**inet** (TCP/IP) ou **x25** (X.25)). Ce processus s'occupera d'établir alors la communication avec le MTA appelant (plus exactement, avec son processus **txp**).

L'arrivée d'un appel par TTXP est détectée différemment: l'appel se traduit par le "login" d'un utilisateur appelé "eannet". Or, en configurant EAN, le programme lancé automatiquement lors du "login" de cet utilisateur n'est pas un shell classique, mais le programme **txp**.

Dans les deux cas, le programme **txp** est donc lancé. Il effectue alors les opérations suivantes:

- il établit la communication avec le processus **txp** appelant;
- il reçoit tous les messages que le MTA appelant veut lui envoyer;
- il traite chacun de ces messages (lancement d'un work program ou appel du MTA Manager);
- il bascule en mode "master" (et le MTA appelant bascule en mode "slave");
- il envoie tous les messages qu'il a pour ce MTA.

Note: la fonction MTA n'est pas réalisée dans EAN par un seul programme: elle est réalisée par un ensemble de procédures appelées par plusieurs programmes (ean, netserve, txp, MTA Manager).

Chapitre 5 : L'implémentation concrète aux FUNDP

Ce chapitre décrit la configuration EAN initiale et les interconnexions des trois systèmes Unix **Junon**, **Alma** et **Zeus**, aux Facultés Universitaires Notre-Dame de la Paix. Il décrit ensuite les améliorations que nous avons tenté d'y apporter.

5.1. Situation initiale (septembre 1988)

Nous nous sommes concentrés sur 3 systèmes Unix présents en septembre 1988:

1°) **Junon** : Unix BSD 4.3 tournant sur VAX 750;
2°) **Alma** : Unix BSD 4.3 tournant sur VAX 750;
3°) **Zeus** : Ultrix-32 V3.0 tournant sur VAX 3600.

En outre, 4 stations de travail ("**ws1**" à "**ws4**") peuvent accéder au système de fichiers "**/usr/users**" de Zeus, grâce au **NFS (Network File System** - voir section 5.1.1.).

Ces systèmes sont tous actuellement reliés au PABX et à une même liaison Ethernet.

Le logiciel EAN Version DFN était déjà implémenté sur Junon et sur Alma, mais sans Directory Service. Une connexion TTXP existait entre ces deux implémentations de EAN.

Les attributs X.400 associés au MTA de Junon sont: **OU=ts;O=info;P=fundp;A=rtt;C=be**. Ceux associés au MTA de Alma sont: **OU=rlab;O=info;P=fundp;A=rtt;C=be**. Le MTA de Junon s'appelle "**info.fundp.rtt**", celui de Alma s'appelle "**rlab.info**".

Remarque importante:

Initialement, sur Junon, EAN a été compilé sous Unix BSD 4.2. Par après, la version BSD 4.3 de Unix a été installée sur Junon. Malheureusement, si les programmes de EAN fonctionnent toujours sous cette nouvelle version du système d'exploitation, les sources de EAN n'y sont plus compilables. Par conséquent, il est impossible de recompiler EAN sur Junon.

5.1.1. Note 1: le NFS

Le NFS (**Network File System**) est un logiciel permettant de partager des fichiers dans un environnement réseau. Il est basé sur le modèle **client/serveur** (voir section 5.1.2.): le **serveur** est le système contenant les fichiers à partager, les **clients** sont les systèmes désirant accéder à ces fichiers. On dit que le serveur **exporte** des fichiers vers les clients, et qu'un client **importe** des fichiers du serveur.

L'importation de fichiers par un client s'effectue en montant sur le client un système de fichiers ("file system") ou un répertoire du serveur. Le client peut alors accéder aux fichiers de ce système de fichiers ou de ce répertoire comme s'ils étaient locaux.

Aux FUNDP, le serveur est Zeus et les clients sont les 4 stations de travail "ws1" à "ws4". Les fichiers partagés sont principalement ceux du système de fichiers "**/usr/users**" de Zeus. Ce système de fichiers est monté au même emplacement ("**/usr/users**") sur chaque station de travail: les accès au système de fichiers "**/usr/users**" des stations de travail se traduisent en réalité par des accès au système de fichiers "**/usr/users**" de Zeus.

La liste des systèmes de fichiers et des répertoires exportables doit être écrite dans le fichier **/etc/exports** du serveur.

Exemple:

Le fichier **/etc/exports** de Zeus comprend notamment les lignes suivantes:

```
/usr/users
/usr/spool/mail ws1 ws2 ws3 ws4 -r=0
```

Ces lignes signifient que le système de fichiers **/usr/users** peut être exporté vers n'importe quel autre système "remote", tandis que les fichiers de **/usr/spool/mail** ne peuvent être exportés que vers les stations de travail "ws1" à "ws4"[1].

La liste des systèmes de fichiers et des répertoires à monter (y compris à importer) automatiquement lors du démarrage d'un serveur doit être écrite dans le fichier **/etc/fstab** du serveur.

Exemple:

Le fichier **/etc/fstab** de la station de travail "ws3" comprend notamment la ligne suivante:

```
/usr/users@zeus:/usr/users:rw:0:1:nfs::
```

Cette ligne signifie que, lors du démarrage de la station, le système de fichiers "**/usr/users**" est automatiquement importé par NFS de Zeus et monté à l'emplacement local "**/usr/users**"[2].

La liste des systèmes de fichiers et des répertoires montés à un instant donné peut être obtenue en exécutant la commande **df** ou la commande **mount** (sans paramètre). Par exemple, sur "ws3":

---

1. La signification précise de tous les paramètres des lignes du fichier **/etc/exports** est donnée dans tout manuel de programmation UNIX (voir la fonction **export**).

2. La signification précise de tous les paramètres des lignes du fichier **/etc/fstab** est donnée dans tout manuel de programmation UNIX (voir la fonction **fstab**).

```
% df
Filesystem                                  Total    Kbytes   Kbytes    %      Mounted on
node                                        Kbytes   used     free      used   /
zeus:/usr/var/diskless/dclient0/ws3.root    105223   93029    1672      98%    /usr
zeus:/usr/var/diskless/dlenv0/root0.vax/usr 210159   172419   16725     91%    /usr/users
zeus:/usr/users                             123063   108898   1859      98%

% mount
zeus:/usr/var/diskless/dclient0/ws3.root on / type nfs (rw,hard)
zeus:/usr/var/diskless/dlenv0/root0.vax/usr on /usr type nfs (ro,hard)
zeus:/usr/users on /usr/users type nfs (rw,hard)
```

Remarque: Le NFS est décrit de façon complète dans [NFS].

5.1.2. Note 2: le modèle client/serveur

Le **modèle client/serveur** est un modèle s'appliquant à des communications non symétriques: le client envoie une requête au serveur, et le serveur renvoie la réponse.

Le principal avantage de ce modèle est que la relation client-serveur peut être implémentée par un simple appel de procédure (remote) plutôt que par une opération d'E/S ou que par un mécanisme d'interruption. L'envoi d'une requête par le client équivaut à l'appel d'une procédure; l'envoi de la réponse par le serveur équivaut au retour de la procédure, avec les résultats désirés.

Remarques:

1°) En cas de panne du serveur, un mécanisme de timeout empêche le client d'attendre indéfiniment la fin de la procédure remote appelée.

2°) Le modèle client/serveur est expliqué de manière plus détaillée par exemple dans [AT], pages 455 à 460.

5.2. Améliorations envisagées

Par rapport à la situation de septembre 1988, nous avons tenté d'apporter les améliorations suivantes:

1°) Première installation de EAN Version DFN sur Zeus.
2°) Création d'une connexion EAN TTXP entre Zeus et Alma.
3°) Création d'une connexion EAN TCP/IP entre Zeus et Junon.
4°) Correction sur Zeus de l'erreur dans EAN relative au Blind Copy Carbon.
5°) Installation du Directory Service de EAN sur Zeus.
6°) Modification de l'installation de EAN sur Zeus pour qu'il puisse être utilisé par les stations de travail.

Ces différentes tentatives d'amélioration sont décrites dans les sections 5.3. à 5.8.. Les manuels [EANUBC] et [EANDFN] nous ont fortement aidés pour les améliorations 1°), 2°), 3°) et 5°). L'amélioration 4°) résulte de l'étude des sources de EAN. L'amélioration 6°) résulte de l'étude du NFS (voir son manuel: [NFS]).

5.3. Installation de EAN Version DFN sur Zeus

Nous avons installé avec succès EAN sur Zeus. La procédure exacte d'installation est décrite notamment dans [EANUBC] section 4 et dans [EANDFN] section 4. Nous la résumons ci-dessous:

1°) Création d'un nouveau groupe ("**ean**") et de deux nouveaux utilisateurs ("**ean**" et "**eannet**"). Chacun de ces utilisateurs doit appartenir au nouveau groupe "**ean**". Le shell de l'utilisateur "**ean**" est **/bin/csh**, celui de "**eannet**" est **/usr/local/lib/ean/txp**.

Remarque: l'utilisateur "**ean**" est utilisé par le gestionnaire de EAN; l'utilisateur "**eannet**" est utilisé par le MTA.

2°) Ajout d'une entrée dans le fichier **/etc/services** pour le MTA Manager:

```
mtamngr         8002/udp
```

3°) Ajout dans le fichier **/etc/rc.local** des lignes suivantes, pour exécuter automatiquement le MTA Manager lors du démarrage de Zeus:

```
if [ -f /usr/local/lib/ean/mtamngr ] ; then
        /usr/local/lib/ean/mtamngr &
        echo -n 'mtamngr'           > /dev/console
fi
```

4°) Compilation des programmes sources de EAN (nous avons pour cela utilisé la commande **make**, lancée en étant **root**).

5°) Initialisation des tables du MTA (en étant l'utilisateur **ean**, à l'aide du programme **mtamaint**):

```
% /usr/local/lib/ean/mtamaint -c[1]
MTA> add local                      % remplissage de la Local Identification Table

Agent name? zeus.info               % le nom du MTA est "zeus.info"
Organization? zeus.info             % le MTA appartient à l'organisation "zeus.info"

Private Domain? fundp               % le MTA fait partie du PRMD "fundp"
Admin Domain? rtt                   % le MTA fait partie de l'ADMD "rtt"
Country? be                         % le MTA fait partie du pays "be"

MTA> add organization               % ajout d'une organisation
entry name? zeus.info               % organisation "zeus.info"
agent name? @local                  % "zeus.info" est l'organisation locale

MTA> add organization               % ajout d'une autre organisation
entry name? info                    % organisation "info"
agent name? <Carriage Return>       % aucun MTA ne permet d'atteindre
                                      l'organisation "info" (elle ne comprend aucun MTA, seules ses Unités
                                      Organisationnelles comprennent
                                      des MTAs et sont accessibles)
```

1. L'option -c est utilisée lors de la première utilisation de **mtamaint**, pour initialiser les fichiers qui lui sont nécessaires.

```
MTA> add domain                    % ajout d'un domaine
entry name? fundp.rtt.be           % domaine "fundp.rtt.be"
agent name? <Carriage Return>      % le domaine "fundp.rtt.be" n'est pas
                                     accessible (il ne comprend pas de
                                     MTA; seuls sont accessibles les OUs
                                     de ce domaine)
```

Donc, le nouveau MTA de Zeus a pour attributs X.400: **OU=Zeus;O=info;P=fundp;A=rtt;C=be**. Il s'appelle "**zeus.info**".

6°) Exécution du MTA Manager (en étant **root**)

```
% /usr/local/lib/ean/mtamngr
```

5.4. Création d'une connexion EAN TTXP entre Zeus et Alma

Nous avons ajouté avec succès une connexion EAN TTXP entre Zeus et Alma. Les opérations effectuées sont décrites aux sections 5.4.1. et 5.4.2..

5.4.1. Sur Zeus

Les opérations effectuées sur Zeus sont décrites ci-dessous.

A) Modification des tables du MTA

Nous avons ajouté des informations dans les tables du MTA de Zeus, à l'aide du programme **mtamaint**:

i) Dans l'Organization Table (Table des Organisations): nous avons ajouté une entrée correspondant à l'organisation nouvelle pour Zeus:

```
% /usr/local/lib/ean/mtamaint
MTA> add organization
entry name? rlab.info              % nom de l'organisation
agent name? rlab.info              % nom du MTA permettant d'atteindre
                                     l'organisation
```

ii) Dans la MTA Table (Table des MTAs): nous avons ajouté une entrée correspondant au nouveau MTA accessible par Zeus:

```
MTA> add mta
agent name? rlab.info              % nom du nouveau MTA
queue file? _rlab.info             % nom de la queue file du MTA
DDA's? no                          % le MTA appartient à une version
                                     DFN de EAN = > pas de DDAs
network info? ttxp,rlab.info,ean   % informations de connexion
create queue file? y               % la queue file est créée
```

Les informations de connexion sont les suivantes:

- **ttxp** : indique que la connexion est une connexion TTXP;
- **rlab.info** : désigne quelle connexion TTXP utiliser (c.-à-d. à quel ordinateur hôte se connecter par TTXP): voir ligne (*) ci-après;
- **ean** : désigne comment accéder à EAN, lorsque la connexion avec Alma est établie par TTXP: voir ligne (**) ci-après.

B) Modification du fichier /usr/spool/ean/sys/rts/termsites

Il était également nécessaire d'ajouter une entrée dans le fichier **/usr/spool/ean/sys/rts/termsites**, décrivant la manière d'établir la connexion TTXP:

```
rlab.info                                                    (*)
        device=/dev/tty02
        baudrate=9600
        handshake
                "\R\D\R";          "SERVICE? "   ¦ "\R\D\R";    "?"
                "\DAlma\R";   "TED "
        ean                                                          (**)
                "\D\R";          "ogin: "
                "\Deannet\R";        "ssword:"
                "\Dxyz\R";      "Ready\R"; 50
        config=U;C8,,100,60;C8,,,60
```

La liste complète de tous les paramètres possibles et de leur signification précise est donnée par exemple dans [EANUBC], section 4.6. Remarquons simplement ici que:

- La première ligne ("rlab.info") contient le nom de la connexion TTXP, c.-à-d. le nom de l'ordinateur hôte auquel on désire se connecter (elle permet de différencier les connexions TTXP les unes des autres). C'est le nom de cette connexion qui est utilisé dans l'information de connexion pour le MTA rlab.info.

- Les deux lignes suivant la ligne "handshake" décrivent la manière d'établir une connexion avec Alma:
  - **"\R\D\R";** : envoyer un caractère Carriage Return[1] (\R), attendre un moment (\D), puis envoyer un autre Carriage Return (\R);
  - **"SERVICE? "** : attendre la chaîne "SERVICE? ";
  - **¦ "\R\D\R"; "? "** : si la chaîne "SERVICE? " n'arrive pas, alors recommencer en attendant plutôt la chaîne "? ";
  - **"\Dalma\R";** : attendre un moment (\D) et envoyer la chaîne "alma", suivie d'un Carriage Return (\R);
  - **"TED "** : attendre la chaîne "TED" (fin de "CONNECTED").

- Les trois lignes suivant la ligne "ean" décrivent la manière d'établir l'accès avec EAN de Alma, lorsque la connexion avec Alma est déjà réalisée:
  - **"\D\R";** : attendre un moment (\D) et envoyer un Carriage Return (\R);
  - **"ogin: "** : attendre la chaîne "ogin: " (fin de "Login: ");
  - **"\Deannet\R"** : attendre un moment (\D) et envoyer la chaîne "eannet", suivie d'un Carriage Return (\R);
  - **"ssword:"** attendre la chaîne "ssword" (fin de "Password:");
  - **"\Dxyz\R"**; attendre un moment ("\D") et envoyer la chaîne "xyz"[2] suivie d'un

1. En hexadécimal: 0DH.
2. "xyz" n'est pas le vrai mot de passe !

Carriage Return (\R);
- **"Ready\R"; 50** : attendre pendant au maximum 50 secondes la chaîne "Ready" suivie d'un Carriage Return (\R).

Remarque:

Les chaînes de caractères sont attendues pendant au maximum 15 secondes (sauf si une autre valeur est spécifiée, comme dans **"Ready\R"; 50**).

C) Réinitialisation du MTA Manager

Enfin, nous avons réinitialisé le MTA Manager pour qu'il prenne en considération les modifications faites:

```
% /usr/local/lib/ean/mtamaint
MTA> manager
MTAmngr> set restart
MTAmngr> quit
MTA> quit
```

5.4.2. Sur Alma

Des opérations similaires à celles effectuées sur Zeus ont été réalisées sur Alma (modification de l'Organization Table, de la MTA Table et du fichier /usr/spool/ean/sys/rts/termsites, redémarrage du MTA Manager).

5.5. Création d'une connexion EAN TCP/IP entre Zeus et Junon

Nous souhaitions également installer une connexion EAN TCP/IP entre Zeus et Junon. Malheureusement, la partie des sources de EAN supportant TCP/IP n'est pas compilée sur Junon (aucune liaison EAN TCP/IP n'y a d'ailleurs déjà été installée), et il est impossible de recompiler des sources de EAN sur Junon. Nous avons donc dû renoncer à notre objectif.

Cependant, nous décrivons aux sections 5.5.1. et 5.5.2. la marche à suivre si EAN était entièrement installé sur Junon.

5.5.1. Sur Zeus

Les opérations à effectuer sur Zeus sont décrites ci-dessous.

A) Modification des tables du MTA

Des informations doivent être ajoutées dans les tables du MTA de Zeus, à l'aide du programme **mtamaint**:

i) Dans l'Organization Table (Table des Organisations): il faut ajouter une entrée correspondant à l'organisation nouvelle pour Zeus:

```
% /usr/local/lib/ean/mtamaint
MTA> add organization                 % ajout d'une organisation
entry name? ts.info                   % nom de l'organisation
agent name? info.fundp.rtt            %nom du MTA permettant d'atteindre
                                        l'organisation
```

ii) Dans la MTA Table (Table des MTAs): il faut ajouter une entrée correspondant au nouveau MTA accessible par Zeus:

```
MTA> add mta                        % ajout d'un MTA
agent name? info.fundp.rtt          % nom du nouveau MTA
queue file? _info.fundp.rtt         % nom de la queue file du MTA
DDA's? no                           %le MTA appartient à une version
                                      DFN de EAN => pas de DDAs
network info? inet,junon,ean        % informations de connexion
create queue file? y                % la queue file est créée
```

Les informations de connexion sont les suivantes:

- **inet** : indique que la connexion est une connexion TCP/IP (Internet);
- **junon** : désigne le nom de l'ordinateur hôte auquel on désire se connecter par TCP/IP: voir ligne (*) ci-après;
- **ean** : désigne le service à utiliser sur l'ordinateur hôte: voir ligne (**) ci-après.

B) Modification des fichiers /etc/hosts et /etc/services

Il est également nécessaire d'ajouter:

- une entrée dans le fichier **/etc/hosts** donnant le nom et l'adresse de l'ordinateur hôte auquel on désire se connecter (Junon):

```
192.9.200.21    junon                ( *)
```

(l'adresse de l'ordinateur hôte (Junon) est ici "192.9.200.21", son nom est "junon")

- une entrée dans le fichier **/etc/services** décrivant à quel port est assigné le service "ean" et quel protocole est utilisé:

```
ean         8801/tcp                ( **)
```

(le service "ean" est assigné au port "8801" et on y accède par le protocole "tcp"; remarquons que tous les ordinateurs hôtes devront avoir le même numéro de port (et le même protocole) pour le service "ean").)

C) Lancement du programme netserve

Le programme **netserve** doit être automatiquement lancé lors du démarrage de Zeus. Il faut donc:

- ajouter les lignes suivantes dans le fichier **/etc/rc.local** :

```
if [ -f /usr/local/lib/ean/netserve ] ; then
      /usr/local/lib/ean/netserve  /usr/spool/ean/sys/rts/netservices &
      echo -n 'netserve'  > /dev/console
fi
```

- ajouter dans le fichier **/usr/spool/ean/sys/rts/netservices** (utilisé par **netserve**):

```
eaninet
      net =inet
      serv=ean
      file=/usr/local/lib/ean/txp
      args=txp -s inet
```

Ces informations signifient que si le programme détecte un appel "inet" (c.-à-d. par TCP/IP), alors il faut lancer le programme **txp** en mode esclave ("slave": **-s**) avec comme paramètre "inet".

- démarrer manuellement le programme **netserve**:

```
% /usr/local/lib/ean/netserve  /usr/spool/ean/sys/rts/netservices
```

D) Réinitialisation du MTA Manager

Enfin, il est nécessaire de réinitialiser le MTA Manager pour qu'il prenne en considération les modifications faites:

```
% /usr/local/lib/ean/mtamaint
MTA > manager
MTAmngr > set restart
MTAmngr > quit
MTA > quit
```

5.5.2. Sur Junon

Des opérations similaires à celles effectuées sur Zeus doivent être réalisées sur Junon (modification de l'Organization Table, de la MTA Table, des fichiers /etc/hosts, /etc/services, /usr/spool/ean/sys/rts/netservices et /etc/rc.local, redémarrage du MTA Manager).

Remarquons que le service "ean" doit être assigné au même port sur Junon et sur Zeus (ici: 8801).

5.6. Correction sur Zeus d'une erreur dans EAN relative au Blind Copy Carbon

Dans la version de EAN disponible aux Facultés, l'option bcc (Blind Copy Carbon) ne respecte pas ses caractéristiques théoriques. Rappelons qu'en principe, les recipients 'bcc' reçoivent le message, comme les recipients 'to' ou les recipients 'cc', mais il y a une différence essentielle qui est la suivante: les recipients 'to' et 'cc' ignorent l'identité des recipients 'bcc', s'il y en a. Prenons un exemple:

Admettons que le message soit destiné aux recipients rto, rcc1, rcc2, rbcc1 et rbcc2: le message envoyé à rto contient l'identité de rto, rcc1 et rcc2

rcc1	rto, rcc1 et rcc2
rcc2	rto, rcc1 et rcc2
rbcc1	rto, rcc1, rcc2 et rbcc1
rbcc2	rto, rcc1, rcc2 et rbcc2

Nous avons vu que, dans le contenu du message, figurent une liste des 'to', une liste des 'cc' et une liste des 'bcc'. Pour remplir la fonctionnalité 'blind copy carbon', il faut donc, avant d'envoyer le message (quel que soit le recipient auquel il est destiné), en extraire de son contenu la liste des 'bcc'. Or, en l'occurrence, les recipients 'bcc' sont apparemment traités exactement comme des recipients 'cc'.

Examinons ce qui se passe dans le programme EAN:

fonction SendCommand() (dans le fichier ua/cmdsend.c)

Construction de la liste de tous les recipients, **reclist**:

```
  .
  .
  .
reclist = NULL;

                {initialisation de la liste de tous les recipients}

reclist=Bld_reclist(reclist,
          P2_find_hdng(Draft->content,P2_RECIPS),
          rep_type,reply, replace, &to_list );

                {ajout des recipients 'to' à la liste reclist}

reclist = Bld_reclist( reclist,
          P2_find_hdng(Draft->content,P2_COPY_RECIPS),
          rep_type, reply, replace, &cc_list );

                {ajout des recipients 'cc' à la liste reclist}


reclist = Bld_reclist( reclist,
          P2_find_hdng(Draft->content,P2_BCOPY_RECIPS),
          rep_type, reply, replace, &bcc_list );

                {ajout des recipients 'bcc' à la liste reclist}

  .
  .
  .
rc = UAL_send( Draft, types, TRUE, TRUE, after?&aft_time:NULL, reclist );
```

fonction UAL_send(m, types, convert, alternate, defer, recip) (fichier ua/ual.c)

(à ce moment, recip est une liste contenant tous les recipients)

```
  .
  .
  .
if( !recip )
     { if( e=P2_find_hdng(m->content,P2_RECIPS) )
          recip = Ecpy( e );
       if( e=P2_find_hdng(m->content,P2_COPY_RECIPS) )
          Seq_add( &recip, Edup(e->constructor) );
       bcc = P2_ext_hdng( m->content, P2_BCOPY_RECIPS );
     }
else
     bcc = NULL;
  .
  .
  .
```

Ce qui s'interprète comme ceci:

- si recip est une liste vide (ce n'est pas le cas qui se présente ici, mais la fonctionnalité de UAL_send le prévoit), alors après exécution de ces instructions, recip sera une liste des recipients 'to' et 'cc'; bcc sera une liste des recipients 'bcc', qui ont été SUPPRIMES du contenu du message (et n'apparaîtront donc pas chez les destinataires).

- si recip n'est pas une liste vide, alors après exécution de ces instructions, recip sera inchangé et bcc sera une liste vide.

C'est à ce point-ci qu'il y a réellement un problème: en effet dans le cas où recip était une liste de tous les recipients (c'est à dire, y compris des 'bcc'), la liste bcc sera, après exécution des instructions, vide. En outre, les recipients 'bcc' n'auront pas été retirés du message.

```
.
.
.
if ((recip = ... ) != NULL)       {si la liste recip n'est pas vide}
      {
            .
            .   (traitement des recipients 'to')
            .
      }
.
.
.
if (bcc != NULL)                  { si la liste bcc n'est pas vide}
      {
            .
            .   (traitement des bcc)
            .
      }
.
.
.
```

Solution proposée:

Il faudrait, dans la fonction UAL_send, forcer la procédure à construire elle-même la liste recip et bcc, que la liste recip reçue en argument soit vide ou non. En effet, cette liste est inutilisable car les recipients 'to' et 'cc' ne sont pas distinguables des recipients 'bcc'. Il faudrait donc écrire, en lieu et place de la suite d'instructions ci-avant:

```
.
.
.
    { if( e=P2_find_hdng(m->content,P2_RECIPS) )
          recip = Ecpy( e );
      if( e=P2_find_hdng(m->content,P2_COPY_RECIPS) )
          Seq_add( &recip, Edup(e->constructor) );
      bcc = P2_ext_hdng( m->content, P2_BCOPY_RECIPS );
    }
.
.
.
```

Cette solution a été testée avec succès.

5.7. Installation du Directory Service de EAN sur Zeus

Pour installer le Directory Service sur Zeus, deux étapes ont été nécessaires:

1°) il a fallu créer un Directory Service centralisé aux FUNDP: création d'un "**Central Nameserver**";
2°) il a fallu configurer EAN sur Zeus pour avoir accès à ce Directory Service centralisé: création d'un "**Local Nameserver**".

Nous avons choisi d'installer le Directory Service centralisé sur Junon, sur une seconde version de EAN fonctionnant sur cette machine : il s'agit d'une version UBC2, caractérisée par les attributs X.400: "OU=ts2;O=info;P=fundp;A=rtt;C=be".

Remarque: il aurait été beaucoup plus intéressant de l'installer sur la version originale de EAN sur Junon ("OU=ts;O=info;P=fundp;A=rtt") ou sur Zeus lui-même, mais:

- sur "ts.info": les programmes relatifs au Directory Service ne sont pas compilés (et il est impossible de les recompiler depuis le passage à la version 4.3 de Unix);

- sur "zeus.info": les programmes relatifs au Directory Service n'ont pas l'air de fonctionner convenablement: **nscentral** affiche "**PANIC: Memory Management Error (mem)**", **nsmaint** affiche "**Unknown error! (must be a bug)**".

Nous commencerons d'abord par expliquer l'installation du Local Nameserver sur Zeus, en supposant le Central Nameserver installé (voir section 5.7.1.). Puis, nous expliquerons l'installation du Central Nameserver (voir section 5.7.2.).

5.7.1. Configuration de EAN sur Zeus pour accéder au Directory Service centralisé

A) Les données nécessaires

Les données nécessaires sont les suivantes:

1°) NUA du Central Nameserver: "S=nameserver;OU=ts2;O=info;P=fundp;A=rtt;C=be";
2°) Nom de l'organisation de Zeus : "zeus.info";

3°) Mot de passe de l'UA du Local Nameserver : "abc";
4°) NUA du Local Nameserver (sur Zeus): "S=nameserver;OU=zeus;O=info;P=fundp;A=rtt;C=be".

B) Les opérations à effectuer

Il faut effectuer les opérations suivantes:

1°) Puisque le Central Nameserver est situé sur une autre machine, nous utiliserons ici un Local Nameserver. Par conséquent, le programme **nameserver** à utiliser sur Zeus est **nsrelay**:

```
% cp /usr/local/lib/ean/nsrelay  /usr/local/lib/ean/nameserver
```

2°) On initialise le Central Nameserver par la commande:

```
% /usr/local/lib/ean/nameserver -c "zeus.info" "abc" \
      "S=nameserver;OU=ts2;O=info;P=fundp;A=rtt;C=be"
```

Ceci a pour effet de créer le fichier **/usr/spool/ean/sys/dsa/local** et d'y placer les quatre lignes suivantes:

```
S=nameserver;P=fundp;A=rtt;C=be
abc
zeus.info
S=nameserver;OU=ts2;O=info;P=fundp;A=rtt;C=be
```

3°) Il faut ensuite créer un UA "S=nameserver" sur Zeus, dont le work program est **nameserver** (c.-à-d. **nsrelay**)[1].

4°) Il reste à créer le fichier **/usr/spool/ean/sys/ua/nameinfo** et d'y placer une ligne contenant le NUA du Local Nameserver:

```
S=nameserver;OU=zeus;O=info;P=fundp;A=rtt;C=be
```

Ces opérations effectuées, le Directory Service fonctionne alors comme suit:

Lorsqu'un utilisateur demande accès au Directory Service, EAN va lire le fichier **/usr/spool/ean/sys/ua/nameinfo** pour connaître le NUA du Local Nameserver. Sur Zeus, il trouve le NUA "S=nameserver;OU=zeus;O=info;P=fundp;A=rtt;C=be". La requête de l'utilisateur est alors envoyée à cette adresse.

Lorsque la requête arrive à cette adresse, le work program associé (**nsrelay**) va lire le fichier **/usr/spool/ean/sys/dsa/local**. Il y trouve le nom de l'organisation de Zeus ("zeus.info") et le NUA du Central Nameserver ("S=nameserver;OU=ts2;O=info;P=fundp;A=rtt;C=be"). Il ajoute alors à la requête le nom de l'organisation et il l'envoie dans la boîte aux lettres du Central Nameserver.

1. Sur certaines versions de EAN, cette création est automatiquement effectuée par le programme **nameserver** lors de l'étape précédente.

C) Les opérations réellement effectuées sur Zeus

Comme nous l'avons vu au début de la section 5.7., les programmes relatifs au Directory Service semblent ne pas fonctionner correctement sur Zeus, une fois compilés. Or la procédure normale d'installation nécessite sur Zeus le programme **nsrelay**. Il serait souhaitable de n'utiliser sur Zeus aucun programme relatif au Directory Service.

Pour ce faire, nous avons simplement modifié le contenu du fichier **/usr/spool/ean/sys/ua/nameinfo**, et nous y avons placé le NUA du Central Nameserver sur Junon, plutôt que celui du Local Nameserver:

S=nameserver;OU=ts2;O=info;P=fundp;A=rtt;C=be

Lorsqu'une requête est envoyée au Directory Service sur Zeus, elle est ainsi directement envoyée au Central Nameserver, sans passer par le Local Nameserver (on peut vérifier que le fichier **/usr/spool/ean/sys/dsa/local** n'est plus nécessaire sur Zeus!).

Remarque: le même principe peut être utilisé pour installer le Directory Service sur Junon !!!

5.7.2. Création du Directory Service centralisé

Les opérations à effectuer sont similaires aux opérations décrites en 5.7.1.:

1°) Puisque l'on désire installer le Central Nameserver, nous utiliserons comme **nameserver** le programme **nscentral**:

% cp /usr/local/lib/ean/nscentral /usr/local/lib/ean/nameserver

2°) On initialise le Central Nameserver par la commande:

% /usr/local/lib/ean/nameserver -c "ts2.info" "abc"[1]

Ceci a pour effet de créer le fichier **/usr/user3/spool/ean/sys/dsa/local** et d'y placer les lignes suivantes:

nameserver@ts2.info.fundp
abc
ts2.info

3°) Il est également nécessaire de créer un UA "nameserver" sur Junon, dont le work program est **nameserver** (c.-à-d. **nscentral**)[2].

4°) Il reste à créer le fichier **/usr/user3/spool/ean/sys/ua/nameinfo** et d'y placer la ligne suivante:

nameserver@ts2.info.fundp

1. mot de passe de l'UA du Central Nameserver.
2. Sur certaines versions de EAN, cette création est automatiquement effectuée par le programme nameserver lors de l'étape précédente.

5.8. Modification de l'installation de EAN sur Zeus pour qu'il puisse être utilisé par les stations de travail

Comme nous l'avons vu dans la section 5.1.1., chaque station de travail a comme système de fichiers "**/usr/users**" le système de fichiers "**/usr/users**" importé de Zeus.

Dès lors, pourquoi ne pas essayer d'utiliser les fichiers de EAN sur Zeus à partir des stations de travail ? Ceci permettrait d'accèder à EAN depuis ces stations comme si l'on travaillait directement sur Zeus.

Pour que les stations de travail puissent accéder à EAN, il faut déplacer les fichiers utilisés par EAN afin qu'ils soient tous dans le système de fichiers **/usr/users.**

Par exemple, on peut effectuer sur Zeus les déplacements de répertoires (et de sous-répertoires) suivants:

- /usr/ean -> /usr/users/ean
- /usr/spool/ean -> /usr/users/ean/spool
- /usr/local/lib/ean -> /usr/users/ean/local

De la sorte, EAN est situé dans des répertoires accessibles par les stations de travail. Cependant, ces répertoires ne sont pas les répertoires où EAN s'attend à trouver ses fichiers. En effet, ces fichiers se trouvent maintenant - aussi bien sur Zeus que sur les stations de travail - dans les répertoires **/usr/users/ean, /usr/users/ean/spool** et **/usr/users/ean/local.** Or, EAN s'attend à trouver ses fichiers dans **/usr/ean, /usr/spool/ean, /usr/local/lib/ean.**

On peut remédier à ce dernier problème en créant sur Zeus et sur chaque station les liens logiques suivants:

```
% ln -s /usr/users/ean  /usr/ean
% ln -s /usr/users/ean/spool  /usr/spool/ean
% ln -s /usr/users/ean/local  /usr/local/lib/ean
```

Une autre solution est de reconfigurer EAN et de le recompiler pour qu'il trouve ses fichiers dans **/usr/users/ean, /usr/users/ean/spool** et **/usr/users/ean/local.**

Note: Le propriétaire et le groupe de la majorité des fichiers EAN est "**ean**". Afin que le propriétaire et le groupe soient connus des stations de travail, il est également nécessaire de créer sur chacune d'elle un utilisateur "**ean**" et un groupe "**ean**".

Nous avons réalisé et testé ces modifications. Nous nous sommes alors rendu compte que EAN ne fonctionnnait sur les stations de travail qu'à condition de ne communiquer qu'avec des UAs locaux à Zeus. Pourquoi ?

Lorsque l'on lance le programme **ean** sur une station de travail (par exemple Ws1), le processus **ean** est créé et exécuté sur Ws1. Lorsqu'un message doit être envoyé à un autre MTA, le processus **ean** doit normalement avertir le MTA Manager. Or, ici, le MTA Manager est exécuté sur Zeus, alors que **ean** s'attend à le trouver sur le même système que lui-même, c.-à-d. Ws1. Donc, la communication entre **ean** et le MTA Manager ne peut s'effectuer correctement.

On pourrait alors croire qu'il suffirait de lancer le MTA Manager sur chaque station de travail. C'est faux. En effet, examinons ce qui se passerait dans ce cas. Supposons que la station Ws1 désire appeler par TCP/IP un MTA d'un système éloigné (appelons ce système "Xyz"). Le processus **ean** sur Ws1 parviendrait à avertir le MTA Manager sur Ws1. Celui-ci programmerait alors un appel à **txp**, sur Ws1, afin d'établir une communication avec le MTA de Xyz. C'est ici qu'il y aurait un problème:

- le processus **txp** appelant (c.-à-d. sur Ws1) essayerait d'établir une communication entre
  le port IPC ean de Ws1 (car **txp** appelant est exécuté sur Ws1)
  et le port IPC ean de Xyz

- le processus **txp** appelé (c.-à-d. sur Xyz) essayerait d'établir une communication entre
  le port IPC ean de Xyz
  et le port IPC ean de Zeus (car, pour le **txp** appelé, le MTA "zeus.info" est accessible par le port IPC ean sur le système Zeus[1])

Conclusion:

Si l'on désire accéder depuis les stations de travail à des UAs autres que les UAs de Zeus, et cela sans modifier les sources de EAN, il semble nécessaire d'installer le programme EAN complet (UA et MTA) sur chaque station. L'utilisateur de chaque station serait ainsi associé à un NUA différent. S'il le désire, il pourrait utiliser l'option "auto-forward" sur Zeus pour rediriger son courrier vers la station sur laquelle il travaille.

1. Ces informations se trouvent dans les tables du MTA de Xyz.

Chapitre 6 : La Norme X409

Dans ce chapitre, nous allons examiner la norme X.409[1] qui définit la syntaxe de transfert utilisée par les protocoles de la couche Application dans les MHSs (Message Handling Systems). Dans l'architecture des systèmes ouverts interconnectés, la syntaxe de transfert est utilisée pour représenter l'information échangée entre deux entités applications.

6.1. Représentation standard

La représentation standard pour une valeur d'un certain type, est un "élément" qui se compose de trois parties :

-l'identificateur du type
-la longueur du contenu
-le contenu.

On dira qu'un élément comporte trois "composants".

6.1.1. L'identificateur

L'identificateur permet de distinguer un type de données d'un autre et permet d'interpréter correctement le contenu.

1) Il y a quatre classes de types :

-universal	indépendants de l'application
-application-wide	spécifiques à une application particulière
-context-specific	utilisés dans un contexte plus restreint; leurs identificateurs ne sont distincts qu'à l'intérieur de ce contexte
-private-use	réservés pour un usage privé.

Les bits 8 et 7 de l'identificateur spécifient la classe :

bit 8	bit 7	classe
0	0	universal
0	1	application-wide
1	0	context-specific
1	1	private-use.

1. appelée aussi ASN.1

2) On distingue également deux formes d'éléments :

-primitive	élément dont le contenu est atomique
-constructor	élément dont le contenu est lui-même un élément ou une suite d'éléments.

Le bit 6 de l'identificateur spécifie la forme de l'élément :

bit 6	forme
0	primitive
1	constructor.

3) Le **code ID** sert à distinguer un type de données d'un autre de la même classe. Si ce code est compris entre 0 et 30, les cinq derniers bits (bits 5 à 1) de l'octet suffisent pour le coder. Sinon, ces cinq bits sont garnis de la valeur 31, et le code ID est contenu dans un ou plusieurs octets d'extension. Le dernier octet d'extension a le bit 8 positionné à 0; les autres octets d'extension ont le bit 8 positionné à 1; la concaténation des bits 1 à 7 des octets d'extension traduit la valeur du code ID.

Exemples:

L'identificateur d'un type de classe universal, de forme primitive et de code ID 10 (= 0AH), serait représenté de la manière suivante:

8	7	6	5	4	3	2	1
0	0	0	0	1	0	1	0

Tandis que l'identificateur d'un type de classe universal, de forme primitive et de code ID 64 (= 40H) serait représenté comme suit:

8	7	6	5	4	3	2	1
0	0	0	1	1	1	1	1

8	7	6	5	4	3	2	1
0	1	0	0	0	0	0	0

6.1.2. La longueur

La longueur spécifie la longueur en octets L du contenu. Il y a trois sortes de longueur :

-short
-long
-indefinite

Si la longueur est de type **short**, alors le bit 8 de l'octet est à 0 et les autres bits donnent la valeur de L (au maximum 127).

Si la longueur est de type **long**, alors le bit 8 de l'octet est à 1, le nombre d'octets d'extension est donné par les autres bits et la valeur de L est donnée par la concaténation des bits 1 à 8 des octets d'extension.

Si la longueur est de type **indefinite**, l'octet contient la valeur 128, la longueur du contenu n'est pas explicitement spécifiée, mais un élément spécial EOC (End Of Contents) termine le contenu. Par définition, la représentation d'un élément EOC est un élément de classe universal, de forme primitive, dont le code ID vaut 0, dont le contenu est absent et dont la longueur vaut 0 (c.-à-d. en hexadécimal les deux octets: 00 00).

6.1.3. Le contenu

Le contenu est l'essence même de l'élément et contient la substance de l'information que l'élément est censé véhiculer. La longueur du contenu est variable, et le contenu lui-même est interprété conformément au type de l'élément. Si l'élément est de forme constructor, son contenu comprend lui-même plusieurs éléments.

6.2. Types prédéfinis

6.2.1. Boolean

La représentation d'un booléen est un élément
- -de classe universal
- -de forme primitive
- -dont le code ID vaut 1
- -dont le contenu est un octet qui encode la valeur du booléen (tous les bits à 0 pour false, n'importe quelle autre valeur pour true)

Ex : la valeur de type booléen true peut être encodée comme suit :

identificateur	longueur	contenu	
01	01	FF	(en notation hexadécimale)

6.2.2. Integer

La représentation d'un entier est un élément
- -de classe universal
- -de forme primitive
- -dont le code ID vaut 2
- -dont le contenu est une suite d'octets dont la concaténation donne la valeur de l'entier.

Ex : la valeur de type entier 128 peut être encodée comme suit :

identificateur	longueur	contenu	
02	01	80	(en notation hexadécimale)

6.2.3. Bit string

La représentation d'un Bit string est un élément
- -de classe universal
- -de forme primitive ou constructor
- -dont le code ID vaut 3
- -dont le contenu dépend de la forme.

Si la forme est primitive, le contenu est une suite d'octets encodant le Bit string, et précédée d'un octet donnant le nombre de bits inutilisés dans le dernier octet de la suite.

Si la forme est constructor, le contenu est une suite d'éléments de type Bit string. Cette forme est classiquement employée avec la longueur de type indefinite.

Ex : la valeur de type Bit string '101010101010' peut être encodée comme suit :

```
identificateur  longueur  contenu
     03           03      04 AA A0          (forme primitive)
                            |  |
                            |  |

                          4 bits inutilisés
```

ou

```
identificateur  longueur  contenu
     23           80      id  lo  co          (forme constructor)
                          03  02  00 AA -- Bit string '10101010'
                          03  02  04 A0 -- Bit string '1010'
                          00 (EOC)
```

6.2.4. Octet string

La représentation d'un Octet string est un élément
- -de classe universal
- -de forme primitive ou constructor
- -dont le code ID vaut 4
- -dont le contenu est la suite d'octets contenus dans l'Octet string.

Ex : la valeur de type Octet string "SALUT" peut être encodée comme suit :

```
identificateur  longueur  contenu
     04           05      53 41 4C 55 54      (forme primitive)
```

ou

```
identificateur  longueur  contenu
     24           80      id  lo  co          (forme constructor)
                          04  04  53 41 4C -- Octet string "SAL"
                          04  05  55 54    -- Octet string "UT"
                          00 (EOC)
```

6.2.5. Null

La représentation d'un Null est un élément
- -de classe universal
- -de forme primitive

-dont le code ID vaut 5
-dont le contenu est absent
-dont la longueur vaut 0.

Il se représente donc comme suit (en hexadécimal): 05 00

6.2.6. Sequence

La représentation d'une séquence est un élément
-de classe universal
-de forme constructor
-dont le code ID vaut 16
-dont le contenu est la suite ordonnée des éléments de cette séquence.

Ex : la valeur de type Sequence d'entiers {128,64} peut être codée comme suit :

```
identificateur  longueur  contenu
     30           06      id  lo  co          (forme constructor)
                          02  01  80  -- l'entier 128
                          02  01  40  -- l'entier 64
```

6.2.7. Tagged (étiqueté)

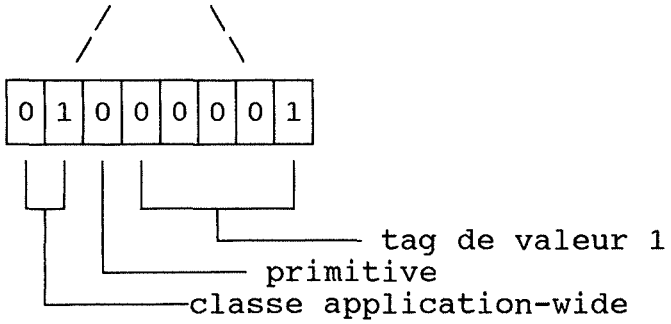
Le type Tagged fournit un moyen de créer de nouveaux types à partir de ceux qui existent déjà, et de telle sorte que les nouveaux types sont distinguables de ceux qui existent. Ce type permet à une valeur d'être étiquetée à un niveau plus haut que ne le permettent les types existants, pour refléter sa sémantique ou ses contraintes syntaxiques. Par exemple, le nom d'une personne et le nom d'un pays peuvent être distingués même si les deux sont des chaînes de caractères.

Exemple : supposons une application de gestion d'adresses qui veuille créer deux types nouveaux: l'un utilisé pour les numéros de rue et un autre utilisé pour les codes postaux. Ces types seront donc de la classe "application-wide" puisque dépendants de l'application, et ils seront tous deux dérivés du type entier; la représentation des valeurs de ces types peut être de forme primitive ou constructor. La valeur du tag est codée sur les cinq bits restants de l'identificateur, et éventuellement dans des octets d'extension, comme vu précédemment.

Bien entendu, le principe du tag peut s'appliquer à toutes les classes; toutefois, si l'on veut l'utiliser dans la classe universal, il faut prendre garde à ne pas choisir pour la valeur du tag le code ID d'un type existant.

Ex : le numéro de rue 24 peut être encodé comme suit, si on a attribué le tag 1 au type utilisé pour les numéros de rue:

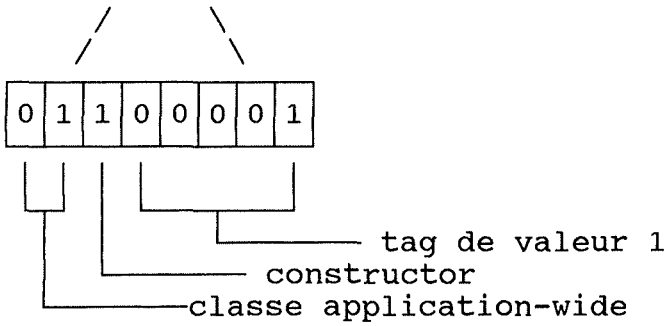
```
identificateur longueur contenu
     41          01       18            (forme primitive)
```



Dans ce cas-ci, la représentation est de forme primitive, et il n'est pas spécifié que la valeur est d'un type dérivé du type entier; on dit que la représentation est **implicite**. Si on veut expliciter que la valeur est d'un type dérivé du type entier, alors il faut utiliser la représentation de forme constructor, qu'on appelle aussi, dans le cas du type Tagged, représentation **explicite** :

Ex : le numéro de rue 24 peut être encodé comme suit :

```
identificateur longueur contenu
     61          03      id  lo  co        (forme constructor)
                         02  01  18 -- l'entier 24
```



Dans ce cas-ci, l'identificateur 02 reflète le type entier.

6.2.8. Set

La représentation d'un Set est un élément
- -de classe universal
- -de forme constructor
- -dont le code ID vaut 17
- -dont le contenu est la série des éléments du Set.

Ex : la valeur de type Set {"SALUT",128} peut être encodée comme suit :

```
identificateur longueur contenu
     31          0A      id  lo  co              (forme constructor)
                         04  05  53 41 4C 55 54 -- Bit string "SALUT"
                         02  01  80 -- l'entier 128
```

Rem : comme dans un Set l'ordre des éléments n'est pas significatif et qu'il faut quand même pouvoir identifier les éléments, il faut nécessairement qu'ils aient des identificateurs différents. Cela peut se faire en utilisant le type Tagged. Prenons un exemple : le Set {n° -de-produit, prix-de-vente}. Comme les deux valeurs sont de type entier, il faut les distinguer l'une de l'autre en attribuant, par exemple, le tag de valeur 1 à n° -de-produit et le tag 2 à prix-de-vente; comme n° -de-produit et prix-de-vente ne doivent être distincts qu'à l'intérieur du Set, on peut utiliser la classe context-specific.

Ex :Si l'on définit le type de n° -de-produit comme étant de classe context-specific, de forme constructor et de tag 1, son identificateur vaudra, en notation hexadécimale, A1 (10100001 en binaire). Si l'on définit da la même manière le type de prix-de-vente, mais avec le tag 2, son identificateur vaudra A2 en hexadécimal. Avec ces conventions, le Set avec n° -de-produit = 10 et prix-de-vente = 64 peut être encodé comme suit :

```
identificateur longueur contenu
    31          A0      id  lo  co                  (forme constructor)
                        A1  03  id  lo  co
                                02  01  0A  -- l'entier 10
                        A2  03  id
                                02  01  40  -- l'entier 64
```

6.2.9. Choice

Un **choice** représente une valeur dont le type est choisi parmi un ensemble de types. La représentation d'un Choice est la représentation du type choisi. Les éléments qui représentent les différents types doivent avoir des identificateurs distincts; cette nécessité est satisfaite en étiquetant chaque type 'context-specific' par l'emploi du type Tagged.

Ex : Imaginons que l'on caractérise un individu par son nom, de type Octet string, ou par son numéro de carte d'identité, de type Integer, et seulement par l'une de ces deux valeurs, **au choix**. Si l'on attribue le tag 1 à l'identificateur du type du nom et le tag 2 à l'identificateur du type de l'identité, alors la valeur "Smith", de type Choice, est représentée comme suit:

```
identificateur longueur contenu
    A1          03      id  lo  co              (forme constructor)
                        04  05  53 6D 69 74 68  -- Octet string "Smith"
```

6.2.10. Any

Un élément de type Any représente une valeur dont le type est choisi parmi un ensemble de types prédéfinis et définis. La représentation d'un type Any est celle du type choisi.

Ex : la valeur entière 10 de type Any peut être encodée comme suit :

```
identificateur longueur contenu
    02          01          0A  -- l'entier 10
```

6.3. Types définis

Un type défini est un type décrit à partir des types prédéfinis et/ou définis, et auquel on a donné un nom. La norme X409 en définit sept, mais nous nous limiterons ici à deux types particulièrement employés par EAN : le type IA5String et le type PrintableString.

6.3.1. IA5String (International Alphabet n° 5)

La représentation d'un IA5String est un élément
-de classe universal
-de forme primitive
-dont le code ID vaut 22.

En fait, le type IA5String est un type Tagged dérivé du type Octet string, dont la représentation est implicite, et dont le tag a la valeur 22. La valeur des octets est celle des codes de IA5 correspondants aux caractères de la chaîne IA5String.

Ex : la valeur 'Salut' de type IA5String peut être encodée comme suit :

```
identificateur  longueur  contenu
     16            05      53 61 6C 75 74  -- "Salut"
```

6.3.2. PrintableString

La représentation d'un PrintableString est un élément
-de classe universal
-de forme primitive
-dont le code ID vaut 19.

En fait, le type PrintableString est un type Tagged dérivé du type IA5String, dont la représentation est implicite, et dont le tag a la valeur 19. Les caractères permis dans une chaîne de caractères de type PrintableString est un sous-ensemble des caractères de l'IA5, qui comprend les lettres majuscules, minuscules, les chiffres, les parenthèses, le plus, le trait d'union, l'apostrophe, la virgule, l'espace, le point, le double point, le slash, le signe égal et le point d'interrogation.

Ex : la valeur 'Salut' de type PrintableString peut être encodée comme suit :

```
identificateur  longueur  contenu
     13            05      53 61 6C 75 74  -- "Salut"
```

Chapitre 7 : Les structures de données essentielles de EAN

L'étude d'un logiciel commence bien souvent par l'étude des données qu'il manipule. Nous avons voulu dans ce chapitre décrire quelques structures de données abondamment utilisées dans EAN. Pour chacune de ces structures, nous décrirons également les principaux opérateurs associés.

7.1. La structure de données ENODE

7.1.1. Définition

Le type de données ENODE est défini par la structure suivante:

```
struct ENODE
        {
                eid             id
                elen            length
                byte *          primitive
                struct ENODE *  constructor
                struct ENODE *  next
        }
```

7.1.2. Principe de fonctionnement

La structure de donnée "ENODE" est très importante: elle permet de représenter n'importe quelle donnée X.409 !

- Les champs **id** et **length** correspondent respectivement aux parties identificateur (type du contenu) et longueur du contenu en X.409.

- Lorsque la donnée est de forme "primitive", son adresse est placée dans le champ **primitive** et les champs **constructor** et **next** ne contiennent rien (NULL). Le champ **length** contient le nombre d'octets occupés par la donnée.

- Lorsque la donnée est de forme "constructor", chacun de ses éléments est représenté par un ENODE. Le premier de ceux-ci est pointé par le champ **constructor**, et les suivants forment une liste chaînée par le champ **next**. Par exemple, si la donnée A est de forme "constructor" et est constituée des données B, C et D, on a les relations suivantes (Figure 7.1):

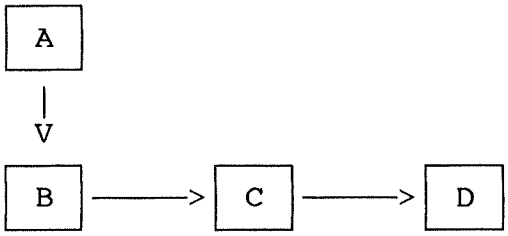


Figure 7.1: Les relations 'next' et 'constructor'.

où - les flèches verticales désignent les pointeurs **constructor** non NULLs;
- les flèches horizontales désignent les pointeurs **next** non NULLs.

Initialement, le champ **length** d'un ENODE "constructor"[1] comprend 0. Cependant, si l'on veut rester conforme à X.409, il est nécessaire que ce champ contienne le nombre exact d'octets nécessaires pour coder le constructor en entier. Le remplissage du champ **length** de tous les ENODEs constructor d'un ENODE donné est réalisé par la fonction X9_len (pour plus de détails, voir 8.3.2).

7.1.3. Exemple

Supposons que l'on désire représenter la donnée A définie comme suit:

- la donnée A est de forme "constructor", constituée des données B, C et D;
- la donnée B est de forme "constructor", constituée des données E et F;
- la donnée D est de forme "constructor", constituée des données G, H et I;
- les données C, E, F, G, H, I sont de type "primitive".

Cette situation est illustrée par l'arbre de données ci-dessous, où seules les feuilles peuvent contenir des données (Figure 7.2):

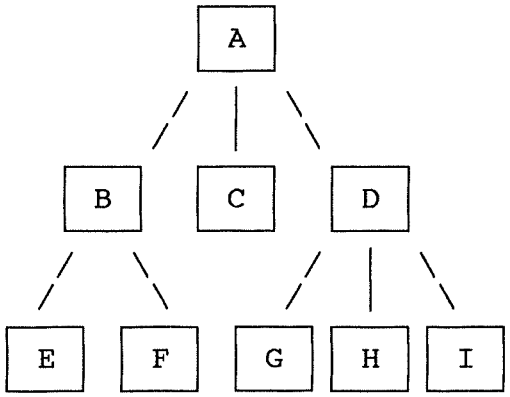


Figure 7.2: Exemple d'arbre de données.

Cet arbre pourra être implémenté par les ENODEs suivants (Figure 7.3), grâce aux champs **next** (flèches horizontales) et **constructor** (flèches verticales):

1. Nous parlerons d'ENODE "**primitive**" pour désigner un ENODE représentant une donnée X.409 de forme "primitive" (champ **primitive** < > NULL, champ **constructor** = = NULL), et d'ENODE "**constructor**" pour désigner une donnée X.409 de forme "constructor" (champ **primitive** = = NULL, champ **constructor** < > NULL).

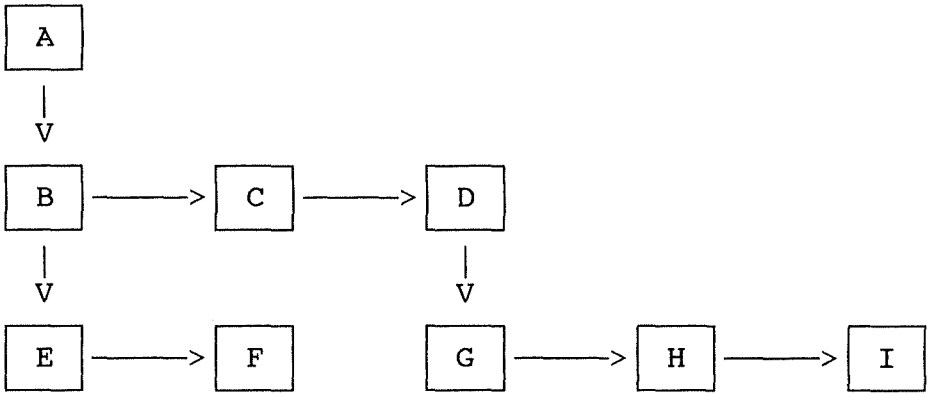


Figure 7.3: Exemple de structure d'ENODEs.

Nous détaillons les valeurs des champs **next** et **constructor** dans le tableau 7.1:

enode	constructor	next
A	pointeur vers B	0
B	pointeur vers E	pointeur vers C
C	0	pointeur vers D
D	pointeur vers G	0
E	0	pointeur vers F
F	0	0
G	0	pointeur vers H
H	0	pointeur vers I
I	0	0

Tableau 7.1: Exemple de valeurs pour **next** et **constructor**.

A, B et D sont des ENODEs de type "constructor" (ils correspondent aux noeuds de l'arbre logique); les autres ENODEs sont de type "primitive".

Les champs **primitive** contiennent les données suivantes (Tableau 7.2):

enode	primitive
A	0
B	0
C	pointeur vers données de C
D	0
E	pointeur vers données de E
F	pointeur vers données de F
G	pointeur vers données de G
H	pointeur vers données de H
I	pointeur vers données de I

Tableau 7.2: Exemple de valeurs pour **primitive**.

Quant au champ **length**:

- dans les ENODEs "primitive": il indique le nombre de bytes nécessaires pour encoder la donnée;
- dans les ENODEs "constructor": il vaut initialement 0 mais sera rempli par après par la valeur correcte, comme en X.409 (voir fonction X9_len, section 8.3.2).

7.1.4. Fonctions C associées

Bien entendu, les ENODEs primitives peuvent contenir n'importe quel type de données. Citons quelques fonctions qui convertissent des données d'un certain type en une suite de bytes et qui créent un ENODE primitive auquel est associée cette suite de bytes:

-**Bld_prim**(eid id, char * data): cette fonction crée un ENODE de type primitive, d'identificateur id et contenant une copie de la chaine de caractères <data>, et renvoie un pointeur vers cet ENODE.

-**Bld_line**(eid id, char * data): fonction identique à **Bld_prim**, mais, avant de construire ENODE, Bld_line ajoute les 2 caractères "\r\n"[1] à la fin de la copie de <data>.

-**Bld_int**(eid id, long val): idem, mais pour l'entier de 4 bytes <val>.

-**Bld_bool**(eid id, int b): idem, pour le booléen <b>.

-**Bld_time**(T_REC * tr): idem, pour le temps <tr>.

-**Bit_set**(ENODE * e, int n): cette fonction met à 1 le <n>ième bit de la chaîne de bits (Bit String) associée à l'ENODE *e, crée cet ENODE s'il n'existait pas (avec tous les bits à 0, sauf le <n>ième), et renvoie un pointeur vers cet ENODE.

-**Bit_clr**(ENODE * e, int n): idem, pour mettre ce bit à 0.

D'autres fonctions réalisent les opérations inverses, c.-à-d. lisent la valeur associée à l'ENODE primitive. Ainsi, les fonctions **Unb_prim**(ENODE * e), **Unb_int**(ENODE * e), **Unb_bool**(ENODE * e), **Unb_time**(ENODE * e) renvoient respectivement la chaîne de caractères, la valeur entière, la valeur booléenne et le temps associé à l'ENODE de type primitive *e. Quant à la fonction **Bit_test**(e, n), elle renvoie 0 ou 1 selon que le <n>ième bit du Bit String associé à l'ENODE *e est à 0 ou à 1.

Les fonctions suivantes permettent de construire des ENODEs constructor:

-**Bld_cons**(eid id, ENODE * e): retourne un pointeur vers un ENODE de type constructor, de longueur nulle, d'identificateur <id>, et dont le fils est l'ENODE *e.

1. "\r" = retour de chariot, "\n" = à la ligne

-**Set_add**(ENODE * * ptr_e, ENODE * e): rajoute à l'ENODE * *ptr_e, de type Set, le fils *e. Comme l'ENODE * *ptr_e est un Set, s'il y avait déjà un fils avec le même identificateur que *e, alors ce fils est supprimé. De plus, Si l'ENODE * *ptr_e n'existe pas (*ptr_e = = NULL), alors cet ENODE (de type Set) est d'abord créé avant d'y ajouter son fils.

-**Seq_add**(ENODE * * ptr_e, ENODE * e): rajoute à l'ENODE * *ptr_e, de type Sequence, le fils *e. De plus, Si l'ENODE * *ptr_e n'existe pas (*ptr_e = = NULL), alors cet ENODE (de type Sequence) est d'abord créé avant d'y ajouter son fils.

Citons enfin les fonctions suivantes, relatives aux ENODEs constuctors:

-**Set_find**(ENODE * e, eid id): examine l'ENODE *e (de type Set) et cherche l'éventuel fils ayant id pour identificateur. Retourne un pointeur vers ce fils (NULL s'il n'existe pas).

-**Set_ext**(ENODE * e, eid id): comme Set_find, mais le fils est supprimé des constructors de *e.

-**Set_rem**(ENODE * e, eid id): examine l'ENODE *e (de type Set) et y supprime l'éventuel fils ayant id pour identificateur.

7.2. La structure de données IO_DESC

7.2.1. Définition

Une structure IO_DESC est un descripteur d'E/S, appelé plus simplement un **buffer**, qui est généralement associé à un périphérique d'E/S. Citons, par exemple, **Pph_istd** (peripheral input standard) qui est le buffer d'entrée associé au clavier, et **Pph_ostd** (peripheral output standard), qui est le buffer de sortie associé à l'écran. Le type de données IO_DESC est défini, d'une manière générale, par la structure suivante:

```
struct IO_DESC
        {
            byte *              ptr
            int                 cnt
            RC[1]               status
            RC                  (*ctl)()
            RC                  (*flush)()
            RC                  (*fill)()
            RC                  (*close)()
            int                 state
            byte *              buff
            int                 size
            struct IO_DESC *    io
        }
```

1. RC (Return Code) est un type équivalent au type entier.

7.2.2. Principe de fonctionnement

A) Les champs buff, size, ptr, cnt

Un IO_DESC utilisé en sortie (Figure 7.4), est initialisé avec les valeurs suivantes:

buff : pointe vers une zone mémoire.
size = taille de cette zone en bytes.
ptr = buff.
cnt = size.

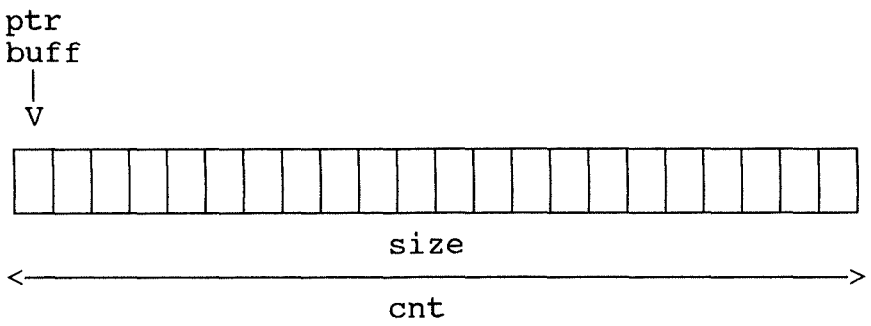


Figure 7.4: Initialisation d'un IO_DESC utilisé en sortie.

En cours de fonctionnement (Figure 7.5), les champs ont les valeurs suivantes:

buff : inchangé.
size = inchangé.
ptr = 1er byte disponible.
cnt = nombre de bytes disponibles.

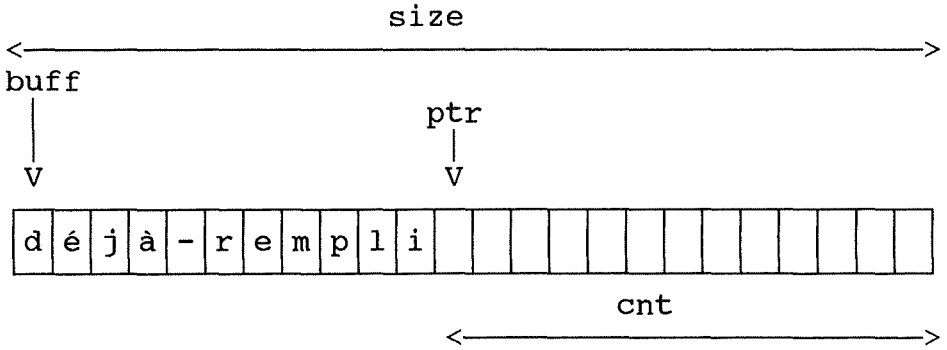


Figure 7.5: Configuration d'un IO_DESC utilisé en sortie.

Un IO_DESC utilisé en entrée (Figure 7.6), est initialisé avec les valeurs suivantes:

buff : pointe vers une zone mémoire.
size = taille de cette zone en bytes.
ptr = buff.
cnt = size.

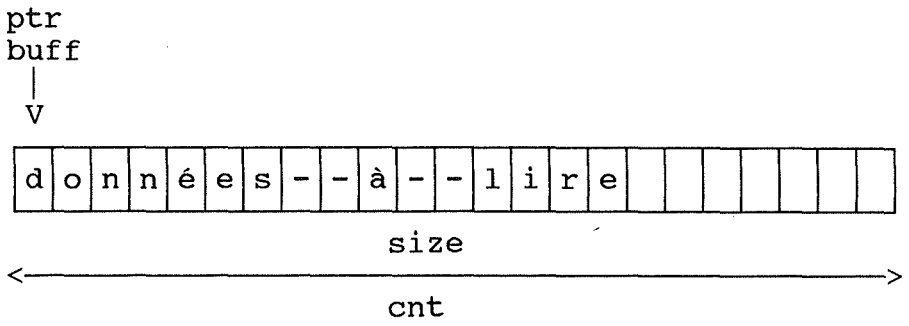


Figure 7.6: Initialisation d'un IO_DESC utilisé en entrée.

En cours de fonctionnement (Figure 7.7), les champs ont les valeurs suivantes:

buff : inchangé.
size = inchangé.
ptr = prochain byte à lire.
cnt = nombre de bytes restant à lire.

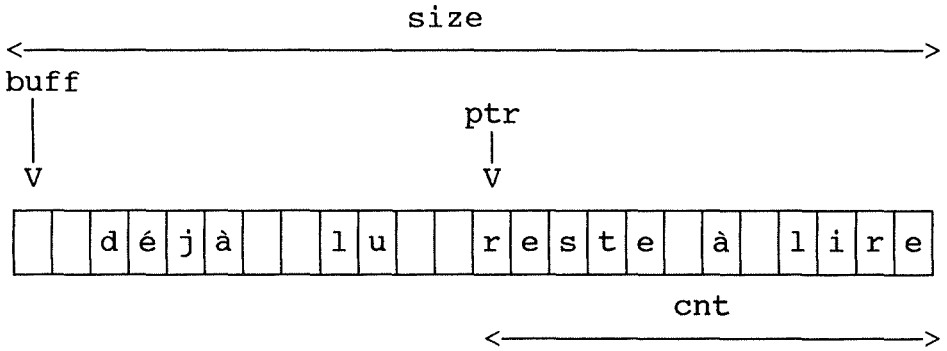


Figure 7.7: Configuration d'un IO_DESC utilisé en entrée.

B) Les champs flush et fill

Le champ 'flush' est initialisé avec le nom d'une fonction qui réalise le **flush**, c'est à dire le vidage du IO_DESC lorsque celui-ci est utilisé en écriture et qu'il est rempli. Il y a plusieurs fonctions de flush, c'est à dire plusieurs modes de flush: le buffer peut être simplement vidé, mais il peut aussi être étendu, et il peut également être linké avec d'autres IO_DESCs. Le champ 'fill' est quant à lui initialisé avec le nom d'une fonction qui réalise le **fill**, c'est à dire le remplissage du buffer lorsqu'il a été entièrement lu et que de nouvelles données doivent être lues.

C) Les champs ctl et status

Le champ 'status' est positionné à OK lorsqu'un flush (ou un fill) s'est correctement effectué, à RC_END dans le cas contraire. Le champ 'ctl' est initialisé avec le nom d'une fonction de contrôle.

D) Le champ close

Le champ 'close' est initialisé avec le nom d'une fonction qui assure la fermeture d'un IO_DESC. S'il est utilisé en sortie, il y a un flush automatique.

E) Le champ state

Le champ 'state' est initialisé avec la valeur 0 pour un IO_DESC utilisé en écriture, avec la valeur <size> s'il est utilisé en lecture.

F) Le champ io

Comme on l'a vu plus haut, il est possible de linker plusieurs IO_DESCs afin d'avoir un buffer de plus grande taille: le champ io pointe alors vers l'IO_DESC suivant.

7.2.3. Fonctions C associées

La structure IO_DESC est utilisée par de nombreuses fonctions. Citons par exemple:

En Sortie:

-**Ioflush**(IO_DESC * io): Force le "flush" (vidage) de l'IO_DESC *io; renvoie un code d'erreur.

-**Ioput**(IO_DESC * io, byte b): écrit dans l'IO_DESC *io le byte <b>; retourne le byte b (sauf si erreur: retourne 0).

-**Ioputs**(IO_DESC * io, char * str): écrit dans l'IO_DESC *io la chaîne <str>.

-**Ioputstr**(IO_DESC * io, char * str, int min): idem, mais en écrivant après la chaîne <str> des caractères blancs pour que le nombre de caractères écrits soit au moins égal à <min>.

-**Ioputbuff**(IO_DESC * io, byte * buff, int length): écrit dans l'IO_DESC *io les <min> premiers octets pointés par <buff>.

-**Ioputn**(IO_DESC * io, unsigned int val, int base, int width, char fill): écrit dans l'IO_DESC *io le nombre <val>, représenté en ASCII dans la base <base>, en y ajoutant des caractères <fill> de sorte que le nombre total de caractères écrits soit au moins égal à <width>.

Iofmt(IO_DESC * io, char * fmt, <liste des arguments>): parcourt la chaîne *fmt de gauche à droite en écrivant un à un dans l'IO_DESC *io les caractères lus. Cette fonction autorise certains caractères de contrôle, tous précédés du caractère '‘' (équivaut au '%' du printf), par exemple:

- ‘‘ : écrit un ‘‘
- ‘**<width>d** : écrit un entier positif ou négatif, en le complétant par des blancs pour qu'il occupe au moins <width> caractères
  Remarque : si <width> commence par '0', alors complète par des '0' plutôt que par des blancs
- ‘**<width>u**, ‘**<width>x**, ‘**<width>o** : comme ‘**<width>d**, mais en écrivant respectivement un entier positif, un entier hexadécimal, un entier octal
- ‘**c** : écrit un caractère
- ‘**<width>S** : écrit une chaîne de <width> caractères (en la complétant éventuellement par des blancs)
- ‘**<width>s** : comme ‘**<width>S** mais, si la chaîne est vide, elle est remplacée par la chaîne "<NULL>"

- '<**width**>**p** : écrit <width> blancs
- '**b** : écrit un buffer de taille donnée (la liste des arguments comprend d'abord un pointeur vers le buffer puis sa taille)

Remarques:
- les variables à sortir sont placées après le paramètre *fmt, dans la liste des arguments
- l'éventuel champ longueur <width> peut être
  - soit une suite de chiffres indiquant la longueur
  - soit le caractère '*' pour indiquer que la longueur est un paramètre fourni par après dans la liste des arguments (juste avant la variable à sortir ayant cette longueur)
- si le champ <width> est nécessaire mais n'est pas spécifié, il est supposé valoir 0

En entrée:

-**Iofill**(IO_DESC * io): Force le "fill" (remplissage) de l'IO_DESC *io; renvoie un code d'erreur.

-**Ioget**(IO_DESC * io): lit un caractère dans l'IO_DESC *io (en provoquant un fill si nécessaire) et retourne celui-ci.

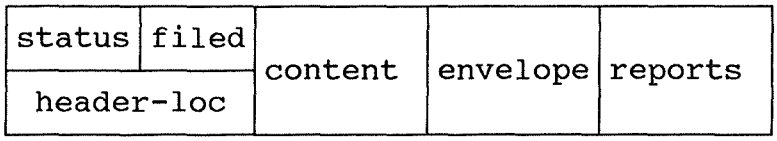
-**Iogetline**(IO_DESC * io): lit une chaîne de caractères l'IO_DESC *io (en provoquant un fill si nécessaire) et retourne un pointeur vers celle-ci.

7.3. La structure de données MESSAGE

7.3.1. Définition

Le type de données MESSAGE est défini par la structure suivante:

```
struct MESSAGE
        {
            int         status
            int         filed
            long        header-loc
            ENODE *     content
            ENODE *     envelope
            ENODE *     reports
        }
```



7.3.2. Le champ status

Le status est un entier codé sur deux bytes et qui reflète l'état, à n'importe quel moment, d'un message.

LSB :	bit 0	(NEWBIT)	: mis à 1 si le message est nouveau, c'est à dire, il vient d'être composé par l'utilisateur.
	bit 1	(READBIT)	: mis à 1 si le message a été lu par l'utilisateur.
	bit 2	(DRAFTBIT)	: mis à 1 si ce message est le draft; il va de soi que sur tous les messages existant à un moment donné, un seul est le draft.
	bit 3	(SENTBIT)	: mis à 1 si ce message a déjà été envoyé.
	bit 4	(RPRTBIT)	: mis à 1 si ce message comporte des options de report.
	bit 5	(RPLYBIT)	: mis à 1 si ce message est un message reply.
	bit 6	(FWRDBIT)	: mis à 1 si ce message est un message forward.
MSB :	bit 0	(DELBIT)	: mis à 1 si ce message a été effacé par l'utilisateur; il sera supprimé physiquement à la fin de l'exécution du programme.
	bits 1 à 7		: inutilisés.

7.3.3. Le champ filed

Le filed est un entier (codé sur deux bytes); il est mis à 0 si le message n'a pas (encore) été enregistré dans un folder, et à 1 s'il l'a été.

7.3.4. Le champ header-loc

Le FileSystem est une structure de données interne qui sert d'interface entre le support de sauvegarde et la mémoire centrale. Nous n'en parlerons pas davantage ici; signalons seulement que ce FileSystem contient, entre autres, un IO_DESC de nom 'content' sur lequel sont écrits les messages en représentation X.409. Il contient également un IO_DESC de nom 'headers' qui contient un résumé de chacun de ces messages. Ce résumé comprend:

- l'identification du message
- la date à laquelle a été composé ce message
- le sujet du message
- le status du message
- l'emplacement du contenu écrit en X.409 dans le IO_DESC 'content'
- l'emplacement de l'enveloppe écrite en X.409 dans le même IO_DESC
- l'emplacement des reports écrits en X.409 dans ce même IO_DESC
- la taille du contenu en X.409
- la taille de l'enveloppe en X.409
- la taille des reports en X.409.

Le champ header-loc contient, quant à lui, l'emplacement du résumé du message dans le IO_DESC 'headers'.

7.3.5. Le champ content

Le champ 'content' contient un pointeur vers un arbre (Figures 7.8, 7.9 & 7.10) représentant le contenu du message (au sens X.400, c.-à-d. l'IM-UAPDU échangé entre IPM UAs):

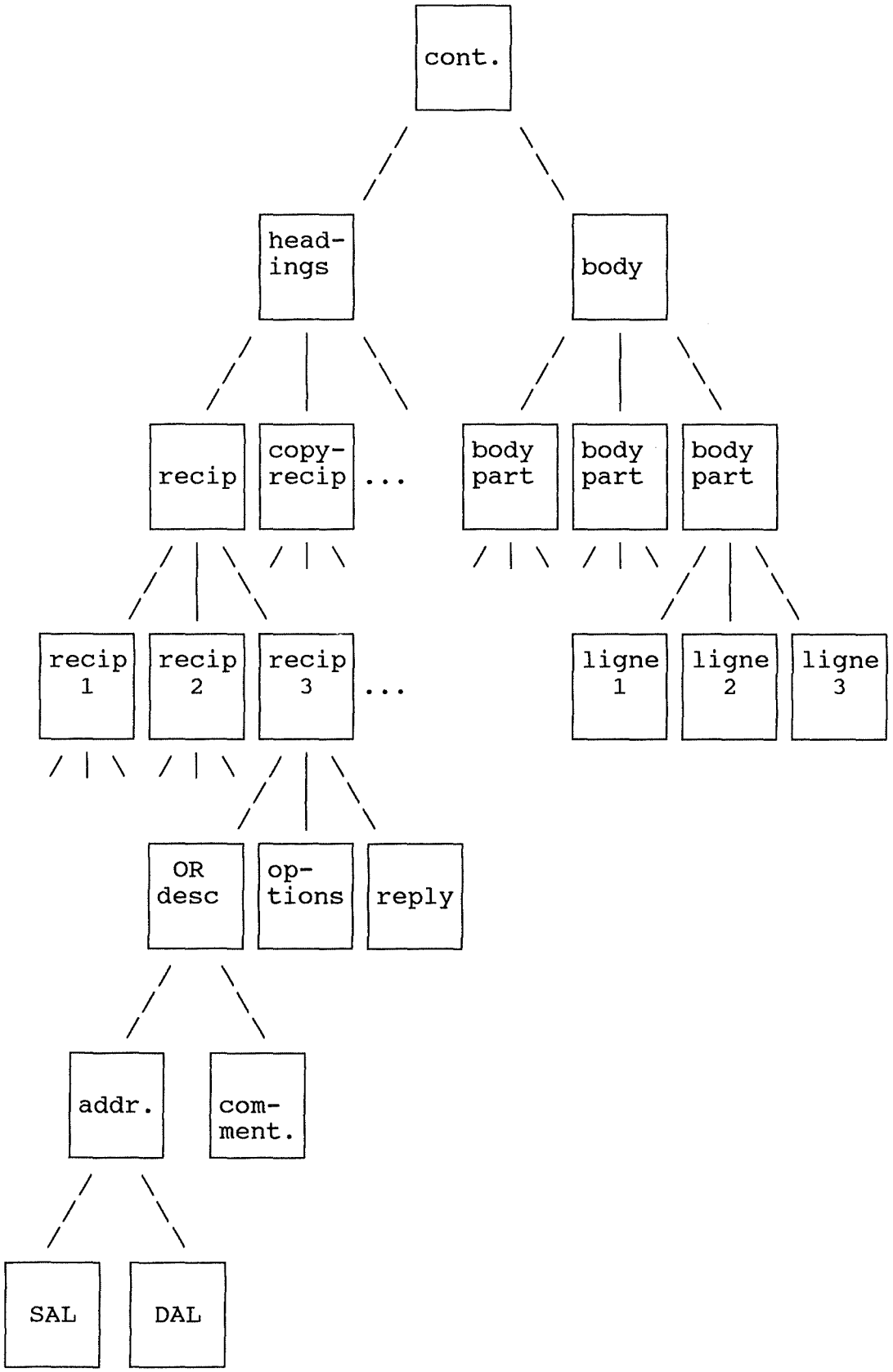


Figure 7.8: Représentation du contenu d'un message.

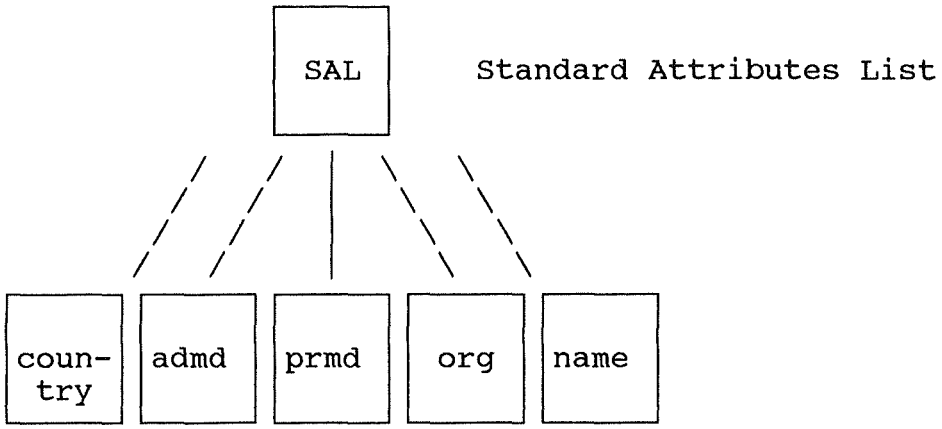


Figure 7.9: SAL (Standard Attributes List).

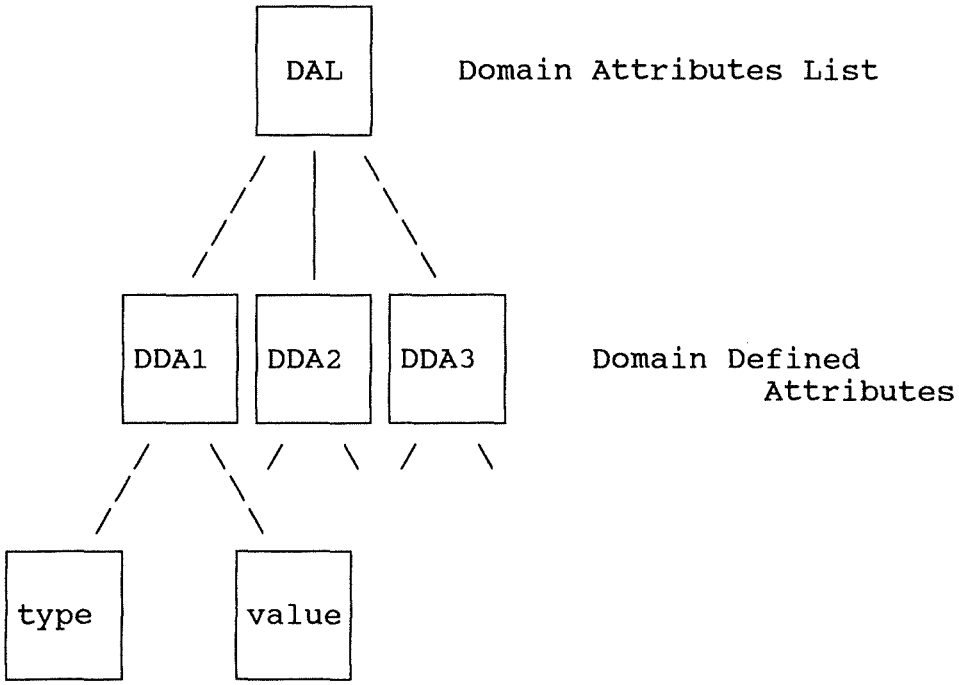


Figure 7.10: DAL (Domain Attributes List).

7.3.6. Le champ envelope

Le champ 'env' contient un pointeur vers un Set d'ENODEs qui constitue l'enveloppe du message (Figure 7.11).

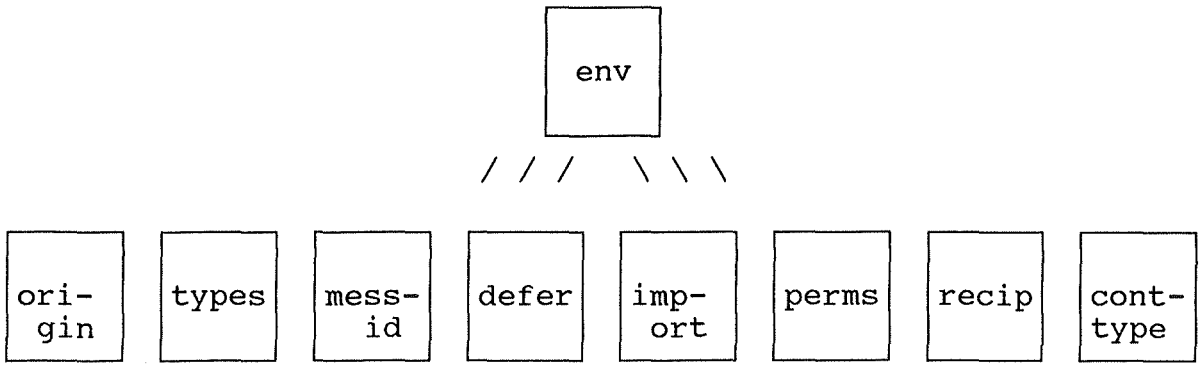


Figure 7.11 : Représentation de l'enveloppe d'un message.

L'enode	**origin**	contient l'adresse du destinateur,
	types	reflète le type d'information encodée,
	mess-id	identifie le message de façon univoque,
	defer	contient la date et l'heure auxquelles il faut délivrer le message au destinataire,
	import	reflète l'importance, ou sensitivité, du message,
	perms	contient les permissions concernant la conversion du message, le retour du contenu,...,
	recip	est un arbre contenant une liste de tous les recipients,
	conttype	, enfin, reflète la nature du contenu.

Remarquons que cette enveloppe n'est pas l'enveloppe X.400 (P1): c'est une "pseudo-enveloppe" utilisée de façon interne par EAN et non définie dans X.400. Elle est appelée "Enveloppe P2". La vraie enveloppe X.400 (P1) sera construite dans EAN à partir de l'enveloppe P2.

7.3.7. Les reports

Le champ 'reports' contient un pointeur vers une séquence de reports (Figure 7.12).

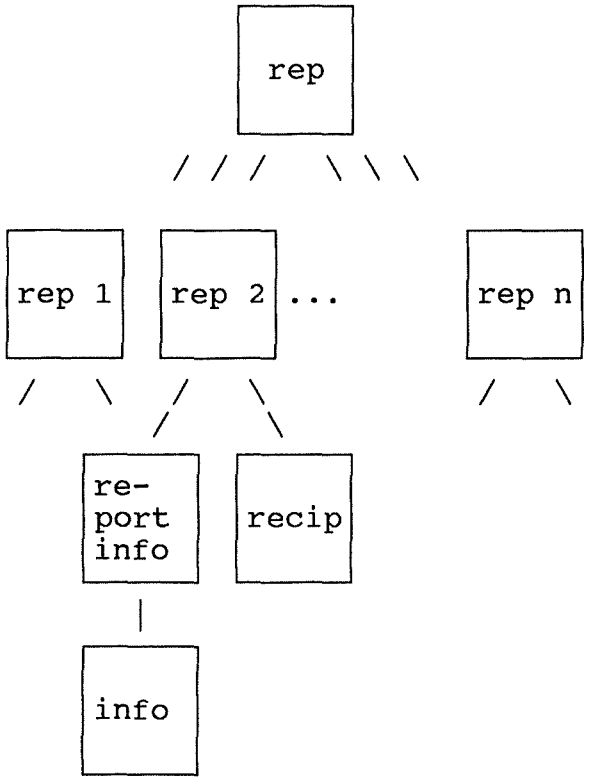


Figure 7.12: Représentation des reports d'un message.

Il y a un ENODE de report pour chaque recipient du message. L'ENODE info peut avoir plusieurs configurations:

- Si le message a été effectivement reçu par le recipient, alors l'ENODE info contient une 'receipt information' (Figure 7.13):

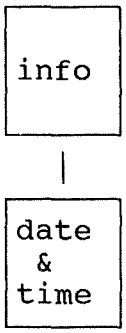


Figure 7.13: L'ENODE 'info'.

Le message affiché pour ce genre de report est le suivant: "Message delivered to <recip> at <date & time>".

- Si le message n'a pas été reçu par le recipient, alors l'ENODE info report contient une 'non-receipt information'; sa configuration est la suivante (Figure 7.14):

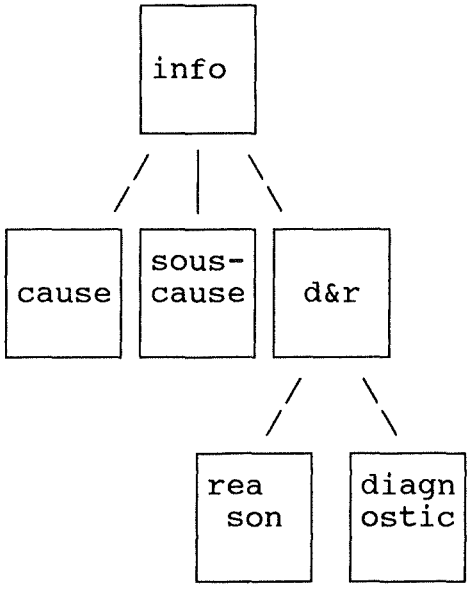


Figure 7.14: L'ENODE 'info'.

Trois cas peuvent se présenter:

- cause 'Discard': dans ce cas seuls les ENODEs 'cause' et 'sous-cause' sont présents dans l'ENODE 'info'; l'ENODE 'cause' contient la valeur P2_RR_DISCARD et l'ENODE 'sous-cause' une des cinq valeurs suivantes: P2_RQ_EXPIRED, P2_RQ_OBSOLETED, P2_RQ_TERMINATED, P2_RQ_SYSTEM ou P2_RQ_GATEWAY. Par exemple, la valeur P2_RQ_EXPIRED correspond au message suivant: "Not delivered at <recip> (expired)", ce qui veut dire que le message est arrivé au recipient mais trop tard pour lui être délivré.

- cause 'Forward': dans ce cas seul l'ENODE 'cause' est présent dans l'ENODE 'info' et contient la valeur P2_RR_AUTOFORWARD. Le message correspondant est le suivant: "Autoforwarded to <recip>".

- cause 'Mta-rejection': dans ce cas seuls les ENODEs 'd&r', 'diagnostic', 'reason' et 'cause' sont présents dans l'ENODE 'info'. L'ENODE 'cause' contient la valeur 2; l'ENODE 'reason' contient une des valeurs suivantes: P1_TRANSFERFAILURE, P1_UNABLETOTRANSFER ou P1_CONV-NOTPERFORMED. Quant à l'ENODE 'diagnostic', il contient une des valeurs suivantes: P1_UNRECOGNIZED, P1_MTACONGESTION ou P1_MAXTIMEEXP. Par exemple, le couple de valeurs P1_TRANSFER-FAILURE et P1_UNRECOGNIZED donnera un message "Not delivered to <recip> (Mta error - Bad address)".

Chapitre 8 : La trace d'un message

Le but de ce chapitre est de suivre le parcours d'un message à travers les différents processus d'EAN, et ce en examinant les fonctions C "rencontrées" par le message.

8.1. Introduction

Les sources de EAN étudiées ici sont celles que nous avons désignées sous le nom d'UBC2. Toutefois, les propos tenus dans ce chapitre sont aussi valables pour les versions UBC1 et DFN. Mais avant de rentrer dans le détail, examinons le schéma général du parcours d'un message dans le système EAN (Figure 8.1):

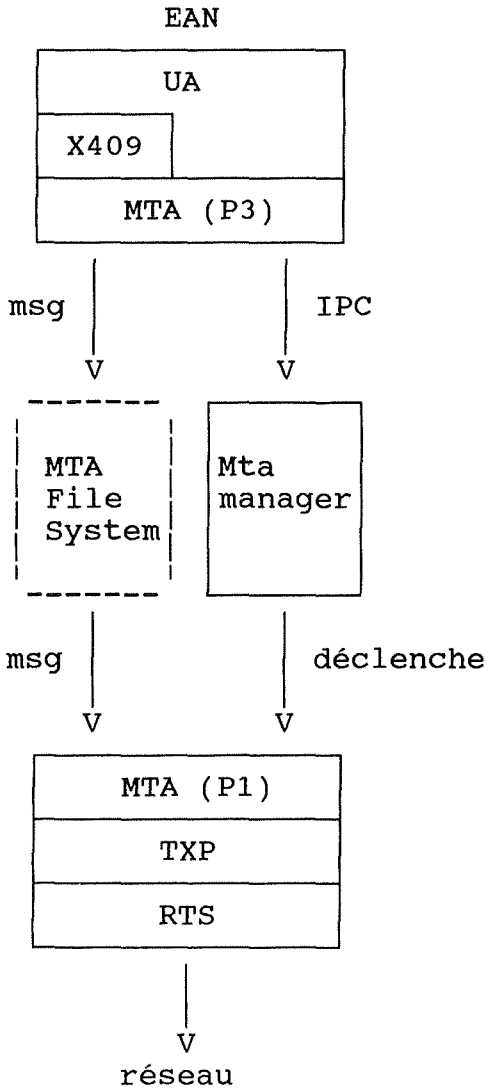


Figure 8.1: Parcours d'un message dans EAN.

Ce schéma s'interprète comme ceci: le programme **EAN** est lancé par l'utilisateur; le module **UA** construit le message et le passe au module **MTA (P3)**, directement pour l'enveloppe, via le module **X.409** pour le contenu. Le module MTA construit l'enveloppe P1, et stocke le message dans le **MTA File System** (pour plus de précisions, voir 2.5.2.A). Après avoir effectué l'algorithme de routage, il envoie un **IPC**[1] au **Mta manager** afin de lui dire à quels MTAs il faut envoyer le message, c'est à dire quelles Queue Files ont un message qui attend d'être envoyé.

Le **Mta manager**, qui est un processus permanent, crée des associations entre le MTA local et certains de ses MTAs adjacents (ceux à qui il faut envoyer le message). Pour chacun ce ces MTAs, il lance un processus **TXP**. Chaque processus TXP gère un dialogue fait d'échange de messages entre le MTA local et UN MTA adjacent. Il utilise à la fois le module **MTA (P1)** pour exporter/importer le message du/vers le MTA File System et le module **RTS** pour transférer UN message.

8.2. Trace du message dans le module UA

Rappelons tout d'abord que, pour exécuter le programme EAN, l'utilisateur doit utiliser la commande suivante :

ean [-options] [<recipient>]

Par exemple, pour ne pas accepter les éventuels messages qui ont été reçus, il faut spécifier l'option "a".

La procédure principale d'EAN, main, est appelée avec les paramètres de la commande ean. Cette procédure se charge de plusieurs initialisations, dont :

1) Le prompt est initialisé pour l'affichage.
2) Les options spécifiées sont mémorisées dans une variable flags (variable globale qui ne variera pas pendant l'exécution).
3) Le <recipient> est mémorisé dans la variable recip.
4) La fonction **ua** est appelée avec les paramètres flags et recip.

Nous allons distinguer trois grandes étapes dans l'exécution de la fonction **ua** :

- la première regroupe diverses initialisations,
- la deuxième étape est la boucle qui attend les commandes de l'utilisateur,
- et la troisième est consacrée à une commande en particulier, la commande "compose".

8.2.1. Les initialisations

La fonction ua s'occupe d'abord de plusieurs opérations d'initialisation dont voici le détail:

1) Une variable globale, Batchmode, est positionnée à 1 ou à 0 suivant que l'option batchmode a été spécifiée ou non dans la commande ean.

1. InterProcess Communication

2) Un appel à la fonction FL_init permet de créer une liste exhaustive de tous les identificateurs de champs. Précisons par un exemple ce qu'on entend par 'champ' et par 'identificateur de champ' :

To : <recipient>

est un champ, ou **heading**, dont l'identificateur est la constante P2_RECIPS. Le mot "To" est un **header**, identifié également par P2_RECIPS. Après cet appel à la fonction FL_init, la variable globale FL_all pointe vers une variable de type LIST (Figure 8.2):

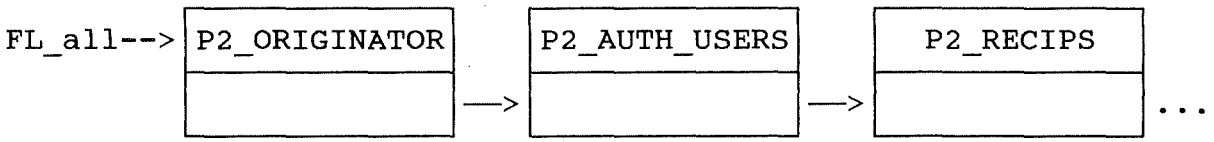


Figure 8.2: Liste des identificateurs de champs.

```
LIST { int      value
       LIST * next }
```

La liste des identificateurs repris dans cette liste est la suivante :

P2_ORIGINATOR	P2_REP_BY	P2_OBSOLETES
P2_AUTH_USERS	P2_INREPLYTO	P2_REFERENCES
P2_RECIPS	P2_IMPORTANCE	P2_EXPIRYDATE
P2_COPY_RECIP	P2_SENSITIVITY	P2_SUBJECT
P2_BCOPY_RECIPS	P2_MESSAGE_ID	
P2_REPTO	P2_AUTOFORWARD	

3) Le langage par défaut est sélectionné; il a été prédéfini dans le fichier ua/language.h :

define LANG_DEFLT LANG_ENGLISH

4) Si l'utilisateur a déjà été enregistré dans le Directory Service (cette vérification est réalisée par un appel à la fonction FoDirGet), alors la fonction Fsys_open ouvre son fichier profile; sinon, la fonction Ua_create crée un nouveau fichier et réalise l'enregistrement dans le Directory Service.

5) Une structure de données, UaProfDesc, de type PROF_DESC[1] et déclarée globalement, est initialisée par la lecture du fichier profile associé à l'utilisateur; cette initialisation est réalisée par la fonction ProfConfigure.

6) Le fichier 'inbox', qui est un folder, est ouvert par la fonction Fo_open. A partir de ce moment, et jusqu'à nouvel ordre, c'est le folder courant.

1. Le type PROF_DESC étant mentionné seulement ici, nous ne l'avons pas présenté; disons simplement que sa structure permet de mémoriser toutes les valeurs du fichier profile.

7) A moins que l'option "a" n'ait été spécifiée dans la commande ean, la fonction AcceptMessages se charge de l'acquisition des messages et de leur mémorisation dans le folder spécifié dans le fichier profile; si effectivement des messages sont arrivés, alors l'utilisateur en est averti.

8.2.2. La boucle d'attente des commandes

La fonction **InterpretCommands** réalise une boucle dont les étapes sont les suivantes :

1) Acquisition d'une commande de l'utilisateur (ex : "compose").

2) Appel de la fonction gérant spécifiquement la commande en question (ex : fonction ComposeCommand), en passant comme paramètre un pointeur vers une chaîne de caractères contenant les paramètres de la commande.

La fonction arrête de boucler lorsqu'une commande "quit" a été entrée par l'utilisateur; cette commande fait se terminer la fonction InterpretCommands et la fonction ua.

8.2.3. La fonction ComposeCommand(char * line)

Cette commande peut se scinder en deux parties distinctes: la première concerne la construction du message, la deuxième l'envoi du message. Selon que le mode d'exécution du processus est batch ou non, la construction du message différera. De plus, l'envoi du message en mode batch ne peut se faire par le biais de la commande "compose", mais bien par une utilisation ultérieure de la commande "send" .Nous allons donc considérer

- la construction du message en mode interactif
- la construction du message en mode batch
- l'envoi du message.

A) Construction du message en mode interactif.

Voici un résumé des fonctions rencontrées dans ce paragraphe:

```
NewDraft
Prmt_msg
    Prmt_list
        P2_find_hdng
        Prmt_hdng
            P2_hdng_name
            Pp_prompt
            Parse_hdng
                Alias_expand
                Bld_rlist
        P2_add_hdng
    Prmt_body
        Iogetline
        Add_IA5
        P2_add_bp
```

Rappelons que line pointe vers une chaîne contenant les paramètres de la commande compose c'est à dire :

> compose {header-list} {NUA-list}

└──────────────────────┘

paramètres

Nous allons examiner la fonction ComposeCommand en supposant qu'il n'y a pas de paramètres à la commande "compose" (line = NULL = 0), c'est à dire, typiquement, une exécution de EAN commencée comme suit :

$ ean
> compose

La fonction ComposeCommand va d'abord effacer le **Draft** s'il y en a un, et l'initialiser pour une nouvelle utilisation, en exécutant la fonction NewDraft. Si cette dernière renvoie la valeur False (i.e. le Draft n'a été ni envoyé ni sauvé dans un folder), alors la fonction ComposeCommand se termine et renvoie la valeur UA_RC_DRAFT_ALREADY; sinon, le texte du message (les headings et le body) est lu par la fonction **Prmt_msg**.

La fonction **Prmt_msg**(LIST * list1, LIST * list2, char * s)

Le rôle de la fonction Prmt_msg est d'acquérir un message et de le stocker dans le Draft. Avec les hypothèses prises çi-dessus,

list1 = Comp_FL (liste des identificateurs des headings à remplir lors d'une commande "compose" et spécifiés dans le fichier profile)
list2 = liste des identificateurs des headers de {header-list} = 0
char = {NUA-list} = ""

L'essentiel de cette fonction se résume aux trois instructions suivantes :

1. **Prmt_list(list1)** -- acquisition des headings dont l'identificateur est présent dans la liste list1
2. **Prmt_list(list2)** -- acquisition des headings dont l'identificateur est présent dans la liste list2
3. **Prmt_body(Pph_istd, Draft, NULL)** -- acquisition du corps du message.

1. La fonction **Prmt_list**(list * fl)

où fl est une liste d'identificateurs de headings. Pour chaque identificateur repris dans la liste fl, la fonction Prmt_list regarde si le heading correspondant à cet identificateur existe déjà dans la représentation interne du Draft (ceci est fait par la fonction **P2_find_hdng**). Si ce n'est pas le cas, alors il y a acquisition de la valeur de ce heading par un appel à la fonction **Prmt_hdng** et cette valeur vient garnir le heading du Draft, par un appel à la fonction **P2_add_hdng**.

a) La fonction **P2_find_hdng**(ENODE * ipm, int eid)

Si ipm pointe vers l'arbre de la représentation du contenu d'un message, alors cette fonction retourne un pointeur vers l'ENODE de type heading dont l'identificateur est eid.

Remarque : Vu la méthode de construction de l'arbre représentant le contenu d'un message, il ne peut y avoir au maximum qu'UN ENODE de type heading et ayant l'identificateur eid. S'il n'y en a aucun alors cette fonction retourne 0.

b) La fonction **Prmt_hdng**(long[1] id, int * ret_rc)

Soit id un identificateur de heading (par exemple, P2_RECIPS identifie le heading 'recipient' et le header "To"). La fonction Prmt_hdng demande à l'utilisateur d'introduire une valeur pour le heading identifié par id, construit l'ENODE correspondant, et renvoie un code de retour dans ret_rc. Examinons cette fonction un peu plus en détail:

- En premier lieu, il faut afficher le header pour que l'utilisateur sache quel est le type de heading pour lequel il lui faut entrer une valeur. La fonction P2_hdng_name(id) retourne le header identifié par id.

- Un appel à la fonction Pp_prompt(header) affiche le header à l'écran, se charge de l'acquisition d'une ligne introduite par l'utilisateur et retourne cette ligne (qui inclut le header lui-même).

- La troisième étape, et la plus importante, consiste à traiter cette ligne et à construire la structure de données mémorisant la (les) valeur(s) introduites par l'utilisateur. Ceci est réalisé par la fonction **Parse_hdng**.

Parse_hdng(long id, char * line, int * ret_rc)

La fonction Parse_hdng a pour but de garnir la représentation interne du message avec le heading de type id écrit dans la ligne line; elle renvoie un code de retour dans ret_rc.

Soit id un identificateur de heading
line une ligne de caractères.

Exemple : id = P2_RECIPS
line = "To : bbh, xyz"

On se limitera, pour la fonction Parse_hdng, à envisager le cas où id peut être un des trois identificateurs suivants: P2_BCOPY_RECIPS, P2_COPY_RECIPS, P2_RECIPS. Cette fonction se résume à deux étapes:

** Il faut d'abord traiter les alias éventuellement contenus dans line. La fonction Alias_expand(line) retourne une chaîne de caractères équivalente à la chaîne d'origine line, mais tous les alias ont été remplacés par leur écriture in extenso (de manière récursive - un alias pouvant avoir été défini à partir d'alias).

Comment cette traduction est-elle faite ? Une variable globale Alias_list pointe sur une variable de type NAMEDLIST,

1. Type d'entier codé sur 4 bytes.

```
NAMEDLIST { NAMEDLIST * next
            char * name
            char * value }
```

c'est à dire sur une suite de variables de type NAMEDLIST. Cette structure de données contient tous les alias définis dans le fichier profile de l'utilisateur :

name = alias
value = traduction de cet alias

** La fonction **Bld_rlist** est appelée en passant comme paramètre la chaîne de caractères retournée par Alias_expand. Elle renvoie un pointeur vers un arbre contenant tous les recipients contenus dans cette chaîne de caractères.

La fonction **Bld_rlist**(char * s, int * rc)

La fonction Bld_rlist examine la chaîne de caractères s, qui est une suite de recipients, renvoie un pointeur vers la liste de ces recipients, et renvoie un code de retour dans rc, s'il y a lieu.

Soit s = "s1,s2,s3,..." où s1, s2, s3 ... sont des recipients. Le résultat est le suivant (Figure 8.3):

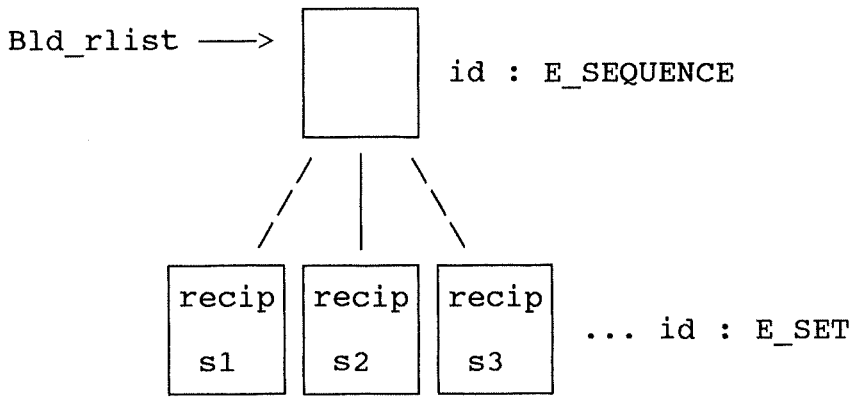


Figure 8.3: Résultat de la fonction Bld_rlist.

L'enode recip s1 est construit par un appel à la fonction **Bld_recip**(s1,rc). De manière générale, la fonction **Bld_recip**(char * r, int * rc) reçoit en argument une chaîne de caractères r dont la syntaxe est la suivante :

r = " c < a > [o1, o2, o3, ...] "

où c est un commentaire, a est l'O/Rname du recipient, et o1, o2, o3 ... sont des options de report. Le résultat de la fonction Bld_recip est le suivant (Figure 8.4):

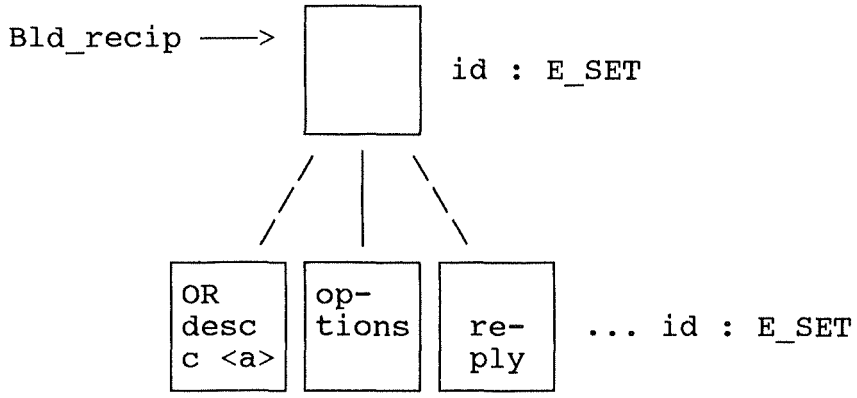


Figure 8.4: Résultat de la fonction Bld_recip.

L'ENODE ORdesc est construit par un appel à la fonction **Bld_ORdesc**("c<a>", rc). De manière générale, la fonction **Bld_ORdesc**(char * or, int * rc) reçoit en argument une chaine de caractères or qui peut avoir une des trois formes suivantes :

or = " a "
or = " c < a > "
or = " < a > "

Le résultat de la fonction Bld_ORdesc est le suivant (Figure 8.5):

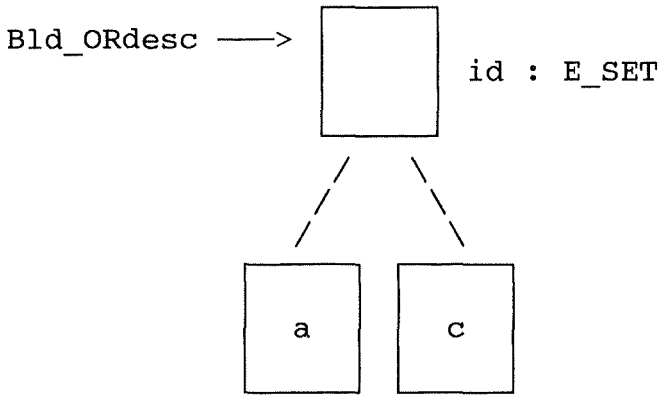


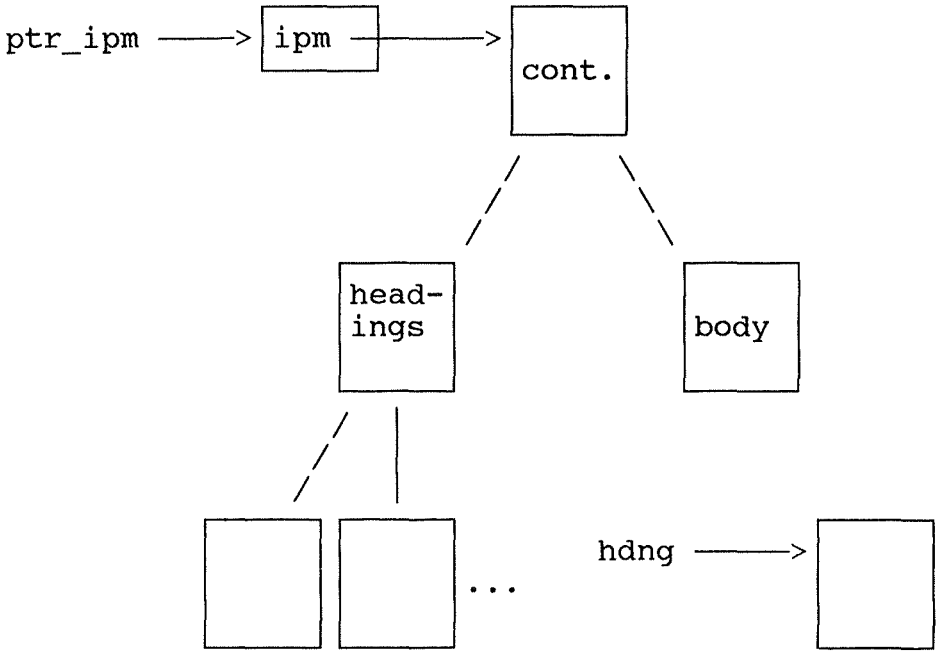
Figure 8.5: Résultat de la fonction Bld_ORdesc.

Et, enfin, l'ENODE contenant l'adresse a est construit par la fonction **Bld_ORname**(" a ",rc) - pour plus de détail sur la représentation d'un O/Rname, voir Figure 7.10, 7.11, 7.12).

c) La fonction **P2_add_hdng**(ENODE * * ptr_ipm, ENODE * hdng)

La fonction P2_add_hdng (Figure 8.6) a pour but de rajouter le heading pointé par hdng à la représentation interne du contenu d'un message, contenu pointé par ptr_ipm.

Soit, avant exécution :



Et après exécution :

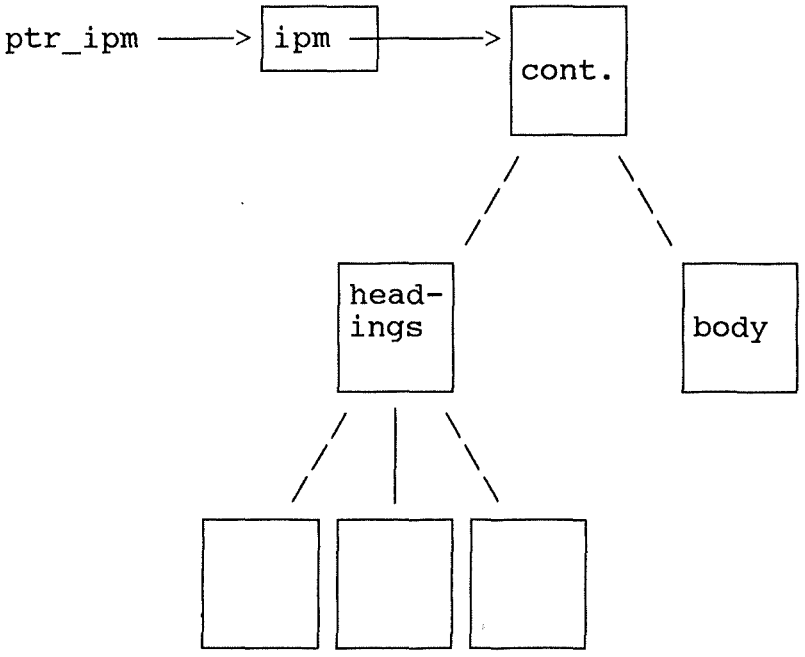


Figure 8.6: La fonction P2_add_hdng.

2. La fonction **Prmt_body** (IO_DESC * in, MESSAGE * m, char * one_line)

La fonction Prmt_body lit un texte dans in, fait avec ce texte un body part, et associe ce body part au message m (içi le message m est le Draft); chaque ligne du texte est lue par la fonction Iogetline et associée au body part par **Add_IA5**; quand le body part est terminé, il est associé au message m par la fonction **P2_add_bp**. Toutefois, si in = 0

alors le body part est seulement constitué de la ligne one_line et associé ensuite au message m. Il est à noter que cette fonction est appelée avec in = Pph_istd, le buffer d'input standard, c'est à dire le clavier.

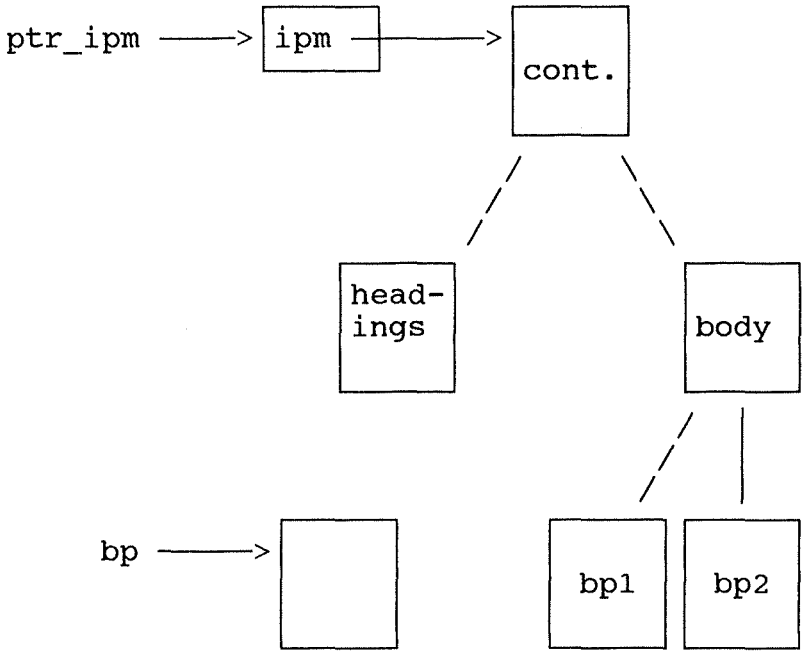
a) La fonction **Add_IA5**(ENODE * * part_ptr, ENODE * * end_ptr, char * s)

La fonction Add_IA5 a pour but de rajouter la ligne s au body part part_ptr. Si ce body part n'existe pas, il est créé automatiquement.

b) La fonction **P2_Add_bp**(ENODE * * ptr_ipm, ENODE * bp)

La fonction P2_add_bp (Figure 8.7) a pour but de rajouter un corps de message (body part) pointé par bp à la représentation interne du contenu d'un message, contenu pointé par ptr_ipm, et à la suite des body parts déjà existants (le body est une SEQUENCE de body parts)

Soit, avant exécution :



Et après exécution :

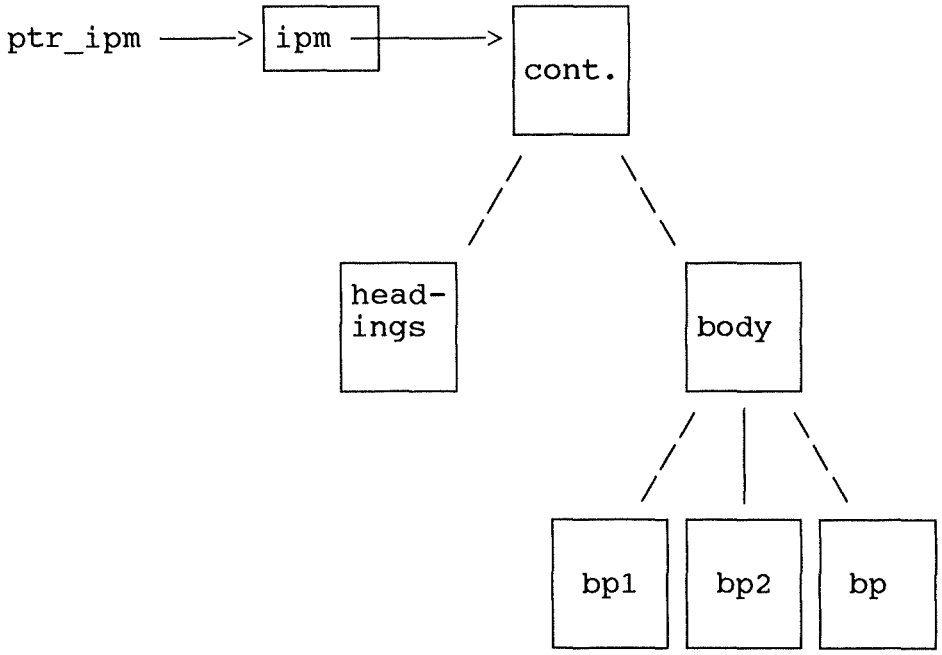


Figure 8.7: La fonction P2_add_bp.

B) Construction du message en mode batch

Voici un résumé des fonctions que l'on rencontrera dans ce paragraphe:

NewDraft
Read_msg
    Read_hdng
    P2_add_hdng
    Add_IA5
    P2_add_bp

Et rappelons que line pointe vers une chaîne contenant les paramètres de la commande compose c'est à dire :

> compose {header-list} {NUA-list}

paramètres

Comme précédemment, nous supposerons qu'il n'y a pas de paramètres à la commande compose (line = NULL = 0).

La fonction ComposeCommand va d'abord effacer le Draft courant s'il y en a un, en exécutant la fonction Me_free. Ensuite, le Draft est initialisé par la fonction NewDraft (rem : comme on est en mode batch, il est inutile de prévenir l'utilisateur que le Draft n'a été ni envoyé ni sauvé dans un folder, si c'est le cas; par conséquent, le Draft est effacé dans tous les cas). Le message entier se trouve dans le buffer Pph_istd; la fonction **Read_msg**(Pph_istd,Draft) se charge de le lire et, à partir de là, de construire le Draft.

La fonction **Read_msg**(IO_DESC * in, MESSAGE * m)

La fonction Read_msg se charge de l'acquisition d'un message dans le IO_DESC

in, et bâtit sa représentation interne dans m (ici, le Draft).

A la différence de la fonction Prmt_msg, cette fonction

- peut construire n'importe quel message et pas seulement le Draft.
- a à sa disposition TOUT le texte du message (headings + body), qui est censé se trouver dans in; pour rappel, la fonction Prmt_msg devait, elle, acquérir ce message au fur et à mesure de son exécution, d'une manière interactive avec l'utilisateur.

Les différents headings sont lus par la fonction **Read_hdng** et ajoutés à la représentation interne du message par la fonction **P2_add_hdng**; quant au corps du message, chaque ligne qui le compose est rajoutée dans un body part par la fonction **Add_IA5**. Ensuite, ce body part est ajouté à la représentation interne du message grâce à la fonction **P2_add_bp**.

La fonction **Read_hdng**(IO_DESC * in, long[1] id)

La fonction Read_hdng lit une ligne dans le buffer in (en l'occurrence le buffer d'entrée standard, le clavier). Cette ligne est un heading et la fonction **Parse_hdng** se charge de la traiter.

C) Envoi d'un message: la fonction SendNow()

Voici les différentes fonctions que nous rencontrerons dans ce paragraphe:

```
SendNow
    SendCommand
        UAL_send
            Conv_recip
            P3_s_env
            P3_submit
            Transmit
        Acc_reports
        P2_w_reports
```

Rappelons d'abord une des fonctionnalités d'EAN : lorsque l'utilisateur a terminé d'écrire son message (le Draft) avec la commande compose, le programme lui offre la possibilité d'envoyer son message immédiatement (seulement en mode interactif). Cette fonctionnalité est réalisée par la fonction SendNow(). L'essentiel de cette fonction réside dans les trois étapes suivantes :

1) affichage à l'écran de "Option ?".
2) acquisition de la <réponse> de l'utilisateur (réponse "send" pour envoyer le message immédiatement).
3) appel à la fonction **SendCommand**(<réponse>).

La fonction **SendCommand**(char * line)

La fonction SendCommand se charge d'envoyer le message contenu dans le Draft (c'est d'ailleurs la même fonction qui est appelée après que l'utilisateur ait entré la commande "send"). Les mots permis dans la ligne line sont les suivants :

1. Type d'entier codé sur 4 bytes.

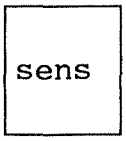
- les mots de la liste SendOpt_kwl : send, wait, after <aft_time>, before <bef_time>,...
- les mots de la liste p2_report_kwl : confirm, return, report,...
- les mots relatifs à l'importance du message : high, normal, low.
- les mots relatifs à la sensitivité du message : personal, private, confidential.

Le but de la fonction est de compléter le contenu du message avant de l'envoyer, en fonction des commandes/options contenues dans la ligne line.

1) Ajout du heading reflétant l'importance du message :

id : P2_IMPORTANCE

2) Ajout du heading reflétant la sensitivité du message :



id : P2_SENSITIVITY

3) Ajout du heading P2_EXPIRYDATE, si l'option "before <bef_time>" a été spécifiée :

id : P2_EXPIRYDATE

4) Ajout du heading P2_ORIGINATOR, s'il n'y en avait pas :

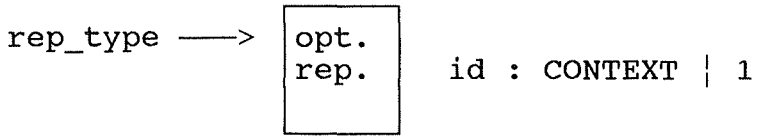
id : P2_ORIGINATOR

5) Ajout du heading P2_MESSAGEID, s'il n'y en avait pas. Ce heading comprend surtout un numéro qui identifie un message de façon univoque.

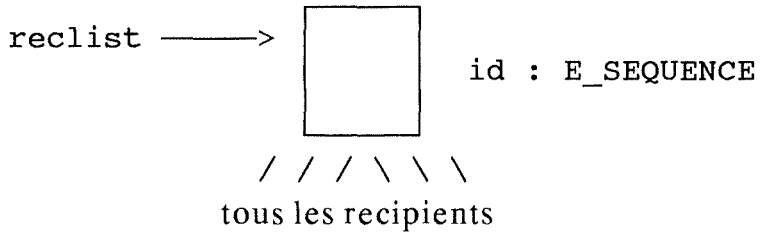
id : P2_MESSAGE_ID

Cet ajout est fait par la fonction Identify.

6) Construction d'un ENODE options de report, reflétant les options de la liste p2_report_kwl demandées dans la ligne line, s'il y en a :

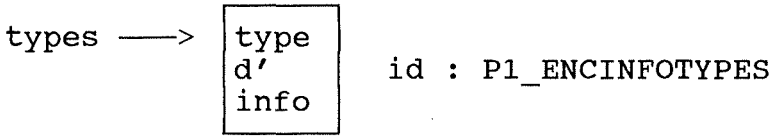


7) A partir de la représentation interne du message, construction d'un ENODE contenant TOUS les recipients ("to", "cc", "bcc") :



mais pour chaque recipient, l'ENODE options de report a été remplacé par l'ENODE rep_type, s'il existe. En effet, les options de report spécifiées par l'utilisateur après "Option ?" remplacent les options demandées précédemment.

8) Construction d'un ENODE indiquant le type d'information encodée dans le message :



L'envoi proprement dit est réalisé par la fonction **UAL_send**(Draft, types, TRUE, TRUE, aft_time, reclist), les reports sont acquis par la fonction **Acc_reports** et sont affichés grâce à **P2w_reports**.

La fonction **UAL_send**(m,types,convert,alternate,defer,recip)

avec	MESSAGE *	m	(le message à envoyer, conforme à P2)
	ENODE *	types	(le type d'information encodée)
	int	convert	(true si la conversion du contenu est autorisée)
	int	alternate	(true si les 'alternate' (voir 2.8.1) sont autorisés)
	T_REC *	defer	(delai avant la remise du message)
	ENODE *	recip	(liste de recipients)

Examinons les fonctionnalités de la fonction UAL_send :

1) L'enode P2_MESSAGE_ID est lu dans le contenu du message et pointé par mid.

2) Si recip = NULL, la liste des P2_RECIPS et des P2_COPY_RECIPS est créée en les lisant à l'intérieur même du contenu du message; elle est pointée par

recip. Une liste des P2_BCOPY_RECIPS est créée de la même manière et pointée par bcc. En outre les P2_BCOPY_RECIPS sont retirés du contenu du message;

si recip est différent de NULL, alors la fonction UAL_send considère recip comme la liste des recipients P2_RECIPS et P2_COPY_RECIPS (alors que la fonction SendCommand a passé en argument, pour recip, la liste des P2_RECIPS, des P2_COPY_RECIPS et des P2_BCOPY_RECIPS ! De plus, dans le cas où recip est différent de NULL, les P2_BCOPY_RECIPS ne sont pas retirés du contenu du message - voir en 5.6 le problème du bcc).

2) Un ENODE contenant l'ORname de l'utilisateur est créé; il est pointé par origin.

3) La liste de recipients est modifiée, de manière à ne garder que les informations nécessaires.

recip = **Conv_recip**(&recip, P3_BASIC, &returncont)

4) si cette liste de recipients n'est pas vide, alors l'enveloppe du message est créée grâce à

P3_s_env(origin, types, mid, defer, prio, convert, alternate, returncont, recip)

où prio est la valeur associée à l'enode P2_IMPORTANCE du contenu du message m (si cet enode est absent, alors la priorité vaut P2_I_NORMAL), et mid pointe vers l'ENODE P2_MESSAGEID du contenu du message m.

5) Cette enveloppe est soumise au MTA par la fonction **P3_submit** (pour plus de détails, voir 8.4.5).

6) Si P3_submit retourne la valeur OK (soumission acceptée), le contenu du message est envoyé à l'interface UA/MTA P3_desc par la fonction **Transmit** (pour plus de détails, voir 8.3.1).

7) Les recipients P2_BCOPY_RECIPS (la liste bcc) sont restaurés dans le contenu du message.

Remarque: Voyons la manière dont la fonction UAL_send traite les 'to' et les 'cc' différemment des 'bcc': L'UA soumet d'abord au MTA **UNE** enveloppe avec tous les 'to' et tous les 'cc', et le contenu comprend également tous les 'to' et tous les 'cc'. Ensuite, pour **CHAQUE** 'bcc', **UNE** enveloppe contenant seulement le 'bcc'

est soumise au MTA, le contenu comprenand les 'to', les 'cc' et le 'bcc' en question. En résumé, s'il y a n 'bcc', alors n + 1 messages seront passés au MTA: un message est envoyé à tous les 'to' et 'cc' et n messages seront envoyés, chacun à un 'bcc'.

a) La fonction **Conv_recip**(ENODE * * recip, int rep_level, int * returncont)

La fonction Conv_recip travaille sur l'arbre recip qui contient tous les recipients du message. Le but de cette fonction est double :

1) S'il n'y a pas d'option de report pour des recipients de la liste, elle leur ajoute l'option de report par défaut rep_level, en l'occurrence P3_BASIC.

2) Elle met returncont à TRUE si l'un au moins des recipients a demandé l'option P2_RETURN.

b) La fonction **P3_s_env**(ENODE * origin, types, mid, T_REC *[1] defer,long[2] import, int convert, alternate, returncont, ENODE * recip)

Cette fonction construit l'enveloppe d'un message suivant le schéma suivant (en fait on ne peut parler d'enveloppe qu'au niveau du MTA, il s'agit ici d'une simili-enveloppe que nous appellerons parfois "**enveloppe de type P2**" (Figure 8.8):

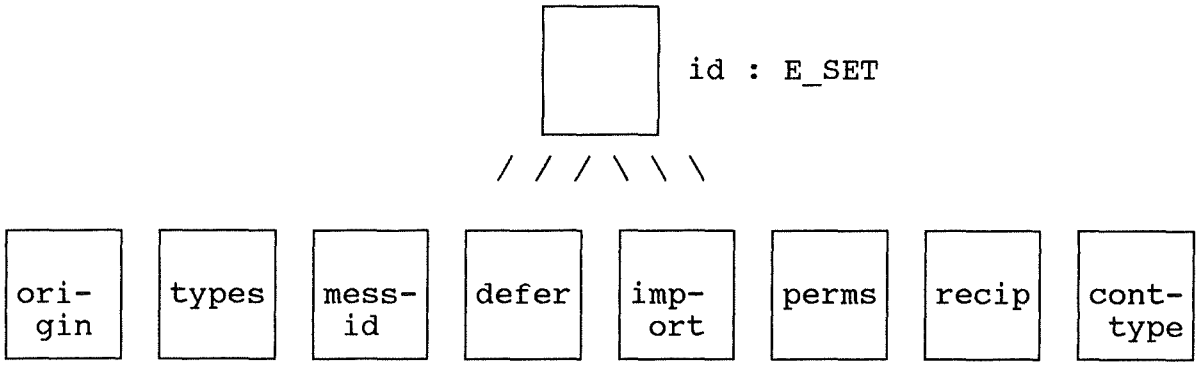


Figure 8.8: L'enveloppe de type P1.

L'enode **perms** contient les permissions convert, alternate et contreturn; l'enode **mess-id** est lu dans mid, qui rappelons-le est un pointeur vers l'enode P2_MESSAGE_ID du message. L'enode **defer**, enfin, n'est présent que si l'option "after <aft_time>" avait été demandée par l'utilisateur. Les identificateurs des enodes sont les suivants :

origin	P1_ORNAME
types	P1_ENCINFOTYPES
n -mess	P1_UACONTID
defer	CONTEXT + 0
import	P1_PRIORITY
perms	P1_PERMSFLAGS
recip	CONTEXT + 1
conttype	P1_CONTTYPE

1. Type de structure de données permettant de stocker une heure/date.
2. Type d'entier codé sur 4 bytes.

8.2.4. Le résumé des fonctions rencontrées

```
main
    ua
        FL_init
        ProfConfigure
        Fo_open
        Accept_messages
        InterpretCommands
            ComposeCommand en mode interactif
                NewDraft
                Prmt_msg
                    Prmt_list
                        P2_find_hdng
                        Prmt_hdng
                            P2_hdng_name
                            Pp_prompt
                            Parse_hdng
                                Alias_expand
                                Bld_rlist
                        P2_add_hdng
                    Prmt_body
                        Iogetline
                        Add_IA5
                        P2_add_bp
                SendNow
                    SendCommand
                        UAL_send
                            Conv_recip
                            P3_s_env
                            P3_submit
                            Transmit
                        Acc_reports
                        P2_w_reports
            ComposeCommand en mode batch
                NewDraft
                Read_msg
                    Read_hdng
                    P2_add_hdng
                    Add_IA5
                    P2_add_bp
```

8.3. Trace du message dans le module X409

8.3.1. Les fonctions utilisées

Nous avons vu que l'UA passait le contenu d'un message au MTA en l'écrivant dans le descripteur d'E/S P3_desc, et que cette écriture était faite par la fonction Transmit(ENODE * e). Celle-ci emploie deux fonctions du module X409 : ces deux appels sont, chronologiquement, X9_len(e) et X9_write(P3_desc, e). Examinons-les plus en détail.

8.3.2. La fonction X9_len(ENODE * e)

Rappelons que e pointe vers une structure en forme d'arbre, dont les noeuds et les feuilles sont des enodes. Prenons immédiatement un exemple détaillé (Figure 8.9):

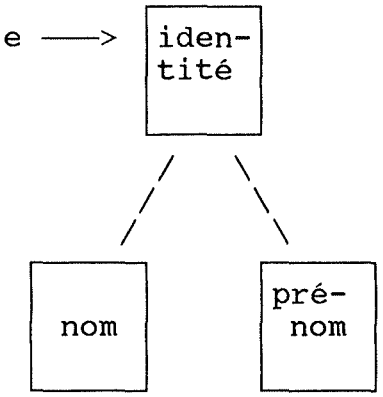


Figure 8.9: Exemple d'arbre de données.

et posons que :

1) identité est une variable de type Set
2) nom est une variable de type IA5String, de tag 1 et de valeur "Dylan"
3) prénom est une variable de type IA5String, de tag 2 et de valeur "Bob".

Ce schéma logique sera implémenté en mémoire par la structure d'ENODEs suivante (Figure 8.10):

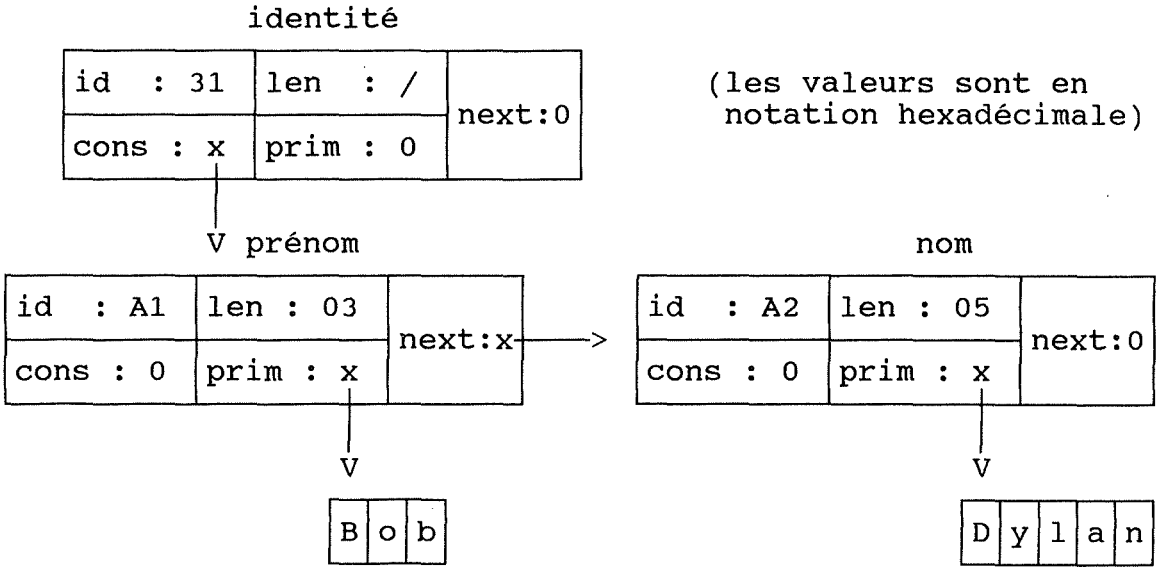


Figure 8.10: Exemple de structure d'ENODEs.

On voit que le champ 'length' de l'enode 'identité' n'est pas garni. En fait, ce champ contient une valeur valide seulement pour les enodes primitives. Avant de faire quoi que ce soit, il faut donc garnir le champ 'length' de tous les enodes constructor. Ceci est réalisé, précisément, par la fonction **X9_len**. Celle-ci se promène dans la structure e, garnit correctement le champ 'length' de chaque enode de type constructor, et renvoie le nombre de bytes requis par l'entièreté de la structure e en vue de son écriture en X409.

La fonction X9_len en appelle une autre, X9_lhdr(id, len), qui calcule le nombre de bytes nécessaires pour le codage en X409 des composants identificateur de valeur id et longueur de valeur len. Dans notre exemple, la démarche de X9_len pour l'enode 'prénom' sera la suivante:

```
length          =   03
lhdr(id, len)   =   02  ( 1 byte pour l'id, 1 pour la longueur)
                    ----
X9_len          =   05
```

Et pour l'enode 'nom':

```
length          =   05
lhdr(id, len)   =   02  ( 1 byte pour l'id, 1 pour la longueur)
                    ----
X9_len          =   07
```

La fonction X9_len est une fonction essentiellement récursive. Son exécution pour l'enode 'identité' sera la suivante:

```
X9_len('prénom')    =   05
X9_len('nom')       =   07
                        ----
X9_len('identité')  =   0C qui vient garnir le champ 'length' de l'enode
                           identité.
```

8.3.3. La fonction X9_write(IO_DESC * io, ENODE * e)

La fonction X9_write écrit la structure e sur le descriptif d'E/S io (c'est à dire, içi, P3_desc) conformément à la norme X409. Le champ 'length' pour chaque enode de la structure e est supposé contenir une valeur valide. De manière générale, cette fonction sort sur le descriptif io, successivement les composants id et length, et ensuite le contenu. La sortie des composants id et length se fait par un appel à la fonction X9_whdr(io, id, len). La sortie du contenu d'un enode primitive se fait par un appel à la fonction Ioputbuff(io, s, len). La sortie du contenu d'un enode constructor se fait, quant à elle, par un (des) appel(s) à la fonction X9_write, de manière récursive. Reprenons l'exemple précédent: l'exécution de X9_write(io, prénom) se déroulera comme suit:

```
X9_whdr(A1, 03)          : sortie sur io de l'identificateur de valeur A1
                           sortie sur io de la longueur de valeur 3
Ioputbuff(io, "Bob", 03) : sortie sur io de "Bob"
```

L'exécution de X9_write(io, nom) se déroulera de la manière suivante:

```
X9_whdr(A2, 05)          : sortie sur io de l'identificateur de valeur A2
                           sortie sur io de la longueur de valeur 5
Ioputbuff(io, "Dylan", 05) : sortie sur io de "Dylan"
```

Et l'exécution de X9_write(identité):

```
X9_whdr(31, 0C)
X9_write(prénom)
X9_write(nom)
```

Et, in fine, pour la structure identité, les bytes suivants seront écrits sur le descriptif io (en notation hexadécimale) :

31 0C A1 03 41 6F 62 A2 05 42 79 6C 61 6E

8.4. Trace du message dans le Module MTA

8.4.1. Introduction

Le point d'entrée de l'UA dans le MTA est le module P3. Celui-ci rassemble des fonctions qui permettent l'établissement d'une session entre l'UA et son MTA, la fermeture de cette session, ainsi que des fonctions de transfert de messages entre UA et MTA. Voyons-les plus en détail.

8.4.2. Le descripteur P3_desc

Ce descripteur, de type P3_DESC, est créé lorsqu'une session est établie entre l'UA et son MTA, reflète à tout moment l'état de la session, et contient des informations qui sont exploitées, soit par le MTA soit par l'UA. Il constitue en quelque sorte un interface.

```
P3_DESC: { IO_DESC *  io           ; descripteur d'E/S permettant de stocker le contenu du message
           int        state        ; état de la session : IDLE ou ACTIVE
           ENODE *    or           ; nom de l'utilisateur (O/Rname)
           char *     pw           ; mot de passe donné par l'utilisateur
           char *     qname        ; nom du fichier de files d'attente de l'utilisateur
           STORE      st           ; structure de données permettant de stocker un message
           char *     describe_str ; chaîne de caractères exprimant à un moment donné, le type d'opération effectuée par le module P3
         }
```

8.4.3. La fonction P3_open

précondition : /

paramètres :

ENODE *	or	-- NUA de l'originator
char *	password	-- son mot de passe
P3_DESC **	ptr_pid	-- pointeur vers P3_desc

rôle : le but de cette fonction est d'ouvrir une session de type P3 entre l'UA et son MTA, et ceci à la demande de l'UA.

résultat :

OK	:	session ouverte
RC_ERROR	:	le nom de l'utilisateur est inconnu
RC_SYSERROR	:	erreur système (une des tables - UA table, Connection Table, ...- n'existe pas); impossible d'ouvrir la session

primitive correspondante: LOGON (l'appel de la fonction coorrespond au REQUEST, le résultat de la fonction à la CONFIRMATION).

Détaillons le fonctionnement de la fonction P3_open; on peut y distinguer trois grandes parties: **l'initialisation du MTA, la reconnaissance de l'utilisateur**, et **la création/initialisation de P3_desc.**

A) Initialisation du MTA.

L'initialisation du MTA est faite par un appel à la fonction Mta_init; celle-ci comprend deux phases :

1) initialisation de l'algorithme de routage, par un appel à la procédure Rt_init. Celle-ci lit également le contenu de l'**Identification Table** et le mémorise dans la variable Rt_local, et initialise la liste Mta_list à la liste vide. Plus tard, celle-ci contiendra tous les MTA qui ont été résultat de l'algorithme de routage.

2) initialisation de la variable mta__expiry au temps d'expiration spécifié dans l'**Identification Table** (temps maximum de présence d'un message dans le MTA File System).

B) Reconnaissance de l'utilisateur

Il s'agit de s'assurer que l'UA de l'utilisateur est adjacent au MTA. Ceci est réalisé en exécutant l'algorithme de routage, après lui avoir passé en argument l'O/R name de l'utilisateur. Cet algorithme est réalisé par la fonction Rt_find(or). De manière générale, cette fonction, en se basant sur les tables du MTA, l'algorithme et or, rend un des quatre verdicts suivants:

- RT_UA : l'O/R name reçu en argument appartient à un UA adjacent au MTA. Dans ce cas, la fonction Lookup_ua va chercher dans la **Table Ua** les données relatives à cet UA (queue name, work, password), et positionne la variable **Rt_ua** avec ces données. En outre, il y a vérification de la correspondance entre le mot de passe password fourni par l'utilisateur or et le mot de passe spécifié dans la Table Ua pour ce même utilisateur.

- RT_MTA : l'O/R name reçu en argument n'appartient pas à un UA adjacent au MTA, mais l'algorithme a trouvé un MTA qui est "plus près" du recipient or. Dans ce cas, la fonction Lookup_mta va chercher dans la liste MTA_list les données relatives à ce MTA (queue name, netinfo), et positionne la variable **Rt_mta** avec ces données. Si ce MTA n'était pas membre de la liste MTA_list, alors les données sont lues directement dans la **Connection Table**. En outre, ce MTA est rajouté à la liste Mta_list, sauf s'il y était déjà.

- RT_UNKNOWN : Avec les données qu'il a à sa disposition, la procédure Rt_find est dans l'incapacité de déterminer un UA ou un MTA.

- RT_ERROR : Erreur dans l'algorithme de routage.

C) Création/Initialisation de P3_desc

Un descripteur de type P3_DESC est créé, son champ 'or' est garni avec la valeur or reçue en argument dans P3_open, son champ 'pw' est garni avec la valeur password, son champ 'qname' est garni avec le nom de la queue name spécifiée dans la Table Ua pour l'utilisateur or, et enfin son IO_DESC 'io' est initialisé.

8.4.4. La fonction P3_close

paramètres : P3_DESC** ptr_pid -- pointeur vers P3_desc

précondition : session ouverte

rôle : le but de cette fonction est de fermer la session de type P3 décrite par ptr_pid.

résultat : OK : session fermée.

primitive correspondante: LOGOFF (l'appel de la fonction coorrespond au REQUEST, le résultat de la fonction à la CONFIRMATION).

La fonction P3_close se résume à effacer le descripteur ptr_pid, et d'appeler la fonction Mta_term, qui se résume elle-même à la fonction Rt_term. Cette dernière efface les variables Mta_list, Rt_ua, Rt_mta, Rt_local.

8.4.5. La fonction P3_submit

précondition : une session a été préalablement ouverte par P3_open.

paramètres:

P3_DESC *	pid	--	pointeur vers P3_desc
ENODE *	args	--	"enveloppe" P2
ENODE **	ptr_result	--	résultat, où est mise notamment la date et heure de soumission de l'enveloppe au MTA.

rôle: l'essence même de cette fonction est de soumettre au MTA, via le paramètre args, une structure d'éléments contenant les informations nécessaires à la construction d'une enveloppe de type P1.

résultat:

OK	:	la soumission s'est exécutée correctement
RC_ERROR	:	l'utilisateur n'a pas de file d'attente
RC_SYSERROR	:	erreur système (il s'est avéré impossible de créer un fichier message sur le disque)

primitive correspondante: la fonction P3_submit implémente la primitive SUBMIT.

Détaillons le fonctionnement de la fonction P3_submit; il est axé autour du concept d'enveloppe de type P1. Divisons-le en deux parties: la **construction** de cette enveloppe, d'une part, et la **manipulation de l'interface P3_desc**, de l'autre part.

A) Construction de l'enveloppe P1

La structure args reçue en argument est déjà en elle-même, une simili-enveloppe; la fonction P3_submit ne fait que modifier cette structure. Entre autres, citons

- si args contient un ENODE 'defer' - c'est à dire une option spécifiant une date et une heure avant laquelle le recipient ne peut recevoir le message -, et que la

date et l'heure spécifiées dans cet ENODE sont périmées, alors cet ENODE est supprimé. En effet, dans ce cas-là, la contrainte exprimée par l'ENODE 'defer' sera toujours respectée et cet ENODE devient caduque.

- un ENODE 'traceinfo' est rajouté à la structure args. Cet ENODE contient une séquence d'informations de type 'trace', ce type pouvant être défini comme suit:

  type 'trace': - nom du MTA
  - la date et l'heure de réception du message par le MTA
  - le type d'action du MTA; cette action peut être de type relais ou reroutage.

- un ENODE type P1_EXTID est rajouté à la structure args; il contient, pour chaque recipient de l'enveloppe, un numéro d'identification. Par la suite, chaque MTA situé sur le parcours du message, rajoutera une information de type 'trace' supplémentaire.

B) Manipulation de l'interface P3_desc

- le champ 'state' est mis à la valeur ACTIVE.

- le champ 'describe_str' est garni d'une phrase indiquant, en langage naturel, le type d'opération associée à P3_desc (dans ce cas-ci, cette phrase sera semblable à "submit <mess-id> : <originator> <recip1> <recip2> ...".

- le champ 'store' est lui-même une structure de données complexe; les fonctions associées à cette structure en font une sorte de fichier virtuel. Ce store contient essentiellement un nom, créé automatiquement, et sous lequel sera sauvé, plus tard, le fichier contenant le message. Il contient également un IO_DESC, dans lequel est écrit le message, en représentation X.409, avant sa sauvegarde physique sur le support de sauvegarde. La fonction P3_submit s'occupe de deux choses: d'abord, elle entrepose l'enveloppe P1 du message, en représentation X.409, dans le IO_DESC du store, ensuite, elle fait en sorte que le IO_DESC de P3_desc et le IO_DESC du store ne fassent physiquement plus qu'un. De cette manière, lorsque plus tard, l'UA transmettra le contenu du message à l'IO_DESC de P3_desc grâce à la fonction Transmit, ce contenu se trouvera également dans l'IO_DESC du store, en représentation X409 et à la suite de l'enveloppe P1 du même message.

8.4.6. La fonction P3_end

précondition : une session a été préalablement ouverte par P3_open, et une opération de type submit, deliver, ... a été commandée (c'est à dire, P3_desc est dans l'état ACTIVE).

paramètres : P3_DESC * pid
int okay -- 0 pour avorter l'opération en cours
-- 1 pour confirmer l'opération.

rôle : La fonction P3_end a pour but de confirmer les données associées à P3_desc, et surtout, de confirmer l'opération y associée.

résultat :

OK	:	session ouverte
RC_USEERR	:	P3_desc est dans l'état IDLE alors qu'il devrait se trouver à l'état ACTIVE
RC_SYSERROR	:	Impossible de lire ou d'écrire un fichier message.

Pour continuer notre trace du message, nous allons supposer que l'opération en cours est un P3_submit.

La fonction P3_end va d'abord sauver physiquement le fichier virtuel associé au store de P3_desc, par un appel à la fonction **St_close**, et ensuite mettra le champ 'state' de P3_desc à la valeur IDLE, de manière à refléter le fait que P3_desc est libre pour une autre opération. Examinons la fonction St_close: elle se charge d'abord de la sauvegarde effective du message sur le disque. Ensuite, elle appelle la fonction **Db_msg**, qui s'occupe du 'management' des fichiers messages, et se termine en exécutant la fonction **Work_perform**.

A) La fonction **Db_msg**

Plus précisément, pour chaque recipient 'responsible' (voir 2.5.2 B pour la notion de 'responsible'), la fonction Db_msg effectue l'algorithme de routage sur ce recipient (comme nous avons vu au chapitre 3).

- Si le recipient est local au MTA (c'est à dire, connecté à un UA adjacent au MTA), alors le message est mis dans la file d'attente de l'UA recipient. Il restera, dès lors, à lancer le programme 'work' correspondant à l'UA recipient. Du côté recipient, c'est la fonction P3_deliver qui se chargera de transférer les fichiers messages de la file d'attente vers les fichiers de type 'folder'.

Remarques:

- En fait, le fichier message n'est pas stocké physiquement dans la file d'attente du recipient, mais une référence à ce fichier message y est mise.

- La file d'attente d'un utilisateur comprend en fait quatre 'queue files': une pour les messages urgents (ayant la priorité P1_URGENT), une pour les messages ayant une option 'defer' (c'est à dire, ne devant pas être livrés tout de suite), une pour les autres messages, et une pour les reports.

- Si par contre, le message est à router vers un autre MTA, alors le message est mis dans la file d'attente de ce MTA. Remarquons que si un message est à envoyer à plusieurs recipients mais à un seul MTA, alors le message n'est mis qu'une fois dans la file d'attente de ce MTA, l'enveloppe du message en lui-même contenant de toute façon la liste des recipients. Il faut toutefois faire exception du cas où un message comporterait, par exemple un recipient 'cc' et un recipient 'bcc', car alors le message serait envoyé en deux exemplaires, comme nous l'avons vu (8.2.3 C, fonction UAL_send), avec deux enveloppes différentes, et il serait mis deux fois dans la même file d'attente.

Remarque:

Il faut faire la même remarque que précédemment: un message n'est pas enregistré physiquement dans une file d'attente, seule une référence à ce message est enregistrée dans la file d'attente.

Cette opération devant se faire pour chaque recipient, il y a finalement un certain nombre d'UAs et un certain nombre de MTAs qui ont reçu le message dans leur file d'attente. Il s'agit maintenant, pour chacun de ces UAs, de lancer le programme 'work' correspondant, et de faire connaître au Mta manager la liste de ces MTAs. Cette dernière étape est réalisée par un appel à la fonction **Work_perform.**

B) La fonction **Work_perform**

La fonction Work_perform examine d'abord l'enveloppe du message. Si celui-ci a une priorité P1_NONURGENT, alors la fonction ne lance aucun programme 'work', mais mémorise les couples (ua, work) dans une variable de type liste de manière à pouvoir lancer les processus ultérieurement. Si l'enveloppe a une priorité P1_URGENT ou P1_NORMAL, alors les programmes 'work' de chaque UA sont lancés successivement.

Enfin, un IPC[1] du type "N <mta1> <mta2> <mta3> ..." (N comme New) est passé au programme Mta manager, qui s'occupera de faire parvenir le message aux MTA mta1, mta2, mta3, ...

8.4.7. La fonction P3_deliver

paramètres : P3_DESC * pid
ENODE * * ptr_arg -- résultat: l'enveloppe P1 reçue.

précondition : une session a été préalablement ouverte par P3_open.

rôle : La fonction P3_deliver a pour but de recevoir du MTA un message destiné à l'UA.

résultat :

OK	:	la réception de l'enveloppe s'est correctement passée
RC_ERROR	:	l'utilisateur (recipient dans ce cas) n'a pas de file d'attente
RC_END	:	Il n'y a plus, dans la file d'attente, de messages à réceptionner.

primitive correspondante: la fonction P3_deliver implémente la primitive DELIVER. Il faut remarquer que c'est le MTA qui est initiateur de la primitive DELIVER, alors que dans notre cas, c'est l'UA qui est initiateur de la fonction P3_deliver.

8.4.8. La fonction P3_notify

C'est une fonction analogue à P3_deliver, mais pour les notifications. Elle implémente la primitive NOTIFY, avec la même remarque que pour P3_deliver.

8.4.9. La fonction P3_register

paramètres : P3_DESC * pid
char * work

précondition : une session a été préalablement ouverte par P3_open.

rôle : La fonction P3_register a pour but d'enregistrer un nouvel utilisateur dans la table des UA. Le nom de l'utilisateur et son password sont contenus dans le P3_DESC * pid, le work program à lui associer est passé en paramètre.

---

1. InterProcess Communication

résultat : OK : l'utilisateur est, ou était déjà enregistré
RC_SYSERROR : impossibilité d'effectuer l'enregistrement.

primitive correspondante: la fonction P3_register implémente la primitive REGISTER.

8.4.10. La fonction P3_changepw

précondition : une session a été préalablement ouverte par P3_open.

paramètres : P3_DESC * pid
ENODE * args -- structure contenant le nouveau et le précédent mot de passe.

rôle : La fonction P3_change_pw a pour but de changer le mot de passe d'un utilisateur identifié par le champ 'or' de pid.

résultat : OK : password changé
RC_SYSERROR : le mot de passe précédent contenu dans args est incorrect.

primitive correspondante: la fonction P3_changepw implémente la primitive CHANGE-PASSWORD.

8.5. Trace du message dans le programme Mta manager

Le Mta manager est un processus qui gère des associations entre le MTA local et ses MTAs adjacents; il s'exécute de façon permanente, attendant un IPC pour s'exécuter d'une manière particulière. Il faut signaler que EAN n'est pas le seul processus pouvant envoyer des IPCs au MTA manager: le MTA maint (programme de maintenance), et les processus TXP le peuvent également

La permanence du processus est assurée, dans le corps du programme, de la manière suivante:

```
main()
    repeat { .
             .
             rc = Mtamngr() ;
             if rc != SV_RC_RESTART break ;
             .
             .
           }
```

C'est-à-dire, la fonction Mtamngr s'exécute indéfiniment, jusqu'à ce qu'elle retourne un résultat différent de SV_RC_RESTART. En fait, c'est à l'intérieur de la fonction Mtamngr que la permanence du processus est assurée, mais elle peut se terminer et être réexécutée pour autant qu'elle ait retourné la valeur SV_RC_RESTART.

8.5.1. Le coeur du programme Mta manager

Le coeur du module est constitué de la liste Schd_calltq, de type TIMEQ.

```
TIMEQ    {    struct TIMEQ *      next    -- champ servant à l'organisation de la
                                             liste.
              struct TIMEQ *      prev    -- champ servant à l'organisation de la
                                             liste.
              t_int[1]            time     -- heure/date à laquelle il faudrait
                                             créer une association avec un MTA
                                             adjacent.
              UNIV *[2]           work    -- pointeur vers un descripteur de ce
         }                                   MTA.
```

Le descripteur d'un MTA adjacent est une structure de données contenant les informations sur l'association (ou l'absence d'association) entre le MTA local et un MTA adjacent. Il comprend quatre types d'information:

- information 'identité' : nom du MTA adjacent
- information 'configuration' : données sur les intervalles de temps séparant deux tentatives successives d'établissement d'une association.
- information 'status' : notamment un pointeur vers une structure de type TIMEQ, un drapeau indiquant s'il y a des messages dans la file d'attente de ce MTA adjacent, le mode d'association (initiative du MTA local ou réponse), un av_id (activity identificator, voir 3.5.2 B).
- information 'statistiques' : nombre d'essais, de succès dans la réalisation d'une association, heure du dernier essai, du dernier succès.

Considérons un exemple de configuration de la liste Schd_calltq (Figure 8.11):

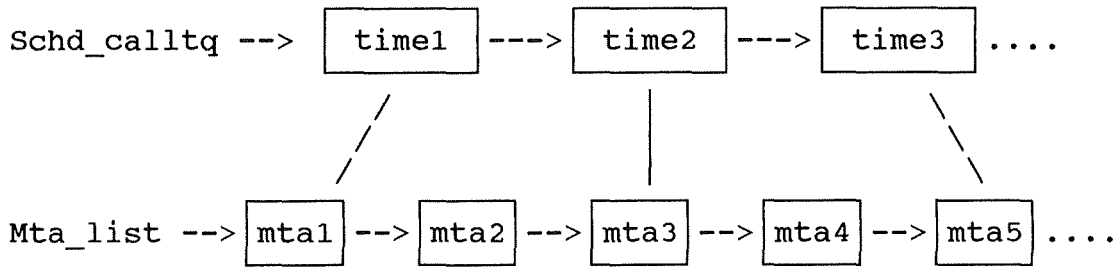


Figure 8.11: Exemple de configuration de la liste Schd_calltq.

A tout moment, la liste Schd_calltq est organisée par ordre de 'time' croissant. Dans l'exemple que nous avons choisi,

- au temps time1, il faudra essayer d'établir une association avec le mta1.
- au temps time2, il faudra essayer d'établir une association avec le mta3.
- au temps time3, il faudra essayer d'établir une association avec le mta5.

1. Le type t_int est une structure permettant de stocker une date&heure.
2. Type Universal.

Il y a également une autre liste, Schd_qchktq, organisée de la même manière que la liste Schd_calltq, et qui concerne les associations ouvertes à l'initiative des MTAs adjacents, contrairement à la liste Schd_calltq qui concerne les associations à établir à l'initiative du MTA local. Remarquons que, comme les deux listes partagent la même liste Mta_list, et comme un descripteur de MTA ne comprend qu'un seul champ 'TIMEQ*', il ne peut y avoir à un moment donné qu'une seule association entre le MTA local et le MTA adjacent.

8.5.2. La fonction Mtamngr()

La fonction Mtamngr exécute un mécanisme que nous diviserons en quatre parties: les **initialisations**, et la boucle principale qui se compose des fonctions **Mm_work**, **Sv_wait**, et **Service_request**.

A) Les initialisations

Nous distinguons deux sortes d'initialisations:

- celles qui sont réalisées par la fonction Sv_init; elles concernent le service d'IPC.
- celles qui sont réalisées par la fonction Mm_init; elles concernent surtout la liste Mta_list. Parmi celles-ci, citons:

  - la constitution de la liste Mta_list; celle-ci contient un descripteur pour chacun des MTAs adjacents au MTA local.
  - pour chaque descripteur de MTA, l'information 'configuration' est initialisée d'après les données lues dans la Connection Table pour ce MTA.
  - pour chaque descripteur de MTA, l'information 'statistiques' est initialisée d'après les données lues dans le fichier de statistiques de ce MTA.

B) La fonction Mm_work

La fonction Mm_work s'occupe d'ouvrir et de fermer les associations entre le MTA local et un des MTAs adjacents de la liste Mta_list. La liste Shd_calltq est parcourue depuis le début, et pour chaque membre ayant un 'time' inférieur ou égal au temps courant, une association est ouverte entre le MTA local et le MTA adjacent pointé par le champ 'work' de ce membre. Ce membre est alors retiré de la liste Schd_calltq.

Une association est ouverte en lançant un processus TXP en appelant la fonction Av_start. Dans le cas où cette association est créée à l'initiative du MTA local, les paramètres passés à la fonction Av_start sont:

- le nom du MTA adjacent,
- le mode 'INITIATE',

et le résultat retourné est un av_id (activity identificator) identifiant le processus TXP lancé. L'information 'status' du descripteur du MTA adjacent est mise à jour (mode 'INITIATE', av_id). A partir de ce moment, le descripteur du MTA adjacent reflète qu'il est 'en activité', et ce jusqu'à la fermeture de l'association.

La fonction Mm_work retourne le délai avant la prochaine association à établir.

C) La fonction Sv_wait

La fonction Sv_wait attend pendant un certain temps un IPC. Ce temps est précisément le délai renvoyé par la fonction Mm_work.

D) La fonction Service_request

Nous allons détailler la fonction Service_request dans le cas où le MTA manager reçoit du processus EAN l'IPC "N <mta>", ce qui traduit l'existence, dans la queue file de ce mta, d'un nouveau message à envoyer. Dans ce cas, la fonction **Mm_new** est exécutée. Cette fonction insère un élément TIMEQ dans la liste Schd_calltq, avec une valeur 'time' égale au temps courant. De cette manière, la fonction Mm_work exécutée ensuite ouvrira une association immédiatement.

8.6. Trace du message dans le programme TXP

8.6.1. Le lancement du programme TXP

Le programme TXP implémente un échange de messages entre le MTA local et un MTA adjacent. Il se compose surtout de deux fonctions:

- la fonction Txp_initiate est appelée si le programme TXP a été lancé avec l'option 'INITIATE'. Dans ce cas, les arguments passés à la fonction Txp_initiate sont les suivants:

  - set_lock : variable mise à 1 ou à 0 suivant que l'option 'l' était présente ou non dans la commande TXP. Si elle était présente, le processus n'enverrait pas d'IPCs au Mta manager (par exemple, pour signaler la fin du processus TXP). Mais, comme nous suivons ici la trace d'un message et que c'est le Mta manager lui-même qui a lancé le processus TXP, l'option 'l' est absente (il faut que le processus TXP informe le Mta manager de la fin de son activité, afin que celui-ci puisse détruire l'association).

  - le nom du MTA adjacent (passé en argument de la commande TXP).

- la fonction Txp_respond est appelée si le programme TXP a été lancé avec l'option 'RESPOND'.

Afin de continuer à suivre notre message, nous allons considérer le cas où le Mta manager a appelé le processus TXP avec l'option 'INITIATE', et en passant comme argument le nom du MTA adjacent.

8.6.2. La fonction Txp_initiate

Appelons mtar, le nom du MTA adjacent que Txp_initiate reçoit en argument. Voici d'abord un exemple de l'enchaînement des fonctions que nous rencontrerons ici (en ne retenant que l'essentiel de leurs arguments) :

```
P1_open(mtar, ..., &mtaP1)
Rts_open(..., &rts)
Txp_actinit(mtaP1, rts, ...)

     Export_msg(mtaP1, rts)   / 1. P1_export(mtaP1, msg)
     Export_msg(mtaP1, rts)  /  2. IOCTLN(rts, RTS_CTL_W_BEGIN)
     Export_msg(mtaP1, rts) /   3. Write_env(rts, msg)
           .                \   4. IOtfr(rts, mtaP1)
           .                 \  5. IOCTLN(rts,RTS_CTL_W_END)
           .                  \ 6. P1_end(mtaP1, ...)
           .
     IOCTLN(rts, RTS_CTL_TURNGIVE)

     Import_msg(mtaP1, rts)   / 1. IOCTLN(rts, RTS_CTL_R_BEGIN)
     Import_msg(mtaP1, rts)  /  2. Read_env(rts, msg)
     Import_msg(mtaP1, rts) /   3. P1_import(mtaP1, msg)
           .                \   4. Txfr_contents(rts, mtaP1)
           .                 \  5. P1_end(mtaP1)
           .                  \ 6. IOCTLN(rts, RTS_CTL_R_ACCEPT)
Ioclose(rts)
P1_close(&mtaP1)
```

Nous voyons qu'ici, une session de type P1 entre le MTA local et le MTA adjacent est ouverte par la fonction P1_open et fermée par P1_close. Cette session est décrite à tout moment par P1_desc, de type P1_DESC:

```
P1_DESC:{ IO_DESC  io      IO_DESC pour stocker le contenu d'un message
          int      type    état de la session: IDLE, IMPORT ou EXPORT
          char *   qname   nom de la file d'attente du MTA adjacent
          STORE    st      structure de données permettant de mémoriser
                           le contenu d'un fichier message.
```

Au cours de cette session, les messages dans la file d'attente de mtar sont envoyés un à un à mtar par des appels à Export_msg. Quand il n'y a plus de messages à envoyer, des messages sont reçus de la part de mtar, s'il y en a, chacun par un appel à la fonction Import_msg.

Le programme TXP utilise le service offert par le module RTS. La structure de données accessible à la fois par TXP et RTS, pour se passer des messages (qui sont vus par le RTS comme une suite de bytes), est un IO_DESC de nom rts. D'abord, la fonction Txp_initiate signale au RTS qu'elle désire utiliser son service, par Rts_open, et en passant comme paramètre le mode 'RTS_INITIATE' (entre autres - voir 8.6.3). Nous pouvons alors détailler la chronologie du transfert d'UN message vers le MTA adjacent (fonction Export_msg):

1. D'abord, le fichier message qui doit être envoyé est lu et écrit dans P1_desc via la structure STORE qui implémente une sorte de fichier virtuel. L'enveloppe P1 du message est stockée dans la structure 'msg' et le contenu du message dans le IO_DESC. Le champ 'type' de P1_desc est mis à 'EXPORT'.

2. Ensuite, la fonction Export_msg fait savoir au RTS qu'elle va écrire un message

dans rts.

3. L'enveloppe stockée dans msg est écrite dans rts.

4. Le contenu du message est écrit dans rts.

5. La fonction Export_msg fait savoir au RTS qu'elle a terminé d'écrire son message dans rts.

6. Le champ 'type' de P1_desc est mis à 'IDLE'.

Lorsque tous les messages ont été envoyés, le RTS est averti qu'il doit donner son tour au MTA adjacent et recevoir des messages de la part de celui-ci (RTS_CTL_TURNGIVE). Détaillons alors le transfert d'UN messsage du MTA adjacent vers le MTA local (fonction Import_msg):

1. D'abord, la fonction Import_msg fait savoir au RTS qu'il va lire un message dans rts.

2. Ensuite, l'enveloppe P1 est lue dans rts et stockée dans msg.

3. L'enveloppe stockée dans msg est écrite dans le descripteur mtaP1 (plus précisément, dans la structure STORE).

4. le contenu est lu dans rts et écrit dans le descripteur mtaP1 (plus précisément, dans la structure STORE, via le IO_DESC).

5. Le message entier est sauvé dans un fichier message.

6. Le RTS est averti que le message a été lu.

8.6.3. Le module RTS

Nous n'allons plus, ici, parler du fonctionnement des fonctions mais nous limiter aux paramètres que le RTS reçoit de TXP lorsque celui-ci lance un Rts_open. Mais avant de les passer en revue, il nous faut reparler des informations contenues dans la partie 'network info' de la CONNECTION TABLE (voir 3.1.2). Cette information comprend quatre parties: MTA-info, RTS-and -session-info, Transport-info et Network info. Les paramètres de Rts_open sont:

- RTS_INITIATE, comme on l'a vu en 8.6.2.

- certains paramètres lus dans MTA-info:

  - le nom du MTA local; ce nom servira uniquement à faire savoir au MTA adjacent quel est le MTA qui l'appelle.

  - le mot de passe à présenter au MTA adjacent (facultatif).

  - le nom par défaut du ss_user (session user): dans notre cas le ss_user est le MTA local.

  - une option 'MONOLOGUE': si cette option est présente alors le RTS refusera de recevoir les messages provenant du MTA adjacent.

- les informations RTS-and-session-info, Transport-info et Network-info.

La fonction Rts_open va alors ouvrir une session par ss_open, en passant notamment comme paramètre les informations Transport-info et Network-info. Ces mêmes informations seront transmises au service Transport via la fonction Tran_open qui, à son tour, lancera la fonction (*net->open) en lui passant, entre autres, Network-info. Ce dernier paramètre contiendra, par exemple, l'adresse X25 du MTA adjacent.

Conclusions

EAN est un logiciel implémentant une bonne partie des fonctionnalités prévues par la recommandation X.400, aujourd'hui largement acceptée dans le monde. Si la version originale de EAN ne supportait pas l'adressage par attributs standards X.400, cette anomalie a été gommée dans la version DFN.

Bien que peu documentées, les sources de EAN sont en général relativement claires et bien structurées (utilisation de noms significatifs). L'architecture globale du logiciel est convenablement décrite dans les manuels.

L'installation et la configuration de EAN ne se font pas toujours sans problèmes, et il est important de bien connaître la structure du logiciel pour comprendre son fonctionnement. Notons à ce propos qu'il serait utile de recompiler une version complète de EAN sur Junon "ts.info", afin de pouvoir y bénéficier de TCP/IP et du Directory Service.

Par contre, une fois installé, EAN se révèle très agréable à utiliser. Il semble également très fiable.

Il serait souhaitable que les prochaines versions de EAN accroissent le nombre de fonctionnalités X.400 implémentées (1984 mais aussi 1988), et utilisent un Directory Service conforme à X.500. De plus, d'autres UAs pourraient être développés dans le cadre de EAN, par exemple pour supporter l'EDI. La structuration de EAN ne devrait pas rendre ces évolutions trop difficiles à intégrer.

Il serait également intéressant de comparer EAN avec des logiciels X.400 apparus plus récemment.

Bibliographie

[AT] *Computer Networks*, Andrew S. Tanenbaum, Prentice-Hall, 1988.

[EAN1] *EAN: a distributed message system*, Gerald Neufeld, Proceedings Canadian Information Processing Society National Meeting, Ottawa, pp. 144-149, May 1983.

[EAN2] *EAN: an X.400 message system*, Gerald Neufeld, John Demco, Brent Hilpert et Rick Sample, Department of Computer Science, University of British Columbia, July 10, 1985.

[EANDFN] *Message Handling System DFN-EAN V2.1 : Administrator's Manual for DEC/VMS (Version 1)*, Volker Denzer, Andreas Dittrich et Thomas Magedanz, Fokus - GMD Berlin, 1988.

[EANMAN] *The Ean Distributed Message System User's Manual (Version 2.1)*, Gerald Neufeld, University of British Columbia (Vancouver, Canada), 1987.

[EANUBC] *The Ean Distributed Message System Administrator's Guide for BSD UNIX (Version 2.1)*, University of British Columbia (Vancouver, Canada), 1987.

[JC] *ISO 10021 - X.400(88) : A Tutorial for Those Familiar with X.400(84)*, Jim Craigie, Computer Networks and ISDN Systems, Vol. 16, pp. 153-160, 1988-1989.

[KR] *The C programming language*, Brian W. Kernighan et Dennis M. Ritchie, Prentice-Hall, 1978.

[MG] *Téléinformatique*, C. Macchi, J.-F. Guilbert et treize co-auteurs, Dunod, Paris, 1987.

[MHS1] *Interconnecting Electronic Mail Systems*, David D. Redell et James E. White, Computer, pp. 55-63, September 1983.

[MHS2] *Message-Handling Systems and Protocols*, Ian Cunningham, Proceedings of the IEEE, Vol. 71, No. 12, pp. 1425-1430, December 1983.

[MHS3] *Standards for Global Messaging: A Progress Report*, Theodore H. Myer, Journal of Telecommunication Networks, pp. 413-433, date inconnue.

[NFS] *Ultrix-32 Guide to Network File System*, Digital Equipment Corporation, 1987.

[PB] *Contribution à la mise en oeuvre d'un logiciel X400*, Philippe Brossel, Mémoire présenté aux Facultés Universitaires Notre-Dame de la Paix de Namur, département d'informatique, 1987.

[RT] *Réseaux et télématique*, Guy Pujolle, Danielle Dromard, Dominique Seret et Eric Horlait, Eyrolles, 1985.

[VF] *Trends in the Development of Public Telecommunication Networks*, Viktor Frantzen, Computer Networks and ISDN Systems, Vol. 14, pp. 339-358, 1987.

[X400] *Recommendation X.400 "Message handling systems : system model - service elements"*, Red Book, Volume VIII - Fascicle VIII.7, International Telegraph and Telephone Consultative Committee, October 1984.

[X401] *Recommendation X.401 "Message handling systems basic service elements and optional user facilities"*, Red Book, Volume VIII - Fascicle VIII.7, International Telegraph and Telephone Consultative Committee, October 1984.

[X408] *Recommendation X.408 "Message handling systems : encoded information type conversion rules"*, Red Book, Volume VIII - Fascicle VIII.7, International Telegraph and Telephone Consultative Committee, October 1984.

[X409] *Recommendation X.409 "Message handling systems : presentation transfer syntax and notation"*, Red Book, Volume VIII - Fascicle VIII.7, International Telegraph and Telephone Consultative Committee, October 1984.

[X410] *Recommendation X.410 "Message handling systems : remote operations and reliable transfer server"*, Red Book, Volume VIII - Fascicle VIII.7, International Telegraph and Telephone Consultative Committee, October 1984.

[X411] *Recommendation X.411 "Message handling systems : message transfer layer"*, Red Book, Volume VIII - Fascicle VIII.7, International Telegraph and Telephone Consultative Committee, October 1984.

[X420] *Recommendation X.420 "Message handling systems : interpersonal messaging user agent layer"*, Red Book, Volume VIII - Fascicle VIII.7, International Telegraph and Telephone Consultative Committee, October 1984.

[X430] *Recommendation X.430 "Message handling systems : access protocol for Teletex terminals"*, Red Book, Volume VIII - Fascicle VIII.7, International Telegraph and Telephone Consultative Committee, October 1984.

Annexe A : Répartition des tâches

Le tableau A.1 reprend la répartition des tâches accomplies par Xavier Rutten (XRU) et Joël Vanthournout (JVA).

La colonne "Etude préparatoire" désigne qui a étudié la matière concernée; la colonne "Rédaction" désigne qui a rédigé le chapitre correspondant; la colonne "Mise en oeuvre" désigne qui a effectué les réalisations concrètes sur ordinateur.

Chapitres	Etude préparatoire	Rédaction	Mise en oeuvre
Chapitres 1 à 4	XRU	XRU	
Chapitre 5 (sauf section 5.6)	XRU	XRU	XRU
Chapitre 5, section 5.6	JVA	JVA	XRU
Chapitre 6	JVA	JVA	
Chapitre 7 (sauf section 7.3)	XRU & JVA	JVA	
Chapitre 7, section 7.3	JVA	JVA	
Chapitre 8	JVA	JVA	

Tableau A.1 : répartition des tâches

Annexe B : Localisation dans les fichiers sources des fonctions C considérées

Cette annexe donne, pour chaque fonction C rencontrée dans ce travail, le fichier du répertoire **/usr/ean/src** où elle est définie.

Acc_report	ua\accept.c
AcceptMessages	ua\accept.c
Add_IA5	ua\p2part.c
Alias_expand	ua\alias.c
Av_start	mta\mngr\host\mmav.c
Bld_ORdesc	ua\p2or.c
Bld_ORname	ua\p2or.c
Bld_recip	ua\p2or.c
Bld_rlist	ua\p2or.c
ComposeCommand	ua\cmdcompo.c
Conv_recip	ua\ual.c
Db_msg	mta\dist.c
Export_msg	mta\txpact.c
FL_init	ua\field.c
Fo_open	ua\folder.c
Identify	ua\fomap.c
Import_msg	mta\txpact.c
InterpretCommands	ua\cmd.c
Ioclose	util\io\iofunct.c
IOCTLN	util\io.h
Iogetline	util\io\iotfr.c
Iotfr	util\io\iotfr.c
Lookup_mta	mta\route.c
Lookup_ua	mta\route.c
Me_free	ua\message.c
Mm_init	mta\mngr\mm.c
Mm_new	mta\mngr\mm.c
Mm_work	mta\mngr\mm.c
Mta_init	mta\mta.c
Mta_term	mta\mta.c
Mtamngr	mta\mngr\mtamngr.c
NewDraft	ua\draft.c
P1_close	mta\mtap1.c
P1_end	mta\mtap1.c
P1_export	mta\mtap1.c
P1_import	mta\mtap1.c
P1_open	mta\mtap1.c
P2_add_bp	ua\p2.c
P2_add_hdng	ua\p2.c
P2_find_heading	ua\p2.c
P2_hdngname	ua\p2bld.c
p2w_reports	ua\p2w.c
p3_changepw	mta\mtap3.c
p3_close	mta\mtap3.c

P3_deliver	mta\mtap3.c
P3_end	mta\mtap3.c
P3_notify	mta\mtap3.c
P3_open	mta\mtap3.c
P3_register	mta\mtap3.c
P3_s_env	ua\ual.c
P3_submit	mta\mtap3.c
Parse_hdng	ua\inmsg.c
Prmt_body	ua\inbody.c
Prmt_hdng	ua\inmsg.c
Prmt_list	ua\inmsg.c
Prmt_msg	ua\inmsg.c
ProfConfigure	ua\prof.c
Read_hdng	ua\inmsg.c
Read_msg	ua\inmsg.c
Rt_find	mta\route.c
Rt_init	mta\route.c
Rt_term	mta\route.c
Rts_open	rts\rtsint.c
SendCommand	ua\cmdsend.c
SendNow	ua\cmdsend.c
Service_request	mta\mngr\mtamngr.c
ss_open	rts\sess\ss.c
St_close	mta\store.c
St_create	mta\store.c
Sv_init	mta\mngr\host\mmsv.c
Sv_wait	mta\mngr\host\mmsv.c
Tran_open	rts\tran\traninit.c
Transmit	ua\ual.c
Txfr_contents	mta\txpact.c
Txp_actinit	mta\txpact.c
Txp_initiate	mta\txp.c
Txp_respond	mta\txp.c
Ua	ua\ua.c
UaCreate	ua\uacreate.c
UAL_send	ua\ual.c
Work_perform	mta\work.c
Write_env	mta\txpact.c
x9_len	ccitt\x9rw.c
x9_lhdr	ccitt\x9rw.c
x9_whdr	ccitt\x9rw.c
x9_write	ccitt\x9rw.c