MASTER EN SCIENCES INFORMATIQUES

Contribution à la mise en œuvre et aux tests d'un produit X.400

Description du Système et Méthodes de Test

Eloy, Marc; Reuviaux, Daniel

Award date:
1988

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix, Namur
Institut d'Informatique
Année académique 1987 - 1988

CONTRIBUTION A LA MISE EN OEUVRE ET AUX TESTS D'UN PRODUIT X.400 :

Description du Système et Méthodes de Test

Marc ELOY - Daniel REUVIAUX

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

Développé avant la parution des normes X.400 du CCITT, le Message Router permet à des systèmes VAX/VMS d'échanger du courrier électronique suivant la méthode "store and forward". Lorsqu'ont été publiées les normes X.400, Digital a produit le Message Router X.400 Gateway qui, comme son nom l'indique, réalise la passerelle entre le Message Router et le réseau ISO.

Tout d'abord les principes du modèle ISO et un résumé des recommandations X.400 du CCITT sont présentés.

Ensuite, l'environnement complet du Message Router, depuis la couche réseau jusqu'à la couche application est détaillé.

Une étude des problèmes d'adressage et de routage dans un réseau X.400 a ensuite été réalisée. Ces problèmes ont également été étudiés dans le cadre de l'implémentation faite par Digital.

Un Agent Utilisateur simple a été développé pour interagir avec le Message Router.

La dernière partie de ce document présente le problème de la vérification de protocoles : théorie et typologie des tests, tests de conformité, organismes et sites de test, développement d'un logiciel de test.

Ce travail rapporte une expérience dans le domaine de l'étude d'un environnement logiciel X.400 et de la réalisation des tests d'un tel environnement.

Developed before the occurence of the X.400 recommendations of the CCITT, the Message Router allows VAX/VMS systems to exchange electronic mail using the store and forward method. When the X.400 recommendations were published, Digital has produced the Message Router X.400 Gateway, which acts as a gateway between the Message Router and the OSI network.

First, the principles of the OSI model are presented with a summary of the X.400 recommendations of the CCITT.

Then, the entire Message Router environment is detailled, from the network layer up to the application layer.

A study of addressing and routing problems in an X.400 network has been carried out. Various aspects of these problems in Digital's implementation have also been studied.

A simple User Agent has been developped to provide an interaction means with the Message Router.

The last part of this document presents the protocol checking problems : tests theory and typology, conformance testing, test organisms, development of a testing software.

This work is the report of an experience in the study of an X.400 software environment and in the testing of this environment.

Nous remercions Monsieur Philippe VAN BASTELAER, Professeur aux Facultés Notre Dame de la Paix à Namur, d'avoir assuré la direction de ce mémoire.

Pour nous avoir accueilli durant notre stage à l'Inter-university Institute for High Energies, que les Professeurs Jacques SACTON et Jacques LEMONNE, trouvent ici l'expression de notre reconnaissance.

Que Monsieur Paul VAN BINST, Professeur à l'Université Libre de Bruxelles soit ici remercié de l'attention qu'il a porté à l'élaboration de ce travail.

Nous tenons à remercier également Madame Rosette VANDENBROUCKE et Monsieur Willem VAN DROOGENBROECK, pour l'aide qu'ils nous ont apportée.

Pour l'autorisation accordée d'installer gracieusement à titre provisoire un site X.400 aux Facultés Notre Dame de la Paix à Namur, que la société DIGITAL soit remerciée.

Que Monsieur François BODART, Professeur aux Facultés Notre Dame de la Paix à Namur, Directeur du Centre de Calcul de ces mêmes Facultés soit assuré de notre gratitute pour nous avoir permis d'utiliser le Centre de Calcul pour réaliser ce travail.

Nous tenons également à remercier :

Messieurs Patrick GEURTS, Manu LORENT et Serge SIMONET, assistants aux FNDP, pour l'intérêt porté à ce travail,
Messieurs Christophe HANON et Bernard SALES, assistants à l'ULB, pour les conseils qu'ils nous ont apportés,
Le personnel de l'IIHE pour son accueil.
L'ensemble des opérateurs du Centre de Calcul des FNDP pour l'aide précieuse qu'ils nous ont offerte.

0. INTRODUCTION

Avant-propos
Plan
Liste des Abbréviations

1. CHAPITRE I : LE MODELE ISO ET LA MESSAGERIE ELECTRONIQUE X.400

1.1 LE MODELE ISO

1.2 LA MESSAGERIE ELECTRONIQUE X.400

2. CHAPITRE II : ENVIRONNEMENT ET OBJET DE L'ETUDE

3. CHAPITRE III : ADRESSAGE ET ROUTAGE

5. CHAPITRE V : VERIFICATION DE PROTOCOLES

0. INTRODUCTION

Depuis le début des années 1970, le courrier électronique est devenu le moyen privilégié d'échanger des informations entre personnes au sein d'organisations ou d'institutions scientifiques.

Les systèmes de transport de l'information sont de plus en plus étendus géographiquement et font intervenir des équipements de fournisseurs différents possédant leurs caractéristiques propres et offrant chacun un service différent. Des instances de normalisation, tel le Comité Consultatif International des Télégraphes et des Téléphones (CCITT) s'efforcent de définir des protocoles standards permettant l'interconnexion des différents services existants afin d'offrir aux utilisateurs un accès à un réseau de messagerie de niveau mondial.

Notre objectif est d'abord de présenter l'implémentation de ces recommandations réalisée par Digital Equipment. Ensuite nous nous proposons de vérifier si cette implémentation est effectivement conforme par rapport aux normes du CCITT par le développement d'outils de test de logiciels de messagerie électronique.

L'implémentation étudiée a été développée en deux phases. Le Message Router est un agent de transfert de messages permettant l'échange de messages entre systèmes Digital. Il est basé sur les recommandations NBS. Ensuite, pour rendre le Message Router conforme aux normes X.400 du CCITT, le Message Router X.400 Gateway a été développé pour permettre l'échange de messages entre un système Digital et n'importe quel autre système X.400.

Ce document s'adresse aux personnes désireuses de découvrir une implémentation des recommandations X.400 du CCITT, de développer un agent utilisateur ou encore de réaliser des tests de conformité et d'interopérabilité entre logiciels X.400.

La première partie de cet ouvrage présente un bref rappel du modèle de référence ISO et des recommandations de la série X.400. Est également présentée la terminologie utilisée tout au long de cette étude.

Le deuxième chapitre a pour but de présenter dans les détails un ensemble de produits logiciels implémentant un agent de transfert de messages. Nous y présentons d'abord les logiciels des couches réseau, transport et session fournissant les services nécessaires à l'établissement d'une connexion d'échange de messages. Ensuite nous montrons comment un agent de transfert de messages, conçu avant la parution des recommandations de la série X.400, a été "complété" pour pouvoir implémenter ces recommandations. Nous cloturons ce chapitre en présentant les problèmes que nous avons rencontrés lors de l'installation et de l'utilisation de ces produits.

Le chapitre trois aborde les problèmes d'adressage et de routage sur le réseau OSI. Nous y expliquons comment s'effectue la prise en charge d'une requête de connexion X.400 par le système demandeur et par le système sollicité. La manière dont l'implémentation étudiée effectue le routage des messages est également étudiée.

Le chapitre quatre aborde le problème de la conception d'un interface usager approprié au système de messagerie utilisé. L'interface développé fournit les services de soumission et de réception de messages conformes

aux recommandations X.400.

Le dernier chapitre étudie en détails les tests de conformité et d'interopérabilité d'un logiciel X.400. Après avoir rappelé les grandes lignes de la théorie des tests et de leur typologie, nous nous intéressons aux problèmes des tests de conformité, de leur méthodologie et des particularités des tests de protocoles X.400. Nous présentons brièvement les organismes de tests et leur rôle dans les tests de conformité. La dernière partie de ce chapitre détaille, après une analyse des outils existant, le logiciel de test MRXTESTER qui a été implémenté par les auteurs dans le but de réaliser les tests du logiciel étudié. Quelques résultats de tests sont encore présentés.

Nous cloturons ce document en indiquant les références bibliographiques que nous avons utilisées. Nous fournissons entre-autres en annexe la trace du premier message X.400 belge et notre implémentation d'interface utilisateur ainsi que du logiciel de test.

Nous avons utilisé les références suivantes tout au long de la réalisation de cet ouvrage : <MACC-87>, <PUJO-85> et <TANE-81>.

LISTE DES ABREVIATIONS

ADMD	Administration Management Domain
CCITT	Comité Consultatif International Télégraphique et Téléphonique
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
DECnet	Réseau de communication utilisé par les sites Digital
DTE	Data Terminal Equipment
IEEE	The Institute of Electrical and Electronic Engineers, INC.
IEEE 802.3	Spécification de réseau local de type CSMA/CD (Ethernet)
INTERNET	Protocole inter-réseau (Department of Defense (DoD)
IUT	Implementation Under Test (Implémentation à tester)
MHS	Message Handling System
MPDU	Message Protocol Data Unit
MR	Message Router (Digital)
MRX	Message Router X.400 gateway (Digital)
MT	Message Transfer
MTA	Message Transfer Agent
MTS	Message Transfer System
NBS	National Bureau of Standards (US)
OSI	Organisme de Standardisation Internationale
OSAK	OSi Applications Kernel (Digital)
P1	Protocole utilisé par le MTS
P2	Protocole utilisé par l'IPMS
P3	Protocole utilisé entre un MTA et un SDE
PDU	Protocol Data Unit
PICS	Protocol Implementation Conformance Statement
PIXIT	Protocol Implementation eXtra Information for Testing
PRMD	Private Management Domain
PSI	Packetnet Switching Interface (Digital)
RTS	Reliable Transfer Server
SAP	Service Access Point
SDE	Submit and Delivery Equipment
UA	User Agent
VOTS	Vax Osi Transport Service (Digital)
X.25	Norme OSI pour les réseaux à commutation par paquets.

LISTE DES FIGURES

1. CHAPITRE I : LE MODELE ISO ET LA MESSAGERIE X.400

Dans ce premier chapitre, nous allons présenter, d'une part, le modèle de référence ISO et, d'autre part, les spécifications du système de messagerie électronique X.400.

1.1 LE MODELE ISO

Qu'il nous soit permis, dans cette première partie, de rappeler les caractéristiques du modèle défini par l'Organisation de Standardisation Internationale (OSI), pour l'Interconnexion de Systèmes Ouverts (ISO). Ce modèle ayant été analysé lors de mémoires précédents et notamment <ALEX-87> et <AN-DE-FR-87>, nous nous en tiendrons donc aux aspects principaux. Pour obtenir plus de détails sur le sujet, nous conseillons de consulter les références suivantes : <X.200>, <X.225>, <X.224>, <IEEE-83> et <MACH-87>.

1.1.1 INTRODUCTION : Pourquoi ISO ?

Depuis le début des années 1970, on a assisté au développement et à l'utilisation des réseaux comme un moyen privilégié pour échanger de l'information entre ordinateurs.
Chaque constructeur offrant des services télématiques s'appuyant sur des systèmes d'exploitation, des architectures de réseau et des protocoles de transfert de données différents et souvent incompatibles, les instances de normalisation ont dû, pour permettre la création d'un réseau télématique le plus ouvert possible, se pencher sur la définition de protocoles standards permettant l'interconnexion des différents services existants.
Dès 1980, l'Organisation de Standardisation Internationale (OSI) a défini un ensemble de normes pour l'Interconnexion de Systèmes Ouverts (ISO).

1.1.2 PRINCIPES

1.1.2.1 Couche

La figure (I.1) montre que l'architecture ISO est une structure hiérarchique en couches. Chaque couche est un ensemble indépendant de fonctions implémentant un protocole particulier.
Une couche (N) offre ses services à la couche qui lui est directement supérieure (N+1) et uniquement à celle-la.
Une couche (N) utilise les services de sa couche inférieure (N-1) et par là même ceux de ses inférieures (1 à N-2) et offre une valeur ajoutée à ces services.

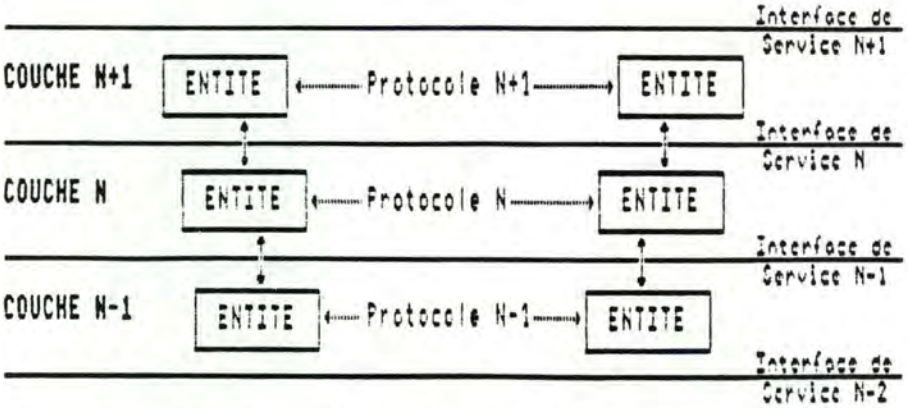


FIG. I.1 : Structure en couche

1.1.2.2 Unité de donnée

Une unité de donnée (Data Unit) contient les informations qui sont échangées entre deux entités paires ou adjacentes.

1.1.2.3 Service

Un service proposé par une couche est utilisé par une entité d'une couche de niveau supérieur par l'intermédiaire de primitives de service qui jouent le rôle d'interface entre couches. Ces primitives s'échangent des messages par les SDU (Service Data Units).
Les principaux services offerts par une couche de niveau N permettent l'ouverture d'une connexion de niveau N au bénéfice de deux entités de niveau N+1, la gestion de cette connexion pour le transfert des données, la gestion du protocole employé et la fermeture de la connexion.

1.1.2.4 Protocole

Un protocole est un ensemble de règles et de conventions à respecter pour permettre la communication entre entités de même niveau (entité paires) sur des systèmes différents. Les messages échangés par ces entités sont les PDU (Protocol Data Units).

1.1.2.5 Architecture en couches

La figure (I.2) présente l'architecture en couche du modèle ISO.

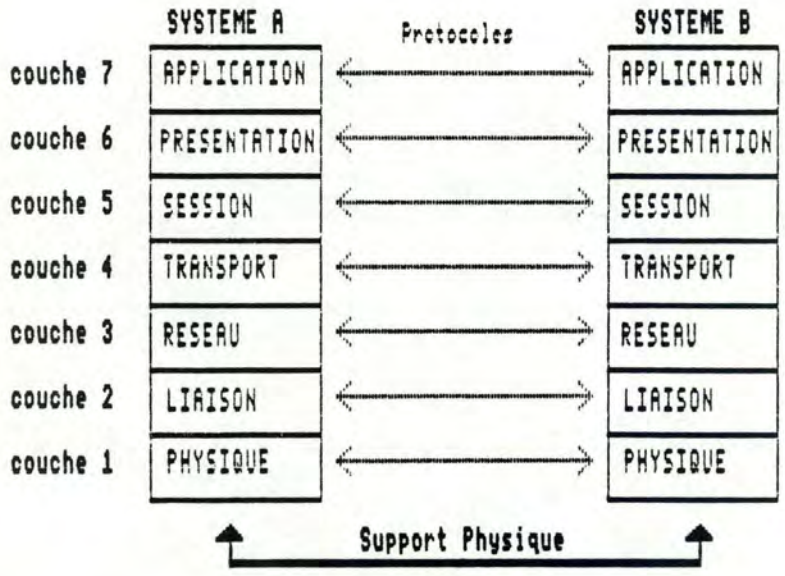


FIG. I.2 : Architecture en couche du modèle ISO

1.1.3 COUCHES

Rappelons brièvement les caractéristiques des différentes couches.

1.1.3.1 COUCHE 1 : PHYSIQUE (PHYSICAL)

La couche PHYSIQUE (1) génére et interpréte les signaux codés pour la transmission physique (électrique, acoustique, optique, ...) de données via une voie de communication. Elle permet l'activation et la désactivation des connexions physiques entre entités.

1.1.3.2 COUCHE 2 : LIAISON (DATA LINK)

La couche LIAISON (2) offre les fonctions et les procédures pour communiquer des données entre des entités. Elle détecte et éventuellement, corrige les erreurs provenant de la couche PHYSIQUE. Les protocoles de couche 2 les plus courants sont HDLC (pour les connexions point à point et multipoint) et IEEE.802 (pour les réseaux locaux).

1.1.3.3 COUCHE 3 : RESEAU (NETWORK)

La couche RESEAU (3) offre aux couches supérieures l'indépendance vis-à-vis de la technologie utilisée pour le transfert de données et des considérations d'adresse et de routage. Elle cache à la couche TRANSPORT toutes les particularités du moyen utilisé pour l'échange (fibres optiques, commutation par paquets, satellite ou réseau local).
C'est le niveau RESEAU qui se charge des adresses et du routage.

1.1.3.4 COUCHE 4 : TRANSPORT

La couche TRANSPORT (4) est la première couche qui offre un protocole de bout en bout. Son but est de permettre un transfert transparent des données entre systèmes, déchargeant ainsi les couches supérieures de toute notion de communication à distance, d'efficacité, d'optimisation ou de fiabilité.
Les utilisateurs de la couche TRANSPORT peuvent choisir, lors de l'établissement d'une connexion, entre différentes classes de protocoles selon la qualité ou le degré de sécurité voulu.
Les classes suivantes sont disponibles :

- classe 0 : fournit une connexion transport de base,
- classe 1 : fournit un recouvrement des erreurs signalées par la couche réseau,
- classe 2 : fournit un multiplexage des connexions transport sur une connexion réseau et un contrôle de flux optionnel,
- classe 3 : fournit un multiplexage, un recouvrement des erreurs signalées et un contrôle de flux,
- classe 4 : fournit un multiplexage, un recouvrement des erreurs non signalées et un contrôle de flux.

1.1.3.5 COUCHE 5 : SESSION

La couche SESSION (5) a pour but d'offrir des mécanismes pour organiser et structurer les interactions et pour gérer le dialogue entre processus. Elle permet les opérations bi-directionnelles simultanées (TWS - Two Way Simultaneous) et bi-directionnelle à l'alternat (TWA - Two Way Alternate), la prise de points de synchronisation (majeur et mineur), la définition de jetons (Token) pour structurer et synchroniser les échanges.
La couche session fournit un ensemble de primitives pour :
- ouvrir et fermer une connexion,
- signaler les exceptions survenants à la connexion,
- procéder à l'échange des jetons,
- transférer de l'information,
- interrompre et reprendre une activité,
- poser les points de synchronisation,
- relancer l'échange aux points de synchronisation.

1.1.3.6 COUCHE 6 : PRESENTATION

La couche PRESENTATION (6) assure l'indépendance des processus vis-à-vis de la représentation des données (syntaxe). Elle permet aux utilisateurs de se définir un "contexte de présentation" spécifique à une application, à un type d'appareil ou à un type de standard de représentation. Elle offre des primitives permettant aux entités de négocier le choix d'un langage commun à adopter pendant la communication et, éventuellement, les moyens de traduction nécessaires.
Elle permet de diviser une connexion session en plusieurs activités divisées à leur tour en plusieurs unités de dialogue.

1.1.3.7 COUCHE 7 : APPLICATION

La couche APPLICATION (7) définit la sémantique de l'échange entre processus d'application. Ses fonctionnalités sont donc propres à chaque type de problème. Suivant leurs besoins, les utilisateurs employent les réseaux pour des applications telles que messagerie électronique, transfert de fichiers, travail à distance, terminal virtuel, ... ainsi que pour de la gestion des applications et du système ISO complet.

1.2 LA MESSAGERIE ELECTRONIQUE X.400

Les principales caractéristiques du système de messagerie sont définies par les recommandations <X.400>, <X.401>, <X.408>, <X.409>, <X.410>, <X.411>, <X.420>, <X.430> et <X.400/88>.

Pour obtenir des renseignements complémentaires sur le sujet, nous conseillons les références suivantes : <IEEE-83>, <HANO-86>, <ALEX-87>, <LABO-87> et <AN-DE-FR-87>.

1.2.1 INTRODUCTION

Depuis le début des années 1970, le courrier électronique est considéré comme un moyen privilégié pour échanger de l'information entre individus via un équipement de télécommunication approprié.
Dans le but de créer un réseau mondial de messagerie électronique, le CCITT a élaboré, dans le cadre des travaux sur l'ISO, une série de recommandations, parues dès 1984, sous le nom de série X.400.
Une nouvelle version des normes, dénommée X.400/88, actuellement en chantier, doit sortir avant 1990. Etant donné le succès remporté par l'ancienne version et l'expérience acquise dans l'implémentation et l'utilisation d'un tel type de messagerie, X.400/88 contiendra de nombreux ajouts et modifications aux normes X.400/84. Il est déjà certain que X.400/88 sera un sur-ensemble de X.400/84 afin que les implémentations existantes restent compatibles avec les nouvelles normes.

La version X.400/88 étant encore instable, c'est la version X.400/84 qui a servi de base à nos travaux.

1.2.2 MESSAGE HANDLING SYSTEM (MHS)

1.2.2.1 Introduction

Nous allons maintenant présenter le modèle fonctionnel du système de messagerie (MHS Message Handling System) et définir les entités qui le composent.

1.2.2.2 Description du modèle fonctionnel de messagerie

Le modèle (Figure I.3) définit les relations de base entre les différentes entités du système de messagerie. Le niveau inférieur est formé par le MTS (Message transfer System) constitué des MTA (Message Transfer Agents) assurant la prise en charge de messages de l'expéditeur, la transmission de ceux-ci, et la livraison au destinataire. Le niveau intermédiaire est le MHS (Message Handling System) contenant, en plus du MTS, les UA (User Agents) qui aident l'utilisateur à construire et interpréter les messages et à utiliser le MTS. Le niveau supérieur est le MHE (Message Handling Environment) qui est formé du MHS et de ses utilisateurs (USER).

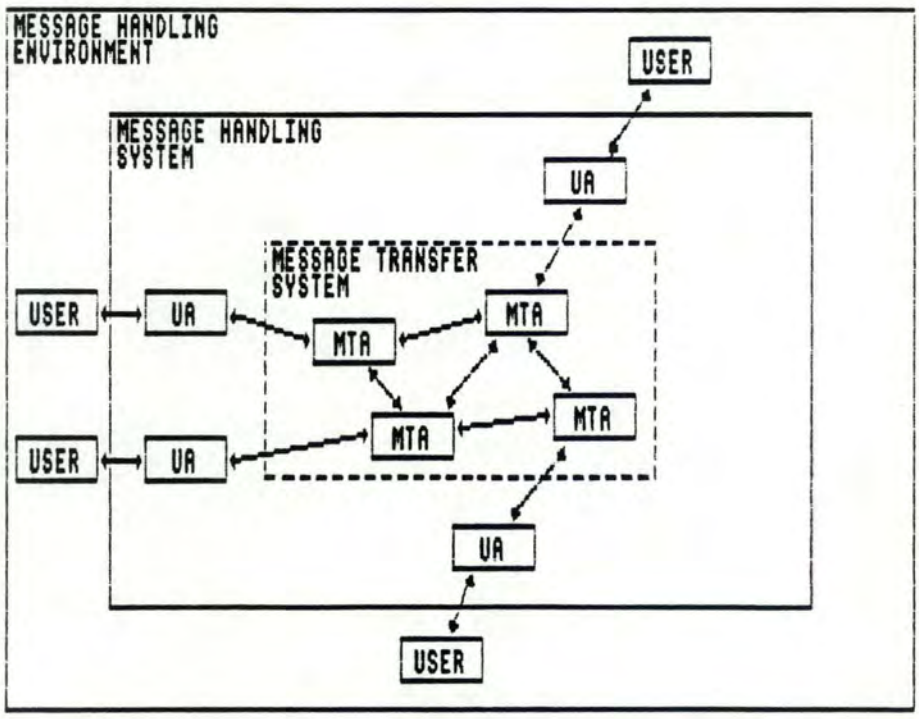


Fig. I.3 : Modèle fonctionnel de la messagerie X.400

1.2.2.2.1 Utilisateur (USER)

Un utilisateur peut être humain ou logiciel (sous la forme d'un processus d'application).

1.2.2.2.2 User Agent (UA)

Un User Agent (Agent Utilisateur) est un ensemble logiciel qui permet à l'utilisateur auquel il est attaché de composer, envoyer, stocker et recevoir des messages.
Une analyse plus fine du USER AGENT est faite dans le chapitre 4 "ANALYSE ET IMPLEMENTATION D'UN USER AGENT".

1.2.2.2.3 Message Transfer Agent (MTA)

Le Message transfer Agent (MTA) est responsable de la prise en charge de messages de l'expéditeur, de la transmission de ceux-ci et de la livraison au destinataire.

1.2.2.2.4 Message

La figure I.4 présente la structure d'un message.
Un MESSAGE transféré par le MTS est composé d'une ENVELOPPE et d'un CONTENU. L'ENVELOPPE contient les adresses de l'expéditeur et des destinataires ainsi que toutes les informations nécessaires aux services offerts par le MTS. Le CONTENU renferme les informations à transférer. Il est ignoré du MTS qui se base uniquement sur l'enveloppe pour faire le routage du message. Le contenu est composé d'un HEADER ou entête et d'un certain nombre de BODYPARTS ou parties de corps.

Une description en syntaxe Bachus-Naur d'un message est jointe en annexe.

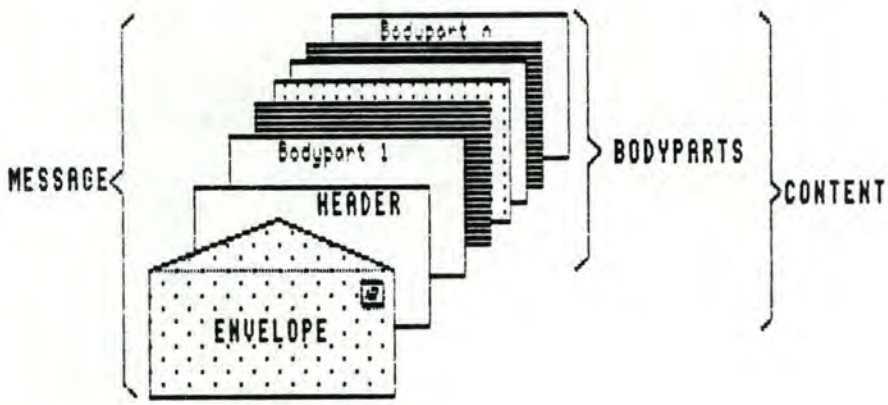


Fig. I.4 : Structure d'un message

1.2.3 INTERPERSONAL MESSAGING SYSTEM (IPMS)

Le système de messagerie interpersonnelle ou IPMS offre à un utilisateur humain des services de communication avec d'autres utilisateurs humains. Le IPMS comprend le MTS, une classe particulière d'UA ainsi que le Telex et les services télématiques définis par le CCITT.
Il étend les fonctionnalités des UA en offrant des services de composition, de soumission, de délivraison et de présentation de message.

1.2.4 NOMS ET ADRESSES

1.2.4.1 Concepts et termes

Lorsque l'on désire envoyer un message, on doit pouvoir spécifier son destinataire de manière unique. Comme un utilisateur est externe au MHS, il importe de pouvoir l'identifier par son User Agent. Chaque UA est identifié par un nom appelé "O/R name".

L'O/R name ("Originator/Recipient name") est constitué d'une liste d'attributs déterminés par des caractéristiques de l'utilisateur de l'UA référencé (pays, domaine, service, nom, ...).

L'O/R adresse est le nom descriptif du UA. Il possède des caractéristiques nécessaires à la localisation sur le réseau ISO du MTA servant l'UA du récepteur.

On peut dire que chaque O/R adresse est un O/R name mais que chaque O/R name n'est pas une O/R adresse.

1.2.4.2 Attributs des Originator/Recipient names

Les principaux Attributs présents dans un O/R name peuvent être standards ou propres à un domaine donné.
Un domaine ou Management Domain (MD) est un ensemble formé d'au-moins un MTA et de zéro ou plus UA. Le MD peut être géré par une administration ou par un organisme privé. Il sera respectivement appelé 'Administration Management Domain' (ADMD) ou 'Private Management Domain' (PRMD).

Liste des Attributs standards :

Country_name	: nom du pays
Administration_domain_name	: nom de l'administration responsable
X121_address	: adresse X121
Telematic_Terminal_Id	: identificateur du terminal
Private_domain_name	: nom du domaine privé
Unique_UA_Identifier	: identificateur de l'UA
Personal_Name	
SurName	: prénom
GivenName	: nom
Initials	: initiales
Generation Qualifier	: génération
Organization_name	: nom de l'organisation
Organizational_unit_name	: nom de l'unité dans l'organisation

Exemple d'attributs propres à un domaine donné (Domain defined) :

Numéro de téléphone de l'utilisateur
Adresse postale d'un utilisateur
Nom du programme qui traitera le message
...

Selon le format d' O/R name choisi, un attribut peut être obligatoire ou optionnel, répétitif ou unique, décomposable ou non.

1.2.4.3 Format des O/R names

Dans la norme X.400, il y a actuellement 4 formats d'O/R names possibles. la notation suivante a été choisie: "{ }" signale un élément répétitif
"[ ]" signale un élément optionnel

Format 1.1 ou 'Mnemonic Format'

```
Country_name
Administration_domain_name
[ Personal_name ]
[ Private_domain_name ]
[ Organization_name ]
[ { Organizational_unit_name } ]
[ { Domain_defined_attributes } ]
Remarque : On doit spécifier au moins l'un des
           attributs optionnels.
```

Format 1.2 ou 'All Numeric Format'

```
Country_name
Administration_domain_name
Unique_UA_identifier
[ { Domain_defined_attributes } ]
```

Format 1.3 ou 'Telex Format'

```
Country_name
Administration_domain_name
X121_address
[ { Domain_defined_attributes } ]
```

Format 2 ou 'Telematic Format'

```
X121_address
[ Telematic_Terminal_ID ]
```

1.2.4.4 Routage

Une route est l'information qui décrit le chemin que doit prendre un message à travers le MTS, du UA origine vers l'UA récipient, pour arriver à sa destination finale. C'est le MTS qui détermine, à partir de l'O/R Address, en faisant transiter le message de MTA en MTA, la route que celui-ci doit suivre.
Les problèmes de routage et d'adressage sont analysés en détail dans le chapitre 3 "ADRESSAGE ET ROUTAGE".

1.2.4.5 Fonction de Répertoire (Directory)

Un répertoire ou Directory est un ensemble de fonctions d'aide qui ont pour but d'assister un utilisateur et son UA en leur fournissant des informations nécessaires la spécification d'un O/R name.

Une description plus complète de cette fonction peut être trouvée dans le chapitre 4 "ANALYSE ET IMPLEMENTATION D'UN USER AGENT".

1.2.5 ELEMENTS DE SERVICE

Il nous parait utile, pour la compréhension des chapitres suivants, de rappeler et d'analyser brièvement les services (et les éléments de données associés) offerts, d'une part, par le MTS et, d'autre part, par l'IPMS.

1.2.5.1 Service de transfert de message (MTS)

Le service de transfert de message (MTS) offre aux UA, par l'intermédaire des MTA, des services de base, de soumission, de livraison, de conversion, de sonde, d'état et d'information.

1.2.5.1.1 Services de base

Les services de base contituent un noyau minimum indispensable au fonctionnement correct d'une messagerie X.400.

En plus de l'émission et de la réception d'un message sont fournis les services suivants.

Access Management (Contrôle d'accès)
Cet élément permet, lorsqu'un UA désire utiliser les services fournis par son MTA, l'identification et la vérification par mot de passe des identités respectives.

Content Type Identification (Identification de type du contenu)
A la soumission d'un message, l'UA d'origine doit spécifier la convention (c'est-à-dire le nom du protocole) qui est suivie pour la formation du contenu du message. Pour l'instant, la seule valeur définie est 2 (protocole P2). Cet élément permet donc d'envisager des extensions futures à la norme X.400 en signalant que le contenu est formé selon un autre protocole que P2.

Converted Indication (Indication de conversion)
Cet élément permet d'indiquer à un UA destinataire que le MTS a effectué un certain nombre de conversions (implicites ou explicites) sur le(s) type(s) des informations contenues dans le message.

Non-Delivery Notification (Notification de non livraison)
Ce service permet au MTS d'indiquer à l'UA expéditeur qu'un des messages qu'il a envoyé n'a pas été délivré au(x) UA(s) destinataire(s). La raison de l'échec est également signalée.

Original Encoded Information Types Indication (Indication des types originaux d'information encodée)
Cet élément permet à l'UA expéditeur de spécifier au MTS et à l'UA destinataire les types de codage d'un message.

Registered Encoded Information Types (Types d'information encodée autorisés)
Cet élément permet au UA de spécifier à son MTA les types de codage de message qu'il accepte. Ces informations sont retenues par celui-ci qui doit alors refuser la livraison de messages qui ne correspondent pas aux caractéristiques données par l'UA.

Submission Time Stamp Indication (Indication du moment de soumission)
Cet élément signale à l'UA récepteur à quel instant (date et heure) le message a été pris en charge par le MTA de l'UA d'origine.

Delivery Time Stamp Indication (Indication du moment de livraison)
Cet élément permet au MTS d'indiquer la date et l'heure à laquelle un message a été délivré.

Message Identification (Identificateur du message)
Cet élément permet au MTS de donner un identificateur unique à chaque message soumis par un UA ou délivré par le MTS. Ce numéro sert, entre-autre, à relier une notification au message auquel elle se rapporte.

1.2.5.1.2 Eléments de soumission et livraison

Les services de soumission et de livraison sont les suivants :

Alternate Repipient Allowed (Autorisation de livraison à un destinataire alternatif)
Cet élément permet au UA d'origine de spécifier que le message peut être délivré à un autre récipient si celui-ci est défini par le UA destinataire.

Deferred Delivery (Livraison différée)
Cet élément permet au UA d'origine de demander à son MTA que le message ne soit pas délivré avant un moment déterminé (date et heure).

Deferred Delivery Cancellation (Annulation de livraison différée)
Cet élément permet au UA d'origine d'obliger le MTS à supprimer la demande de livraison différée liée à un message déjà soumis.

Delivery Notification (Notification de livraison)
Cet élément permet au UA d'origine d'exiger que l'arrivée du message au MTA de l'UA récepteur soit confirmée par l'envoi d'une notification de livraison.

Disclosure of Others Recipients (Divulgation des autres destinataires)
Cet élément signale que chaque récepteur du message peut recevoir la liste de tous les récepteurs d'un message multi-destination.

Grade of Delivery Selection (Choix du degré de livraison)
Cet élément permet au UA d'origine de spécifier le degré de priorité d'un message (urgent, normal ou non urgent). Cela affecte le traitement à effectuer pour la livraison du message.

Multi-Destination Delivery (Livraison multi-destinataires)
Cet élément permet au UA d'origine de dire que le message doit être délivré à plusieurs UA destinataires.

Prevention of Non-Delivery Notification (Interdiction de notification de non livraison)
Cet élément permet au UA d'origine de forcer le MTS à ne pas renvoyer une éventuelle notification de non livraison.

Return of Contents (Retour du contenu)
Cet élément permet au UA d'origine de demander que le contenu du message original soit renvoyé avec la notification en cas de non livraison.

1.2.5.1.3 Eléments de conversion

Les services de conversion sont les suivants :

Conversion Prohibition (Interdiction de conversion)
Cet élément permet au UA d'origine d'empécher le MTS d'effectuer des conversions sur le(s) type(s) d'information du message.

Explicit Conversion (Conversion explicite)
Cet élément permet au UA d'origine de demander au MTS d'effectuer une conversion d'un type précis.

Implicit Conversion (Conversion implicite)
Par cet élément, l'UA d'origine permet au MTS d'effectuer toutes les conversions qu'il trouvera nécessaires sur le contenu d'un message.

1.2.5.1.4 Eléments de sonde (Probe)

Le service de sonde consiste à envoyer un message de test (probe) vers chaque destinataire spécifié. Ce message cause l'envoi d'un certain nombre de notifications de livraison ou de non livraison à l'UA demandeur du test. Cela permet au UA origine d'établir, avant d'envoyer un message, si celui-ci arrivera à sa destination.

1.2.5.1.5 Eléments d'état et d'information

Les éléments d'état et d'information sont les suivants :

Alternate Recipient Assignment (Désignation d'un destinataire alternatif)
Ce service permet à un UA de recevoir des messages dont le nom (O/R name) de destination ne correspond pas exactement au sien. Le but de ce service (réservé aux gestionnaires du domaine auquel appartient l'UA) est de permettre le traitement de messages dont la destination est incorrectement spécifiée.

Hold for Delivery (Stockage avant livraison)
Cet élément permet à un UA de dire à son MTA de conserver les messages et les notifications éventuelles qui lui sont destinés jusqu'à un moment donné.

1.2.5.2 Service de messagerie interpersonnelle (IPMS)

Nous allons décrire brièvement les éléments du service de messagerie interpersonnelle ou IPMS. Ce service, basé sur le protocole P2, permet l'échange de messages à l'usage des utilisateurs humains.

1.2.5.2.1 Eléments de base

En plus des éléments de base du MTS, l'IPM offre les services suivants.

IP Message Identification (Identificateur du Message interpersonnel)
Cette identification est indépendante de l'identification donnée par le MTS. A la différence de l'identification du message donné par le MTS, variant à chaque soumission, tranfert ou livraison, l'IP-MESSAGE-ID reste statiquement liée à un IP-message aussi longtemps que celui-ci existe. Cela permet les références croisées, les indications d'obsolescence, ...

Typed Body (Corps typé)
Cet élément signale le type de codage particulier d'information (IA5text, TLX, Voice, ...) lié à une partie du corps d'un message donné.

1.2.5.2.2 Eléments de soumission, livraison et conversion

Les éléments de soumission, de livraison et de conversion du service d'IPM sont identiques à ceux décrits pour le MTS aux paragraphes 1.2.5.1.2 et 1.2.5.1.3.

1.2.5.2.3 Eléments de coopération (action)

Les services nécessitant la coopération entre UA sont les suivants :

Blind Copy Recipient Indication (Indic. de récipient en copie aveugle)
Cet élément spécifie des récepteurs supplémentaires qui ne seront pas connus des autres récepteurs (primaires ou secondaires). Il signale que le message doit être traité comme une copie ignorée par les autres destinataires (Copie Aveugle).

Non-Receipt Notification (Notification de non-réception)
Par cet élément, l'UA d'origine demande à être averti en cas de non réception du message (pour cause d' auto-forward, de fin d'abonnement du destinataire ou parce que le message a été refusé par l'UA récepteur).

Receipt Notification (Notification de réception)
Cet élément demande à l'utilisateur récepteur l'envoi (manuel et non obligatoire) d'une confirmation de réception. Bien entendu, l'UA du recepteur peut se charger de cela automatiquement.

Auto-Forwarded Indication (Indication de suivi automatique)
Cet élément permet à un utilisateur de savoir si le message qu'il recoit, contient un message que l'on a fait suivre automatiquement (par un mécanisme de suivi de courrier) et qu'il était originellement destiné à un autre utilisateur.

1.2.5.2.4 Eléments de coopération (information)

Les services nécessitant une coopération par transmission d'informations entre UA sont les suivants :

Originator Indication (Indication d'expéditeur)
Cet élément identifie l'UA de l'utilisateur qui a envoyé le message.

Authorizing Users Indication (Indication des utilisateurs autorisants)
Cet élément signale les utilisateurs qui ont autorisé l'envoi du message.

Primary and Copy Recipients Indication (Indication de destinataire primaire ou copie)
Cet élément identifie les utilisateurs qui doivent recevoir le message. Un récipient peut être destinataire principal (Primary) ou de copie (Copy).

Expiry Date Indication (Indication de date d'expiration)
Cet élément indique le moment (date et heure) après lequel l'expéditeur du message ne considère plus celui-ci comme valide ou utile.

Cross-Referencing Indication (Indication de référence)
Cet élément indique les messages qui sont référencés par celui-ci.

Importance Indication (Indication d'importance)
Cet élément permet à l'expéditeur de spécifier l'importance qu'il accorde à un message. Celle-ci peut être 'faible', 'normale' ou 'haute'. Cette indication n'influence en rien sur le comportement du système.

Obsoleting Indication (Indication d'obsolescence)
Cet élément donne les identificateurs des messages qui ne sont plus valables suite à la réception de ce message.

Sensitivity Indication (Indication de sensivité)
Cet élément permet à l'expéditeur de spécifier le degré de sécurité lié à un message. Celui-ci peut être 'Personnel', 'Privé' ou 'Confidentiel à la compagnie'.

Subject Indication (Indication de sujet)
Cet élément contient des informations spécifiées par l'expéditeur comme étant le sujet du message.

Replying IP-Message Indication (Indication de réponse)
Cet élément signale que le message est envoyé en réponse d'un autre message.

Reply Request Indication (Indication de demande de réponse)
Cet élément indique au destinataire du message que l'expéditeur désire une réponse.

Forwarded IP-Message Indication (Indication de suivi)
Cet élément indique si ce message contient un autre message que l'on a fait suivre (non automatiquement).

Body Part Encryption Indication (Indication de criptage)
Cet élément signale qu'une partie du contenu du message a été encryptée.

Multi-Part Body (Corps multi-partie)
Un message IP peut posséder un corps subdivisé en différentes parties (Body parts) distinctes de type différents.

1.2.5.2.5 Eléments de sonde

Les éléments de sonde du service d'IPM sont identiques à ceux décrits pour le MTS au paragraphe 1.2.5.1.4.

1.2.5.2.6 Eléments d'état et d'information

Les éléments d'état et d'information du service d'IPM sont identiques à ceux décrits pour le MTS au paragraphe 1.2.5.1.5.

1.2.6 DESCRIPTION EN COUCHE DU MHS

1.2.6.1 Principes et concepts

Les entités et les protocoles de messagerie sont situés dans la couche application du modèle de référence ISO.
Le MHS peut, lui-même, être divisé en deux couches :

1. la couche User Agent (UAL) qui contient les fonctionnalités de l'UA associées au contenu des messages.
2. la couche de transfert (MTL) qui contient les fonctionnalités du MTA et qui offre le service de transfert de message (MTS).

1.2.6.2 MHS

La figure I.5 décrit la découpe en couche du MHS.

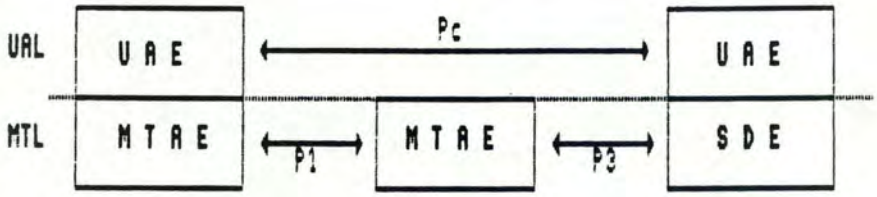


Fig. I.5 : Découpe en couche du MHS

Le protocole P1 est le protocole de Transfert de Message, il gére les interactions entre les entités MTA au sein du MTS.
Le protocole P3 est le protocole de soumission et de livraison, il gére les interactions entre les entités MTA et les entités SDE (Submission and Delivery Equipment). Celles-ci permettent à une UAE non associée directement à un MTA de dialoguer à un MTA éloigné.
'Pc' représente une famille de protocoles définis par les utilisateurs qui gére la communication au niveau du traitement du contenu des messages.

1.2.6.3 IPMS

La figure I.6 décrit la découpe en couche pour le service d'IPM. le protocole 'Pc' a été remplacé par le protocole P2. Le protocole P2 est le protocole de Messagerie Inter-Personnelle, il gère le dialogue entre entités UA.

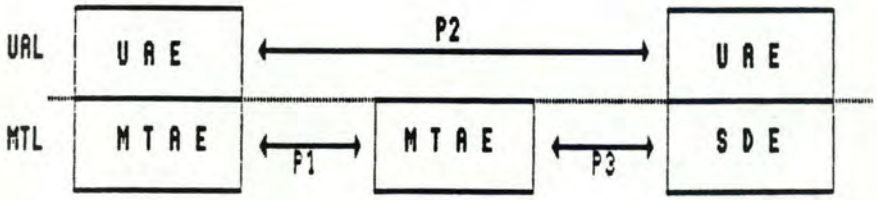


Fig. I.6 : Découpe en couche du IPMS

1.2.7 NORMES ET RECOMMANDATIONS

Nous reprenons ci-dessous la listes des normes et recommandations concernant le système de messagerie X.400.

1.2.7.1 X.400 : MHS Modèle et éléments de service

Ce document présente le modéle fonctionnel du MHS et son applicabilité à une série de configurations physiques et organisationnelles. Il définit les concepts d'adressage, de routage et les différents élements de service du MHS. Il situe enfin le MHS dans le cadre du modèle en couche ISO.

1.2.7.2 X.401 : MHS service de base et éléments optionnels

Ce document décrit la liste des services standards (de base et optionnels) offerts par le MHS. Il présente le service de messagerie interpersonnelle (IMP) et le service de tranfert de messages (MT).

1.2.7.3 X.408 : MHS règles de conversion et d'encodage

Ce document décrit les types d'encodage du MHS et les conversions entre les différents types ainsi que les règles à respecter pour définir de nouvelles conversions.

1.2.7.4 X.409 : MHS syntaxe et notation

Ce document définit la syntaxe et les modes de représentation physique utilisés pour échanger des informations entre entités.

1.2.7.5 X.410 : MHS opération à distance et RTS

Ce document définit le concept d'opération à distance, spécifie le RTS (Reliable Transfer Server) et décrit les primitives de niveau inférieur utilisées par MHS.

1.2.7.6 X.411 : MHS Message transfer Layer

Ce document spécifie les diverses primitives offertes par la couche de tranfert de message (MTL Message Transfer Layer) et définit les protocoles P1 (Message Transfer Protocol) et P3 (Submission and delivery protocol).

1.2.7.7 X.420 : MHS Messagerie interpersonnelle: Couche UA

Ce document décrit le modèle de la messagerie interpersonnelle (IPM), les fonctions offertes par le protocole P2 (User Agent) et la structure SFD (Simple Formattable Document).

1.2.7.8 X.430 : MHS Protocole d'accès pour terminaux TELEX

Ce document décrit comment les terminaux TELEX et TELETEX peuvent accéder au MHS.

2. CHAPITRE II : ENVIRONNEMENT ET OBJET DE L'ETUDE

Le chapitre deux de cet ouvrage expose brièvement, dans sa première partie, l'environnement qui fut le notre pendant la réalisation de cette étude; la deuxième partie est entiérement consacrée à une présentation détaillée du sujet sur lequel porte ce travail.

2.1 ENVIRONNEMENT D'ETUDE : l'I.I.H.E.

L'Inter-university Institute for High Energies (IIHE) est un laboratoire de recherche en physique des particules. Il s'agit d'une collaboration entre l'Université Libre de Bruxelles et la Vrije Universiteit Brussel. Outre des physiciens, le laboratoire est constitué d'une équipe d'informaticiens spécialisés dans le domaine des télécommunications. Le laboratoire participe à de nombreuses expériences réalisées entre-autres sur l'accélérateur de particules du C.E.R.N. à Genève, Suisse. Les chercheurs de l'IIHE utilisent les télécommunications pour demander le démarrage de celles-ci et pour en recevoir les résultats. La quantité de données est telle que le laboratoire s'est doté d'une antenne satellite, afin de concevoir et de pouvoir utiliser une liaison à grande vitesse (projet HELIOS). Le parc de l'IIHE se compose, entre-autres, d'un Bull SPS-7 qui est relié à l'antenne satellite, d'un Digital Equipment System 20 (DEC-20), d'un VAX 8200 sous VMS 4.6 et de micros VAX. Toutes ces machines sont interconnectées via un réseau local Ethernet.

2.2 OBJET DE L'ETUDE : LE LOGICIEL X.400 DE DIGITAL

L'objet de cette étude est de décrire, d'implanter et de tester le logiciel Message Router X.400 Gateway de Digital Equipment. Ce logiciel permet à un système VAX/VMS d'échanger des messages conformes à la norme X.400 du Comité Consultatif International des Télégraphes et Téléphones ou CCITT, avec n'importe quel autre système implémentant également ces recommandations.

Notre mission consistait tout d'abord à installer le logiciel et son environnement, ensuite à étudier des méthodes de test de logiciel de communication de manière à pouvoir concevoir et développer des outils permettant de réaliser des tests de conformité et d'interopérabilité de l'implémentation X.400 de Digital. Il nous a également été demandé de mener à bien, dans la mesure du possible, ces tests de conformité.

C'est pendant cette période que nous avons eu le plaisir de réaliser le premier échange d'un message X.400 en Belgique, le 6 janvier 1988. Ce message fut envoyé depuis les Facultés Universitaires Notre-Dame de la Paix de Namur vers l'IIHE La trace de ce message peut être trouvée en annexe.

2.3 LE LOGICIEL X.400 DE DIGITAL EQUIPMENT ET SON ENVIRONNEMENT

Après une étude de l'architecture générale du logiciel X.400 de Digital Equipement et des différentes parties qui le composent, nous analyserons quelques problèmes rencontrés lors de leur utilisation.

2.3.1 ARCHITECTURE GENERALE

Dans ce qui va suivre, nous avons choisi de garder la terminologie anglaise utilisée par Digital pour définir les noms des logiciels ainsi que des modules composant ces logiciels, ceci afin de garder une forte cohérence avec la documentation. On trouvera les références de cette documentation dans <VAX-PSI>, <VAX-VOTS>, <VAX-OSAK>, <VAX-MR> et <VAX-MRX>.

Qu'il nous soit permis, avant de passer à une présentation plus détaillée des produits, de situer brièvement ceux-ci par rapport au modèle ISO. La figure II.1 présente la répartition des logiciels dans les sept couches du modèle de référence ISO. Les logiciels Message Router (MR) et Message Router X.400 Gateway (MRX) sont des applications. Les services session et transport sont respectivement assurés par les produits OSI Session Application Kernel (OSAK) et Vax OSI Transport Service (VOTS), les services des couches inférieures sont pris en charge par un ensemble de produits logiciels appelé Packetnet Switching Interface (PSI). MR et MRX n'utilisent pas les services de la couche présentation.

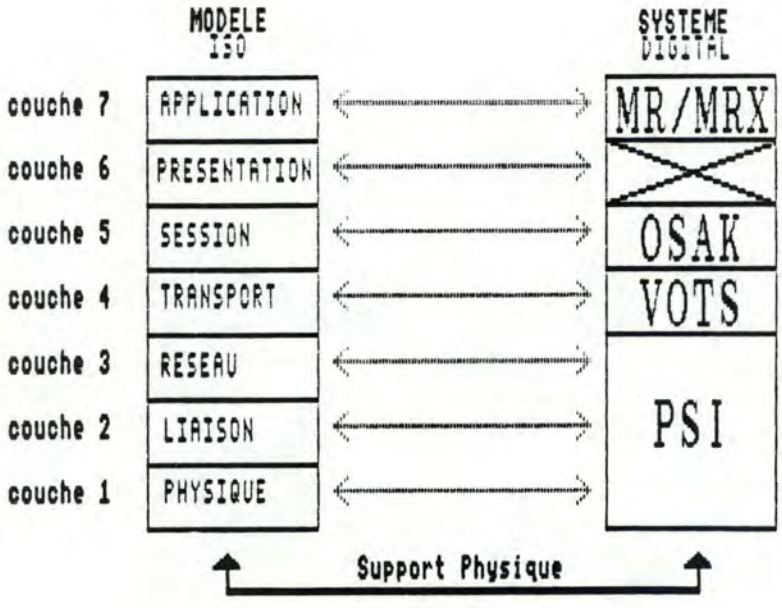


FIG II.1 : Répartition des logiciels Digital dans les sept couches ISO.

2.3.2 PACKETNET SWITCHING INTERFACE (PSI)

2.3.2.1 Description

Packetnet Switching Interface (PSI) est un ensemble de produits logiciels permettant à un système VAX/VMS de se connecter à un réseau à commutation par paquets (PSDN) supportant le protocole X.25 (interface entre équipement terminal de traitement de données (DTE) et équipement de terminaison de circuits (DCE) pour terminaux fonctionnant en mode-paquet, raccordés à un PSDN <MACCHI-87>), et de fonctionner comme un DTE. VAX PSI permet à un processus de communiquer avec un autre processus sur un autre système (Digital ou non) connecté à un PSDN. PSI permet enfin à deux systèmes VAX/VMS d'établir une connexion DECnet à travers un PSDN.

L'interface X.25 permet l'établissement, l'acceptation, le rejet, la redirection et la libération d'un circuit virtuel, la transmission et la réception de données.

L'interface X.29 permet à des terminaux asynchrones connectés sur un PSDN à l'aide d'un service d'assemblage et de désassemblage de paquets (PAD) de communiquer avec un système VAX/VMS à la manière d'un terminal local. Les communications par terminaux supportées par VAX PSI sont celles respectant les recommandations X.3 (service PAD sur un réseau public), X.28 (interface DTE/DCE pour l'accès d'un DTE asynchrone à un service PAD situé dans le meme pays) et X.29 (procédures d'échange de l'information de commande et des données de l'utilisateur entre un PAD et un DTE fonctionnant en mode paquet ou un autre PAD) du CCITT.

2.3.2.2 Bases de données PSI

PSI utilise les bases de données suivantes :

- une base de données permanente (**PERMANENT DATABASE**).
Elle contient la configuration statique complète de PSI, les caractéristiques des lignes physiques, la définition des différents serveurs, les informations de protocole (Adresse DTE, nom local, nom du réseau X.25).

- une base de données volatile (**VOLATILE DATABASE**)
Elle contient les informations dynamiques liées à l'activité de PSI telles que le nombre et le type des circuits virtuels ouverts et utilisés, les compteurs, les timers, ...

- une base de données de contrôle d'accès (**GRANT DATABASE**)
Elle permet de spécifier les utilisateurs autorisés à utiliser l'accès X.25 et/ou l'accès X.29 ainsi que les adresses DTE des systèmes éloignés avec lesquels l'établissement de circuits virtuels est permis.

2.3.2.3 Outils et Primitives PSI

PSI offre des outils permettant la gestion du DTE local :

NETWORK CONTROL PROGRAM (NCP ou Programme de contrôle du réseau) est l'utilitaire qui permet aux responsables de configurer le logiciel PSI, de définir les composants et les paramètres dans les bases de données, de les modifier et de les afficher, de contrôler le bon fonctionnement des opérations.

INSTALLATION CHECKOUT PROCEDURE (ICP) permet la vérification de l'installation en ouvrant, utilisant et fermant un circuit virtuel.

USER ENVIRONMENT TEST PACKAGE (UETP) permet, entre-autre, la vérification des appareils physiques, l'analyse des problèmes et le diagnostic des erreurs Hardware et Software.

PSI-AUTHORIZE gère les contrôles d'accès X.25 et X.29.

PSI-TRACE permet de suivre à la trace le déroulement des actions. Il permet d'examiner le contenu exact des paquets reçus ou envoyés, de calculer le nombre d'évènements survenus (Reset, Timeout, ...) et d'en faire des statistiques.

PSI-ACCOUNTING permet de comptabiliser les coûts d'utilisation du logiciel.

2.3.3 VAX OSI TRANSPORT SERVICE (VOTS)

2.3.3.1 Description

Le logiciel VAX OSI Transport Service (VOTS) permet à un processus tournant sur un système VAX/VMS d'établir une connexion transport avec un autre processus sur un autre système VAX/VMS supportant VOTS ou avec n'importe quel système implémentant le protocole transport du modèle ISO. Une connexion transport est une liaison bi-directionnelle de transfert de données entre deux processus. Ces processus peuvent établir, gérer et terminer une connexion transport.

La couche transport fournit un interface avec le réseau, indépendamment de la manière dont celui-ci transfère les données.

VOTS peut être utilisé sur plusieurs types de réseaux :
a) sur des réseaux locaux CSMA/CD (IEEE 802.3),
b) sur des réseaux X.25, en conjonction avec VAX PSI,
c) sur un système VAX/VMS utilisant DECnet.

VOTS implémente les classes 0, 2 et 4 du protocole de transport. Les protocoles des classes 0 et 2 sont souvent les plus indiqués pour les réseaux de type X.25 qui sont les plus fiables et offrent un taux d'erreur suffisamment bas pour pouvoir se passer d'un contrôle d'erreur supplémentaire. Le protocole de classe 4 est approprié pour les réseaux locaux où il n'y a pas garantie de fiabilité.

2.3.3.2 Fonctionnalités

VOTS offre une série de fonctionnalités aux utilisateurs du protocole transport pour établir des connexions et réaliser l'échange de données.

2.3.3.2.1 Fonction de connexion

Pour établir une connexion, une entité de niveau 5 fait une requête de connexion transport vers une autre entité de niveau 5. La couche transport établit la connexion si les deux entités sont d'accord sur le type de service transport à utiliser. Une fois la connexion établie avec succès, le transfert des données peut avoir lieu dans chaque direction. Quand les deux entités ont terminé de communiquer, chacune d'entre-elle peut libérer la connexion transport.

2.3.3.2.2 Fonction de traduction d'adresse

La couche transport doit pouvoir traduire les adresses transport en adresses réseau. En établissant une connexion, la couche transport fournit des adresses uniques pour les entités communicantes. Quand un processus fait une requête de connexion transport, il utilise une adresse transport de la forme : TSAP.VOTS_address. TSAP (Transport Service Access Point) est utilisé par le logiciel transport à qui la requête est envoyée pour déterminer le processus devant traiter le demande de connexion transport. VOTS_address est constitué du nom d'un Network Service Provider (fournisseur de service réseau) et d'une adresse réseau. Le NS-Provider permet à VOTS de savoir sur quel type de réseau demander l'établissement d'une connexion réseau et aussi de pouvoir s'identifier selon le format d'adressage utilisé par le réseau. La base de données

des NS-Providers est étudiée plus en détail au paragraphe 2.3.2.2.1.

Exemple 2.1 : FNDP_VENUS.X25_DCS%28160000

L'exemple 2.1 montre une adresse transport. FNDP_VENUS est un TSAP d'un noeud éloigné. La VOTS_address (X25_DCS%28160000) est constituée d'un NS-Provider (X25_DCS) et d'une adresse réseau (28160000) qui se décompose en une adresse DTE (2816000) et une sous-adresse (0).
Une description plus précise du format d'adressage utilisé dans l'implémentation ISO de DIGITAL peut être trouvée dans le chapitre III "ADRESSAGE ET ROUTAGE".

2.3.3.2.3 Multiplexage

Les classes 2 et 4 du protocole de transport permettent une utilisation plus efficace des services offerts par la couche 3 en utilisant une même connexion réseau pour plusieurs connexions transport.

2.3.3.2.4 Détection et recouvrement d'erreurs

La couche transport doit assurer que l'échange de messages se déroule sans corruption ou perte et que les messages sont délivrés en séquence. C'est le rôle de la classe 4 du protocole transport.

2.3.3.3 Bases de données VOTS

Les bases de données utilisées par VOTS contiennent des informations sur le noeud local et les noeuds éloignés avec lesquels des connexions transport sont ou ont été établies. Leurs objectifs sont de déterminer le type de service transport fourni par VOTS et de collecter des données statistiques.

La description complète de tous les éléments des bases de données est longue et fastidieuse. Le lecteur intéressé trouvera dans <VAX-VOTS> le détail de ce qui va être exposé brièvement ici.

2.3.3.3.1 Base de données Network Service Providers (NS-PROVIDER)

Le rôle de la base de données Network Service Providers (NS-Providers ou Fournisseurs de Service Réseau) est de contrôler comment VOTS doit utiliser un service réseau. VOTS autorise trois types de service réseau: IEEE 802.3, INTERNET et X.25. A chaque NS-Provider doit correspondre une entrée dans la base de données. Les informations sur un NS-Provider peuvent se regrouper en trois groupes : informations d'identification, de contrôle et statistiques.

Les informations d'identification concernent le nom et le type du réseau sur lequel opère le NS-Provider, l'adresse et, éventuellement, la sous-adresse du noeud local.

Les informations de contrôle reprennent la ou les classes de protocole transport que VOTS doit utiliser sur ce NS-Provider, le nombre maximum de connexions réseau pouvant être ouvertes et le nombre de connexions transport pouvant être multiplexées sur un même circuit virtuel. Sont également définis, le nombre de tampons alloués à la transmission et à la réception des données, ainsi qu'un certain nombre de timers dont le rôle

est d'indiquer à VOTS l'intervalle de temps pendant lequel il doit attendre avant de cloturer une connexion ou de renvoyer un TPDU selon le type d'événement. Enfin, on retrouve ici les tailles des NSDU, des TPDU, des paquets X.25 et la taille de la fenêtre X.25.

Les statistiques retiennent le nombre de bytes total reçus et émis sur ce NS-Provider, le nombre d'erreurs survenues, le nombre de retransmissions de TPDU et de réinitialisations de circuits virtuels.

2.3.3.3.2 Base de données des Transport Service Access Points (TSAP)

La base de données des Transport Service Access Point (TSAP ou Point d'Accès du Service Transport) contient des informations sur les utilisateurs transport. Ces informations permettent à VOTS de déterminer à quel processus est destinée une demande de connexion transport.

On distingue deux types de TSAP : les TSAP actifs qui sont des processus en train de tourner sur le système et les TSAP passifs qui sont des processus déclenchés lors de l'arrivée d'une requête de connexion transport.

TSAP id : FNDP_VENUS

DECnet compatible	*= no*
TSAP name	*= 2180097A*
Account	*= not initialised*
User	*= not initialised*
Password	*= not initialised*

Fig. II.2a : Entrée d'un TSAP actif.

TSAP id : VOTSIVP

DECnet compatible	*= no*
TSAP name	*= (SYS$TEST)VOTSIVPRESP.COM*
Account	*= VOTSTESTER*
User	*= VOTSTESTER*
Password	*= VOTSTESTER*

Fig. II.2b : Entrée d'un TSAP passif.

Le TSAP (FNDP_VENUS) de la figure II.2a est un TSAP actif. Son TSAP name est son numéro d'identification de processus sur le système (2180097A) et ce n'est pas un objet DECnet (DECnet compatible = no).

Le TSAP (VOTSIVP) de la figure II.2b est un TSAP passif. Son TSAP name est un nom de procédure (<SYS$TEST>VOTSIVPRESP). Cette procédure sera déclenchée par VOTS lors de l'arrivée d'une requête de connexion transport pour ce TSAP. Les paramètres Account, User et Password indiquent le compte (VOTSTESTER) qui doit être utilisé à l'exécution de cette procédure. Ces paramètres doivent être spécifiés dans la requête de connexion, sinon le système refusera le démarrage de la procédure.

Si dans la base de données TSAP, les paramètres Account, User et Password sont vides ('not initialised') alors, VOTS utilisera un compte par défaut défini dans la base de données de la couche transport.

2.3.3.3.3 Base de données de la couche Transport (TRANSPORT)

La base de données de la couche transport (fig. II.3) définit le service transport offert par VOTS.

TRANSPORT ALL information

Max Receive Buffers	*= 40*
Max Trans. Connections	*= 100*
Current Trans. Connections	*= 0*
Local Acknowledgment	*= 01.000 (sec)*
RTSP database enable	*= On*
Default access	*= Both*
Default Account	*= OSI*
Default User	*= OSI*
Default Password	*= OSI*

Fig. II.3 : Base de données de la couche Transport.

Elle permet de contrôler et de signaler :
- la taille des tampons (Max Receive Buffers) alloués à la réception,
- le nombre maximum de connexions transport simultanées (Max Trans. Connections) et en cours (Current Trans. Connections),
- le temps de réponse estimé du noeud local (Local Acknowledgment),
- l'enregistrement de statistiques dans la base de données RTSP,
- l'autorisation d'accès au service transport (Both = accès autorisé en entrée et en sortie),
- le compte VMS (OSI) utilisé par VOTS pour démarrer des procédures pour lesquelles aucune information d'accès n'est spécifiée dans la base de données TSAP.

2.3.3.3.4 Base de données des Remote Transport Service Providers (RTSP)

La base de données des Remote Transport Service Providers (RTSP ou Fournisseurs de Service Transport Eloignés) contient les détails des interactions passées entre le noeud VOTS local et les fournisseurs de service transport éloignés. Cette base de données possède une entrée pour chaque noeud avec lequel des connexions transport ont été réalisées. Lors de l'établissement de la première connexion transport avec un noeud, VOTS crée une entrée dans la base de données. RTSP est uniquement une base de données statistiques. Les informations qu'elle contient sont utiles pour isoler un problème survenant sur le réseau.

Une entrée de la base de données RTSP est une suite de compteurs pouvant être divisée en deux parties.

1. Les compteurs de situations normales indiquent le nombre de connexions transport complètes qui ont été effectuées à ce jour avec le noeud concerné, ainsi que la somme, exprimée en bytes et en DT-TPDU (Data Transfer), des données échangées.

2. Les compteurs de situations d'exceptions rapportent le nombre de requêtes de connexions transport rejetées par le partenaire éloigné et par le noeud local. Ces dernières sont classées par causes de rejet : mauvais CR-TPDU (Connection Request), mauvais TSAP, défaut de temps. Sont comptabilisées également dans RTSP, les violations de protocoles du noeud éloigné signalées par VOTS et inversément. Enfin RTSP permet de voir le nombre de retransmissions de TPDU effectuées.

2.3.3.3.5 Base de données de trace d'événement (EVENT LOGGING)

La base de données Event Logging contient des informations provenant de l'enregistrement des événements et des erreurs survenues lors des connexions transport. Les événements peuvent être consignés sur trois types de journaux : un fichier, un processus de monitoring ou une console. Une entrée de la base de données (fig. II.4) montre dans quel état est l'enregistreur d'événements et quel type d'événements il consigne.

```
EVL CONSOLE ALL information
timed at 7-FEB-1988 12:43:27.99
EVL Device : CONSOLE
State      = Off
Name       = Not initialised
Events     = 1 2 3 4 5 6
```

Fig. II.4 : Entrée de la base de données Event Logging.

Le type d'enregistreur est défini par **EVL Device** et peut prendre comme valeur : CONSOLE, FILE ou MONITOR. **State** indique l'état opérationnel de l'enregistreur (Off : pas d'enregistrement, On : les événements sont détectés et enregistrés, Hold : les événements sont détectés et mis en attente d'être enregistrés). **Name** donne le nom du moyen d'enregistrement. Ce moyen peut être un nom de fichier logging, un nom de processus de monitoring ou un nom d'appareil faisant office de console. **Events** contient une combinaison de numéros d'événements qu'il faut enregistrer, dont la description se trouve dans <VAX-VOTS>.

2.3.3.3.6 Bases de données Inter-réseau (INTERNET)

Développé par le Departement of Defense (DoD) américain, le protocole inter-réseau ou INTERNET est le protocole de communication entre des réseaux de types différents.
Le trajet des données depuis leur origine vers leur destination passe par un ou des systèmes intermédiaires reliant un ou plusieurs réseaux. Les réseaux communiquant entre-eux au moyen du protocole Internet peuvent être vus comme des parties d'un seul grand réseau. VOTS supporte Internet pour les connexions transport de classe 4 uniquement.
Un noeud VOTS peut agir comme un utilisateur Internet pouvant :

- envoyer et recevoir des Internet Protocol Data Units (IPDU) à un autre utilisateur Internet du même réseau ou à un système intermédiaire pour atteindre un utilisateur sur un autre réseau déservi par le système intermédiaire,
- fixer une durée de vie limitée à un IPDU quittant le système,
- réassembler des messages à partir des IPDU reçus,
- rapporter à un système d'un autre réseau la réception de données erronées,
- ajouter des caractères de contrôle aux IPDU émis pour faciliter la détection d'erreur à la réception.

VOTS ne pouvant pas relayer les unités de données, il ne peut donc pas servir de système intermédiaire avec un autre réseau. Pour VOTS, Internet est un fournisseur de service réseau (NS-Provider) implicite.
Lorsqu'une application veut envoyer des données par le protocole Internet, elle utilise une adresse VOTS de la forme :

```
INTERNET % Internet_subnetwork_address.
```

Pour pouvoir utiliser Internet, VOTS a besoin de deux bases de données : la base de données Internet et la base de données de transformation d'adresse.

La base de données Internet contient les informations nécessaires au fonctionnement du protocole Internet. Ces informations permettent de définir la durée de vie d'un IPDU, la présence de sommes de contrôle (checksums) dans les IPDU émis, l'espace mémoire alloué à Internet pour réassembler les IPDU ainsi que la taille maximale d'un IPDU que Internet tentera de réassembler. Internet permet également de définir le nombre de bytes de données de l'IPDU refusé qu'il devra inclure dans un IPDU rapportant une erreur.

La base de données de transformation d'adresses (fig. II.5a) contient les informations utilisées par Internet pour diriger les données à travers plusieurs réseaux.

```
Destination     = 12647A
Intermediate    = X25_DCS%22101680
Via             =
```

Fig. II.5a : Base de données de transformation d'adresses.

Chaque entrée dans la base de données définit une destination, c'est à dire une adresse par laquelle le noeud destinataire et le noeud passerelle peuvent identifier l'endroit où doivent arriver les données. Il peut y avoir autant de formats d'adresses destination que de réseaux interconnectés entre-eux, chaque réseau ayant sa propre syntaxe d'adressage. L'adresse intermédiaire 'Intermediate' est l'adresse d'une passerelle vers le noeud cible. Le paramètre 'Via' spécifie une route complète ou partielle de tous les systèmes passerelles par lesquels les données doivent transiter pour atteindre leur destination.

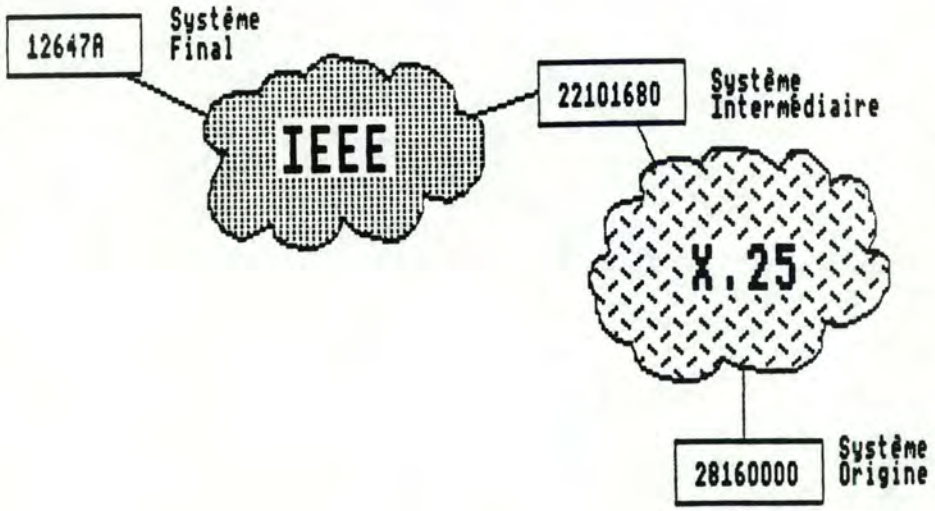


Fig. II.5b : Exemple de configuration Inter-réseau.

La figure II.5b illustre la situation des noeuds utilisés à la figure II.5a. Le noeud émetteur (28160000) et le noeud destination (12647A) se trouvent sur des réseaux différents (X.25 et IEEE). Pour pouvoir envoyer des données au noeud destination, le système origine doit passer par un système intermédiaire (22101680) qui sert de passerelle entre les réseaux X.25 et IEEE.

La figure II.5a montre comment le système origine doit définir les différents noeuds dans sa base de données Internet. **Destination** est l'adresse Internet du noeud cible, soit 12647A. **Intermediate** est la VOTS_address du noeud passerelle : X25_DCS est le NS-Provider, 2210168 l'adresse DTE et 0 la sous-adresse. Le paramètre VIA est nul car le noeud intermédiaire est directement accessible sur le réseau X.25.

2.3.3.4 Outils et primitives VOTS

VOTS offre une série d'utilitaires (fig. II.6) permettant aux utilisateurs d'accéder aux services de la couche transport et aux opérateurs de configurer le logiciel et contrôler le bon déroulement des opérations.

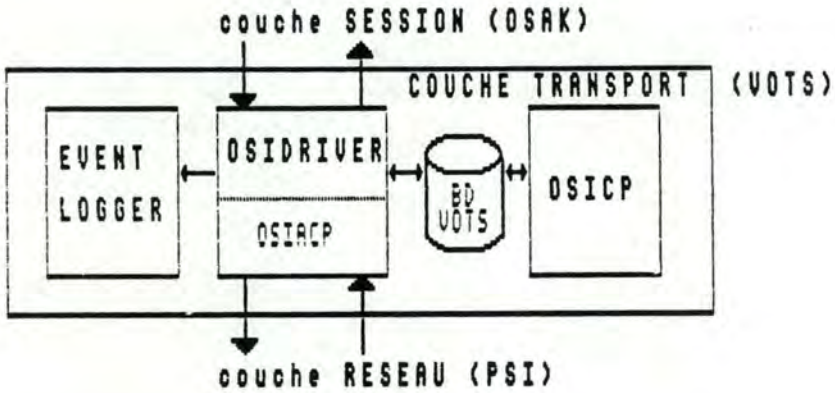


Fig II.6 Architecture de VOTS.

OSIDRIVER est le module implémentant le protocole transport.

OSI Ancillary Control Program **(OSIACP)** coopère avec OSIDRIVER pour fournir le service transport. C'est lui qui est chargé de diriger les requêtes de connexion transport vers la couche réseau ou vers les processus de niveaux supérieurs.

OSI Control Program **(OSICP)** est un utilitaire interactif permettant la gestion des composants et des bases de données VOTS.

OSI Event Logger **(OSIEVL)** permet de garder une trace des événements significatifs.

2.3.4 VAX OSI APPLICATION KERNEL (OSAK)

2.3.4.1 Description

VAX OSI Application Kernel (OSAK) implémente la cinquième couche du modèle ISO : la couche session. Celle-ci ajoute aux services offerts par la couche transport des facilités que les applications utilisent pour structurer et contrôler leur communication. OSAK utilise le service transport fourni par VOTS pour permettre à un processus sur un système VAX/VMS d'établir une connexion session avec un autre processus sur un autre système VAX/VMS supportant OSAK ou avec n'importe quel autre processus sur un système implémentant la couche session de l'ISO. Deux processus peuvent avoir plusieurs sessions concourantes ou un processus peut avoir plusieurs connexions sessions concourantes (chacune avec un partenaire différent).

OSAK fournit des services réalisant :
- l'établissement d'une connexion session entre deux entités d'adresses sessions spécifiées,
- la gestion du dialogue, pour définir une ou plusieurs activités, contrôler le type d'échange des données (half- ou full-duplex), reprendre une session qui aurait été interrompue, permettre à un utilisateur d'utiliser les services de la couche session et rapporter les événements et les erreurs,
- la libération de session (terminaison ou avortement).

Les services sessions sont divisés en unités fonctionnelles. Une unité fonctionnelle est un groupe de services qui aident un utilisateur à exprimer ses besoins pour une connexion session. Le service session standard définit trois sous-ensembles d'unités fonctionnelles :
- Basic Combined Subset (BCS),
- Basic Synchronised Subset (BSS),
- Basic Activity Subset (BAS).

Une unité fonctionnelle est choisie pour une connexion si les deux entités concernées proposent son utilisation et si elle est supportée et par OSAK et par l'implémentation session avec laquelle on désire communiquer.

2.3.4.2 Base de données OSAK

OSAK utilise une base de données où sont consignés tous les renseignements concernant le niveau session du modèle ISO.

La base de données Session contient des informations statistiques et de contrôle. OSAK y range les données statistiques des interactions achevées entre OSAK et un autre service session éloigné.
Les données statistiques concernent le nombre de requêtes de connexion issues ou reçues , le nombre de requêtes reçues qui ont été refusées, le nombre d'interruptions et de libérations de connexion session envoyées et reçues.
La base de données permet également de définir comment OSAK utilise les connexions transport, le temps pendant lequel OSAK reste attaché à un TSAP après la déconnexion d'une liaison session et le temps pendant lequel OSAK attend une demande de déconnexion d'un service session éloigné, au-delà duquel il cloturera la session.

Enfin, la base de données Session permet de définir et de contrôler l'état et le type des appareils d'enregistrement d'erreur.

2.3.4.3 Outils et primitives OSAK

La figure II.7 présente l'architecture des outils composant OSAK.

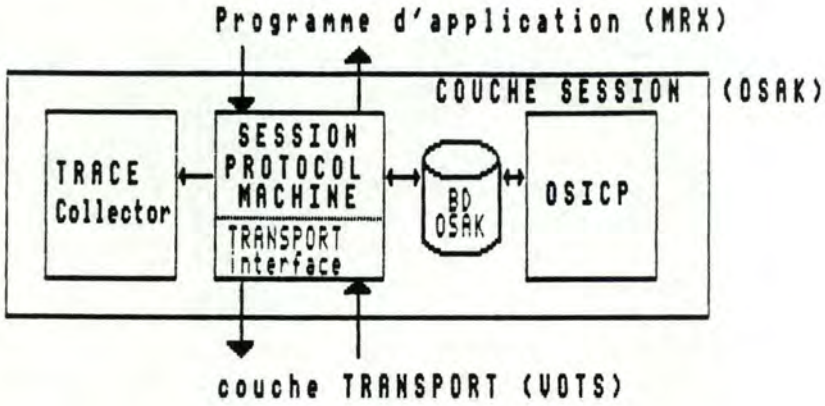


Fig. II.7 : Architecture d'OSAK

La **SESSION PROTOCOL MACHINE** (SPM) qui est le noyau de l'implémentation du protocole session et qui est constitué de structures de données et de routines qui implémentent le protocole session.

L'interface transport (**TRANSPORT INTERFACE**) qui est le moyen de communication entre OSAK et la couche transport (VOTS). C'est lui qui transforme les SPDU en TSDU en effectuant la segmentation des données quand cela est nécessaire.

Le **TRACE COLLECTOR** qui traite les détails et les évènements enregistrés pendant la surveillance des SPDU. Celle-ci suit le flux des SPDU entre le système local et un système à distance. Les informations données par le Trace Collector peuvent, après traitement par l'utilitaire VOTSTRACE, aider l'utilisateur à isoler des problèmes d'échange de protocole entre OSAK et un fournisseur de service session éloigné.

L'utilitaire **OSICP** permet de consulter et de mettre à jour la base de données Session.

2.3.5 MESSAGE ROUTER (MR) ET MESSAGE ROUTER X.400 GATEWAY (MRX)

Ce paragraphe décrit le Message Router et le Message Router X.400 Gateway. Nous avons choisi de présenter ces deux produits simultanément, le Message Router X.400 Gateway étant un complément du Message Router.

Avant de continuer, et pour éviter toute confusion, nous allons dès à présent définir deux objets similaires par leur nom, mais fondamentalement différents par leurs fonctionnalités. Nous appelons **mailbox VMS** un mécanisme du système d'exploitation permettant la communication entre processus sur un système VAX/VMS. Nous appelons par ailleurs **mailbox MR** ou, plus simplement **mailbox**, l'objet qui permet au Message Router d'identifier un utilisateur du système VMS. C'est via cette mailbox MR qu'un abonné au système de messagerie pourra envoyer ou recevoir des messages.

Le Message Router est un agent de transfert de messages de type store-and-forward opérant sur le réseau DECnet. Le Message Router X.400 Gateway permet l'interconnexion entre le réseau ISO et le réseau de Message Routers.

L'addition du Message Router X.400 Gateway au réseau MR rend les messages quittant le domaine conformes aux recommandations X.400, et permet également la réception de messages provenant de domaines implémentant du logiciel X.400. Tous les messages à destination du réseau ISO doivent passer par la passerelle avant de sortir et tous les messages destinés à un abonné du réseau de Message Routers doivent transiter par le Message Router X.400 Gateway avant d'arriver à destination.

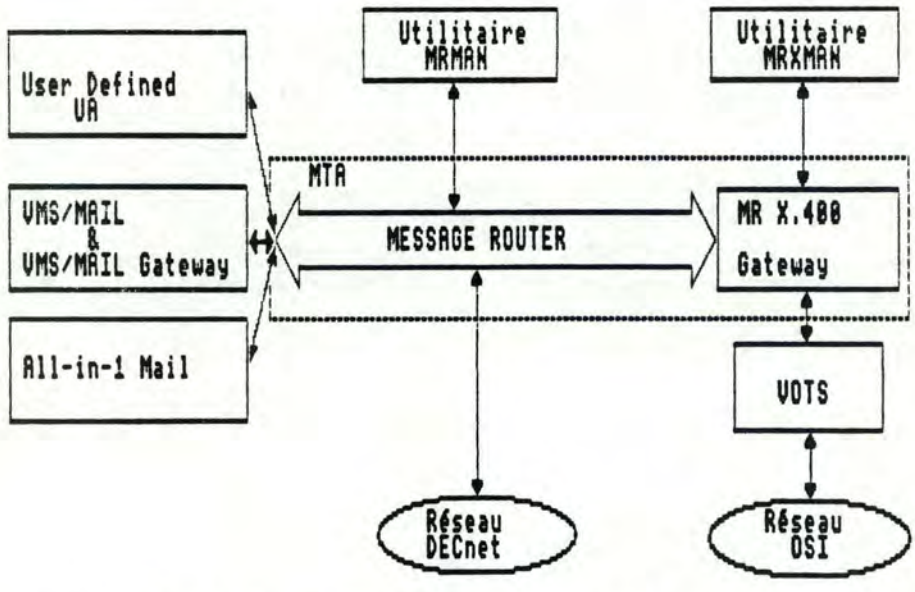


Fig. II.8 : Environnement du Message Router.

La figure II.8 décrit l'ensemble du système de communication store-and-forward de Digital Equipment. Cette figure ne présente que les logiciels de haut niveau, depuis la couche transport jusqu'à la couche application. Rappelons cependant que le produit réalisant l'interface réseau (VAX PSI) peut servir aussi bien à l'ouverture d'un circuit virtuel qu'à l'établissement d'une liaison DECnet sur un PSDN.

Le Message Router (MR) est le centre du système. C'est lui qui permet

aux User Agents de poster ou de lire des messages. Le Message Router permet de relayer des messages sur le réseau DECnet uniquement.

Le Message Router permet l'accès à plusieurs types de User Agents :

- ALL-IN-1 Mail (user agent de Digital),

- un user agent développé par l'utilisateur (User Defined UA) au moyen du Message Router Interface Routines, une boite à outils logiciels offrant des fonctions permettant la construction d'un UA,

- le VMS/Mail en combinaison avec le logiciel VMS/Mail Gateway,

- un user agent de gestion du Message Router (MRMAN).

Enfin, si l'on ajoute le Message Router X.400 Gateway et VOTS (VAX OSI Transport Service), on peut communiquer tant avec des systèmes VAX/VMS implémentant MR et MRX que des systèmes implémentant un protocole de communication conforme aux normes X.400 du CCITT. On considérera qu'un noeud implémentant MR et MRX est un agent de transfert de messages (MTA) du réseau ISO.

Quelques remarques peuvent encore se dégager de l'analyse de ce schéma : d'abord, le Message Router, le Message Router X.400 Gateway et le VMS/Mail Gateway sont des applications. VMS/Mail est le nom du système de messagerie du VAX. Ce système offre des fonctionnalités similaires à celles d'un User Agent (édition de messages, classement, ...). Le logiciel VMS/Mail Gateway permet d'utiliser ces fonctionnalités avec le Message Router. MRX possède son propre User Agent de gestion : MRXMAN.

Ensuite, on remarquera que deux systèmes VAX/VMS peuvent faire du store-and-forward aussi bien sur le réseau DECnet de Digital, que sur le réseau X.400.
Enfin, le lecteur attentif et perspicace aura constaté l'absence du logiciel OSAK (OSI Session Application Kernel) de ce schéma. Il ne s'agit nullement d'un oubli ou d'une erreur de notre part. En effet, MRX contient un sous-ensemble du logiciel OSAK suffisant pour établir les connexions sessions qui lui sont nécessaires pour réaliser l'échange de messages.

2.3.5.1 Le Message Router (MR)

2.3.5.1.1 Description

Le Message Router est un agent de transfert de message de type store-and-forward qui opère sur le réseau DECnet de Digital Equipment. Un réseau de Message Routers forme un système de transfert de messages.

Le Message Router est invisible aux utilisateurs et interagit avec des User Agents (UA). Un utilisateur peut être une personne ou un processus d'application. Un User Agent est un logiciel d'application qui traite les messages pour les utilisateurs. La figure II.9 montre l'interaction entre les utilisateurs, les User Agents et les Message Routers.

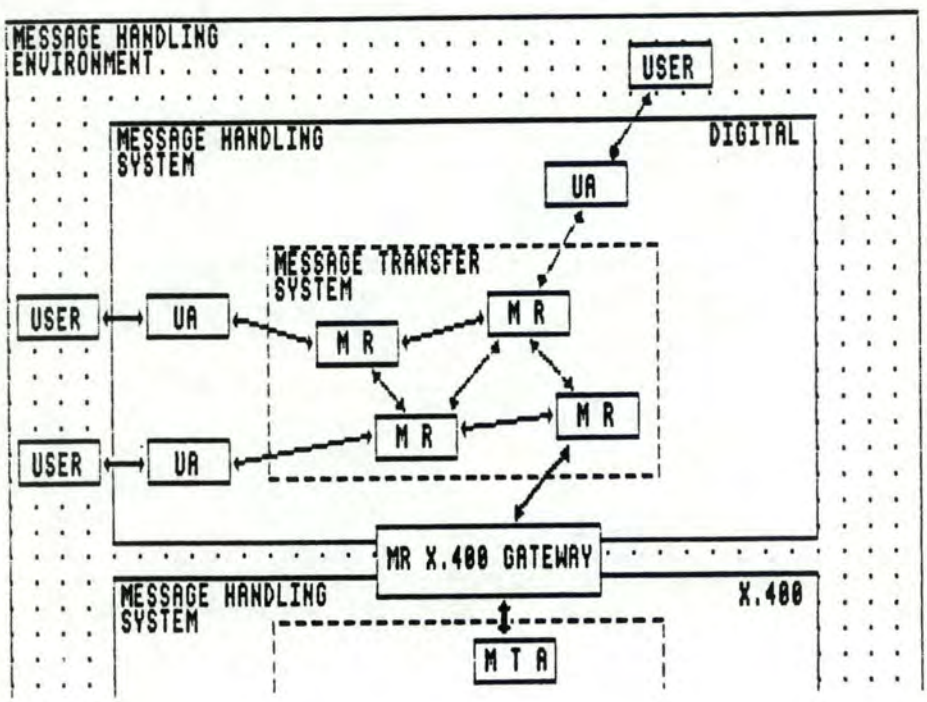


FIG II.9 : Interactions entre utilisateurs, UA et MR.

Chaque Utilisateur interagit avec le réseau MR par l'intermédiaire de son User Agent pour créer ou délivrer des messages. Les utilisateurs et les User Agents sont identifiés dans le Message Router par des mailboxes. Un Message Router possède une base de données qui contient une liste de mailboxes. Elle sert à diriger les messages.

Les Message Routers sur des noeuds séparés communiquent entre-eux pour transférer des messages sur le réseau. Si un noeud sur une route ne fonctionne pas, le Message Router du noeud précédent garde les messages destinés à ce noeud, jusqu'au moment où il sera à nouveau opérationnel. Le message enregistré est alors envoyé et continue sa route à travers le réseau.

Les liaisons de Message Router à Message Router sont établies au moyen d'une connexion DECnet. Les liaisons de Message Router à User Agent peuvent être faites de deux manières : soit par une connexion DECnet soit par un programme d'application utilisant un interface de communication avec le Message Router.

Si on suppose que le Message Transfer System DECnet de la figure II.9 ne contient qu'un seul Message Router, et que celui-ci est associé au Message Router X.400 Gateway, le Message Handling System DECnet ne contiendra qu'un seul agent de transfert de message qui agira comme un MTA du Message Handling System X.400. Cette situation est celle existant actuellement à l'IIHE où le VAX/VMS ne communique avec aucun autre système sur le réseau DECnet, mais par contre est un MTA de la messagerie X.400.

2.3.5.1.2 Bases de données MR

La figure II.10 présente les bases de données de MR et les situent dans leur environnement.

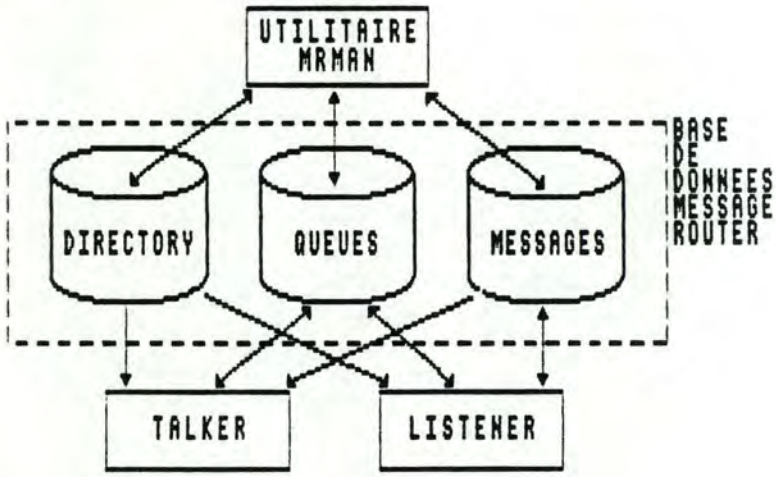


Fig. II.10 : Bases de données MR.

Le Message Router utilise les bases de données suivantes :

- une base de données (**DIRECTORY**) qui contient tous les renseignements nécessaires à la définition d'un noeud MR :

  - les abonnés au système de messagerie (nom d'utilisateur, nom de mailbox, mot de passe et caractéristiques personnelles)

    Exemple 2.2 : abonné au Message Router.
    FDUPONT /owner=FDUPONT /default /beep /password=SECRET

    Ceci signifie que quand cette personne voudra se connecter au Message Router à partir de son compte <FDUPONT> sur le système, elle ne devra pas spécifier de nom de mailbox, celle-ci étant définie comme la mailbox associée par défaut à ce compte (**default**). Par contre, si quelqu'un veut accéder à cette mailbox, à partir d'un autre compte, cette personne devra en donner le nom (FDUPONT) et le mot de passe (SECRET) pour que le Message Router lui accorde l'accès. L'option **beep** indique au Message Router qu'il devra avertir l'abonné de l'arrivée d'un message dans cette mailbox en affichant un avertissement sur son terminal (New mail for FDUPONT from ...).

- la définition des noeuds connus du noeud local

  Exemple 2.3 : noeud à distance.
  NODEi /owner=MRMANAGER /network_node /talker=...

  NODEi est une mailbox qui est utilisée pour réaliser l'échange de messages entre le noeud local et le noeud NODEi. Le propriétaire de cette mailbox est MRMANAGER, c'est à dire le compte utilisé par le Message Router. La clause **talker=** spécifie comment doit démarrer le programme de communication entre noeuds. Les différentes manières de d'exécuter ce programme sont définies au paragraphe 2.3.5.1.3.

- des informations de routage dans le réseau MR local et/ou à distance

  Exemple 2.4 : information de routage.
  NODEx /route= @ NODEi @ NODEj

  NODEx est accessible en routant les messages d'abord vers NODEj ensuite à NODEi qui peut réaliser une liaison directe avec NODEx. Les messages destinés à NODEj seront placés dans la mailbox NODEj d'où ils seront envoyés.

- une base de données de messages (**MESSAGES**) qui contient tous les messages envoyés ou reçus sur ce noeud.

- une base de données de pointeurs (**QUEUES**) qui lient un message particulier à son destinataire ou envoyeur.

2.3.5.1.3 Outils et Primitives MR

Ce paragraphe présente d'abord la structure interne du Message Router et en explique ensuite son fonctionnement par un exemple. La figure II.11 montre les composants logiques du Message Router. Nous verrons au paragraphe suivant comment le Message Router X.400 Gateway vient se greffer dans cette structure.

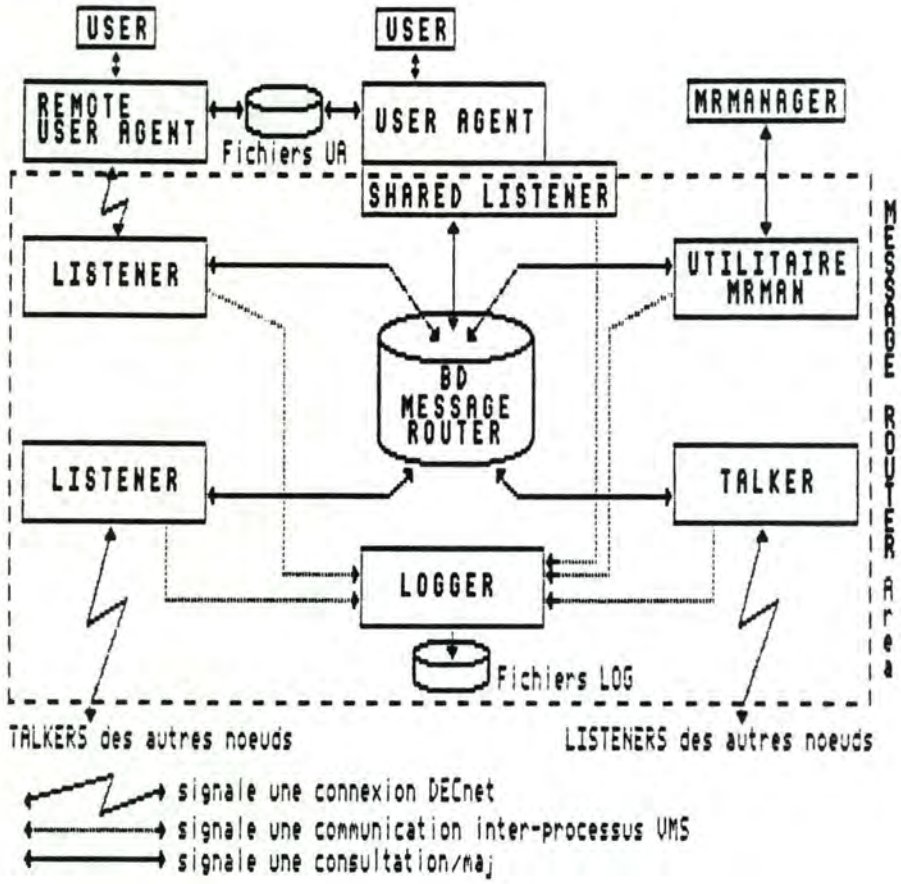


Fig. II.11 : Composants du Message Router.

Un **LISTENER** est un module pouvant être partagé ou accessible sur DECnet. Le Listener partagé est intégré au User Agent pour poster et rechercher des messages. Le Listener DECnet est connecté via une liaison DECnet à un User Agent éloigné pour qui il envoie ou recherche des messages. Le Listener DECnet est également chargé de recevoir les messages en provenance des autres Message Routers. Le Listener utilise la base de données DIRECTORY pour résoudre les adresses quand il reçoit un message. Pour le Listener, le destinataire d'un message est soit un abonné local, soit un noeud éloigné. A chaque destinataire est associée une mailbox. Quand le Listener reçoit un message d'un utilisateur ou d'un autre noeud, il l'ajoute à la base de données Messages et ajoute un pointeur vers ce message sur la file d'attente associée à chacune des mailboxes destinatrices du message. Si le Listener ne peut délivrer le message, il ajoutera un pointeur à la file d'attente de la mailbox de l'opérateur. Quand le message a été lu par un User Agent, le Listener retire le pointeur.

Le **TALKER** est responsable du transfert des messages aux Message Routers sur des autres noeuds. Il transfère les messages dont le pointeur est associé à un noeud éloigné. Une fois ces messages transférés, il efface

leur pointeur de la base de données QUEUES.

L'utilitaire **MRMAN** permet de configurer et de maintenir la base de données du Message Router. Il permet d'ajouter, de détruire et de modifier des informations relatives aux abonnés locaux et aux noeuds éloignés. Il permet également de détruire un message manuellement en générant un message de service. L'utilitaire MRMAN accepte également des commandes de gestion se trouvant dans des messages envoyés par d'autres Message Router à une mailbox spéciale (MRMAN).

Pour pouvoir envoyer ou recevoir des messages, un abonné du système de messagerie doit être connu du Message Router, autrement dit, la Directory doit contenir une entrée correspondant à l'abonné. Nous avons vu que l'utilitaire MRMAN permet de définir un abonné. L'exemple 2.5 illustre cette opération.

Exemple 2.5 : Definition d'un abonné.

Soit l'utilisateur "Frédéric DUPONT" ayant comme Userid <FDUPONT>.
Sa mailbox sera définie de la façon suivante dans la Directory :

```
add FDUPONT /owner=FDUPONT /default /beep /password=SECRET
```

Le fait d'utiliser le nom du Userid comme nom de mailbox est une convention qui permet de garder une cohérence dans l'identification d'un utilisateur sur le divers composants du système. On aurait pu appeler la mailbox "FREDERIC DUPONT" ou plus simplement "DUPONT".

Les utilisateurs construisent leurs messages à l'aide d'un User Agent. Le User Agent utilise les routines d'interface fournies par le Message Router. Les messages sont encodés selon la méthode décrite dans la spécification du format des messages produite par le United States National Bureau of Standard (NBS). La sémantique utilisée est proche de celles des messages X.400, mais la manière de coder cette sémantique est différente de celle décrite dans la recommandation X.409 du CCITT. Le lecteur intéressé peut néanmoins trouver en annexe un exemple de message codé suivant chacune des méthodes.

Une fois le message construit, le User Agent doit sélectionner une mailbox pour entrer en communication avec le Message Router.

Une mailbox peut être sélectionnée de trois manières :

- par défaut, par un utilisateur local ne spécifiant pas de nom,
- localement, par un utilisateur local qui spécifie un nom de mailbox et un mot de passe,
- par un utilisateur à distance qui n'a pas de compte sur le noeud local et qui doit spécifier un nom de mailbox et le mot de passe de cette mailbox.

Une fois l'identification réussie, le User Agent passe le message au Message Router sous forme de fichier. Par le module Listener, le Message Router reçoit ce message et en accepte la responsabilité. Il lui assigne un code d'identification unique et lui ajoute la date et l'heure de remise du message. Le code d'identification d'un message est de la forme **(ccssmmhhJJMMAAAA/serial_nr@node_name)**. (ccssmmhhJJMMAAAA) est la date et l'heure de soumission du message au Message Router (écrit à l'envers pour une raison inexpliquée), <serial_nr> est le numéro de série du message pour le Message Router et <node_name> est le nom du

noeud où a été posté le message. On dit alors que le message est marqué et estampillé. Le message est rangé dans la base de données Messages et un pointeur associe ce message à la mailbox qui a servi pour le transfert du message du User Agent vers le Message Router.

Le Listener va alors tenter de délivrer le message. S'il constate que le message est destiné à un abonné local, il va retirer le pointeur associant la mailbox d'origine au message, et va en ajouter un à la file d'attente de la mailbox réceptrice. Le propriétaire de cette mailbox sera averti de l'arrivée de courrier par l'affichage d'un message sur son terminal (clause BEEP de l'exemple 2.2). Le Message Router attend que le User Agent de l'utilisateur vienne chercher le message, c'est à dire que le pointeur reste associé au message jusqu'à ce qu'il soit lu. Dans ce cas, le module Listener transmet le message au User Agent et, une fois le transfert terminé, il enléve le pointeur de la file d'attente. A partir de ce moment la responsabilité du message incombe au User Agent.

Si le message est destiné à un abonné se trouvant sur un autre noeud, alors le Listener associe le message à une mailbox correspondant à un noeud éloigné. Chaque entrée de noeud éloigné indique comment démarrer le Talker. Il y a quatre façons de démarrer le Talker :
- soumission comme un job batch ordonnancé,
- soumission comme un job batch programmé,
- exécution par Talker permanent,
- exécution interactive.

On démarre le Talker comme un job batch ordonnancé si la liaison DECnet vers le noeud concerné n'est disponible que sur un certain intervalle de temps par jour ou si l'on veut contrôler le moment ou le Message Router utilisera le CPU et les ressources du réseau, par exemple on programmera le démarrage du Talker la nuit, à un moment ou le CPU et les ressources seront peu utilisées. La procédure MRTALK_SCH.COM (SCHeduled) permet de démarrer le Talker comme job batch ordonnancé.

Exemple 2.6 : Soumission en batch ordonnancé.
```
NODE1 /owner=MRMANAGER /network_node /run=MRTALK_SCH.COM
```

La méthode consistant à exécuter le Talker comme un job batch programmé ne permet pas de contrôler efficacement le trafic sur le réseau, mais a l'avantage de relayer les messages dès leur arrivée au Message Router. On utilisera cette méthode quand le trafic de messages est bas, si l'on désire un transfert rapide des messages ou si la liaison est disponible à temps plein.

Exemple 2.7 : Soumission en batch programmé.
```
NODE2 /owner=MRMANAGER /network_node /run=MRTALK_RUN.COM
```

Les procédures MRTALK_SCH.COM et MRTALK_RUN.COM s'exécutent comme des jobs batchs. Elles examinent chaque file d'attente des noeuds de la base de données auxquelles elles sont associées (clauses RUN=). Si il y a un message en attente, le Talker indiqué établit une connexion DECnet vers le noeud et transfère le message.

Si le trafic des messages est très intense, on passera les messages au moyen d'une mailbox VMS à un Talker tournant en permanence sur le système. Au démarrage du système, le talker permanent est mis en route par la procédure MRTALK_NOT.COM (NOTify). Le talker crée une mailbox VMS et attend d'être averti de l'existence d'un message à transmettre dans la file d'attente d'un noeud servi par ce Talker.

Exemple 2.8 : Exécution par Talker permanent.
NODE3 /owner=MRMANAGER /network_node /notify=MRTALK_MBX

La mailbox VMS MRTALK_MBX sert de moyen de communication entre les Listeners et le Talker du Message Router sur le noeud.

Il peut arriver qu'il soit impossible de prédire quand le Message Router sera capable de contacter un noeud donné. Dans ce cas, quand la liaison est disponible, on exécute interactivement le démarrage du Talker en entrant la commande de l'exemple 2.9.

Exemple 2.9 : Exécution interactive.
$ submit MRTALK_SCH.COM/queue=SYS$BATCH

Le Talker utilise les données de routage de la Directory pour atteindre le noeud cible. Une route se composant d'au moins un noeud, le Talker transfère le message vers le premier noeud indiqué dans la route, qui est en fait le dernier de la liste de routage. Une fois le message envoyé avec succès, le Talker effacera le pointeur associant le message à la mailbox du noeud éloigné. Les Message Routers des différents noeuds relaieront le message à travers le réseau jusqu'à sa destination.

Le **LOGGER** enregistre toutes les informations relatives aux événements survenant dans l'environnement du Message Router. Le Logger classe ces informations dans des fichiers (Log Files) selon le type d'événements.

1 - Le fichier des **transactions** (exemple 2.10) contient tous les renseignements concernant la progression d'un message à travers les composants du Message Router.

Le format des informations contenues dans le fichier des transactions du Message Router se compose la date d'enregistrement de l'événement (sous la forme AAAAMMJJhhmmsscc), d'un code de transaction en une lettre suivi du numéro de série du message concerné et d'un complément. Ce complément est selon le type de transaction soit le nom d'une mailbox, soit le numéro d'identification (ccmmhhJJMMAAAA/serial_number@node_name) suivi d'une longueur de message, soit de rien.

Exemple 2.10 : Fichier des transactions du Message Router.

Les mailbox utilisées dans cet exemple sont définies comme suit dans la Directory du Message Router du noeud VENUS :

```
MELOY     /owner=MELOY     /default /beep
DREUVIAUX /owner=DREUVIAUX /default /beep
```

1988010615401400, S79, DREUVIAUX
La première transaction indique que le Message Router a commencé à recevoir un message (Start) le 6 janvier 1988 à 15h40 et 14sec posté dans la mailbox DREUVIAUX. Le message porte le numéro de série 79.

1988010615401400, E79, 1341045160108891/41@VENUS, 29
La deuxième transaction marque l'acceptation du message par le Message Router (End). L'identification est '1341045160108891/41@VENUS', ce qui signifie que ce message a été accepté par le Message Router sur le noeud VENUS, le 6 janvier 1988 à 15h40 14 sec et 31 centièmes et que le numéro de série de ce message sur ce noeud était le 41. La longueur du contenu du message est de 29 bytes.

1988010615401500, D79, MELOY
Le Message Router indique ensuite qu'il a livré (Delivery) le message 79 à la mailbox MELOY, destinataire du message.

1988010615504100, F79, MELOY
A 15h50, le message 79 a été lu par un User Agent (Fetch) dans la mailbox MELOY.

1988010616023400, S80, MELOY
A 16h02, le Message Router commence à recevoir un message dans la mailbox MELOY.

1988010616023400, E80, 1843206160108891/80@VENUS, 38
Le Message Router accepte le message, le copie dans le fichier MR80.NBS (80 est son numéro dans la BD 'MESSAGES' du MR) et lui attribue un numéro d'identification.

1988010616023400, D80, DREUVIAUX
Le message est ensuite délivré à la mailbox DREUVIAUX.

1988010616203800, F80, DREUVIAUX
Le destinataire du message le lit dans la mailbox DREUVIAUX.

1988010712191500, K79
1988010712191500, K80
Le lendemain, les messages 79 et 80 ayant été lus par leur destinataire, le Message Router décide de les effacer (Kill).

2 - Le fichier des **statistiques du trafic** des messages produit un résumé des informations sur le trafic contenues dans le fichier des transactions. Les statistiques couvrent l'ensemble du Message Router et chacune des mailboxes.

Les informations statistiques indiquent le nombre de messages émis et reçus par le Message Router, ceux qui ont été lus et ceux pour lesquels une erreur s'est produite (expiration, disparition),

3 - le fichier des **erreurs** enregistre tous les messages d'erreur générés par tous les composants du Message Router,

4 - le fichier **journal de l'utilitaire MRMAN** rapporte toutes les commandes exécutées par MRMAN et qui ont provoqué un changement dans la base de données ou un message d'erreur.

En plus de ces composants, le Message Router offre d'autres procèdures n'apparaissant pas sur le schèma de la figure II.11.

- la procédure MRMWRK.COM qui est chargée d'exécuter les procédures de commandes venant de noeuds éloignés qui se trouvent dans la mailbox MRMAN.
- la procédure de maintenance automatique MRTIDY.COM qui s'exécute chaque jour et assure la gestion de la base de données. C'est elle qui effacera les messages auxquels n'est plus associé de pointeur et ceux qui n'ont pas été lus après un temps défini par le gestionnaire du système.
- la procédure MRCOMPRESS.COM dont le rôle est de réorganiser les fichiers DIRECTORY et QUEUES.

Exemple 2.11 : Illustration du fonctionnement du Message Router

Soit la configuration MR (Fig II.12) suivante :

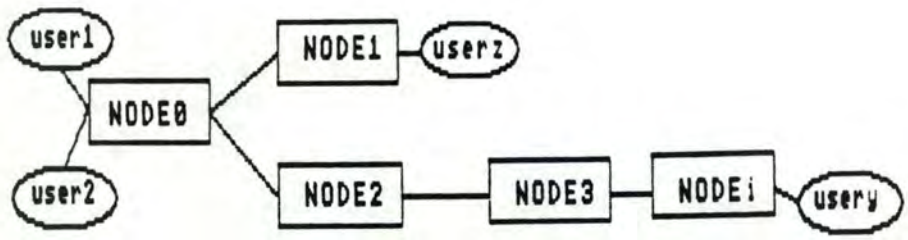
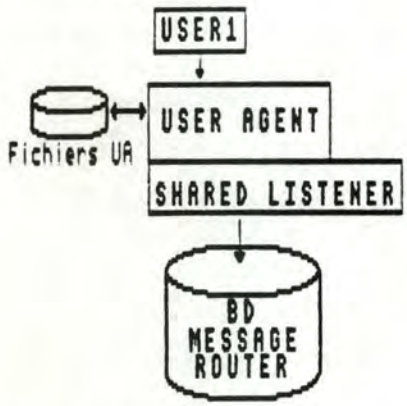


Fig. II.12 : Configuration de Message Routers.

Considérons le noeud NODE0 de la figure II.12.
Soient D, la base de données Directory,
Q, la base de données Queues,
M, la base de données Messages,
Q,M vides (pas de message en attente)

Situation initiale :
D = USER1 /owner = USER1 /default /beep
USER2 /owner = USER2 /default /beep
NODE1 /network_node /owner = MRMANAGER /talker = MRTALK_SCH.COM
NODE2 /network_node /owner = MRMANAGER /talker = MRTALK_RUN.COM
NODEi /route = @ NODE3 @ NODE2

Evénement 1 :

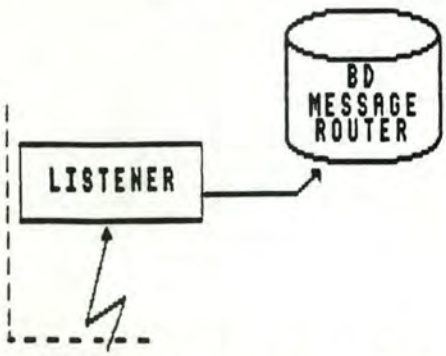


USER1 envoie un message à USER2.

Le Listener reçoit le message et lui attribue le numéro M1 :
Q = (USER1, M1)
M = M1.

Le Listener consulte D et y trouve USER1, il retire M1 de la mailbox USER1 et le place dans celle de USER2.
Nouvelle situation :
Q = (USER2, M1)
M = M1.

Evénement 2 :

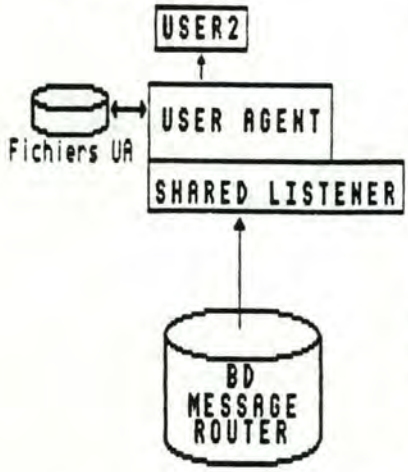


Le Listener reçoit un message M2 d'un autre Message Router pour le destinataire USERz sur le Noeud NODE1.

Il consulte D, y trouve NODE1 et place donc M2 dans la mailbox NODE1.

Nouvelle situation :
Q = (USER2, M1) & (NODE1, M2)
M = M1 & M2.

Evénement 3 :

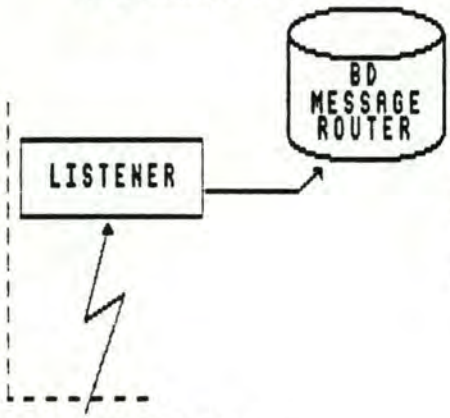


Le User Agent de USER2 consulte sa mailbox et lit le message qui s'y trouve.
Le Listener passe le message au User Agent et le retire de la mailbox.

Nouvelle situation :
Q = (NODE1, M2)
M = M2.

Le message n'est pas physiquement détruit. Le Listener ne fait qu'enlever le pointeur l'associant à la mailbox. Il n'est plus accessible par un User Agent, donc n'existe plus logiquement. Il sera effacé par la procédure de gestion automatique des bases de données MRTIDY.

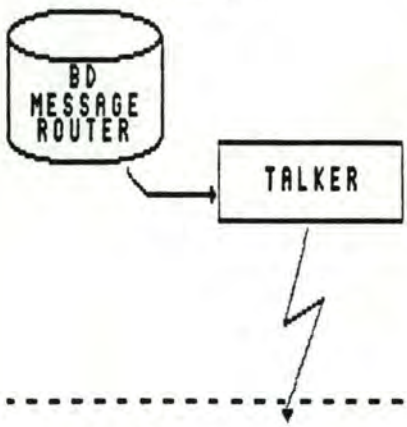
Evénement 4 :



Le Listener reçoit un message M3 d'un autre Message Router pour le destinataire USERy sur le noeud NODEi.
Il consulte D et constate que NODEi est présent. NODEi est accessible par une route passant par NODE2 et NODE3. Le Listener va placer le message dans la mailbox NODE2.

Nouvelle situation :
Q = (NODE1, M2) & (NODE2, M3)
M = M2 & M3.

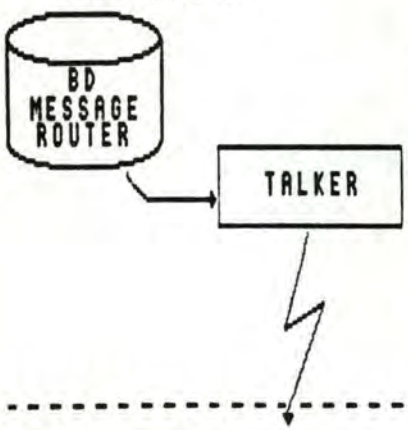
Evénement 5 :



La réception du message M3 par la mailbox NODE2 a provoqué la mise en route immédiate du Talker MRTALK_RUN.COM. Celui-ci consulte D et constate qu'un message est en attente pour lui dans la mailbox NODE2. Il établit une connexion DECnet avec NODE2 et commence la transmission du message. Une fois la transmission effectuée, il retire le message de la mailbox NODE2.

Nouvelle situation :
Q = (NODE1, M2)
M = M2.

Evénement 6 :



Le Talker MRTALK_SCH.COM arrive à son heure d'exécution. Il passe en revue toutes les mailboxes de D dépendantes de lui et constate qu'un message est présent dans la mailbox NODE1.
Il établit une connexion DECnet avec le noeud NODE1 et commence la transmission du message. Une fois celle-ci effectuée, il retire le message de la mailbox NODE1.

Nouvelle situation :
Q,M vides.

2.3.5.2 Le Message Router X.400 Gateway (MRX)

2.3.5.2.1 Description

Le Message Router X.400 Gateway est une passerelle entre le réseau DECnet et le réseau ISO pour la messagerie interpersonnelle de type store-and-forward.

Le Message Router X.400 Gateway permet la traduction de messages du format interne propre à Digital (NBS) en un format compatible avec un réseau ISO (X.409) et vice versa. MRX fait également la correspondance entre mailbox MR et O/R names en utilisant une base de données qui contient les données relatives aux abonnés du domaine local et aux autres domaines du service de transfert de messages.

MRX peut être utilisé indifférement par un domaine public ou par un domaine privé. Un domaine privé est propre à une organisation. Bien que les communications à l'intérieur d'un domaine privé ne doivent pas nécessairement être conformes aux recommandations X.400, il est cependant nécessaire de disposer d'un interface compatible X.400 pour communiquer avec d'autres domaines publics ou privés.

A chaque domaine externe correspond une mailbox MR. C'est par ces mailbox que les abonnés du Message Router échangeront des messages avec des domaines situés sur le réseau ISO. Un domaine avec lequel on ne sait pas établir de connexion directe peut être atteint en adressant le courrier qui lui est destiné à la mailbox correspondant à l'administration (domaine public) à laquelle le noeud local appartient.

2.3.5.2.2 Bases de données MRX

MRX utilise les bases de données suivantes :

1 - une base de données de **paramètres de contrôle** (FIG II.12). Les valeurs contenues dans cette base de données sont crées pendant la procédure d'installation du logiciel.

Admin Domain	*RTT*	*Operator Mailbox*	*OPERATOR*
Country Name	*BE*	*Per Recipient Flag*	*00*
Dialogue_Mode	*MONOLOGUE*	*Private Domain*	*FNDP*
Disconnect Timer	*1*	*Retry Timer*	*15*
GMT Time Offset (Min)	*-60*	*Service Class*	*CLASS_4*
Hopcount Limit	*10*	*Session limit*	*10*
Logfile Size (Blocks)	*80*	*Transport Address*	*FNDP_VENUS*
Maximun Data Size (Kb)	*10*	*TTX Out Flag (Hex)*	*00*
MRnet Time Offset (Min)	*0*	*VMSmailbox Name*	*MRX$NOTIFY*
MR Poll Timer	*30*	*Window Size*	*19*
Number of Retries	*40*		

CURRENT Checkpoint file : D18$:MRMANAGER.MRX$MANAGERCHECKPOINT.DAT;

FIG II.13 : Paramètres de contrôle MRX.

La figure II.13 montre comment est définie la base de données des paramètres de contrôle pour le domaine privé **FNDP**.

ADMIN DOMAIN identifie l'administration à laquelle le domaine privé appartient. Tous les messages dont le destinataire ne comprend pas de domaine public sont supposés devoir être adressés via cette administration.
CHECKPOINT FILE NAME est le nom du fichier utilisé par MRX pour enregistrer les informations sur les messages actuellement en cours de transmission. Ce fichier contient toutes les informations nécessaires pour reprendre des connexions et des messages partiellement interrompus, MRX s'étant arrété pour une raison quelconque.
COUNTRY NAME est le code identifiant le pays dans lequel le domaine est situé. Tous les messages dont le destinataire ne comprendrait pas de nom de pays sont supposé être adressé à un domaine (public ou privé) situé dans ce pays.
DIALOGUE MODE indique à MRX quel mode d'exploitation de la ligne il doit utiliser lors de l'expédition d'un message. Le mode d'exploitation peut être unidirectionnel (Monologue) ou bidirectionnel à l'alternat (Two_Way_Alternate).
DISCONNECT TIMER dit à MRX l'intervalle de temps qu'il doit attendre avant de fermer une connexion inactive, au cas où la connexion pourrait être utilisée pour envoyer ou recevoir un autre message.
GMT TIME OFFSET est la différence horaire en minutes, entre l'heure de Greenwich (Greenwich Meridian Time) et le domaine MRX.
HOPCOUNT LIMIT est le nombre maximum de fois qu'un message peut être relayé avant d'être déclaré en train de cycler dans le réseau, et provoquer ainsi la génération d'une notification de non-livraison.
LOGFILE SIZE est la taille en blocs VMS du fichier de rapport d'événements.
MAXIMUM DATA SIZE est la quantité maximale de données (en Kbytes) pouvant être transmise avant d'émettre un point de contrôle. Ce paramètre et la taille de la fenêtre (WINDOW SIZE) déterminent la fréquence d'émission de points de contrôle dans l'optique d'un recouvrement d'erreur.
MRNET TIME OFFSET est la différence en minutes, entre le temps "officiel" du réseau de Message Routers et l'endroit où est située la passerelle.
MR POLL TIMER indique à MRX quand il doit vérifier l'existence de messages en file d'attente dans les mailboxes MR qui lui sont attachées. Ceci est un mécanisme d'exception car, quand un message arrive à une mailbox attachée à MRX, le Message Router avertit MRX en utilisant une mailbox VMS. Ce paramètre assure cependant que MRX controlera les mailboxes MR régulièrement, au cas ou le message de la mailbox VMS serait manquant pour quelque raison que ce soit.
NUMBER OF RETRIES est le nombre de fois que MRX tentera d'établir une connexion avec une destination particulière sur le réseau ISO.
OPERATOR MAILBOX est le nom d'une mailbox MR qui sera utilisée pour recevoir le courrier ambigu ou mal adressé.
PER RECIPIENT FLAG est utilisé dans la génération de notifications de non-délivraison pour les messages mal formés.
PRIVATE DOMAIN identifie le domaine privé auquel appartient MRX.
RETRY TIMER est l'intervalle de temps pendant lequel MRX attend avant de tenter une nouvelle connexion à une destination sur le réseau ISO, si la tentative précédente a échoué.
SERVICE CLASS spécifie la classe de service transport à utiliser lors de l'échange de messages. La classe 0 est utilisée pour les communications de domaine public à domaine privé, la classe 4 est utilisée entre domaines privés.
SESSION LIMIT indique à MRX le nombre maximum de sessions pouvant être traitées en meme temps. Cette valeur dépend du nombre de canaux logiques définit au niveau réseau (VAX PSI).

TRANSPORT ADDRESS identifie MRX et l'associe à une destination de la couche transport. C'est ce paramètre qui déclare MRX comme un processus pouvant recevoir des requêtes de connexion transport. A la valeur introduite ici correpond une entrée dans la base de données TSAP de VOTS.
TTX OUT FLAG permet de transmettre des textes Teletext (0) ou IA5 (1) à partir d'un domaine Digital.
VMSmailbox NAME identifie la mailbox VMS que le Message Router utilise pour prévenir MRX que des messages sont en attente dans des mailboxes MR.
WINDOW SIZE indique à MRX combien de paquets de données il peut envoyer avant de devoir attendre une confirmation du partenaire éloigné.

2 - un fichier de **paramètres d'initialisation** (MRX$INIT.DAT) qui permet entre autres de définir un mot de passe pour les mailbox MR attachées à MRX, le nom du fichier d'enregistrement des événements, le nom du fichier contenant la base de données MRX, la taille des SPDU gérés par le noyau session de MRX.

3 - une base de données **d'abonnés à la messagerie** X.400 définissant les utilisateurs MR pouvant accéder aux services du Message Router X.400 Gateway. Elle permet de faire la traduction entre la mailbox d'un utilisateur et l'O/R name associé à celui-ci.

```
Message Router address      : FDUPONT

PERSONAL DETAILS            Surname          : DUPONT
                            Initials         : FD
                            Given name       : FREDERIC
                            Generation       : JUNIOR

ORGANIZATION DETAILS        Organization name : Inst. d'Informatique
                            Unit name         : UER Gestion
```

FIG II.14 : Abonné MRX.

La figure II.14 montre comment est défini un abonné à la messagerie X.400 dans la base de données MRX du domain privé FNDP. 'Message Router address' est le nom de la mailbox MR associée à cet abonné. C'est dans cette mailbox que seront rangés les messages que MRX recevra de l'extérieur et c'est via cette mailbox que l'abonné postera ses messages à destination d'autres domaines X.400. MRX supporte la forme d'O/R name mnémonique (forme 1.1) pour enregistrer les abonnés. Les noms de pays, de domaine public et de domaine privé sont ceux définis dans la base de données des paramètres de contrôle.

4 - une base de données de **domaines X.400** contenant les informations de routage sur les domaines privés ou publics avec lesquels on désire communiquer. La figure II.15 montre comment doit être défini le domaine privé IIHE dans la base de données du domaine FNDP.

```
Domain name               : IIHE
Network address           : .IIHE_BVX82.X25_DCS%22101680
Message Router address    : IIHE_MRX
External MTA name         : IIHE
Incoming password         :
Internal MTA name         : FNDP
Outgoing password         :
```

FIG II.15 : Domaine privé MRX.

MRX utilise les renseignements contenus dans cette base de données pour pouvoir établir des connexions avec d'autres domaines. Le nom du domaine l'identifie dans la base de données. L'adresse réseau X.400 (Network address) permet de localiser le domaine dans le réseau ISO. Les formats d'adressage utilisé par DIGITAL sont expliqués au chapitre 3 "ADRESSAGE ET ROUTAGE".

La MESSAGE ROUTER ADDRESS est le nom de la mailbox MR à laquelle les abonnés expédieront les messages destinés à ce domaine et où MRX postera les messages qu'il recevra de ce domaine.

Les paramètres EXTERNAL et INTERNAL MTA NAMES doivent être définis par arrangement réciproque entre les deux domaines. **External MTA name** identifie le MTA du domaine externe auquel on désire se connecter, étant entendu qu'un domaine peut avoir plusieurs MTAs. C'est le nom par lequel MRX connait le MTA du domaine éloigné. **Internal MTA name** identifie le MTA du domaine local. C'est le nom par lequel le domaine externe reconnait le domaine local. Des mots de passe peuvent être également associés aux noms de MTA. Ces mots de passe sont vérifiés par MRX ou par un autre logiciel X.400 lors de l'établissement de la connexion.

5 - une base de données des **transmissions** qui contient les informations sur les connexions sessions X.400 en cours. Elle permet de vérifier le bon déroulement des opérations.

```
MRX SESSION INFORMATION      ID  : 12        21-JAN-1988 12:03:13.43

    Session Connection ID        : FNDP_VENUS880121110047Z12

    Creation Time                : 21-JAN-1988 12:00:47.22
    Last Connection Time         : 21-JAN-1988 12:00:53.25
    Connection Attempts          : 1
    Message Router Mailbox       : IIHE_MRX
    Window Size        (Neg/Def) : 19 / 19
    Checkpoint Size    (Neg/Def) : 4 / 4  Kbytes
    Number of retries            : 20
    Retry Time                   : 10
    Disconnect Time              : 1
    Attribute                    : Initiator
    State                        : Active Sender Idle

    Activity ID (Hex)            : 00000001
    Activity Started             : 21-JAN-1988 12:03:04.73
    Activity Count               : 1
    Checkpoint No.     (Conf/High) : 3 / 6
    Transmission (Current/Total) : 20480 / 152920    Bytes
    Message File Name            : MRX$:MRX$19.X409
```

FIG II.16 : Session X.400.

La figure II.16 montre l'état d'une connexion courante d'émission de message à partir du domaine privé "FNDP". Une connexion est identifiée dans la base de données par un numéro de série (ID). La connexion session utilisée pour l'échange de messages est identifiée en concaténant l'adresse transport du MTA émetteur (FNDP_VENUS) avec la date et l'heure GMT (AAMMJJhhmmss) de création de la session, suivi de "Z" et du numéro d'identification de la connexion (ID). La lettre Z (Zoulou) provient de

la notation aérienne ou militaire des fuseaux horaires et indique l'heure universelle (GMT) <LABO-87>.

MESSAGE ROUTER MAILBOX est la mailbox associée au domaine destinataire du message.

WINDOW SIZE indique le nombre de paquets de données qui peuvent être émis avant que MRX n'attende un accusé de réception. NEG est la valeur qui a été négociée à l'établissement de la connexion, MRX acceptant la plus petite valeur. DEF est la valeur par défaut définie dans la base de données des paramètres de contrôle.

CHECKPOINT SIZE est le montant de données émis d'une traite.

NUMBER OF RETRIES définit le nombre de tentatives que MRX effectuera pour établir une connexion. L'intervalle entre deux essais est défini par RETRY TIME.

DISCONNECT TIME indique le durée pendant laquelle MRX attendra après transmission d'un message, avant de cloturer la liaison, celle-ci pouvant servir pour un autre message.

ATTRIBUTE indique le type de session (Initiator ou Responder), le mode d'utilisation de la ligne (Monologue ou Two_Way_Alternate) ainsi que le type de protocole utilisé (P1).

STATE montre l'état de la connexion (Active ou Inactive). Quand une connexion est active, MRX montre quelle est l'opération de transmission actuellement en cours.

Lorsqu'une connexion est active, MRX donne en plus des renseignements sur l'identifiant X.400 de l'activité en cours (ACTIVITY ID) et sur le nombre de messages traités (ACTIVITY COUNT).

CHECKPOINT No. indique le nombre de paquets de données échangés et confirmés (CONF) et le nombre total de paquets à émettre. En mode réception, Conf est déterminé par la taille de la fenêtre.

TRANSMISSION est le nombre de bytes envoyés (Current) sur le total du message. En réception ces deux nombres sont égaux, étant entendu qu'il est impossible de savoir à l'avance le nombre de bytes qu'il faudra recevoir.

MESSAGE FILE NAME est le nom du fichier actuellement transmis.

Lors de l'envoi d'un message, les opérations reprises dans STATE sont successivement : la lecture du message à envoyer dans une mailbox MR (Fetching_msg), la traduction du format NBS vers le format X.409 (Encoding_msg), l'établissement d' une connexion avec le MTA destinataire (Connecting), l'envoi du message (Sending_msg), l'attente des accusés de réception (Sender Idle) et, une fois le message transmis, l'attente du temps voulu pour libérer la ligne (Sender Waiting_disconnection).

Lors de la réception d'un message, une fois la connexion acceptée, MRX signale qu'il attend le message (Receiver Waiting_for_msg), qu'il est en train de recevoir (Receiving_msg) ou qu'il traduit le message du format X.409 en NBS (Receiver Decoding_msg).

6 - Le fichier d'enregistrement d'événements sur MRX.

Le fichier des transactions MRX contient le détail de tous les échanges de messages réalisé par MRX. La première partie reprend la date et l'heure de l'événement, le composant de MRX impliqué et le code de l'événement survenu. La suite signale l'événement en clair.

Exemple 2.12 : Fichier des transactions de MRX.

6JAN88 15:39:43.95 OSAK %MRX-I-RTSCONACP,
FNDP_VENUS88010601211347Z12 new con. from .FNDP_VENUS.X25_DCS%28160000 is accepted
Le sous-ensemble session de MRX (OSAK) signale qu'il a accepté une connexion d'un domaine dont l'adresse est .FNDP_VENUS.X25_DCS%28160000. Le numéro d'identification de la session est celui qui a été attribué par le MTA émetteur.

6JAN88 15:39:44.49 OSAK %MRX-I-RTSOSIWAI,
FNDP_VENUS - Session waiting for incoming connections.
Ayant signalé à l'autre MTA qu'il avait accepté la connexion, la couche session de MRX attend l'arrivée de données.

6JAN88 15:39:46.56 0065 %MRX-I-RTSACTCRE,
FNDP_VENUS88010601211347Z12 activity 02010100000 creating X400 message file MRX$:MRX$11.X409
MRX signale qu'il reçoit un message X.400 et qu'il le range dans un fichier.

6JAN88 15:39:49.43 0065 %MRX-I-RTSACTRCV,
FNDP_VENUS88010601211347Z12 X400 message file MRX$:MRX$11.X409 created
MRX signale la fin de réception de données.

6JAN88 15:39:50.17 D001 %MRX-I-STAMSGTRN,
Started Message Translation
MRX commence à traduire le message reçu en format NBS. C'est ici qu'il vérifie si le destinataire du message lui est connu.

6JAN88 15:39:54.96 D001 %MRX-I-RTSMRPOST,
FNDP_VENUS88010601211347Z12 posted MRX$:MRX$11.X409 at MR-mailbox FNDP_MRX as 13735160108891/41@VENUS
MRX a pu identifier la mailbox MR du destinataire, il poste le message au Message Router en utilisant la mailbox FNDP_MRX pour s'identifier. FNDP_MRX est la mailbox associée au domaine qui a émis le message. Le numéro d'identification du message pour le Message Router est celui qui lui a été attribué par le Message Router émetteur.

6JAN88 15:40:56.31 0065 %MRX-I-RTSSESREL,
FNDP_VENUS88010601211347Z12 has been disconnected
MRX n'ayant plus rien à transmettre ou à recevoir, cloture la connexion.

6JAN88 16:02:38:60 E014 %MRX-I-RTSMRFETC,
has fetched from message router mailbox FNDP_MRX into file MRX$:MRX$19.NBS
MRX signale la réception d'un message dans la mailbox FNDP_MRX.

6JAN88 16:02:38:01 E014 %MRX-I-STAMSGTRN,
Started Message Translation
Le message est traduit du format NBS en format X.409. MRX vérifie ici que l'expéditeur du message existe dans sa base de données.

6JAN88 16:02:47.43 0012 %MRX-I-RTSNEWCON,
IIHE_BVX82880106150247Z12 to .FNDP_VENUS.X25_DCS%28160000,
connection attempt count 1/41
MRX démarre une session de connexion identifiée par son adresse transport (IIHE_BVX82) et l'heure GMT du début de la session, avec un MTA dont l'adresse est donnée. 1/41 indique le nombre de tentatives de connexions effectuées sur un nombre total d'essais.

6JAN88 16:02:49.13 0012 %MRX-W-RTSCONFAI,
IIHE_BVX82880106150247Z12 connection attempt failed for reasons given below :
MRX signale (-W- warning) que la tentative de connexion a échoué.

6JAN88 16:02:49.78 0012 -OSIS-E-NOCONNECT, failed to connect to target SAP
Une erreur s'est produite lors de l'établissement de la connexion session avec l'autre MTA.

6JAN88 16:02:50.23 0012 -SYSTEM-F-LINKDISCON,
network partner disconnected logical link
Le MTA a provoqué la rupture de la liaison, ce qui a provoqué une erreur fatale (-F-).

6JAN88 16:02:51.09 0012 %MRX-I-RTSRECONW,
IIHE_BVX82880106150247Z12 waiting 15 minutes before reconnection
MRX informe qu'il refera une tentative de connexion dans 15 minutes.

Cet exemple montre comment s'est effectué le premier échange d'un message X.400 entre les Facultés Notre-Dame de la Paix à Namur et l'IIHE à Bruxelles.

La tentative de connexion vers FNDP a échoué à cause d'un problème interne au noeud de Namur. En fait nous pouvions émettre des messages depuis Namur vers l'IIHE sans problème, par contre le VAX des FNDP refusait systématiquement toute connexion transport ou MRX venant de l'extérieur, mais acceptait d'ouvrir un circuit virtuel pour le VMS/Mail entre les deux machines. Le numéro DTE des Facultés ayant été mal introduit dans la base de données de PSI (il était de 2062816000 au lieu de 2816000), la présence du numéro international belge (206) causait un rejet de l'appel (de type national) à l'entrée du réseau X.25.

La couche réseau pouvait accepter une connexion réseau pour le PSI_MAIL, celui-ci utilisant un numéro d'objet DECnet dans son paquet d'appel, par contre les demandes de connexion transport étaient refusées par VOTS, car celui-ci ne contenait dans sa base de données aucun NS-Provider dont l'adresse réseau correspondait au numéro DTE spécifié dans le paquet d'appel.

2.3.5.2.3 Outils et Primitives MRX

Les différentes actions de gestion du MRX sont faites par l'intermédiaire de l'utilitaire MRXMAN qui permet de créer, de modifier, de supprimer des abonnés et des domaines et de vérifier le bon fonctionnement du Message Router X.400 Gateway.

De plus, un traçage (LOGGING) des événements et messages passant par le Message Router X.400 GATEWAY est réalisé automatiquement.

Comme nous l'avons dit plus haut au point 2.3.5, le Message Router X.400 Gateway contient un sous-ensemble de OSAK. Cette partie du logiciel session permet à MRX de gérer lui-meme les connexions sessions dont il a besoin pour transmettre et recevoir des messages X.400. Il est à noter que les services sessions de MRX ne sont pas disponibles aux autres utilisateurs, ceux-ci devant obligatoirement passer par le logiciel OSAK "complet".

2.3.5.2.4 Structure interne de MRX

La structure interne du Message Router X.400 Gateway peut être déduite de ce que nous venons de voir. Nous utilisons le terme "déduire" car cette structure ne se trouve pas de manière explicite dans la documentation fournie par Digital. La figure II.17 présente les divers composants de MRX. Pour établir cette architecture interne, nous nous sommes basés sur <LABO-87>, <MACCHI-87>, <VAX-MRX> et <X.410>.

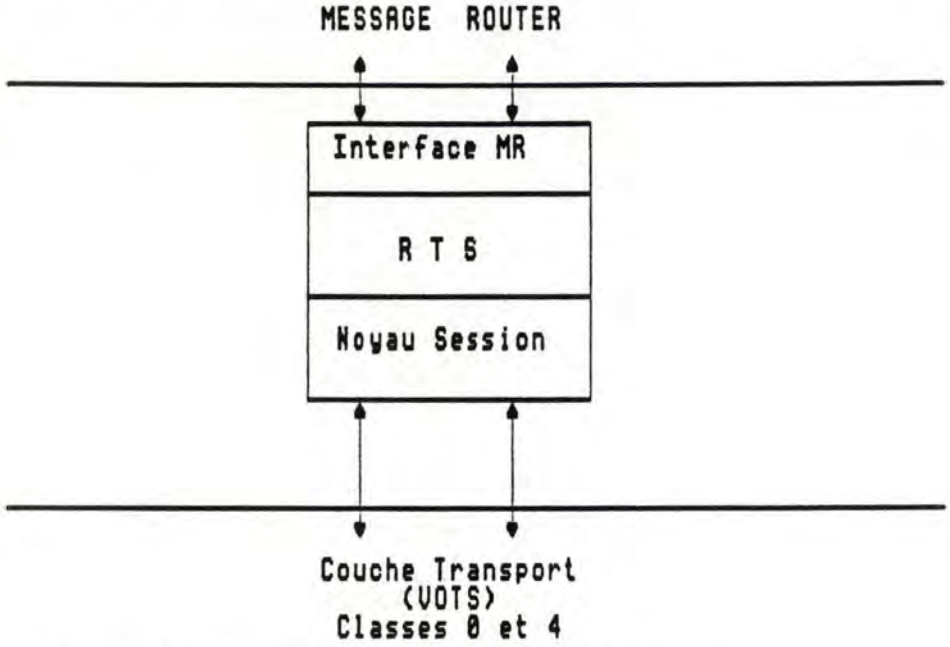


Fig. II.17 : Structure Interne de MRX.

Le Message Router X.400 Gateway est composé d'un **interface** avec le Message Router qui intéragit avec celui-ci pour réaliser l'échange des messages. C'est lui qui poste les messages destinés aux abonnés locaux et qui va lire les mailboxes MR attachées aux domaines avec lesquels MRX communique. L'interface utilise un service de présentation qui réalise la conversion des messages du format NBS en X.409 et inversément.

Le **Reliable Transfer Service** (RTS ou Serveur de Transfer Fiable) est responsable de la création et de la gestion de connexions entre deux MTA. Ces connexions sont appelées associations. Une association peut être unidirectionnelle (Monologue) ou bidirectionnelle à l'alternat (Two_Way_Alternate).

Le RTS est responsable des messages à transférer. C'est lui qui garantit qu'un message ne sera pas perdu ou duplicié. Le RTS assure la reprise des connexions session qui auraient été coupées en utilisant un service de reprise d'activité basé sur des points de synchronisation (voir figure II.16). Le RTS est chargé de la négociation des tailles de fenêtre et du moment d'établir des points de reprise. La recommandation X.410 prévoit que les RTS reprennent une session interrompue en ne transmettant que les SSDU perdus et non encore émis.

Le noyau session utilisé par le RTS de MRX est basé sur le logiciel OSAK, mais est inclus dans le Message Router X.400 Gateway. Le noyau session permet l'établissement de connexion avec négociation des paramètres du dialogue (taille de fenêtre, ...), offre un service d'aide à la négociation au RTS et de données utilisateur du RTS. Il permet une transmission duplex ou semi-duplex avec utilisation d'un jeton de

données. Le service de gestion d'activité fourni permet le transfert de données et la signalisation d'anomalies. Le service session permet également la pose de points de synchronisation au cours du dialogue, ainsi que la reprise de celui-ci à partir d'un point de synchronisation.

Lorsqu'il demande l'établissement d'une connexion session, le RTS spéciie comme paramètres : la taille de la fenêtre et des points de reprise (négociables), le mode de dialogue et le protocole d'application (P1, la version 1.1 de MRX ne supportant pas les abonnés P3), enfin le RTS fournit un identificateur de connexion session (Session Connection ID).

2.3.5.3 Intégration de MRX parmi les composants du MR

Ce paragraphe explique d'abord comment le Message Router X.400 Gateway vient s'intégrer parmi les composants logiques du Message Router décrits au paragraphe 2.3.5.1.3 'Outils et primitives' du Message Router et ensuite présente un exemple d'échange de messages entre Message Router et Message Router X.400 Gateway.

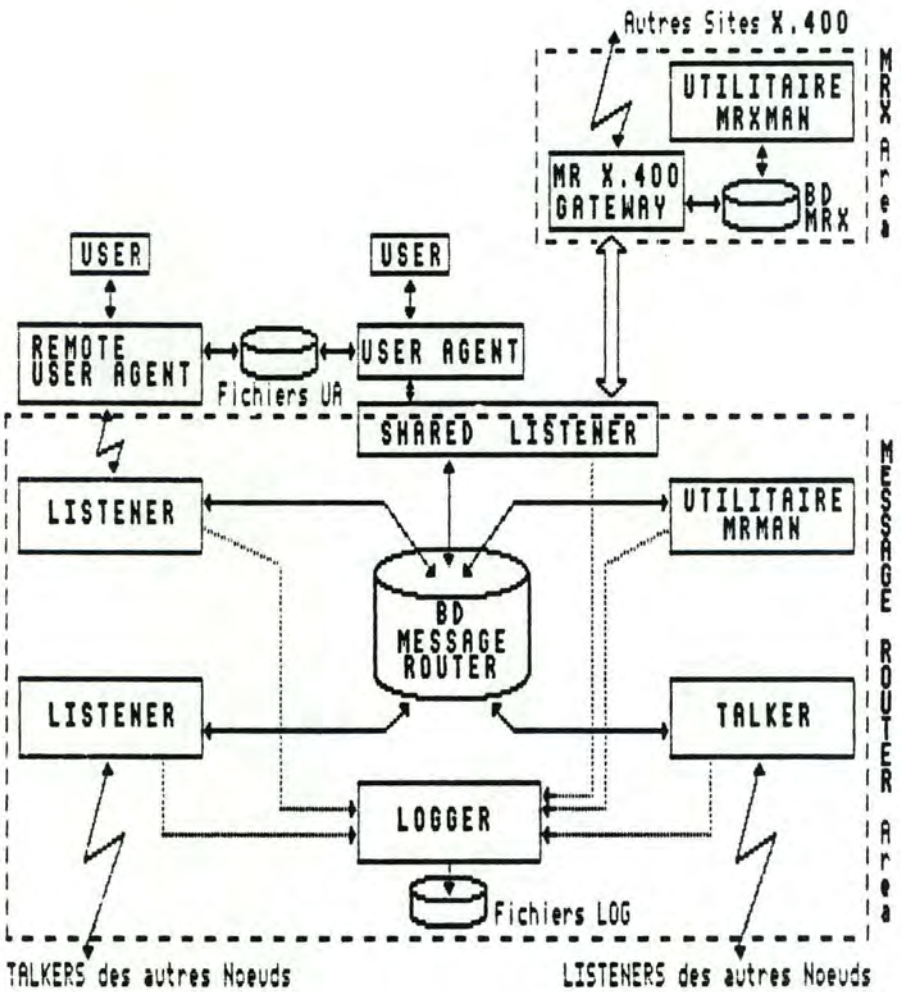


FIG II.18 : Intégration de MRX dans MR.

La figure II.18 montre comment se réalise l'intégration du Message Router X.400 Gateway parmi les composants logiques du Message Router. Il est

nécessaire d'expliquer comment les messages peuvent passer de MR vers MRX et inversément.

Nous avons vu à l'exemple 2.6 que dans la définition d'un domaine éloigné se trouvait le nom d'une mailbox MR (mradress). Voyons comment cette mailbox est définie dans la Directory.

Exemple 2.13 : Définition d'une mradress pour un domaine privé.
add FNDP_MRX /owner=MRMANAGER /system /notify=MRX$NOTIFY /password=""

La clause 'notify=MRX$NOTIFY' signifie que chaque fois qu'un message sera déposé dans la mailbox MR **FNDP_MRX**, la mailbox VMS MRX$NOTIFY avertira MRX de la présence d'un message qui lui est destiné.
Le mot de passe associé aux mailboxes servant de moyen de communication entre MR et MRX doit toujours être égal à la chaine vide, sinon MRX ne pourrait pas les utiliser. En effet, quand un message est déposé dans une mailbox, MRX en est averti par la mailbox VMS. Il va essayer d'aller lire le message en donnant comme mot de passe de cette mailbox la chaine vide. Si les mots de passe ne concordent pas, MRX ne sera pas autorisé à lire le message.

Le Message Router X.400 Gateway peut être vu comme un Talker tournant en permanence qui est averti de la présence d'un message à émettre par une mailbox VMS. Mais à la différence d'un Talker, MRX peut également recevoir des messages d'un autre domaine.

Lorsqu'il reçoit un message destiné à un abonné du noeud local, MRX vérifie l'existence du destinataire dans sa base de données en cherchant une entrée correspondant à l'O/R name contenu dans le message. Si celui-ci existe, alors il lui correspondra une mailbox MR. Dans ce cas MRX agira comme un User Agent : il s'identifiera au Message Router via la mailbox MR correspondant au domaine d'où provient le message et il y postera le message en lui donnant comme destination la mailbox correspondant à l'O/R name du destinataire. Le Listener recevra le message et le traitera de la même manière que celle indiquée au paragraphe précédent.

Exemple 2.14 : Exemple d'échanges de messages X.400

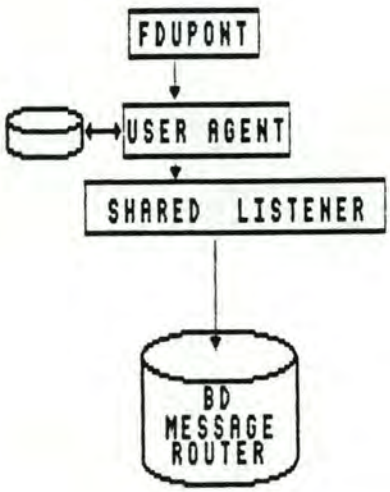
Reprenons la situation initiale décrite à l'exemple 2.11 et ajoutons à D les lignes :

```
IIHE_MRX /owner=MRMANAGER /system /notify=MRX$NOTIFY /password = ""
FDUPONT /owner=FDUPONT /default /beep
```

Le mot de passe d'une mailbox utilisée par MRX doit être égale à la chaine vide, sinon MRX ne pourra l'utiliser pour poster ou rechercher des messages.

Considérons un abonné et un domaine MRX tels qu'ils sont définis respectivement aux exemples 2.14 et 2.15.

Evénement 1 :



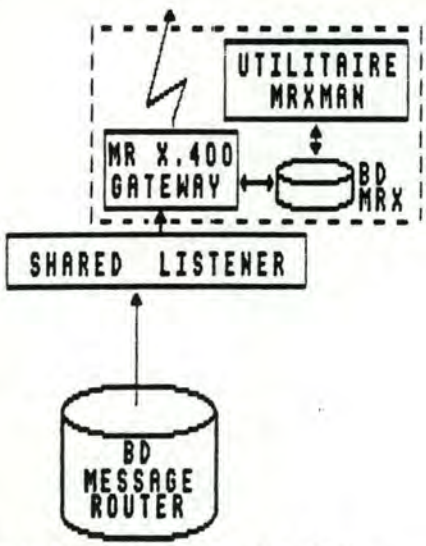
L'abonné FDUPONT désire envoyer un message à une autre personne (USERx) située sur le noeud FNDP et accessible par le réseau X.400. Lors de la construction de son message, FDUPONT décrira l'O/Rname de son destinataire dans les zones prévue à cet effet dans l'enveloppe, mais il devra EN PLUS spécifier le nom de la mailbox MR où son message pourra être pris en charge par MRX. Pour le domaine FNDP, la mailbox MR est IIHE_MRX. Pour le Message Router, ce message sera envoyé à la mailbox IIHE_MRX. Comme décrit précédemment à l'exemple 2.10, le User Agent de l'utilisateur passera le message au Listener qui, dans un premier temps, le rangera dans la mailbox émettrice.

Nouvelle situation : Q = (FDUPONT, M1)
M = M1.

Le Listener voit que le message est destiné à la mailbox IIHE_MRX, il retire donc le message de FDUPONT et le place dans IIHE_MRX.

Nouvelle situation : Q = (IIHE_MRX, M1)
M = M1.

Evénement 2 :



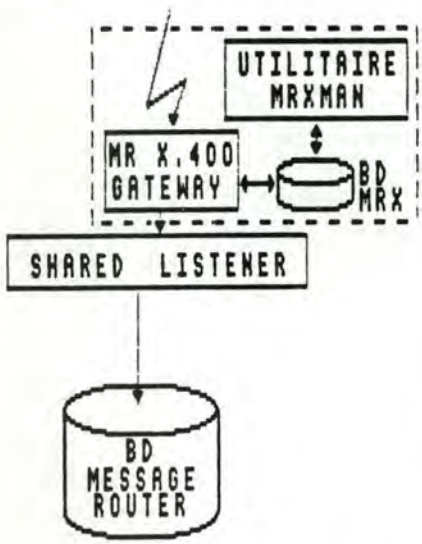
Averti par la mailbox VMS de la présence d'un message dans la mailbox IIHE_MRX, le Message Router X.400 Gateway va lire ce message.

Pour le Listener le message a été délivré, donc il peut l'effacer de la mailbox IIHE_MRX.

Nouvelle situation : Q et M vides.

A partir de ce moment, le message est dans le monde X.400 : MRX va traduire le message du format NBS en format X.409. Il va vérifier que l'expéditeur est autorisé a utiliser ses services. Ceci s'effectue en vérifiant si la mailbox d'origine de ce message se trouve sous forme de Mradress associée à un abonné MRX. Si tous les contrôles sont corrects, MRX ajoute à l'enveloppe du message les renseignements sur l'expéditeur qu'il contient dans sa base de données selon le format d'O/R name mnémonique (forme 1.1) et envoie le message au domaine dont l'adresse sur le réseau X.400 est dans l'entrée MRX correspondante à la mailbox de destination.

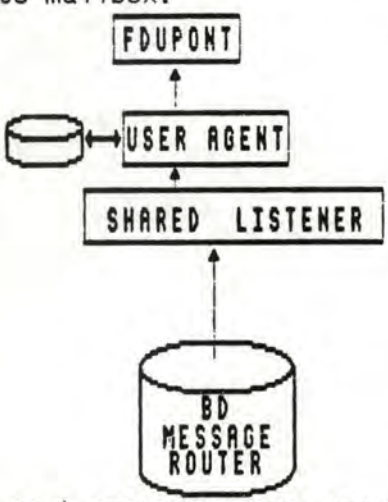
Evénement 3 :



MRX a recu un message M2 venant du domaine FNDP pour Frédéric Dupont.
MRX va vérifier si le destinataire du message est présent dans la base de données des abonnés MRX. Il traduit ensuite le contenu du message en NBS et le fait passer au Message Router. Il va utiliser la mailbox correspondant au domaine d'où provient le message pour poster le message au Message Router et indiquera que la mailbox destination est celle correspondant à l'abonné Frédéric Dupont, c'est à dire FDUPONT.

Nouvelle situation : Q = (IIHE_MRX, M2)
M = M2.

Pour le Message Router, la mailbox IIHE_MRX est utilisée comme origine du message et non comme destination. La clause notify = MRX$NOTIFY n'intervient pas dans ce cas, car sinon le message serait renvoyé vers MRX ! Le Listener va agir comme décrit dans l'exemple 2.10 : il va voir que le message est destiné à la mailbox FDUPONT et va le placer dans cette mailbox.



Nouvelle situation : Q = (FDUPONT, M2)
M = M2.

L'utilisateur FDUPONT sera averti de l'arrivée d'un message dans sa mailbox par l'affichage d'un message à son terminal : "New mail from IIHE_MRX for FDUPONT". Le Message Router gardera le message jusqu'à ce que son destinataire vienne le rechercher.

Situation finale : Q et M vides

On peut remarquer que, pour l'utilisateur, le passage d'O/R name à mailbox MR n'est pas transparent. L'abonné MRX doit connaitre le mécanisme de transfert de message, il doit savoir que le courrier destiné à un domaine X.400 doit d'abord passer par une mailbox MR locale. Ceci peut préter à confusion, car si un expéditeur demande une confirmation de livraison, il en recevra deux ! Une confirmation sera générée par le Message Router quand MRX viendra lire le message dans la mailbox du domaine destination, et l'autre sera envoyée par le User Agent du

correspondant. Il est donc important de disposer d'un User Agent "intelligent" permettant, au moyen d'une gestion de base d'une base de données des noeuds X.400 et de leur mailbox MR, de rendre le transfert de message transparent à l'utilisateur.

2.3.5.4 Taches à remplir par un Message Router Manager

Un Message Router Manager ou MRMANAGER est la personne responsable de la gestion et de la maintenance des logiciels Message Router et Message Router X.400 Gateway. Une partie importante de notre travail a été de faire la maintenance quotidienne de l'ensemble MR/MRX. Nous décrivons ci-après les différentes taches qu'un MRMANAGER doit remplir. Ces taches peuvent être mises en parallèle avec celles décrites dans <BOVE-87>, <HEAG-87> et <HENK-87>.

- Concevoir la configuration du réseau MR, en tenant compte du nombre de noeuds sur le système et du taux de messages prévu,

- Décider du mécanisme de routage à utiliser sur le système,

- Installer le système,

- Créer la base de données en utilisant l'utilitaire MRMAN,

- Purger ou compresser périodiquement les fichiers,

- Mettre à jour la base de données,

- Etablir le diagnostic des problèmes et interpréter les fichiers d'informations,

- Créer des entrées MRX pour chaque domaine avec lequel on veut échanger des messages X.400,

- Créer des entrées MRX pour les abonnés locaux qui désirent utiliser X.400,

- Pour chaque entrée MRX, s'assurer qu'il existe une boite aux lettres MR,

- Aider et informer les utilisateurs du système,

- Résoudre les problèmes survenant sur MR ou/et MRX.

2.3.6 PROBLEMES RENCONTRES AVEC LES LOGICIELS ISO DE DIGITAL

Les problèmes que nous avons eu pendant l'utilisation des logiciels ISO de DIGITAL sont de deux types.
Le premier type de problème provient directement de l'implémentation des logiciels du fait que ceux-ci sont encore "jeunes" (Version 1.0 ou 1.1) et que certaines erreurs restent non corrigées et/ou non signalées.
Le second type de problème vient du contenu de la documentation, manuels et mises à jour, qui n'est pas toujours correct et qui est donc cause d'erreurs.

2.3.6.1 Problèmes d'implémentation

2.3.6.1.1 VOTS

VOTS offre une procédure de démarrage des modules OSIDRIVER et OSIACP mais paradoxalement, il n'y a pas de procédure permettant de les arrêter. Seule une mise en repos des processus est permise en mettant tous les NS-Providers dans l'état OFF.

Ceci nous a considérablement gêné lors de l'installation du produit VOTS aussi bien sur le site FNDP à Namur que sur celui de l'IIHE à Bruxelles. En effet, si l'on essaye de stopper OSIDRIVER et OSIACP par la commande DCL : **STOP process**, le système d'exploitation place ces processus dans un état RWAST (Ressource Waiting Asynchronous System Trap). Cet état est tel que le seul moyen d'arrêter ces processus est d'arrêter la machine.
De plus, ni les opérateurs des Facultés ni ceux de l'IIHE ne connaissent le moyen de "récupérer" un processus dans un tel état. Cette solution, on s'en doute, est loin d'être une solution pratique, surtout dans un Centre de Calcul où la ressource ordinateur est fort partagée.

Une limite importante de la version 1.1 du logiciel VOTS est l'incapacité d'établir une connexion transport avec des systèmes n'utilisant pas de sous-adresse X.25. Cette restriction empêche MRX de se connecter avec ces systèmes (EAN, par exemple). Nous analyserons au chapitre suivant les différences entre MRX et EAN ainsi que les solutions permettant l'interconnexion de ces deux systèmes.
On trouvera dans <TUCK-87> une étude abordant certains problèmes de sous-adressage.

L'installation de VOTS aux FNDP nous a permis de détecter une anomalie de comportement de PSI. Lors des tests de vérification d'installation, nous avons établis plusieurs connexions transport avec l'IIHE. Les premières connexions échouèrent, l'IIHE étant alors confrontée au problème mentionné ci-dessus.

Quelques semaines plus tard, les opérateurs du Centre de Calcul de Namur constatèrent qu'un circuit virtuel entre les FNDP et l'IIHE avait été ouvert le 23 décembre 1987 et était resté actif pendant 11 jours sans qu'aucune donnée n'ait été émise ou reçue. La date d'ouverture de ce circuit virtuel correspond aux premiers tests effectués avec VOTS.

La durée de vie de ce circuit virtuel est anormale, sinon impossible. En effet, durant ces 11 jours, le système de l'IIHE s'est arrêté à plusieurs reprises, ce qui aurait du normalement entrainer la libération de la connexion réseau. De plus, l'Accounting de PSI de l'IIHE n'a gardé aucune trace de cette connexion.

Nous ne pouvons conclure qu'à une défaillance de l'Accounting de PSI.
Nous ne pouvons malheureusement isoler d'avantage le problème, n'ayant

pas pu recréer les circonstances de son avénement, ni même en expliquer les causes.

2.3.6.1.2 TRACE

Nous avons pu remarquer que, dans certaines conditions, le tracage par les utilitaires PSI-TRACE et VOTS-TRACE respectivement des packets X.25 (PSI) et des TPDUs (VOTS) mêne à une erreur système fatale "FATAL BUG CHECK" qui entraine un arrêt total de la machine.

2.3.6.2 Problèmes de documentation

2.3.6.2.1 PASSWORD MR-MRX

La clause 'notify=MRX$NOTIFY' signifie que chaque fois qu'un message est déposé dans la mailbox MR correspondante, la mailbox VMS MRX$NOTIFY avertira MRX de la présence d'un message qui lui est destiné. Le mot de passe associé aux mailboxes servant de moyen de communication entre MR et MRX doit toujours être égal à la chaine spécifiée dans le fichier MRX$INIT.DAT (chaine vide par défaut), sinon MRX ne peut pas les utiliser. En effet, quand un message est déposé dans une mailbox, MRX en est averti par la mailbox VMS. Il va essayer d'aller lire le message en utilisant comme mot de passe la chaine définie par défaut. Si les mots de passe ne concordent pas, MRX ne sera pas autorisé à lire le message. Il génére alors un message dans son fichier d'enregistrement d'événement :

Error on Operation "I", Error "M", check Message Router documentation

Le problème est que cette erreur n'est pas définie dans la documentation du Message Router. Ce n'est que par déduction que nous sommes arrivés à identifier la cause de cette erreur.

2.3.6.2.2 ARRET DU MRX

La documentation spécifie que l'utilitaire MRXMAN permet aux propriétaires des comptes MRMANAGER et SYSTEM d'arrêter le Message Router X.400 Gateway par une commande SHUTDOWN. Or il s'est avéré que l'exécution de cette commande depuis le compte MRMANAGER provoquait l'apparition du message : "Only SYSTEM can shutdown MRX". Il y a donc une contradiction entre la documentation et la réalité, ce qui pose des problèmes d'organisation quand ce sont des personnes différentes qui sont chargées des fonctions de gestion du système et de l'ensemble MR/MRX.

2.3.6.3 Problèmes d'accounting

Le logiciel MRX version 1.1 est dépourvu de système d'accounting. Ceci rend impossible la facturation des messages aux utilisateurs à partir d'une facture d'utilisation du PSDN.

2.3.6.4 Problèmes de relais

D'après Monsieur Frans Haselbacher de l'Université de Graz, Autriche, MRX n'effectuerait pas le relais des messages correctement. Nous n'avons pas pu personnellement vérifier ces propos, mais nous nous devions néanmoins de le signaler.

2.3.6.5 Critique générale du système

L'ensemble MR/MRX apparait à plusieurs observateurs comme du "bricolage". En effet, la coexistence des deux logiciels hybrides est souvent complexe et artificielle. Par exemple un même abonné au système de messagerie doit être défini à deux endroits différents (MR et MRX). De plus, le courrier destiné à d'autres noeuds X.400 doit transiter par des mailbox du Message Router et les User Agents doivent travailler avec celui-ci exclusivement. Il serait préférable de disposer d'un seul logiciel implémentant complétement X.400.

D'après notre expérience du système, ce but ne semble pas inaccessible. En effet, le Message Router X.400 Gateway possède le RTS et le noyau session. La définition des noeuds et des abonnés peut être revue pour permettre une indépendance totale par rapport au Message Router. Le seul élément manquant à MRX est l'équivalent du Listener du Message Router, c'est à dire un interface entre MRX et les User Agents, et un gestionnaire d'une base de données de messages X.400.

Nous pouvons néanmoins dire que l'ensemble MR/MRX est une réussite dans son rôle de passerelle entre, d'une part le réseau DECnet et, d'autre part le réseau X.400. Par contre, nous ne pouvons pas en dire autant de son utilisation en tant que MTA.

3. CHAPITRE III : ADRESSAGE ET ROUTAGE

3.1 INTRODUCTION

Dans ce troisième chapitre, nous allons étudier, dans le cadre d'un service de messagerie X.400, les fonctionnalités liées, d'une part, aux techniques d'adressage et, d'autre part, au routage.

Pour obtenir plus de détails sur le sujet, nous conseillons de consulter, en plus de l'ensemble des recommandations de messagerie électronique <X.400>, les références suivantes : <X.121>, <IEEE-83>, <MACH-87>, <TUCK-87>, <BOVE-87> et <HENK-87>.

3.2 ADRESSAGE

Après un bref exposé des plans d'adressage prévus pour les systèmes X.400, nous analysons, en comparant différentes implémentations de messagerie électronique, les problèmes de notation d'adresse. Nous traitons ensuite, d'un point de vue théorique, la traduction des adresses à l'intérieur du modèle ISO. Nous concluons, sur base d'un exemple pratique, par l'analyse de l'implémentation faite par DIGITAL de ce modèle .

3.2.1 RECOMMANDATIONS DU CCITT

La recommandation X.400 limite le choix parmi les différentes combinaisons possibles d'attributs standards et d'attributs définis pour un domaine particulier. Ainsi, actuellement, deux formes d'O/R name sont prévues et spécifiées. La première forme (FORMAT 1) est destinée à identifier les utilisateurs du MHS, tandis que la seconde (FORMAT 2) est prévue essentiellement pour identifier les utilisateurs du service télétex ou d'autres services télématiques.
Ces différents formats ont déjà été définis dans les paragraphes 1.2.3.2 et 1.2.3.3 du chapitre I "LE MODELE ISO ET LA MESSAGERIE X.400".

3.2.2 EXEMPLES DE NOTATION D'ADRESSE

Nous allons nous pencher sur les problèmes liés à la notation d'une adresse en analysant la représentation syntaxique de celle-ci dans différents systèmes de messagerie X.400.

Pour présenter ces différents systèmes, nous prendrons comme utilisateur X.400, un certain FREDERIC DUPONT Junior (initiales FD) enregistré à la régie belge (BE) des téléphones et télégraphes (RTT). Cet utilisateur travaille pour la section informatique (INFO) des Facultés Notre-Dame de la Paix à Namur (FNDP) et est spécialisé en télécommunications (TELECOMS).

3.2.2.1 EAN

Développé au Canada par l'Université de Colombie Britannique (UCB), EAN fut la première implémentation d'un système de messagerie électronique conforme aux standards du CCITT. Sortie en Décembre 1983, elle se basait donc sur les versions préliminaires des recommandations X.400.

Nous nous limitons, sur base de <LABO-87>, à une brève approche au niveau de l'adressage. Le lecteur intéressé pourra trouver plus d'informations en consultant les manuels suivants : <EAN-83>, <EAN-84> et <EAN-85>.

Comme expliqué sur la figure III.1, un domaine privé EAN est formé d'un ensemble de MTA. Chaque MTA définit un sous-domaine (de même nom que ce MTA). Ce sous-domaine est également divisé en sous-domaines (chacun lié à un MTA particulier).

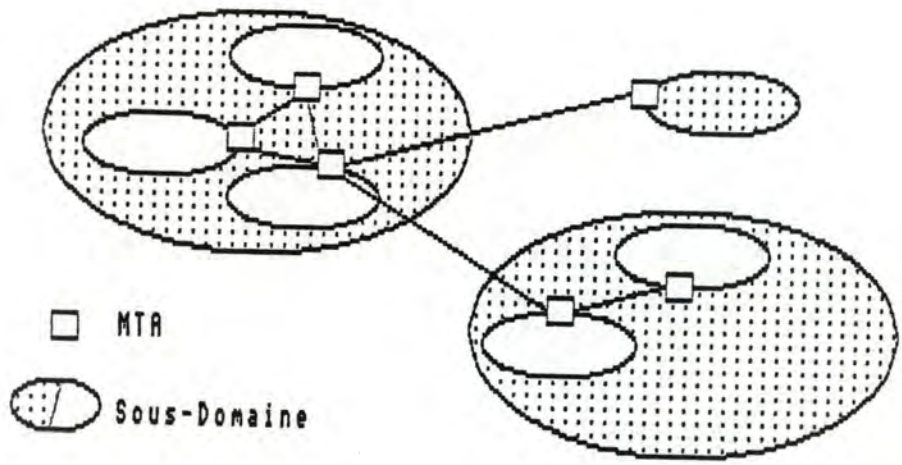


FIG III.1 Domaines et sous-domaines EAN

La syntaxe d'adressage utilisée pour accéder aux UA a la forme suivante :

"user @ subdomain.domain".

La partie "user" permet d'identifier un utilisateur sur sa machine; la partie "subdomain" permet d'identifier une machine particulière et peut se composer de plusieurs noms séparés par des "."; la partie "domain" permet d'identifier un domaine privé.

La représentation interne de ces adresses (O/R name) fait correspondre la partie "domain" au champ PRIVATE_DOMAIN_NAME et les deux autres parties à une séquence de DOMAIN_DEFINED_ATTRIBUTES.

Voici l'adresse EAN de FREDERIC DUPONT :

"FDUPONT @ TELECOMS . INFO . FNDP"

Cette adresse identifie l'utilisateur "FDUPONT", qui appartient au sous-domaine "TELECOMS", lui-même contenu dans le sous-domaine "INFO", qui à son tour fait partie du domaine privé "FNDP".

Voici l'O/R NAME correspondant à cette adresse EAN.

	Termes EAN
COUNTRY_NAME	=
ADMINISTRATION_DOMAIN_NAME	=
[PERSONAL_NAME]	=

[PRIVATE_DOMAIN_NAME]	= FNDP	domain
[ORGANIZATION_NAME]	=	
[{ ORGANIZATIONAL_UNIT_NAME }]	=	
[{ DOMAIN_DEFINED_ATTRIBUTES }]	= FDUPONT	user
	= TELECOMS.INFO	subdomain

Ce choix de notation d'adresse présente l'inconvénient de devoir utiliser pour le routage des éléments propres à un domaine particulier et donc de limiter les possibilités d'interconnexion avec des domaines n'ayant pas fait les mêmes choix. On peut également remarquer l'absence des éléments obligatoires relatifs aux pays, aux domaines administratifs.

3.2.2.2 DFN-EAN

DFN-EAN est une implémentation de messagerie électronique, basée sur EAN, qui a été modifiée par le Deutsches ForschungsNetz ou DFN (Réseau allemand (RFA) de recherche).

Alors que EAN s'attache à garder la possibilité d'interconnexion avec des systèmes de messagerie électronique existants (UUCP, CSNET ou ARPANET) en utilisant une syntaxe d'adressage de type RFC822 ("user @ domain") DFN étend cet adressage au monde X.400.

Une introduction aux différentes méthodes de traduction du format EAN en format DFN (et vice-versa) peut être trouvée dans <HENK-87>.

La syntaxe d'adressage utilisée dans DFN-EAN est de la forme suivante :

" < S=surname; O=organization; OU=org_unit; P=private; A=administration; C=country > "

L'adresse DFN de FREDERIC DUPONT et son correspondant en O/R name sont les suivants :

" < S=FDUPONT; O=INFO; OU=TELECOMS; P=FNDP; A=RTT; C=BE > "

		Termes DFN
COUNTRY_NAME	= BE	country
ADMINISTRATION_DOMAIN_NAME	= RTT	administration
[PERSONAL_NAME]	= FDUPONT	surname
[PRIVATE_DOMAIN_NAME]	= FNDP	private
[ORGANIZATION_NAME]	= INFO	organization
[{ ORGANIZATIONAL_UNIT_NAME }]	= TELECOMS	org_unit
[{ DOMAIN_DEFINED_ATTRIBUTES }]	=	

Ce choix de notation d'adresse, à l'inverse de la notation EAN, permet une interconnexion plus aisée avec d'autres implémentations malgré qu'elle ne tienne pas compte des informations sur les utilisateurs (Prénom, Initiales, Génération).
Nous pouvons remarquer que tout en étant incomplet (présence des éléments obligatoire uniquement), ce choix d'adressage respecte le format 1.1 des O/R names.

Remarquons de plus que la dernière version de EAN utilise l'extension d'adressage introduite par DFN.

3.2.2.3 GIPSI

Le Groupement d'Intérêt Public francais GIPSY-SM90, a développé sous UNIX un système de messagerie X.400. Celui-ci a été repris par le groupe BULL qui l'a distribué dans le cadre du projet ROSE du programme ESPRIT et qui le commercialise dans son système SPIX. Nous conseillons, pour obtenir plus de renseignements sur ce sujet, de consulter le thèse de doctorat de Bernard LABORIE <LABO-87> qui présente un aperçu des objectifs, des problèmes rencontrés et des solutions apportées dans l'implémentation, l'interconnexion et l'administration d'une messagerie X.400 sous UNIX.

La syntaxe d'adressage utilisée dans GIPSY est de la forme suivante :

" pays / administration / privé / organisation / machine / usager ".

Voici un exemple d'adresse GIPSY et son correspondant en format O/R name.

" BE / RTT / FNDP / INFO / VENUS / DUPONT "

		Termes GIPSY
COUNTRY_NAME	= BE	pays
ADMINISTRATION_DOMAIN_NAME	= RTT	administration
[PERSONAL_NAME]	= DUPONT	usager
[PRIVATE_DOMAIN_NAME]	= FNDP	privé
[ORGANIZATION_NAME]	= INFO	organisation
[{ ORGANIZATIONAL_UNIT_NAME }]	= VENUS	machine
[{ DOMAIN_DEFINED_ATTRIBUTES }]	=	

Ce choix de notation d'adresse, comme celui de DFN, respecte le format 1.1 des O/R names.

L'ajout de l'élément "machine" permet de gérer éfficacement le routage entre domaine privé lorsque les utilisateurs appartenant à ces domaines sont situés sur des machines ou des sites différents.
Nous avons supposé que la machine utilisée dans l'unité organisationnelle "TELECOMS", où l'utilisateur "DUPONT" est employé, s'appelle "VENUS".

3.2.2.4 MRX DE DIGITAL

Le logiciel X.400 développé par DIGITAL permet différents types de User Agents (Chapitre II Figure II.8). Nous allons analyser les choix syntaxiques de ces différents UA. Les références concernant ces produits sont les suivantes : <VAX-MR>, <VAX-MRX>, <VAX-MRIF> et <VAX-VMSMAIL>.

3.2.2.4.1 MESSAGE ROUTER X.400 GATEWAY (MRX)

La définition de l'ensemble logiciel MRX ayant été faite au chapitre II, nous nous contenterons ici de présenter les interfaces utilisateurs fournis par DIGITAL.

Le logiciel MRX supporte partiellement les formats d'O/R name définis actuellement dans la norme et présentés au chapitre I "LE MODELE OSI ET LA MESSAGERIE X.400".
En effet, alors que le format 1.1 est entièrement supporté, le format 1.2 est non délivré, le format 1.3 est uniquement relayé et le format 2 n'est pas du tout supporté.

ALL-IN-ONE, VMS-MAIL GATEWAY et MRX GATEWAY travaillent tous avec le format 1.1.

3.2.2.4.2 ALL-IN-ONE

ALL-IN-ONE est un logiciel intégré développé par DIGITAL, il permet entre-autres l'édition de texte, le travail sur tableur, la gestion de fichiers et offre un interface de messagerie électronique.

Les syntaxes des adresses ALL-IN-ONE destinées à la messagerie X.400 sont les suivantes :

```
1.    " KEYWORD=value       { [ @ KEYWORD=value ] } @ MRaddress "
2.    " CODE=value          { [ @ CODE=value    ] } @ MRaddress "
3.    " Given_name Surname { [ @ CODE=value    ] } @ MRaddress "
```

ou 'value' représente un champ de l'O/R name.
'MRaddress' représente l'adresse MR de la boite aux lettres initiale et ne sert que pour le routage au niveau local.
'KEYWORD' représente un mot-clé appartenant à la liste suivante

1	COUNTRY	2	ADMIN_DOMAIN
3	PRIVATE_DOMAIN	4	UNIT_NAME
5	ORGANIZATION	6	SURNAME
7	GIVEN_NAME	8	INITIALS
9	GENERATION	10	P3_SUBSCRIBER_ID
11	TELEMATIC_ADDRESS		

'CODE' représente le code associé à un mot-clé.
'Given_name' et 'Surname' sont respectivement le nom et le prénom d'un utilisateur.

Voici l'O/R name correspondant à notre exemple :

		Termes MRX
COUNTRY_NAME	= BE	country
ADMINISTRATION_DOMAIN_NAME	= RTT	admin_domain
[PERSONAL_NAME]		
GIVEN_NAME	= FREDERIC	given_name
SURNAME	= DUPONT	surname
INITIALS	= FD	initials
GENERATION	= JUNIOR	generation
[PRIVATE_DOMAIN_NAME]	= FNDP	private_domain

```
[ ORGANIZATION_NAME ]                = INFO          organization
[ { ORGANIZATIONAL_UNIT_NAME } ]     = TELECOMS      unit_name
[ { DOMAIN_DEFINED_ATTRIBUTES } ]    =
```

Les différentes notations suivantes sont permises :

1. Mots-clés "PRIVATE_DOMAIN=FNDP
   @ SURNAME=DUPONT @ GIVEN_NAME=FREDERIC
   @ UNIT_NAME=TELECOMS @ ORGANIZATION=INFO @ RTT"

2. Codes "3=FNDP @ 6=DUPONT @ 7=FREDERIC @ 4=TELECOMS
   @ 5=INFO @ RTT"

3. Codes et noms "FREDERIC DUPONT @ 4=TELECOMS @ 5=INFO @ 3=FNDP @ RTT"

L'élément 'RTT' est la MRaddress liée au domaine administratif RTT.

3.2.2.4.3 VMS-MAIL GATEWAY

VMS-MAIL est le logiciel implémentant le courrier inter-personnel sur le système d'exploitation VMS de DIGITAL.

Grace à la passerelle "VMS-MAIL GATEWAY", on peut envoyer à partir du logiciel existant, via une boite aux lettres spéciale "MRGATE", des messages vers le monde X.400 via le Message Router.

Les adresses du VMS-MAIL ont la forme suivante :

```
" MRGATE :: MRaddress :: { KEYWORD=value } :: given_name surname "
```

ou 'value, given_name et surname' représentent un champ d'O/R name.
'MRaddress' représente l'adresse MR de la boite aux lettres initiale
'KEYWORD' représente un mot-clé appartenant à la liste déjà définie pour ALL-IN-ONE.

Voici l'adresse VMS-MAIL correspondant à l'exemple pris pour ALL-IN-ONE.

```
" MRGATE :: RTT :: 5=INFO :: 4=TELECOMS :: 3=FNDP :: FREDERIC DUPONT "
```

3.2.2.5 REMARQUES GENERALES SUR L'ADRESSAGE

Dans les différents systèmes X.400 que nous venons d'analyser, on peut remarquer qu'il est très important de bien choisir le type de syntaxe que l'on utilisera sur un site et en relation avec d'autres sites.
Le format d'adressage à l'intérieur d'un site étant lié aux besoins des utilisateurs, il doit donc répondre à ces besoins et être facile à utiliser tout en restant compatible avec la norme.
De plus, l'interconnexion avec d'autres sites devant être la plus aisée possible, le site local ne doit pas faire, vis-à-vis de l'extérieur, de restrictions dans les types d'adressage envoyés et/ou reçus.
Il importe, dans les deux cas, de permettre une syntaxe évolutive et extensible pour la notation d'une adresse. En effet, la recommandation X.400 et les récents travaux sur X.400/88, laissent place à la création par les instances de normalisation de nouveaux formats d'adressage.

3.2.3 TRADUCTION DES ADRESSES

Présentons maintenant les différents niveaux d'adressage utilisés pour le transfert d'un message X.400.
Nous analyserons tout d'abord le modèle OSI et ce qu'il prévoit lors de l'établissement d'une connexion entre entités paires.
Ensuite, nous réfléchirons sur l'ensemble des transformations nécessaires à la traduction d'un O/R name en une adresse RESEAU via les différentes couches.

3.2.3.1 SCHEMA GENERAL D'UNE CONNEXION

Sur la figure (FIG III.2), nous apercevons les différents éléments utilisés lors d'une connexion entre entités.

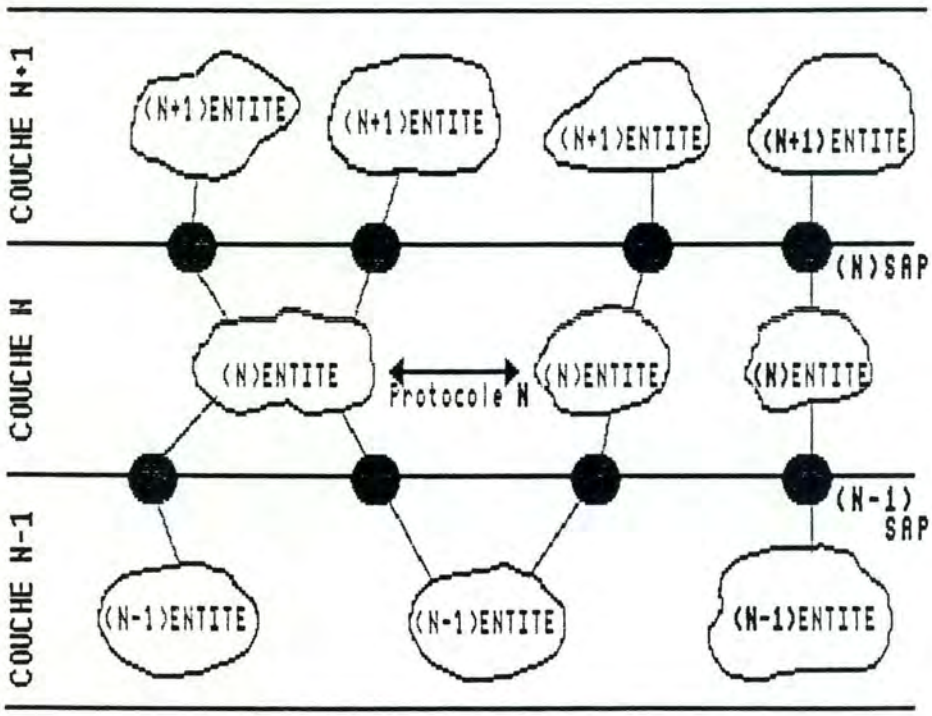


FIG III.2 ENTITES ET SERVICE ACCESS POINT (SAP)

Une (N) ENTITE est un processus de niveau (N).

Un SERVICE ACCESS POINT de niveau N ou (N)SAP permet aux entités (N+1) d'utiliser les services de la couche N.
Une (N+1) entité communique avec une (N) entité du même système via une (N)SAP qui joue le rôle d'interface logique entre les deux couches.
Un (N)SAP ne peut être servi que par une seule (N) entité et utilisé par une seule (N+1) entité. Une (N) entité peut servir plusieurs (N)SAP et une (N+1) entité peut utiliser plusieurs (N)SAP.

L'élément PROTOCOLE a été défini au paragraphe 1.1.2.4.

3.2.3.2 PROBLEMES DE DENOMINATION (NAMING)

3.2.3.2.1 IDENTIFICATION

Les objets (types et instanciations) doivent être identifiables de façon unique à l'intérieur d'une couche et entre différentes couches.
Pour pouvoir établir une connexion entre deux SAP, on doit être capable de les identifier de manière univoque.

Comme le montre la figure (FIG III.3), la norme ISO definit des identifiants pour les entités, les SAP, les connexions et les relations entre ces identifiants.

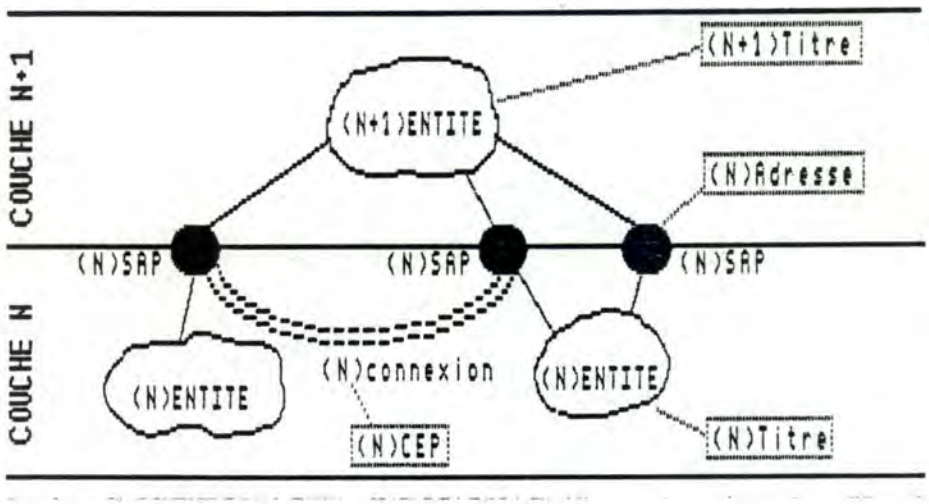


FIG III.3 IDENTIFIANTS UTILISES LORS D'UNE CONNEXION

Une (N) entité est identifiée par un titre (TITLE) global unique.
Dans une couche (N), les (N) entités ont des noms qui sont identifiants pour cette couche uniquement.

Un (N)SAP est identifié par une (N) adresse qui le désigne de manière unique à la frontière des couches (N) et (N+1) dans le monde ISO.

La liaison entre les (N) entités et les (N-1) SAP qu'elles utilisent est définie dans une (N)directory qui indique la correspondance entre les titres globaux des (N) entités et les (N) adresses par lesquelles ils peuvent être atteints.

La correspondance entre les (N) adresses servies par une (N) entité et la (N-1) adresse utilisée est faite par une fonction de MAPPING de niveau (N). Ce mapping peut être simple (1-1), hiérarchisé (1-n) ou sous forme de table. Dans le mapping hiérarchisé, une (N) adresse est composée d'une (N-1) adresse à laquelle on ajoute un (N) préfixe. Ce (N) préfixe est l'identifiant local du (N)SAP utilisé.

Par exemple l'adresse transport (N=4) "FNDP_VENUS.X25_DCS%2816000" est composée d'une adresse réseau (N=3) "X25_DCS%2816000" et d'un préfixe "FNDP_VENUS". Un exemple plus complexe est présenté au paragraphe 3.2.4.1.

Une (N) connexion est identifiée par son (N)CEP (ou (N) Connexion End Point) composé de l'identifiant des (N+1) entités et des (N)entités ainsi que des (N)SAP utilisées.

3.2.3.2.2 NOMS PRIMITIFS ET DESCRIPTIFS

Il existe deux types de noms, les noms primitifs et les noms descriptifs.

Un nom PRIMITIF est un nom UNIQUE donné par une administration.
Ex: un numéro de téléphone.

Un nom DESCRIPTIF est une composition de noms primitifs et de mots-clés.
Ex: une adresse (NOM, PRENOM, RUE, NUMERO, CODE POSTAL, LOCALITE)

3.2.3.3 ETABLISSEMENT D'UNE CONNEXION ENTRE ENTITES PAIRES

3.2.3.3.1 ASPECT THEORIQUE

Sur base de la figure (FIG III.4), nous allons analyser les différentes phases qui permettent l'établissement d'un connexion entre entités paires de niveau N+1.

Une (N+1) entité demande l'établissement d'une (N) connexion à partir d'un (N)SAP local vers un (N)SAP distant.

Elle donne au (N)SAP LOCAL, la (N) adresse du (N)SAP distant.
Celui-ci y ajoute sa propre (N)ADRESSE pour s'identifier par rapport au noeud appelé.

Dès que la connexion est établie, la (N+1) entité et la (N) entité utilisent la même (N)CEP pour désigner la (N) connexion.

Pour établir une (N) connexion, on doit avoir établi une (N-1) connexion.

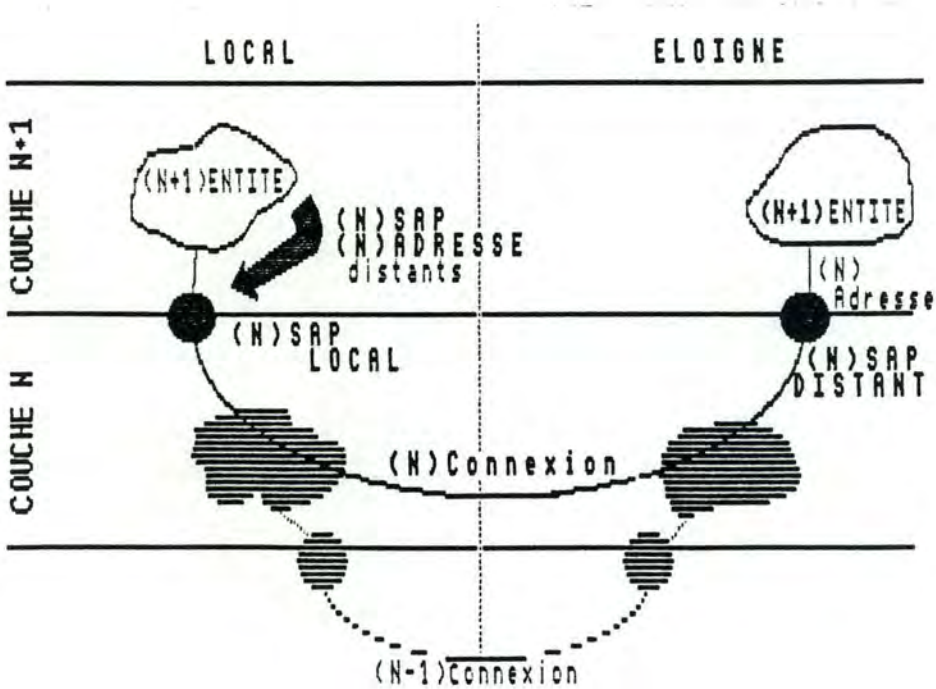


FIG III.4 ETABLISSEMENT D'UNE CONNEXION

3.2.3.3.2 ASPECT PRATIQUE

Nous allons illustrer la figure (III.3) par un exemple d'établissement d'une connexion TRANSPORT faite entre deux systèmes DIGITAL où le logiciel VOTS est installé. On supposera, pour simplifier, que le circuit virtuel (couche réseau) a déjà été ouvert.

La connexion TRANSPORT sera faite en utilisant un protocole de CLASSE 0 entre une entité locale et une entité éloignée. Cette entité est identifiée sur le réseau de communication par paquets par l'adresse DTE "22101680".

Les TPDU sont tracés par l'utilitaire VOTSTRACE par rapport à la vue du site local.

PHASE 1: Envoi du TPDU de demande de connexion (CR)

TPDU: 1 Time: 12:08:00.47 Direction: SENT Lng: 27
Src: 0000 Dst: 000D Nr: EOT: Ack: Credit: 00
Type: (E) Connect Request
(C2) Called TSAP: VOTSIVP
(C1) Calling TSAP: VOTSIVP - requestor
(C0) TPDU size: 512

L'entité locale demande l'établissement d'une connexion Transport à partir d'un TSAP local "VOTSIVP - requestor" vers un TSAP distant. Dans le CR-TPDU se trouvent bien, en plus des paramètres liés au type de protocole choisi, les noms des deux TSAP.

PHASE 2: Réception du TPDU de confirmation de connexion (CC)

TPDU: 2 Time: 12:08:16.61 Direction: RECEIVED Lng: 09
Src: 000D Dst: 000E Nr: EOT: Ack: Credit: 00
Type: (D) Connect Confirm
(C0) TPDU size: 512

L'entité distante confirme l'établissement de la connexion.

PHASE 3: Transfert des données

TPDU: 3 Time: 12:08:17.01 Direction: SENT Lng: 02
Src: 000E Dst: Nr: 00 EOT: E Ack: Credit: 00
Type: (F) Data

TPDU: 4 Time: 12:08:17.21 Direction: SENT Lng: 02
Src: 000E Dst: Nr: 00 EOT: E Ack: Credit: 00
Type: (F) Data

TPDU: 5 (...)

Un certain nombre de DATA-TPDU sont transférés.

Remarquons que l'adresse Transport du TSAP distant n'est évidemment pas présente à l'intérieur des TPDU. C'est la couche TRANSPORT qui seule utilise cette adresse pour établir la connexion et transmettre les TPDU.

3.2.3.4 SCHEMA GENERAL D'UNE CONNEXION X.400

Sur la figure (FIG III.5), nous apercevons les différents éléments utilisés lors d'une connexion (de niveau 7) entre deux noeuds X.400 et donc des différentes connexions créées dans les couches inférieures.

Nous nous limiterons à l'analyse des transformations d'adressage dans les couches de haut niveau (3 à 7).

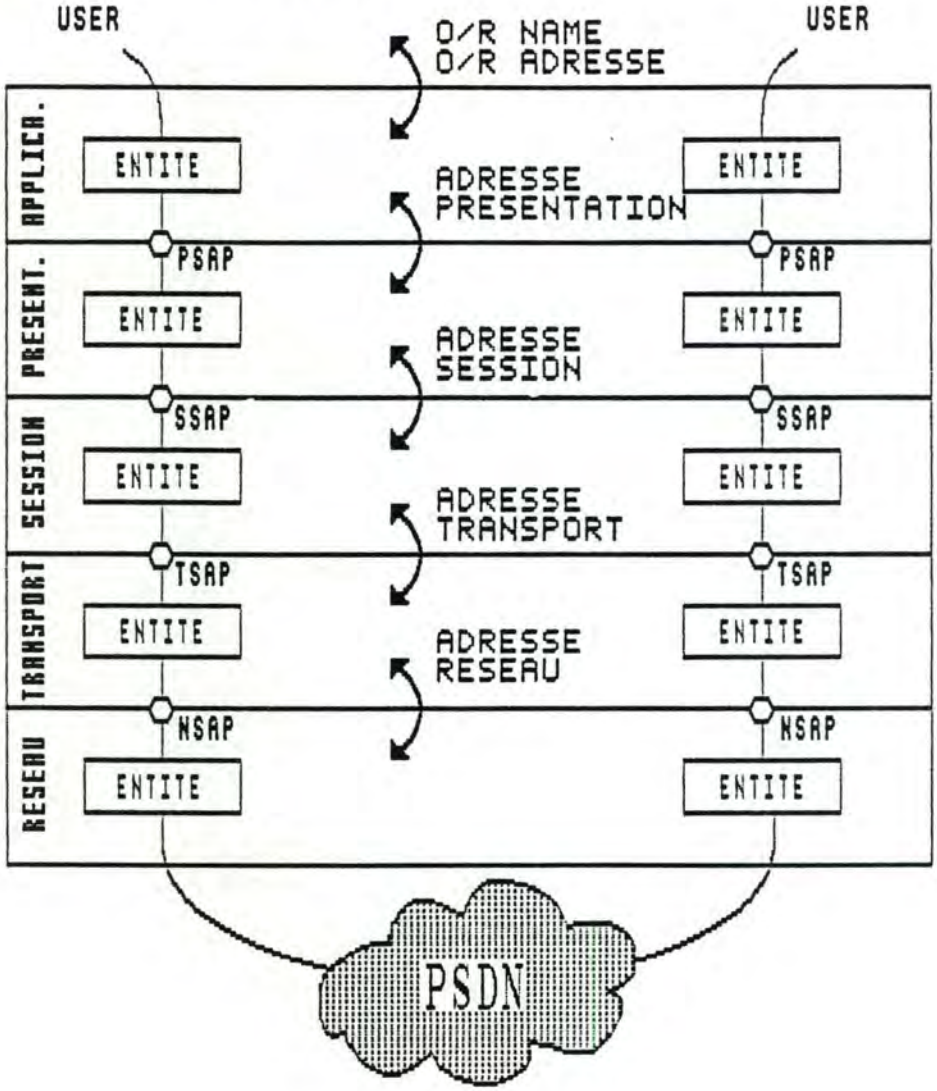


FIG III.5 SCHEMA GENERAL D'UNE CONNEXION X.400

L'O/R NAME par lequel un utilisateur local identifie le destinataire de son message doit être traduit lors de la connexion réelle en une série d'adresses intermédiaires et de noms de SAP.

C'est par des mécanismes de directory et de mapping d'adresse que l'O/R ADRESSE, liée à cet O/R NAME, sera successivement transformée en ADRESSE PRESENTATION, en ADRESSE SESSION, en ADRESSE TRANSPORT et enfin en ADRESSE RESEAU.

3.2.4 EXEMPLE D'ADRESSAGE (MRX).

Nous allons, sur base d'une l'architecture ISO de DIGITAL, montrer comment se fait la traduction ou MAPPING des adresses à travers les différentes couches qui supportent l'implémentation X.400.

La figure III.6 présente le schéma complet des transformations d'adressage entre les différentes couches traversées.

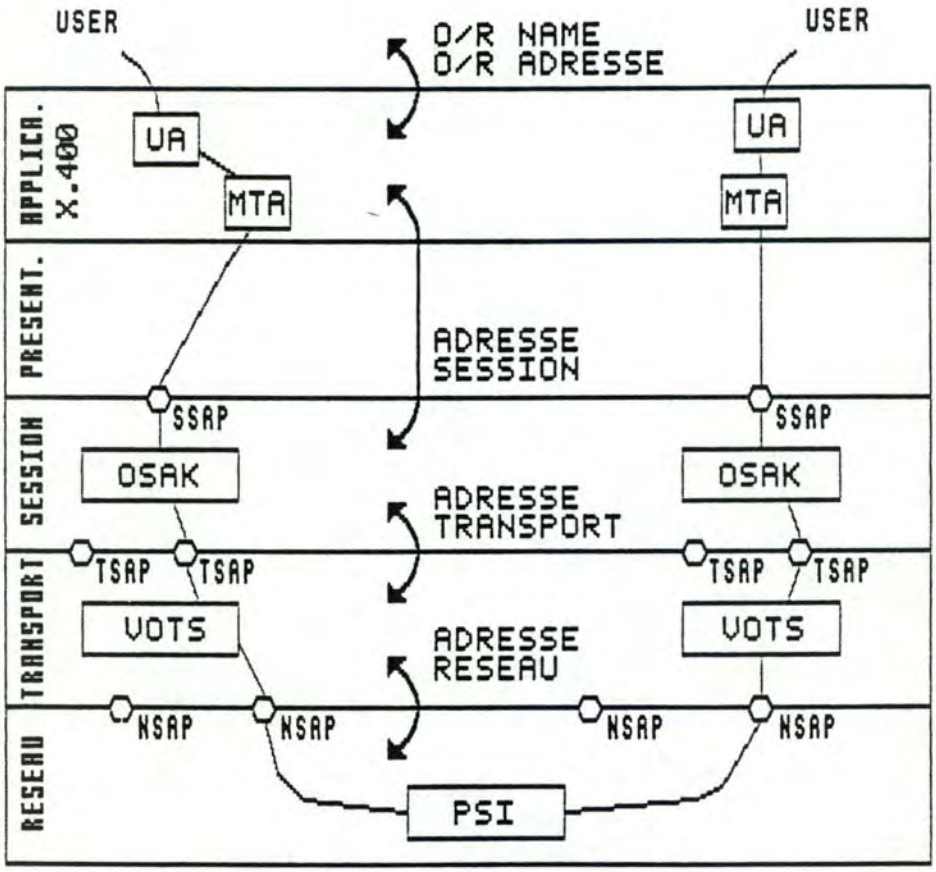


FIG III.6 MAPPING D'ADRESSE

En se basant sur les différents outils et bases de données des différentes couches définis au chapitre II, nous pouvons analyser plus en détail chaque étape de la transformation.

Remarquons l'absence de la couche PRESENTATION dans l'architecture ISO appliquée à la messagerie électronique X.400.

3.2.4.1 Exemple de mapping d'adresse

Analysons maintenant sur base d'un exemple, le MAPPING, premièrement, entre un nom de boite aux lettres du MESSAGE ROUTER (MRMAILBOX) d'un ABONNE et l'O/R name correspondant, et, deuxièmement, entre un nom de MRMAILBOX d'un DOMAINE et l'adresse réseau du MTA par lequel celui-ci peut être accédé.

Prenons le cas d'un message MR envoyé, depuis le domaine FNDP, par l'abonné "FDUPONT" vers un destinataire "MDURANT" spécifié par l'ADRESSE MR suivante :

" MDURANT @ IIHE_mbx "

Cette adresse MR est constituée de deux parties. La première est le nom de la MRMAILBOX d'un UTILISATEUR ("MDURANT") et la seconde est la MRMAILBOX du domaine auquel appartient cet utilisateur.

Deux types de mapping différents vont se produire lors du passage par la passerelle MRX :

1. Mapping entre la MRMAILBOX UTILISATEUR et l'O/R name associé

2. Mapping entre la MRMAILBOX DOMAINE et l'adresse réseau du MTA associé

3.2.4.1.1 Mapping entre MRMAILBOX UTILISATEUR et O/R NAME

Dès la prise en charge du message, le logiciel MRX du domaine FNDP consulte ses bases de données (Voir Chapitre II Fig. II.13) et, pour créer l'O/R NAME de l'envoyeur "FDUPONT", effectue la traduction suivante :

Selon la base de donnée "ABONNES" du domaine FNDP,

FDUPONT	devient	Surname	= DUPONT
		Initials	= FD
		Given Name	= FREDERIC
		Generation	= JUNIOR
		Organization	= INFO
		Unit Name	= TELECOMS

Pour compléter l'O/R NAME, MRX prend les valeurs par défaut suivantes dans la base de données de paramètres de contrôle du domaine FNDP :

Private Domain	=	FNDP
Admin Domain	=	RTT
Country Name	=	BE

3.2.4.1.2 Mapping entre MRMAILBOX DOMAINE et ADRESSE RESEAU DU MTA

3.2.4.1.2.1 Mapping entre MRMAILBOX et ADRESSE SESSION

La base de données "DOMAINES" du domaine FNDP s'occupe de la traduction de la MRMAILBOX du domaine de destination "IIHE_mbx" en adresse session.

IIHE_mbx fournit l'ADRESSE SESSION suivante :

" vide " . IIHE_MRX.X25_DCS%22101680

< SSAP > . < ADRESSE TRANSPORT >

Remarquons que, comme défini en 3.2.3.2.1, le mapping utilisé à partir de la couche session est HIERARCHISE. En effet, la (N) adresse (SESSION) est composée d'une (N-1) adresse (TRANSPORT) à laquelle on ajoute un (N) préfixe qui identifie le SSAP au niveau local.

Comme défini dans la norme <X.410>, le système de messagerie n'utilise pas l'adressage session. L'absence de SSAP dans la base de donnée de MRX, permet de passer directement l'"ADRESSE SESSION" à la couche transport.

3.2.4.1.2.2 Mapping entre ADRESSE SESSION et ADRESSE TRANSPORT

Cette traduction se fait directement car le MAPPING est hiérarchisé.

L'ADRESSE TRANSPORT est donc IIHE_MRX . X25_DCS%22101680

et se décompose comme suit < TSAP > . < ADRESSE RESEAU >

3.2.4.1.2.3 Mapping entre ADRESSE TRANSPORT et ADRESSE RESEAU

L'adresse RESEAU, extraite de l'adresse TRANSPORT par le principe de mapping hiérarchisé, est la suivante

X25_DCS % 22101680

nous y trouvons < NSAP > % < ADRESSE DTE >

Le NSAP ou Network Service Access Point est, selon la terminologie Digital, le nom du NS-PROVIDER (Fournisseur du service réseau) ou NSP. Ce NSP est composé, dans l'architecture OSI de Digital, d'un type de réseau et d'un nom de réseau séparé par convention par le caractère "_". Il identifie le service réseau que l'on utilise pour accéder au noeud éloigné. Dans l'exemple, nous utilisons un réseau de type X.25 qui s'appelle "DCS".

L'ADRESSE DTE a le format suivant :

[DNIC] (NATIONAL NUMBER) (SUBADDRESS)

Le DNIC ou "Data Network Identification Code" identifie le pays et le réseau dans ce pays. Pour DCS, le code DNIC est de 2062. Il est absent dans notre exemple car l'envoyeur et le destinataire sont sur le même réseau et dans le même pays.

Le NATIONAL NUMBER identifie le DTE dans un DNIC donné.

La SUBADDRESS est réservée à l'usage interne du DTE.

Le numéro DCS complet des FNDP est le suivant : **206 28160000**
et est composé d'un DNIC valant **206(2)**,
d'un NATIONAL NUMBER valant **2816000**
et d'une SUBADDRESS valant **0.**

Remarquons que les tailles de ces différents champs varient en fonction du pays et du réseau utilisé.

Le réseau DCS (BELGIQUE) a choisi un DNIC de 4 chiffres, un NATIONAL NUMBER de 7 chiffres (dont le premier est le dernier du DNIC) et une SUBADDRESS de 4 chiffres.

Le choix de TRANSPAC (FRANCE) est différent: il utilise aussi un DNIC de 4 chiffres mais le NATIONAL NUMBER a une longueur de 9 chiffres (en reprenant le dernier chiffre du DNIC) et la SUBADDRESS contient 2 chiffres.

3.2.4.1.3 TABLEAU RESUME DES DIVERS MAPPINGS

Adresse	Contenu et structure
MR	**IIHE_mbx**
SESSION	**" vide " . IIHE_MRX . X25_DCS % 22101680** < SSAP > . < ADRESSE TRANSPORT >
TRANSPORT	**IIHE_MRX . X25_DCS % 22101680** < TSAP > . < ADRESSE RESEAU >
RESEAU	**X25_DCS % 2210168 0** < NSAP > % < ADRESSE DTE >
AD.DTE	**2210168 0** (NATIONAL NR) (SUBADDRESS)

3.2.4.1.4 REMARQUE GENERALE SUR LE MAPPING DES ADRESSES

Il est bien évident que les éléments qui composent les différents niveaux d'adresse sont repris dans les bases de données des couches respectives et définies au chapitre II.

3.2.4.2 EXPLICATION DU FONCTIONNEMENT DE L'ETABLISSEMENT D'UNE ASSOCIATION X.400 DANS L'ARCHITECTURE DE DIGITAL.

Nous prendrons comme exemple la couche TRANSPORT (VOTS), pour expliquer comment on identifie les TSAP lors de l'établissement d'une connexion. On supposera qu'une connexion RESEAU est déjà ouverte.

3.2.4.2.1 Méthode de traitement de REQUETE DE CONNEXION TRANSPORT (CR-TPDU).

La réception d'un TPDU de REQUETE DE CONNEXION (CR-TPDU), entraine :

1. La création d'un NETWORK CONTROL BLOCK (NCB) dans le système appelé. Il contient des informations sur l'adresse DTE éloignée (ligne commutée) ou le nom de la ligne (ligne louée). De plus, on y trouve :

   - la SUBADDRESS du DTE éloigné,
   - la SUBADDRESS locale,
   - des codes de diagnostic,
   - des informations de contrôle d'accès,
   - le nom et les facilités offertes par le réseau X.25 utilisé.

2. La consultation de la base de données des TSAP.

   Si le nom du TSAP (ou TSAP_ID) est connu l'appel sera accepté.

   S'il s'agit d'un TSAP PASSIF (Voir Chapitre II Figure II.2b), après vérification des informations de contrôle d'accès (Account, User et Password), le système activera un processus "TSAP name" qui répondra alors à la requête de connexion.

   S'il s'agit d'un TSAP ACTIF (Voir Chapitre II Figure II.2a), le système préviendra le processus appelé "TSAP name" qui traitera l'appel.

3.2.4.2.2 Exemple de traitement de REQUETE D'ASSOCIATION X.400

Sur base de notre expérience, nous donnons ici un aperçu des INTERACTIONS entre les différents modules composant l'architecture physique qui supporte l'implémentation du modèle ISO faite par DIGITAL.

Une requête d'association X.400 entraine la demande d'ouverture de connexion aux différents niveaux sur le noeud local. Ces demandes sont transmises aux couches distantes respectives.

On supposera ici qu'il n'existe pas de connexion aux niveaux RESEAU, TRANSPORT et SESSION.

La requête d'association X.400 qui servira d'exemple est initiée dans le MTA "IIHE" (adresse DTE 2210168) et a pour destination le MTA "FNDP" (adresse DTE 2816000).

La couche APPLICATION de l'IIHE demande l'ouverture d'une connexion SESSION avec le MTA "FNDP". Comme vu au 3.2.4.1.2, l'Adresse SESSION vaut **<SSAP> . <TSAP>.<NSAP>%<ADRESSE DTE>.**

La couche SESSION de L'IIHE reçoit cette demande, elle se rend compte qu'il n'existe pas encore de connexion TRANSPORT avec le noeud "FNDP" et donc demande à son tour l'ouverture d'une connexion TRANSPORT entre les adresses **<TSAP>.<NSAP>%<ADRESSE DTE>.**

De même, La couche TRANSPORT de l'IIHE va recevoir cette demande, l'examiner et demander l'ouverture d'une connexion RESEAU entre les adresses réseau **<NSAP>%<ADRESSE DTE>** des deux MTA.

La couche RESEAU va alors demander l'ouverture d'un circuit virtuel entre les deux adresses DTE.

Dès que les différentes connexions sont ouvertes, le transfert des données peut alors commencer.

A l'IIHE, un message X.400 va être envoyé, il sera transmis successivement à la couche SESSION, puis à la couche TRANSPORT et enfin à la couche RESEAU. Celle-ci va envoyer un paquet de données vers le site FNDP.

Intéressons-nous au traitement de ce paquet dès son arrivée sur le site FNDP (Adresse DTE 2816000).

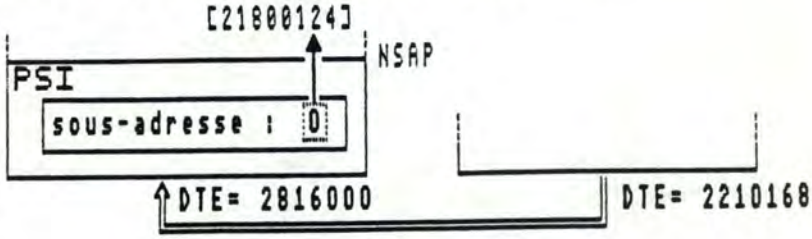
1. COUCHE RESEAU (PSI)

La couche réseau reçoit un paquet en entrée.
Le problème est de savoir comment cette couche peut le passer à un processus de niveau supérieur.
Ce paquet provient du circuit virtuel ouvert avec l'IIHE. Ce circuit a été ouvert entre notre noeud (2816000 plus une sous-adresse 0) et le numéro DTE 2210168 sur le réseau DCS.

Dans la base de données VOLATILE de PSI, on trouve par exemple :

```
Priority        = 126
Network         = DCS
Subaddress      = 0
Destination     = i 21800124 0001
```

Les éléments "Network" et "Subaddress" concordent. En supposant qu'il n'y ait pas d'entrée de plus haute priorité concordant elle aussi, l'appel est passé au processus de numéro "21800124".



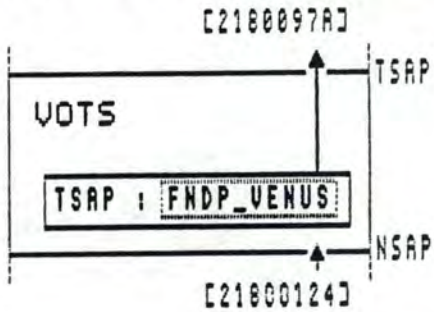
2. COUCHE TRANSPORT (VOTS)

"21800124" correspond au processus OSIACP, le gestionnaire de la couche TRANSPORT. Il se rend compte que le contenu du TPDU signale que celui-ci est destiné à passer vers un autre processus via le TSAP "FNDP_VENUS". OSIACP par l'OSIDRIVER consulte la base de donnée des TSAP.

Exemple de contenu :

TSAP id	: FNDP_VENUS
Decnet compatible	: no
TSAP name	: 2180097A
Account	: Not initialised
User	: Not initialised
Password	: Not initialised

Il remarque que "FNDP_VENUS" est un TSAP ACTIF et après vérification des informations d'accès, il signale au processus "2180097A" qu'il a un appel pour lui.

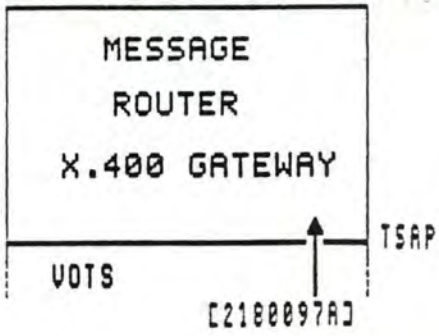


3. COUCHE SESSION (OSAK)

Le passage dans le couche SESSION est transparent car il n'y a pas de SSAP défini dans l'adressage X.400 donc aucune traduction d'adresse. L'appel est directement routé à la couche supérieure via le processus numéro "2180097A".

4. COUCHE APPLICATION (MR-MRX)

"2180097A" correspond au processus MRX$GATEWAY (ou Message Router X.400 Gateway) qui gère la passerelle X.400. Celui-ci va se charger du traitement du message.



3.2.4.3 Interconnexion entre MRX et EAN

Les manières de gérer les requêtes d'associations dans les architectures MRX et EAN étant essentiellement différentes, il nous a paru utile de consacrer un paragraphe au problème de l'interconnexion entre le Message Router X.400 Gateway et EAN.

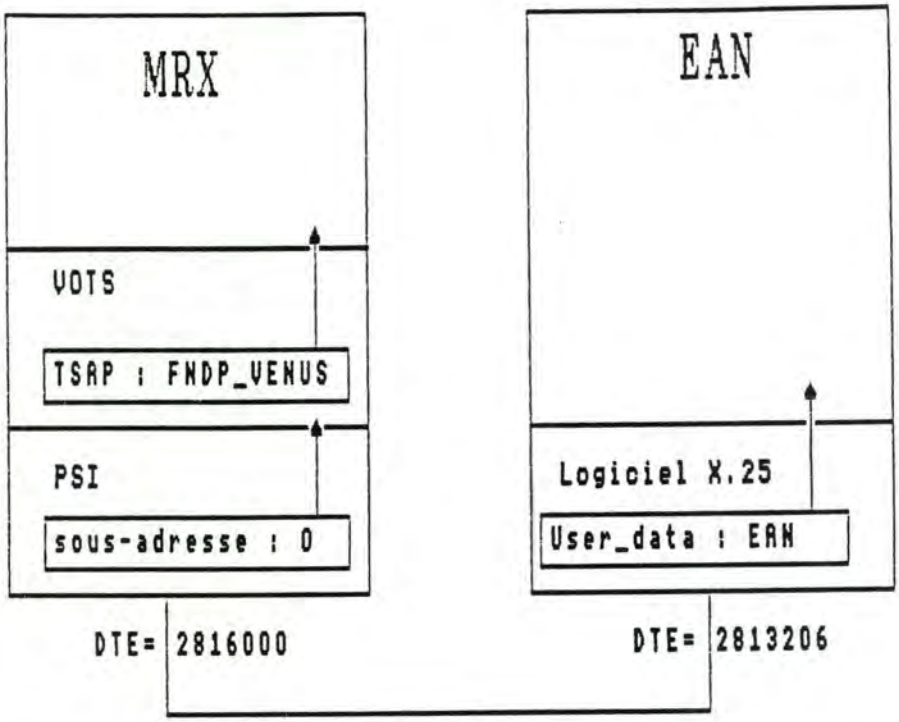


Fig. III.17 : Architectures comparées de MRX et de EAN

La figure III.17 montre comment est dirigée une requête de connexion sur les deux systèmes.

Nous avons vu que MRX est identifié sur un site par une adresse transport de la forme <SSAP><TSAP><ADRESSE RESEAU>. Cette ADRESSE RESEAU est constituée d'un NS-Provider (ou NSAP), d'une adresse DTE et d'une sous-adresse. C'est la valeur de cette **sous-adresse** (0 dans l'exemple) contenue dans le paquet d'appel, qui permet au logiciel de gestion de la couche réseau (PSI) de diriger l'appel vers le logiciel de la couche transport (VOTS). Le TPDU d'appel contient un TSAP (FNDP_VENUS) qui permet au logiciel VOTS de diriger l'appel vers MRX et d'établir ainsi un dialogue entre MRX et un autre logiciel X.400.
Une des limites de VOTS ne permet pas à celui-ci d'établir de connexion transport avec des systèmes n'utilisant pas de sous-adresse X.25. MRX utilisant VOTS, il hérite donc de cette restriction.

Dans le cas de EAN, une requête de connexion contient dans le paquet d'appel la valeur "EAN" dans le champ **User_data.** Le logiciel de gestion de la couche réseau (X.25 sur le schéma) dirige ces demandes de connexion vers le processus EAN.

Quand il envoie un message, EAN a la possibilité d'utiliser une adresse de la forme <TSAP><adresse DTE><sous-adresse>. Sur la figure III.17, nous aurions: **FNDP_VENUS.2816000.0.** Il n'y a donc pas à première vue, de problème d'interconnexion dans le sens EAN vers MRX.

Pour réaliser l'échange de messages dans l'autre sens (de MRX vers EAN), il est indispensable que la couche réseau (Logiciel X.25) responsable de EAN permette de diriger vers celui-ci les requêtes de connexion dont le paquet d'appel contient dans le champ User_data la valeur "EAN" (pour l'interconnexion EAN-EAN) OU une sous-adresse d'une valeur prédéfinie (pour l'interconnexion EAN-MRX).

MRX utiliserait alors une adresse de la forme **..X25_DCS%28132060**, où le SSAP ET le TSAP seraient nuls, l'adresse RESEAU étant constituée d'un NS-Provider (X25_DCS), d'une adresse DTE (2813206) et d'une sous-adresse.

Une autre solution consiste à disposer d'une version du logiciel VOTS supportant les réseaux X.25 n'utilisant pas de sous-adresse (comme dans le cas de EAN par exemple).

Ces inconvénients pourraient être évités par l'existence d'un NSAP standard qui remplacerait l'usage d'une sous-adresse ou d'un champ User_data. On utiliserait dans ce cas des adresses de la forme <SSAP><TSAP><NSAP><adresse DTE>.

Par exemple : **.FNDP_VENUS.VOTS%2816000** et **..EAN%2813206**

"EAN" et "VOTS" étant les NSAP via lesquels seront routées les requêtes de connexion pour permettre le traitement des TPDU par la couche transport.

3.3 ROUTAGE

Le routage, ou acheminement, est l'action prise par un MTA qui consiste a sélectionner un MTA "voisin" auquel il transmettra un message, une sonde ou une notification. Cet événement détermine de manière incrémentale la route d'un objet à travers le MTS. Comme il est mentionné dans <LABO-87>, " Le routage est lié à la fois à l'adressage au sein de notre propre système et à l'adressage des systèmes avec lesquels nous désirons nous interconnecter. Il est fortement dépendant des plans d'adressage utilisés dans ces différents systèmes, et a naturellement évolué en même temps que nous avons voulu nous interconnecter avec d'autres réseaux et que notre propre réseau s'est agrandi ".

Les fonctions de routage s'appuient sur des tables de routage locales au MTA. Nous nous proposons d'étudier comment s'effectue le routage dans le réseau Message Router d'abord, ensuite nous verrons comment le Message Router X.400 Gateway achemine les messages sur le réseau ISO.

3.3.1 Le routage sur le réseau des Message Routers

Pour comprendre comment le Message Router effectue le routage d'un message, il faut tout d'abord distinguer comment peuvent être définis les noeuds locaux et éloignés dans la base de données Directory. En effet, les différentes méthodes d'acheminement utilisées par le Message Router dépendent de la façon dont est défini le prochain noeud sur la route. A partir de ces différentes manières de diriger un message, nous proposerons un algorithme de synthèse.

3.3.1.1 Définition des noeuds

Un noeud du réseau des Message Routers peut être local ou éloigné. Un noeud local peut être unique, ou faisant partie d'un VAX Cluster, c'est à dire un ensemble de machines (VAX, micros-VAX, stations de travail, ...) partageant les mêmes ressources et une même unité spatiale. Un noeud éloigné peut être un noeud DECnet, voisin ou distant. Examinons plus en détails ces différences, et voyons comment elles sont définies dans la Directory.

3.3.1.1.1 Noeud local

Le noeud local est le noeud où est définie la Directory. Si le noeud local fait partie d'un VAX Cluster, les autres noeuds utilisant la même base de données du Message Router sont aussi appelés noeuds locaux.

Exemple 3.1 : Définition de noeuds locaux d'un VAX Cluster.

```
VENUS /replace = ""
DIANE /replace = ""
```

L'exemple 3.1 indique au Message Router qu'il doit remplacer le terme de l'adresse correspondant à une de ces deux entrées par la chaine vide.
Supposons que le destinataire d'un message soit défini comme étant "FDUPONT@VENUS".
Le Listener recherche l'entrée correspondante à VENUS. Il effectue l'instruction Replace. Comme cette entrée n'est pas une mailbox d'un abonné ou d'un noeud, le Listener recherche à nouveau dans la Directory une entrée correspondante à la nouvelle destination obtenue, soit <FDUPONT>.

Nous pouvons déja définir une partie de l'algorithme de routage du Message Router : le Listener repète la recherche d'une entrée dans la Directory correspondant au premier terme de l'adresse jusqu'à ce que celui-ci ne soit pas une entrée de remplacement.

3.3.1.1.2 Noeuds éloignés

Un noeud éloigné est un noeud avec lequel une liaison DECnet est nécessaire pour pouvoir échanger des messages. Le Message Router distingue trois types de noeuds éloignés : les noeuds DECnets, les noeuds voisins et les noeuds distants.

3.3.1.1.2.1 Noeuds DECnets

Les noeuds DECnets, ou noeuds par défaut, sont des systèmes VAX/VMS implémentant le Message Router, mais qui n'ont pas d'entrée dans la Directory du Message Router. Le Talker consulte la base de données DECnet pour envoyer des messages destinés à ces noeuds. Ces messages se trouvent dans une mailbox spéciale du Message Router :

```
DEFAULT_DECNET /owner=MRMANAGER /network_node /talker= ...
```

Quand le Listener ne peut résoudre une adresse, il mettra le message dans la mailbox DEFAULT_DECNET, si celle-ci est présente. Si le Talker trouve une entrée correspondante à l'adresse considérée dans la base de données DECnet, et que ce noeud est un noeud Message Router, le Talker lui enverra le message. Sinon, le message est effacé de la base de données et une notification de non-livraison est envoyée à son expéditeur. Ceci constitue une autre phase de l'algorithme.

3.3.1.1.2.2 Noeuds voisins

Un noeud voisin est un noeud que le Message Router local peut atteindre directement. Un noeud peut être défini comme voisin, même s'il n'est pas physiquement adjacent au noeud local, pourvu qu'il soit accessible directement par une liaison DECnet, c'est à dire un lien logique entre deux systèmes VAX/VMS éloignés.

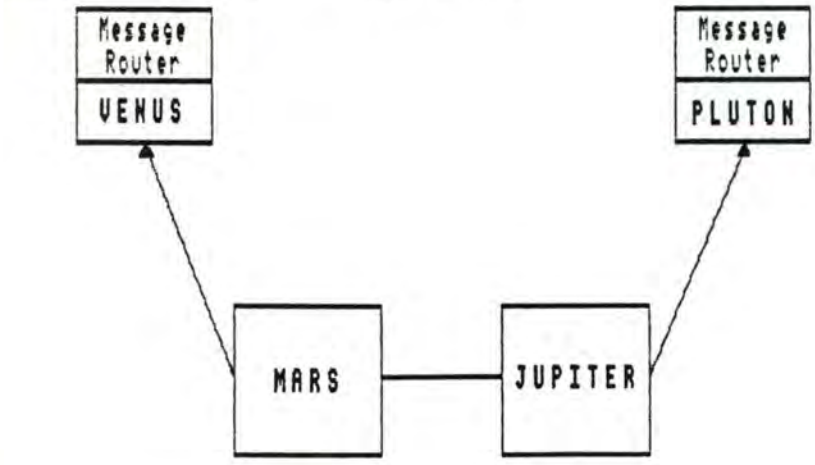


FIG III.7 : NOEUDS VOISINS

La figure III.7 montre comment le noeud Pluton, bien que distant de trois noeuds du noeud Venus, peut être défini comme noeud voisin dans la Directory de celui-ci. Le Message Router de Venus peut passer un message directement à celui de Pluton sur une liaison DECnet passant par Mars et Jupiter. Le message ne sera enregistré et relayé que sur Pluton. Le routage est effectué uniquement en utilisant le réseau DECnet. Ce mécanisme est semblable à l'établissement d'un circuit virtuel sur un PSDN où les noeuds intermédiaires ne font que relayer des paquets de données et non des messages à travers le réseau, le noeud destinataire se chargeant du réassemblage. Cette méthode ne fonctionne cependant que si toutes les liaisons sont disponibles en même temps.

Exemple 3.2 : Noeud voisin.

```
PLUTON /owner=MRMANAGER /network_node /talker= ...
```

PLUTON	est le nom du noeud voisin Message Router,
owner=	identifie le propiétaire de la mailbox,
Network_node	définit l'entrée comme une mailbox d'un noeud voisin,
talker=	indique comment démarrer le Talker pour communiquer avec ce noeud.

3.3.1.1.2.3 Noeuds distants

Un noeud distant est un noeud sur le réseau qui n'est accessible que par une route nécessitant plus d'une liaison DECnet. Les messages sont transférés entre le noeud local vers le noeud distant en étant acheminés vers le noeud voisin approprié sur le réseau des Message Routers. Le voisin est alors responsable de l'enregistrement du message et de son relais vers le noeud suivant sur la route.

Reprenons l'exemple de la figure III.7 et supposons maintenant que les noeuds Mars et Jupiter soient des noeuds Message Router et voyons comment la Directory du Message Router de Venus définira le noeud Pluton.

Exemple 3.3 : Noeuds distants.

```
MARS   /owner=MRMANAGER /network_node /talker= ...
PLUTON /route=@JUPITER@MARS
```

Quand le Message Router de Venus doit envoyer un message pour le noeud Pluton "FDUPONT@PLUTON", son Listener consulte la Directory. Il constate que l'entrée Pluton contient des informations de routage. Il ajoute la route définie pour atteindre le destinataire du message, ce qui donne : "FDUPONT@PLUTON@JUPITER@MARS". Le Listener va refaire une recherche dans la base de données pour trouver l'entrée correspondant au "nouveau destinataire" : Mars. Celui-ci est un noeud voisin (network_node) avec lequel une liaison DECnet est réalisable. Le Talker de Venus va envoyer le message vers Mars où il sera enregistré et relayé vers le noeud suivant.

Nous obtenons ainsi encore un élément pour la construction de l'algorithme : le Listener repète la recherche d'une entrée dans la Directory correspondant au premier terme de l'adresse jusqu'à ce que celui-ci ne soit pas une entrée de routage.

3.3.1.2 Méthodes de routage

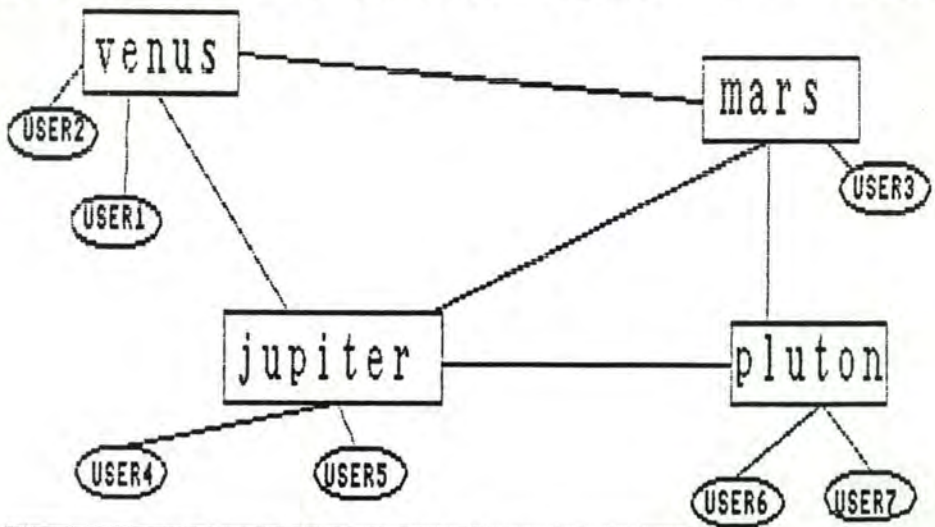
Le Message Router permet de configurer le réseau avec cinq types de méthodes de routage. La combinaison des méthodes de routage dépend de la taille et de la complexité du réseau, ainsi que du trafic de messages escompté. Dans ce qui va suivre, le terme "utilisateur" désigne aussi bien un abonné du système de messagerie qu'un User Agent.

3.3.1.2.1 Routage par défaut

Le routage par défaut est la méthode la plus simple. Cette méthode utilise la base de données du réseau DECnet sur chaque noeud pour acheminer les messages. La base de données du Message Router ne doit donc pas contenir de mailbox pour les autres noeuds. Cependant, ceci ne permet pas d'utiliser tous les avantages des possibilités de store-and-forward du Message Router. En effet, le message est enregistré sur le noeud émetteur jusqu'à ce qu'une route complète soit disponible vers le noeud destination. Alors seulement, le message est transféré, en une fois, sur la route définie.

3.3.1.2.2 Routage implicite

Le routage implicite suppose que chaque base de données Message Router spécifie la route complète vers chaque abonné sur le réseau. L'envoyeur d'un message ne doit spécifier que le nom de la mailbox réceptrice du message, le Message Router se chargeant d'ajouter les informations de routage.



DIRECTORY DU NOEUD "VENUS"

JUPITER	/network_node	User4	/route= @JUPITER
MARS	/network_node	User5	/route= @JUPITER
PLUTON	/network_node	User6	/route= @PLUTON
User1	/owner=...	User7	/route= @PLUTON
User2	/owner=...	VENUS	/replace=""
User3	/route= @MARS		

FIG III.8 : EXEMPLE DE ROUTAGE IMPLICITE

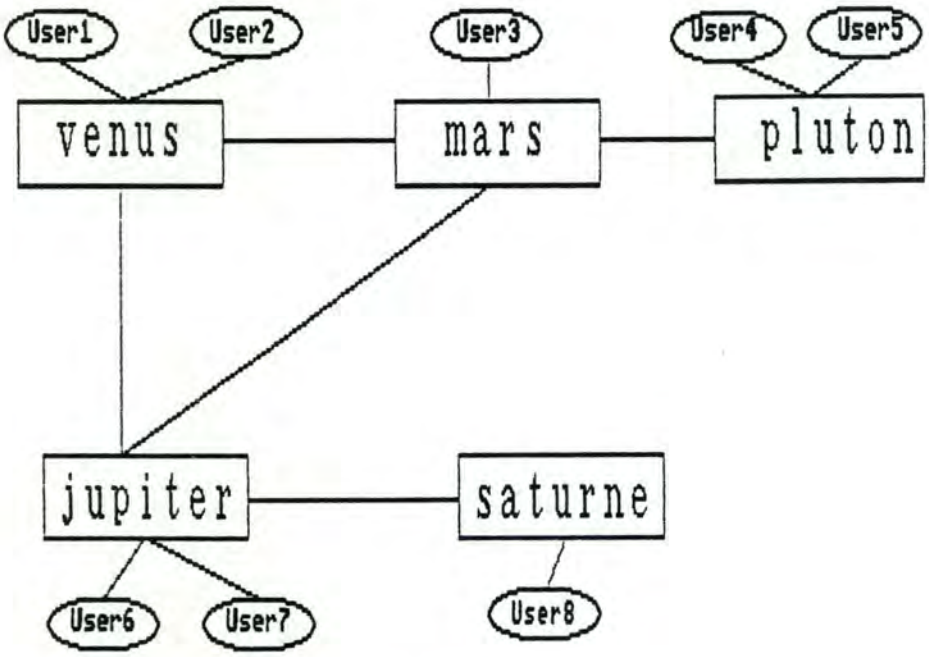
La Directory de chaque noeud de la figure III.8 définit tous les utilisateurs et tous les noeuds dans le réseau.
Quand l'abonné User1 sur le noeud Venus veut envoyer un message à User7 sur le noeud Pluton, il indique comme destinataire du message <User7>. Le Listener de Venus consulte sa Directory et ajoute les informations de routage correspondant à l'entrée User7, ce qui nous donne :

"USER7@PLUTON".
Pluton devenant le premier noeud à atteindre, le Listener va poster le message dans la mailbox Pluton où il sera pris en charge par le Talker.

Cette méthode de routage est idéale pour un réseau stable de petite taille. L'utilisation pour les abonnés en est facilitée, mais la gestion peut s'avérer être très difficile si la configuration du réseau est instable. La Directory sur chaque noeud contient des entrées pour tous les autres noeuds, les abonnés et les User Agents du réseau. Toutes les entrées des Directories doivent donc avoir un nom unique sur tout le réseau. Il faut mettre à jour toutes les Directories à chaque modification de structure du réseau : ajout ou suppression d'un noeud, d'un abonné. La base de données grandira très rapidement avec l'ajout de nouveaux utilisateurs au système.

3.3.1.2.3 Routage de destination

Le routage de destination permet à l'expéditeur d'un message de ne spécifier que le nom de la mailbox de son destinataire, ainsi que le nom du noeud auquel il appartient. La Directory spécifie la route complète pour atteindre les autres noeuds du réseau, ce qui décharge l'utilisateur de la spécification des noeuds intermédiaires sur le chemin.



DIRECTORY DU NOEUD "VENUS"

JUPITER	/network_node	User1	/owner=...
MARS	/network_node	User2	/owner=...
PLUTON	/route= @MARS	VENUS	/replace=""
SATURNE	/route= @JUPITER		

FIG III.9 : EXEMPLE DE ROUTAGE DE DESTINATION

La figure III.9 montre comment se déroule le routage par destination. Pour envoyer un message à User3 sur Mars, un abonné sur Venus doit indiquer le destinataire et son noeud, soit "User3@Mars". Le Listener place le message dans la mailbox correspondant à Mars et le Talker envoie

le message. Le courrier adressé à User8 sur Saturne sera libellé comme suit : "User8@Saturne".
Le Listener voit que Saturne est une entrée de routage, il va ajouter à l'adresse les informations s'y trouvant, ce qui donne : "User8@Saturne@Jupiter".
L'entrée correspondant à Jupiter est une mailbox, le Listener va y déposer le message, ce qui provoquera le démarrage du Talker et l'envoi du message vers Jupiter qui se chargera d'enregistrer le message et de le relayer vers Saturne. Seuls les utilisateurs locaux sont définis dans la Directory.

Cette méthode de routage est valable pour de grands réseaux. Pour les abonnés, l'utilisation en est aisée, mais la gestion peut être compliquée si le réseau n'est pas stable. La Directory contenant des informations d'acheminement vers chaque noeud sur le réseau, elle doit être mise à jour à chaque changement de configuration du réseau.

3.3.1.2.4 Routage explicite

Le routage explicite (FIG III.10) impose à l'émetteur d'un message de spécifier le nom de la mailbox réceptrice et, en plus, la route complète à accomplir pour atteindre cette mailbox. La Directory ne contient plus que les noms des noeuds voisins sur le réseau, qui sont les premiers noeuds sur la route d'un message.

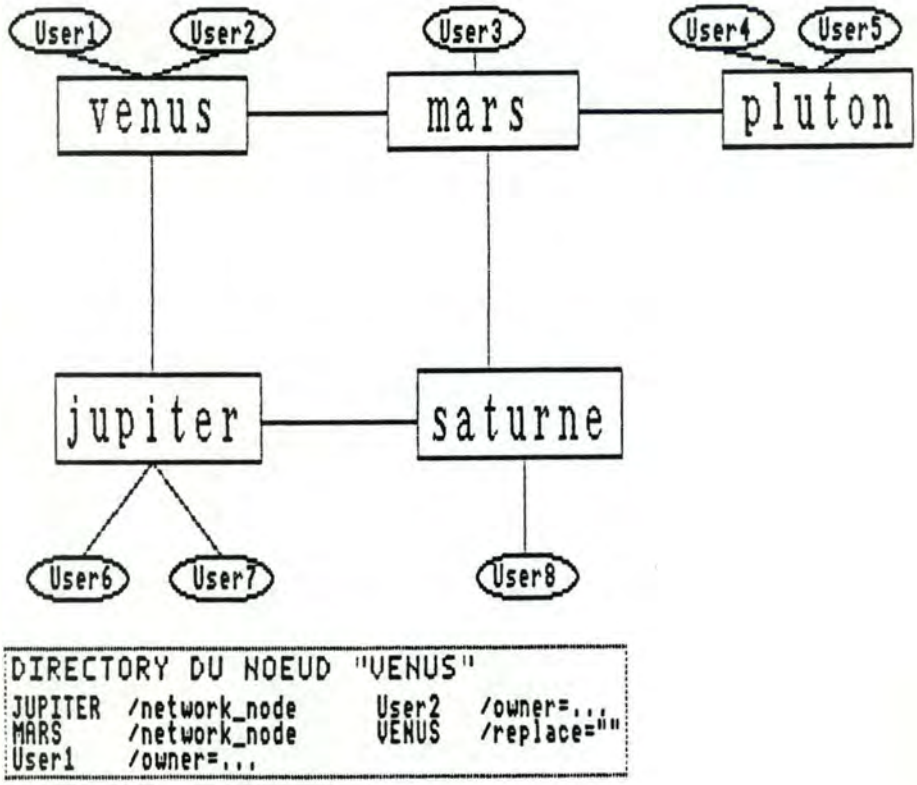


FIG III.10 : EXEMPLE DE ROUTAGE EXPLICITE

Pour adresser un message à l'abonné User7, User1 sur Venus devra spécifier la route complète devant être suivie par le message, soit :

"User7@Pluton@Mars". Le Listener reconnait Mars comme une mailbox et y poste le message, ce qui provoque le démarrage du Talker et l'envoi vers Mars. La Directory de Mars contient les informations de ses noeuds voisins, donc Mars pourra relayer le message à son tour vers Pluton. Pour envoyer un message vers Saturne, les utilisateurs de Venus ont le choix entre deux routes possibles : "User8@Saturne@Mars" ou "User8@Saturne@Jupiter".

Cette méthode convient pour de grands réseaux où le trafic n'est pas très intense et où les listes de distributions ne sont pas souvent utilisées. La gestion de la Directory est simplifiée par le fait que seuls les noeuds voisins y sont définis et concernés par un changement de configuration. Par contre, les utilisateurs voient leurs taches compliquées par le fait qu'ils doivent spécifier la mailbox destination, le noeud auquel elle appartient et la route complète pour l'atteindre.

3.3.1.2.5 Routage par zones

Le routage par zones consiste à diviser logiquement de grands réseaux complexes en un certain nombre de zones, chaque zone comprenant un certain nombre de noeuds. Une zone peut être déterminée selon des critères géographiques ou fonctionnels. Chaque zone possède un noeud passerelle vers les autres zones. Ce noeud passerelle enregistre et relaye les messages destinés aux noeuds de sa zone ainsi que ceux provenant de ses noeuds vers les autres zones. Le rôle du noeud passerelle peut être vu comme celui d'un domaine public dans le réseau X.400.

Les Directories des noeuds de chaque zone doivent spécifier :

- la route vers les autres noeuds de la zone, en routage par défaut ou explicite,
- la route vers les autres zones,
- la route vers les noeuds d'une autre zone.

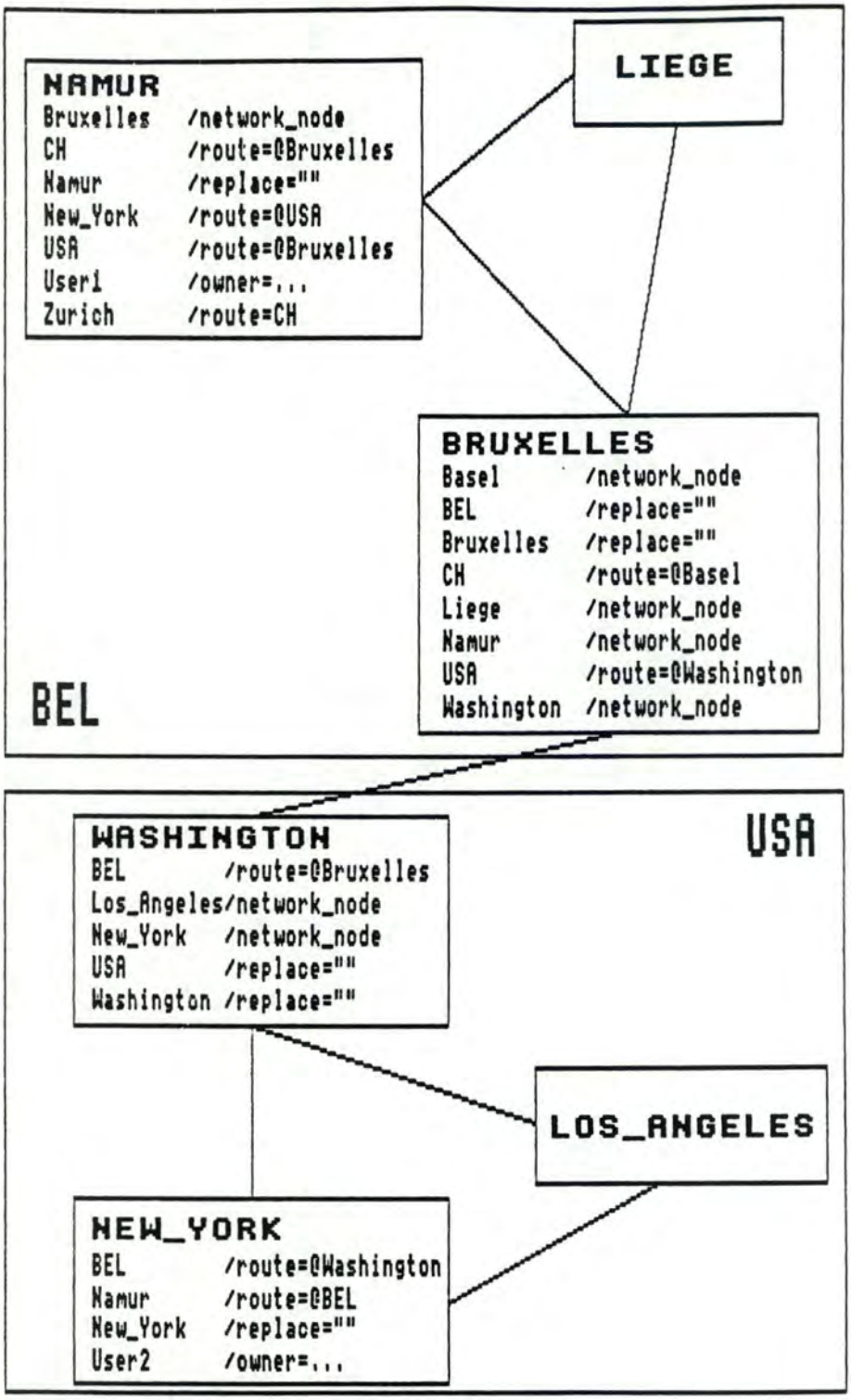


FIG III.11 : EXEMPLE DE ROUTAGE PAR ZONES

La figure III.11 montre comment un réseau important peut être divisé en plusieurs sous-réseaux logiques. Nous allons voir comment évolue l'adresse destination lors de l'envoi d'un message de User1 à Namur vers User2 à New York. Les changements de l'adresse sont présentés noeud par noeud avec la clause de la base de données qui a provoqué la modification.

A NAMUR, le Listener reçoit le message d'un User agent

User2@New_York

(New_York, route=@USA)

User2@New_York@USA

(USA, route=@Bruxelles)

User2@New_York@USA@Bruxelles

Livraison dans la mailbox Bruxelles.
Envoi du message au noeud Bruxelles par le Talker.

A **BRUXELLES**, le Listener reçoit le message
User2@New_York@USA@Bruxelles
(Bruxelles, replace="")
User2@New_York@USA
(USA, route=@Washington)
User2@New_York@USA@Washington
Livraison du message dans la mailbox Washington.
Envoi du message au noeud Washington par le Taklker.

A **WASHINGTON**, le Listener reçoit le message
User2@New_York@Washington
(Washington, replace="")
User2@New_York@USA
(USA, replace="")
User2@New_York
Livraison dans la mailbox New_York .
Envoi au noeud New_York par le Talker.

A **NEW YORK**, le Listener reçoit le message
User2@New_York
(New_York, replace="")
<User2>
Livraison dans la mailbox User2.

Nous laissons au lecteur le soin de reconstituer le chemin pris par la réponse à ce message depuis User2 à New York jusqu'à User1 à Namur.

La méthode de routage par zones permet de diminuer le trafic de messages sur des liaisons fort utilisées. Si, par exemple, un message est adressé à plusieurs utilisateurs sur des noeuds d'une même zone, seule une copie du message est transférée entre les zones. L'avantage de ce système est de diviser le réseau en plusieurs unités facilement gérables et d'offrir une méthode simple d'adressage aux utilisateurs.

Il faut souligner que les noeuds passerelles doivent pouvoir traiter les messages très rapidement et être capable de d'absorber un taux de messages suffisant de manière à ne pas dégrader les performances du système. De plus, le temps d'indisponibilité d'une passerelle doit être le plus bas possible, car une zone entière peut être coupée du reste du réseau en cas de défaillance de la passerelle.

3.3.1.3 ALGORITHME DE ROUTAGE DU MESSAGE ROUTER

Après l'analyse des différentes façons de définir un noeud dans la base de données et des différentes méthodes de routage, nous pouvons donner un algorithme de routage des messages dans le Message Router.

L'algorithme de routage d'un message dans le Message Router peut être présenté ainsi :

-a) on teste le dernier élément de la liste des récipients

- a.a) SI celui-ci possède une entrée dans la Directory alors, selon le type d'entrée,
  - a.a.a) SI c'est une entrée </replace=string>, on remplace l'élément testé par <string> et on retourne en a),
  - a.a.b) SI c'est une entrée </route=string>, on ajoute <string> à la liste To et on retourne en a),
  - a.a.c) SI c'est une entrée d'une mailbox d'un noeud </talker=string> ou </notify=string>, alors on place le message dans cette mailbox et on active le Talker correspondant,
  - a.a.d) SI c'est une entrée de mailbox d'un abonné </owner=string), on place le message dans la mailbox et si l'entrée contient </beep>, on avertit le propriétaire en affichant un message sur son terminal.
- a.b) SINON (il n'y a pas d'entrée correspondante dans la Directory),
  - a.b.a) SI l'option <DEFAULT_DECnet> est présente dans la Directory, on y place le message et on active le Talker qui va consulter la base de données DECnet.
    - a.b.a.a) SI le noeud est connu de la base de données DECnet, alors le TALKER transfère le message.
      - a.b.a.a.a) SI le noeud est un noeud Message Router, le transfert réussit,
      - a.b.a.a.b) SINON (le noeud n'est pas un noeud MR) , il est impossible de transférer le message, celui-ci est effacé et une notification de non-livraison pour cause d'adressage invalide est envoyée à l'expéditeur.
    - a.b.a.b) SINON (adresse non résolue avec la BD DECnet) , le message est effacé et une notification de non-livraison pour cause d'adressage invalide est envoyée à l'expéditeur.
  - a.b.b) SINON (l'option <DEFAUT_DECnet> est absente), le message est effacé et une notification de non-livraison pour cause d'adressage invalide est envoyée à l'expéditeur.

3.3.2 LE ROUTAGE SUR LE MESSAGE ROUTER X.400 GATEWAY

Le Message Router X.400 Gateway, comme nous l'avons déjà dit plus haut, est une passerelle entre un ou plusieurs Message Routers et le réseau ISO. Dans cette optique, quand il reçoit un message à destination d'un autre MTA X.400, le Message Router X.400 Gateway ne peut pas utiliser les informations de routage contenues dans les bases de données du Message Router. Seules, les informations contenues dans les bases de données MRX, abonnés et domaines, sont utiles au relais de messages X.400.

3.3.2.1 Différences entre domaines publics et privés

Le Message Router X.400 Gateway se comporte différement selon qu'il appartient à un domaine public ou privé.

Un domaine privé est dirigé par une compagnie ou une organisation. Il échange des messages avec d'autres domaines privés ou avec le domaine public auquel il appartient. Il possède une base de données recensant tous les domaines avec lesquels il peut communiquer directement. Le courrier adressé à des domaines ne se trouvant pas dans sa base de données est envoyé au domaine public qui se chargera de les router vers leur destination. Un domaine privé ne relaye pas de messages, sauf s'il simule le comportement d'un domaine public.

Un domaine public est dirigé par une administration d'un seul pays. Un pays peut posséder plusieurs domaines publics. Son rôle est de controler le relais de messages entre domaines. Quand il reçoit un message d'un domaine privé, le domaine public en est responsable, c'est à dire qu'il doit s'assurer que les messages qu'il reçoit sont conformes à X.400 au niveau P1. Il doit pouvoir communiquer avec tous les domaines privés dépendant de lui, ainsi qu'avec tous les autres domaines publics du réseau X.400.

Il faut cependant nuancer ces propos. En effet, lors de son installation dans un domaine privé, MRX demande s'il doit ou non relayer les messages qu'il recevrait d'autres domaines. Ce choix dépend de la configuration du MHS.

3.3.2.2 Configurations du Message Handling System

Le Message Handling System peut être configuré de nombreuses manières différentes. Les combinaisons possibles de domaines (publics et/ou privés) sont illimitées et ne peuvent pas être énumérées. Nous présentons ci-après quelques configurations importantes et représentatives.

L'élément de base du MHS est le Management Domain (MD ou Domaine de Gestion). Un MD est un ensemble se composant d'au moins un MTA et de zéro ou plusieurs UA. Il appartient à une administration ou à une organisation. Les communications entre machines à l'intérieur du MD ne doivent pas nécessairement être conformes aux recommandations X.400. Cependant si le MD veut communiquer avec d'autres domaines X.400, il doit s'assurer que les messages qu'il émet sont conformes à ces recommandations.

3.3.2.2.1 Connexions directes

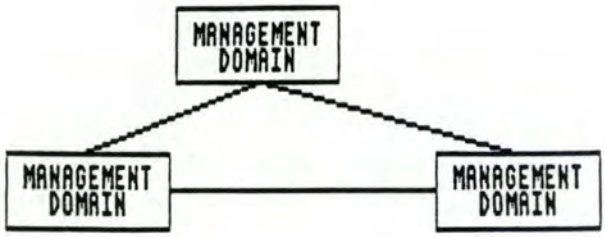


FIG III.12 : DOMAINES DIRECTEMENT CONNECTES

Le MHS peut être consitué de plusieurs domaines gérés par des organisations dont les équipements sont interconnectés. Sur la figure III.12 nous voyons plusieurs Management Domains interconnectés. Chaque domaine peut échanger des messages avec tous les autres, ce qui signifie qu'il doit les connaitre tous. Cette conception impose qu'un changement de configuration du réseau doit être répercuté parmi tous les composants du MHS.

3.3.2.2.2 Connexions indirectes

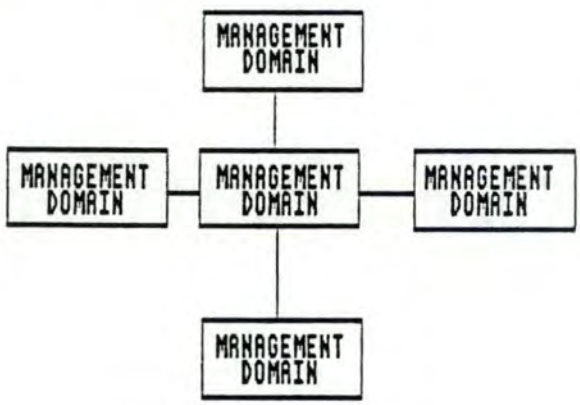


FIG III.13 : DOMAINES INDIRECTEMENT CONNECTES

La figure III.13 montre un MHS constitué de plusieurs domaines dont l'un sert de relais pour tous les autres. Seul le domaine "central" doit connaitre tous les autres domaines du MHS. Un domaine quelconque adresse tout son courrier au domaine central qui se charge de le diriger vers sa destination. Les changements de configuration du réseau n'auront d'importance que pour le MD intermédiaire.

3.3.2.2.3 Configurations hybrides

Nous appelons configuration hybride, une configuration de MHS qui peut être décomposée en différentes parties correspondant à une des configurations "pures" décrites ci-dessus.

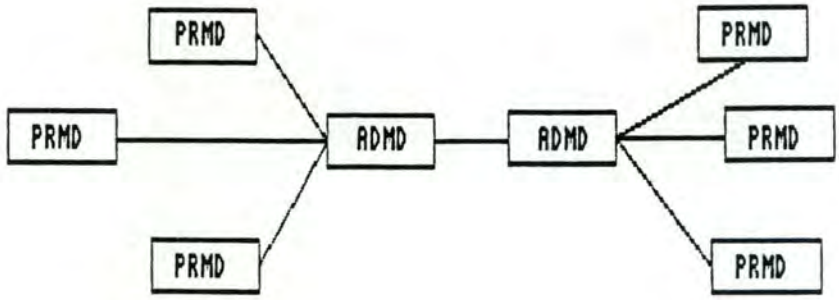


FIG III.14 : RELAIS PAR DES ADMD

Sur la figure III.14, chaque domaine privé (PRMD) est connecté à un seul domaine public (ADMD). Les communications entre PRMD se font par l'intermédiaire d'un ou de plusieurs ADMD exclusivement. Les ADMD sont connectés entre-eux. Les PRMD et leur ADMD constituent une configuration centralisée, tandis que les ADMD entre-eux forment une configuration à connexion directe. Dans un tel système, il est important que les ADMD aient une capacité de traitement de messages en rapport avec le nombre de PRMD dont ils ont la charge, ainsi que du trafic de messages entre les PRMD. La gestion du routage au niveau des PRMD est extrémement simple : ils ne communiquent directement qu'avec leur ADMD.

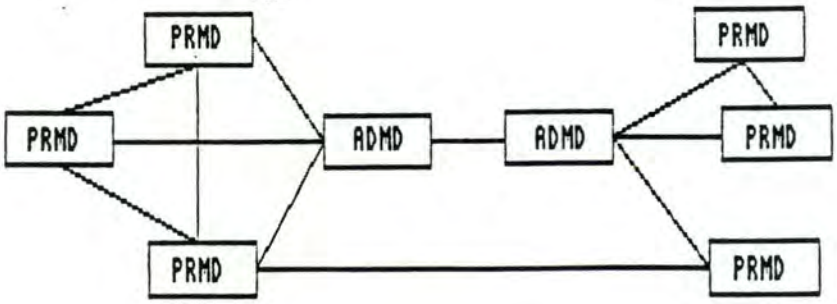


FIG III.15 : RELAIS PAR PRMD ET ADMD

La figure III.15 constitue une variante de la figure III.14. Les domaines privés peuvent communiquer directement entre-eux sans passer par un ADMD.

Les PRMD peuvent servir d'intermédiaires entre deux PRMD. Quand un PRMD ne peut se connecter directement à un autre PRMD, il envoie le message à son ADMD, qui lui peut se connecter à tous les PRMD de son administration. Ici, chaque domaine privé maintient une base de données de domaines avec lesquels il communique souvent. Un changement de configuration d'un PRMD n'aura d'influence que sur les domaines avec lesquels il est directement connecté. L'algorithme de routage du Message Router X.400 Gateway du paragraphe suivant est donné pour un PRMD d'une configuration semblable à celle de la figure III.15

L'avantage de cette solution est qu'elle permet de diminuer le nombre de relais entre MD pour lesquels l'échange de messages est très élevé. D'autre part, des connexions directes entre PRMD permettent de diminuer la charge d'utilisation des ADMD.

L'autorisation donnée ou refusée à deux PRMD d'échanger directement des messages au sein d'un même ADMD ou entre deux ADMD est en-dehors de la portée des recommandations du CCITT. Cette autorisation doit résulter d'accords entre les PRMD concernés et leurs ADMD, ces accords étant basés par exemple, sur des considérations techniques, politiques, économiques ou autres.

3.3.2.3 Algorithme de routage du Message Router X.400 Gateway

Soient : PRD name, le nom de domaine privé,
Recipient, l'O/R name de la personne à qui est adressé le message.

-a) On teste le PRD name de destination du message,

- a.a) SI il est différent de celui défini dans la base de données des paramètres de contrôle MRX du domaine local,
  - a.a.a) SI celui-ci est connu (il correspond à une entrée dans la base de données des domaines), on aura une transmission directe avec ce domaine.
  - a.a.b) SINON (celui-ci est inconnu),
    - a.a.b.a) SI le message peut être délivré dans un récipient alternatif et qu'une mailbox Opérateur existe, ALORS le message y est posté,
    - a.a.b.b) SINON il est impossible de router le message et une notification de non-livraison pour O/R name non reconnu est envoyée à l'expéditeur.
- a.b) SINON (il correspond au PRD name local),

  On teste le Personal Name,
  - a.b.a) SI le nom est connu (il correspond à une entrée dans la base de données des abonnés MRX), MRX dépose le message au Message Router qui le délivrera à son destinataire final.
  - a.b.b) SINON (le nom est inconnu),
    - a.b.b.a) SI le message peut-être délivré dans un récipient alternatif et qu'une mailbox Opérateur existe, ALORS le message y est posté,
    - a.b.b.b) SINON il est impossible de router le message et une notification de non-livraison pour O/R name non reconnu est envoyée à l'expéditeur.

Nous nous devons de souligner ici le rôle que doit jouer le propriétaire de la mailbox Opérateur. Chaque fois qu'un message mal adressé ou incomplet arrivera sur le domaine MRX, c'est l'opérateur qui devra décider soit de le faire suivre vers le domaine public auquel MRX appartient, soit de le poster à une mailbox d'un abonné local qu'il aurait pu identifier, soit de refuser le message en renvoyant une explication de non-livraison à son expéditeur.

Nous nous permettons sur base de <LABO-87> de proposer une amélioration au comportement de cet algorithme. En effet, le rôle d'un domaine public étant de connaitre tous les domaines privés qui dépendent de son administration, il parait plus judicieux de router vers ce domaine tous les messages dont le nom de domaine est inconnu de notre site et de ne délivrer à la mailbox Opérateur que les messages destinés à notre site,

mais de destinataire inconnu. Cette amélioration devrait diminuer la charge de travail de l'opérateur.

L'algorithme deviendrait :

Soit : AMD name, le nom de domaine public.

-a) On teste le PRD name de destination du message,

- a.a) SI il est différent de celui défini dans la base de données des paramètres de contrôle MRX du domaine local,
  - a.a.a) SI celui-ci est connu (il correspond à une entrée dans la base de données des domaines), on aura une transmission directe avec ce domaine.
  - a.a.b) SINON (celui-ci est inconnu)
    - a.a.b.a) SI il y a dans la base de données des domaines MRX une entrée correspondant au AMD name défini dans des paramètres de contrôle, on transmet le message vers ce domaine.
    - a.a.b.b) SINON il est impossible de router le message et une notification de non-livraison pour O/R name non reconnu est envoyée à l'expéditeur.
- a.b) SINON (il correspond au PRD name local),

  On teste le Personnal Name,

  - a.b.a) SI le nom est connu (il correspond à une entrée dans la base de données des abonnés MRX), MRX dépose le message au Message Router qui le délivrera à son destinataire final.
  - a.b.b) SINON (le nom est inconnu)
    - a.b.b.a) SI le message peut être délivré dans un récipient alternatif et qu'une mailbox Opérateur existe, alors le message y est posté,
    - a.b.b.b) SINON il est impossible de router le message et une notification de non-livraison pour O/R name non reconnu est envoyée à l'expéditeur.

3.4 ADRESSAGE ET ROUTAGE DANS UN RESEAU MR/MRX

L'objectif de ce paragraphe est de montrer comment doivent être adressés les messages dans un système comportant plusieurs Message Routers reliés à un Message Router X.400 Gateway.

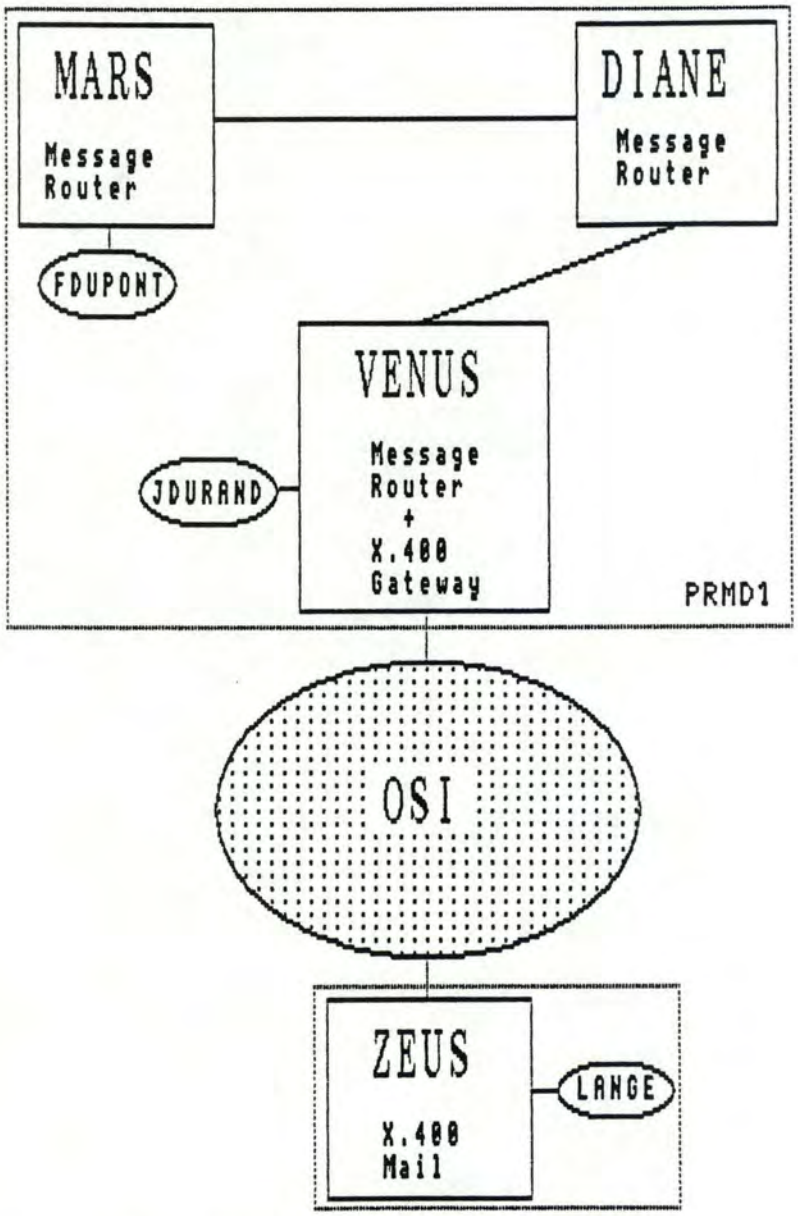


FIG III.16 : EXEMPLE DE CONFIGURATION MR/MRX

La figure III.16 montre deux domaines privés (PRMD1 et ZEUS) qui s'échangent des messages X.400. Le domaine Zeus est basé sur un logiciel X.400 quelconque, tandis que PRMD1 utilise du logiciel Digital uniquement. PRMD1 se compose de trois noeuds Message Router : Mars, Diane et Venus. Venus est le noeud passerelle vers le réseau ISO. Les abonnés des trois systèmes échangent des messages avec ceux de Zeus.

Considérons les Directories des Message Routers sur les noeuds Mars et Venus :

```
MARS :
  DIANE          /network_node /talker = ...
  FDUPONT        /owner = FDUPONT /beep
  MARS           /replace = ""
```

```
    VENUS          /route = @DIANE
    ZEUS_MRX       /route = @VENUS@DIANE

  VENUS :
    DIANE          /network_node /talker = ...
    JDURAND        /owner = JDURAND /beep
    MARS           /route = @DIANE
    VENUS          /replace = ""
    ZEUS_MRX       /owner = MRMANAGER /notify = MRX$NOTIFY
```

Les bases de données MRX sur Venus contient les informations suivantes:

Abonnés :

```
    Message Router address: FDUPONT@MARS@DIANE
    / surname      = Dupont
    / given_name   = Frédéric
    / initials     = FD
    / unit_name    = Personnel

    Message Router address: JDURAND
    / surname      = Durand
    / given_name   = Jean
    / initials     = JD
```

Domaines:

```
    Message Router address: ZEUS_MRX
    / prdname           = ZEUS
    / network_address   =  ...
```

Paramètres de contrôle:

```
    Country_name        = BE
    Admin_domain        = RTT
    Private_domain      = PRMD1
```

Un correspondant sur le domaine Zeus :

```
    Surname         = Lange
    /Given_name     = David
    /Unit_name      = Finance
    /organization = ORG1
    /Prdname        = Zeus
```

3.4.1 Envoi de message

Tous les messages issus du Message Router à destination d'un autre domaine sur le réseau X.400 doivent transiter par la passerelle du noeud Venus. Les abonnés doivent donc adresser leur courrier à une mailbox du Message Router du noeud Venus. Celui-ci gére une mailbox pour chacun des domaines avec lesquels des messages sont échangés.

L'abonné Frédéric Dupont sur Mars envoie un message à David Lange sur Zeus.

Sur MARS :

```
    TO:   SURNAME = Lange
          @GIVEN_NAME = David
          @UNIT_NAME = Finance
          @ORGANIZATION = ORG1
          @PRIVATE_DOMAIN = Zeus
          @ADMIN_DOMAIN = RTT
          @COUNTRY = BE
          @ZEUS_MRX@VENUS@DIANE          (Message Router address)

    FROM: FDUPONT                        (Message Router address)
```

La liste TO: va être analysée par le Listener qui va tenter de résoudre le premier destinataire du message, c'est à dire le dernier élément de la liste, soit : DIANE. Comme cet élément correspond à une mailbox dans la Directory, le Listener va y délivrer le message. Le Talker se chargera de l'envoyer vers le noeud Diane.

Sur DIANE :
Le Message Router de Diane modifie le dernier terme de l'adresse à l'arrivée du message sur le noeud et va le délivrer à la mailbox correspondant au nouveau dernier terme.

```
    TO:   SURNAME = Lange
          @GIVEN_NAME = David
          @UNIT_NAME = Finance
          @ORGANIZATION = ORG1
          @PRIVATE_DOMAIN = Zeus
          @ADMIN_DOMAIN = RTT
          @COUNTRY = BE
          @ZEUS_MRX@VENUS              (Message Router address modifiée)

    FROM: FDUPONT@MARS                 (Message Router address modifiée)
```

Sur VENUS :
Le Message Router de Venus va agir comme celui de Diane et va poster le message dans la mailbox ZEUS_MRX qui avertira MRX.

```
    TO:   SURNAME = Lange
          @GIVEN_NAME = David
          @UNIT_NAME = Finance
          @ORGANIZATION = ORG1
          @PRIVATE_DOMAIN = Zeus
          @ADMIN_DOMAIN = RTT
          @COUNTRY = BE
          @ZEUS_MRX

    FROM: FDUPONT@MARS@DIANE
```

Le Message Router X.400 Gateway va générer les O/R names sur l'enveloppe. Il va rechercher une MR address correspondant à celle de la liste FROM : et si il y en a une, il ajoutera à cette liste l'O/R name de l'expéditeur en se servant des paramètres de l'entrée correspondant à la MR address.

```
    TO:   SURNAME = Lange
          GIVEN_NAME = David
          INITIALS =
          GENERATION =
          UNIT_NAME = Finance
          ORGANIZATION = ORG1
          PRIVATE_DOMAIN = Zeus
          ADMIN_DOMAIN = RTT
          COUNTRY = BE

  FROM:   SURNAME = Dupont              (de la BD des abonnés)
          GIVEN_NAME = Frédéric         (de la BD des abonnés)
          INITIALS =                    (de la BD des abonnés)
          GENERATION =                  (de la BD des abonnés)
          UNIT_NAME = Personnel         (de la BD des abonnés)
          ORGANIZATION =                (de la BD des abonnés)
          PRIVATE_DOMAIN = PRMD1        (des paramètres de contrôle)
          ADMIN_DOMAIN = RTT            (des paramètres de contrôle)
          COUNTRY = BE                  (des paramètres de contrôle)
```

MRX peut maintenant envoyer le message sur ZEUS.
Pour envoyer un message depuis Venus, il suffit de le poster à la mailbox ZEUS_MRX et depuis Diane : ZEUS_MRX@Venus.

3.4.2 Réception de message

Les messages provenant d'autres domaines X.400 à destination d'un ou de plusieurs abonnés du domaine PRMD1 arrivent d'abord sur le Message Router X.400 Gateway du noeud Venus avant d'atteindre leur destinataire final.

David Lange de Zeus a envoyé un message à Frédéric Dupont du domaine PRMD1. Le message arrive sur le noeud Venus.

Noeud VENUS:

```
    TO:   SURNAME = Dupont
          GIVEN_NAME = Frédéric
          INITIALS =
          GENERATION =
          UNIT_NAME = Personnel
          ORGANIZATION =
          PRIVATE_DOMAIN = PRMD1
          ADMIN_DOMAIN = RTT
          COUNTRY = BE

    FROM: SURNAME = Lange
          GIVEN_NAME = David
          INITIALS =
          GENERATION =
          UNIT_NAME = Finance
          ORGANIZATION = ORG1
          PRIVATE_DOMAIN = Zeus
          ADMIN_DOMAIN = RTT
          COUNTRY = BE
```

MRX recherche dans sa base de données des abonnés une entrée correspondant à la liste TO:. MRX va traduire les O/R names en adresses MR, puis il va poster le message en utilisant la mailbox ZEUS_MRX qui correspond au domaine d'où provient le message.

Voici les adresses que MRX donne au Message Router du noeud Venus:

```
  TO: FDUPONT@MARS@DIANE        (Message Router address de l'abonné)

  FROM:   SURNAME = Lange
          @GIVEN_NAME = David
          @UNIT_NAME = Finance
          @ORGANIZATION = ORG1
          @PRIVATE_DOMAIN = Zeus
          @ADMIN_DOMAIN = RTT
          @COUNTRY = BE
          @ZEUS_MRX             (Message Router address d'origine)
```

Le Message Router de Venus va déposer le message dans la mailbox Diane et le message sera envoyé par le Talker.

sur DIANE:
Le Message Router modifie le dernier terme de l'adresse TO: et le délivre à la mailbox du noeud MARS.

```
     TO:  FDUPONT@MARS

    FROM: SURNAME = Lange
          @GIVEN_NAME = David
          @UNIT_NAME = Finance
          @ORGANIZATION = ORG1
          @PRIVATE_DOMAIN = Zeus
          @ADMIN_DOMAIN = RTT
          @COUNTRY = BE
          @ZEUS_MRX@VENUS
```

sur MARS :
Le Message Router efface son nom de la liste TO: et dépose le message dans la mailbox de l'abonné FDUPONT qui sera averti de l'arrivée du message. Nous aurons comme adresses :

```
    TO:   FDUPONT

    FROM: SURNAME = Lange
          @GIVEN_NAME = David
          @UNIT_NAME = Finance
          @ORGANIZATION = ORG1
          @PRIVATE_DOMAIN = Zeus
          @ADMIN_DOMAIN = RTT
          @COUNTRY = BE
          @ZEUS_MRX@VENUS@DIANE
```

4. CHAPITRE IV : ANALYSE ET IMPLEMENTATION D'UN USER AGENT

Le présent chapitre se propose de définir les fonctionnalités que doit offrir un User Agent pour réaliser le service de messagerie inter-personnelle (IPMS), en particulier dans le cadre d'une coopération avec le Message Router. La première partie de ce chapitre est consacrée à la définition d'un UA et de ses différentes fonctionnalités. Nous verrons ensuite quelle a été notre démarche de conception de User Agent. Les références suivantes nous ont guidés tout au long de ce chapitre : <EAN-85>, <LABO-87>, <SHNE-87>, <VAX-MRIF>, <X.400/88>, <X.420>.

4.1 DEFINITION

Un User Agent, ou Agent Utilisateur, est un ensemble de programmes d'application permettant à un utilisateur, typiquement un abonné d'un système de messagerie, de communiquer avec son MTA. Le comportement de la couche User Agent (UAL) est décrit dans la recommandation X.420 du MHS. Elle définit les fonctionnalités que doit remplir un UA pour assurer le service de messagerie inter-personnelle (IPMS).

Les User Agents utilisent le protocole P2 de messagerie de personnes à personnes pour assurer l'échange de messages. Les messages sont transférés entre deux UA par des User Agent Protocol Data Units (UAPDU ou Unités de Données de Protocole d'Agent Utilisateur).

On distingue deux types d'UAPDU : les IM-UAPDU et les SR-UAPDU. Un IM-UAPDU transporte un message entre personnes. Il est généré par une entité UA au nom d'un abonné du système de messagerie. Un IM-UAPDU est composé de l'entête et du corps du message. L'entête est un ensemble de zones ordonnées contenant des informations de contrôle qui caractérisent le message (par exemple : "Subject", "To", "Cc", ...). Le corps du message constitue l'information que l'utilisateur désire communiquer. Il est composé d'une ou plusieurs parties, chacune contenant un type particulier d'information (paragraphes de texte, commentaires vocaux, graphiques, ...).

Un SR-UAPDU (Status Report ou Rapport d'Etat) est un ensemble ordonné de zones contenant des informations relatives à un IP Message. Ces informations sont générées par et destinées à une entité UA. Un SR-UAPDU peut être généré quand un utilisateur a demandé un rapport de livraison à son ou ses correspondants. La recommandation X.420 prévoit deux types de réaction de l'UA récepteur : soit il renvoie automatiquement un SR-UAPDU à l'émetteur dès que le destinataire a lu le message, soit il prévient l'abonné de la demande et le laisse libre d'envoyer un acquittement s'il le désire. L'SR-UAPDU indiquera quelle cause, automatique ou manuelle, à provoqué l'émission de l'acquittement.

Les UAPDU (IM et SR) sont placés par le MTA dans des User Message Protocol Data Units (UMPDU ou Unités de Données de Protocole de Message Utilisateur). Un UMPDU est un élément du protocole P1 et est composé d'une enveloppe et d'un UAPDU (IM ou SR). Les informations contenues dans l'enveloppe sont utilisées par la couche de transfert de messages (MTL) pour router le MPDU vers sa destination.

En dehors des UMPDU, le protocole P1 supporte deux types de Service Message Protocol Data Units (SMPDU ou Unité de Données de Protocole de Message de Service) : les SMPDU d'essai et les SMPDU de rapport de remise.

Un SMPDU Essai ou Probe transporte une demande d'informations sur la

possibilité de remise d'un message à un ou plusieurs UA destinataires. Il n'est constitué que d'une enveloppe.

Un SMPDU de rapport de remise transporte un rapport de livraison ou de non-livraison d'un MPDU à son expéditeur. Ce SMPDU n'est généré que par un MTA. Il est composé d'une enveloppe et d'un contenu. Le contenu indique la cause de non-remise le cas échéant. Le message concerné par le rapport de remise peut également être inclus dans ce rapport, si l'expéditeur en a fait la demande.

On peut consulter, en annexe, la structure complète d'un IP Message et d'un IP Status Report.

4.1.1 Fonctionnalités

Les fonctionnalités d'un User Agent doivent permettre aux abonnés de créer, d'envoyer, de recevoir et de gérer des messages. Un UA communique avec son MTA pour poster ou rechercher des messages. Nous avons choisi de diviser ces fonctionnalités en fonctionnalités de base et fonctionnalités additionnelles. Cette découpe étant purement subjective, elle ne fait partie d'aucune recommandation. Nous expliquons quand cela est nécessaire, les spécificités que doit avoir un User Agent travaillant sur du matériel Digital.

4.1.1.1 Fonctionnalités de base

Nous appelons "fonctionnalités de base", l'ensemble des opérations qui permettent à un utilisateur d'envoyer et de recevoir des messages. Ces fonctionnalités constituent le service minimum que doit fournir un User Agent.

4.1.1.1.1 Création de message

Un User Agent doit permettre à l'abonné auquel il est attaché de composer des messages. Les messages pouvant être composés par un utilisateur sont les messages personnels (IM-UAPDU), les acquittements de réception d'un message (SR-UAPDU). Un UA doit également pouvoir demander à son MTA d'émettre des MPDU de Probe. Notons dès à présent que MRX ne supporte pas les MPDU de Probe, ni les SR-UAPDU. Il ne les génère jamais et leur réception provoque une erreur.

Sur base des conseils de conception d'interface usager définis dans <SHNE-87>, nous pouvons considérer que la création d'un message peut s'effectuer de deux manières : soit suivant un patron, soit selon des mots-clés. La création suivant un patron permet à l'utilisateur de voir le message comme un formulaire de remplissage. Il remplit une à une les zones de son choix, le User Agent se chargeant de vérifier si toutes les informations nécessaires sont présentes. La création par mots-clés consiste à n'afficher que certaines parties du message. Par exemple "To:", l'utilisateur se chargeant lui-même d'écrire les éléments intervenant dans cette partie (To : surname = DUPONT @ given_name = FREDERIC @ ...).

Le choix de l'une ou l'autre des méthodes dépend du profil des abonnés du système de messagerie. Les abonnés novices émettant peu de messages trouveront le schéma de guide par patron très attrayant : ils ne doivent

pas retenir la syntaxe des messages, et pour peu qu'un système d'aide interactive soit fourni, ils pourront vite manipuler l'UA. Par contre, des utilisateurs chevronnés, devant envoyer un grand nombre de messages, trouveront cette méthode trop lente et préféreront un système plus rapide.

Un patron de conception de message se doit d'être compréhensible par tous les utilisateurs. Pour ce faire, il faut veiller à choisir des noms cohérents et familiers pour les champs de message à entrer. Par exemple, nous préférons demander "Surname (string) :", plutôt que "<S> :". La longueur de chaque élément de message se doit d'être visible sur l'écran par l'affichage d'une limite. Quand le nombre d'éléments à encoder est très grand, il faut penser à fournir un mécanisme d'aide interactive pour l'élément sur lequel on est positionné, par exemple en frappant le caractère "?". L'utilisateur doit également être informé du caractère obligatoire (par exemple : (SURNAME)), optionnel ([GIVEN_NAME]) ou répétitif ({ORGUNIT}) des éléments du message.

Nous avons choisi la méthode de remplissage par patron pour des raisons de convivialité.

4.1.1.1.2 Assemblage d'un message

L'assemblage d'un message est l'opération consistant à transformer le message encodé par l'utilisateur dans un format tel qu'il puisse être interprété par le MTA. Le mécanisme d'assemblage est en général transparent à l'utilisateur.

Rappelons que si un User Agent encode un message en format X.409, les User Agents travaillant avec le Message Router et le Message Router X.400 Gateway doivent assembler leurs messages en format NBS, puisque ces messages sont d'abord destinés à être traité par le Message Router. C'est le Message Router X.400 Gateway qui se charge de la conversion du format NBS en X.409, cette conversion étant également transparente à l'utilisateur.

Le Message Router permet, lors de l'assemblage du message de construire l'enveloppe et d'insérer l'UAPDU dans un MPDU.

4.1.1.1.3 Soumission d'un message

La soumission de message consiste à poster un message à un MTA, pour l'expédier à un autre utilisateur. Il est entendu que le MTA doit recevoir un message dans un format qu'il puisse interpréter. Une fois le message posté, le MTA en assure la responsabilité. Il utilise les éléments contenus dans l'enveloppe pour assurer le routage du message vers son destinataire.

Pour réaliser la soumission d'un message au Message Router, le User Agent doit tout d'abord commencer une session de travail, c'est à dire prévenir le Message Router qu'il désire entrer en communication avec lui. Ensuite l'utilisateur doit s'identifier au Message Router en donnant son nom de mailbox MR. Une fois l'identification réussie, la soumission de message peut avoir lieu.

4.1.1.1.4 Acceptation d'un message

L'acceptation de message est l'acte par lequel un User Agent, à la demande de son utilisateur, ou de manière automatique, va rechercher les messages adressés à son utilisateur. Les messages peuvent aussi bien être des messages d'autres utilisateurs, que des messages de service produits par d'autres UA (SR-UAPDU) ou d'autres MTA (SMPDU). En acceptant un message, le User Agent délivre le MTA de sa responsabilité sur ce message.

4.1.1.1.5 Désassemblage d'un message

Le désassemblage d'un message est l'opération inverse de l'assemblage : elle consiste à traduire un message d'un format compréhensible par le MTA (NBS ou X.409) en un format compréhensible par les utilisateurs. Le désassemblage doit pouvoir traduire tous les types de messages qu'un User Agent est susceptible de recevoir.

Les messages placés par le Message Router dans une mailbox d'abonné sont des UMPDU (enveloppe et contenu). Un User Agent qui collabore avec le Message Router doit donc être capable de désassembler des UMPDU contenant un SR-UAPDU ou un IM-UAPDU ainsi que des SMPDU d'essai ou de rapport de remise.

4.1.1.1.6 Affichage d'un message

L'affichage d'un message est l'action de présenter un message à un utilisateur dans un format qui lui est compréhensible et sur un support (terminal, fichier, ...) de son choix.

4.1.1.2 Fonctionnalités additionnelles

Les fonctionnalités additionnelles constituent l'ensemble des opérations permettant à l'utilisateur de gérer les messages. Nous en décrivons ici quelques unes, étant entendu que cette liste ne saurait être considérée comme exhaustive.

4.1.1.2.1 Edition de messages

L'édition de messages consiste à offrir des facilités fournies généralement par les systèmes de traitement de texte pour faciliter la mise en forme des messages. L'utilisateur peut ainsi composer son message et en corriger certains éléments avant de l'envoyer.

4.1.1.2.2 Réponse automatique

Quand le User Agent reçoit un message, soit un UMPDU véhiculant un IM-UAPDU (un message personnel), soit un SMPDU de Probe, il vérifie si un rapport de remise a été demandé par l'expéditeur. Dans l'affirmative, il génére un SR-UAPDU d'acquittement du message.

Une alternative à cette génération automatique d'SR-UAPDU consiste, comme nous l'avons déjà dit, à signaler à l'abonné qu'une réponse est attendue pour ce message.

4.1.1.2.3 Gestion d'alias

Comme nous l'avons vu au chapitre précédent, les adresses d'abonnés des systèmes de messagerie sont rarement simples à mémoriser, en raison de leur nombre, de leur diversité et de leur longueur. Il peut paraitre indispensable de prévoir une fonction qui à un nom mnémonique simple fait correspondre un O/R name complet. Le User Agent parcourra une base de données privée chaque fois que l'abonné encodera un nom simple.

Exemple 4.1 : Agenda privé.

L'utilisateur entre "<FDUPONT>" au champ "TO: "

Son UA en déduit :

```
To : surname=DUPONT @ given_name=FREDERIC @ unit=INFO
     @ private_domain=FNDP @ administration=RTT
     @ country=BE @ FNDP_MRX
```

La base de données "alias" peut faire correspondre des noms mnémoniques à des structures d'O/R name déjà assemblées. Dans ce cas l'UA n'aura qu'à les ajouter telles quelles au message. Une correspondance avec une structure d'O/R name affichable permettra cependant à l'utilisateur de contrôler si celui-ci est bien l'O/R name souhaité et donc de pouvoir le corriger le cas échéant.

Une base de données "alias" peut être privée ou accessible à tous les abonnés d'un site.

4.1.1.2.4 Gestion d'une base de données de messages

Il peut être intéressant pour un abonné de disposer d'un gestionnaire de messages qui lui permet de classer les messages qu'il a reçu et/ou émis. Cet utilitaire devrait être assez intelligent pour effectuer des liaisons et des références croisées entre messages. Par exemple : retrouver les messages provenant d'un abonné donné depuis une date donnée, les messages traitant d'un sujet particulier, annuler les messages rendus obsolètes par l'arrivée d'un nouveau,...

4.1.2 Motivations

Ce paragraphe constitue une justification de la démarche de conception de User Agent que nous avons suivie.

Notre démarche a tout d'abord été guidée par des impératifs d'ordre pratique. En effet, avec le Message Router doit normalement être livré le VMS/Mail Gateway et le Message Router Interface Routines. Le VMS/Mail Gateway qui, rappelons-le constitue un interface entre le VMS/Mail et le Message Router, permet d'utiliser les fonctionnalités du VMS/Mail comme User Agent pour le Message Router. Or ce produit ne fut pas livré à temps. Dès lors il nous était impératif de construire nous-même un User Agent simple, mais extensible, qui nous permettrait de tester les fonctionnalités du Message Router et du Message Router X.400 Gateway et de prendre ainsi contact rapidement avec ces logiciels.

Le but de notre travail étant de mettre au point des outils de test d'un

logiciel X.400, il nous a fallu ensuite concevoir et développer un User Agent orienté vers les tests de conformité et d'interopérabilité. Les spécifications du User Agent de test se trouvent au chapitre 5 "VERIFICATION DE PROTOCOLES".

4.2 SPECIFICATION DU USER AGENT

Le User Agent développé doit permettre la création d'un message destiné à un autre utilisateur, local ou à distance, et la réception d'un message provenant d'un autre abonné.

Le User Agent offre des fonctionnalités de création, d'assemblage et de soumission de messages d'une part, d'acceptation, de désassemblage et d'affichage d'autre part.

Le User Agent traite des messages encodés en format NBS. En effet, comme indiqué au chapitre II (FIG II.1), les User Agents sont connectés d'abord au Message Router, qui ne "comprend" que ce format. Les messages à destination d'abonnés locaux seront traités par le Message Router uniquement, tandis que les messages destinés à des domaines X.400 transiteront par le Message Router X.400 Gateway qui en assurera leur traduction en X.409.

Les arguments en entrée du User Agent sont les éléments d'un en-tête et d'un corps. Les éléments de l'en-tête constituent l'O/R name du destinataire du message (nom, prénom, initiales, génération, téléphone, localisation, pays, domaine public, domaine privé) ainsi qu'une Message Router Address. Le corps est constitué d'une suite de lignes encodées au terminal et d'un fichier texte.

Ces arguments possèdent les propriétés suivantes : le nom, le prénom et la Message Router Address sont obligatoires, les autres éléments sont optionnels. La Message Router Address doit correspondre à une mailbox de la Directory du Message Router. Cette mailbox est celle d'un abonné local ou d'un domaine X.400 accessible via le Message Router X.400 Gateway et sur lequel se trouve l'abonné décrit dans l'O/R name.

Dès que le message a été encodé, le User Agent démarre une session de travail avec le Message Router en s'identifiant au moyen de la mailbox de l'expéditeur. Si l'identification réussit, il poste le message dans cette mailbox. Si le Message Router accepte le message, il renvoie au UA un numéro d'identification qui sera affiché à l'écran par le User Agent.

Des situations d'exception se produisent quand le User Agent ne parvient pas à établir une session avec le Message Router (ce qui provoque l'arrêt du programme), quand l'identification à la mailbox associée par défaut au compte où s'exécute le UA échoue (le UA demande alors à l'utilisateur de lui donner un nom de mailbox et éventuellement, un mot de passe valide), et enfin si le User Agent ne parvient pas à trouver le fichier texte spécifié par l'utilisateur. Celui-ci sera prié de donner un nouveau nom de fichier.

Le Message Router se charge de livrer le message dans la mailbox spécifiée dans la clause Message Router Address.

4.2.1 Architecture logique

La figure IV.1 décrit l'architecture logique du User Agent.

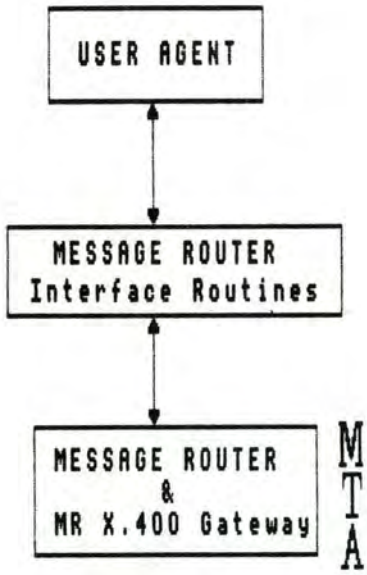


FIG IV.1 : Architecture Logique.

Le Message Router Interface Routines est un produit logiciel qui permet la construction de User Agents ou de passerelles. La description complète de ces routines se trouve dans <VAX-MRIF>.

Les fonctionnalités du MRIF sont :

- établir ou terminer une session de travail entre un programme d'application et le Message Router,
- démarrer ou terminer l'assemblage ou le désassemblage d'un message,
- assembler ou désassembler un élément de message (chaine de caractères, entier, date, fichier texte, ...),
- sélectionner le mode d'assemblage d'un message : un utilisateur peut assembler une enveloppe seule, un contenu seul, une enveloppe et un contenu ou un contenu à partir duquel le Message Router construira l'enveloppe lui-même,
- identifier un utilisateur au Message Router,
- déterminer le nombre de messages se trouvant dans une mailbox,
- rechercher un message pour le lire,
- poster un message assemblé au Message Router.

4.2.1.1 Définition des modules du User Agent

Le User Agent se compose de deux modules fonctionnels : un module d'envoi de messages (**SEND**) et un module de lecture de messages (**READ**).

4.2.1.1.1 Module SEND

Le module SEND permet à un abonné de créer un message à destination d'une autre personne. La création d'un message se fait en parallèle avec l'assemblage : dès qu'un élément est encodé par l'utilisateur, celui-ci est ajouté au message par la routine MRIF correspondante.

Le module SEND permet la création d'un IM-UAPDU. Le Header de cet UAPDU est composé de l'O/R name du destinataire du message et de la mailbox MR où doit d'abord être livré le message. Cette mailbox sera celle d'un abonné au site local ou celle d'un domaine éloigné, auquel cas elle devra être attachée au Message Router X.400 Gateway par la clause /notify=MRX$NOTIFY.

Une fois le message créé, le module SEND démarre une session de travail avec le Message Router : il tente d'abord d'utiliser la mailbox associée par défaut à l'expéditeur. Si cette manoeuvre échoue, il demande à l'utilisateur de donner un nom de mailbox MR valide. Si l'identification a réussi, le module SEND poste le message au Message Router et ensuite libère la session de travail. Une fois le message accepté par le Message Router, celui-ci se chargera de le déposer à la mailbox spécifiée par l'utilisateur.

Enfin, le module SEND crée deux fichiers : 'ENVELOPE.NBS' et 'CONTENT.NBS' qui contiennent respectivement l'enveloppe et le contenu du message codés dans le format NBS. Ce sont des fichiers de travail utilisés par les routines MRIF.

4.2.1.1.2 Module READ

Le module READ permet à un utilisateur de consulter une mailbox en lui indiquant le nombre de messages s'y trouvant éventuellement. A la demande de l'utilisateur, il recherche les messages dans leur ordre d'arrivée (FIFO), les désassemble et les lui présente sur son terminal ou dans un fichier texte. Le module READ permet la lecture et le désassemblage de tous les éléments de tous les types de messages que ce soient des UMPDU ou des SMPDU. Lorsqu'il lit un SMPDU de non-remise, le module en indique la cause.

Le désassemblage s'effectue en une passe : d'abord l'enveloppe, puis le contenu. Les routines de désassemblage du MRIF permettent de distinguer un SMPDU d'un UMPDU. Les éléments du message sont désassemblés un à un et placés sur le support spécifié par l'utilisateur. Un élément de message est traduit sous la forme : <item> : <valeur> où <item> est le nom de l'élément du message et <valeur> sa valeur. <valeur> est toujours différente de la chaine vide, ce qui veut dire que seuls les éléments présents dans le message sont affichés sur le support.

4.2.2 Architecture physique

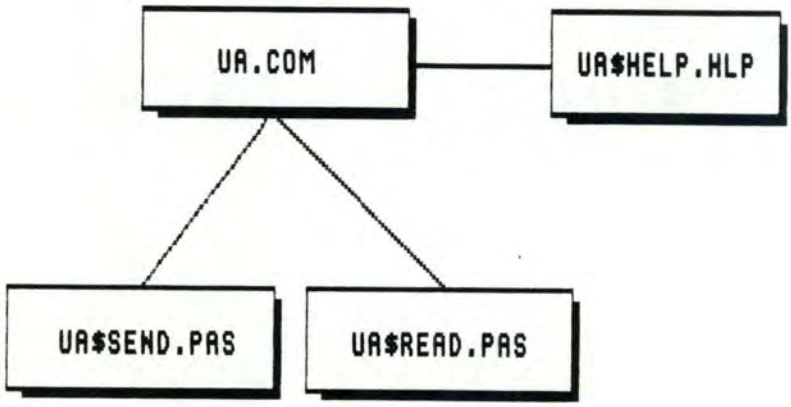


FIG IV.2 : Architecture physique.

L'architecture physique du User Agent (fig IV.2) est constituée d'une procédure de commande DCL (UA.COM) qui agit comme coordinateur. Cette procédure permet de démarrer à la demande de l'utilisateur l'un des deux programmes Pascal UA$SEND ou UA$READ. Un service d'aide interactive est offert à l'utilisateur au niveau de la procédure (UA$HELP.HLP).

4.2.3 Exemple d'utilisation

Nous présentons ci-après un exemple de session de travail effectuée avec le User Agent.
On trouvera en annexe dans le manuel d'utilisateur **"UA USER GUIDE"** un exemple plus documenté ainsi que la description complète du User Agent développé.

Les phrases en **GRAS** signalent les commandes ou les données entrées par l'utilisateur.

```
$ @UA

UA - X.400 USER AGENT - v1.1
(c) DRX & ME jan'88.

UA> send

This program allows you to create and to post a message to a local or
remote user.  It will prompt you for some informations.  If you want to
accept a default or skip an optionnal information, simply press
<RETURN>, otherwise type your reply and press <RETURN>.

Given name of person to send to      : MARC
Surname of person to send to         : ELOY
Initials [optional]                  : ME
Generation [optional]                :
Telephone [optional]                 :
Location [optional]                  :
Country [BE]                         :
Administration domain [RTT]          :
Private domain [IIHE]                : FNDP
Message Router Address               : FNDP_MRX

Subject : Nouvelle version du UA.
Enter your text below, "." when ready.

La nouvelle version de UA$SEND est prête, je te l'envoie dans ce
message.
DRX.
.

Text filename to add to the message [optional] : [MRMANAGER]UA$SEND.PAS

Connecting to the default node ...
Using your default mailbox ...
Identified to mailbox DREUVIAUX
Message posted successfully with message id 015627172108891/436@BVX82
```

Après avoir démarré le User Agent par la commande : **UA**, l'utilisateur décide d'envoyer un message (**SEND**). Une fois son message encodé, le User Agent signale qu'il démarre une session de travail avec le Message Router (**Connecting to the default node**), qu'il essaye de s'identifier en utilisant la mailbox associée par défaut au compte où il se trouve (**Using your default mailbox**). L'identification ayant réussi (**Identified to mailbox DREUVIAUX**), le User Agent poste le message. Le User Agent signale enfin que le Message Router a accepté le message et lui a attribué un numéro d'identification (**Message posted successfully with message id ...**).

READ permet à l'utilisateur de lire une mailbox quelconque pour peu qu'il en connaisse le nom et le mot de passe. Dans l'exemple ci-dessus, l'utilisateur répond à Mailbox et Password par blanc, ce qui oblige le User Agent à s'identifier par la mailbox associée par défaut au compte où il se trouve.

```
UA> read

Connecting to the default node ...
Mailbox [default mailbox]              :
Password [optionnal]                   :
Mailbox [DREUVIAUX] contains      1 new message.
Do you wish to read it/them ? [Y]      : Y
Output file [DREUVIAUX01.MHS]          : tt:

Message nr  1 fetched successfully.

MESSAGE NR  1 :
MESSAGE-ENVELOPE
  Message_ID                  : 54636102108891/322@BVX82
  PDATE                       : 19-JAN-1988 15:03:32.00
  TO
    Country           : BE
    Amdname           : RTT
    Prdname           : IIHE
    Surname           : REUVIAUX
    Given_name        : DANIEL
    MR Address        : DREUVIAUX
  PERRECFLG           : %X000000A8
    action            : Y
    mrbasic           : Y
    mrconfirmed       : N
    uabasic           : Y
    uaconfirmed       : N
  SENDER
    MR Address        : MSPELTENS
  HOPCOUNT            : %X00000001
  CONTENTTYPES        : %X00000001

MESSAGE-CONTENT
  SUBJECT             : Souper HELIOS.
  FROM
    MR Address        : MSPELTENS
  TO
    Country           : BE
    Amdname           : RTT
    Prdname           : IIHE
    Surname           : REUVIAUX
    Given_name        : DANIEL
    MR Address        : DREUVIAUX
  TEXT
    Je te confirme ma participation au souper HELIOS ce vendredi.
    J'espère t'y voir également.
    Marc.

UA> exit
End of UA.
```

Le User Agent signale à l'utilisateur le nombre de message(s) en attente dans sa mailbox (**Mailbox [DREUVIAUX] contains 1 new message**) et lui

propose de les lire. Le résultat de la lecture d'un message se fait soit dans un fichier, soit à l'écran (option TT:). Le User Agent crée par défaut un fichier dont le nom est <mailbox_name><message_number>.MHS, où <mailbox_name> est le nom de la mailbox où a été lu le message et <message_number> le numéro d'ordre du message dans la mailbox. L'utilisateur peut cependant choisir un nom qui lui est propre et lui facilite le rangement. Si le message est affiché au terminal, celui-ci ne pourra être lu qu'une fois et sera perdu après la lecture.

Le message lu ici a été envoyé depuis le noeud local (MR Address : MSPELTENS) et se compose d'une enveloppe et d'un contenu. L'envelope a été crée par le User Agent sur base des données encodées par l'expéditeur dans l'entête du message.

4.2.4 Limites du User Agent

Si le module de lecture de message READ peut être considéré comme capable de lire et de désassembler n'importe quel message NBS, le module SEND est lui fortement extensible. Il n'autorise dans sa forme actuelle que l'envoi d'un seul message à un seul destinataire, ce qui exclut les listes de distribution. De plus, il ne garnit que certains éléments de l'enveloppe et du contenu, laissant bon nombre d'options (confirmation de livraison entre-autres) non-définies.

La fonction de classement se limite à ranger le message lu dans un fichier texte dont la gestion devra être supportée par l'utilisateur.

Nous rappelons que cet outil a été développé dans une optique de prise de contact rapide avec le matériel à notre disposition et constitue une solution provisoire au problème du manque de User Agent. D'autre part ce produit est extensible, dans le sens qu'il peut très bien devenir un User Agent intelligent.

5. CHAPITRE 5 : VERIFICATION DE PROTOCOLES

Dans ce cinquième chapitre, nous rappelons tout d'abord la méthodologie générale de vérification d'un logiciel. Ensuite, après avoir donné une typologie des différents tests, nous analysons les aspects théoriques et pratiques des tests de conformité aux protocoles ISO et en particulier aux recommandations X.400. Après une approche de quelques organismes de test, nous présentons le logiciel "MRXTESTER" que nous avons conçu et la méthode que nous avons choisie pour les tests. Enfin, nous concluons par l'analyse des résultats obtenus.

5.1 METHODOLOGIE

5.1.1 INTRODUCTION

Qu'il nous soit permis, dans cette première partie, de rappeler les raisons et principes des tests de logiciels et de mettre cette méthodologie en rapport avec le problème posé, à savoir la vérification d'une implémentation X.400.

5.1.1.1 Le problème posé : VERIFICATION D'UN X.400

Notre but est, par des méthodes diverses, de répondre à la question suivante : "Le produit logiciel X.400 mis à notre disposition est-il conforme aux spécifications ?"
Il s'agit dans ce cas précis du logiciel DIGITAL implémentant les recommandations X.400 du CCITT (messagerie électronique).

5.1.1.2 Pourquoi vérifier un logiciel X.400?

En plus des raisons habituelles à la vérification d'un produit logiciel, nous devons réfléchir aux caractéristiques particulières d'un logiciel implémentant la norme X.400.

A cause de la complexité des normes proposées, les implémenteurs ont du faire face à de nombreux problèmes et ont eu sur de nombreux points leur propre interprétation. Ils ont introduit des choix, définit des options et ajouté des caractéristiques propres (Implementor defined).

Le CCITT, lui-même, a édité un guide, le "X.400 Implementor's Guide" <X.400/IG>, qui corrigeait les erreurs découvertes et comblait les vides existant dans la norme.

De plus, on a pu remarquer des différences dans les standards proposés par les organismes de standardisation (CEN/CENELEC et CEPT pour l'Europe, NBS pour les USA et NTT pour le Japon). Nous conseillons de consulter le rapport comparatif de Charles FOX <FOX-87> sur ce sujet.

Le but de l'OSI sera complètement réalisé quand les systèmes implémentant le modèle ISO pourront être testés pour déterminer jusqu'à quel point ils sont conformes avec les différents protocoles qu'ils implémentent.

5.1.2 Méthodes de vérification

5.1.2.1 Trois approches possibles

5.1.2.1.1 Les TESTS

Les TESTS permettent de détecter la présence d'erreurs, ils ne prouvent pas la fiabilité absolue (absence d'erreur) d'un système. Trois phases sont nécessaires dans cette approche :

1. la CONCEPTION des tests
   On choisit de manière judicieuse les ensembles des valeurs des données en entrée du produit.
2. la CONDUITE des tests
   On exécute le produit logiciel sur base des jeux de test définis par la phase 1.
3. l'ANALYSE des tests
   On confronte les résultats obtenus avec ceux attendus.

5.1.2.1.2 La VERIFICATION

La VERIFICATION d'un produit logiciel permet d'établir l'absence d'erreur par une preuve, une démonstration de la cohérence entre le texte de ce produit et ses spécifications.

5.1.2.1.3 La DERIVATION de produits corrects par construction

La DERIVATION de produits corrects par construction essaye d'obtenir à la fois le produit logiciel et une démonstration de sa fiabilité.

5.1.2.2 Comparaison des trois approches

5.1.2.2.1 TESTS

Cette méthode ne permet jamais de prouver l'absence d'erreur. Par des tests, on decouvre uniquement les symptômes mais pas les causes des problèmes. On teste le produit dans sa totalité (Différents modules et environnement). L'application de la méthode est assez évidente.

5.1.2.2.2 VERIFICATION et DERIVATION

Malgré leur très grande efficacité, ces méthodes ne sont valables que pour des produits de taille réduite et dont on connait toutes les spécifications. De plus, elles sont difficilement automatisables.

5.1.2.3 Choix

Pour vérifier la fiabilité du produit X.400 de DIGITAL, nous avons choisi, après analyse des différentes approches et du problème posé, d'utiliser les TESTS.

5.1.3 THEORIE DES TESTS

5.1.3.1 Objectif d'un test

L'objectif du test est de trouver un maximum d'erreurs. Un test fructueux est un test débouchant sur la mise en évidence d'un grand nombre d'erreurs. On peut dire qu'un test est un PROCESSUS DESTRUCTIF alors que les approches vérification et dérivation sont des PROCESSUS CONSTRUCTIFS.

5.1.3.2 Typologie de tests

On peut distinguer différents types de tests.

5.1.3.2.1 BLACK BOX TESTING

Un test BLACK BOX se fait sur base d'une analyse du texte des spécifications fonctionnelles ou des spécifications des modules. Il se compose d'analyses par cas sur les propriétés des arguments et des résultats.

5.1.3.2.2 WHITE BOX TESTING

Un test WHITE BOX se fait sur base d'une analyse du texte des algorithmes. Il se compose d'analyses par cas sur les chemins d'exécution de l'algorithme considéré.

5.1.3.2.3 TEST D'INTEGRATION

Un test d'INTEGRATION se fait sur base de l'architecture physique, il permet de tester la bonne coopération des différentes parties du logiciel. Il se compose d'analyses par cas sur toutes les combinaisons possibles d'exécution des différents modules physiques intégrés.

5.1.3.3 Démarche choisie

Notre but étant de donner une méthode de test permettant d'évaluer le respect des recommandations X.400 dans des produits logiciels de messagerie électronique (supposés conformes), nous choisirons donc les tests BLACK BOX car ils analysent le concordance entre un produit, la messagerie X.400 et ses spécifications, les recommandations du CCITT.

Notre travail portera plus particulièrement, sur l'implémentation X.400 réalisée par DIGITAL, à savoir l'ensemble formé par les produits MR (Message Router) et MRX (Message Router X.400 Gateway).

5.1.3.4 Conception d'un test BLACK BOX

5.1.3.4.1 But

Le but d'un test BLACK BOX est de couvrir toutes les relations de cause à effet.

5.1.3.4.2 Classes d'équivalences

On va sérier les problèmes en différentes classes, dites classes d'équivalence, identifiées sur les entrées (valeurs des arguments) et indépendantes les unes des autres.

Exemple 5.1 :

Pour la fonction factorielle FACT(n), selon ses spécifications, on peut définir les classes suivantes :

Résultats attendus :

pour	n < 0	ERREUR
	n = 0	FACT(0) = 1
	n >= 1	FACT(n) = n * (n-1) * ... * 2 * 1

Les classes seront donc "n<0", "n=0" et "n>=1".

5.1.3.4.3 Différentes méthodes

Nous pouvons définir les classes d'équivalence selon différentes méthodes ou critères.

5.1.3.4.3.1 Critère I
On couvre toutes les classes d'équivalence identifiées à partir des propriétés, associées aux arguments en entrée du module, définies dans la spécification de ce module.

5.1.3.4.3.2 Critère II
On couvre chaque cas limite dans les propriétés des arguments des modules (frontières de classes).

5.1.3.4.3.3 Critère III
On couvre toutes les relations de cause à effet pouvant être détectées par l'analyse des spécifications. Les causes sont les éléments correspondants à des sous-propriétés sur les données. Les effets sont les éléments correspondants à des sous-propriétés sur les résultats.
On va considérer toutes les combinaisons possibles des sous-propriétés des données en y faisant correspondre toutes les sous-propriétés des résultats.

5.1.3.4.4 CLASSES CHOISIES

Nous pouvons considérer le produit logiciel X.400 de DIGITAL comme étant une boite noire composée d'un seul module. Les spécifications de ce module sont les recommandations X.400 du CCITT.

Les arguments de ce module sont des MESSAGES.
On distinguera les messages selon leur type (service ou utilisateur) et selon leur contenu (obligatoire ou optionnel).

Les résultats de ce module sont des REPONSES.
Ces réponses peuvent être :
- la livraison d'un message
- une notification de livraison ou de non-livraison
- la génération d'un message d'erreur
- la réception d'un message

5.2 Typologie des TESTS

Nous présentons ci-dessous les différents types de tests réalisables.

5.2.1 Tests de spécification et de codage

Comme pour tout développement de logiciel, nous devons faire attention aux erreurs de spécification et de codage, et ceci, le plus tôt possible.

5.2.2 Tests de conformité et de coopération

Ces tests concernent la vérification du respect des spécifications, données par la norme X.400, par l'implémentation à tester. Cette vérification doit se faire tant au niveau des services offerts qu'au niveau des protocoles à respecter.

5.2.3 Tests de sécurité

Les tests de sécurité ont pour but d'estimer le contrôle d'accès aux ressources du système de messagerie électronique. Ils comportent l'analyse des mesures de protection (identification des utilisateurs, gestion des mots de passe) pour la connexion aux UA et MTA ainsi qu'une estimation du respect des privilèges et priorités des différents utilisateurs (locaux ou éloignés). Ces tests sont à rapprocher de la gestion globale de la sécurité dans un système informatique complet.

5.2.4 Tests de performance

Ces tests ont pour but de mesurer, dans différentes conditions, la consommation de ressources faite par l'implémentation, son rendement (throughput) et son temps de réponse.

La performance d'une implémentation X.400 dépend des performances des couches sous-jacentes (1 à 6) et de la charge du réseau et des MTA employés pour faire transiter les messages entre les sites.

La performance d'une implémentation est souvent déterminée par un ensemble de données recueillies par simulation ou analyse du comportement réel du système à tester en utilisant des "benchmarks" (ensembles de routines standards). Les résultats obtenus varient en fonction de la charge du réseau (légère, moyenne ou élevée), du type de ligne utilisée (dans le réseau et lors de l'accès à celui-ci), de la configuration et de la localisation des implémentations.
Malgré qu'il soit assez difficile de chiffrer les résultats obtenus, ils donnent néanmoins un aperçu de la qualité du service offert. On s'attardera lors d'un tel test à deux valeurs particulières :

- le temps de transmission d'un message.
- la durée d'établissement d'une connexion.

5.2.5 Tests de robustesse

Ces tests ont pour but d'estimer la capacité de l'implémentation à faire face et à répondre aux erreurs. Ces erreurs peuvent être internes (générées par l'implémentation elle-même) ou externes (créées par l'environnement). Ces tests regroupent la détection d'erreurs, la sauvegarde d'états stables, la prise de photos du système (AUDIT), la pose de points de contrôle (CHECKPOINTS) et de reprise, et la relance après erreur (RECOVERY).

5.3 TESTS DE CONFORMITE

Après avoir défini une série de termes et de notations, nous définirons la conformité et analyserons plusieurs méthodes de tests de protocoles. Cette partie de notre travail se base principalement sur les documents de standardisation suivants : <ISO-1446>, <ISO-1525>, <ISO-1526> et <X.CNF>. De plus, l'aide fournie par divers ouvrages et articles (tels <ANSA-86>, <BO-CR-86>, <DICK-85>, <FOLL-86>, <HOLM-85>, <HUTT-87>, <KATO-86>, <LONC-86>, <RAFI-86>, <RAVE-85>, <RAYN-82>, <RAYN-86> et <ZENG-85>) nous a été précieuse.

5.3.1 TERMINOLOGIE ET NOTATIONS

Nous rappelons ici quelques définitions importantes. Nous nous en tenons à la notation anglaise pour les termes importants pour des raisons de cohérence avec les références.

5.3.1.1 Implémentation à Tester (IUT) (Implementation Under Test)

L'IUT ou implémentation à tester définit la partie d'un système ouvert réel que l'on va étudier par la méthode de test. L'IUT peut être formé d'une ou plusieurs couches adjacentes du modèle ISO.

5.3.1.2 Système à tester (SUT) (System Under Test)

Le SUT ou Système à tester est le système ouvert réel dans lequel se trouve l'IUT.

5.3.1.3 Exigences Dynamiques (Dynamic Requirements)

Les exigences dynamiques reprennent tous les besoins et les options qui déterminent si un comportement observable est permis par le(s) standard(s) ISO en question.

5.3.1.4 Exigences Statiques (Static Requirements)

Les exigences statiques sont des contraintes qui sont fournies dans les standards ISO pour faciliter l'inter-fonctionnement en définissant les besoins pour des ensembles de facultés d'une implémentation.

5.3.1.5 Facultés d'une IUT

Les facultés d'une IUT concernent l'ensemble des fonctions et des options du ou des protocole(s) utilisé(s) et du service offert par l'IUT.

5.3.1.6 Protocol Implementation Conformance Statement (PICS)

Un PICS est un rapport fait par le fournisseur ou l'implémenteur d'une IUT ou d'un SUT qui signale les possibilités et les options qui ont étés implémentées et celles qui ont étés omises.

5.3.1.7 Protocol Implementation eXtra Information for Testing (PIXIT)

Un PIXIT est un rapport fait par le fournisseur ou l'implémenteur d'une IUT qui contient toutes les informations (en plus de celles données dans le PICS) concernant l'IUT et son environnement de test. Ce document permettra au conducteur du test (Test Operator) de préciser encore mieux les tests à exécuter.

5.3.1.8 Suite de tests

La figure (V.1) montre la structure hiérachisée d'une suite de tests.

FIG V.1 STRUCTURE HIERARCHISEE D'UNE SUITE DE TESTS

Une suite de tests est un ensemble hiérarchisé suffisant de groupes de tests nécessaires à la vérification des tests de conformité (CONFORMANCE) ou d'interconnexion (BASIC INTERCONNECTION) d'une IUT ou d'une couche à l'intérieur de cette IUT. Une suite de tests contient de plus, toutes les informations indiquant l'ordre dans lequel ces groupes doivent être exécutés.

5.3.1.9 Groupe de tests

Un groupe de tests est un ensemble de cas de tests liés par une logique commune.
Un sous-groupe de tests est un sous-ensemble d'un groupe de tests.

Ces groupes et sous-groupes sont utilisés pour aider la mise au point, la compréhension et l'exécution de la suite de tests.

5.3.1.10 Cas de test

Un cas de test possède un but particulier concernant la vérification d'une capacité requise par l'IUT ou le comportement de cette IUT vis-à-vis d'un évènement donné dans un état donné. Un cas de test est défini par une spécification complète et indépendante.

Cette spécification est dite "complète" car on n'a besoin, avant ou après son exécution, de rien d'autre (à part l'analyse des résultats), pour déterminer si oui ou non le but du test a été rempli.
Cette spécification est dite "indépendante" car il est possible de l'exécuter isolément du reste des autres tests apparentés.

Un cas de test possède au-moins une étape de test. Il peut également être accompagné d'une étape de début ou "préambule" qui permet de mettre l'IUT dans un état initial particulier et d'une étape de fin ou "postambule" qui nettoie le système aprés l'exécution du test.

Un cas de test peut être "générique", "abstrait" ou "exécutable".

Un cas de test **GENERIQUE** offre les caractéristiques suivantes :

1. Il est défini indépendamment de la méthode de test utilisée.
2. Il spécifie le but du cas de test.
3. Il spécifie les séquences d'évènements auxquelles correspondent les verdicts "réussi" ("pass"), "raté" ("fail") ou "indéterminé" ("inconclusive").
4. Il définit, de manière informelle, les "préambule" et "postambule" en langage naturel.

Un cas de test **ABSTRAIT** est dérivé d'un cas de test générique. Lié à un protocole particulier, il offre les caractéristiques suivantes :

1. Il spécifie un cas de test dans les termes d'une méthode de test particulière.
2. Il définit de manière plus précise les évènements et les "préambule" et "postambule" spécifiés de manière informelle.

Un cas de test **EXECUTABLE** est dérivé d'un cas de test abstrait et se trouve dans une forme qui permet de l'exécuter sur un système réel pour tester une implémentation réelle.

5.3.1.11 Etape de test

Une étape de test est une subdivision utilisée pour modulariser les cas de tests. Une étape de test est faite d'évènements de test.

5.3.1.12 Evènement de test

Un évènement de test est une unité indivisible de spécification au niveau d'abstraction choisi. (Par exemple, "envoyer ou recevoir un PDU")
Le résultat d'un évènement de test déterminera en général quel sera le prochain évènement.

5.3.2 CONFORMITE

5.3.2.1 Définition

Il est important de bien comprendre le terme de conformité (CONFORMANCE) au sens de l'OSI. La première chose que l'on peut dire est que ce terme est restreint à la conformité d'une implémentation ou d'un système à un ou plusieurs protocoles standards.

Le test de conformité consiste, d'une part, à tester les possibilités et le comportement d'une implémentation et, d'autre part, de vérifier les résultats de ces tests avec les besoins de conformité envers un standard particulier et envers ce qu'un implémenteur affirme offrir comme support du même standard.

Un test de conformité ne s'occupe pas de considération de performance ou de robustesse d'une implémentation. Il ne doit pas donner de jugement sur la réalisation physique des fonctionnalités ni sur l'architecture du SUT.

Le but des tests de conformité est, comme indiqué dans <ISO-1525>, "d'augmenter la probabilité d'interaction de deux implémentations différentes".

La méthode générale des tests de conformité consiste à vérifier que deux implémentations réagissent de la même manière à une même suite de tests. On ne doit pas oublier que, à cause de la complexité des protocoles, pour des raisons économiques et techniques, le test exhaustif est rendu impossible.
Les considérations générales faites sur les tests au paragraphe 5.1.2.2.1 sont donc applicables ici.

5.3.2.2 Conformité

La conformité d'une implémentation ne garantit pas nécessairement l'interaction entre implémentations (INTERWORKING).
Même si deux implémentations sont conformes aux spécifications données dans les standards, elles peuvent ne pas interagir correctement entre-elles.

La bonne interconnexion de deux ou plusieurs systèmes ouverts réels, tous respectant un même sous-ensemble des standards ISO, est facilitée si l'on a tenu compte des aspects liés à l'interopérabilité (INTEROPERABILITY) suivants:

1. Si un standard possède plusieurs versions, les différences entre celles-ci doivent être identifiées et analysées en fonction d'une interaction de deux implémentations de versions différentes.

2. En plus des PICS doivent être fournies les informations concernant :
   - la justification de la non-implémentation de certaines fonctions,
   - la définition des mécanismes additionnels voués à combler les vides ou à corriger un standard,
   - la liste des options non reprises dans l'implémentation,
   - les valeurs limites des différents paramètres et des timers.

De plus, une comparaison des PICS et des PIXIT fournit des renseignements intéressants sur les capacités des systèmes à interagir correctement.

5.3.3 CONFORMITE ET TESTS

En principe, les objectifs des tests de conformité est d'établir si une implémentation respecte les spécifications d'un standard particulier.
En fonction de l'étendue des tests, on peut définir quatre types : tests d'interconnexion de base, tests des fonctionnalités, tests de comportement et tests de délibération de conformité.

5.3.3.1 TESTS D'INTERCONNEXION DE BASE (Basic Interconnection)

Les tests d'interconnexion de base permettent un test limité des particularités principales d'un standard. Ils permettent d'établir si une implémentation posséde une conformité suffisante permettant une interconnexion.

Ils détectent les cas graves de non-conformité et servent de premier filtre avant d'effectuer des tests plus coûteux.

5.3.3.2 TESTS DE FONCTIONNALITES (Capability)

Les tests de fonctionnalités offrent un test limité de chaque exigence statique d'un standard.

Agissant comme un deuxième filtre de test, ils permettent d'établir la liste des fonctionnalités offertes par une IUT et de vérifier leur validité en fonction des exigences statiques et des PICS.

5.3.3.3 TESTS DE COMPORTEMENT (Behaviour)

Les tests de comportement offrent un test en profondeur d'une implémentation sur base de l'ensemble complet des exigences dynamiques spécifiées dans le standard.
Ces tests sont limités car non-exhaustifs du fait de leur nature. De plus, ils sont conçus pour être exécutés de manière indivisible sur un environnement bien défini et isolé et donc toute erreur difficile ou impossible à détecter dans cet environnement, du fait de ses particularités, ne sera pas signalée.

De ce fait, il est raisonnable de considérer une implémentation comme conforme car satisfaisant aux tests aussi longtemps qu'il n'existe pas de preuve du contraire.

Les tests de comportement doivent être conduits en même temps que les tests de fonctionnalités et peuvent être considérés comme prouvant la conformité (jusqu'à preuve du contraire).

5.3.3.4 TESTS DE DELIBERATION DE CONFORMITE (Conformance Resolution)

Les tests de délibération de conformité offrent un diagnostic, aussi définitif que possible, de la conformité ou de la non-conformité d'une implémentation.

5.3.4 PROCEDURE DE TEST

Une procédure de test comprend plusieurs étapes, mélangeant vérification statique de la conformité (sur base de documents) et test actif, qui doivent converger vers l'évaluation complète de l'IUT.

La figure (V.2) montre les différentes étapes de la procédure de test.

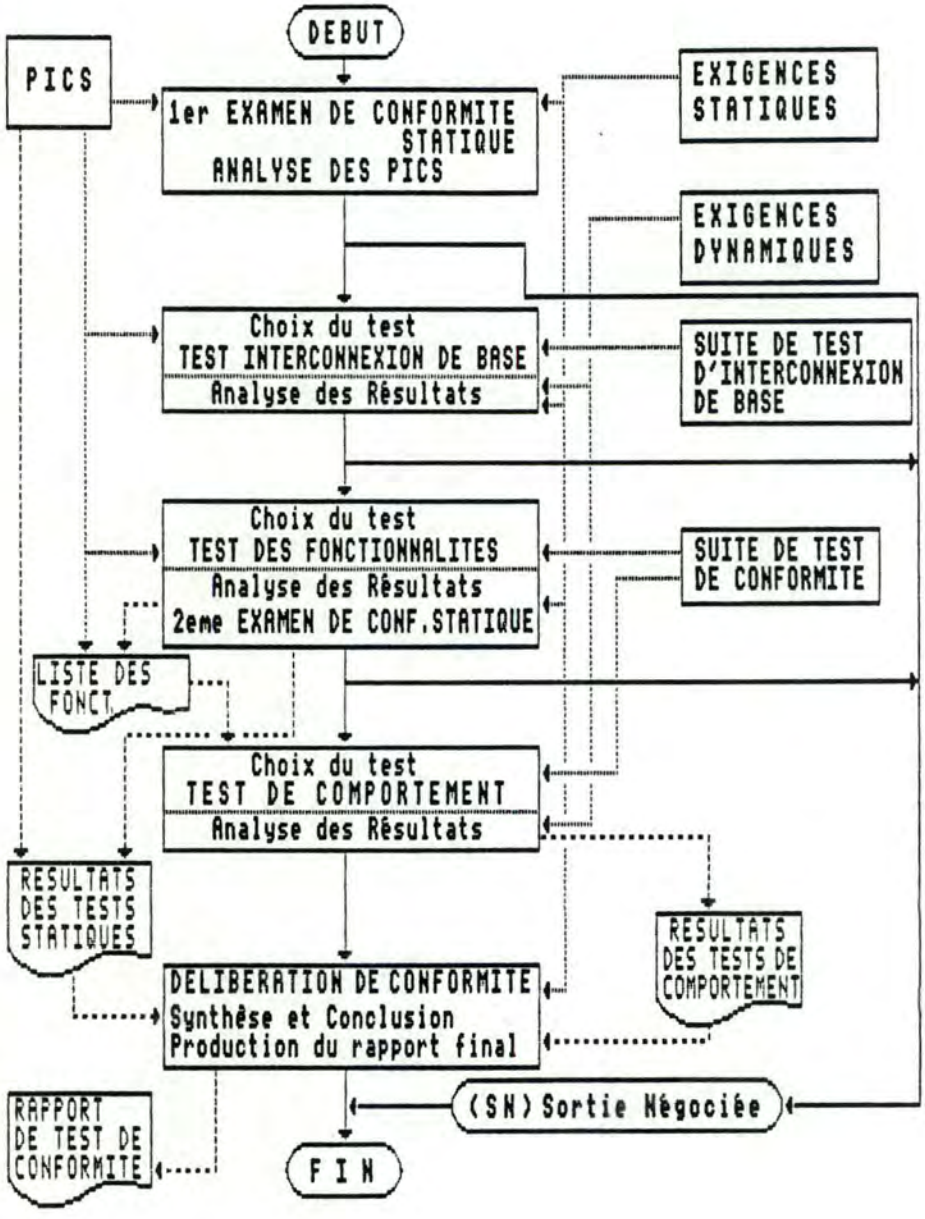


FIG V.2 PROCEDURE DE TEST

1. La première étape analyse la cohérence des PICS vis-à-vis des exigences statiques et génère un rapport appelé "1er examen de conformité statique".

2. Ce travail sur document est suivi des éventuels tests d'interconnexion de base qui permettront de détecter les cas graves de non-conformité sur base des PICS et des suites de tests d'interconnexion.

3. Ensuite, on effectue les tests de fonctionnalités pour vérifier l'adéquation réelle et observable des fonctions du l'IUT avec les PICS. Cette étape produit deux rapports, le premier est constitué de la liste des fonctionnalités réellement supportées par l'IUT, le second appelé "2ème examen de conformité statique" complète celui généré pendant la première étape.

4. Sur base de la liste des fonctionnalités réellement supportée par l'IUT, pourront se faire les tests de comportement.

Les documents appelés respectivement "Résultats des Tests Statiques" et "Résultats des Tests de Comportement" seront rédigés en comparant les résultats des différents tests.

5. La phase de délibération de conformité constitue la dernière étape de la procédure. Elle fait la synthèse des documents pré-cités et donne le verdict final dans le rapport "Rapport de Test de Conformité".

On peut remarquer que la procédure permet des possibilités de "sortie négociée" (SN). Elles représentent les moments auquels les deux parties décident de s'arrêter pour, par exemple, des raisons de non-conformité grave.

5.3.5 METHODES DE TEST

5.3.5.1 Introduction

La principale caractéristique des tests de conformité réside dans le choix d'une méthode adaptée à la configuration du SUT qui permette l'échange d'informations entre l'IUT et le système testeur.

5.3.5.2 Classification des SUT et IUT

Le test de conformité à un standard implique la création d'un ensemble d'outils de test capables d'observer et de diagnostiquer le comportement de l'IUT.

5.3.5.2.1 SUT

La méthode de test est liée à la configuration du SUT. Cette configuration dépend de :
- la fonction principale du système (système-relais (ADMD) ou non),
- des couches ISO implémentées,
- de la possibilité d'utiliser des protocoles non-ISO en parallèle.

La figure (V.3) présente les différentes configurations typiques.

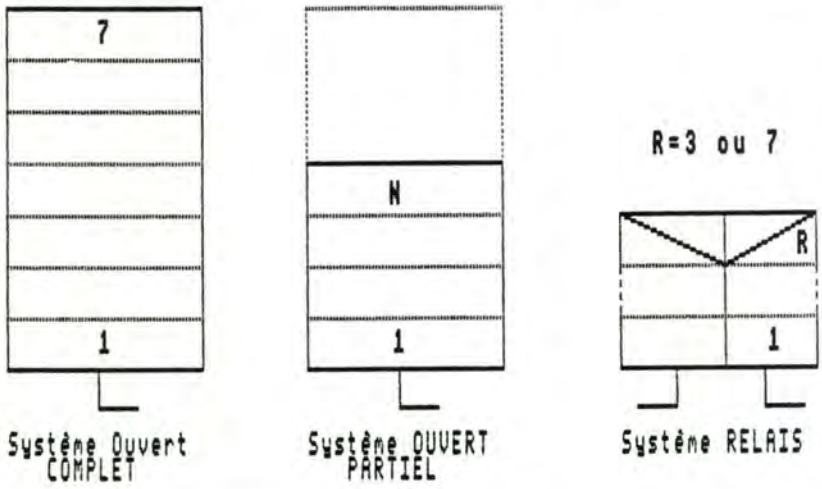


FIG V.3 CONFIGURATIONS TYPIQUES

Un **SYSTEME OUVERT COMPLET** utilise les standards ISO dans ses 7 couches.
Un **SYSTEME OUVERT PARTIEL** utilise les standards ISO dans N couches.
Un **SYSTEME RELAIS** n'implémente que les fonctions de relais et utilise les standards ISO dans les couches inférieures (système de relais réseau) ou dans les 7 couches (système de relais application).
D'autres configurations peuvent être dérivées des configurations de base. On peut, en combinant les deux premières configurations par exemple, définir un système qui permet un choix de protocoles ISO ou non-ISO pour les couches situées au dessus du niveau N (les couches inférieures étant toutes ISO).

5.3.5.2.2 IUT

Une IUT, définie sur un système ouvert complet ou partiel, peut être dite "IUT mono-couche", quand le test ne porte que sur une couche particulière, ou "IUT multi-couche", quand le test porte sur un ensemble de couche adjacentes. Une IUT, définie sur un système relais, sera considérée comme "multi-couche".

5.3.5.3 Architecture conceptuelle de test

L'architecture conceptuelle de test, basée sur les configurations de SUT et d'IUT, doit permettre une observation et un contrôle des entrées et sorties d'une entité à tester le plus proche possible de cette entité.
La figure (V.4) présente l'architecture conceptuelle de test.

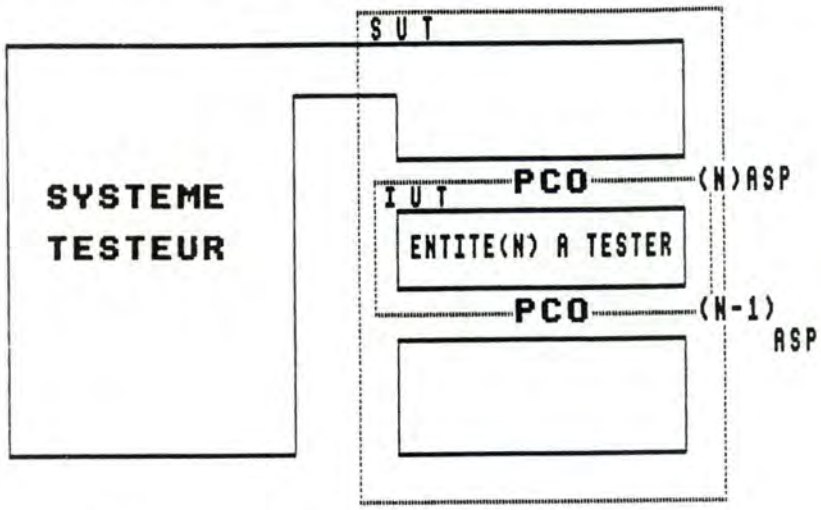


FIG V.4 ARCHITECTURE CONCEPTUELLE DE TEST

Les PCO ou "Point de Contrôle et d'Observation" permettent de contrôler et d'observer, directement ou à distance, deux types d'interactions entre le système testeur et l'entité à tester.
Ces deux types d'interactions sont représentés par les ASP ou ABSTRACT SERVICE PRIMITIVES de niveau N ou N-1. Les ASP définisent le comportement de l'entité à tester en terme d'action/réaction ou d'entrée/sortie. Remarquons que les (N-1)ASP incluent les (N)PDU.

5.3.5.4 Topologie d'un système de test

La figure V.5 présente la topologie d'un système de test.

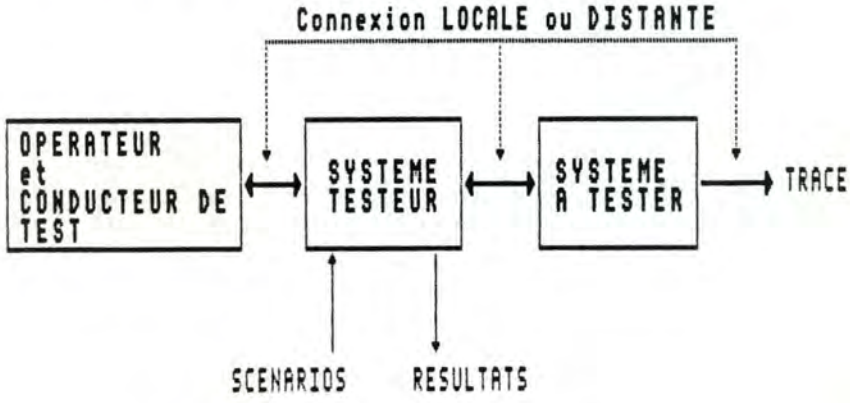


FIG V.5 TOPOLOGIE D'UN SYSTEME DE TEST

Un système de test se compose d'un système à tester (SUT), d'un système testeur et d'un ensemble d'éléments permettant la réalisation des tests.
Ces éléments se composent d'un conducteur de test (manuel ou automatisé), d'un ensemble de scénarios, d'éléments de traçage et de résultats.
De plus, ces objets peuvent se situer dans le même environnement (connexion locale) ou sur des sites distants (connexion remote).

5.3.5.4.1 Méthode locale

Dans la méthode locale (LOCAL ou L), les Points de Contrôle et d'Observation (PCO) sont implantés directement dans le SUT et situés aux frontières de couches de l'IUT. Ils représentent un (des) fournisseur(s) du service (N-1) et un (des) utilisateurs du service (N).

Cette méthode est couramment utilisée pendant la mise au point des logiciels mis en oeuvre dans une entité (N).

Nous numéroterons "Nt" ("Top") la couche supérieure et "Nb" ("Bottom") la couche inférieure d'une IUT.
Pour une IUT mono-couche, "Nt" sera égal à "Nb".

La figure (V.6) présente l'architecture de cette méthode.

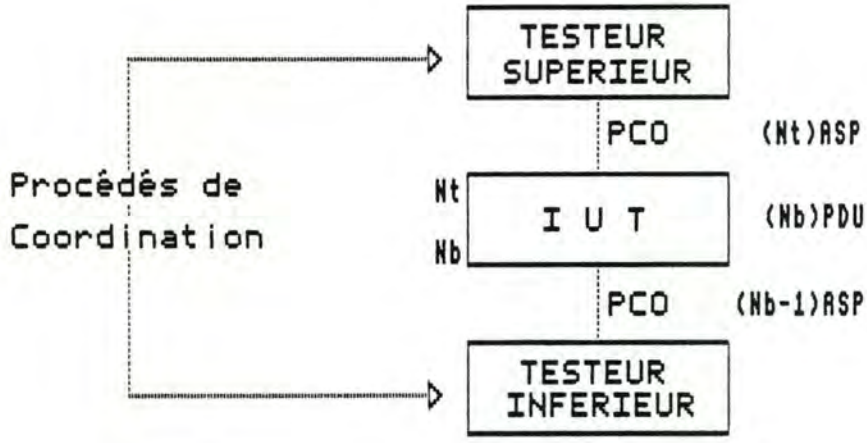


FIG V.6 ARCHITECTURE DE LA METHODE LOCALE

Cette méthode exige l'accès direct (par traduction des ASP en évènements pour l'interface de la couche) ou indirect (par un autre moyen que l'interface de la couche) aux frontières supérieures et inférieures de l'IUT.

La présence de deux testeurs, inférieur et supérieur, nécessite une certaine coordination. Le testeur inférieur sera appelé SIMULATEUR tandis que le testeur supérieur sera appelé REPONDEUR (Test responder).

Le REPONDEUR a pour but de réagir aux demandes qui lui sont faites et d'accepter les résultats qui lui parviennent. Il peut être soit codé statiquement, soit conduit par des données provenant de fichiers locaux ou par des données manuelles locales.

5.3.5.4.2 Méthodes externes

Les méthodes externes contrôlent et observent les (Nb-1)ASP par l'intermédiaire d'un testeur inférieur situé à distance par rapport au SUT.
Une méthode de test externe peut être DISTRIBUEE, COORDONNEE ou ELOIGNEE.

5.3.5.4.2.1 Méthode Distribuée

Dans la méthode distribuée (DISTRIBUTED ou D), un PCO est implanté dans le SUT, et représente un (des) utilisateur(s) du service (N); un autre PCO est implanté dans un système testeur (généralement distinct du SUT) où il simule une (des) entité(s) communiquant avec l'entité (N) à tester.
L'intérêt de la méthode distribuée réside principalement dans la possibilité de mise en oeuvre de Centres de Test.
La figure (V.7) présente l'architecture de cette méthode.

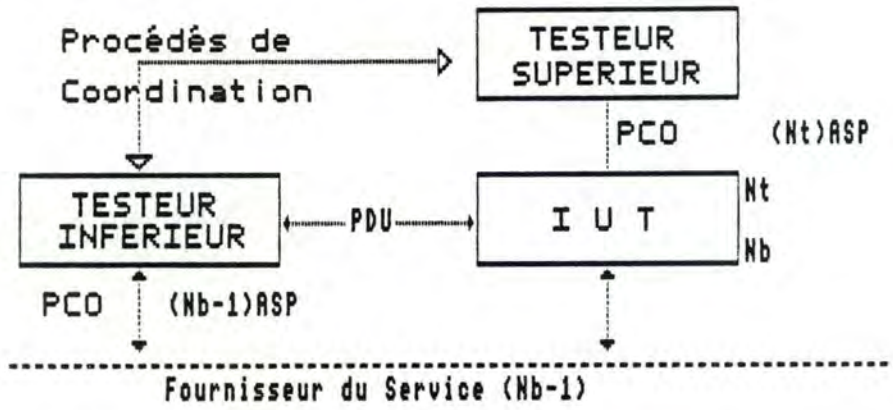


FIG V.7 ARCHITECTURE DE LA METHODE DISTRIBUEE

5.3.5.4.2.2 Méthode Coordonnée

Dans la méthode coordonnée (COORDINATED ou C), un seul PCO est défini. Ce PCO est implanté dans le système testeur, on ne définit donc pas de (Nt)ASP mais bien des TM-PDU ou TEST MANAGEMENT PDU. Le testeur reçoit ces TM-PDU du testeur inférieur et réagit en conséquence.
Il est donc nécessaire d'avoir un protocole standard de gestion de test comme procédé de coordination des testeurs.

La figure (V.8) présente l'architecture de cette méthode.

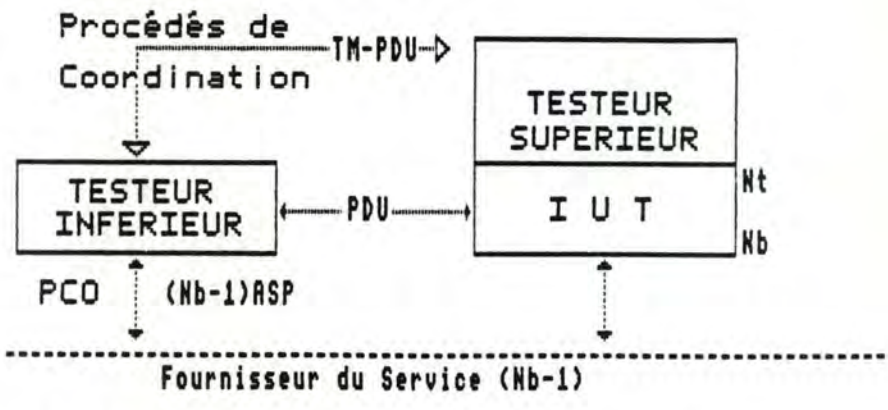


FIG V.8 ARCHITECTURE DE LA METHODE COORDONNEE

5.3.5.4.2.3 Méthode Eloignée

Dans la méthode éloignée (REMOTE ou R), un seul PCO est défini; ce PCO est implanté dans le système testeur, de manière analogue à la méthode distribuée.

Par rapport à la méthode distribuée, la méthode distante possède l'avantage de ne pas nécessiter de développement d'un testeur supérieur.

Elle présente cependant de sérieux inconvénients :

- Les tests sont moins exhaustifs,
- Il est nécessaire d'adapter le testeur inférieur à chaque type de SUT.

Il est donc préférable de limiter son utilisation à des tests de protocoles d'application (auquel cas un opérateur peut remplacer le testeur supérieur).

La figure (V.9) présente l'architecture de cette méthode.

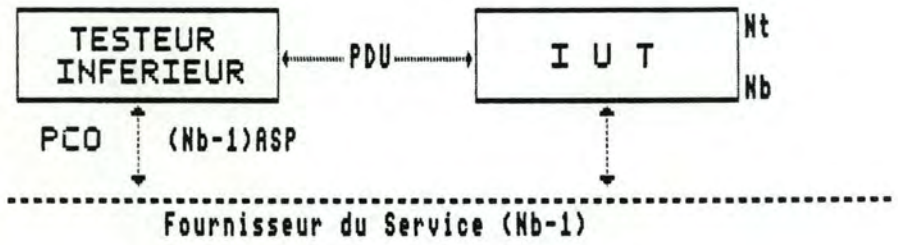


FIG V.9 ARCHITECTURE DE LA METHODE ELOIGNEE

5.3.5.5 Architecture réelle

Après avoir vu l'architecture conceptuelle d'un système de test, analysons quelques architectures typiques dérivées du type de testeur supérieur utilisé.

5.3.5.5.1 Couches paires

La figure (V.10) montre l'architecture en couche paires (PEER LAYERS).

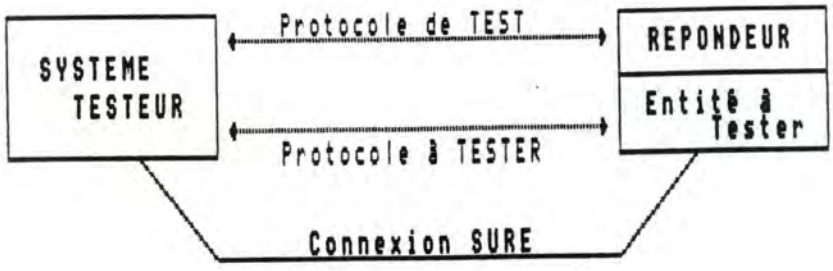


FIG V.10 ARCHITECTURE EN COUCHES PAIRES

Le système testeur contrôle un protocole de test et est guidé par un interpréteur de scénario. Il fonctionne en collaboration avec un système répondeur. Il éxecute le protocole à tester et stimule l'entité (ou la couche) à tester en utilisant ses services. Le système testeur assure le bon fonctionnement des couches inférieures.
La méthode de test en couches paires n'est valable que lorsqu'une seule couche est à tester.

5.3.5.5.2 Multi-couches

La méthode de test multi-couches étend celle des couches paires.
Dans ce cas, deux couches ou plus doivent être testées simultanément par l'opération de la couche à tester la plus haute.
La figure (V.11) montre l'architecture multi-couches (MULTI-LAYERS).

FIG V.11 ARCHITECTURE MULTI-COUCHES

5.3.5.5.3 En escalier

La figure (V.12) montre l'architecture en escalier (STAIRCASE).

FIG V.12 ARCHITECTURE EN ESCALIER

Cette architecture provient de la généralisation du système de test pour le rendre indépendant de la couche sur laquelle il s'applique.

En effet, comme indiqué sur la figure V.12, aussi bien le testeur que le répondeur a accès aux primitives de toutes ses couches inférieures et donc peut appliquer ses fonctions de test indifféremment à une (ou plusieurs) de celles-ci.

5.3.5.5.4 Répondeur à cheval

L'inconvénient de toutes les méthodes de test est le fait que l'IUT est aussi responsable de la transmission des données de protocole.
Bien que cette fonction est susceptible d'être plus facilement corrigée aux niveaux de base, il reste la possibilité d'erreurs non détectées par le système de test. Ces chances sont augmentées lorsque plusieurs couches sont en test. Tout cela détruit la connexion "sure".
Pour contourner cet inconvénient, le répondeur va devoir maintenir simultanément deux connexions et utiliser deux ensembles de primitives de service, le premier pour l'entité à tester et le second pour les couches inférieures déjà testées.

La figure (V.13) montre l'architecture à cheval (ASTRIDE).

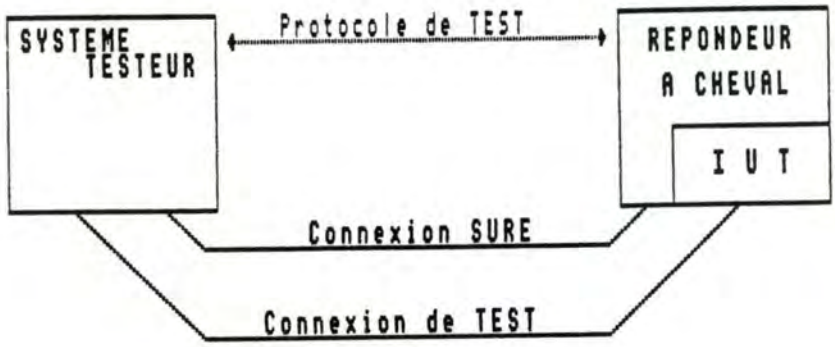


FIG V.13 ARCHITECTURE A CHEVAL

La technique dite 'à cheval' posséde les avantages suivants :

- Aucun stockage de fichier local à l'IUT n'est nécessaire
- La structure générale est par definition plus fiable, surtout lorsque plusieurs couches sont en test.

Bien entendu, le prix à payer est celui du maintien de deux interfaces différents ainsi que de deux connexions.

5.3.6 PARTICULARITES DES TESTS DE PROTOCOLES X.400

Après avoir analysé la méthodologie de test générale aux protocoles ISO, nous allons, sur base des documents <BEYS-87>, <GENE-86>, <SPAG-GUS>, <SPAG-IOG>, <X.CNF> et surtout <X.403>, voir son application aux recommandations X.400.

5.3.6.1 CONFORMITE D'UN X.400

5.3.6.1.1 X.403 : Conformance Testing

Selon <X.403>, un système conforme au service IPM X.400 se doit de supporter correctement :

- les éléments de base de l'IPMS (X.400/Table 2)
- les éléments optionnels de l'IPMS qui sont considérés comme essentiels (X.401 Tables 1 et 2)
- les éléments optionnels additionnels (X.401 Tables 1 et 2) s'il sont présents dans les PICS.

De même, un système conforme au service MT X.400 doit supporter correctement :

- les éléments de base du MTS liés au protocole P1 (X.400 Table 1)
- les éléments optionnels essentiels du MTS(P1) (X.400 table 3 et 4)
- les éléments optionnels additionnels (X.401 Table 3 et 4) s'ils sont présents dans les PICS
- les indications présentes dans le X.400 Implementor's Guide <X.400/IG>

Enfin, un système conforme au service RTS X.400 doit supporter correctement :

- le service RTS défini dans la norme X.410
- les corrections et indications présentes dans le X.400 Implementor's Guide <X.400/IG>

5.3.6.1.2 Suites de tests

Les suites de tests, applicables à une implémentation X.400, doivent monter que :

- l'IUT n'agit ou ne réagit pas d'une manière différente à celle décrite dans les recommandations,
- l'IUT est capable de détecter et de gérer les erreurs de protocole,
- l'IUT repecte les exigences définies dans le <X.400/IG> concernant les longueurs maximales et le nombre maximum d'occurences de certains éléments.

5.3.6.2 Configurations

Deux types de configurations sont utilisés.

La figure V.14 présente la configuration définie pour les tests de l'IPMS(P2), MTS(P1) et RTS.

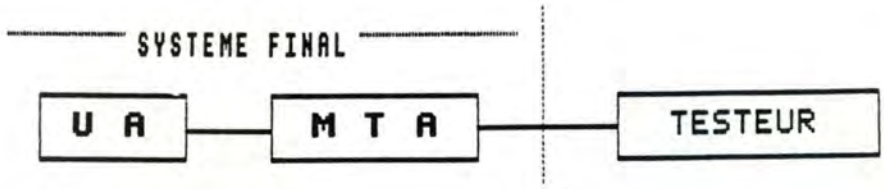


FIG V.14 : PREMIERE CONFIGURATION

La seconde configuration est présentée sur la figure V.15. Elle est utilisée pour tester les possibilités de relais du protocole MTS(P1).

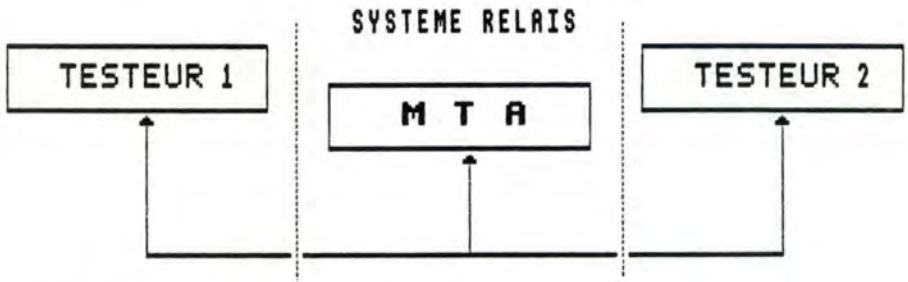


FIG V.15 : SECONDE CONFIGURATION

5.3.6.3 Points de controle et d'observation

X.403 définit les points de contrôle et d'observation (PCO) suivants :

5.3.6.3.1 PCO pour l'IPMS

La figure V.16 montre l'emplacement des PCO utilisés pour les tests d'IPMS.

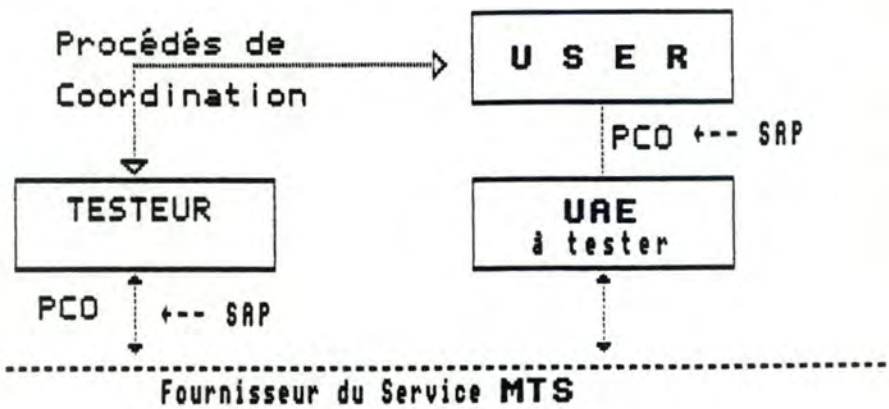


FIG V.16 : PCO POUR L'IPMS

Pour le testeur, le PCO est situé au Service Acces Point (SAP) défini à la frontière entre les couches UAL et MTL.
Pour l'IUT, le PCO est situé à l'endroit où se fait l'intéraction entre l'utilisateur et son UA.

5.3.6.3.2 PCO pour le MTS

La figure V.17 montre l'emplacement des PCO utilisés pour les tests du MTS.

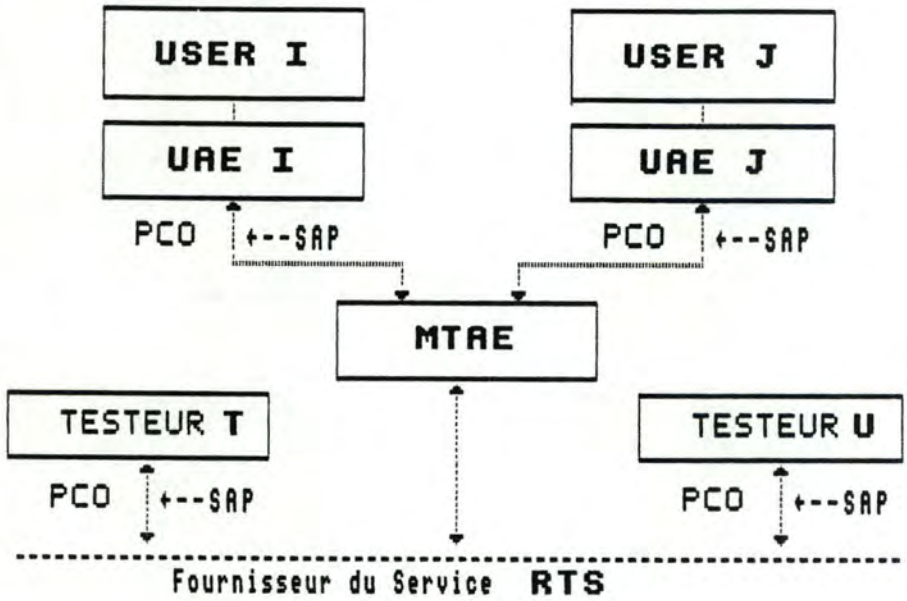


FIG V.17 : PCO POUR LE MTS

Pour le testeur, le PCO est situé au SAP défini à la frontière entre les couches MTL et RTS.
Pour l'IUT, le PCO est le SAP défini à la frontière entre les couches UAL et MTL.
Le test des fonctions de relais demande la présence de plusieurs PCO testeurs. De même, le test d'envoi de message à destinations multiples demande l'existence de plusieurs UA dans l'IUT.

5.3.6.3.3 PCO pour le RTS

La figure V.18 montre l'emplacement des PCO utilisés pour les tests du RTS.

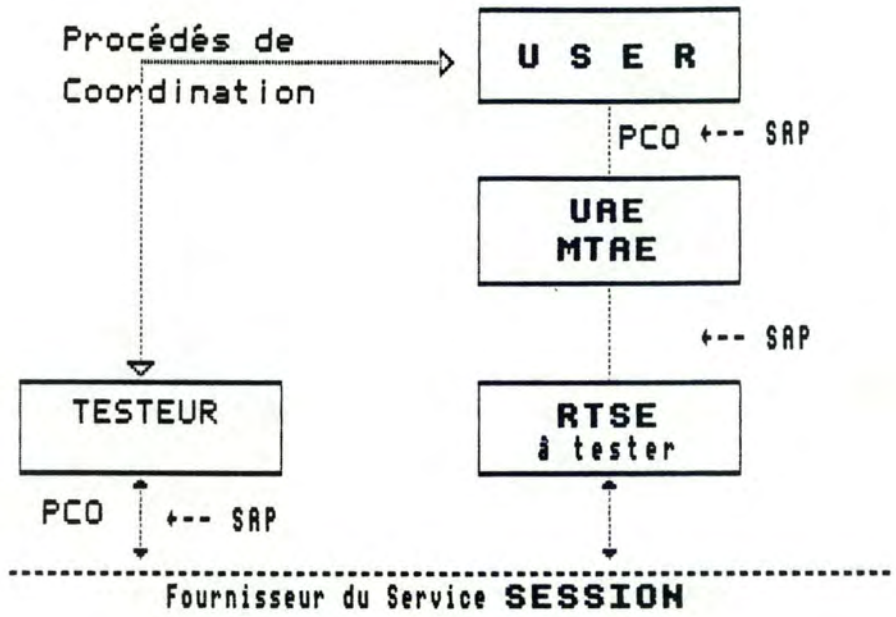


FIG V.18 : PCO POUR LE RTS

Pour le testeur, le PCO est situé au SAP défini à la frontière entre le RTS et la couche SESSION.
Pour l'IUT, le PCO est situé à l'endroit où se fait l'intéraction entre l'utilisateur et son UA.
Le test du RTS exige l'observation (pour des mesures de contrôle) des évènements survenant à la frontière des couches MTL et RTS.

5.3.6.4 Principes de conception des tests

5.3.6.4.1 Principes

Sur base des définitions et des méthodes déja présentées au sections 5.1 "Méthodologie" et 5.3 "Tests de conformité", la recommandation X.403 applique les principes de construction de stratégie de test suivant :

- un cas de test définit la séquence des évènements liés aux ASP et les paramètres associés (en particulier les PDU),

- les cas de test de comportement valide définissent les évènements ASP selon la recommandation X.400,

- les cas de test de comportement invalide sont caractérisés par :
  - un évènement inopportun c'est-à-dire un évènement correct (ou un PDU correct) généré par le testeur dans un état de protocole où celui-ci n'est pas permis.
  - un PDU correct incorporant un élément syntaxiquement correct mais en conflit avec la valeur qui avait été négociée
  - un PDU syntaxiquement incorrect (un élément obligatoire manquant par exemple)
  - pour le RTS, un évènement ASP qui contient des paramètres interdits ou non appropriés (mauvais paramètre dans le SConnect par exemple).

- la profondeur du test est limitée par les principes suivants :
  pour un comportement correct :
  - s'il y a un petit nombre d'éléments de protocole, il faut les tester tous.
  - s'il existe une étendue de validité d'une valeur (range), il faut tester les limites et quelques valeurs habituelles.
  - s'il n'existe pas de limite, il faut tester un valeur extrême en rapport avec les valeurs habituelles.

  pour un comportement incorrect :
  - le nombre des cas de tests pour un type particulier d'erreur est limité à un ou plusieurs cas habituels.

5.3.6.4.2 Test d'encodage/décodage X.409

Le test du X.409 fait partie des tests de MTS(P1), d'IPMS(P2) et de RTS. Deux catégories de tests X.409 ont été identifiées :

5.3.6.4.2.1 Encodage X.409
Les tests d'encodage X.409 sont construits en identifiant un ensemble de requètes de l'utilisateur qui vont causer la génération par l'IUT de PDU dont l'encodage va utiliser la plupart des fonctionnalités majeures du X.409. Le testeur vérifiera la validité de ces PDU.

5.3.6.4.2.2 Décodage X.409
Les tests de décodage X.409 sont contruits sur base d'une liste de particularités X.409 à tester. Les PDU ainsi générés vont causer des réactions aux erreurs qu'ils contiennents dans l'IUT.

On peut remarquer que les tests d'encodage ne permettent de tester que des comportements valides alors que les tests de décodage permettent en plus de tester des comportements invalides.

5.3.6.4.3 Tests P2 (IPMS)

Deux catégories de tests sont identifiées, la première concerne l'IUT comme origine et la seconde l'IUT comme récipient.

5.3.6.4.3.1 IUT comme origine
Quand l'IUT est origine, les tests sont réalisés en invoquant le service à tester, en vérifiant (par le testeur) la validité des PDU résultats, et en recevant un PDU contenant une réponse valide ou invalide selon l'élément testé.

5.3.6.4.3.2 IUT comme récipient
Quand l'IUT est récipient, les tests sont réalisés en envoyant (par le testeur) des PDU valides ou invalides pour le service testé, en observant la réaction locale du UA et en vérifiant les PDU générés par celui-ci.

5.3.6.4.4 Tests P1 (MTS)

Les catégories de test suivantes ont été identifiées :
- IUT comme origine
- IUT comme récipient
- IUT comme relais
- IUT comme récipient et relais
- IUT comme origine et récipient

5.3.6.4.4.1 IUT comme origine
Lorsque l'IUT est origine, les tests sont réalisés en invoquant le service à tester et en vérifiant la validité des PDU résultats.

5.3.6.4.4.2 IUT comme récipient
Quand l'IUT est récipient, les tests sont réalisés en envoyant (par le testeur) des PDU valides ou invalides en fonction du service testé, en observant la réaction de l'UA et en vérifiant la validité des éventuels PDU générés par celui-ci.

5.3.6.4.4.3 IUT comme relais
Quand l'IUT agit comme relais, les tests sont réalisés en envoyant (par le testeur) des PDU valides ou invalides pour le relais et en vérifiant la réaction de l'IUT.

5.3.6.4.4.4 IUT comme récipient et relais
Lorsque l'IUT est à la fois relais et récipient, les tests sont réalisés en envoyant un ensemble de PDU valides et invalides destinés à plus d'un destinataire. Un des PDU valides doit être destiné à l'IUT et un autre doit être relayé.
On vérifie ensuite la réaction de l'IUT en tant que récipient et le bon relais des PDU destinés aux autres implémentations.

5.3.6.4.4.5 IUT comme origine et récipient
Quand l'IUT est à la fois origine et récipient, les tests sont réalisés en demandant à l'IUT d'envoyer un message à des récipients multiples
Un des ces récipients doit être local à l'IUT et un autre doit être attaché à un UA distant.
On vérifiera ensuite la réaction de l'IUT comme récipient et la validité des PDU envoyés vers l'UA distant.

5.3.6.4.5 Tests RTS

Les phases de tests suivantes sont utilisées :

5.3.6.4.5.1 Connexion et négociation

La phase de connexion est testée selon les différentes options négociables. Chaque option est testée exhaustivement dans les cas valides et invalides.

5.3.6.4.5.2 Libération

La phase de libération étant assez simple, le test de sa bonne implémentation est évident.

5.3.6.4.5.3 Transfert de données

La phase de transfert de données avec échange de jeton doit être testée en fonction :
- du fonctionnement correct avec les valeurs négociées,
- du fonctionnement correct de l'échange de jeton,
- des bonnes confirmations des services testés,
- de la réaction correcte aux éléments invalides.

5.3.6.4.5.4 Reprise sur incident

La phase de reprise sur incident (recovery) doit être testée en fonction des arrêts (aborts) générés par l'utilisateur, par le fournisseur, par la survenance d'une erreur et par des checkpoints non confirmés.

5.3.6.5 Structure des suites de tests (P1/P2 et RTS)

5.3.6.5.1 Tests d'IPMS(P2) et de MTS(P1)

La structure des suites de tests d'IPMS(P2) et de MTS(P1) est identique et se compose des phases suivantes :
- les TESTS INITIAUX qui vérifient les fonctions obligatoires dans un nombre de cas limités. Ils permettent de vérifier que l'implémentation supporte correctement les principales fonctions de base et de décider s'il est raisonnable de continuer ou non la procédure,
- les TESTS X.409 qui vérifient le bon encodage et décodage des PDU,
- les TESTS D'ELEMENTS DE PROTOCOLE qui vérifient le bon support du protocole utilisé (P1 ou P2),
- les TESTS D'ELEMENTS DE SERVICE qui contrôlent la capacité de l'IUT à supporter les éléments de service définis dans la norme appropriée (IPMS ou MTS).
- les TESTS ADITIONNELS qui vérifient les particularités non testées jusqu'à présent.

5.3.6.5.2 Tests du RTS

La structure des tests du RTS se compose des phases suivantes :
- les TESTS D'ETABLISSEMENT D'ASSOCIATION,
- les TESTS DE LIBERATION D'ASSOCIATION,
- les TESTS DE TRANSFERT DE DONNEES
- les TESTS DE REPRISE SUR INCIDENT
- les TESTS X.409

5.3.6.6 PICS et PIXIT

Les PICS X.400 sont constitués par l'ensemble des informations fournies par l'implémenteur qui listent et spécifient les fonctionnalités implémentées dans l'IUT.
Le document <X.403> définit dans ses annexes B (IPMS), C (MTS) et D (RTS), des grilles écrites dans un format standardisé ou "PICS proformas" a utiliser pour rédiger ces informations.

Les PIXIT X.400 sont constitués par les informations supplémentaires fournies par l'implémenteur qui seront utilisées par le testeur pour exécuter une suite de tests.

5.3.6.7 Procedure de test X.400

La figure V.19 reprend la procédure de test utilisée pour réaliser les tests de conformité X.400.

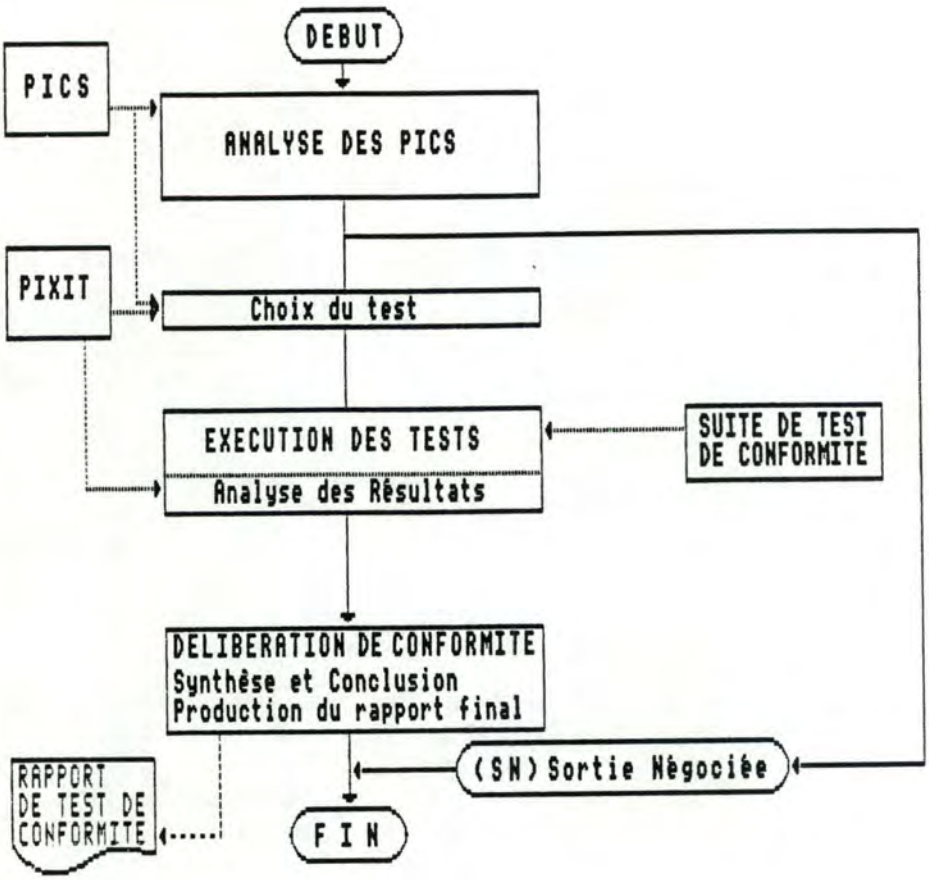


FIG V.19 : PROCEDURE DE TEST X.400

Après la phase d'analyse des PICS, on choisi sur base des PIXIT une suite de test de conformité.
Après son exécution, on analyse les résultats et on prononce le verdict lors de la phase de délibération de conformité. Un rapport final de test de conformité est alors généré.

Cette procédure de test provient de la simplification de la procédure de test générale exposée au paragraphe 5.3.4 de ce chapitre.

5.3.7 OUTILS DE TEST

Après avoir analysé les aspects théoriques des tests de conformité X.400, nous présentons les différents types d'outils utilisables pour la bonne réalisation de ces tests.
Nous basons notre approche sur les articles suivants : <BREM-85>, <CAST-87>, <CHRA-87>, <GENE-86>, <PLAT-87> et <SARI-82>.

Les différents outils que nous allons présenter sont : les langages de définition et de génération de scénario ou de séquence de tests, les langages de contrôle du système de test, l'analyseur de résultats, le traceur et le moniteur.

5.3.7.1 Langages de définition et de génération de scénario

Un langage de définition de scénario ou de séquence de test permet, à l'aide d'une syntaxe de haut-niveau, de définir, dans le cadre particulier des tests de X.400, des suites de tests liées à un but particulier.

Ces langages se basent sur les langages de spécification de protocoles définis par les organismes de normalisation.
Nous pouvons citer ici, TTCN (Tree and Tabular Combined Notation), ESTELLE (ESTL - Extended State Transition Language) et LOTOS (Language for Temporal Ordering Specification) de l'OSI, ainsi que SDL (Specification and Description Language) du CCITT.

Ces langages permettent de décrire de manière formelle des services et des protocoles dans le but d'évaluer leurs possibilités.

Notre but n'étant pas d'analyser les différents langages, on trouvera des informations complémentaires sur ces langages et leurs dérivés dans <PLAT-87>, <CAST-87>, <CHRA-87> et <SARI-82>.

5.3.7.2 Langages de contrôle du système de test

Un langage de contrôle de système de test permet, sur base d'une séquence de test définie (ou générée) par un langage de définition (ou de génération) de scénario, de mener à bien l'exécution d'un test.

5.3.7.3 Analyseur de résultats

l'analyseur de résultats est un module qui permet de réduire à l'essentiel l'interaction humaine dans l'observation de l'IUT aux différents PCO.
Son but principal est d'offrir aux conducteurs du test les résultats de celui-ci sous la forme la plus précise et la plus concise possible.

5.3.7.4 Traceur

Un traceur est un outil qui permet d'obtenir le maximum de renseignements sur le comportement d'un module ou un ensemble de modules de l'IUT.
Il agira principalement aux endroits définis par les PCO pour permettre la vérification du bon comportment de l'IUT en fonction du test réalisé.

Dans nos tests du logiciel MRX, les traceurs suivants ont étés utilisés : fichiers MRX$LOG.LOG, MRLOG.LOG et MRERR.LOG au niveau MR-MRX, logiciels VOTSTRACE et PSITRACE au niveau TRANSPORT et RESEAU.

5.3.7.5 Moniteur

Dans certains cas où la création d'un système testeur s'avère difficile voire impossible, il est plus simple d'observer le SUT à l'aide d'un moniteur que de le stimuler artificiellement comme on le fait dans les méthodes à répondeur et/ou simulateur.
C'est le cas lorsque le SUT est indivisible ou atomique, c'est-à-dire que l'on ne peut facilement en isoler les couches tant au niveau logiciel que matériel.
Un exemple de ceci concernerait le test d'un composant électronique implémentant à la fois le niveau 3 de X.25 (Réseau) et le niveau 4 (Transport).

Un moniteur est un module extérieur au système qui observe celui-ci et qui garde trace de toutes ses activités (messages reçus, traitements effectués et message envoyés). On analyse donc le système dans son ensemble et non plus une partie limitée de celui-ci.
On va donc observer le comportement de ces systèmes en analysant leur trafic par la confrontation de leurs entrées et sorties.

Le moniteur doit être complêtement invisible pour le SUT et l'IUT. On peut néanmoins distinguer deux types différents de moniteur, le moniteur passif et le moniteur actif.
Dans le monitoring passif, on se contente d'observer le système en fonctionnement.
Dans le monitoring actif, en plus du trafic normal, on permet l'injection dans l'IUT de perturbations temporelles (timers), de modification dans le contrôle de flux ou de génération de messages erronés.

5.3.8 AUTOMATISATION DES TESTS

Toute automatisation dans le déroulement du processus de test sera la bienvenue. En effet, celle-ci permettra d'aller encore plus vite, de généraliser les problèmes et d'ainsi encore minimiser l'impact du test en cours sur le SUT.

5.3.9 IMPACT D'UN TEST

Les méthodes de tests doivent avoir pour but une charge de travail minimum dans le développement de l'IUT. Ceci signifie placer le moins d'obstacles possibles dans le chemin des concepteurs et des développeurs, avoir le moins de changement possible dans le comportement du SUT et augmenter la sécurité et l'intégrité du système complet. De plus, les différents outils de test ne doivent pas nécessiter la présence de ressources supplémentaires (ou le moins possible).

5.4 ORGANISMES ET SITES DE TEST

Présentons brièvement quelques organismes et sites de test et leur rôle principal dans les tests de conformité. Nous utiliserons pour ce faire <DAVI-85> (National Computing Center COMMS-AID), <JACO-87> (Communauté Européenne) et <PIQU-87> (SPAG).

On peut définir deux types différents de sites de tests, premièrement les sites offrant des possibilités d'interconnexion de base et deuxièmement les sites faisant de la délibération de conformité.

Le rôle d'un site de tests d'interconnexion de base est d'offrir une infrastructure permettant ce type de test et de servir ainsi de premier filtre dans toute analyse de conformité.
Un exemple de ce genre de site est **SPAG Services** (Standards Application Promotion Group), une compagnie regroupant les participants suivants : Bull, ICL, Nixdorf, Olivetti, Philips, Siemens, Stet et Thompson).

Le rôle des sites de tests de délibération de comformité est entièrement différent. Ces sites, souvent créés à l'initiative d'organismes de normalisation, ont comme fonction principale le test précis et complet des différentes couches d'une IUT dans le but final de décerner un document officiel de conformité.
Parmi les nombreux centres existants, on peut citer, dans le cadre du programme européen CTS (Conformance Testing Services), ceux de la BRITISH TELECOM, de la DEUTSCHES BUNDESPOST (FTZ) ou encore du CNET (France).

5.5 LE LOGICIEL MRXTESTER

Le logiciel MRXTESTER est un User Agent permettant à un utilisateur de soumettre des messages de test au MTA de Digital (Message Router et Message Router X.400 Gateway). Il permet de tester l'implémentation X.400 supportée par MRX (au niveau des protocoles P1 et P2), ainsi que l'interopérabilité de MRX avec d'autres systèmes X.400.

MRXTESTER ne permet pas le test de l'encodage/décodage X.409, ni celui du RTS. Ces limitations proviennent, d'une part, des outils du toolkit MRIF (Message Router Interface Functions) que nous avons utilisés et, d'autre part, du fait que les modules en question sont cachés dans l'architecture de MRX et sont donc inaccessibles.

Nous avons donc du admettre le bon fonctionnement de ces modules (encodage/décodage X.409 et RTS) sur base du bon comportement général du système lors des tests initiaux d'interconnexion de base.

Toutefois, nous pouvons remarquer que, bien que nous n'ayons pas les moyens de stimuler ces modules, nous avons, par contre, la possibilité de vérifier de façon limitée leur comportement lors du fonctionnement du système par le biais du monitoring des fichiers d'enregistrement d'évènements (logging) au niveau du MRX.
Cela permettra, lors des tests, d'isoler la source d'une erreur dans l'encodage, le décodage et/ou le RTS.

5.5.1 SPECIFICATIONS

Le produit à tester est une implémentation d'un MTA, soit un produit de la couche application du modèle ISO. Le bon fonctionnement des couches inférieures (1 à 6) est supposé acquis. Nous considérons ici que le noyau session implémenté dans le Message Router X.400 Gateway est correct, c'est à dire que MRX offre les fonctionnalités session nécessaires à l'échange de messages X.400.

Un MTA reçoit comme argument un message. Le résultat fourni par le MTA consiste soit en la réception du message par le destinataire avec, éventuellement la réception d'une confirmation de livraison sur le site émetteur, soit en la non réception du message par le destinataire avec éventuellement une notification de non-livraison expliquant la cause de l'échec. Un MTA permet d'envoyer, de recevoir et de relayer un message. Les tests devront veiller à vérifier ces trois fonctionnalités.

Les tests se déroulent en trois phases :

1 - tests locaux en boucle (Loopback)
Ils permettent de vérifier l'installation et de réaliser des tests d'intégration des différents logiciels.

2 - tests à distance avec un noeud Message Router X.400 Gateway
Ils permettent de vérifier la configuration des bases de données MRX et de tester les capacités d'échange de messages,

3 - tests à distance avec un noeud de configuration X.400 différente
Ils permettent de vérifier l'interopérabilité et la conformance du logiciel. Ces tests seront d'autant plus fructueux que le noeud éloigné sera reconnu comme site de référence, c'est à dire que sa conformité aux normes X.400 aura déjà été prouvée.

Un User Agent de tests doit permettre à un utilisateur de soumettre un message quelconque à son MTA en vue de tester ses réactions et celles du MHS à ce message. Le User Agent développé au chapitre 4 n'autorisait la création que d'UAPDU, lui même se chargeant de l'insertion de cet UAPDU dans un UMPDU. Un User Agent de tests doit permettre la création interactive de UMPDU (enveloppe et contenu), ceci pour rendre possible la conduite de tests au niveau des protocoles P1 et P2.

5.5.2 IMPLEMENTATION

Nous présentons ici l'implémentation du logiciel "MRXTESTER" que nous avons réalisée. Après l'analyse des architectures logique et physique de ce logiciel, nous examinons les différents modules qui les composent et les différentes fonctions qu'ils réalisent.
Ensuite, nous expliquons la méthode que nous avons choisie pour réaliser les tests de conformité et d'interopérabilité.

5.5.2.1 Architecture logique

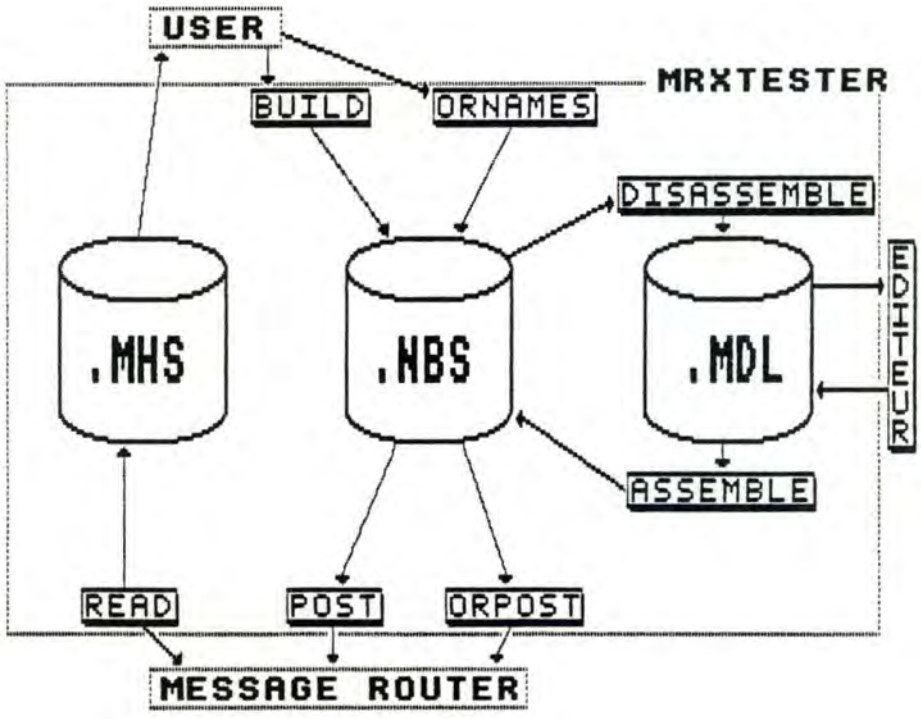


FIG V.20 : Architecture logique de MRXTESTER.

La figure V.20 décrit l'architecture logique de MRXTESTER ainsi que le flux des messages transitant dans le système. Avant de détailler les différents composants du système, voyons brièvement comment se déroule le transit d'un message dans MRXTESTER.

Un utilisateur crée un message au moyen de l'utilitaire BUILD qui construit des messages NBS à partir des informations encodées par l'utilisateur. L'utilisateur utilise POST pour soumettre un message NBS au Message Router. Un message reçu peut être lu au moyen de l'utilitaire READ. Cet utilitaire est le même que celui définit au chapitre IV. Il lit une mailbox MR et crée un fichier texte, que nous appelerons MHS, compréhensible par l'utilisateur.

Un fichier créé par BUILD peut-être modifié par l'utilisateur. Pour cela, il doit désassembler le message NBS, éditer le fichier MDL et l'assembler pour obtenir à nouveau un fichier NBS.

Les modules ORNAMES et ORPOST permettent respectivement de générer de manière automatique une série de cas de test d'adressage (O/R NAMES) liés à un destinataire particulier et de les soumettre au Message Router.

5.5.2.2 Architecture physique

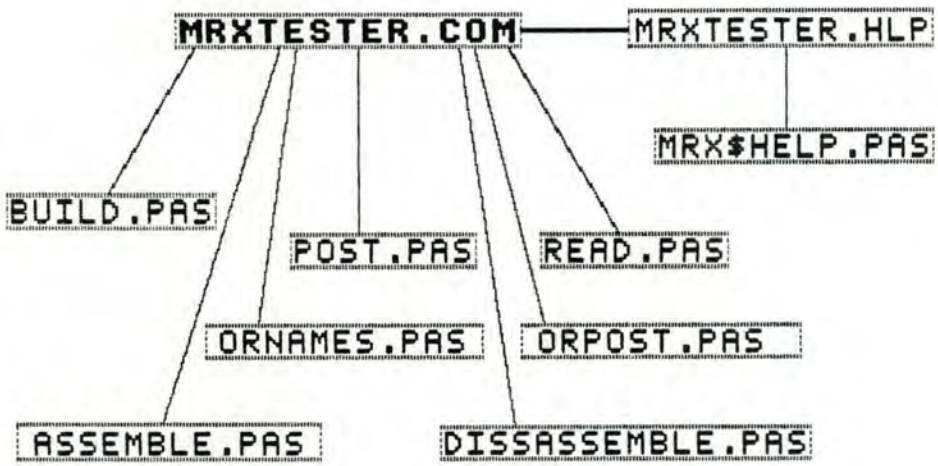


FIG V.21 : Architecture physique de MRXTESTER.

La figure V.21 présente l'architecture physique de MRXTESTER. MRXTESTER.COM est la procédure DCL coordonant les différents modules. Une aide interactive est fournie par la procédure MRXTESTER$HELP.HLP. Les modules de travail sont respectivement MRXTESTER$BUILD.PAS, création interactive de messages, MRXTESTER$POST.PAS, soumission d'un message NBS au Message Router, MRXTESTER$READ.PAS, lecture d'une mailbox et affichage des messages (ce programme est le même que UA$READ.PAS du chapitre IV), MRXTESTER$ORNAMES.PAS, génération automatique de cas de test d'adressage, MRXTESTER$ORPOST.PAS, soumission des cas de test générés automatiquement, MRXTESTER$ASSEMBLE.PAS, traduction de fichiers MDL en NBS, MRXTESTER$DISASSEMBLE.PAS, traduction de fichiers NBS en MDL.

MRXTESTER$HELP.PAS fournit un système d'aide interactive au niveau des modules de travail.

Un exemple de session de travail avec MRXTESTER est donné en annexe.

5.5.2.3 Fonctionnalités de MRXTESTER

Nous décrivons ici briévement les fonctionnalités offertes par le logiciel.

5.5.2.3.1 Initialisation du logiciel

L'initialisation du logiciel consiste à créer une mailbox MR MRXTESTER et un abonné MRX correspondant à cette mailbox. Cette initialisation est optionnelle, cependant les composants entrant en contact avec le Message Router (POST, ORPOST) essaient toujours de s'identifier par cette mailbox. Cependant, si celle ci est absente, ils demandent à l'utilisateur un nom de mailbox MR valide pour l'identification.

5.5.2.3.2 Création de messages

MRXTESTER autorise la création interactive de messages (BUILD). Nous avons vu au chapitre IV que les routines d'interface avec le Message Router (MRIF) prévoient la création de quatre types de messages : une enveloppe seule, un contenu seul, une enveloppe suivie d'un contenu, un contenu dont l'enveloppe est crée à partir d'informations contenues dans l'en-tête. L'utilisateur peut choisir explicitement quel type de message il désire créer.

Lors de la phase de création de message, l'utilisateur est guidé pas à pas selon une technique de garnissage de zones par patron. MRXTESTER signale les zones obligatoires, optionnelles, répétitives, simples et composées. Un aide interactive est fournie pour chaque élément de message. Contrairement au User Agent du chapitre IV, MRXTESTER n'effectue aucun contrôle sur la présence ou non d'éléments obligatoires dans le message. Ceci en vue de tester les réactions du MTA et du MHS.

De plus, pour éviter Les erreurs lors de l'encodage de chaines de caractères (pour les tests de longueurs par exemple) et pour faciliter les échanges avec S.P.A.G. (Standard Promotion Application Group) SERVICE, le logiciel intègre la bibliothèque de composants prédéfinis "SPAG Component Library" publiée dans le "SPAG Interworking Guide" <SPAG-IG>.

BUILD crée un ou deux fichiers NBS contenant la ou les parties de message assemblés , selon le type de message créé. Ces fichiers pourront être soumis au Message Router autant de fois que nécessaire.

BUILD permet la création d'éléments des classes d'équivalence des tests des protocoles P1 et P2.

5.5.2.3.3 Soumission de messages

Une fois que l'utilisateur a créé son message, il peut le soumettre au Message Router (POST). Cette soumission s'effectue par fichiers : un pour l'enveloppe, un pour le contenu, avec possibilité d'envoyer une

enveloppe seule. Cette forme de soumission est particuliérement souple, puisqu'elle permet d'envoyer plusieurs enveloppes différentes avec le même contenu, et inversément. La procédure de soumission est similaire à celle décrite au chapitre IV, c'est à dire : démarrage d'une session de travail avec le Message Router, identification, soumission du message, confirmation d'acceptation.

5.5.2.3.4 Lecture d'une mailbox

Les fonctionnalités de lecture **READ** sont identiques à celles décrites au chapitre IV.

5.5.2.3.5 Génération automatique d'O/R names

Pour faciliter les tests, nous avons conçu deux modules, appelés respectivement ORNAMES et ORPOST, qui permettent de générer de manière automatique une série de cas de test d'adressage (O/R NAMES) liés à un destinataire particulier.

Ces tests créent, à partir de l'adresse d'un utilisateur du MHS, une série de fichiers correspondant aux quatres types d'O/R NAMES définis dans le norme et présentés au chapitre I. De plus, pour chaque type, ils créent les cas suivant : orname vide, orname minimum, orname complet et orname erroné.

5.5.2.3.6 Assemblage et désassemblage de messages

Les utilitaires **ASSEMBLE** et **DESASSEMBLE** permettent de modifier un fichier NBS créé par BUILD. Le désassembleur transforme le message du format NBS en format MDL. MDL ou Message Description Language (Langage de Description de Messages) a été développé par les auteurs de cet ouvrage. Ce langage se base principalement sur la syntaxe des routines MRIF. Il a été conçu pour répondre à un besoin d'édition de messages déjà assemblés. L'assembleur permet de traduire un fichier MDL en fichier NBS.

Présentons le langage MDL en syntaxe EBNF.

```
Message  = Envelope ¦ Content ¦ Envelope Content ¦ Complete_Message.

Envelope = "Start_assemble(Envelope);" [Item] "End_assemble".
Content  = "Start_assemble(Content);" [Item] "End_assemble".
Complete_Message = "Start_assemble(Complete_Message);" [Item]
                   "End_assemble".

Item = Item_code ":=" Item_value ";" ¦ Composite_item ¦ Flag_item.
Composite_item = "Start(" Start_class "," Start_item ") ;" [item] "End;".
Flag_item = "Flag(" Flag_name ") ;" [Flag] "End_flag ;".

Item_code = Integer_item ¦ String_item ¦ Date_item.
Integer_item = "precedence" ¦ "contenttypes" ¦ "hopcount" ¦ "extensionid"
  ¦ "explicconv" ¦ "autoforward" ¦ "sensitivity" ¦  "replyreq".

String_item  =  "message_id" ¦ "userid"  ¦  "route"  ¦  "x121_address"  ¦
  "terminlaid"  ¦  "orgname" ¦ "orgunit"  ¦  "surname"  ¦  "givenname"  ¦
  "initials" ¦ "generation" ¦ "telephone"  ¦  "location"  ¦  "country"  ¦
  "amdname" ¦ "prdname" ¦ "uniqueuaid"  ¦ "listname" ¦ "app_message_id" ¦
  "subject" ¦ "inreplyto" ¦ "obsoletes"  ¦  "references"  ¦  "msgclass"  ¦
```

```
  "freeform".

Date_item = "pdate" ¦ "enddate" ¦ "replyby" ¦ "date".

Start_class = "bodypart" ¦ "list" ¦ "name".
Start_item = "text" ¦ "forwarded" ¦ "g3fax" ¦ "tif0" ¦ "tif1" ¦ "voice" ¦
  "telex" ¦ "sfd" ¦ "videotex" ¦ "wpsplus" ¦ "rmsfile" ¦ "odif" ¦ "ddif"
  ¦ "ia5text" ¦ "voicenotif" ¦ "vendor1..16" ¦ "to" ¦ "cc" ¦ "bcc" ¦
  "replytousers" ¦ "sender" ¦ "from" ¦ "author".

Flag_name = "encodedtypes" ¦ "permsgflg" ¦ "perrecflg" ¦ "perlistflg" ¦
  "reportflg".
Flag = "action" ¦ "mr_basic" ¦ "mr_confirmed" ¦ "ua_basic" ¦
  "ua_confirmed" ¦ "discloserec" ¦ "convprohib" ¦ "altrecip" ¦
  "contretreq" ¦ "expandedlist" ¦ "recnotif" ¦ "nonrecnotif" ¦
  "returnmsg" ¦ "rmdfile" ¦ "wpsplus" ¦ "ddif" ¦ "voicenotif" ¦ "text" ¦
  "odif" ¦ "tif0" ¦ "tif1" ¦ "sfd" ¦ "voice" ¦ "videotex" ¦ "teletex" ¦
  "g3fax" ¦ "ia5text" ¦ "telex" ¦ "undefined".
```

Une description plus détaillée dépasserait le cadre de cette étude et ne constituerait qu'un simple recopiage de <VAX-MRIF>. Signalons cependant que certains éléments (par exemple : "wpsplus") sont propres à Digital. Ceci vient du fait que le système de communication est basé sur le Message Router qui supporte ces éléments, c'est pourquoi nous les retrouvons ici.

Voici à titre d'exemple le message encodé par le UA au chapitre IV traduit en MDL.

```
Start_assemble(complete_message);
   Start(name,to);
       Givenname  := Marc;
       Surname    := Eloy;
       Initials   := ME;
       Country    := BE;
       Amdname    := RTT;
       Prdname    := FNDP;
       MR_address := FNDP_MRX;
   End;

   Subject := Nouvelle version du UA.;
   Start(bodypart,text);
       Text_record := La nouvelle version de UA$SEND est prête,
       Text_record := je te l'envoie dans ce message.
       Text_record := DRX.;

       Text_file   := [MRMANAGER]UA$SEND.PAS
   End;
End_assemble;
```

Une autre façon d'encoder le nom pourrait être :

```
Start(name,to);
  MR_address := 1=BE @ 2=RTT @ 3=FNDP @ 6=Eloy @ 7=Marc @ 8=ME @ FNDP_MRX;
End;
```

La limite principale du langage MDL sous sa forme actuelle est qu'il ne peut y avoir qu'une instruction par ligne et une ligne par instruction. Cette limite peut être lourde lorsqu'on désire encoder plusieurs lignes

de texte ou bien donner une Message Router Address sur plusieurs lignes. Cependant, cette "lourdeur" peut-être contournée en utilisant le User Agent décrit au chapitre IV. En effet, MDL a été conçu avant tout dans le but d'apporter des modifications mineures à des messages déjà assemblés sans avoir à les recommencer entiérement. Cet objectif semble avoir été atteint. Néanmoins l'architecture des modules d'assemblage et de désassemblage permet d'effectuer les modifications nécessaires pour lever cette limite.

5.5.2.4 LIMITES DU MRXTESTER

Outre les limites déjà données au paragraphe 5.5.2.3.6 relatives au langage MDL, et le fait que l'on ne sait pas tester l'encodage et le décodage X.409 ainsi que le RTS, nous allons présenter les limites du produit MRXTESTER.

La principale limite est l'importabilité du logiciel. L'UA MRXTESTER a été développé, commme son nom l'indique pour réaliser des tests de conformité sur le produit X.400 de Digital, Message Router et Message Router X.400 Gateway. Il utilise intensivement les routines MRIF d'interface avec le Message Router, ainsi que la notion de mailbox MR qui est propre à Digital.

Les résultats d'envois de messages de test peuvent être analysés en des endroits divers : lors de l'assemblage de fichier MDL (erreur de syntaxe MDL), par les routines d'interface MRIF (élément de message mal placé : "subject" sur l'enveloppe par exemple), par le Message Router (Mailbox MR inconnue, ...), par le Message Router X.400 Gateway (Expéditeur ou domaine inconnu, message intraduisible, ...). MRXTESTER ne permet pas de situer directement la localisation de l'erreur. Celle-ci peut être relevée dans un fichier résultat de l'assembleur, dans une mailbox MR (notification de non-livraison) ou dans les fichiers d'enregistrement du Message Router et du Message Router X.400 Gateway. C'est à l'opérateur de tests de suivre le cheminement de son message à travers le système local et éventuellement sur le MTA cible.

De plus, le logiciel ne contient pas un analyseur automatisé des fichiers de d'enregistrement d'évènement des différents logiciels (NETSERVER.LOG, MRLOG.ERR, MRLOG.LOG et MRX$LOG.LOG) qui, à l'usage, s'est révélé quasi-indispensable. En effet, le format peu convivial et les spécifications de ces fichiers ainsi que leur caractére distribué (entre deux sites par exemple) rendent leur analyse longue et compliquée pour un opérateur de test peu expérimenté.

5.6 CONDUITE DES TESTS

5.6.1 PRELIMINAIRES

La conduite des tests exige la connaissance des paramètres suivants :

5.6.1.1 Mécanismes d'accès

On doit, avant de pouvoir réaliser un test, connaitre la définition des noeuds X.400 que l'on va utiliser. Les éléments suivants doivent être définis, pour chaque noeud, en termes de longeurs et de valeurs:

MTANAME MTAPASSWORD	Pour le contrôle d'accès
SSAP Session Service Access Point TSAP Transport Service Access Point NSAP Network Service Access Point	si utilisés

DNIC (Country & Administration)
X121+extension
Country
ADMD
PRMD

5.6.1.2 O/R Names

Pour pouvoir exécuter des tests (en boucle ou bilatéral), il peut être nécessaire de définir des conventions dans les noms qui seront utilisés. Celles-ci concernent les formats de O/R names supportés et les longueurs des différents champs.

5.6.2 APPLICATION DE LA PROCEDURE DE TEST

Appliquons sur un exemple simple la procédure de test d'un X.400 définie dans le <X.403> et présentée au paragraphe 5.3.6.7.

L'exemple choisi traite du test de la conformité de l'implémentation des O/R NAMES dans le logiciel MRX.

5.6.2.1 PICS et ANALYSE DES PICS

En ce qui concerne l'implémentation des O/R NAMES, DIGITAL affirme que :
1. Le format 1.1 est supporté
2. Le format 1.2 est non délivré
3. Le format 1.3 est non délivré mais relayé
4. Le format 2 n'est pas du tout supporté

Pour simplifier l'exemple, nous allons nous limiter à tester les affirmations 1 et 4 qui traitent respectivement du support des formats 1.1 et 2.

5.6.2.2 CHOIX DU TEST

5.6.2.2.1 Cas de test

Les tests d'O/R names seront faits à l'aide des modules ORNAMES (génération automatique) et ORPOST (conduite des tests) en définissant les cas de test suivants :

1. O/R NAME VIDE : aucun attribut présent,
2. O/R NAME MINIMUM : seuls les attributs obligatoires sont présents,
3. O/R NAME COMPLET : tous les attributs (obligatoires et optionnels) sont présents,
4. O.R NAME ERRONE : les attributs obligatoires comprennent des valeurs erronées (un mauvais nom de domaine administratif par exemple).

Les cas de test ne s'occupent pas des problêmes de longueurs d'attributs.

5.6.2.2.2 Messages générés

Les messages générés sont décrits dans la syntaxe MDL définie au paragraphe 5.5.2.3.6.

5.6.2.2.2.1 CONTENU

Le contenu des messages qui serviront aux tests est unique et à la forme suivante :

```
START_ASSEMBLE(content);
   subject := TEST O/R NAMES;
   date :=  1-FEB-1988 14:18:59.00;
   START(bodypart,text);
     Text_record := CECI REPRESENTE UNE LIGNE ----------------------;
     Text_record := CECI REPRESENTE UNE LIGNE ----------------------;
     Text_record := CECI REPRESENTE UNE LIGNE ----------------------;
     Text_record := ( plus un message indiquant le type de test );
   END;
END_ASSEMBLE;
```

5.6.2.2.2.2 ENVELOPPE

L'envoyeur du message sera l'utilisateur MRXTESTER aux FNDP (NAMUR) et le destinataire sera MARC ELOY à l'IIHE (BRUXELLES).

Les O/R names générés sont les suivants :

POUR LE TEST DU FORMAT 2 :

```
START_ASSEMBLE(envelope);                    [ORNAME VIDE]
   uacontid := TEST O/R names [01001];
   START(name,to);
      FLAG(perrecflg);
      action;
      ua_basic;
      END_FLAG;
      MR_address := IIHE;
   END;
END_ASSEMBLE;
```

```
START_ASSEMBLE(envelope);                    [ORNAME MINIMUM]
   uacontid := TEST O/R names [01002];
   START(name,to);
      FLAG(perrecflg);
      action;
      ua_basic;
      END_FLAG;
      x121address := 2210168;
      MR_address := IIHE;
   END;
END_ASSEMBLE;

START_ASSEMBLE(envelope);                    [ORNAME COMPLET]
   uacontid := TEST O/R names [01003];
   START(name,to);
      FLAG(perrecflg);
      action;
      ua_basic;
      END_FLAG;
      x121address := 2210168;
      terminalid := terminal a BX1;
      MR_address := IIHE;
   END;
END_ASSEMBLE;

START_ASSEMBLE(envelope);                    [ORNAME ERRONE]
   uacontid := TEST O/R names [01004];
   START(name,to);
      FLAG(perrecflg);
      action;
      ua_basic;
      END_FLAG;
      x121address := 000000000000;
      terminalid := 000000000000;
      MR_address := IIHE;
   END;
END_ASSEMBLE;
```

POUR LE TEST DU FORMAT 1.1 :

```
START_ASSEMBLE(envelope);                    [ORNAME VIDE]
   uacontid := TEST O/R names [04001];
   START(name,to);
      FLAG(perrecflg);
      action;
      ua_basic;
      END_FLAG;
      MR_address := IIHE;
   END;
END_ASSEMBLE;

START_ASSEMBLE(envelope);                    [ORNAME MINIMUM]
   uacontid := TEST O/R names [04002];
   START(name,to);
      FLAG(perrecflg);
      action;
      ua_basic;
      END_FLAG;
      country := BE;
      amdname := RTT;
```

```
        surname := ELOY;
        MR_address := IIHE;
     END;
  END_ASSEMBLE;

  START_ASSEMBLE(envelope);                    [ORNAME COMPLET]
     uacontid := TEST O/R names [04003];
     START(name,to);
        FLAG(perrecflg);
        action;
        ua_basic;
        END_FLAG;
        country := BE;
        amdname := RTT;
        prdname := IIHE;
        surname := ELOY;
        givenname := MARC;
        initials := ME;
        MR_address := IIHE;
     END;
  END_ASSEMBLE;

  START_ASSEMBLE(envelope);                    [ORNAME ERRONE]
     uacontid := TEST O/R names [04004];
     START(name,to);
        FLAG(perrecflg);
        action;
        ua_basic;
        END_FLAG;
        country := 000000000000;
        amdname := 000000000000;
        surname := 000000000000;
        MR_address := IIHE;
     END;
  END_ASSEMBLE;
```

5.6.2.3 EXECUTION DU TEST

L'exécution des tests produit un listing de trace d'évènement au niveau des logiciels MRX testés.

Prenons par exemple le cas de l'envoi du MPDU correspondant au test d'ORNAME DE FORMAT 2 COMPLET (CAS 3).

On trouve dans le fichier de trace de Namur (Domaine FNDP) au moment de la traduction du format NBS (MR) en format X.409 (X.400) les enregistements suivants :

1FEB88 14:40:42.37 D004 %MRX-I-STAMSGTRN, Started Message Translation
Le logiciel MRX commence la traduction du message.

1FEB88 14:40:46.28 D004 %MRX-E-SEQELTMIS, Syntax error in message file MRX$43.X409: mandatory sequence element COUNTRYNAME missing
1FEB88 14:40:46.96 D004 -MRX-I-STATEIS, Parser state is STANDARDATTRIBUTELIST
Une erreur se produit, MRX s'attend à rencontrer l'élément COUNTRYNAME, or celui-ci est absent conformément à la spécification du format 2. Cet exemple montre que MRX ne supporte pas le format 2 car, par défaut, il s'attend à traduire un O/Rname de format 1.

1FEB88 14:40:47.47 D004 %MRX-E-SEQELTMIS, Syntax error in message file MRX$43.X409: mandatory sequence element ADMINISTRATIONDOMAINNAME missing
1FEB88 14:40:47.85 D004 -MRX-I-STATEIS, Parser state is STANDARDATTRIBUTELIST
MRX continue la traduction et remarque que l'élément ADMD est également absent.

1FEB88 14:40:52.92 D004 %MRX-I-RTSMRPOST, FNDP_VENUS880201154255Z07 posted MRX$43.NBS at MR-mailbox MRXTESTER as 03244110208891/368@VENUS
Il décide donc de générer une notification de non-livraison et de la poster à l'envoyeur.

1FEB88 14:40:58.98 0021 %MRX-I-RTSSESREL, 2816000880201153938Z21 has been disconnected
MRX se déconnecte.

5.6.2.4 ANALYSE DES RESULTATS

Avant l'exécution des tests, selon les PICS, on s'attend à avoir les résultats suivants:

TEST O/RNAME FORMAT 1.1	:	CAS 1 : erreur de syntaxe
		CAS 2 : bonne livraison
		CAS 3 : bonne livraison
		CAS 4 : erreur de contenu
TEST O/RNAME FORMAT 2	:	CAS 1 : erreur de syntaxe
		CAS 2 : erreur de syntaxe
		CAS 3 : erreur de syntaxe
		CAS 4 : erreur de syntaxe

Après avoir réalisés les différents tests et analysés les divers listings de trace d'évènement (similaires à celui du paragraphe 5.6.2.3), nous obtenons les résultats suivants :

1. TEST O/RNAME FORMAT 1.1	:	CAS 1 : erreur de syntaxe 'MANDATORY SEQUENCE ELEMENT MISSING'
		CAS 2 : bonne livraison
		CAS 3 : bonne livraison
		CAS 4 : réception d'une notification de non livraison pour erreur d'adressage.
2. TEST O/RNAME FORMAT 2	:	DANS TOUS LES CAS (1 à 4) : erreur de syntaxe 'MANDATORY SEQUENCE ELEMENT MISSING'

5.6.2.5 DELIBERATION

Dans le cadre très limité de notre exemple, sur base des tests que nous venons de faire, nous pouvons affirmer, jusqu'à preuve du contraire que :

1. Le format 2 des O/R names n'est pas supporté car, dans tous les cas testés, il est impossible d'envoyer un message utilisant ce format.

2. Le format 1.1 est entièrement supporté car tout message utilisant ce format arrive à destination si celle-ci est correcte (cas 2 et 3).

Ces tests ne permettent pas d'affirmer quoi que ce soit sur les longueurs supportées, ni sur les autres formats d'ORNAMES (1.2 et 1.3).

5.7 COMPTE-RENDU DES TESTS REALISES SUR LE LOGICIEL MRX

5.7.1 TESTS D'INTERCONNEXION DE BASE

Ces tests d'interconnexion de base ont étés faits, d'une part au niveau local sur le site de l'IIHE et, d'autre part, entre deux sites distants (l'IIHE à Bruxelles et le site FNDP à Namur) implémentant le même logiciel (MRX) dans des configurations identiques. Ces tests informels se composaient de l'envoi et de la réception, à l'aide du User Agent développé et exposé au chapitre IV, de messages usuels. La communication entre les deux sites se faisait alors indifféremment en utilisant la messagerie VMS-MAIL (VAX) ou la messagerie X.400.

5.7.2 TESTS DE FONCTIONNALITES

Confiants dans les résultats obtenus, nous avons entamés les tests de fonctionnalités. Nous avons donc définit, selon la méthode utilisée au paragraphe 5.6.2.2. et sur base de l'annexe 8.2 "STRUCTURE BACHUS-NAUR D'UN MESSAGE", quelques séquences de test.

Nous avons utilisés, pour la définition de ces séquences, toutes les fonctionnalités supportées par MRX (voir à ce sujet l'annexe 8.1 "IMPLEMENTATION X.400 FAITE PAR DIGITAL").

Les éléments ont été testés en fonction de leur présence (élément obligatoire ou optionnel), de leur répétitivité (élément récursif) et de leur valeur (élément correct ou non).
Aucun test sur les longueurs n'a été fait par manque de temps et de données sur le sujet.

De l'exécution de ces tests, nous concluons que tout élément de service (IPMS ou MTS) que Digital affirmait avoir implémenté, l'est réellement en envoi (éléments "GENeration" de l'annexe 8.1) et en réception (éléments "DELivery").
Nous ne pouvons rien dire sur le relais ("RELay") de ces mêmes éléments.

5.7.3 TESTS RTS ET X.409

Nos tests ne couvrant pas le RTS et le codage X.409, nous ne pouvons donc rien en dire.

VI. CONCLUSION

Ce travail "Contribution à la mise en oeuvre et aux tests d'un produit X.400" comporte deux parties.

La première partie "Description du système" est consacrée à l'étude d'un produit logiciel X.400. Nous avons étudié les détails de l'implémentation faite par Digital Equipment des recommandations X.400 du CCITT. Nous nous sommes attardés à exposer les fonctionnalités des logiciels implémentant les couches réseau (PSI), transport (VOTS), session (OSAK) du modèle ISO avant de présenter les produits (Message Router et Message Router X.400 Gateway) réalisant les services d'un MTA. Nous nous sommes permis de citer les problèmes et erreurs auxquels nous avons été confrontés lors de notre utilisation de ces différents produits.

Nous avons également étudié, dans le cadre des recommandations du CCITT, les problèmes d'adressage et de routage rencontrés sur un réseau X.400 en général, et sur le système MR/MRX en particulier. Nous avons aussi tenté de résoudre le problème que posait l'interconnexion de MRX avec un système EAN.

Enfin, nous nous sommes penchés sur la conception et la réalisation d'un User Agent simple adapté à la configuration de Digital.

La seconde partie "Méthodes de test" est consacrée, d'une part à l'étude théorique des principes de vérification de protocole, et d'autre part à l'application pratique de ces principes aux test de conformité et d'interopérabilité d'un logiciel X.400 et au développement d'outils de test. Nous avons utilisé ces outils sur différents cas limités.

Enfin, ce travail ouvre des possibilités de recherche ultérieure dans, d'une part le développement d'un agent utilisateur convivial, et d'autre part dans l'approfondissement des tests de conformité du logiciel MRX, notamment la réalisation de contacts avec des centres de test reconnus, l'interconnexion avec le réseau EAN et les tests au niveau du RTS et du codage X.409 ainsi que des fonctions de relais.

VII. REFERENCES BIBLIOGRAPHIQUES

<ALEX-87> Frédéric Alexandre, ETUDE DES COUCHES OSI 5 A 7 DANS LE CONTEXTE DE LA MESSAGERIE ELECTRONIQUE, Mémoire de fin d'études, U.L.B., Faculté des Sciences, Laboratoire d'Informatique Théorique, Année académique 1986 - 1987.

<AN-DE-FR-87> Marie-Elise Angelot, Thierry Delroisse et Christine Franssens, SPECICATION D'UN AGENT DE TRANSFERT DE MESSAGE DANS LE CADRE D'UNE MESSAGERIE ELECTRONIQUE DE TYPE X.400, Mémoire de fin d'études, F.N.D.P., Institut d'Informatique, Année académique 1986 - 1987.

<ANSA-86> J.P. Ansart (ADI), J.D. Colas (IBM), QUELQUES PROBLEMES TECHNIQUES LIES AUX TESTS DE CONFORMITE DES PRODUITS SE RECLAMANT DES NORMES DE TELECOMMUNICATIONS : Résultats d'une Etude Conjointe IBM France - ADI, Des Nouvelles Architecture Pour Les Communications, Eyrolles, 1986, pp.190-199.

<BEYS-87> Ulf Beyschlag (SOFTLAB gmbh Munchen), X.400 CONFORMANCE TESTING - QUO VADIS ?, IFIP WG 6.5, Message Handling System, 1987, pp. (9.1)1-11.

<BO-CR-86> Ph. Bovy et S. Crasset, PROTOCOLES DE COMMUNICATION : ETUDE DES OUTILS DE VERIFICATION DE CONFORMITE D'UNE IMPLEMENTATION, Mémoire de fin d'études, F.N.D.P., Institut d'Informatique Année académique 1985 - 1986.

<BOVE-87> P. Bovenga, MANAGEMENT OF NETWORK SERVICES, Computer Networks And ISDN Systems, VOL.13 Nr.3, 1987, pp.155-160.

<BREM-85> J. Bremer, SOME EXPERIENCE WITH TEST SEQUENCE GENERATION IN APPLICATION LAYER, Protocol Specification, Testing And Verification (IV), R. STROM, S. YEMINI, Y. YEMINI (Editors), Elsevier Science Publishers B.V. (NORTH-HOLLAND), (C) IFIP, 1985, pp.623-635.

<CAST-87> R. Castanet, UN SYSTEME D'AIDE A LA PREPARATION DE TEST DE PROTOCOLES UTILISANT LE LANGAGE TTCN, Actes des Neuvièmes Journées Francophones sur l'Informatique, Liège, Janvier 1987, pp. (7)1-15.

<CHRA-87> C. Chraibi, O. Rafiq et R. Castanet, LTP : LANGAGE DE TEST DE PROTOCOLES, Actes des Neuvièmes Journées Francophones sur l'Informatique, Liège, Janvier 1987, pp. (8)1-13.

<DAVI-85> Ian Davidson (National Computing Center COMMS-AID), OSI : CONFORMANCE & TESTING : A TEST CENTRE VIEW, Open System Interconnection Developments And Implementation Online Publications, Pinner, UK, 1985, pp.95-101.

<DFN-85> K. Truol, G. Henken, P. Kaufmann, DFS : Deutsches Forschungsnetz AN APPLICATION ORIENTED DEVELOPMENT BASED ON OSI STANDARDS, X.400 Message handling system im DFN, Betrieb von Message-Systemen im DFN (1985), DFN-VEREIN Postfach 15 02 09, 1000 Berlin 15.

<DICK-85> J.C.C. Dicks (ICL), OSI : CONFORMANCE & TESTING - The Supplier's View, Open System Interconnection Developments And Implementation, Online Publications, Pinner, UK, 1985, pp.71-88.

<EAN-83> EAN : A DISTRIBUTED MESSAGE SYSTEM, Proceedings Canadian Information Processing Society National Meeting, Ottawa, May, 1983, pp.144-149.

<EAN-84> S. Kille, EAN : Evaluation of the EAN Message Handling System Computer based message systems interconnection CERN/DD/COMICS - WGR/T6, 19 Novembre 1984.

<EAN-85> G. Neufeld, EAN : The EAN Distributed Message System User's manual, 1985.

<FOLL-86> J.P. Foll (BULL), P. Goyer, R. Hubert (COGINTEL), GENEPIX : DES PRODUITS INDUSTRIELS SOUS UNIX POUR LES TESTS DE CONFORMITE OSI, Des Nouvelles Architectures pour les Communications, Eyrolles, 1986, pp.322-329.

<FOX-87> Charles Fox (DEC), CONTRIBUTION TO NBS X.400 SIG : WORLD WIDE X.400 (1984) Conformance Profile Matrix, DECpark II Reading, Berkshire, England, 19 Janvier 1987.

<GENE-86> GENEPIX400 : A TESTSYSTEM FOR X.400 BASED MESSAGE HANDLING SYSTEM, COGINTEL (SEMA-METRA), Des Nouvelles Architectures pour les Communications, Eyrolles, 1986.

<HANO-86> Christophe Hanon, ETUDE DE PROTOCOLES DE HAUT NIVEAU DE MESSAGERIE ELECTRONIQUE, Mémoire de fin d'études, U.L.B., Faculté des Sciences, Laboratoire d'Informatique Théorique, Année académique 1985 - 1986.

<HEAG-87> D. Heagerty, PRACTICAL EXPERIENCE WITH HIGH LEVEL GATEWAYS FOR MAIL TRANSFER, Computer Networks And ISDN Systems, VOL.13 Nr.3, 1987, pp.195-200.

<HENK-87> G. Henken, MAPPING OF X.400 AND RFC822 ADDRESSES, Computer Networks And ISDN Systems, VOL.13 Nr.3, 1987, pp.161-164.

<HOLM-85> Chris Holmes (British Telecom), OSI : CONFORMANCE & TESTING - A Carrier's View, Open System Interconnection Developments And Implementation, Online Publications, Pinner, UK, 1985, pp.89-94.

<HUTT-87> J. Hutton, STANDARDIZATION UPDATE, Computer Networks And ISDN Systems, VOL.13 Nr.3, 1987, pp.171-174.

<ISO-1446> ISO/TC97/SC21 N.1446, PROPOSAL FOR A NEW WORK ITEM ON CONFORMANCE TEST SUITES FOR FTAM, Novembre 1986.

<ISO-1525> ISO/TC97/SC21 N.1525 REV. (Draft Proposal ISO/DP 9646/1), INFORMATION PROCESSING SYSTEMS - OSI CONFORMANCE TESTING METHODOLOGY AND FRAMEWORK - Part1 : General Concepts, Janvier 1987.

<ISO-1526> ISO/TC97/SC21 N.1526 REV. (Draft Proposal ISO/DP 9646/2), INFORMATION PROCESSING SYSTEMS - OSI CONFORMANCE TESTING METHODOLOGY AND FRAMEWORK - Part2 : Abstract Test Suite Specification, Janvier 1987.

<IEEE-83> PROCEDINGS OF THE IEEE, OPEN SYSTEMS INTERCONNECTION (OSI) : NEW INTERNATIONNAL STANDARDS ARCHITECTURE AND PROTOCOLS FOR DISTRIBUTED INFORMATION SYSTEMS, VOL.71 Nr.12, Decembre 1983.

<JACO-87> T. Jacobsen, THE EUROPEAN COMMUNITY IS DEVELOPING CONFORMANCE TESTING SERVICE, Computer Networks And ISDN Systems, VOL.13 Nr.3, 1987, pp.175-180.

<KATO-86> T. Kato, K. Suzuki, Y. Urano, CONFORMANCE TESTING FOR OSI PROTOCOL IN THE MULTIPLE LAYER ENVIRONMENT BASED ON THE AUTOMATON MODEL, New Communication Services : A Challenge to Computer Technology, P.Kuhn (Editor), Elsevier Science Publishers B.V. (North-Holland), ICCC, 1986, pp.519-524.

<LABO-87> Bernard Laborie, UNE MESSAGERIE AUX NORMES X.400 SOUS UNIX : IMPLEMENTATION, INTERCONNEXION ET ADMINISTRATION, Thèse de Doctorat, Université Pierre et Marie Curie - Paris VI - Informatique, Septembre 1987.

<LONC-86> B. Lonc, GENEPIX : A PORTABLE VERSION UNDER UNIX (tm) OF THE OSI PROTOCOL TESTER GENEPI, Protocol Specification, Testing And Verification (V), M. Diaz (Editors) , Elsevier Science Publishers B.V. (North-Holland), (C) IFIP, 1986, pp.507-519.

<MACH-87> César Macchi, Jean-François Guilbert et 13 co-auteurs, TELEINFORMATIQUE, Transport et Traitement de l'informatique dans les réseaux et systèmes téléinformatiques et télématiques, DUNOD, Paris, 1987.

<PIQU-87> J-M. Piquet, AN INTRODUCTION TO SPAG SERVICES, Computer Networks And ISDN Systems, VOL.13 Nr.3, 1987, pp.181-186.

<PLAT-87> B. Plattner, E. A. Blau, T. Walter (ETH Zurich), PROSPECT - A TOOL FOR PROTOCOL SPECIFICATION AND CONFORMANCE TESTING, IFIP WG 6.5, Message Handling System, 1987, pp.(9.2)1-10.

<PUJO-85> G. Pujolle, D. Seret, D. Dromard, RESEAUX ET TELEMATIQUE, 2 Tomes, Eyrolles, Paris, 1985.

<RAFI-86> O. Rafiq, R. Castanet, TOWARDS AN ENVIRONMENT FOR TESTING OSI PROTOCOLS, Protocol Specification, Testing And Verification (V), M. Diaz (Editors), Elsevier Science Publishers B.V. (North-Holland), (C) IFIP, 1986, pp.533-544.

<RAVE-85> R. Ravel, D. J. Dwyer, SOME EXPERIENCE OF TESTING PROTOCOL IMPLEMENTATION, Protocol Specification, Testing And Verification (IV), R. Strom, S. Yemini, Y. Yemini (Editors), Elsevier Science Publishers B.V. (North-Holland), (C) IFIP, 1985, pp.657-677.

<RAYN-82> D. Rayner, A SYSTEM FOR TESTING PROTOCOL IMPLEMENTATION, Protocol Specification, Testing And Verification (I), C. Sunshine (Editor), North-Holland Publishing Company, (C) IFIP, 1982, pp.539-555.

<RAYN-86> D. Rayner, TOWARDS STANDARDIZED OSI CONFORMANCE TESTING, Protocol Specification, Testing And Verification (V), M. Diaz (Editors), Elsevier Science Publishers B.V. (North-Holland), (C) IFIP, 1986, pp.441-460.

<SARI-82> B. Sarikaya, G. v. Bochmann, SOME EXPERIENCE WITH TEST SEQUENCE GENERATION FOR PROTOCOLS, Protocol Specification, Testing And Verification (I), C. Sunshine (Editor), North-Holland Publishing Company, (C) IFIP, 1982, pp.555-567.

<SPAG-GUS> SPAG SERVICES - GUS - GUIDE TO USE OF STANDARDS, Specification of functional standards and guide to their implementation.

<SPAG-IOG> SPAG SERVICES - X.400 MHS INTEROPERABILITY GUIDE - V1.0 Guide to the testing of interoperability of X.400 Message Handling Systems implemented to ENV41201, 1986.

<SHNE-87> Ben Shneiderman, DESIGNING THE USER INTERFACE : Strategies for Effective Human-Computer Interaction, Addison-Wesley Publishing Company, Reading, Mass., 1987.

<TANN-81> A. S. Tannenbaum, COMPUTER NETWORKS, Prentice-Hall, Englewood Cliffs, 1981.

<TUCK-87> J. Tucker, NAMING AND ADRESSING ISSUES - OSI NETWORK ADDRESSES, Computer Networks And ISDN Systems, VOL.13 Nr.3, 1987, pp.149-155.

<VAX-MR> VAX/VMS MR, Message Router, Manager Guide, Installation Guide.

<VAX-MRIF> VAX/VMS MRIF, Message Router Interface Functions, How to write a Message Router Application, Installation Guide.

<VAX-MRX> VAX/VMS Message Router X400 Gateway, Manager Guide, Installation Guide.

<VAX-OSAK> VAX/VMS OSAK Vax OSI Applications Kernel, System manager's guide, User guide, Installation Guide.

<VAX-PSI> VAX/VMS PSI, Packetnet Switching Interface.

<VAX-VOTS> VAX/VMS VOTS, Vax OSI Transport Service, Introduction, User Guide, Installation guide.

<WILL-85> Robert Willmott, OSI : IMPLEMENTATION OF THE CCITT X.400 RECOMMANDATIONS FOR MHS, Open System Interconnection Developments And Implementation, Online Publications, Pinner, UK, 1985, pp.55-69.

<X.CNF> Benyt Ackzell, Swedish Telecom, CCITT, Draft Recommendation X.CNF, MESSAGE HANDLING : Conformance Testing (Version 1.2), London, Juin 1987.

<X.121> Recommandation X.121 du CCITT : Adressage Réseau.

<X.200> Recommandation X.200 du CCITT : Définition du modèle de référence pour interconnexion de systèmes ouverts (MODELE ISO), Novembre 1984.

<X.224> Recommandation X.224 du CCITT : Spécification du protocole Transport pour l'interconnexion de Systèmes Ouverts, Novembre 1984.

<X.225> Recommandation X.225 du CCITT : Spécification du protocole Session pour l'interconnexion de Systèmes Ouverts, Novembre 1984.

<X.400> Livre Rouge du CCITT, Tome VIII, Fascicule VIII.7, X.400 : SYSTEME DE MESSAGERIE : MODELE ET ELEMENTS DE SERVICE, Avis du CCITT concernant les systèmes de traitement de messages, Genève, 1985.

<X.401> Livre Rouge du CCITT, Tome VIII, Fascicule VIII.7, X.401 : SYSTEME DE MESSAGERIE : ELEMENTS DE SERVICE DE BASE ET ELEMENTS DE SERVICE SUPPLEMENTAIRES OPTIONNELS, Avis du CCITT concernant les systèmes de traitement de messages, Genève, 1985.

<X.403> CCITT Draft Recommendation X.403 Message Handling Systems : Conformance Testing (version 5.0), Glouchester, Novembre 1987

<X.408> Livre Rouge du CCITT, Tome VIII, Fascicule VIII.7, X.408 : SYSTEME DE MESSAGERIE : REGLES DE CONVERSION DE TYPE DE CODAGE, Avis du CCITT concernant les systèmes de traitement de messages, Genève, 1985.

<X.409> Livre Rouge du CCITT, Tome VIII, Fascicule VIII.7, X.409 : SYSTEME DE MESSAGERIE : SYNTAXE ET NOTATION DE TRANSFERT DE PRESENTATION, Avis du CCITT concernant les systèmes de traitement de messages, Genève, 1985.

<X.410> Livre Rouge du CCITT, Tome VIII, Fascicule VIII.7, X.410 : SYSTEME DE MESSAGERIE : OPERATIONS DISTANTE ET SERVEURS DE TRANSFERT FIABLE, Avis du CCITT concernant les systèmes de traitement de messages, Genève, 1985.

<X.411> Livre Rouge du CCITT, Tome VIII, Fascicule VIII.7, X.411 : SYSTEME DE MESSAGERIE : COUCHE TRANSFERT DE MESSAGES, Avis du CCITT concernant les systèmes de traitement de messages, Genève, 1985.

<X.420> Livre Rouge du CCITT, Tome VIII, Fascicule VIII.7, X.420 : SYSTEME DE MESSAGERIE : COUCHE AGENT UTILISATEUR DE MESSAGERIE DE PERSONNE A PERSONNE, Avis du CCITT concernant les systèmes de traitement de messages, Genève, 1985.

<X.430> Livre Rouge du CCITT, Tome VIII, Fascicule VIII.7, X.430 : SYSTEME DE MESSAGERIE : PROTOCOLES D'ACCES POUR TERMINAUX TELETEX, Avis du CCITT concernant les systèmes de traitement de messages, Genève, 1985.

<X.400/88> Documents de travail X.400/88 (Extraits).

<X.400/IG> X.400 Implementor's Guide, Version 5, CCITT Study Group VII, Octobre 1986.

<ZENG-85> H. X. Zeng, D. Rayner, R. Ravel, D. J. Dwyer, GATEWAY TESTING TECHNIQUES, Protocol Specification, Testing And Verification (IV), R.STROM, S.YEMINI, Y.YEMINI (Editors), Elsevier Science Publishers B.V. (NORTH-HOLLAND), (C) IFIP, 1985, pp.637-655.

ANNEXE 8.1 :

IMPLEMENTATION X.400 FAITE PAR DIGITAL

Cette annexe constitue une liste des divers éléments des recommandations X.400 et l'implémentation qui en a été faite par DIGITAL dans le logiciel MRX.
Cette liste ne présentant que les éléments posant problème, tout objet absent est donc entièrement supporté.

8.1 IMPLEMENTATION X.400 FAITE PAR DIGITAL

8.1.1 INTRODUCTION

Le 'Message Router X400 Gateway' (MRX) de DIGITAL implemente un sous-ensemble de la norme X400.
L'implementation MRX a ete concue pour interagir avec des systemes X400 qui respectent les standards du NBS, CEPT et CEN/CENELEC. (Ces standards n'etant pas entierement uniformes, MRX respecte le profil NBS lorsque des differences se presentent.)

8.1.2 PROTOCOLE P1

Cette section decrit l'implementation (protocole P1) des elements du MTS.

8.1.2.1 MTS ELEMENTS

Les elements du Message Transfer Service suivants sont pas supportes :

BASIC MTS :	Access management (non applicable)
	Registred encoded information types
SUBMISSION & DELIVERY :	Deferred delivery
	Deferred delivery cancellation
	Disclosure of other recipients
CONVERSION :	Explicit conversion
QUERY :	Probe
STATUS AND INFORM :	Hold for delivery

8.1.2.2 P1 ENVELOPE PROTOCOL ELEMENTS

ProbeENVELOPE	
ProbeMPDU	GEN : Jamais DEL : Jamais (sinon erreur) REL : Si present, le message ne sera pas relaye Les messages de Probe arrivants declencheront u echec.
EncodedInformationType	GEN : Toujours DEL : Si present, il sera ignore REL : Toujours
UAContentID	Supporte REL : Champs tronque a 16 caracteres.
PerMessageFlag	Supporte GEN : On ne genere que des flags d'un octet. REL : Aucune contrainte de longueur.
RecipientInfo	Obligatoire On ne verifie pas le nombre
IA5String	Obligatoire REL : Champs tronque a 32 caracteres
DiscloseRecipients	GEN : Toujours DEL : Si present, il sera ignore REL : Toujours mais pas acte
ConversionProhibited	GEN : Jamais DEL : Si Present, il sera acte REL : Si Present, il sera acte Si Positionne, les conversions implicites (DEC169 vers TTX, TTX vers DEC169) sont non permises. Pour TTX vers DEC169, une notification de non-livraison est generee "Conversion Prohibee" Pour DEC169 vers TTX, on utilise DEC vers IA5.
AlternateRecipientAllowed	Supporte (Jamais genere) Si positionne et si une boite aux lettres OPERATEUR existe, les messages mals adresses y seront postes (DEL et REL)
ContentReturnRequest	Non supporte (Aucune contrainte au RELais)
ExplicitConversion	Non supporte (Jamais genere) Voir TTX_OUT_FLAG
UserReportRequest	Obligatoire Seule la valeur 00 affecte le MTA, toutes les autres entrainent l'utilisation de RR.
Action	Obligatoire Toujours relaye
OriginalEncoded InformationTypes	Ils sont encodes par domaine via le parametre MRX 'Deliverable_types' - IA5 est toujours delivrable - TTX est delivrable si converti en DEC169
BilingInformation	Non supporte (Aucun test au relais)
SupplementaryInform.	Non supporte (Aucun test au relais)
TypeOfUA	Toujours genere pour un domaine prive

ORNAME Non contenu dans la directory MRX
X121 address, ù format 1.1 supporte
TerminalID, ù format 1.2 non delivre
UniqueUAIndentifier ù format 1.3 non delivre mais relaye

8.1.8 PROTOCOLE P2

Cette section decrit l'implementation (protocole P2) des elements du MTS.

8.1.8.1 BASIC IMP SERVICE ELEMENTS (P2 SUPPORT)

Les tables suivantes montrent l'etendue de l'implementation de MRX pour les elements du MTS.

S = Supporte
N = Non supporte
N/A = Non disponible
N/U = Non utilise
A = Service optionnel additionnel
E = Service optionnel essentiel

8.1.8.1.1 BASIC MESSAGE TRANSFERT SERVICE ELEMENTS

Service element	Envoi	Reception (par UA)
Access management	N/U	N/U
Content type indication	S	S
Converted indication	N/A	N (Rendu possible par MRX)
Delivery timestamp indication	N/A	S
Message indentification	S	S
Non-delivery notification	S	N/A
Original encoded information type indication	S	S
Registered encoded information types	N/A	N/U
Submission timestamp indication	S	S
IP-Message identification	S	S
Typed body	S	S

8.1.8.1.2 IMP OPTIONAL FACILITIES AGREED FOR A CONTRACTUAL PERIOD OF TIME

Service element	Categ.	Support
Alternate recipient Assignment	A	S
Hold for delivery	A	N
Implicit conversion	A	S (IA5 -- T61 seulement)

8.1.8.1.3 IMP OPTIONAL USER FACILITIES SELECTABLE ON A PER-MESSAGE BASIS

Service element	Envoi	Reception (par UA)	
Alternate recipient allowed	A (N)	A (N)	
Authorizing users indication	A (N)	E (N)	
Auto-forward indication	A (N)	E (N)	
Blind copy recipient indication	A (N)	E (S)	les BC sont traites comme des CC
Body part encryption indication	A (N)	E (N)	
Conversion prohibition	E (N)	E (S)	
Deferred delivery	E (N)	N/A	
Deferred delivery cancellation	A (N/U)	N/A	
Delivery notification	E (N)	N/A	
Disclosure of other recipients	A (N)	E (S)	
Expiry date indication	A (N)	E (N)	
Explicit conversion	A (N)	N/A	
Forwarded IP-Message indication	A (N)	E (S)	
Grade of delivery selection	E (S)	E (S)	
Importance indication	A (N)	E (S)	
Multi-destination delivery	E (S)	N/A	
Multi-part body	A (N)	E (S)	
Non-receipt notification	A (N)	A (N)	
Obsoleting indication	A (N)	E (N)	
Originator indication	E (S)	E (S)	
Prevention of non-delivery not.	A (N)	N/A	
Primary and copy recipients ind.	E (S)	E (S)	
Probe	A (N)	N/A	
Receipt notification	A (N)	A (N)	
Reply request indication	A (N)	E (N)	
Replying IP-Message indication	E (S)	E (S)	
Return of contents	A (N)	N/A	
Sensitivity indication	A (N)	E (N)	
Subject indication	E (S)	E (S)	

8.1.8.2 P2-HEADING PROTOCOL ELEMENTS

Les tables suivantes listent les accords SIG X400 NBS et l'implementation MRX des elements de protocole dans le P2-HEADING.

IM-UAPDU IP-message UAPDU
Tous les elements des protocoles P2 et non-P2 sont relayes.

HEADING	
Originator	Peut etre vide dans un message que l'on a fait suivre si l'envoyeur n'est pas dans le Gateway.
AuthorizingUsers	GEN: Jamais (le UA ne peut les afficher
BlindCopyRecipients	Les BCC sont consideres comme des CC.
InReplyTo, Obsoletes, CrossReferences	Geres par MRX pas par le UA (jamais de forward)

SR-UAPDU Status report UAPDU
Jamais generes, s'ils sont presents un erreur se produira.
Tous ses messages sont sauves dans un fichier d'erreur.

8.1.8.3 P2 PROTOCOL BODY PARTS

Les tables suivantes listent les accords SIG X400 NBS et l'implementation MRX des elements de protocole dans le P2-BODYPARTS.

Sont supportes les formats suivants : IA5TEXT, TTX et ForwardedIPMessag
Non supportes : TLX, Voice, G3FAX, TIF0, TIF1, Videotex, NationallyDefined, Encrypted, SDF

IA5Text	repertoire, IA5String		
TTX	numberOfPages	GEN: jamais	DEL: si present
	TelexCompatible	GEN: toujours	DEL: si present
	TeletexNonBasicParameters	GEN: jamais	DEL: si present
	GraphicsCharacterSets		
	ControlCharacterSets		
	PageFormats		
	MiscTerminalCapabilities		
	PrivateUse		
	Sequence (of T61String)	obligatoire	
	T61String	obligatoire	
Forwarded IPMessage			
	DeliveryInformation	GEN: jamais	DEL: ignore
	ContentType		
	Originator		
	Original		
	Priority		
	DeliveryFlags		
	OtherRecipients		
	ThisRecipient		
	IntendedRecipient		
	Converted		
	Submission		
	IM-UAPDU	GEN: toujours	DEL: toujours

8.1.8.4 P3 PROTOCOL SUPPORT

La version V1.1 du "Message Router X400 Gateway" ne supporte pas les abonnes P3.

8.1.4 RTS PROTOCOL LIMITATIONS

Element	Classe	Restriction	Commentaires
PConnect	M		CEPT max 512 bytes
DataTransferSyntax		valeur = 0	
PUserData	M		
ConnectionData	G		soit OPEN ou RECOVER
OPEN			
RTS User Data			
MTAname	G	32 caracteres max ù	
Password	G	64 caracteres max ù sous-ens graphique de IA5	
Null	G	generes si d'autres methodes de validation sont utilisees.	
RECOVER			
Session Connection ID	(G)		
CallingUser			
Reference	M	max 64 bytes incluant codage (62 bytes de T.61)	
Additional			
RefInfo	G	max 4 bytes incluant codage (2 bytes de T.61)	
PAccept	G		
DataTransfer			
Syntax	M	valeur = 0	

ANNEXE 8.2 :

STRUCTURE BNF D'UN MESSAGE

Cette annexe présente la structure BACHUS-NAUR (BNF) d'un message.
Cette structure est présentée en fonction de l'implémentation qui en a été faite par DIGITAL.

8.2 STRUCTURE BACKUS-NAUR D'UN MESSAGE X.400 (DIGITAL)

8.2.1. STRUCTURES

```
<message>           ::=  <user_message> ¦ <service_message>

<user_message>      ::=  <user_envelope> <user_content>

<service_message>   ::=  <service_envelope> <service_content>

<user_envelope>     ::=  [ <message_id> ]
                         [ <uacontid> ]
                         [ <hopcount> ]
                         [ <contenttypes> ]
                         [ <env_precedence> ]
                         [ <env_pdate> ]
                         [ <deferred> ]
                         [ <encodedtypes> ]
                         [ <permsgflg> ]
                         [ <sender_name> ]
                         <recipient> { <recipient> }

<service_envelope>  ::=  <message_id>
                         <hopcount>
                         <env_pdate>
                         <service_to_name>

<user_content>      ::=  <header>
                         { <body> }

<header>            ::=  [ <app_message_id> ]
                         [ <subject> ]
                         [ <in_reply_to> ]
                         [ <obsoletes> ]
                         [ <references> ]
                         [ <msgclass> ]
                         [ <cnt_precedence> ]
                         [ <autoforward> ]
                         [ <sensitivity> ]
                         [ <cnd_pdate> ]
                         [ <end_date> ]
                         [ <reply_by> ]
                         [ <create_date> ]
                         [ <from_name> ]
                         { <author_name> }
                         { <cnt_to> }
                         { <cnt_cc> }
                         { <cnt_bcc> }
                         { <reply_to_users> }

<body>              ::=  <text_bodypart> ¦
                         <forwarded_bodypart> ¦
                         <rmsfile_bodypart> ¦
                         <wpsplus_bodypart> ¦
                         <voicenotif_bodypart> ¦
                         <g3fax_bodypart> ¦
                         <tif0_bodypart> ¦
                         <tif1_bodypart> ¦
                         <voice_bodypart> ¦
                         <telex_bodypart> ¦
                         <teletex_bodypart> ¦
                         <videotex_bodypart> ¦
                         <odif_bodypart> ¦
                         <ddif_bodypart> ¦
                         <sfd_bodypart> ¦
                         <ia5text_bodypart> ¦
                         <vendor>

<service_content>   ::=  <message_id>
                         <uacontid>
                         <generated>
                         <info_report> { <info_report> }
                         [ <forwarded_bodypart> ]
```

8.2.2. NOMS

```
<sender_name>       ::=  <orname>

<recipient>         ::=  <env_to> ¦ <env_cc> ¦ <env_bcc>
<env_to>            ::=  <env_to_name>  ¦ <env_to_list>
<env_cc>            ::=  <env_cc_name>  ¦ <env_cc_list>
<env_bcc>           ::=  <env_bcc_name> ¦ <env_bcc_list>
<env_to_name>       ::=  <env_name>
<env_cc_name>       ::=  <env_name>
```

```
<env_bcc_name>      ::=  <env_name>

<env_name>          ::=  <orname>
                         <perrecflg>
                         [ <extensionid> ]
                         [ <explicconv> ]

<env_to_list>       ::=  [ <listname> ]
                         <perlistflg>
                         { <env_to> }

<env_cc_list>       ::=  [ <listname> ]
                         <perlistflg>
                         { <env_cc> }

<env_bcc_list>      ::=  [ <listname> ]
                         <perlistflg>
                         { <env_bcc> }

<service_to_name>   ::=  <orname>
<from_name>         ::=  <orname> ¦ <descriptive_name>
<author_name>       ::=  <orname> ¦ <descriptive_name>
<descriptive_name>  ::=  <freeform> [ <orname> ]

<cnt_to>            ::=  <cnt_to_name> ¦ <cnt_to_list>
<cnt_cc>            ::=  <cnt_cc_name> ¦ <cnt_cc_list>
<cnt_bcc>           ::=  <cnt_bcc_name> ¦ <cnt_bcc_list>

<reply_to_users>    ::=  <cnt_reply_to_users_name> ¦
                         <cnt_reply_to_users_list>

<cnt_to_name>       ::=  <form_1> ¦ <form_2>
<cnt_cc_name>       ::=  <form_1> ¦ <form_2>
<cnt_bcc_name>      ::=  <form_1> ¦ <form_2>

<cnt_reply_to_users_name>  ::=  <orname> [ <freeform> ]

<form_1>            ::=  <orname>
                         [ <reportflg> ]
                         [ <replyreq> ]

<form_2>            ::=  <descriptive_name>
                         [ <reportflg> ]
                         [ <replyreq> ]

<cnt_to_list>       ::=  [ <listname> ]
                         <perlistflg>
                         { <cnt_to> }

<cnt_cc_list>       ::=  [ <listname> ]
                         <perlistflg>
                         { <cnt_cc> }

<cnt_bcc_list>      ::=  [ <listname> ]
                         <perlistflg>
                         { <cnt_bcc> }

<cnt_reply_to_users_list>
                    ::=  [ <listname> ]
                         <perlistflg>
                         { <reply_to_users> }

<info_report>       ::=  <arrivaldate>
                         [ <a_report> ]
                         [ <report_name> ]
<a_report>          ::=  <delivered> ¦ <not_delivered>
<delivered>         ::=  <deliver_date>
<not_delivered>     ::=  <reason> <diagnostic>
<report_name>       ::=  <orname> <perrecflg> [ <extensionid> ]

<orname>            ::=  <mr_orname> ¦
                         <x400_orname_1> ¦
                         <x400_orname_2> ¦
                         <x400_orname_3> ¦
                         <x400_orname_4>

<mr_orname>         ::=  <mr_address>
                         [ <country> ]
                         [ <amdname> ]
                         [ <choice> ]

<mr_address>        ::=  <userid>
                         { <route> }
```

```
<choice>            ::=  [ <personal_name> ]
                         [ <prdname> ]
                         [ <orgname> ]
                         { <orgunit> }
                         [ <domainspecific> ]   REM: At least one!
<personal_name>     ::=  <surname>
                         [ <givenname> ]
                         [ <initials> ]
                         [ <generation> ]
<domainspecific>    ::=  [ <telephone> ]
                         [ <location> ]

<x400_orname_1>     ::=  <x121address>  [ <terminalid> ]  <mr_address>
<x400_orname_2>     ::=  <country> <amdname> <uniqueuaid> <mr_address>
                         { <domainspecific> }
<x400_orname_3>     ::=  <country> <amdname> <x121address> <mr_address>
                         { <domainspecific> }
<x400_orname_4>     ::=  <country> <amdname> <mr_address> <choice>
```

B.2.3. SYMBOLES TERMINAUX

```
<message_id>        ::=  STRING
<uacontid>          ::=  STRING
<app_message_id>    ::=  STRING
<subject>           ::=  STRING
<in_reply_to>       ::=  STRING
<obsoletes>         ::=  STRING
<references>        ::=  STRING
<msgclass>          ::=  STRING
<listname>          ::=  STRING
<freeform>          ::=  STRING
<generated>         ::=  STRING
<userid>            ::=  STRING
<route>             ::=  STRING
<telephone>         ::=  STRING
<location>          ::=  STRING
<orgname>           ::=  STRING
<orgunit>           ::=  STRING
<surname>           ::=  STRING
<givenname>         ::=  STRING
<initials>          ::=  STRING
<generation>        ::=  STRING
<x121address>       ::=  STRING
<terminalid>        ::=  STRING
<country>           ::=  STRING
<amdname>           ::=  STRING
<prdname>           ::=  STRING
<uniqueuaid>        ::=  STRING

<hopcount>          ::=  INTEGER
<contenttype>       ::=  INTEGER
<env_precedence>    ::=  INTEGER
<cnt_precedence>    ::=  INTEGER
<autoforward>       ::=  INTEGER
<sensitivity>       ::=  INTEGER
<extensionid>       ::=  INTEGER
<explicconv>        ::=  INTEGER
<replyreq>          ::=  INTEGER
<reason>            ::=  INTEGER
<diagnostic>        ::=  INTEGER

<env_pdate>         ::=  DATE
<deferred>          ::=  DATE
<cnt_pdate>         ::=  DATE
<enddate>           ::=  DATE
<reply_by>          ::=  DATE
<create_date>       ::=  DATE
<arrivaldate>       ::=  DATE
<deliverdate>       ::=  DATE

<encodedtype>       ::=  FLAGS
<permsgflg>         ::=  FLAGS
<perrecflg>         ::=  FLAGS
<perlistflg>        ::=  FLAGS
<reportflg>         ::=  FLAGS

<text_bodypart>          ::=  TEXT
<forwarded_bodypart>     ::=  <user_content>
<rmsfile_bodypart>       ::=  NBS
<wpsplus_bodypart>       ::=  NBS
<voicenotif_bodypart>    ::=  NBS
<g3fax_bodypart>         ::=  NBS
<tif0_bodypart>          ::=  NBS
<tif1_bodypart>          ::=  NBS
```

```
<voice_bodypart>          ::=  NBS
<telex_bodypart>          ::=  NBS
<teletex_bodypart>        ::=  NBS
<videotex_bodypart>       ::=  NBS
<odif_bodypart>           ::=  NBS
<ddif_bodypart>           ::=  NBS
<sfd_bodypart>            ::=  NBS
<ia5text_bodypart>        ::=  NBS

<vendor>             ::=  <vendor1_bodypart>
                          <vendor2_bodypart>
                          <vendor3_bodypart>
                          <vendor4_bodypart>
                          <vendor5_bodypart>
                          <vendor6_bodypart>
                          <vendor7_bodypart>
                          <vendor8_bodypart>
                          <vendor9_bodypart>
                          <vendor10_bodypart>
                          <vendor11_bodypart>
                          <vendor12_bodypart>
                          <vendor13_bodypart>
                          <vendor14_bodypart>
                          <vendor15_bodypart>
                          <vendor16_bodypart>

<vendor1_bodypart>        ::=  NBS
<vendor2_bodypart>        ::=  NBS
<vendor3_bodypart>        ::=  NBS
<vendor4_bodypart>        ::=  NBS
<vendor5_bodypart>        ::=  NBS
<vendor6_bodypart>        ::=  NBS
<vendor7_bodypart>        ::=  NBS
<vendor8_bodypart>        ::=  NBS
<vendor9_bodypart>        ::=  NBS
<vendor10_bodypart>       ::=  NBS
<vendor11_bodypart>       ::=  NBS
<vendor12_bodypart>       ::=  NBS
<vendor13_bodypart>       ::=  NBS
<vendor14_bodypart>       ::=  NBS
<vendor15_bodypart>       ::=  NBS
<vendor16_bodypart>       ::=  NBS
```

ANNEXE 8.3 :

CODAGE NBS ET X.409 D'UN MESSAGE

Cette annexe présente la représentation de deux messages différents en codage NBS et en codage X.409.
Pour plus de renseignements, veuillez vous référer aux documents définissant respectivement la structure NBS et le codage X.409.

8.8 CODAGE MBS ET X.409 D'UN MESSAGE

8.8.1 EXEMPLE DE CODAGE MBS

```
MSG[V2ENV]
  MSG[V2ENV]
    FIELD[MID]
      ASCII                                                    Len = 00000017
        21731170308891/11@VENUS
    FIELD[PDATE]
      DATE
        ASCII                                                  Len = 0000000E
          19880307113712
    FIELD[TO]
      SEQ
        ENT[NAME]
          SEQ
            ATTR[COUNTRY]
              ASCII                                            Len = 00000003
                USA %X555341
            ATTR[ADMINDOMAIN]
              ASCII                                            Len = 00000004
                AT&T %X41542654
            ATTR[PRIVATEDOM]
              ASCII                                            Len = 0000000B
                USA_MICK_01
            ATTR[PNAME]
              ATTR[SURNAME]
                ASCII                                          Len = 00000005
                  mouse %X6D6F757365
              ATTR[GIVENNAME]
                ASCII                                          Len = 00000006
                  mickey
              ATTR[INITIALS]
                ASCII                                          Len = 00000002
                  MM %X4D4D
              ATTR[GENERATION]
                ASCII                                          Len = 00000003
                  Jr. %X4A722E
          SEQ
            ATTR[TEL]
              ASCII                                            Len = 00000010
                123.123.13.14.14
            ATTR[LOCATION]
              ASCII                                            Len = 0000000A
                DISNEYLAND
            ATTR[USERID]
              ASCII                                            Len = 0000000B
                USA_MICK_01
          ATTR[PERRECFLG]
            BITS                                               Len = 00000005
              ..... %X00A8000000
    FIELD[UACONTID]
      ASCII                                                    Len = 0000001C
        ceci est l'identificateur UA
    FIELD[PREC]
      INT                                                      Len = 00000001
        . %X00
    FIELD[SENDER]
      ENT[NAME]
        SEQ
        SEQ
          ATTR[USERID]
            ASCII                                              Len = 00000009
              MRMANAGER
    FIELD[ITRACE]
      SEQ
        FIELD[MTA]
          ASCII                                                Len = 00000005
            VENUS %X56454E5553
        FIELD[ARVDATE]
          DATE
            ASCII                                              Len = 0000000E
              19880307113714
        FIELD[ACTION]
          INT                                                  Len = 00000001
            . %X00
    FIELD[HOPCOUNT]
      INT                                                      Len = 00000001
        . %X01
    FIELD[CONTENTDIA]
      INT                                                      Len = 00000001
        . %X01
    MSG[V2CONT]
```

```
    FIELD[SUBJ]
      ASCII                                                    Len = 00000020
        ceci est un test (adresse Bidon)
    FIELD[CRTDATE]
      DATE
        ASCII                                                  Len = 0000000E
          19880307113622
    FIELD[ATTACH]
      MSG[TEXT]
        ASCII                                                  Len = 00000060
          Ceci est un test (adresse Bidon)  pour avoir un dump des syntaxes
          de codage x409 et NBS.
```

8.8.2 EXEMPLE DE CODAGE X.409

```
CON_0
  CON_0
    SET
      ENCTYPES
        CON_0                                                  Len = 00000002
          . %X0520
      MID
        GDOM
          COUNTRY
            PRINTABLE                                          Len = 00000002
              BE %X4245
          ADMIN
            PRINTABLE                                          Len = 00000003
              RTT %X525454
          PRINTABLE                                            Len = 00000004
            FNDP %X464E4450
        IA5                                                    Len = 00000016
          00145120208891/88VENUS
      ORNAME
        SEQ
          COUNTRY
            PRINTABLE                                          Len = 00000002
              BE %X4245
          ADMIN
            PRINTABLE                                          Len = 00000003
              RTT %X525454
          CON_2
            PRINTABLE                                          Len = 00000004
              FNDP %X464E4450
          CON_5
            CON_0                                              Len = 00000009
              MRXTESTER
            CON_1                                              Len = 00000009
              MRXTESTER
            CON_2                                              Len = 00000003
              MRX %X4D5258
      CONTENT_TYPE                                             Len = 00000001
        . %X02
      UACONTID                                                 Len = 00000006
        uatest
      MTRACE
        SEQ
          GDOM
            COUNTRY
              PRINTABLE                                        Len = 00000002
                BE %X4245
            ADMIN
              PRINTABLE                                        Len = 00000003
                RTT %X525454
            PRINTABLE                                          Len = 00000004
              FNDP %X464E4450
          SET
            CON_0                                              Len = 0000000D
              880202164121Z
            CON_2                                              Len = 00000001
              . %X00
      CON_2
        SET
          CON_1                                                Len = 00000002
            .. %X03A8
          ORNAME
            SEQ
              COUNTRY
                PRINTABLE                                      Len = 00000002
                  be %X6265
              ADMIN
                PRINTABLE                                      Len = 00000003
                  rtt %X727474
```

```
            CON_2
              PRINTABLE                                          Len = 00000004
                iihe %X69696865
            CON_5
              CON_0                                              Len = 00000008
                reuviaux
              CON_1                                              Len = 00000006
                daniel
              CON_2                                              Len = 00000003
                drx %X647278
        CON_0                                                    Len = 00000001
          . %X01
  OCT                                                            Len = 000000BE
    ..1.k...12145120208891/185 2816000....0...1.........salut marcus!....tu p
    eux faire un REPLY!..merci.......Comment ca va a part ca?..Tu as les A600
    0 de daniel?....Bye....(Bon boulot!).........
```

ANNEXE 8.4 :

TRACE DES PREMIERS MESSAGES X.400

Ces premiers messages ont été envoyés le 6 janvier 1988 par le Message Router X.400 Gateway sur le VAX/VMS des Facultés Notre Dame de la Paix à Namur, et ont été reçus par le Message Router X.400 Gateway de l'Inter-university Institute for High Energies à Bruxelles.

8.4 PREMIERS MESSAGES ECHANGES ENTRE LES FACULTES N.D. DE LA PAIX ET L'IIHE

```
MESSAGE-ENVELOPE
   PDATE :              6-JAN-1988 15:40:09.00
   MESSAGE_ID :         13735160108891/410VENUS
   SENDER :
      COUNTRY :         BE
      AMDNAME :         RTT
      PRDNAME :         FNDP
      SURNAME :         MRMANAGER
      INITIALS :        MRM
   CONTENTTYPES :       %X00000002
   ENCODEDTYPES :       %X7FF452A0
   PRECEDENCE :         %X00000001
   TO :
      COUNTRY :         BE
      AMDNAME :         RTT
      PRDNAME :         IIHE
      SURNAME :         ELOY
      GIVENNAME :       MARC
      USERID :          MARC ELOY
      PERRECFLG :       %X000000D0
       ACTION       :   Y
       MRBASIC      :   N
       MRCONFIRMED  :   Y
       UABASIC      :   N
       UACONFIRMED  :   Y
      EXTENSIONID :     %X00000001
   HOPCOUNT :           %X00000001
MESSAGE-CONTENT
   APP_MESSAGE_ID :     35735160108891/65 2816000
   SUBJECT :
   TEXT :
        SGBNSGFNSGNS JJKJJ

MESSAGE-ENVELOPE
   PDATE :              6-JAN-1988 16:48:13.00
   MESSAGE_ID :         10646160108891/430VENUS
   SENDER :
      COUNTRY :         BE
      AMDNAME :         RTT
      PRDNAME :         FNDP
      SURNAME :         MRMANAGER
      INITIALS :        MRM
      USERID :          MRMANAGER
   CONTENTTYPES :       %X00000002
   ENCODEDTYPES :       %X7FF452A0
   UACONTID :       1ere connection
   PRECEDENCE :         %X00000001
   TO :
      COUNTRY :         BE
      AMDNAME :         RTT
      PRDNAME :         IIHE
      SURNAME :         eloy
      GIVENNAME :       marc
      USERID :          MARC ELOY
      PERRECFLG :       %X000000D0
       ACTION       :   Y
       MRBASIC      :   N
       MRCONFIRMED  :   Y
       UABASIC      :   N
       UACONFIRMED  :   Y
      EXTENSIONID :     %X00000001
   HOPCOUNT :           %X00000001
MESSAGE-CONTENT
   APP_MESSAGE_ID :     31646160108891/69 2816000
   SUBJECT :            1ere connection x400
   TEXT :
     Hello !

     Nous avons le plaisir de vous annoncer la reussite de la premiere
     transmission X.400 entre Namur (FNDP) et Bruxelles (IIHE).

     Nous refermons l'acces X29 de Namur.
     Nous continuons les tests.

     M.ELOY & D.REUVIAUX
```

ANNEXE 8.5 : LISTING DES PROGRAMMES

8.5.1 USER AGENT

Le User Agent a été développé par les auteurs sur le VAX/VMS de l'Inter-university Institute for High Energies à Bruxelles. Il comprend une procédure en langage DCL (UA.COM) qui joue le rôle de coordinateur entre le module de composition et d'envoi de messages (UA$SEND.PAS) et le module de lecture de mailbox MR et d'affichage de messages (UA$READ.PAS).

Les modules UA$SEND.PAS et UA$READ.PAS sont écrits en langage PASCAL VMS.

U A . C O M

```
$!
$!   UA - X.400 USER AGENT -
$!
$! AUTHORS:
$!      Marc ELOY & Daniel REUVIAUX.
$!
$! CREATION DATE: January 1988
$!
$! MODIFY BY :
$!
$! IDENT    INITLS      DATE      REASON
$!         Drx & Me  xx-Jan-88    original
$!
$ type sys$input
$ deck

       UA - X.400 USER AGENT - V1.0

       (c) DRX & ME jan'88.


$ eod
$!
$ on control_y then goto bye
$ file_created = 0
$!
$enter_command:
$ command = ""
$ inquire/nopunctuation command "UA> "
$ if command .eqs. ""  then goto enter_command
$ command = f$edit(command,"UPCASE")
$ if f$locate(command,"SEND") .eq. 0    then goto send
$ if f$locate(command,"EXIT") .eq. 0     then goto bye
$ if f$locate(command,"HELP") .eq. 0     then goto help
$ if f$locate(command,"READ") .eq. 0     then goto read
$ write sys$output -
   "UA-W-UNRCGNSD, Unrecognised Command ''command'"
$ goto enter_command
$!
$send:
$ present =  f$search("ua_send.tmp")
$ if f$length(present) .gt. 0 then goto skip_send_tmp
$ open/write/error=noopen cmdfile ua_send.tmp
$ write cmdfile "$ deassign sys$input"
$ write cmdfile "$ run [mrmanager]UA$SEND"
$ close cmdfile
$ file_created = 1
$skip_send_tmp:
$ @ua_send.tmp
$ goto enter_command
$!
$help:
$ @[mrmanager]UA$help.hlp
$ goto enter_command
$!
$read:
$ present =  f$search("ua_read.tmp")
$ if f$length(present) .gt. 0 then goto skip_read_tmp
$ open/write/error=noopen cmdfile ua_read.tmp
$ write cmdfile "$ deassign sys$input"
$ write cmdfile "$ run [mrmanager.mrxtester]MRXTESTER$read"
$ close cmdfile
$ file_created = 1
$skip_read_tmp:
$ @ua_read.tmp
$ goto enter_command
$!
$bye:
$ if file_created then delete/noconfirm ua_*.tmp;*
$ua_end:
$ write sys$output "End of UA"
$ write sys$output " "
```

U A S S E M D . P A S

```
[INHERIT('d18$:[mrmanager]MRIFDEFS.PEN','d18$:[mrmanager]STARLET.PEN')]

PROGRAM asmmsg(INPUT,OUTPUT);

{ Declare data types that are not standard Pascal }
TYPE
   str_type             = VARYING[133] OF CHAR;
   $QUAD                = [QUAD,UNSAFE] RECORD LO:UNSIGNED; L1:INTEGER; END;
   $UBYTE               = [BYTE] 0..255;
   $UWORD               = [WORD] 0..65535;

{ Declare variables }
VAR

   msgctx,                          { assemble context                 }
   lnkctx:          UNSIGNED;       { link context                     }
   envfile,                         { name of envelope file            }
   cntfile:         str_type;       { name of content file             }

   assemble,                        { assemble in progress (true/false) }
   connected:       BOOLEAN;        { connected to MR (true/false)      }


{  Declaration for Message Router Programmers Kit Routines.
   These are non-standard PASCAL declarations that use extensions to
   Pascal and are used in a similar way to the Run Time Library. The
   Interface Routines follow the VMS Procedure Calling Standard.
}

[EXTERNAL]
   FUNCTION mrif$start_assemble(VAR msgctxid : UNSIGNED;
                                    flag : UNSIGNED := %IMMED 0)
                                :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$select_assemble(msgctxid:UNSIGNED;
                                 msgtype:UNSIGNED:=%IMMED 0)
                                :UNSIGNED;EXTERNAL;
[EXTERNAL]
   FUNCTION mrif$end_assemble(msgctxid:UNSIGNED;
                              %DESCR contents:VARYING[a1] OF CHAR:=%IMMED 0;
                             %DESCR envelope:VARYING[b1] OF CHAR:=%IMMED 0)
                                :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$start_put(msgctxid:UNSIGNED;
                           class:UNSIGNED;
                           item:UNSIGNED)
                                :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$end_put(msgctxid:UNSIGNED):UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_string(msgctxid:UNSIGNED;
                            item:UNSIGNED;
                            %DESCR string:VARYING[a1] OF CHAR)
                            :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_integer(msgctxid:UNSIGNED;
                             item:UNSIGNED;
                             %REF number:ARRAY[u1..u2:INTEGER] OF $UBYTE;
                             size:$UWORD:=%IMMED 0)
                             :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_flags(msgctxid:UNSIGNED;
                           item:UNSIGNED;
                           flags:UNSIGNED)
                          :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_date(msgctxid : UNSIGNED;
                          item:UNSIGNED;
                          date:$QUAD)
                          :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_text_record(msgctxid:UNSIGNED;
                          %DESCR string:VARYING[a1] OF CHAR;
                          flags:UNSIGNED:=%IMMED 0)
```

```
                              :UNSIGNED;EXTERNAL;

    [EXTERNAL]
       FUNCTION mrif$put_text_file(msgctxid:UNSIGNED;
                                   %DESCR string:VARYING[a1] OF CHAR)
                                  :UNSIGNED;EXTERNAL;

    [EXTERNAL]
       FUNCTION mrif$start_nbs_put(msgctxid:UNSIGNED;
                                   nbstype:UNSIGNED;
                                   nbsqual:UNSIGNED) :UNSIGNED;EXTERNAL;

    [EXTERNAL]
       FUNCTION mrif$end_nbs_put(msgctxid:UNSIGNED):UNSIGNED;EXTERNAL;

    [EXTERNAL]
       FUNCTION mrif$put_nbs_record(msgctxid:UNSIGNED;
                                    %DESCR string:VARYING[a1] OF CHAR)
                                   :UNSIGNED;EXTERNAL;

    [EXTERNAL]
       FUNCTION mrif$connect(VAR lnkctx:UNSIGNED;
                            %DESCR node:VARYING[a1] OF CHAR:=%IMMED 0)
                            :UNSIGNED;EXTERNAL;

    [EXTERNAL]
       FUNCTION mrif$disconnect(lnkctx:UNSIGNED):UNSIGNED;EXTERNAL;

    [EXTERNAL]
       FUNCTION mrif$identify(lnkctx:UNSIGNED;
                              %DESCR mailbox:VARYING[a1] OF CHAR;
                              %DESCR password:VARYING[b1] OF CHAR:=%IMMED 0)
                              :UNSIGNED;EXTERNAL;

    [EXTERNAL]
       FUNCTION mrif$post(lnkctx:UNSIGNED;
                          %DESCR content:VARYING[a1] OF CHAR;
                          %DESCR envelope:VARYING[b1] OF CHAR:=%IMMED 0;
                          flags:UNSIGNED:=%IMMED 0;
                          %DESCR msgid:VARYING[c1] OF CHAR:=%IMMED 0)
                           :UNSIGNED;EXTERNAL;

    { Declare Run Time Library that wil be needed. }

    [EXTERNAL]FUNCTION LIB$STOP(%IMMED sts:UNSIGNED):UNSIGNED;EXTERNAL;

    [EXTERNAL]FUNCTION LIB$SIGNAL(%IMMED sts:UNSIGNED):UNSIGNED;EXTERNAL;


    PROCEDURE check(status_code:UNSIGNED);

    { Procedure to determine whether the status returned by another routine
      was a VMS success or error status. If the routine returned an error,
      the program terminates.

      Input : status_code - a 32 bit VMS status returned by another routine
      Implicit input msgctx or lnkctx
    }

    BEGIN
    if not odd(status_code)          { Bottom bit indicates success }
    then
        BEGIN
        if assemble then mrif$end_assemble(msgctx);
        if connected then mrif$disconnect(lnkctx);
        LIB$STOP(status_code);
        END;
    END;

    PROCEDURE assemble_message(VAR envfile,cntfile : str_type);

    {
      Procedure to assemble a message. The user is asked for information about
      the recipient of the message, and for the name of a text file.

      Note that all the compulsory elements are added to the message.

      OUTPUT : envfile, cntfile - names of envelope and content file that
               contain the message.

    }

    VAR
        ok      :UNSIGNED;
```

```
        buf    :ARRAY[1..16] OF $UBYTE;
        string :str_type;
        time   :$QUAD;

    BEGIN

    { Get names of files to asssemble }
    envfile:='ENVELOPE.NBS';
    cntfile:='CONTENT.NBS';

    { Start to assemble the message }
    ok:=mrif$start_assemble(msgctx);
    check(ok);
    assemble:=TRUE;

    { Assemble the UAPDU IM }
    ok:=mrif$select_assemble(msgctx,mrif$k_complete_message);
    check(ok);

    { Start to add a recipient O/R name }
    ok:=mrif$start_put(msgctx,mrif$k_name,mrif$k_to);
    check(ok);

    { Mandatory add a Given_name }
    REPEAT
       BEGIN
       write('Given name of person to send to            > ');
       readln(string);
       END
    UNTIL LENGTH(string)<>0;
    ok:=mrif$put_string(msgctx,mrif$k_givenname,string);
    check(ok);

    { Mandatory add a Surname }
    REPEAT
       BEGIN
       write('Surname of person to send to               > ');
       readln(string);
       END
    UNTIL LENGTH(string)<>0;
    ok:=mrif$put_string(msgctx,mrif$k_surname,string);
    check(ok);

    { Adding optionnal personnal elements }
    write('Initials [optional]                        > ');
    readln(string);
    if LENGTH(string)<>0
    then
        BEGIN
        ok:=mrif$put_string(msgctx,mrif$k_initials,string);
        check(ok);
        END;

    write('Generation [optional]                      > ');
    readln(string);
    if LENGTH(string)<>0
    then
        BEGIN
        ok:=mrif$put_string(msgctx,mrif$k_generation,string);
        check(ok);
        END;

    write('Telephone [optional]                       > ');
    readln(string);
    if length(string)<>0
    then
        BEGIN
        ok:=mrif$put_string(msgctx,mrif$k_telephone,string);
        check(ok);
        END;

    write('Location [optional]                        > ');
    readln(string);
    if length(string)<>0
    then
        BEGIN
        ok:=mrif$put_string(msgctx,mrif$k_location,string);
        check(ok);
        END;

    write('Country [BE]                               > ');
    readln(string);
    if length(string)=0 then string:='BE';
    ok:=mrif$put_string(msgctx,mrif$k_country,string);
```

```
check(ok);

write('Administration domain [RTT]                > ');
readln(string);
if length(string)=0 then string:='RTT';
ok:=mrif$put_string(msgctx,mrif$k_amdname,string);
check(ok);

write('Private domain [FNDP]                      > ');
readln(string);
if length(string)=0 then string:='FNDP';
ok:=mrif$put_string(msgctx,mrif$k_prdname,string);
check(ok);


{ Adding the Message Router Address, that is the destination mailbox for
  this message. If the recipient is a local user, this is the name of his
  mailbox. If the recipient is a remote X.400 user, this is the name of
  the mailbox attached to his PRDName }

repeat
   write('Message Router Address                    > ');
   readln(string);
until length(string)<>0;
ok:=mrif$put_string(msgctx,mrif$k_userid,string);
check(ok);

{ End the name }
ok:=mrif$end_put(msgctx);
check(ok);
writeln;

{ Compose the bodyparts }

{ Enter the Subject }
writeln;
write('subject > ');
readln(string);
if (length(string)<>0) then ok:=mrif$put_string(msgctx,mrif$k_subject,string);
check(ok);

{ Start a text bodypart }
ok:=mrif$start_put(msgctx,mrif$k_bodypart,mrif$k_text);
check(ok);

{ Start the text }
string:=' ';
writeln('Enter your text below, "." when ready.');
while (string <> '.') do
BEGIN
   readln(string);
   string:=string+' ';
   ok:=mrif$put_text_record(msgctx,string);
   check(ok);
END;
writeln;

{ Optionally add a text file to the same bodypart }
write('Filename of the text to add to the message [optional] > ');
readln(string);
if length(string)<>0
then
    BEGIN
    { output any errors found here, but continue program }
    ok:=mrif$put_text_file(msgctx,string);
    if not odd(ok) then LIB$SIGNAL(ok);
    END;

{ End of the bodypart }
ok:=mrif$end_put(msgctx);
check(ok);
writeln;

{ Write the message out to the envelope and content files }
ok:=mrif$end_assemble(msgctx,cntfile,envfile);
assemble:=FALSE;
check(ok);
END;


PROCEDURE connect_mr;

{ Procedure to call the CONNECT routine to connect to Message Router on a
  given node. If you specify a node, a DECnet connection is made to that
```

```
  node. Otherwise the shareable Message Router image on your local node is
  used.

  Implicit Output : lnkctx
}

VAR
   ok       :UNSIGNED;
   node     :str_type;

BEGIN
     node:='VENUS';
     writeln('Connecting to the default node ... ');
     ok:=mrif$connect(lnkctx,node);
     if not odd(ok)
     then BEGIN
          writeln('Unable to connect to Message Router on node ',node);
          writeln('Your message cannot be processed. Please try to sent');
          writeln('again when Message Router is running.');
          $EXIT;
          END;
     connected:=TRUE;
END;


PROCEDURE identify_mr;

{
  Procedure to identify to a Message Router mailbox. Note that the IDENTIFY
  routine returns the full name of the mailbox.

  Implicit Input : lnkctx
}

VAR
    ok     :UNSIGNED;
    mbx,
    pwd    :str_type;

BEGIN
     writeln('Using your default mailbox ... ');
     mbx:='';
     pwd:='';
     ok:=mrif$identify(lnkctx,mbx);

     if not odd(ok)
     then writeln('Unable to identify to mailbox ',mbx);
     writeln('Identified to mailbox ',mbx);
END;


PROCEDURE post_mr(VAR envfile,cntfile:str_type);

{
  Procedure to post a message into Message Router. The message_id that
  Message Router adds to the message is returned and displayed.

  INPUT : envfile, cntfile - then names of the envelope and content files
                             to post
  Implicit Input : lnkctx
}

VAR
    ok       :UNSIGNED;
    mid      :str_type;

BEGIN
ok:=mrif$post(lnkctx,cntfile,envfile,0,mid);
check(ok);
writeln('Message posted successfully with message id ',mid);
END;


PROCEDURE disconnect_mr;

{
 Procedure to disconnect from Message Router.

 Implicit Input : lnkctx
}

VAR
    ok      :UNSIGNED;
```

```
BEGIN
ok:=mrif$disconnect(lnkctx);
connected:=FALSE;
check(ok);
END;


{ MAIN ROUTINE }
BEGIN

        assemble:=FALSE;
        connected:=FALSE;
        writeln;
        writeln('This program allows you to create and post a message to a loca
l or');
        writeln('remote user. It will prompt you for some informations. If you
want to ');
        writeln('accept a default or skip an optionnal information, simply pres
s');
        writeln('<RETURN>, otherwise type your reply and press <RETURN>.');
        writeln;
        assemble_message(envfile,cntfile);
        connect_mr;
        identify_mr;
        post_mr(envfile,cntfile);
        disconnect_mr;

END.
```

READ.PAS

```
[INHERIT('sys$library:MRIFDEFS.PEN','sys$library:STARLET.PEN')]
PROGRAM mrxtester$read(INPUT,OUTPUT);

{ Declare data types that are not standard Pascal }
TYPE
   str_type              = VARYING[256] OF CHAR;
   small_byte            = [BYTE] 1..16;
   $QUAD                 = [QUAD,UNSAFE] RECORD
                             LO:UNSIGNED; L1:INTEGER; END;
   $UBYTE                = [BYTE] 0..255;
   $UWORD                = [WORD] 0..65535;

{ Declare variables }
VAR
   out: TEXT;                       { identifies the output file }
   msgctx,                          { disassemble context }
   lnkctx,                           { link context }
   ok,                              { general status variable containing
                                      value returned }
   class,                           { mrif$k_class }
   item,                            { mrif$k_item }
   find_length,                     { length returned by find call }
   bit_value,                       { value of bit returned by GET_FLAGS }
   nbsqual,                         { qualifier returned by START_NBS_GET }
   nbstype:                         { type returned by START_NBS_GET }
              UNSIGNED;

   ascii_item: VARYING [20] OF CHAR; { ASCII version of mrif$k_item }
   int_length,                      { length returned by GET_INTEGER call }
   length_cnt:                      { used as a FOR variable comparing to
                                      int_length }
              $UWORD;
   integer_value:ARRAY[1..16]OF $UBYTE; { value of integer returned by
                                          GET_INTEGER }
   string_value:VARYING[60]OF CHAR; { value of string returned by
                                      GET_STRING }
   date_value:$QUAD;                { value of date returned by GET_DATE }
   ascii_date:PACKED ARRAY[1..26] OF CHAR;
                                    { ASCII version of date_value }
   date_length:$UWORD;               { length returned by $ASCTIM }

   assemble, connected :BOOLEAN;
   outfile,
   envfile,
   cntfile,
   mbx,mbx2,yn          :str_type;
   number,i             :UNSIGNED;
   j                    :INTEGER;

{  Declaration for Message Router Programmers Kit Routines.
   These are non-standard PASCAL declarations that use extensions to
   Pascal and are used in a similar way to the Run Time Library. The
   Interface Routines follow the VMS Procedure Calling Standard.
}

[EXTERNAL]
   FUNCTION mrif$connect(VAR lnkctx : UNSIGNED;
            %DESCR node : VARYING[a1] OF CHAR := %IMMED 0)
                              :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$disconnect(lnkctx : UNSIGNED) :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$identify(lnkctx:UNSIGNED;
            %DESCR mailbox : VARYING[a1] OF CHAR;
            %DESCR password : VARYING[b1] OF CHAR := %IMMED 0)
                              :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$start_disassemble(VAR msgctxid : UNSIGNED;
            %DESCR contents : VARYING[a1] OF CHAR := %IMMED 0;
            %DESCR envelope : VARYING[b1] OF CHAR := %IMMED 0)
                              :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$select_disassemble(msgctxid:UNSIGNED;
                               msgtype:UNSIGNED:=%IMMED 0)
                              :UNSIGNED;EXTERNAL;
[EXTERNAL]
   FUNCTION mrif$end_disassemble(msgctxid:UNSIGNED)
                              :UNSIGNED;EXTERNAL;
```

```
[EXTERNAL]
   FUNCTION mrif$find(msgctxid : UNSIGNED;
                      VAR class : UNSIGNED;
                      VAR item  : UNSIGNED;
                      range : UNSIGNED;
                      VAR length : UNSIGNED)
                          :UNSIGNED;EXTERNAL;
[EXTERNAL]
   FUNCTION mrif$start_get(msgctxid:UNSIGNED;
                      VAR class:UNSIGNED;
                      VAR item:UNSIGNED)
                            :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$end_get(msgctxid:UNSIGNED):UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$get_string(msgctxid:UNSIGNED;
                            item:UNSIGNED;
                            %DESCR value:VARYING[a1] OF CHAR;
                            VAR length : $UWORD := %IMMED 0)
                            :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$get_integer(msgctxid:UNSIGNED;
                             item:UNSIGNED;
                             %REF number:ARRAY[u1..u2:INTEGER] OF $UBYTE;
                             VAR length:$UWORD:=%IMMED 0)
                             :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$get_flags(msgctxid:UNSIGNED;
                           item:UNSIGNED;
                           VAR value :UNSIGNED)
                          :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$get_date(msgctxid : UNSIGNED;
                          item:UNSIGNED;
                          VAR value :$QUAD)
                          :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$get_text_record(msgctxid:UNSIGNED;
                          %DESCR value:VARYING[a1] OF CHAR;
                          options:UNSIGNED:=%IMMED 0;
                          VAR length : $UWORD := %IMMED 0)
                         :UNSIGNED;EXTERNAL;


[EXTERNAL]
   FUNCTION mrif$start_nbs_get(msgctxid:UNSIGNED;
                           VAR nbstype:UNSIGNED;
                           VAR nbsqual:UNSIGNED) :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$end_nbs_get(msgctxid:UNSIGNED):UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$get_nbs_record(msgctxid:UNSIGNED;
                                %DESCR value:VARYING[a1] OF CHAR;
                                VAR length : $UWORD := %IMMED 0)
                               :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$isprim(nbstype : UNSIGNED):BOOLEAN;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$fetch(lnkctx:UNSIGNED;
        %DESCR content:VARYING[a1] OF CHAR;
        %DESCR envelope:VARYING[b1] OF CHAR := %IMMED 0;
        flags:UNSIGNED := %IMMED 0)
           :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$query(lnkctx : UNSIGNED;
                       VAR number : UNSIGNED)
          :UNSIGNED;EXTERNAL;

{ Declare Run Time Library that wil be needed. }

[EXTERNAL]FUNCTION LIB$STOP(%IMMED sts:UNSIGNED):UNSIGNED;EXTERNAL;
```

```
[EXTERNAL]FUNCTION LIB$SIGNAL(%IMMED sts:UNSIGNED):UNSIGNED;EXTERNAL;


PROCEDURE check (status_code : UNSIGNED);

{
  Procedure to determine whether the status returned by another routine
  was a VMS success or error status. If the routine returned an error,
  the program terminates.

  INPUT : status_code - a 32 bit VMS status returned by another routine
  Implicit Input : msgctx,out
}

BEGIN
IF NOT ODD(status_code)          { Bottom bit indicates success }
THEN
   BEGIN
   mrif$end_disassemble(msgctx);
   close(out);
   writeln('MRXTESTER-F-FTLERR, ***** FATAL ERROR *****');
   writeln('The program will terminate due to the reason bellow :');
   writeln;
   LIB$SIGNAL(status_code);
   $exit;
   END;
END;

PROCEDURE init(messagefile:str_type);

{
 This procedure prompts the user for the name of the file to be
 disassembled and the name of the file to send the output to. If the user
 does not specify a file for the output, the message is displayed on the
 terminal screen. The procedure determines whether the message file
 contains the envelope or the content of a message and starts to
 disassemble it. It calls SELECT_DISASSEMBLE to see whether the message is
 a service message or a user message.

 Implicit Output : msgctx - message is left started and selected
                   out - output file is left open for write
}

VAR
    envelope:BOOLEAN;

BEGIN

REPEAT

    { Quit if no more files to look at }
    IF messagefile.length = 0 THEN $EXIT(ss$_normal);

    { Assume message file contains an envelope }
    ok:=mrif$start_disassemble(msgctx,,messagefile);
    envelope:=TRUE;
    IF ok = mrif$_invalidmsg
    THEN
        BEGIN
        { It is not an envelope, so assume it is a content }
        ok:=mrif$start_disassemble(msgctx,messagefile);
        envelope:=FALSE;
        END;
    IF NOT ODD(ok)
    THEN
        BEGIN
        { Message file does not contain an envelope or a content,
          so it cannot be disassembled
        }
        WRITELN('Unable to disassemble message in ',messagefile);
        LIB$SIGNAL(ok);
        END;
UNTIL ODD(ok);

{
  Prompt for the file where the message should be written. If you do not
  give a file, the message will be displayed on your screen.
}


IF outfile.length = 0 THEN outfile:='TT:';

IF envelope
```

```
    THEN
        BEGIN
        {
          File contains an envelope. Call SELECT to see if it is a service
          message or a user message.
         }
         ok:=mrif$select_disassemble(msgctx,mrif$k_envelope);
         check(ok);
         IF ok=mrif$_envelope
         THEN WRITELN(out,'MESSAGE-ENVELOPE')
         ELSE WRITELN(out,'SERVICE-MESSAGE-ENVELOPE');
         END
    ELSE
        BEGIN
        {
          File contains a select. Call SELECT to see if it is a service
          message or a user message.
        }
        ok:=mrif$select_disassemble(msgctx,mrif$k_content);
        check(ok);
        IF ok=mrif$_content
        THEN WRITELN(out,'MESSAGE-CONTENT')
        ELSE WRITELN(out,'SERVICE-MESSAGE-CONTENT');
        END;
    END;


    PROCEDURE finish;

    {
      Procedure to tidy up storage available for disassembling this message
      file.

      Implicit Input : msgctx - left unusable
                       out    - left closed.
    }

    BEGIN
    ok:=mrif$end_disassemble(msgctx);
    check(ok);
    END;


    PROCEDURE item_to_ascii( item : UNSIGNED;
                             VAR ascii_item : VARYING[a1] OF CHAR );

    {
      Procedure to return the name of an element in the message, given its item
      code.

      INPUT : item - mrif$k_item
      OUTPUT : ascii_item - readable description of mrif$k_item
    }

    BEGIN
         CASE INT(item) OF    { item is unsigned, constants are integers }

    mrif$k_amdname:          ascii_item:='AMDNAME       ';
    mrif$k_app_message_id:   ascii_item:='APP_MESSAGE_ID';
    mrif$k_arrivaldate:      ascii_item:='ARRIVALDATE   ';
    mrif$k_author:           ascii_item:='AUTHOR        ';
    mrif$k_autoforward:      ascii_item:='AUTOFORWARD   ';
    mrif$k_bcc:              ascii_item:='BCC           ';
    mrif$k_cc:               ascii_item:='CC            ';
    mrif$k_contenttypes:     ascii_item:='CONTENTTYPES  ';
    mrif$k_country:          ascii_item:='COUNTRY       ';
    mrif$k_date:             ascii_item:='DATE          ';
    mrif$k_ddif:             ascii_item:='DDIF          ';
    mrif$k_deferred:         ascii_item:='DEFERRED      ';
    mrif$k_deliverdate:      ascii_item:='DELIVERDATE   ';
    mrif$k_diagnostic:       ascii_item:='DIAGNOSTIC    ';
    mrif$k_encodedtypes:     ascii_item:='ENCODEDTYPES  ';
    mrif$k_enddate:          ascii_item:='ENDDATE       ';
    mrif$k_explicconv:       ascii_item:='EXPLICCONV    ';
    mrif$k_extensionid:      ascii_item:='EXTENSIONID   ';
    mrif$k_forwarded:        ascii_item:='FORWARDED     ';
    mrif$k_freeform:         ascii_item:='FREEFORM      ';
    mrif$k_from:             ascii_item:='FROM          ';
    mrif$k_g3fax:            ascii_item:='G3FAX         ';
    mrif$k_generation:       ascii_item:='GENERATION    ';
    mrif$k_generated:        ascii_item:='GENERATED     ';
    mrif$k_givenname:        ascii_item:='GIVENNAME     ';
    mrif$k_hopcount:         ascii_item:='HOPCOUNT      ';
    mrif$k_ia5text:          ascii_item:='IA5TEXT       ';
```

```
        mrif$k_info:            ascii_item:='INFO          ';
        mrif$k_initials:        ascii_item:='INITIALS      ';
        mrif$k_inreplyto:       ascii_item:='INREPLYTO     ';
        mrif$k_listname:        ascii_item:='LISTNAME      ';
        mrif$k_location:        ascii_item:='LOCATION      ';
        mrif$k_message_id:      ascii_item:='MESSAGE_ID    ';
        mrif$k_msgclass:        ascii_item:='MSGCLASS      ';
        mrif$k_obsoletes:       ascii_item:='OBSOLETES     ';
        mrif$k_odif:            ascii_item:='ODIF          ';
        mrif$k_orgname:         ascii_item:='ORGNAME       ';
        mrif$k_orgunit:         ascii_item:='ORGUNIT       ';
        mrif$k_pdate:           ascii_item:='PDATE         ';
        mrif$k_perlistflg:      ascii_item:='PERLISTFLG    ';
        mrif$k_permsgflg:       ascii_item:='PERMSGFLG     ';
        mrif$k_perrecflg:       ascii_item:='PERRECFLG     ';
        mrif$k_prdname:         ascii_item:='PRDNAME       ';
        mrif$k_precedence:      ascii_item:='PRECEDENCE    ';
        mrif$k_reason:          ascii_item:='REASON        ';
        mrif$k_references:      ascii_item:='REFERENCES    ';
        mrif$k_replyby:         ascii_item:='REPLYBY       ';
        mrif$k_replyreq:        ascii_item:='REPLYREQ      ';
        mrif$k_replytousers:    ascii_item:='REPLYTOUSERS  ';
        mrif$k_reportflg:       ascii_item:='REPORTFLG     ';
        mrif$k_rmsfile:         ascii_item:='RMSFILE       ';
        mrif$k_route:           ascii_item:='ROUTE         ';
        mrif$k_sender:          ascii_item:='SENDER        ';
        mrif$k_sensitivity:     ascii_item:='SENSITIVITY   ';
        mrif$k_sfd:             ascii_item:='SFD           ';
        mrif$k_subject:         ascii_item:='SUBJECT       ';
        mrif$k_surname:         ascii_item:='SURNAME       ';
        mrif$k_telephone:       ascii_item:='TELEPHONE     ';
        mrif$k_teletex:         ascii_item:='TELETEX       ';
        mrif$k_telex:           ascii_item:='TELEX         ';
        mrif$k_terminalid:      ascii_item:='TERMINALID    ';
        mrif$k_text:            ascii_item:='TEXT          ';
        mrif$k_tif0:            ascii_item:='TIF0          ';
        mrif$k_tif1:            ascii_item:='TIF1          ';
        mrif$k_to:              ascii_item:='TO            ';
        mrif$k_uacontid:        ascii_item:='UACONTID      ';
        mrif$k_uniqueuaid:      ascii_item:='UNIQUEUAID    ';
        mrif$k_userid:          ascii_item:='MR_ADDRESS    ';
        mrif$k_vendor1:         ascii_item:='VENDOR1       ';
        mrif$k_vendor2:         ascii_item:='VENDOR2       ';
        mrif$k_vendor3:         ascii_item:='VENDOR3       ';
        mrif$k_vendor4:         ascii_item:='VENDOR4       ';
        mrif$k_vendor5:         ascii_item:='VENDOR5       ';
        mrif$k_vendor6:         ascii_item:='VENDOR6       ';
        mrif$k_vendor7:         ascii_item:='VENDOR7       ';
        mrif$k_vendor8:         ascii_item:='VENDOR8       ';
        mrif$k_vendor9:         ascii_item:='VENDOR9       ';
        mrif$k_vendor10:        ascii_item:='VENDOR10      ';
        mrif$k_vendor11:        ascii_item:='VENDOR11      ';
        mrif$k_vendor12:        ascii_item:='VENDOR12      ';
        mrif$k_vendor13:        ascii_item:='VENDOR13      ';
        mrif$k_vendor14:        ascii_item:='VENDOR14      ';
        mrif$k_vendor15:        ascii_item:='VENDOR15      ';
        mrif$k_vendor16:        ascii_item:='VENDOR16      ';
        mrif$k_videotex:        ascii_item:='VIDEOTEX      ';
        mrif$k_voice:           ascii_item:='VOICE         ';
        mrif$k_voicenotif:      ascii_item:='VOICENOTIF    ';
        mrif$k_wpsplus:         ascii_item:='WPSPLUS       ';
        mrif$k_x121address:     ascii_item:='X121ADDRESS   ';
        OTHERWISE
                                ascii_item:='UNKNOWN       '
        END;
        END;


        PROCEDURE diagnostic_explication(VAR ent : $UBYTE);
        {
          Procedure to return the text of the 'diagnostic' item
        }
        BEGIN
           CASE INT(ent) OF
              0:write(out,' Unrecognized recipient name');
              1:write(out,' Ambiguous recipient name');
              2:write(out,' MTA congestion');
              3:write(out,' Hopcount exceeded');
              4:write(out,' User Agent unavailable');
              5:write(out,' Max time expired');
              6:write(out,' Unsupported encoded information type');
              7:write(out,' Content to long for User Agent');
              8:write(out,' Conversion impratical');
              9:write(out,' Conversion prohibited');
```

```
        10:write(out,' Implicit conversion not registrated');
        11:write(out,' Invalid parameters');
        32:write(out,' Directory loop detected');
        33:write(out,' No actionable recipients');
        34:write(out,' Message purged by operator');
      OTHERWISE
        write(out,' UNKNOWN!');
      END;
    END;


    PROCEDURE reason_explication(VAR ent : $UBYTE);
    {
      Procedure to return the text of the 'reason' item
    }
    BEGIN
      CASE INT(ent) OF
        0:write(out,' Transfer Failure');
        1:write(out,' Unable to Transfer');
        2:write(out,' Conversion not performed');
      OTHERWISE
        write(' UNKNOWN!');
      END;
    END;

    PROCEDURE out_flags (spaces : str_type;item,bit_value : UNSIGNED);

    {
      Procedure to output the details on the bits defined in a flags longword.

      Note the non-standard PASCAL command (UAND) used to do unsigned bit by
      bit comparaisons. It could be done instead using ODD and DIV, or by
      using SETs.

      INPUT : spaces - string containing number of spaces last output
              item - mrif$k_item of flag
              bit_value - flags value contained in mrif$k_item
      Implicit Input : out
      Implicit Output : out
    }

    BEGIN

        CASE INT(item) OF       { item is unsigned, constants are integers. }

    mrif$k_perrecflg:
    BEGIN
    WRITE(out,spaces,'   ACTION     : ');
    IF UAND(bit_value,mrif$m_action)>0 THEN WRITELN(out,'Y')
                                       ELSE WRITELN(out,'N');
    WRITE(out,spaces,'   MRBASIC    : ');
    IF UAND(bit_value,mrif$m_mr_basic)>0 THEN WRITELN(out,'Y')
                                         ELSE WRITELN(out,'N');
    WRITE(out,spaces,'   MRCONFIRMED: ');
    IF UAND(bit_value,mrif$m_mr_confirmed)>0 THEN WRITELN(out,'Y')
                                             ELSE WRITELN(out,'N');
    WRITE(out,spaces,'   UABASIC    : ');
    IF UAND(bit_value,mrif$m_ua_basic)>0 THEN WRITELN(out,'Y')
                                         ELSE WRITELN(out,'N');
    WRITE(out,spaces,'   UACONFIRMED: ');
    IF UAND(bit_value,mrif$m_ua_confirmed)>0 THEN WRITELN(out,'Y')
                                             ELSE WRITELN(out,'N');
    END;

    mrif$k_permsgflg:
    BEGIN

    WRITE(out,spaces,'   DISCLOSEREC: ');
    IF UAND(bit_value,mrif$m_discloserec)>0 THEN WRITELN(out,'Y')
                                            ELSE WRITELN(out,'N');
    WRITE(out,spaces,'   CONVPROHIB : ');
    IF UAND(bit_value,mrif$m_convprohib)>0 THEN WRITELN(out,'Y')
                                           ELSE WRITELN(out,'N');
    WRITE(out,spaces,'   ALTRECIP   : ');
    IF UAND(bit_value,mrif$m_altrecip)>0 THEN WRITELN(out,'Y')
                                         ELSE WRITELN(out,'N');
    WRITE(out,spaces,'   CONTRETREQ : ');
    IF UAND(bit_value,mrif$m_contretreq)>0 THEN WRITELN(out,'Y')
                                           ELSE WRITELN(out,'N');
    END;

    mrif$k_perlistflg:

    BEGIN
```

```
WRITE(out,spaces,'   EXPANDEDLST: ');
IF UAND(bit_value,mrif$m_expandedlist)>0 THEN WRITELN(out,'Y')
                                         ELSE WRITELN(out,'N');
END;

mrif$k_reportflg:
BEGIN

WRITE(out,spaces,'   RECNOTIF    : ');
IF UAND(bit_value,mrif$m_recnotif)>0 THEN WRITELN(out,'Y')
                                     ELSE WRITELN(out,'N');
WRITE(out,spaces,'   NONRECNOTIF: ');
IF UAND(bit_value,mrif$m_nonrecnotif)>0 THEN WRITELN(out,'Y')
                                        ELSE WRITELN(out,'N');
WRITE(out,spaces,'   RETURNMSG   : ');
IF UAND(bit_value,mrif$m_returnmsg)>0 THEN WRITELN(out,'Y')
                                      ELSE WRITELN(out,'N');
END;


mrif$k_encodedtypes:

BEGIN

WRITE(out,spaces,'   RMSFILE     : ');
IF UAND(bit_value,mrif$m_rmsfile)>0 THEN WRITELN(out,'Y')
                                    ELSE WRITELN(out,'N');
WRITE(out,spaces,'   WPSPLUS     : ');
IF UAND(bit_value,mrif$m_wpsplus)>0 THEN WRITELN(out,'Y')
                                    ELSE WRITELN(out,'N');
WRITE(out,spaces,'   DDIF        : ');
IF UAND(bit_value,mrif$m_ddif)>0 THEN WRITELN(out,'Y')
                                 ELSE WRITELN(out,'N');
WRITE(out,spaces,'   VOICENOTIF  : ');
IF UAND(bit_value,mrif$m_voicenotif)>0 THEN WRITELN(out,'Y')
                                       ELSE WRITELN(out,'N');
WRITE(out,spaces,'   TEXT        : ');
IF UAND(bit_value,mrif$m_text)>0 THEN WRITELN(out,'Y')
                                 ELSE WRITELN(out,'N');
WRITE(out,spaces,'   ODIF        : ');
IF UAND(bit_value,mrif$m_odif)>0 THEN WRITELN(out,'Y')
                                 ELSE WRITELN(out,'N');
WRITE(out,spaces,'   TIF1        : ');
IF UAND(bit_value,mrif$m_tif1)>0 THEN WRITELN(out,'Y')
                                 ELSE WRITELN(out,'N');
WRITE(out,spaces,'   SFD         : ');
IF UAND(bit_value,mrif$m_sfd)>0 THEN WRITELN(out,'Y')
                                ELSE WRITELN(out,'N');
WRITE(out,spaces,'   VOICE       : ');
IF UAND(bit_value,mrif$m_voice)>0 THEN WRITELN(out,'Y')
                                  ELSE WRITELN(out,'N');
WRITE(out,spaces,'   VIDEOTEX    : ');
IF UAND(bit_value,mrif$m_videotex)>0 THEN WRITELN(out,'Y')
                                     ELSE WRITELN(out,'N');
WRITE(out,spaces,'   TELETEX     : ');
IF UAND(bit_value,mrif$m_teletex)>0 THEN WRITELN(out,'Y')
                                    ELSE WRITELN(out,'N');
WRITE(out,spaces,'   TIF0        : ');
IF UAND(bit_value,mrif$m_tif0)>0 THEN WRITELN(out,'Y')
                                 ELSE WRITELN(out,'N');
WRITE(out,spaces,'   G3FAX       : ');
IF UAND(bit_value,mrif$m_g3fax)>0 THEN WRITELN(out,'Y')
                                  ELSE WRITELN(out,'N');
WRITE(out,spaces,'   IA5TEXT     : ');
IF UAND(bit_value,mrif$m_ia5text)>0 THEN WRITELN(out,'Y')
                                    ELSE WRITELN(out,'N');
WRITE(out,spaces,'   TELEX       : ');
IF UAND(bit_value,mrif$m_telex)>0 THEN WRITELN(out,'Y')
                                  ELSE WRITELN(out,'N');
WRITE(out,spaces,'   UNDEFINED   : ');
IF UAND(bit_value,mrif$m_undefined)>0 THEN WRITELN(out,'Y')
                                      ELSE WRITELN(out,'N');
END;

OTHERWISE
              { Bitstring is not a recognized flags longword }
     WRITELN(out,spaces,' UNKNOWN bitstring found');
END;
END;

PROCEDURE nbsdis(indent:INTEGER);

{
```

```
      Recursive procedure to extract the NBS data from the message and to
      output details about it.

      INPUT : indent - number of spaces used is indent*3
      Implicit Input : msgctx,out
      Implicit Output : msgctx,out
    }

    VAR
       spaces:str_type;      { declared local so there for nested nbsdis calls }
       space_cnt:INTEGER;    { used in FOR loop in comparison with indent }

    BEGIN

       spaces:='';
       FOR space_cnt:=1 TO indent DO
            spaces:=spaces+'   ';

       WHILE ODD(mrif$start_nbs_get(msgctx,nbstype,nbsqual)) DO
         BEGIN
         WRITELN(out,spaces,'NBS            : %X',HEX(nbstype,8),' %X',HEX(nbsqual,8)
    );
         IF mrif$isprim(nbstype)
         THEN
               WHILE ODD(mrif$get_nbs_record(msgctx,string_value))
               DO WRITELN(out,spaces,' DATA         : ',string_value)
         ELSE nbsdis(indent+1);
         mrif$end_nbs_get(msgctx)
         END;
    END;

    PROCEDURE connect_mr;

    {

    }

    CONST

        node = 'VENUS';

    VAR
        ok : UNSIGNED;

    BEGIN

    writeln('Connecting to node ',node);
    ok:=mrif$connect(lnkctx,node);
    if not odd(ok)
    then BEGIN

               writeln('Unable to connect to M.R. on node ',node);
               check(ok);

             END;

    connected:=TRUE;
    END;

    PROCEDURE identify_mr;

    {
    }

    VAR
       ok        :UNSIGNED;
       pwd       :str_type;

    BEGIN

    REPEAT
       WRITE('Mailbox [default mailbox]              > ');
       READLN(mbx);
       WRITE('Password [optional]                    > ');
       READLN(pwd);

       IF LENGTH(pwd)<>0
       THEN ok:=mrif$identify(lnkctx,mbx,pwd)
       ELSE ok:=mrif$identify(lnkctx,mbx);

       IF NOT ODD(ok)
       THEN WRITELN('Unable to identify to mailbox ',mbx);
    UNTIL ODD(ok);
    END;
```

```
PROCEDURE disconnect_mr;

{
}

VAR ok:UNSIGNED;

BEGIN
ok:=mrif$disconnect(lnkctx);
connected:=TRUE;
check(ok);
END;

PROCEDURE fetch_mr(VAR envfile,cntfile:str_type);
{
}

VAR ok    :UNSIGNED;
    mid   :str_type;

BEGIN
ok:=mrif$fetch(lnkctx,cntfile,envfile,0);
check(ok);
END;


PROCEDURE disass(indent:INTEGER);

{
  Recursive procedure to find the next element in the message and to output
  details about it.

  INPUT : indent - number of spaces is ident*3
  Implicit Input : msgctx,out
  Implicit Output : msgctx,out
}

VAR
   spaces : str_type;       { declared local so there for all disass calls }
   space_cnt : INTEGER;     { used in FOR loop in comparison with ident }

BEGIN

   class:=mrif$k_any;
   item:=mrif$k_any;

spaces:='';
FOR space_cnt:=1 TO indent DO
    spaces:=spaces+'   ';
WHILE ODD(mrif$find(msgctx,class,item,mrif$k_range_cont,find_length)) DO
BEGIN
     item_to_ascii(item,ascii_item);

CASE INT(class) OF

mrif$k_prim_integer:
BEGIN
{
 This routine assumes an integer length of 4 bytes and uses 16 bytes only
 if the integer overflows. It is possible to use FIND_LENGTH to determine
 the actual length of the integer and use this value instead of using a
 default.
}

WRITE(out,spaces,ascii_item,': %X');
int_length:=4;
ok:=mrif$get_integer(msgctx,item,integer_value,int_length);
IF ok=mrif$_overflow
THEN
   BEGIN
   int_length:=16;
   ok:=mrif$get_integer(msgctx,item,integer_value,int_length);
   END;
check(ok);
FOR length_cnt:=int_length DOWNTO 1 DO
   WRITE (out,HEX(integer_value[(length_cnt)::small_byte],2,2));
IF ascii_item='REASON'     THEN reason_explication(integer_value[1]);
IF ascii_item='DIAGNOSTIC' THEN diagnostic_explication(integer_value[1]);
WRITELN(out,'');
END;

mrif$k_prim_string:
BEGIN
```

```
      {
        Code to handle truncated strings.
      }
      ok:=mrif$get_string(msgctx,item,string_value);
      WRITELN(out,spaces,ascii_item,': ',string_value);
      WHILE ok=mrif$_truncate
      DO
      BEGIN
         ok:=mrif$get_string(msgctx,item,string_value);
         WRITELN(out,spaces,ascii_item,': ',string_value);
      END;
      check(ok);
      END;

      mrif$k_prim_date:

      BEGIN
      {
        Convert a date to a readable format.
      }
      mrif$get_date(msgctx,item,date_value);
      $asctim(date_length,ascii_date,date_value);
      WRITELN(out,spaces,ascii_item,': ',ascii_date);
      END;

      mrif$k_prim_flags:
      BEGIN
      mrif$get_flags(msgctx,item,bit_value);
      WRITELN(out,spaces,ascii_item,': %X',HEX(bit_value,8,8));
      {
        Call procedure to look at the bits in the flags longword.
      }
      out_flags(spaces,item,bit_value);
      END;

      mrif$k_name,mrif$k_list,mrif$k_report :
      BEGIN
      WRITELN(out,spaces,ascii_item);
      mrif$start_get(msgctx,class,item);
      disass(indent+1);
      mrif$end_get(msgctx);
      END;

      mrif$k_bodypart:
      BEGIN
      WRITELN(out,spaces,ascii_item);
      mrif$start_get(msgctx,class,item);
      {
        Determine bodypart item and call appropriate routine.
      }
      IF item=mrif$k_text
      THEN
         WHILE ODD(mrif$get_text_record(msgctx,string_value,mrif$m_crlf))
         DO WRITELN(out,spaces,'  ',string_value)
      ELSE
         IF item=mrif$k_forwarded
            THEN disass(indent+1)
            ELSE nbsdis(indent+1);
      mrif$end_get(msgctx);
      END;

      OTHERWISE;
      END;

      class:=mrif$k_any;
      item:=mrif$k_any;

      END;
      END;

      PROCEDURE query_mr(VAR number : UNSIGNED);

      VAR
         ok       :UNSIGNED;
         mid      :str_type;

      BEGIN
      check(ok);
      END;


      {
         MAIN ROUTINE.
         =============
```

```
}

BEGIN

 {

 }

connected:=FALSE;
writeln;
connect_mr;


REPEAT
identify_mr;
ok:=mrif$query(lnkctx,number);
WRITE('Mailbox [',mbx,'] ');
IF number=0 THEN WRITELN('is EMPTY.')
ELSE
    BEGIN
    WRITE('contains ',number:3,' NEW message');
    IF number>1 THEN WRITELN('s.') ELSE WRITELN('.');
    WRITE('Do you wish to read it/them ? [Y] ');
    yn:='';
    READLN(yn);
    IF ((yn='') or (yn='y') or (yn='Y'))
    THEN
        BEGIN

           envfile := 'ENVELOPE.NBS';
           cntfile := 'CONTENT.NBS';

           mbx2:=mbx;
           FOR j:=1 TO LENGTH(mbx2) DO
           BEGIN
             IF mbx2[j]=' ' THEN mbx2[j]:='_';
           END;

           FOR i:=1 TO number DO
           BEGIN
               WRITE('Output file [',mbx2,DEC(i,2),'.MHS] (TT:) > ');
               READLN(outfile);
               IF LENGTH(outfile)=0 THEN outfile:=mbx2+DEC(i,2)+'.MHS';
               open(out,outfile);
               rewrite(out);
               fetch_mr(envfile,cntfile);
               writeln('Message nr ',i:2,' fetched successfully.');

              { This section disassembles the ENVELOPE and CONTENT parts. }

               writeln(out,'-----------------------------------------------');
               writeln(out,'MESSAGE NR ',i:2,' :');
               writeln(out);
               init(envfile);
               disass(1);
               finish;
               init(cntfile);
               disass(1);
               finish;
               writeln(out);
               writeln(out,'-----------------------------------------------');
               close(out);

           END;
        END;
     END;
yn:='';
write('Do you want to fetch to another mailbox ? [N] ');
readln(yn);
UNTIL((length(yn)=0) or (yn='N') or (yn='n'));

disconnect_mr;
writeln;
END.
```

ANNEXE 8.5 : LISTING DES PROGRAMMES

8.5.2 LOGICIEL DE TESTS MRXTESTER

Le logiciel MRXTESTER a été développé par les auteurs sur le VAX/VMS de l'Inter-university Institute for High Energies à Bruxelles. Il comprend une procédure en langage DCL (MRXTESTER.COM) qui joue le rôle de coordinateur entre les différents modules. Le module MRXTESTER$BUILD.PAS réalise la composition de messages de tests, le module MRXTESTER$POST.PAS réalise la soumission de messages au MTA, le module MRXTESTER$ORNAMES.PAS réalise la génération automatique de tests d'O/R names, le module MRXTESTER$ORPOST.PAS soumet les tests d'O/R names au MTA, le module MRXTESTER$ASSEMBLE.PAS traduit un message MDL en message NBS et le module MRXTESTER$DISASSEMBLE.PAS traduit un message NBS en message MDL. Le module MRXTESTER$READ.PAS est le même que le module UA$READ.PAS.

Les différents modules sont écrits en PASCAL VMS.

MRX.COM

```
$!
$!   MRXTESTER - MRX Conformance Testing Procedure.
$!
$! AUTHORS:
$!      Marc ELOY & Daniel REUVIAUX.
$!
$! CREATION DATE: January 1988
$!
$! MODIFY BY :
$! IDENT    INITLS      DATE      REASON
$!         Drx & Me  xx-Jan-88    original
$!
$ type sys$input
$ deck

       MRXTESTER - MRX Conformance Testing Procedure V.1.0

       (c) Drx & Me jan'88.


$ eod
$enter_command:
$ inquire/nopunct command "MRXTESTER> "
$ if command .eqs. "CREATE" then goto create
$ if command .eqs. "BUILD" then goto build
$ if command .eqs. "POST"  then goto post
$ if command .eqs. "READ"  then goto read
$ if command .eqs. "HELP"  then goto help
$ if command .eqs. "EXIT"  then goto bye
$ goto enter_command
$!
$create:
$ set noon
$ run sys$system:mrman
ADD MRXTESTER/OWNER=SYSTEM/SYSTEM/PASSWORD="MRXTESTER"
$ set on
$ goto enter_command
$build:
$ present =  f$search("mrxtester_build.tmp")
$ if f$length(present) .gt. 0 then goto skip_build_tmp
$ open/write/error=noopen cmdfile mrxtester_build.tmp
$ write cmdfile "$ deassign sys$input"
$ write cmdfile "$ run mrxtester$build"
$ close cmdfile
$skip_build_tmp:
$ @mrxtester_build.tmp
$ goto enter_command
$!
$post:
$ present =  f$search("mrxtester_post.tmp")
$ if f$length(present) .gt. 0 then goto skip_post_tmp
$ open/write/error=noopen cmdfile mrxtester_post.tmp
$ write cmdfile "$ deassign sys$input"
$ write cmdfile "$ run mrxtester$post"
$ close cmdfile
$skip_post_tmp:
$ @mrxtester_post.tmp
$ goto enter_command
$!
$read:
$ present =  f$search("mrxtester_read.tmp")
$ if f$length(present) .gt. 0 then goto skip_read_tmp
$ open/write/error=noopen cmdfile mrxtester_read.tmp
$ write cmdfile "$ deassign sys$input"
$ write cmdfile "$ run mrxtester$read"
$ close cmdfile
$skip_read_tmp:
$ @mrxtester_read.tmp
$ goto enter_command
$!
$help:
$ type sys$input
$ deck

       MRXTESTER Utilities

 BUILD  : Construct a X.400 message following a template.
 CREATE : Create the MRXTESTER mailbox and subscriber.
 EXIT   : Return to VMS.
 HELP   : Display this text.
 POST   : Post a X.400 message constructed by BUILD.
```

```
$ eod
$enter_help_command:
$ inquire/nopunct help_command "Help> "
$ if help_command .eqs. "BUILD" then goto help_build
$ if help_command .eqs. "POST" then goto help_post
$ if help_command .eqs. "CREATE" then goto help_create
$ if help_command .eqs. "HELP" then goto help
$ if help_command .eqs. "EXIT" then goto help
$ goto enter_command
$
$help_build:
$ type sys$input/page
$ deck

  BUILD

        This facility allows you to create a X.400 message file following
        a template. This message will be stored in a file in order to be
        posted by the POST utility.

        It allows you to build an ENVELOPE only, a CONTENT only,
        an ENVELOPE and a CONTENT or a COMPLETE_MESSAGE.
        The template consists of a set of questions.
        A typical entry for an item is :

          [ <subject> (str) ]

        This ask you if you want to include a <subject> in your message.
        This element is optional : <>.
        If you want a subject in your message, enter the string, followed
        by <Return>.
        If you don't want a subject, simply hit <Return>.

        Data types are : (str) string,
                         (int) integer,
                         (flg) flags,
                         (gr)  group of item(s),
                         (date) date.

        Items can be : Mandatory <item>,
                       Optionnal [ <item> ],
                       Repetitive { <item> } or { [ <item> ] }.

        When a question mark appears in an item : [ <reportflg> (flg) ] ?
        type <Y> if you want to add this item, otherwise <N>.

                         ********* WARNING ! *********

        This utility is test utility.
        Users should know how to write a X.400 message before using it.
        This utility allows the user to enter wrong messages in order to
        test the MTA's reaction to them.

              DO NOT USE THIS UTILITY AS A COMMON USER AGENT

        unless you are sure of what you are doing.

$ eod
$ goto enter_help_command
$help_post:
$ type sys$input/page
$ deck


  POST
        This utility allows you to post a message which has been
        assembled before with the BUILD utility.

        You can post a COMPLETE_MESSAGE (the Message Router will
        construct the envelope from the content it receives);
                    an ENVELOPE and a CONTENT which have been assembled
        separetly or together.

$eod
$ goto enter_help_command
$help_create:
$ type sys$input/page
$ deck

    CREATE
        This utility allows you to create the MRXTESTER mailbox and
        subscriber.

        It runs first the MRMAN utility to create the mailbox, and then
```

```
            calls the MRXMAN utility to create the subscriber which will look
            like this :
                        surname = MRXTESTER_SURNAME,
                        given_name = MRXTESTER_GIVE_NAME,
                        initials = TEST,
                        orgunit = MRXTESTER_ORGUNIT.

                          ***** WARNING *****

            You need some privileges to run MRXMAN.

$ eod
$ goto enter_help_command
$bye:
$ delete/noconfirm mrxtester_*.tmp;*
$ write sys$output "End of MRXTESTER"
$ write sys$output " "
```

MRXTESTER$BUILD.PAS

```
[INHERIT('mrxtester$defs.pen','sys$library:MRIFDEFS.PEN','sys$library:STARLET.PE
N')]

PROGRAM mrxtester$build(input,output);

{ ------------------------------------------------------------------------- }

TYPE
   str_type = VARYING[133] OF CHAR;
   $QUAD    = [QUAD,UNSAFE] RECORD
                            LO:UNSIGNED;
                            L1:INTEGER;
                            END;
   $UBYTE   = [BYTE] 0..255;
   $UWORD   = [WORD] 0..65535;
   comp     = (
              TB1,TB11,TB2X,TB3,TB4X,TB5,TB6,
              IP1,IP1X,IP2,IP2X,IP21,IP21X,IP3,IP3X,IP4,IP4X,IP5,IP5X,IP6,IP6X,
              MS1,MS2X,
              SJ1,SJ11,SJ2X,
              RI1,RI2X,
              ps1,ps2,ps3,ps4,ps5,ps6,ps7,ps8,ps9,psA,
              T610,fin
              );

VAR
   yn,                                { user's answer to questions   }
   string,                            { field used to collect strings }
   envfile,                           {  file containing the ENVELOPE }
   cntfile,                           {  file containing the CONTENT  }
   outfile              : str_type;   {  output file main name        }
   number               : INTEGER;
   ok,
   msgctx,                  { the message context used when assembling }
   lnkctx,       { the link context used when communicating with M.R. }
   u1,              { u1 an unsigned where flags will be constructed }
   assemble_flag        : UNSIGNED;
   complete,
   envelope,
   content,
   both,                     { signal the message part being assembled }
   assemble,                       { signal if assembly is in progress }
   connected            : BOOLEAN;
   time                 : $QUAD;
   buf                  : ARRAY [1..16] OF $UBYTE;
   c,nom                : comp;

{ ------------------------------------------------------------------------

   Declaration for Message Router Programmers Kit Routines.
   These are non-standard PASCAL declarations that use extensions to
   Pascal and are used in a similar way to the Run Time Library. The
   Interface Routines follow the VMS Procedure Calling Standard.
}

[EXTERNAL]
   FUNCTION mrif$start_assemble(VAR msgctxid : UNSIGNED;
                                    flag : UNSIGNED := %IMMED 0)
                               :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$select_assemble(msgctxid:UNSIGNED;
                                 msgtype:UNSIGNED:=%IMMED 0)
                                 :UNSIGNED;EXTERNAL;
[EXTERNAL]
   FUNCTION mrif$end_assemble(msgctxid:UNSIGNED;
                              %DESCR contents:VARYING[a1] OF CHAR:=%IMMED 0;
                              %DESCR envelope:VARYING[b1] OF CHAR:=%IMMED 0)
                                :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$start_put(msgctxid:UNSIGNED;
                           class:UNSIGNED;
                           item:UNSIGNED)
                                :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$end_put(msgctxid:UNSIGNED):UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_string(msgctxid:UNSIGNED;
```

```
                            item:UNSIGNED;
                            %DESCR string:VARYING[a1] OF CHAR)
                            :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_integer(msgctxid:UNSIGNED;
                             item:UNSIGNED;
                             %REF number:ARRAY[u1..u2:INTEGER] OF $UBYTE;
                             size:$UWORD:=%IMMED 0)
                             :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_flags(msgctxid:UNSIGNED;
                           item:UNSIGNED;
                           flags:UNSIGNED)
                          :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_date(msgctxid : UNSIGNED;
                          item:UNSIGNED;
                          date:$QUAD)
                          :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_text_record(msgctxid:UNSIGNED;
                            %DESCR string:VARYING[a1] OF CHAR;
                            flags:UNSIGNED:=%IMMED 0)
                           :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_text_file(msgctxid:UNSIGNED;
                               %DESCR string:VARYING[a1] OF CHAR)
                              :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$start_nbs_put(msgctxid:UNSIGNED;
                               nbstype:UNSIGNED;
                               nbsqual:UNSIGNED) :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$end_nbs_put(msgctxid:UNSIGNED):UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_nbs_record(msgctxid:UNSIGNED;
                                %DESCR string:VARYING[a1] OF CHAR)
                               :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$connect(VAR lnkctx:UNSIGNED;
                         %DESCR node:VARYING[a1] OF CHAR:=%IMMED 0)
                         :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$disconnect(lnkctx:UNSIGNED):UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$identify(lnkctx:UNSIGNED;
                          %DESCR mailbox:VARYING[a1] OF CHAR;
                          %DESCR password:VARYING[b1] OF CHAR:=%IMMED 0)
                          :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$post(lnkctx:UNSIGNED;
                      %DESCR content:VARYING[a1] OF CHAR;
                      %DESCR envelope:VARYING[b1] OF CHAR:=%IMMED 0;
                      flags:UNSIGNED:=%IMMED 0;
                      %DESCR msgid:VARYING[c1] OF CHAR:=%IMMED 0)
                       :UNSIGNED;EXTERNAL;

{ Declare Run Time Library that will be needed. }

[EXTERNAL]FUNCTION LIB$STOP(%IMMED sts:UNSIGNED):UNSIGNED;EXTERNAL;

[EXTERNAL]FUNCTION LIB$SIGNAL(%IMMED sts:UNSIGNED):UNSIGNED;EXTERNAL;

{  Declare external procedure which contains the help on-line system
   for this program.
   This procedure is in the MRXTESTER$HELP.PAS file.
   This file must first be compiled by $PASCAL MRXTESTER,
   then linked with the object of this program by the command :
   $LINK MRXTESTER$BUILD,MRXTESTER$HELP,OPT/OPTIONS
   wher OPT is the file containing the declaration of the MRIF Routines.

}
```

```
[EXTERNAL]PROCEDURE mrxtester$get_help( item : INTEGER);EXTERNAL;


PROCEDURE check(status_code:UNSIGNED);

{ Procedure to determine whether the status returned by another routine
  was a VMS success or error status. If the routine returned an error,
  the program terminates and control is returned to the MRXTESTER.COM
  DCL procedure.

  Input : status_code - a 32 bit VMS status returned by another routine
  Implicit input msgctx or lnkctx
}

BEGIN

IF NOT ODD(status_code)          { Bottom bit indicates success }
THEN
    BEGIN

    IF assemble THEN mrif$end_assemble(msgctx);
    IF connected THEN mrif$disconnect(lnkctx);
    writeln('MRXTESTER-F-FTLERR, ***** FATAL ERROR *****');
    writeln('The program will terminates due to the reason bellow : ');
    writeln;
    LIB$SIGNAL(status_code);
    $exit;

    END;
END;


PROCEDURE connect_mr;

{ Procedure to call the CONNECT routine to connect to Message Router on
  the local node. If the connection is impossible (system too loaded) the
  program terminates.

  Implicit Input : lnkctx
}

CONST

    node = 'VENUS';

BEGIN

    writeln('Connecting to node ',node);
    ok:=mrif$connect(lnkctx,node);
    if not odd(ok)
    then BEGIN

            writeln('Unable to connect to M.R. on local node.');
            check(ok);

         END;

    connected:=TRUE;

END;


PROCEDURE identify_mr;

{ Procedure to identify to a Message Router mailbox.
  The procedure tries first to identify to the mailbox MRXTESTER which is
  the default mailbox for the MRXTESTER utility.
  If the identification fails, the user is asked to enter an existing
  mailbox.

  Implicit Input : lnkctx
}

CONST

    def_mbx = 'MRXTESTER';
    def_pwd = 'MRXTESTER';

VAR
```

```
    mbx,
    pwd     :str_type;

BEGIN

    mbx:=def_mbx;
    pwd:=def_pwd;
    ok:=mrif$identify(lnkctx,mbx,pwd);
    if not odd(ok)
    then BEGIN

        writeln('Unable to identify to mailbox ',def_mbx);

        REPEAT

            write('Mailbox   : ');
            readln(mbx);
            write('Password  : ');
            readln(pwd);
            if length(pwd)<>0
            then ok:=mrif$identify(lnkctx,mbx,pwd)
            else ok:=mrif$identify(lnkctx,mbx);
            if not odd(ok)
            then writeln('Unable to identify to mailbox ',mbx);

        UNTIL odd(ok);

        END;
    writeln('Identified to mailbox ',mbx);

END;


PROCEDURE post_mr;

{ Procedure to post a message into Message Router. The message_id that
  Message Router adds to the message is returned and displayed.

  Implicit Input  : lnkctx,envfile,cntfile
  Implicit Output : msg_id.

}

VAR
    msg_id : str_type; { The message identifier given to the posted message
                         by the Message Router.
                       }

BEGIN

msg_id:='';
ok:=mrif$post(lnkctx,cntfile,envfile,0,msg_id);
check(ok);
write('%MRXTESTER-I-MSGPSTDS, ');
writeln('Message posted successfully with message_id : ',msg_id,'.');
writeln;

END;


PROCEDURE disconnect_mr;

{ Procedure to disconnect from Message Router.

  Implicit Input : lnkctx

}

BEGIN

ok:=mrif$disconnect(lnkctx);
connected:=FALSE;
check(ok);

END;


PROCEDURE post_message;

{ Procedure to prompt the user if he wants to post the message he has just
  assembled into Message Router.
```

```
  If post is required then the procedure establishes a connection with the
  Message Router on the local node, identifies to a Message Router mailbox,
  posts the message into the Message Router and disconnects from Message
  Router.
}

BEGIN

    yn:='';
    write('Do you want to post this message NOW [N] ? ');
    readln(yn);
    if ((yn='y') or (yn='Y'))
    then BEGIN
         connect_mr;
         identify_mr;
         post_mr;
         disconnect_mr;
         END;

END;


PROCEDURE prompt( item : UNSIGNED);

{ Procedure which display on the terminal screen the name of the message
  item which will have to be filled by the user.
  Items can be mandatory : <item>,
               optional  : [ <item> ],
               repetitive: { <item>.}
{
  Input : an integer representing the code of a message item.
          The codes used are those described in the MRIFDEFS.PAS file,
          plus our own codes described in the MRXTESTER$DEFS.PAS file.

}

BEGIN
     CASE INT(item) OF

mrif$k_amdname       :write('[ <amdname> ]        (str) : ');
mrif$k_app_message_id:write('[ <app_message_id> ] (str) : ');
mrif$k_arrivaldate   :;
mrif$k_author        :write('{ <author> }         (Y/N) : ');
mrif$k_autoforward   :write('[ <autoforward> ]    (int) : ');
mrif$k_bcc           :write('{ <bcc> }            (Y/N) ? ');
mrif$k_cc            :write('{ <cc> }             (Y/N) ? ');
mrif$k_contenttypes  :write('[ <contenttypes> ]   (int) : ');
mrif$k_country       :write('[ <country> ]        (str) : ');
mrif$k_date          :write('[ <create_date> ]     (Y/N) ? ');
mrif$k_ddif          :write('<ddif>               (Y/N) ? ');
mrif$k_deferred      :write('[ <deferred> ]       (Y/N) ? ');
mrif$k_deliverdate   :;
mrif$k_diagnostic    :;
mrif$k_encodedtypes  :write('[ <encodedtypes> ]   (Y/N) ? ');
mrif$k_enddate       :write('[ <enddate> ]        (Y/N) ? ');
mrif$k_explicconv    :write('[ <explicconv> ]     (int) : ');
mrif$k_extensionid   :write('[ <extensionid> ]    (int) : ');
mrif$k_forwarded     :write('<forwarded>          (Y/N) ? ');
mrif$k_freeform      :write('[ <freeform> ]       (str) : ');
mrif$k_from          :write('[ <from> ]           (Y/N) ? ');
mrif$k_g3fax         :write('<g3fax>              (Y/N) ? ');
mrif$k_generation    :write('[ <generation> ]     (str) : ');
mrif$k_generated     :;
mrif$k_givenname     :write('[ <givenname> ]      (str) : ');
mrif$k_hopcount      :write('[ <hopcount> ]       (int) : ');
mrif$k_ia5text       :write('<ia5text>            (Y/N) ? ');
mrif$k_info          :;
mrif$k_initials      :write('[ <initials> ]       (str) : ');
mrif$k_inreplyto     :write('[ <inreplyto> ]      (str) : ');
mrif$k_listname      :write('[ <listname> ]       (str) : ');
mrif$k_location      :write('[ <location> ]       (str) : ');
mrif$k_message_id    :write('<message_id>         (str) : ');
mrif$k_msgclass      :write('[ <msgclass> ]       (str) : ');
mrif$k_obsoletes     :write('{ <obsoletes> }      (str) : ');
mrif$k_odif          :write('<odif>               (Y/N) ? ');
mrif$k_orgname       :write('[ <orgname> ]        (str) : ');
mrif$k_orgunit       :write('[ <orgunit> ]        (str) : ');
mrif$k_pdate         :write('[ <pdate> ]          (Y/N) ? ');
mrif$k_perlistflg    :write('<perlistflg>         (flg) ? ');
mrif$k_permsgflg     :write('[ <permsgflg> ]      (flg) ? ');
mrif$k_perrecflg     :write('<perrecflg>          (flg) ? ');
mrif$k_prdname       :write('[ <prdname> ]        (str) : ');
mrif$k_precedence    :write('[ <precedence> ]     (int) : ');
```

```
mrif$k_reason          :;
mrif$k_references      :write('{ <references> }      (str) : ');
mrif$k_replyby         :write('[ <replyby> ]         (Y/N) ? ');
mrif$k_replyreq        :write('[ <replyreq> ]        (int) : ');
mrif$k_replytousers    :write('{ <replytousers> }    (Y/N) ? ');
mrif$k_reportflg       :write('[ <reportflg> ]       (flg) ? ');
mrif$k_rmsfile         :write('<rmsfile>             (Y/N) ? ');
mrif$k_route           :write('{ <route> }           (str) : ');
mrif$k_sender          :write('[ <sender> ]          (Y/N) ? ');
mrif$k_sensitivity     :write('[ <sensitivity> ]     (int) : ');
mrif$k_sfd             :write('<sfd>                 (Y/N) ? ');
mrif$k_subject         :write('[ <subject> ]         (str) : ');
mrif$k_surname         :write('<surname>             (str) : ');
mrif$k_telephone       :write('[ <telephone> ]       (str) : ');
mrif$k_teletex         :write('<teletex>             (Y/N) ? ');
mrif$k_telex           :write('<telex>               (Y/N) ? ');
mrif$k_terminalid      :write('[ <terminalid> ]      (str) : ');
mrif$k_text            :write('<text>                (Y/N) ? ');
mrif$k_tif0            :write('<tif0>                (Y/N) ? ');
mrif$k_tif1            :write('<tif1>                (Y/N) ? ');
mrif$k_to              :write('{ <to> }              (Y/N) ? ');
mrif$k_uacontid        :write('[ <uacontid> ]        (str) : ');
mrif$k_undefined       :write('<undefined>           (Y/N) ? ');
mrif$k_uniqueuaid      :write('<uniqueuaid>          (str) : ');
mrif$k_userid          :write('<mr_address>          (mbx) : ');
mrif$k_vendor1:;
mrif$k_vendor2:;
mrif$k_vendor3:;
mrif$k_vendor4:;
mrif$k_vendor5:;
mrif$k_vendor6:;
mrif$k_vendor7:;
mrif$k_vendor8:;
mrif$k_vendor9:;
mrif$k_vendor10:;
mrif$k_vendor11:;
mrif$k_vendor12:;
mrif$k_vendor13:;
mrif$k_vendor14:;
mrif$k_vendor15:;
mrif$k_vendor16:;
mrif$k_videotex        :write('<videotex>            (Y/N) ? ');
mrif$k_voice           :write('<voice>               (Y/N) ? ');
mrif$k_voicenotif      :write('<voicenotif>          (Y/N) ? ');
mrif$k_wpsplus         :write('<wpsplus>             (Y/N) ? ');
mrif$k_x121address     :write('<x121address>         (str) ? ');

(** Flags informations **)

mrif$k_contretreq      :write('<contretreq>          (Y/N) ? ');
mrif$k_altrecip        :write('<altrecip>            (Y/N) ? ');
mrif$k_convprohib      :write('<convprohib>          (Y/N) ? ');
mrif$k_discloserec     :write('<discloserec>         (Y/N) ? ');
mrif$k_returnmsg       :write('<returnmsg>           (Y/N) ? ');
mrif$k_nonrecnotif     :write('<nonrecnotif>         (Y/N) ? ');
mrif$k_recnotif        :write('<recnotif>            (Y/N) ? ');
mrif$k_ua_basic        :write('<ua_basic>            (Y/N) ? ');
mrif$k_ua_confirmed    :write('<ua_confirmed>        (Y/N) ? ');
mrif$k_mr_basic        :write('<mr_basic>            (Y/N) ? ');
mrif$k_mr_confirmed    :write('<mr_confirmed>        (Y/N) ? ');
mrif$k_action          :write('<action>              (Y/N) ? ');
mrif$k_expandedlist    :write('<expandedlist>        (Y/N) ? ');

(**  MRXTESTER Constants  **)

mrxtester$k_name_list:write('[N]ame or [L]ist      (N/L) : ');
mrxtester$k_recipient:write('{ <recipient> }       (Y/N) ? ');
mrxtester$k_domain_1 :write('[ <domainspecific> ] (Y/N) ? ');
mrxtester$k_domain_2 :write('{ <domainspecific> } (Y/N) ? ');
mrxtester$k_orname   :write('<X400_orname>     (1,2,3,4) : ');
mrxtester$k_header   :write('[ <header> ]          (Y/N) ? ');
mrxtester$k_body     :write('{ <body> }            (Y/N) ? ');
mrxtester$k_buimsgtype:write('Message Type      : ');

OTHERWISE

writeln('%MRXTESTER-W-UNKNWNLMT, Unknown element : ',item);

END;

END;


FUNCTION component(c :comp) : str_type;
```

```
{
  Function which implements the 'SPAG Component Library' as defined in the
  'SPAG Interworking Guide' (pages 6.1-4)

  Input  : The name of the component

           Typed Body Components

                TB1     L_IA5text_1     Small IA5 string
                TB11                    Full IA5 character set
                TB2X                    1500 characters IA5 string
                TB3                     ISO6937 body
                TB4X                    1500 characters ISO6937
                TB5                     Forwarded bodypart (Test1 V->R)
                TB6                     Forwarded bodypart (Test2 R->V)

           IP-Message-ID Components

                IP1     L_IPMessa_ID_0  small prt
                IP1X                    max 64 prt
                IP2                     small prt
                IP2X                    max 64 prt
                IP21                    small prt
                IP21X                   max 64 prt
                IP3                     small prt
                IP3X                    max 64 prt
                IP4                     small prt
                IP4X                    max 64 prt
                IP5                     small prt
                IP5X                    max 64 prt
                IP6                     small prt
                IP6X                    max 64 prt

           Message_ID components

                MS1                     vendor specific
                MS2X                    max 16 prt

           Subject Components

                SJ1     L_T61string_4   Small T61 string
                SJ11                    Long T61 string
                SJ2X                    Max 128 T61 string

           Replying-IP-Message-ID Components

                RI1     L_IPMessagIP_1  small prt
                RI2X                    marx 64 prt

           Printable Strings

                ps1                     length 51
                ps2                     length 23
                ps3                     length 23
                ps4                     length 23
                ps5                     length 23
                ps6                     length 32
                ps7                     length 6
                ps8                     length 16
                ps9                     length 25
                psA                     length 53

           T.61 Strings

                T610                    length 51

  Output : The corresponding string of characters
}

BEGIN

   CASE c OF

      TB1  : component:='ABCDEFGHIJLKM0123456789';
      TB11 : component:='ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnopqrstuvwxyz !"ü$%''(*+,-./:;<=>?@[\]^_6{
;
      TB2X : component:='(1500 char IA5String)';
      TB3  : component:='A!"%&()*+,-./A:;<=>?END';
      TB4X : component:='(1500 char ISO6937)';
      TB5  : component:='(Forwarded bodypart (test1 Vendor to Reference))';
      TB6  : component:='(Forwarded bodypart (test2 Reference to Vendor))';

      IP1  : component:='Test number 1.';
```

```
      IP1X : component:='Test number 1.Maximum length of 64 Printable string.....END.';
      IP2  : component:='Test number 2.';
      IP2X : component:='Test number 2.Maximum length of 64 Printable string.....END.';
      IP21 : component:='Suite of test number 2.';
      IP21X: component:='Suite of test number 2.Maximum length of 64 Printable StringEND.';
      IP3  : component:='Test number 3.';
      IP3X : component:='Test number 3.Maximum length of 64 Printable string.....END.';
      IP4  : component:='Test number 4.';
      IP4X : component:='Test number 4.Maximum length of 64 Printable string.....END.';
      IP5  : component:='Test number 5.';
      IP5X : component:='Test number 5.Maximum length of 64 Printable string.....END.';
      IP6  : component:='Test number 6.';
      IP6X : component:='Test number 6.Maximum length of 64 Printable string.....END.';

      MS1  : component:='(vendor defined)';
      MS2X : component:='ABCDEFGHIJKLMNOP';

      SJ1  : component:='Subject number 1.';
      SJ11 : component:='ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';
      SJ2X : component:='(max 128 T61 string)';

      RI1  : component:='ABCDEFGH.';
      RI2X : component:='ABCDEFGHIJKLMNOPQRSTUVWXYZ''()+,-./:=?0123456789abcdefghijklmnop';

      ps1  : component:='ABCDEFGHIJKLMNOPQRSTUVWXYZ ''()+,-./:=?0123456789abc';
      ps2  : component:='PQRSTUVWXYZ ''()+,-./:=?';
      ps3  : component:='0123456789abcdefghijklm';
      ps4  : component:='nopqrstuvwxyzABCDEFGHIJ';
      ps5  : component:='MNOPQRSTUVWXYZ ''()+,-./';
      ps6  : component:='defghijklmnopqrstuvwxyzABCDEFGHI';
      ps7  : component:='JKLMNO';
      ps8  : component:=':=?0123456789abc';
      ps9  : component:='defghijklmnopqrstuvwxyzAB';
      psA  : component:='CDEFGHIJKLMNOPQRSTUVWXYZ''()+,-./:=?0123456789abcdefgh';

      T610 : component:='(T.61 string (51 char))';

      OTHERWISE component:='';

   END;

END;


FUNCTION no_punct(s : str_type):str_type;

{ Function to skip a file extension in a file name.
  The file extension for all files assembled by this program will be .NBS
  which is the default used by the MRIF Routines.

  Input   : a string representing the name of the output file
            and containing an file extension (even empty).
  Returns : a string containing the file name in input, but without
            its extension.
}

VAR
    str : str_type;
    p   : INTEGER;

BEGIN

    str:=s;
    p:=INDEX(s,'.');
    if p<>0 then str:=substr(s,1,p-1);
    no_punct:=str;

END;


FUNCTION str_to_int(s : str_type):INTEGER;

{ Function to translate a char string to an integer.

  Input  : A char string containing a number.
  Returns: An integer number with the value containing in the string.

}

VAR  n,
     i   : INTEGER;
     c   : CHAR;
```

```
    ch  : str_type;

BEGIN

n:=0;
FOR i:=1 to length(s) DO
  BEGIN

  ch:=substr(s,i,1);
  readv(ch,c);
  n:= n * 10 + ord(c)-ord('0');

  END;
str_to_int:=n;

END;


FUNCTION get_choice(item : INTEGER):str_type;

{ Functions which prompts the user to enter a choice for a message item.
  This choice is mainly (Y/N) and ask the user if he wants to enter a
  composite message element (i.e. : names, dates, flags, ...)

  Input   : an integer containing the code of the message item to be prompted.
  Output  : a string containing the user's amswer to the prompt.

}

VAR
    ok : str_type;

BEGIN

    ok:='';
    REPEAT

       prompt(item);
       readln(ok);
       if ( ok = '?' ) then mrxtester$get_help(item);

    UNTIL ( ok <> '?');

    get_choice:=ok;

END;

PROCEDURE get_flag(flag,item : UNSIGNED; help : INTEGER);

{ Procedure to prompt the user for a specified flag in a flag element.
  If the user wants a flag, this is added by the UOR function.

  Input   : flag : an UNSIGNED where the flag is being constructed,
            item : a MRIFDEFS code for the flag (i.e. : mrif$m_returmsg)
            help : the MRXTESTER code for this flag (mrxtester$k_returmsg).

  Output  : flag modified with the MRIFDEFS code, at user's will

}

VAR ok : str_type;

BEGIN

REPEAT

   ok:='';
   prompt(help);
   readln(ok);
   if (ok = '?') then mrxtester$get_help(help)
   else if ((ok = 'Y') or (ok = 'y')) then flag:=UOR(flag,item);

UNTIL (ok <> '?');

END;

PROCEDURE get_integer(item:INTEGER);

{
  Procedure to receive a number and add it to the message.
  If the user enter <Return>, the item is not added to the message.

  Input   : the MRIFDEFS code of an integer message item.
```

```
}

VAR
    n : INTEGER;
    s : str_type;

BEGIN

REPEAT

   s:='';
   prompt(item);
   readln(s);
   if (s = '?') then mrxtester$get_help(item);

UNTIL (s <> '?');

    n:=0;
    if (length(s)<>0)
    then BEGIN

         n:=str_to_int(s);
         buf[1]:=n;
         ok:=mrif$put_integer(msgctx,item,buf);
         check(ok);

         END;
END;

FUNCTION mrx_orname(item:INTEGER;s:str_type):str_type;

{ Function to add the MRX code to the message elements which are used by
  MRX. ( country : 1=be) If the char '=' is already present, the function
  assumes that the code is correct.

  input : item, the MRXTESTER code of the message element
          s, the string containing the value of that element

  returns : a string containing the MRX code

}

BEGIN

if (index(s,'=')=0)
then BEGIN
     CASE item OF

      mrif$k_country     : s := '1='  +s;
      mrif$k_amdname     : s := '2='  +s;
      mrif$k_prdname     : s := '3='  +s;
      mrif$k_orgunit     : s := '4='  +s;
      mrif$k_orgname     : s := '5='  +s;
      mrif$k_surname     : s := '6='  +s;
      mrif$k_givenname   : s := '7='  +s;
      mrif$k_initials    : s := '8='  +s;
      mrif$k_generation  : s := '9='  +s;
      mrif$k_terminalid  : s := '10=' +s;
      mrif$k_x121address : s := '11=' +s;

     END;
     END;

  mrx_orname:=s;

END;

PROCEDURE get_string(item : INTEGER);

{ Procedure to prompt the user to add a string to the message.
  The string to be added may be a string taken from the SPAG component
  library.
  <Return> causes the string item not to be added in the message.

  Input   : the MRIFDEFS code of the string.

}

VAR head,tail : str_type;
    c         : comp;

BEGIN
```

```
REPEAT

    string:='';
    prompt(item);
    readln(string);
    if (string = '?') then mrxtester$get_help(item);

UNTIL (string <> '?');

    if (length(string)<>0) then
    BEGIN
        { if the string begin with '@' then search in SPAG Components table }
        head:=SUBSTR(string,1,1);
        tail:=SUBSTR(string,2,LENGTH(string)-1);

        if head='@' then
        BEGIN
           { readv() to assign the string 'tail' to the comp_type 'c' }
           c:=fin;
           readv(tail,c,error:=CONTINUE);
           string:=component( c );
        END;

 {      if (length(string)<>0) then
        BEGIN
           ok:=mrif$put_string(msgctx,item,mrx_orname(item,string));
           check(ok);
        END;
}
      END;
  END;


PROCEDURE get_date(item : INTEGER);

{ Procedure to prompt the user to enter a date in a Digital format.
  The date (if not empty) is then added to the message.

  Input : the MRIFDEFS code for the date.

}

VAR

   yn   : str_type;
   time : $QUAD;
   date : PACKED ARRAY [1..24] OF CHAR;

BEGIN

yn:=get_choice(item);
if ((yn='y') or (yn='Y'))
then BEGIN

        REPEAT

                prompt(item);
                write('dd-MMM-yyyy hh:mm:ss.cc  : ');
                readln(date);
                if (date = '?') then mrxtester$get_help(item)

        UNTIL (date <> '?');

        $bintim(date,time);
        ok:=mrif$put_date(msgctx,item,time);
        check(ok);

        END;
END;


PROCEDURE perrecflg;

{ Procedure to prompt the user to construct the PERRECFLG and to add this
  flag to the message.
  The permitted combinations are :<none>, <action>, <ua_basic>, <ua_confirmed>,
                                  <action>&<ua_basic>, <action>&<ua_confirmed>.

  There is no check on the choosen combination.
}

VAR     yn : str_type;
        u1 : UNSIGNED;
```

```
BEGIN

yn:=get_choice(mrif$k_perrecflg);
if ((yn='y') or (yn='Y'))
then BEGIN

        u1:=0;
        get_flag(u1,mrif$m_action,mrif$k_action);
        get_flag(u1,mrif$m_ua_basic,mrif$k_ua_basic);
        get_flag(u1,mrif$m_ua_confirmed,mrif$k_ua_confirmed);
        get_flag(u1,mrif$m_mr_basic,mrif$k_mr_basic);
        get_flag(u1,mrif$m_mr_confirmed,mrif$k_mr_confirmed);
        ok:=mrif$put_flags(msgctx,mrif$k_perrecflg,u1);
        check(ok);
        writeln;

        END;
END;


PROCEDURE permsgflg;

{ Procedure to prompt the user to construct the PERMSGFLG and add it to the
  message.
  The Message Router considers <discloserec>&<convprohib> as always set,
                    ignores <altrecip>,
                    treats <contretreq>.
}

VAR     yn : str_type;
        u1 : UNSIGNED;

BEGIN

yn:=get_choice(mrif$k_permsgflg);
if ((yn='y') or (yn='Y'))
then BEGIN

        u1:=0;
        get_flag(u1,mrif$m_contretreq,mrif$k_contretreq);
        get_flag(u1,mrif$m_altrecip,mrif$k_altrecip);
        get_flag(u1,mrif$m_convprohib,mrif$k_convprohib);
        get_flag(u1,mrif$m_discloserec,mrif$k_discloserec);
        ok:=mrif$put_flags(msgctx,mrif$k_permsgflg,u1);
        check(ok);
        writeln;

        END;

END;


PROCEDURE perlistflg;

{ Procedure to prompt the user for the PERLISTFLG. This item is mandatory
  if there's at least one list in the message.

  Implicit Input : msgctx

}

VAR     yn : str_type;
        u1 : UNSIGNED;

BEGIN

u1:=0;

yn:=get_choice(mrif$k_perlistflg);
if ((yn='y') or (yn='Y'))
then BEGIN

        get_flag(u1,mrif$m_expandedlist,mrif$k_expandedlist);
        u1:=uor(u1,mrif$m_expandedlist);
        ok:=mrif$put_flags(msgctx,mrif$k_perlistflg,u1);
        check(ok);

        END;

END;
```

```
PROCEDURE reportflg;

{ Procedure to prompt the user to construct the REPORTFLG and add it to the
  message.

  The permitted combinations are : <none>, <recnotif>, <nonrecnotif>,
                                   <returnmsg>, <recnotif>&<returnmsg>,
                                                <nonrecnotif>&<returnmsg>

  Implicit Input : msgctx
}

VAR     yn : str_type;
        u1 : UNSIGNED;

BEGIN

yn:=get_choice(mrif$k_reportflg);
if ((yn='y') or (yn='Y'))
then BEGIN

        u1:=0;
        get_flag(u1,mrif$m_recnotif,mrif$k_recnotif);
        get_flag(u1,mrif$m_nonrecnotif,mrif$k_nonrecnotif);
        get_flag(u1,mrif$m_returnmsg,mrif$k_returnmsg);
        ok:=mrif$put_flags(msgctx,mrif$k_reportflg,u1);
        check(ok);
        writeln;

        END;

END;


PROCEDURE encodedtypes;

{ Procedure to prompt the user to construct the ENCODEDTYPES flag and add
  it to the message.

  Implicit Input : msgctx

}

VAR     yn : str_type;
        u1 : UNSIGNED;

BEGIN

yn:=get_choice(mrif$k_encodedtypes);
if ((yn='y') or (yn='Y'))
then BEGIN

        u1:=0;
        writeln('WARNING : This program adds only TEXT bodyparts.');
        get_flag(u1,mrif$m_text,mrif$k_text);
        get_flag(u1,mrif$m_voice,mrif$k_voice);
        get_flag(u1,mrif$m_videotex,mrif$k_videotex);
        get_flag(u1,mrif$m_teletex,mrif$k_teletex);
        get_flag(u1,mrif$m_tif0,mrif$k_tif0);
        get_flag(u1,mrif$m_tif1,mrif$k_tif1);
        get_flag(u1,mrif$m_g3fax,mrif$k_g3fax);
        get_flag(u1,mrif$m_ia5text,mrif$k_ia5text);
        get_flag(u1,mrif$m_telex,mrif$k_telex);
        get_flag(u1,mrif$m_undefined,mrif$k_undefined);
        get_flag(u1,mrif$m_sfd,mrif$k_sfd);
        get_flag(u1,mrif$m_rmsfile,mrif$k_rmsfile);
        get_flag(u1,mrif$m_wpsplus,mrif$k_wpsplus);
        get_flag(u1,mrif$m_ddif,mrif$k_ddif);
        get_flag(u1,mrif$m_voicenotif,mrif$k_voicenotif);
        get_flag(u1,mrif$m_odif,mrif$k_odif);
        ok:=mrif$put_flags(msgctx,mrif$k_encodedtypes,u1);
        check(ok);
        writeln;

        END;

END;
```

```
PROCEDURE mr_address;

{ Procedure to prompt the user for a mr_address :
  <mr_address>::=<userid> + { <route> }
{
  Note that <userid> must be a M.R. mailbox name of a subscriber or of a
  remote domain.
}

BEGIN

get_string(mrif$k_userid);

REPEAT get_string(mrif$k_route); UNTIL (length(string)=0);

END;


PROCEDURE domainspecific(item : INTEGER);

{ Procedure to prompt for <domainspecific>::=[<telephone>]+[<location>]
}

BEGIN

REPEAT

        yn:=get_choice(item);
        if ((yn='y') or (yn='Y'))
        then BEGIN
                get_string(mrif$k_telephone);
                get_string(mrif$k_location);
                END;

UNTIL ((length(yn)=0) or (yn='n') or (yn='N') or (item=mrxtester$k_domain_1));

END;



PROCEDURE x400_orname_1;

{ Procedure to prompt for X.400_orname_1 :
<orname_1>::=<x121address> + [<terminalid>] + <mr_address>
}

BEGIN

get_string(mrif$k_x121address);
get_string(mrif$k_terminalid);

writeln('<mr_address>');
mr_address;

END;



PROCEDURE x400_orname_2;

{ Procedure to prompt for X.400_orname_2 :
<orname_2>::=<country>+<amdname>+<uniqueuaid>+<mr_address>+{<domainspecific>}
{}

BEGIN

get_string(mrif$k_country);
get_string(mrif$k_amdname);
get_string(mrif$k_uniqueuaid);
writeln('<mr_address>');
mr_address;
domainspecific(mrxtester$k_domain_1);

END;



PROCEDURE x400_orname_3;

{ Procedure to prompt for the X.400_orname_3 :
<orname_3>::=<country>+<amdname>+<x121address>+<mr_address>+{<domainspecific>}
{}

BEGIN
```

```
get_string(mrif$k_country);
get_string(mrif$k_amdname);
get_string(mrif$k_x121address);
writeln('<mr_address> (gr)');
mr_address;
domainspecific(mrxtester$k_domain_1);

END;


PROCEDURE x400_orname_4;

{ Procedure X.400_orname_4 :
<orname_4>::=<country>+<amdname>+[<choice>]
}

BEGIN

get_string(mrif$k_country);
get_string(mrif$k_amdname);
writeln('<mr_address>');
mr_address;

writeln('[ <CHOICE> : At least one of the followings. ]');
writeln('[ <personnal_name> ]');
writeln;

get_string(mrif$k_surname);
get_string(mrif$k_givenname);
get_string(mrif$k_initials);
get_string(mrif$k_generation);
get_string(mrif$k_prdname);
get_string(mrif$k_orgname);
REPEAT get_string(mrif$k_orgunit); UNTIL (length(string)=0);
domainspecific(mrxtester$k_domain_1);

END;


PROCEDURE orname;

{ Procedure to prompt for an X.400_orname formats supported by MRX
}

BEGIN

yn:=get_choice(mrxtester$k_orname);
if yn='1' then x400_orname_1
else if yn='2' then x400_orname_2
else if yn='3' then x400_orname_3
else if yn='4' then x400_orname_4;

END;

PROCEDURE get_name(msg_part,item : INTEGER);

{ Procedure to construct a name in the form required by its place in the
  message.

  Permitted name items on ENVELOPE : sender, to, cc, bcc
  Permitted name items on CONTENT  : from, author, to, cc, bcc, replytousers

  Input          : msg_part,item
  Implicit Input : msgctx

}

BEGIN

ok:=mrif$start_put(msgctx,mrif$k_name,item);
check(ok);
CASE msg_part OF
       mrif$k_envelope : BEGIN
                           orname;
                           CASE item OF
                              mrif$k_sender :;
                              mrif$k_to,
                              mrif$k_cc,
                              mrif$k_bcc : BEGIN
                                           perrecflg;
                                           get_integer(mrif$k_extensionid);
```

```
                                        get_integer(mrif$k_explicconv);
                                        END;
                              END;
                         END;
          mrif$k_content  : BEGIN
                              get_string(mrif$k_freeform);
                              orname;
                              CASE item OF
                                   mrif$k_from,
                                   mrif$k_author,
                                   mrif$k_replytousers :;
                                   mrif$k_to,
                                   mrif$k_cc,
                                   mrif$k_bcc : BEGIN
                                                reportflg;
                                                get_integer(mrif$k_replyreq);
                                                END;
                              END;
                         END;
     END;

ok:=mrif$end_put(msgctx);
check(ok);

END;

PROCEDURE recipient(msg_part,item,flag : INTEGER); FORWARD;

PROCEDURE enter_list(msg_part,item:INTEGER; listname:str_type);

{ This procedure helps to construct list of names, by calling recursively
  the RECIPIENT procedure. The user is asked for the number of element the
  list will contain. Note that a list in a list will count for one element.

  Input : msg_part : (envelope or content),
          item     : (to, cc, bcc, replytousers),
          listname : the name of the current list.

}

VAR  nb_list,i : INTEGER;
     display   : str_type;

BEGIN

   display:='';
   CASE msg_part OF
        mrif$k_envelope : display := 'env_';
        mrif$k_content  : display := 'cnt_';
        END;
   CASE item OF
        mrif$k_to  : display := display + 'to';
        mrif$k_cc  : display := display + 'cc';
        mrif$k_bcc : display := display + 'bcc';
        END;
   nb_list:=0;
   write('     Enter <',display,'> ',listname,' list range : ');
   readln(nb_list);
   for i:=1 to nb_list do
   BEGIN
      writeln('---> list item ',i:2,'/',nb_list:2,' of ',listname);
      recipient(msg_part,item,1);
   END;
   writeln('---> end of list ',listname,' <',display,'>');

END;


PROCEDURE get_list(msg_part,item: INTEGER);

{ Procedure to construct a list and add it to the message.

  Lists can occur with : to, cc, bcc, replytousers

  Input          : msg_part,item
  Implicit Input : msgctx

}

VAR listname : str_type;

BEGIN

ok:=mrif$start_put(msgctx,mrif$k_list,item);
```

```
check(ok);
get_string(mrif$k_listname);
listname:=string;
perlistflg;
enter_list(msg_part,item,listname);
ok:=mrif$end_put(msgctx);
check(ok);

END;


PROCEDURE recipient;

{ Procedure to construct a recipient (name or list)

  Input          : msg_part,
                   item,
                   flag,
  Implicit Input : msgctx
}

VAR nl,yn : str_type;

BEGIN

REPEAT

if (flag=0) then yn:=get_choice(item);
if ((yn='y') or (yn='Y') or (flag=1))
then BEGIN

        nl:=get_choice(mrxtester$k_name_list);
        if ((nl='N') or (nl='n')) then get_name(msg_part,item)
        else if ((nl='L') or (nl='l')) then get_list(msg_part,item);

     END;

UNTIL ((length(yn)=0) or (yn='n') or (yn='N') or (flag=1));

END;

PROCEDURE build_header;

BEGIN

writeln;
writeln('[ <user_header> ] (gr) ----------------------------------------------
--------');
yn:='';

get_string(mrif$k_app_message_id);
get_string(mrif$k_subject);
get_string(mrif$k_inreplyto);
REPEAT get_string(mrif$k_obsoletes); UNTIL (length(string)=0);
REPEAT get_string(mrif$k_references); UNTIL (length(string)=0);
get_string(mrif$k_msgclass);
get_integer(mrif$k_precedence);
get_integer(mrif$k_autoforward);
get_integer(mrif$k_sensitivity);
get_date(mrif$k_pdate);
get_date(mrif$k_enddate);
get_date(mrif$k_replyby);
get_date(mrif$k_date);

yn:=get_choice(mrif$k_from);
if ((yn='y') or (yn='Y')) then get_name(mrif$k_content,mrif$k_from);

REPEAT
yn:=get_choice(mrif$k_author);
if ((yn='y') or (yn='Y')) then get_name(mrif$k_content,mrif$k_author);
UNTIL ((length(yn)=0) or (yn='n') or (yn='N'));

recipient(mrif$k_content,mrif$k_to,0);
recipient(mrif$k_content,mrif$k_cc,0);
recipient(mrif$k_content,mrif$k_bcc,0);
recipient(mrif$k_content,mrif$k_replytousers,0);

writeln;

END;


PROCEDURE build_body;
```

```
BEGIN

writeln('We just limit ourselves to <text_bodypart> for the moment.');

REPEAT

string:='';
write('<text_bodypart> (file) : ');
readln(string);
if length(string)<>0 then BEGIN

                    ok:=mrif$start_put(msgctx,mrif$k_bodypart,mrif$k_text);
                    check(ok);
                    ok:=mrif$put_text_file(msgctx,string);
                    if not odd(ok) then lib$signal(ok);
                    ok:=mrif$end_put(msgctx);
                    check(ok);

                         END;

yn:=get_choice(mrxtester$k_body);

UNTIL ((length(yn)=0) or (yn='n') or (yn='N'));

END;


PROCEDURE build_envelope;

BEGIN

ok:=mrif$select_assemble(msgctx,mrif$k_envelope);
check(ok);
writeln('<user_envelope> ----------------------------------------------------
--------');
writeln;
writeln('[ <message_id> ] (sys:always)');

get_string(mrif$k_uacontid);
get_integer(mrif$k_hopcount);
get_integer(mrif$k_contenttypes);
get_integer(mrif$k_precedence);
writeln('[ <env_pdate> ] (sys:always) ');
get_date(mrif$k_deferred);
encodedtypes;
permsgflg;
yn:=get_choice(mrif$k_sender);
if ((yn='y') or (yn='Y')) then get_name(mrif$k_envelope,mrif$k_sender);
recipient(mrif$k_envelope,mrif$k_to,0);
recipient(mrif$k_envelope,mrif$k_cc,0);
recipient(mrif$k_envelope,mrif$k_bcc,0);

writeln;

END;

PROCEDURE build_content;

BEGIN

if complete
then ok:=mrif$select_assemble(msgctx,mrif$k_complete_message)
else ok:=mrif$select_assemble(msgctx,mrif$k_content);
check(ok);
writeln;
writeln('<user_content> ------------------------------------------------------
--------');
yn:=get_choice(mrxtester$k_header);
if ((length(yn)=0) or (yn='Y') or (yn='y')) then build_header;
writeln;
yn:=get_choice(mrxtester$k_body);
if ((length(yn)=0) or (yn='Y') or (yn='y')) then build_body;
writeln;

END;

{ ------------------------------------------------------------------------ }

BEGIN

writeln;
writeln('CREATION OF PATTERN BEGINS ...');
```

```
outfile:='';

{ Prompt the user for the result file name}

REPEAT

   write('Output File      : ');
   readln(outfile);
   if (outfile='?') then mrxtester$get_help(mrxtester$k_buioutput)
   else outfile:=no_punct(outfile);

UNTIL ((length(outfile)<>0) and (outfile<>'?'));

{ Prompt the user for the message type to build.
  Message type can be ENVELOPE, CONTENT, BOTH Envelope or Content or
  Complete_Message that is, the user build only the Content, the program
  will create the Envelope from the data it will found in the User Header
}

complete:=FALSE;
envelope:=FALSE;
content:=FALSE;
writeln;
yn:=get_choice(mrxtester$k_buimsgtype);
if ((yn='m') or (yn='M')) then complete:=TRUE;
if ((yn='e') or (yn='E')) then envelope:=TRUE;
if ((yn='c') or (yn='C')) then content:=TRUE;
if ((yn='b') or (yn='B')) then both:=TRUE;
writeln;

{ Starting message assembly }

ok:=mrif$start_assemble(msgctx,0);
assemble:=TRUE;
check(ok);
if ((both) or (complete))
then BEGIN

        envfile:=outfile+'$env';
        cntfile:=outfile+'$cnt';

     END;

if ((both) or (envelope)) then build_envelope;
if ((complete) or (both) or (content)) then build_content;

{ Write the message out to the file(s) }

writeln;
if ((both) or (complete))
then BEGIN
     ok:=mrif$end_assemble(msgctx,cntfile,envfile);
     writeln('%MRXTESTER-I-ENVLFLNM, Envelope file name is ',envfile,'.');
     writeln('%MRXTESTER-I-CONTFLNM, Content file name is ',cntfile,'.');
     END
else if envelope
     then BEGIN
          ok:=mrif$end_assemble(msgctx,,outfile);
          writeln('%MRXTESTER-I-ENVLFLNM, Envelope file name is ',outfile,'.');
          END
else if content
     then BEGIN
          ok:=mrif$end_assemble(msgctx,outfile);
          writeln('%MRXTESTER-I-CONTFLNM, Content file name is ',outfile,'.');
          END;
check(ok);

writeln('%MRXTESTER-I-CRTNCMPLT, The creation is now complete.');
writeln;
if ((both) or (complete)) then post_message;

END.


HELIOS>
```

MRXTESTER$POST.PAS

```
[INHERIT('sys$library:MRIFDEFS.PEN','sys$library:STARLET.PEN')]

PROGRAM mrxtester$post(input,output);


TYPE
   str_type = VARYING[133] OF CHAR;
   $QUAD    = [QUAD,UNSAFE] RECORD
                            LO:UNSIGNED;
                            L1:INTEGER;
                            END;
   $UBYTE   = [BYTE] 0..255;
   $UWORD   = [WORD] 0..65535;

VAR
   msgfile,                              { Name of Message to post }
   envfile,                                    { Message Envelope }
   cntfile,                                     { Message Content }
   yn                                  { User's answer to questions }
            : str_type;

   ok,
   msgctx,                             { Message context identifier }
   lnkctx                                 { Link context identifier }
            : UNSIGNED;
   complete,
   envelope,
   content,                                   { Message type to post }
   connected                           { Indicates connection to MR }
            : BOOLEAN;
   time
            : $QUAD;

{  Declaration for Message Router Programmers Kit Routines.
   These are non-standard PASCAL declarations that use extensions to
   Pascal and are used in a similar way to the Run Time Library. The
   Interface Routines follow the VMS Procedure Calling Standard.
}

[EXTERNAL]
   FUNCTION mrif$start_assemble(VAR msgctxid : UNSIGNED;
                                    flag : UNSIGNED := %IMMED 0)
                                :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$select_assemble(msgctxid:UNSIGNED;
                                  msgtype:UNSIGNED:=%IMMED 0)
                                 :UNSIGNED;EXTERNAL;
[EXTERNAL]
   FUNCTION mrif$end_assemble(msgctxid:UNSIGNED;
                              %DESCR contents:VARYING[a1] OF CHAR:=%IMMED 0;
                              %DESCR envelope:VARYING[b1] OF CHAR:=%IMMED 0)
                                :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$start_put(msgctxid:UNSIGNED;
                           class:UNSIGNED;
                           item:UNSIGNED)
                                 :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$end_put(msgctxid:UNSIGNED):UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_string(msgctxid:UNSIGNED;
                            item:UNSIGNED;
                            %DESCR string:VARYING[a1] OF CHAR)
                            :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_integer(msgctxid:UNSIGNED;
                             item:UNSIGNED;
                             %REF number:ARRAY[u1..u2:INTEGER] OF $UBYTE;
                             size:$UWORD:=%IMMED 0)
                             :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_flags(msgctxid:UNSIGNED;
                           item:UNSIGNED;
                           flags:UNSIGNED)
                           :UNSIGNED;EXTERNAL;
```

```
[EXTERNAL]
   FUNCTION mrif$put_date(msgctxid : UNSIGNED;
                          item:UNSIGNED;
                          date:$QUAD)
                          :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_text_record(msgctxid:UNSIGNED;
                          %DESCR string:VARYING[a1] OF CHAR;
                          flags:UNSIGNED:=%IMMED 0)
                         :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_text_file(msgctxid:UNSIGNED;
                               %DESCR string:VARYING[a1] OF CHAR)
                              :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$start_nbs_put(msgctxid:UNSIGNED;
                               nbstype:UNSIGNED;
                               nbsqual:UNSIGNED) :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$end_nbs_put(msgctxid:UNSIGNED):UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_nbs_record(msgctxid:UNSIGNED;
                                %DESCR string:VARYING[a1] OF CHAR)
                               :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$connect(VAR lnkctx:UNSIGNED;
                         %DESCR node:VARYING[a1] OF CHAR:=%IMMED 0)
                         :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$disconnect(lnkctx:UNSIGNED):UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$identify(lnkctx:UNSIGNED;
                          %DESCR mailbox:VARYING[a1] OF CHAR;
                          %DESCR password:VARYING[b1] OF CHAR:=%IMMED 0)
                          :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$post(lnkctx:UNSIGNED;
                      %DESCR content:VARYING[a1] OF CHAR;
                      %DESCR envelope:VARYING[b1] OF CHAR:=%IMMED 0;
                      flags:UNSIGNED:=%IMMED 0;
                      %DESCR msgid:VARYING[c1] OF CHAR:=%IMMED 0)
                       :UNSIGNED;EXTERNAL;

{ Declare Run Time Library that wil be needed. }

[EXTERNAL]FUNCTION LIB$STOP(%IMMED sts:UNSIGNED):UNSIGNED;EXTERNAL;

[EXTERNAL]FUNCTION LIB$SIGNAL(%IMMED sts:UNSIGNED):UNSIGNED;EXTERNAL;

FUNCTION no_punct(s : str_type):str_type;

VAR
    str : str_type;
    p   : INTEGER;

BEGIN

    str:=s;
    p:=INDEX(s,'.');
    if p<>0 then str:=substr(s,1,p-1);
    no_punct:=str;

END;

PROCEDURE check(status_code:UNSIGNED);

{ Procedure to determine whether the status returned by another routine
  was a VMS success or error status. If the routine returned an error,
  the program terminates.

  Input : status_code - a 32 bit VMS status returned by another routine
  Implicit input msgctx or lnkctx
}

BEGIN
```

```
IF NOT ODD(status_code)          { Bottom bit indicates success }
THEN
    BEGIN
    IF connected THEN mrif$disconnect(lnkctx);
    writeln('MRXTESTER-F-FTLERR, ***** FATAL ERROR *****');
    writeln('The program will terminate due to the reason bellow : ');
    writeln;
    LIB$SIGNAL(status_code);
    $exit;
    END;
END;


PROCEDURE connect_mr;

{ Procedure to connect to Message Router on the local node
}

CONST

    node = 'VENUS';

BEGIN

    writeln('Connecting to node ',node);
    ok:=mrif$connect(lnkctx,node);
    if not odd(ok)
    then BEGIN
         writeln('Unable to connect to M.R. on local node.');
         connected:=FALSE;
         check(ok);
         END
    else connected:=TRUE;
END;

PROCEDURE identify_mr;

CONST

    def_mbx = 'MRXTESTER';
    def_pwd = 'MRXTESTER';

VAR

    mbx,
    pwd     :str_type;

BEGIN

    mbx:=def_mbx;
    pwd:=def_pwd;
    ok:=mrif$identify(lnkctx,mbx,pwd);
    if not odd(ok)
    then BEGIN
         writeln('Unable to identify to mailbox ',def_mbx);
         REPEAT
             write('Mailbox   : ');
             readln(mbx);
             write('Password  : ');
             readln(pwd);
             if length(pwd)<>0
             then ok:=mrif$identify(lnkctx,mbx,pwd)
             else ok:=mrif$identify(lnkctx,mbx);
             if not odd(ok)
             then writeln('Unable to identify to mailbox ',mbx);
         UNTIL odd(ok);
         END;
    writeln('Identified to mailbox ',mbx);
END;

PROCEDURE post_mr;

VAR
    msg_id : str_type;

BEGIN

if complete
then BEGIN
     envfile:=msgfile+'$env';
     cntfile:=msgfile+'$cnt';
     END;
ok:=mrif$post(lnkctx,cntfile,envfile,0,msg_id);
check(ok);
```

```
writeln('Message posted successfully with message_id : ',msg_id);

END;

PROCEDURE disconnect_mr;

BEGIN

ok:=mrif$disconnect(lnkctx);
connected:=FALSE;
check(ok);

END;

BEGIN

   connected:=FALSE;
   connect_mr;
   if connected
   then BEGIN

        identify_mr;
        msgfile:='';
        envfile:='';
        cntfile:='';
        yn:='';
        REPEAT

          complete:=FALSE;
          write('Complete message file : ');
          readln(msgfile);
          msgfile:=no_punct(msgfile);
          write('Envelope message file : ');
          readln(envfile);
          msgfile:=no_punct(msgfile);
          write('Content message file  : ');
          readln(cntfile);

          msgfile:=no_punct(msgfile);
          if (length(msgfile)<>0) then complete:=TRUE;
          if ((complete) or (length(envfile)<>0) or (length(cntfile)<>0))
          then post_mr;


          writeln;
          write('Do you want to post another message [Y] ? ');
          readln(yn);
          IF yn='n' THEN yn:='N';

          UNTIL (yn='N');
          disconnect_mr;

     END;

END.
```

MRXTESTERSORNAMES.PAS

```
[INHERIT('sys$library:MRIFDEFS.PEN','sys$library:STARLET.PEN')]

PROGRAM mrxtester$ornames(input,output);

TYPE
   str_type = VARYING[133] OF CHAR;
   txt_type = VARYING[3000] OF CHAR;
   $QUAD    = [QUAD,UNSAFE] RECORD
                            LO:UNSIGNED;
                            L1:INTEGER;
                            END;
   $UBYTE   = [BYTE] 0..255;
   $UWORD   = [WORD] 0..65535;

VAR
   string, envfile, cntfile, outfile, fichier,
   testname                                        : str_type;
   number, numero,format, i                        : INTEGER;
   ok, msgctx, lnkctx, u1, assemble_flag           : UNSIGNED;
   complete, envelope, content, both, assemble,
   connected, err                                  : BOOLEAN;
   texte                                           : txt_type;
   out                                             : TEXT;
   x121adr, terminalid, userid, country,
   amdname, uniqueuaid, surname,
   givenname, initials, generation,
   prdname, orgname, orgunit, telephone,
   location, txt1, badvalue,
   x12, ter, use, cou, amd, uni, sur, giv,
   ini, gen, prd, orgn, orgu, tel, loc             : str_type;
   time                                            : $QUAD;

{ ------------------------------------------------------------------------

   Declaration for Message Router Programmers Kit Routines.
   These are non-standard PASCAL declarations that use extensions to
   Pascal and are used in a similar way to the Run Time Library. The
   Interface Routines follow the VMS Procedure Calling Standard.
}

[EXTERNAL]
   FUNCTION mrif$start_assemble(VAR msgctxid : UNSIGNED;
                                    flag : UNSIGNED := %IMMED 0)
                                :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$select_assemble(msgctxid:UNSIGNED;
                                  msgtype:UNSIGNED:=%IMMED 0)
                                :UNSIGNED;EXTERNAL;
[EXTERNAL]
   FUNCTION mrif$end_assemble(msgctxid:UNSIGNED;
                              %DESCR contents:VARYING[a1] OF CHAR:=%IMMED 0;
                              %DESCR envelope:VARYING[b1] OF CHAR:=%IMMED 0)
                                :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$start_put(msgctxid:UNSIGNED;
                           class:UNSIGNED;
                           item:UNSIGNED)
                                :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$end_put(msgctxid:UNSIGNED):UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_string(msgctxid:UNSIGNED;
                            item:UNSIGNED;
                            %DESCR string:VARYING[a1] OF CHAR)
                            :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_integer(msgctxid:UNSIGNED;
                             item:UNSIGNED;
                             %REF number:ARRAY[u1..u2:INTEGER] OF $UBYTE;
                             size:$UWORD:=%IMMED 0)
                             :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_flags(msgctxid:UNSIGNED;
                           item:UNSIGNED;
                           flags:UNSIGNED)
                          :UNSIGNED;EXTERNAL;
```

```
[EXTERNAL]
   FUNCTION mrif$put_date(msgctxid : UNSIGNED;
                          item:UNSIGNED;
                          date:$QUAD)
                          :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_text_record(msgctxid:UNSIGNED;
                          %DESCR string:VARYING[a1] OF CHAR;
                          flags:UNSIGNED:=%IMMED 0)
                         :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_text_file(msgctxid:UNSIGNED;
                               %DESCR string:VARYING[a1] OF CHAR)
                              :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$start_nbs_put(msgctxid:UNSIGNED;
                               nbstype:UNSIGNED;
                               nbsqual:UNSIGNED) :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$end_nbs_put(msgctxid:UNSIGNED):UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_nbs_record(msgctxid:UNSIGNED;
                                %DESCR string:VARYING[a1] OF CHAR)
                               :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$connect(VAR lnkctx:UNSIGNED;
                         %DESCR node:VARYING[a1] OF CHAR:=%IMMED 0)
                         :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$disconnect(lnkctx:UNSIGNED):UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$identify(lnkctx:UNSIGNED;
                          %DESCR mailbox:VARYING[a1] OF CHAR;
                          %DESCR password:VARYING[b1] OF CHAR:=%IMMED 0)
                          :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$post(lnkctx:UNSIGNED;
                      %DESCR content:VARYING[a1] OF CHAR;
                      %DESCR envelope:VARYING[b1] OF CHAR:=%IMMED 0;
                      flags:UNSIGNED:=%IMMED 0;
                      %DESCR msgid:VARYING[c1] OF CHAR:=%IMMED 0)
                       :UNSIGNED;EXTERNAL;

{ Declare Run Time Library that will be needed. }

[EXTERNAL]FUNCTION LIB$STOP(%IMMED sts:UNSIGNED):UNSIGNED;EXTERNAL;

[EXTERNAL]FUNCTION LIB$SIGNAL(%IMMED sts:UNSIGNED):UNSIGNED;EXTERNAL;

{ ------------------------------------------------------------------------- }

PROCEDURE check(status_code:UNSIGNED);

{ Procedure to determine whether the status returned by another routine
  was a VMS success or error status. If the routine returned an error,
  the program terminates.

  Input : status_code - a 32 bit VMS status returned by another routine
  Implicit input msgctx or lnkctx
}

BEGIN

IF NOT ODD(status_code)          { Bottom bit indicates success }
THEN
    BEGIN

    IF assemble THEN mrif$end_assemble(msgctx);
    IF connected THEN mrif$disconnect(lnkctx);
    writeln('MRXTESTER-F-FTLERR, ***** FATAL ERROR *****');
    writeln('The program will terminate due to the reason given below :');
    writeln;
    LIB$SIGNAL(status_code);
    $exit;
```

```
    END;
END;

{ ------------------------------------------------------------------------ }

PROCEDURE perrecflg;
VAR   u1 : UNSIGNED;
BEGIN
   u1:=0;
   u1:=UOR(u1,mrif$m_action);
   u1:=UOR(u1,mrif$m_ua_confirmed); u1:=UOR(u1,mrif$m_mr_confirmed);
   ok:=mrif$put_flags(msgctx,mrif$k_perrecflg,u1);
   check(ok);
END;

PROCEDURE ouvrir;
BEGIN
  ok:=mrif$start_assemble(msgctx,0);
  assemble:=TRUE;
  check(ok);
  outfile:='ORTEST'+testname+DEC(format,1)+DEC(numero,2);
  envfile:=outfile+'$env';
  cntfile:=outfile+'$cnt';
END;

PROCEDURE fermer;
BEGIN
  ok:=mrif$end_assemble(msgctx,cntfile,envfile);
  assemble:=FALSE;
  check(ok);
  writeln('TEST built with filenames : ',envfile,' & ',cntfile);
END;

PROCEDURE Uenvelope;
BEGIN
  ok:=mrif$select_assemble(msgctx,mrif$k_envelope);
  check(ok);
  string:='TEST O/R names ['+fichier+DEC(format,2)+DEC(numero,3)+']';
  ok:=mrif$put_string(msgctx,mrif$k_uacontid,string);
  check(ok);
END;

PROCEDURE Ucontent;
{ Procedure to create a body containing
  subject    : TEST O/R NAMES
  date       : (today)
  text       : (4 lines)
}
BEGIN
  ok:=mrif$select_assemble(msgctx,mrif$k_content);
  check(ok);
  string:='TEST O/R NAMES';
  ok:=mrif$put_string(msgctx,mrif$k_subject,string);
  check(ok);
  ok:=$gettim(time);
  ok:=mrif$put_date(msgctx,mrif$k_date,time);
  check(ok);

  { texte }
  ok:=mrif$start_put(msgctx,mrif$k_bodypart,mrif$k_text);
  check(ok);
  for i:=1 to 3 do
  BEGIN
     ok:=mrif$put_text_record(msgctx,texte,mrif$m_crlf);
     check(ok);
  END;
  ok:=mrif$put_text_record(msgctx,txt1,mrif$m_crlf);
  check(ok);
  ok:=mrif$end_put(msgctx);
  check(ok);

  numero := numero +1;
END;

PROCEDURE or1(x12,ter:str_type);
BEGIN
  ouvrir;
  Uenvelope;

  { recipient }
  ok:=mrif$start_put(msgctx,mrif$k_name,mrif$k_to);
  check(ok);
  if x12<>'' then ok:=mrif$put_string(msgctx,mrif$k_x121address,x12);
```

```
  check(ok);
  if ter<>'' then ok:=mrif$put_string(msgctx,mrif$k_terminalid,ter);
  check(ok);
  ok:=mrif$put_string(msgctx,mrif$k_userid,userid);
  check(ok);
  ok:=mrif$end_put(msgctx);
  check(ok);

  Ucontent;
  fermer;
END;

PROCEDURE x400_orname_1;
BEGIN
  txt1:='KO : orname1 vide';    or1('','');
  txt1:='OK : orname1 minimum'; or1(x121adr,'');
  txt1:='OK : orname1 complet'; or1(x121adr,terminalid);
  txt1:='KO : orname1 errone';  or1(badvalue,badvalue);
END;

PROCEDURE or2(cou,amd,uni,tel,loc:str_type);
BEGIN
  ouvrir;
  Uenvelope;

  { recipient }
  ok:=mrif$start_put(msgctx,mrif$k_name,mrif$k_to);
  check(ok);
  if cou<>'' then ok:=mrif$put_string(msgctx,mrif$k_country,cou);
  check(ok);
  if amd<>'' then ok:=mrif$put_string(msgctx,mrif$k_amdname,amd);
  check(ok);
  if uni<>'' then ok:=mrif$put_string(msgctx,mrif$k_uniqueuaid,uni);
  check(ok);
  ok:=mrif$put_string(msgctx,mrif$k_userid,userid);
  check(ok);
  if tel<>'' then ok:=mrif$put_string(msgctx,mrif$k_telephone,tel);
  check(ok);
  if loc<>'' then ok:=mrif$put_string(msgctx,mrif$k_location,loc);
  check(ok);
  ok:=mrif$end_put(msgctx);
  check(ok);

  Ucontent;
  fermer;
END;

PROCEDURE x400_orname_2;
BEGIN
 txt1:='KO: orname2 vide';    or2('','','','','');
 txt1:='OK: orname2 minimum'; or2(country,amdname,uniqueuaid,'','');
 txt1:='OK: orname2 complet'; or2(country,amdname,uniqueuaid,telephone,location)
;
 txt1:='KO: orname2 errone';  or2(badvalue,badvalue,badvalue,'','');
END;

PROCEDURE or3(cou,amd,x12,tel,loc:str_type);
BEGIN
  ouvrir;
  Uenvelope;

  { recipient }
  ok:=mrif$start_put(msgctx,mrif$k_name,mrif$k_to);
  check(ok);
  if cou<>'' then ok:=mrif$put_string(msgctx,mrif$k_country,cou);
  check(ok);
  if amd<>'' then ok:=mrif$put_string(msgctx,mrif$k_amdname,amd);
  check(ok);
  if x12<>'' then ok:=mrif$put_string(msgctx,mrif$k_x121address,x12);
  check(ok);
  ok:=mrif$put_string(msgctx,mrif$k_userid,userid);
  check(ok);
  if tel<>'' then ok:=mrif$put_string(msgctx,mrif$k_telephone,tel);
  check(ok);
  if loc<>'' then ok:=mrif$put_string(msgctx,mrif$k_location,loc);
  check(ok);
  ok:=mrif$end_put(msgctx);
  check(ok);

  Ucontent;
  fermer;
END;

PROCEDURE x400_orname_3;
```

```
BEGIN
  txt1:='KO: orname3 vide';    or3('','','','','');
  txt1:='OK: orname3 minimum'; or3(country,amdname,x121adr,'','');
  txt1:='OK: orname3 complet'; or3(country,amdname,x121adr,telephone,location);
  txt1:='KO: orname3 errone';  or3(badvalue,badvalue,badvalue,'','');
END;

PROCEDURE or4(cou,amd,sur,giv,ini,gen,prd,orgn,orgu,tel,loc:str_type);
BEGIN
  ouvrir;
  Uenvelope;

  { recipient }
  ok:=mrif$start_put(msgctx,mrif$k_name,mrif$k_to);
  check(ok);
  if cou<>'' then ok:=mrif$put_string(msgctx,mrif$k_country,cou);
  check(ok);
  if amd<>'' then ok:=mrif$put_string(msgctx,mrif$k_amdname,amd);
  check(ok);
  ok:=mrif$put_string(msgctx,mrif$k_userid,userid);
  check(ok);
  if sur<>'' then ok:=mrif$put_string(msgctx,mrif$k_surname,sur);
  check(ok);
  if giv<>'' then ok:=mrif$put_string(msgctx,mrif$k_givenname,giv);
  check(ok);
  if ini<>'' then ok:=mrif$put_string(msgctx,mrif$k_initials,ini);
  check(ok);
  if gen<>'' then ok:=mrif$put_string(msgctx,mrif$k_generation,gen);
  check(ok);
  if prd<>'' then ok:=mrif$put_string(msgctx,mrif$k_prdname,prd);
  check(ok);
  if orgn<>'' then ok:=mrif$put_string(msgctx,mrif$k_orgname,orgn);
  check(ok);
  if orgu<>'' then ok:=mrif$put_string(msgctx,mrif$k_orgunit,orgu);
  check(ok);
  if tel<>'' then ok:=mrif$put_string(msgctx,mrif$k_telephone,tel);
  check(ok);
  if loc<>'' then ok:=mrif$put_string(msgctx,mrif$k_location,loc);
  check(ok);
  ok:=mrif$end_put(msgctx);
  check(ok);

  Ucontent;
  fermer;
END;

PROCEDURE x400_orname_4;
BEGIN
  txt1:='KO: orname4 vide';
  or4('','','','','','','','','','','');
  txt1:='OK: orname4 minimum';
  or4(country,amdname,surname,'','','','','','','','');
  txt1:='OK: orname4 maximum';
  or4(country,amdname,surname,givenname,initials,generation,prdname,orgname,
      orgunit,telephone,location);
  txt1:='KO: orname4 errone';
  or4(badvalue,badvalue,badvalue,'','','','','','','','');
END;

{ --- Generateur automatique de cas de tests pour O/R names --- }

BEGIN

  connected:=FALSE;
  assemble:=FALSE;

  writeln;
  writeln('This program AUTOMATICALY generates suites of O/R naming and adressin
g tests');
  writeln('patterns which will have to be used with the X400 Message Tester');
  writeln('                                                (c) Me & Drx 1987');
  writeln('Results files are called "ORTESTname(1-4)(1-n)$ENV.NBS" (envelope)');

  writeln('                      for "ORTESTname(1-4)(1-n)$CNT.NBS" (contents)');

  writeln;
  write('Enter name of TEST [] :');
  readln(testname);
  writeln;
  writeln('(ORIGINATOR is always MRXTESTER)');
  writeln;
  writeln('Please enter DATA on RECIPIENT :');
  write ('x121adr                : '); readln(x121adr);
  write ('terminalid             : '); readln(terminalid);
```

```
        write ('country     [BE]        : '); readln(country);
        if country='' then country:='BE';
        write ('amdname     [RTT]       : '); readln(amdname);
        if amdname='' then amdname:='RTT';
        write ('uniqueuaid              : '); readln(uniqueuaid);
        write ('surname                 : '); readln(surname);
        write ('givenname               : '); readln(givenname);
        write ('initials                : '); readln(initials);
        write ('generation              : '); readln(generation);
        write ('PRDname                 : '); readln(prdname);
        write ('userid      [MRXTESTER] : '); readln(userid);
        if userid='' then userid:='MRXTESTER';
        write ('ORGname                 : '); readln(orgname);
        write ('orgunit                 : '); readln(orgunit);
        write ('telephone               : '); readln(telephone);
        write ('location                : '); readln(location);
        badvalue:='000000000000';

        writeln('GENERATION OF PATTERN BEGINS ...');
        writeln;

        texte := 'CECI REPRESENTE UNE LIGNE';
        for i:=1 to 3 do
           texte := texte +'¦----------------------------------';

        for format:=1 to 4 do
        BEGIN

           numero := 1;
           txt1:='';

           CASE format OF
           1: x400_orname_1;
           2: x400_orname_2;
           3: x400_orname_3;
           4: x400_orname_4;
           END;

        END;

        writeln;
        writeln('END OF GENERATION');
        writeln;

      END.
```

MRXTESTER$ORPOST.PAS

```
[INHERIT('sys$library:MRIFDEFS.PEN','sys$library:STARLET.PEN')]

PROGRAM mrxtester$post(input,output);


TYPE
   str_type = VARYING[133] OF CHAR;
   $QUAD    = [QUAD,UNSAFE] RECORD
                            LO:UNSIGNED;
                            L1:INTEGER;
                            END;
   $UBYTE   = [BYTE] 0..255;
   $UWORD   = [WORD] 0..65535;

VAR
   msgfile,
   envfile,
   cntfile,
   outfile,
   yn           : str_type;

   ok,
   msgctx,
   lnkctx       : UNSIGNED;
   complete,
   envelope,
   content,
   assemble,
   connected    : BOOLEAN;
   time         : $QUAD;

{  Declaration for Message Router Programmers Kit Routines.
   These are non-standard PASCAL declarations that use extensions to
   Pascal and are used in a similar way to the Run Time Library. The
   Interface Routines follow the VMS Procedure Calling Standard.
}

[EXTERNAL]
   FUNCTION mrif$start_assemble(VAR msgctxid : UNSIGNED;
                                    flag : UNSIGNED := %IMMED 0)
                                :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$select_assemble(msgctxid:UNSIGNED;
                                  msgtype:UNSIGNED:=%IMMED 0)
                                :UNSIGNED;EXTERNAL;
[EXTERNAL]
   FUNCTION mrif$end_assemble(msgctxid:UNSIGNED;
                              %DESCR contents:VARYING[a1] OF CHAR:=%IMMED 0;
                              %DESCR envelope:VARYING[b1] OF CHAR:=%IMMED 0)
                                :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$start_put(msgctxid:UNSIGNED;
                           class:UNSIGNED;
                           item:UNSIGNED)
                                :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$end_put(msgctxid:UNSIGNED):UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_string(msgctxid:UNSIGNED;
                            item:UNSIGNED;
                            %DESCR string:VARYING[a1] OF CHAR)
                            :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_integer(msgctxid:UNSIGNED;
                             item:UNSIGNED;
                             %REF number:ARRAY[u1..u2:INTEGER] OF $UBYTE;
                             size:$UWORD:=%IMMED 0)
                             :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_flags(msgctxid:UNSIGNED;
                           item:UNSIGNED;
                           flags:UNSIGNED)
                          :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_date(msgctxid : UNSIGNED;
```

```
                              item:UNSIGNED;
                              date:$QUAD)
                              :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_text_record(msgctxid:UNSIGNED;
                              %DESCR string:VARYING[a1] OF CHAR;
                              flags:UNSIGNED:=%IMMED 0)
                             :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_text_file(msgctxid:UNSIGNED;
                               %DESCR string:VARYING[a1] OF CHAR)
                              :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$start_nbs_put(msgctxid:UNSIGNED;
                               nbstype:UNSIGNED;
                               nbsqual:UNSIGNED) :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$end_nbs_put(msgctxid:UNSIGNED):UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_nbs_record(msgctxid:UNSIGNED;
                                %DESCR string:VARYING[a1] OF CHAR)
                               :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$connect(VAR lnkctx:UNSIGNED;
                         %DESCR node:VARYING[a1] OF CHAR:=%IMMED 0)
                         :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$disconnect(lnkctx:UNSIGNED):UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$identify(lnkctx:UNSIGNED;
                          %DESCR mailbox:VARYING[a1] OF CHAR;
                          %DESCR password:VARYING[b1] OF CHAR:=%IMMED 0)
                          :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$post(lnkctx:UNSIGNED;
                      %DESCR content:VARYING[a1] OF CHAR;
                      %DESCR envelope:VARYING[b1] OF CHAR:=%IMMED 0;
                      flags:UNSIGNED:=%IMMED 0;
                      %DESCR msgid:VARYING[c1] OF CHAR:=%IMMED 0)
                       :UNSIGNED;EXTERNAL;

{ Declare Run Time Library that wil be needed. }

[EXTERNAL]FUNCTION LIB$STOP(%IMMED sts:UNSIGNED):UNSIGNED;EXTERNAL;

[EXTERNAL]FUNCTION LIB$SIGNAL(%IMMED sts:UNSIGNED):UNSIGNED;EXTERNAL;

FUNCTION no_punct(s : str_type):str_type;

VAR
    str : str_type;
    p   : INTEGER;

BEGIN

    str:=s;
    p:=INDEX(s,'.');
    if p<>0 then str:=substr(s,1,p-1);
    no_punct:=str;

END;

PROCEDURE check(status_code:UNSIGNED);

{ Procedure to determine whether the status returned by another routine
  was a VMS success or error status. If the routine returned an error,
  the program terminates.

  Input : status_code - a 32 bit VMS status returned by another routine
  Implicit input msgctx or lnkctx
}

BEGIN
IF NOT ODD(status_code)          { Bottom bit indicates success }
THEN
```

```
        BEGIN
        IF assemble THEN mrif$end_assemble(msgctx);
        IF connected THEN mrif$disconnect(lnkctx);
        writeln('MRXTESTER-F-FTLERR, ***** FATAL ERROR *****');
        writeln('The program will terminate due to the reason bellow :');
        writeln;
        LIB$SIGNAL(status_code);
        $exit;
        END;
    END;


    PROCEDURE connect_mr;

    CONST

        node = 'VENUS';

    BEGIN

        writeln('Connecting to node ',node);
        ok:=mrif$connect(lnkctx,node);
        if not odd(ok)
        then BEGIN
             writeln('Unable to connect to M.R. on local node.');
             check(ok);
             END;
        connected:=TRUE;
    END;

    PROCEDURE identify_mr;

    CONST

        def_mbx = 'MRXTESTER';
        def_pwd = 'MRXTESTER';

    VAR

        mbx,
        pwd     :str_type;

    BEGIN

        mbx:=def_mbx;
        pwd:=def_pwd;
        ok:=mrif$identify(lnkctx,mbx,pwd);
        if not odd(ok)
        then BEGIN
             writeln('Unable to identify to mailbox ',def_mbx);
             REPEAT
                 write('Mailbox   : ');
                 readln(mbx);
                 write('Password  : ');
                 readln(pwd);
                 if length(pwd)<>0
                 then ok:=mrif$identify(lnkctx,mbx,pwd)
                 else ok:=mrif$identify(lnkctx,mbx);
                 if not odd(ok)
                 then writeln('Unable to identify to mailbox ',mbx);
             UNTIL odd(ok);
             END;
        writeln('Identified to mailbox ',mbx);
    END;

    PROCEDURE post_mr;

    VAR
        msg_id : str_type;

    BEGIN

    ok:=mrif$post(lnkctx,cntfile,envfile,0,msg_id);
    check(ok);
    writeln('Message posted successfully with message_id : ',msg_id);
    writeln;

    END;

    PROCEDURE disconnect_mr;

    BEGIN

    ok:=mrif$disconnect(lnkctx);
```

```
      connected:=FALSE;
      check(ok);

      END;

      BEGIN

         writeln;
         connected:=FALSE;
         connect_mr;
         identify_mr;
         msgfile:='';
         envfile:='';
         cntfile:='';
         yn:='';
         REPEAT

             write('Name of ORTEST message file : ORTEST');
             readln(msgfile);
             msgfile:=no_punct(msgfile);

             if (length(msgfile)<>0) then
             BEGIN
                envfile:='ORTEST'+msgfile+'$ENV';
                cntfile:='ORTEST'+msgfile+'$CNT';
                writeln('Envelope message file : ',envfile);
                writeln('Content message file  : ',cntfile);
                post_mr;
             END;

             write('Do you want to post another message [Y] ? ');
             readln(yn);
             IF yn='n' THEN yn:='N';
             writeln;

         UNTIL (yn='N');
         disconnect_mr;

      END.
```

MRXTESTER$DISASSEMBLE.PAS

```
(*************************************************************************)
(*                                                                       *)
(* THE ORIGINAL VERSION OF THIS SOFTWARE CAN BEEN FOUND IN THE MESSAGE   *)
(*   ROUTER INTERFACE ROUTINES MANUAL. THE ORIGINAL VERSION HAS BEEN     *)
(*   MODIFIED IN ORDER TO DISASSEMBLE AN MHS FILE INTO A FILE WHOSE      *)
(*   SYNTAX BELONGS TO THE MRXTESTER MESSAGE DESCRIPTION LANGUAGE.       *)
(*                                                                       *)
(* MODIFICATIONS MADE BY ME & DRX 'JAN 88                                *)
(*                                                                       *)
(*************************************************************************)

[INHERIT('d18$:[mrmanager]MRXTESTER$DEFS.PEN','d18$:[mrmanager]MRIFDEFS.PEN','d1
8$:[mrmanager]STARLET.PEN')]
PROGRAM mrxtester$disassemble(INPUT,OUTPUT);

{ Declare data types that are not standard Pascal }
TYPE
   str_type              = VARYING[256] OF CHAR;
   small_byte            = [BYTE] 1..16;
   $QUAD                 = [QUAD,UNSAFE] RECORD
                             LO:UNSIGNED; L1:INTEGER; END;
   $UBYTE                = [BYTE] 0..255;
   $UWORD                = [WORD] 0..65535;

{ Declare variables }
VAR
   out: TEXT;                       { identifies the output file }
   msgctx,                          { disassemble context }
   lnkctx,                           { link context }
   ok,                              { general status variable containing
                                      value returned }
   class,                           { mrif$k_class }
   item,                            { mrif$k_item }
   find_length,                     { length returned by find call }
   bit_value,                       { value of bit returned by GET_FLAGS }
   nbsqual,                         { qualifier returned by START_NBS_GET }
   nbstype:                         { type returned by START_NBS_GET }
            UNSIGNED;

   ascii_item: VARYING [20] OF CHAR; { ASCII version of mrif$k_item }
   int_length,                      { length returned by GET_INTEGER call }
   length_cnt:                      { used as a FOR variable comparing to
                                      int_length }
            $UWORD;
   integer_value:ARRAY[1..16]OF $UBYTE; { value of integer returned by
                                          GET_INTEGER }
   string_value:VARYING[60]OF CHAR; { value of string returned by
                                      GET_STRING }
   date_value:$QUAD;                { value of date returned by GET_DATE }
   ascii_date:PACKED ARRAY[1..26] OF CHAR;
                                    { ASCII version of date_value }
   date_length:$UWORD;               { length returned by $ASCTIM }

   assemble, connected :BOOLEAN;
   outfile,
   inpfile             :str_type;

{  Declaration for Message Router Programmers Kit Routines.
   These are non-standard PASCAL declarations that use extensions to
   Pascal and are used in a similar way to the Run Time Library. The
   Interface Routines follow the VMS Procedure Calling Standard.
}

[EXTERNAL]
   FUNCTION mrif$connect(VAR lnkctx : UNSIGNED;
            %DESCR node : VARYING[a1] OF CHAR := %IMMED 0)
                              :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$disconnect(lnkctx : UNSIGNED) :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$identify(lnkctx:UNSIGNED;
            %DESCR mailbox : VARYING[a1] OF CHAR;
            %DESCR password : VARYING[b1] OF CHAR := %IMMED 0)
                              :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$start_disassemble(VAR msgctxid : UNSIGNED;
            %DESCR contents : VARYING[a1] OF CHAR := %IMMED 0;
            %DESCR envelope : VARYING[b1] OF CHAR := %IMMED 0)
                              :UNSIGNED;EXTERNAL;
```

```
[EXTERNAL]
   FUNCTION mrif$select_disassemble(msgctxid:UNSIGNED;
                                    msgtype:UNSIGNED:=%IMMED 0)
                                   :UNSIGNED;EXTERNAL;
[EXTERNAL]
   FUNCTION mrif$end_disassemble(msgctxid:UNSIGNED)
                                   :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$find(msgctxid : UNSIGNED;
                      VAR class : UNSIGNED;
                      VAR item  : UNSIGNED;
                      range : UNSIGNED;
                      VAR length : UNSIGNED)
                          :UNSIGNED;EXTERNAL;
[EXTERNAL]
   FUNCTION mrif$start_get(msgctxid:UNSIGNED;
                      VAR class:UNSIGNED;
                      VAR item:UNSIGNED)
                                   :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$end_get(msgctxid:UNSIGNED):UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$get_string(msgctxid:UNSIGNED;
                            item:UNSIGNED;
                            %DESCR value:VARYING[a1] OF CHAR;
                            VAR length : $UWORD := %IMMED 0)
                            :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$get_integer(msgctxid:UNSIGNED;
                             item:UNSIGNED;
                             %REF number:ARRAY[u1..u2:INTEGER] OF $UBYTE;
                             VAR length:$UWORD:=%IMMED 0)
                             :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$get_flags(msgctxid:UNSIGNED;
                           item:UNSIGNED;
                           VAR value :UNSIGNED)
                          :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$get_date(msgctxid : UNSIGNED;
                          item:UNSIGNED;
                          VAR value :$QUAD)
                          :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$get_text_record(msgctxid:UNSIGNED;
                          %DESCR value:VARYING[a1] OF CHAR;
                          options:UNSIGNED:=%IMMED 0;
                          VAR length : $UWORD := %IMMED 0)
                         :UNSIGNED;EXTERNAL;


[EXTERNAL]
   FUNCTION mrif$start_nbs_get(msgctxid:UNSIGNED;
                            VAR nbstype:UNSIGNED;
                            VAR nbsqual:UNSIGNED) :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$end_nbs_get(msgctxid:UNSIGNED):UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$get_nbs_record(msgctxid:UNSIGNED;
                               %DESCR value:VARYING[a1] OF CHAR;
                               VAR length : $UWORD := %IMMED 0)
                              :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$isprim(nbstype : UNSIGNED):BOOLEAN;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$fetch(lnkctx:UNSIGNED;
        %DESCR content:VARYING[a1] OF CHAR;
        %DESCR envelope:VARYING[b1] OF CHAR := %IMMED 0;
        flags:UNSIGNED := %IMMED 0)
           :UNSIGNED;EXTERNAL;

[EXTERNAL]
```

```
   FUNCTION mrif$query(lnkctx : UNSIGNED;
                       VAR number : UNSIGNED)
           :UNSIGNED;EXTERNAL;

{ Declare Run Time Library that wil be needed. }

[EXTERNAL]FUNCTION LIB$STOP(%IMMED sts:UNSIGNED):UNSIGNED;EXTERNAL;

[EXTERNAL]FUNCTION LIB$SIGNAL(%IMMED sts:UNSIGNED):UNSIGNED;EXTERNAL;

{ Declare help library (cfr MRXTESTER$BUILD.PAS) }

[EXTERNAL]PROCEDURE mrxtester$get_help(item:INTEGER);EXTERNAL;

PROCEDURE check (status_code : UNSIGNED);

{
  Procedure to determine whether the status returned by another routine
  was a VMS success or error status. If the routine returned an error,
  the program doesn't terminates.

  INPUT : status_code - a 32 bit VMS status returned by another routine
  Implicit Input : msgctx,out
}

BEGIN
IF NOT ODD(status_code)          { Bottom bit indicates success }
THEN
   BEGIN
   mrif$end_disassemble(msgctx);
   close(out);
   LIB$SIGNAL(status_code);
   END;
END;

PROCEDURE init(messagefile:str_type);

{
 This procedure determines whether the message file
 contains the envelope or the content of a message and starts to
 disassemble it. It calls SELECT_DISASSEMBLE to see whether the message is
 a service message or a user message.

 Implicit Output : msgctx - message is left started and selected
                   out - output file is left open for write
}

VAR
    envelope:BOOLEAN;

BEGIN

REPEAT

    { Quit if no more files to look at }
    IF messagefile.length = 0 THEN $EXIT(ss$_normal);

    { Assume message file contains an envelope }
    ok:=mrif$start_disassemble(msgctx,,messagefile);
    envelope:=TRUE;
    IF ok = mrif$_invalidmsg
    THEN
        BEGIN
        { It is not an envelope, so assume it is a content }
        ok:=mrif$start_disassemble(msgctx,messagefile);
        envelope:=FALSE;
        END;
    IF NOT ODD(ok)
    THEN
        BEGIN
        { Message file does not contain an envelope or a content,
          so it cannot be disassembled
        }
        WRITELN('Unable to disassemble message in ',messagefile);
        LIB$SIGNAL(ok);
        END;
UNTIL ODD(ok);

{
  Prompt for the file where the message should be written. If you do not
  give a file, the message will be displayed on your screen.
}
```

```
IF outfile.length = 0 THEN outfile:='TT:';

IF envelope
THEN
    BEGIN
    {
      File contains an envelope. Call SELECT to see if it is a service
      message or a user message.
     }
     ok:=mrif$select_disassemble(msgctx,mrif$k_envelope);
     check(ok);
     IF ok=mrif$_envelope
     THEN WRITELN(out,'START_ASSEMBLE(envelope);')
     ELSE WRITELN(out,'SERVICE-MESSAGE-ENVELOPE');
     END
ELSE
    BEGIN
    {
      File contains a select. Call SELECT to see if it is a service
      message or a user message.
    }
    ok:=mrif$select_disassemble(msgctx,mrif$k_content);
    check(ok);
    IF ok=mrif$_content
    THEN WRITELN(out,'START_ASSEMBLE(content);')
    ELSE WRITELN(out,'SERVICE-MESSAGE-CONTENT');
    END;
END;


PROCEDURE finish;

{
  Procedure to tidy up storage available for disassembling this message
  file.

  Implicit Input : msgctx - left unusable
                   out    - left closed.
}

BEGIN
writeln(out,'END_ASSEMBLE;');
ok:=mrif$end_disassemble(msgctx);
check(ok);
END;


PROCEDURE item_to_ascii( item : UNSIGNED;
                         VAR ascii_item : VARYING[a1] OF CHAR );

{
  Procedure to return the name of an element in the message, given its item
  code.

  INPUT : item - mrif$k_item
  OUTPUT : ascii_item - readable description of mrif$k_item
}

BEGIN
     CASE INT(item) OF    { item is unsigned, constants are integers }

mrif$k_amdname:ascii_item:='amdname';
mrif$k_app_message_id:ascii_item:='app_message_id';
mrif$k_arrivaldate:ascii_item:='arrivaldate';
mrif$k_author:ascii_item:='author';
mrif$k_autoforward:ascii_item:='autoforward';
mrif$k_bcc:ascii_item:='bcc';
mrif$k_cc:ascii_item:='cc';
mrif$k_contenttypes:ascii_item:='contenttypes';
mrif$k_country:ascii_item:='country';
mrif$k_date:ascii_item:='date';
mrif$k_ddif:ascii_item:='ddif';
mrif$k_deferred:ascii_item:='deferred';
mrif$k_deliverdate:ascii_item:='deliverdate';
mrif$k_diagnostic:ascii_item:='diagnostic';
mrif$k_encodedtypes:ascii_item:='encodedtypes';
mrif$k_enddate:ascii_item:='enddate';
mrif$k_explicconv:ascii_item:='explicconv';
mrif$k_extensionid:ascii_item:='extensionid';
mrif$k_forwarded:ascii_item:='forwarded';
mrif$k_freeform:ascii_item:='freeform';
mrif$k_from:ascii_item:='from';
mrif$k_g3fax:ascii_item:='g3fax';
mrif$k_generation:ascii_item:='generation';
```

```
mrif$k_generated:ascii_item:='generated';
mrif$k_givenname:ascii_item:='givenname';
mrif$k_hopcount:ascii_item:='hopcount';
mrif$k_ia5text:ascii_item:='ia5text';
mrif$k_info:ascii_item:='info';
mrif$k_initials:ascii_item:='initials';
mrif$k_inreplyto:ascii_item:='inreplyto';
mrif$k_listname:ascii_item:='listname';
mrif$k_location:ascii_item:='location';
mrif$k_message_id:ascii_item:='message_id';
mrif$k_msgclass:ascii_item:='msgclass';
mrif$k_obsoletes:ascii_item:='obsoletes';
mrif$k_odif:ascii_item:='odif';
mrif$k_orgname:ascii_item:='orgname';
mrif$k_orgunit:ascii_item:='orgunit';
mrif$k_pdate:ascii_item:='pdate';
mrif$k_perlistflg:ascii_item:='perlistflg';
mrif$k_permsgflg:ascii_item:='permsgflg';
mrif$k_perrecflg:ascii_item:='perrecflg';
mrif$k_prdname:ascii_item:='prdname';
mrif$k_precedence:ascii_item:='precedence';
mrif$k_reason:ascii_item:='reason';
mrif$k_references:ascii_item:='references';
mrif$k_replyby:ascii_item:='replyby';
mrif$k_replyreq:ascii_item:='replyreq';
mrif$k_replytousers:ascii_item:='replytousers';
mrif$k_reportflg:ascii_item:='reportflg';
mrif$k_rmsfile:ascii_item:='rmsfile';
mrif$k_route:ascii_item:='route';
mrif$k_sender:ascii_item:='sender';
mrif$k_sensitivity:ascii_item:='sensitivity';
mrif$k_sfd:ascii_item:='sfd';
mrif$k_subject:ascii_item:='subject';
mrif$k_surname:ascii_item:='surname';
mrif$k_telephone:ascii_item:='telephone';
mrif$k_teletex:ascii_item:='teletex';
mrif$k_telex:ascii_item:='telex';
mrif$k_terminalid:ascii_item:='terminalid';
mrif$k_text:ascii_item:='text_record';
mrif$k_tif0:ascii_item:='tif0';
mrif$k_tif1:ascii_item:='tif1';
mrif$k_to:ascii_item:='to';
mrif$k_uacontid:ascii_item:='uacontid';
mrif$k_uniqueuaid:ascii_item:='uniqueuaid';
mrif$k_userid:ascii_item:='userid';
mrif$k_vendor1:ascii_item:='vendor1';
mrif$k_vendor2:ascii_item:='vendor2';
mrif$k_vendor3:ascii_item:='vendor3';
mrif$k_vendor4:ascii_item:='vendor4';
mrif$k_vendor5:ascii_item:='vendor5';
mrif$k_vendor6:ascii_item:='vendor6';
mrif$k_vendor7:ascii_item:='vendor7';
mrif$k_vendor8:ascii_item:='vendor8';
mrif$k_vendor9:ascii_item:='vendor9';
mrif$k_vendor10:ascii_item:='vendor10';
mrif$k_vendor11:ascii_item:='vendor11';
mrif$k_vendor12:ascii_item:='vendor12';
mrif$k_vendor13:ascii_item:='vendor13';
mrif$k_vendor14:ascii_item:='vendor14';
mrif$k_vendor15:ascii_item:='vendor15';
mrif$k_vendor16:ascii_item:='vendor16';
mrif$k_videotex:ascii_item:='videotex';
mrif$k_voice:ascii_item:='voice';
mrif$k_voicenotif:ascii_item:='voicenotif';
mrif$k_wpsplus:ascii_item:='wpsplus';
mrif$k_x121address:ascii_item:='x121address';
OTHERWISE
     ascii_item:='UNKNOWN'
   END;

END;

PROCEDURE out_flags (spaces : str_type;item,bit_value : UNSIGNED);

{
  Procedure to output the details on the bits defined in a flags longword.

  Note the non-standard PASCAL command (UAND) used to do unsigned bit by
  bit comparaisons. It could be done instead using ODD and DIV, or by
  using SETs.

  INPUT : spaces - string containing number of spaces last output
          item - mrif$k_item of flag
          bit_value - flags value contained in mrif$k_item
```

```
  Implicit Input : out
  Implicit Output : out
}

BEGIN

    CASE INT(item) OF       { item is unsigned, constants are integers. }

mrif$k_perrecflg:
BEGIN
IF UAND(bit_value,mrif$m_action)>0
 THEN WRITELN(out,spaces,'action;');

IF UAND(bit_value,mrif$m_mr_basic)>0
 THEN writeln(out,spaces,'mr_basic;');

IF UAND(bit_value,mrif$m_mr_confirmed)>0
 THEN writeln(out,spaces,'mr_confirmed;');

IF UAND(bit_value,mrif$m_ua_basic)>0
 THEN writeln(out,spaces,'ua_basic;');

IF UAND(bit_value,mrif$m_ua_confirmed)>0
 THEN writeln(out,spaces,'ua_confirmed;');

END;

mrif$k_permsgflg:
BEGIN

IF UAND(bit_value,mrif$m_discloserec)>0
 THEN writeln(out,spaces,'discloserec;');

IF UAND(bit_value,mrif$m_convprohib)>0
 THEN writeln(out,spaces,'convprohib;');

IF UAND(bit_value,mrif$m_altrecip)>0
 THEN writeln(out,spaces,'altrecip;');

IF UAND(bit_value,mrif$m_contretreq)>0
 THEN writeln(out,spaces,'contretreq;');

END;

mrif$k_perlistflg:

BEGIN

IF UAND(bit_value,mrif$m_expandedlist)>0
 THEN writeln(out,spaces,'expandedlist;');

END;

mrif$k_reportflg:
BEGIN

IF UAND(bit_value,mrif$m_recnotif)>0
 THEN writeln(out,spaces,'recnotif;');

IF UAND(bit_value,mrif$m_nonrecnotif)>0
 THEN writeln(out,spaces,'nonrecnotif;');

IF UAND(bit_value,mrif$m_returnmsg)>0
 THEN writeln(out,spaces,'returnmsg;');

END;


mrif$k_encodedtypes:

BEGIN
IF UAND(bit_value,mrif$m_rmsfile)>0
 THEN writeln(out,spaces,'rmsfile;');

IF UAND(bit_value,mrif$m_wpsplus)>0
 THEN writeln(out,spaces,'wpsplus;');

IF UAND(bit_value,mrif$m_ddif)>0
 THEN writeln(out,spaces,'ddif;');

IF UAND(bit_value,mrif$m_voicenotif)>0
 THEN writeln(out,spaces,'voicenotif;');

IF UAND(bit_value,mrif$m_text)>0
```

```
 THEN writeln(out,spaces,'text;');

IF UAND(bit_value,mrif$m_odif)>0
 THEN writeln(out,spaces,'odif;');

IF UAND(bit_value,mrif$m_tif1)>0
 THEN writeln(out,spaces,'mrif$_m_tif1;');

IF UAND(bit_value,mrif$m_sfd)>0
 THEN writeln(out,spaces,'sfd;');

IF UAND(bit_value,mrif$m_voice)>0
 THEN writeln(out,spaces,'voice;');

IF UAND(bit_value,mrif$m_videotex)>0
 THEN writeln(out,spaces,'videotex;');

IF UAND(bit_value,mrif$m_teletex)>0
 THEN writeln(out,spaces,'teletex;');

IF UAND(bit_value,mrif$m_tif0)>0
 THEN writeln(out,spaces,'tif0;');

IF UAND(bit_value,mrif$m_g3fax)>0
 THEN writeln(out,spaces,'g3fax;');

IF UAND(bit_value,mrif$m_ia5text)>0
 THEN writeln(out,spaces,'ia5text;');

IF UAND(bit_value,mrif$m_telex)>0
 THEN writeln(out,spaces,'telex;');

IF UAND(bit_value,mrif$m_undefined)>0
 THEN writeln(out,spaces,'undefined;');

END;

OTHERWISE
              { Bitstring is not a recognized flags longword }
     writeln(out,spaces,' UNKNOWN bitstring found');
END;
END;

PROCEDURE nbsdis(indent:INTEGER);

{
  Recursive procedure to extract the NBS data from the message and to
  output details about it.

  INPUT : indent - number of spaces used is indent*3
  Implicit Input : msgctx,out
  Implicit Output : msgctx,out
}

VAR
   spaces:str_type;      { declared local so there for nested nbsdis calls }
   space_cnt:INTEGER;    { used in FOR loop in comparison with indent }

BEGIN

   spaces:='';
   FOR space_cnt:=1 TO indent DO
        spaces:=spaces+'    ';

   WHILE ODD(mrif$start_nbs_get(msgctx,nbstype,nbsqual)) DO
     BEGIN
     writeln(out,spaces,'NBS: %X',HEX(nbstype,8),' %X',HEX(nbsqual,8));
     IF mrif$isprim(nbstype)
     THEN
          WHILE ODD(mrif$get_nbs_record(msgctx,string_value))
          DO WRITELN(out,spaces,' DATA: ',string_value)
     ELSE nbsdis(indent+1);
     mrif$end_nbs_get(msgctx)
     END;
END;


PROCEDURE disass(indent:INTEGER);

{
  Recursive procedure to find the next element in the message and to output
  details about it.

  INPUT : indent - number of spaces is ident*3
```

```
  Implicit Input : msgctx,out
  Implicit Output : msgctx,out
}

VAR
   spaces : str_type;       { declared local so there for all disass calls }
   space_cnt : INTEGER;     { used in FOR loop in comparison with ident }

BEGIN

   class:=mrif$k_any;
   item:=mrif$k_any;

spaces:='';
FOR space_cnt:=1 TO indent DO
    spaces:=spaces+'    ';
WHILE ODD(mrif$find(msgctx,class,item,mrif$k_range_cont,find_length)) DO
BEGIN
        item_to_ascii(item,ascii_item);

CASE INT(class) OF

mrif$k_prim_integer:
BEGIN
{
 This routine assumes an integer length of 4 bytes and uses 16 bytes only
 if the integer overflows. It is possible to use FIND_LENGTH to determine
 the actual length of the string
}
IF ok=mrif$_overflow
THEN
   BEGIN
   int_length:=16;
   ok:=mrif$get_integer(msgctx,item,integer_value,int_length);
   END;
{check(ok);}
WRITE(out,spaces,ascii_item,' := ');
FOR length_cnt:=int_length DOWNTO 1 DO
        if (integer_value[length_cnt]<>0)
                then WRITE(out,integer_value[length_cnt]:2);
WRITELN(out,';');
END;

mrif$k_prim_string:
BEGIN
{
  Code to handle truncated strings.
}
ok:=mrif$get_string(msgctx,item,string_value);
WRITE(out,spaces,ascii_item,' := ',string_value);
if (ok<>mrif$_truncate) then WRITELN(out,';');
WHILE ok=mrif$_truncate
DO
BEGIN
   ok:=mrif$get_string(msgctx,item,string_value);
   WRITE(out,spaces,string_value);
   if (ok<>mrif$_truncate) then WRITE(out,';');
   WRITELN;
END;

check(ok);

END;

mrif$k_prim_date:

BEGIN
{
  Convert a date to a readable format.
}
mrif$get_date(msgctx,item,date_value);
$asctim(date_length,ascii_date,date_value);
WRITELN(out,spaces,ascii_item,' := ',ascii_date,';');
END;

mrif$k_prim_flags:
BEGIN
mrif$get_flags(msgctx,item,bit_value);
if (bit_value<>0)
then BEGIN
        WRITELN(out,spaces,'FLAG(',ascii_item,');');
        {
        Call procedure to look at the bits in the flags longword.
        }
```

```
        out_flags(spaces,item,bit_value);
        WRITELN(out,spaces,'END_FLAG;');
      END;

END;

mrif$k_name,mrif$k_list,mrif$k_report :
BEGIN
WRITE(out,spaces,'START(');
CASE INT(class) OF
        mrif$k_name   : WRITE(out,'name');
        mrif$k_list   : WRITE(out,'list');
        mrif$k_report : WRITE(out,'report');
        END;
WRITELN(out,',',ascii_item,');');
mrif$start_get(msgctx,class,item);
disass(indent+1);
mrif$end_get(msgctx);
WRITELN(out,spaces,'END;');
END;

mrif$k_bodypart:
BEGIN
WRITELN(out,spaces,'START(bodypart,',ascii_item,');');
mrif$start_get(msgctx,class,item);
{
  Determine bodypart item and call appropriate routine.
}
IF item=mrif$k_text
THEN
   WHILE ODD(mrif$get_text_record(msgctx,string_value,mrif$m_crlf))
   DO WRITELN(out,string_value)
ELSE
   IF item=mrif$k_forwarded
      THEN disass(indent+1)
      ELSE nbsdis(indent+1);
mrif$end_get(msgctx);
WRITELN(out,spaces,'END;');
END;

OTHERWISE;
END;

class:=mrif$k_any;
item:=mrif$k_any;

END;
END;



{
   MAIN ROUTINE.
   =============
}

BEGIN

 {

 }

WRITELN('This program disassembles messages from a M.R. file.');
writeln;

REPEAT
        write('Input file  > ');
        readln(inpfile);
        if (length(inpfile)=0) then $EXIT;
        if (inpfile='?') then mrxtester$get_help(mrxtester$k_disinput);
UNTIL (inpfile<>'?');

REPEAT
        write('Output file > ');
        readln(outfile);
        if (length(outfile)=0) then outfile:='tt:';
        if (outfile='?') then mrxtester$get_help(mrxtester$k_disoutput);
UNTIL (outfile<>'?');

open(out,outfile);
rewrite(out);

{ This section disassembles the ENVELOPE and CONTENT parts. }
```

```
init(inpfile);
disass(1);
finish;

close(out);

END.
```

MRXTESTER$ASSEMBLE.PAS

```
[INHERIT('d18$:[mrmanager]MRXTESTER$DEFS.PEN','d18$:[mrmanager]MRIFDEFS.PEN','d18$:[mrmanager]STARLET.PEN')]
PROGRAM mrxtester$assemble(INPUT,OUTPUT);

CONST
   text_record = 1000;
   text_file   = 1001;


{ Declare data types that are not standard Pascal }
TYPE
   str_type              = VARYING[256] OF CHAR;
   small_byte            = [BYTE] 1..16;
   $QUAD                 = [QUAD,UNSAFE] RECORD
                            LO:UNSIGNED; L1:INTEGER; END;
   $UBYTE                = [BYTE] 0..255;
   $UWORD                = [WORD] 0..65535;

{ Declare variables }
VAR
   inp,                              { identifies the input file }
   out                               { identifies the output file }
         : TEXT;
   msgctx,                           { assemble context }
   ok,                               { general status variable containing
                                       value returned }
   class,                            { mrif$k_class }
   find_length,                      { length returned by find call }
   bit_value,                        { value of bit returned by GET_FLAGS }
   nbsqual,                          { qualifier returned by START_NBS_GET }
   nbstype,                          { type returned by START_NBS_GET }
   mrif$m_flag,
   flag:
            UNSIGNED;

   ascii_item: VARYING [20] OF CHAR; { ASCII version of mrif$k_item }
   resfile,                           { compilation results file }
   envfile,                           { file containing the envelope }
   cntfile,                           { file contining the content }
   value,                             { value of a message element }
   string,
   item_str,                          { the string value of a message item }
   class_str    :str_type;            { the string value of a message class }
   int_length,                       { length returned by GET_INTEGER call }
   length_cnt:                       { used as a FOR variable comparing to
                                       int_length }
            $UWORD;
   integer_value,
   buf          :ARRAY[1..16]OF $UBYTE; { value of integer returned by
                                          GET_INTEGER }
   string_value:VARYING[60]OF CHAR; { value of string returned by
                                      GET_STRING }
   date_value:$QUAD;                 { value of date returned by GET_DATE }
   ascii_date:PACKED ARRAY[1..26] OF CHAR;
                                     { ASCII version of date_value }
   date_length:$UWORD;                { length returned by $ASCTIM }

   complete,
   envelope,
   content,                         { boolean elements to identifies the message }
   on_flag,                         { part being handled }
   on_string    :BOOLEAN;
   outfile,
   inpfile             :str_type;
   item,                            { position of an item in the line }
   fl,                              { position of 'FLAG' in the line }
   sa,                              { position of 'START_ASSEMBLE' in the line }
   st,                              { position of 'START' in the line }
   ea,                              { position of 'END_ASSEMBLE' in the line }
   en,                              { position of 'END' in the line }
   ef,                              { position of 'END_FLAG' in the line }
   pg,                              { position of '(' in the line }
   pd,                              { position of ')' in the line }
   pv,                              { position of ';' in the line }
   vi,                              { position of ',' in the line }
   pe,                              { position of ':=' in the line }
   line,                            { line counter }
   flag_item                        { the MRIF code of the flag being handled }
            :INTEGER;

{  Declaration for Message Router Programmers Kit Routines.
   These are non-standard PASCAL declarations that use extensions to
   Pascal and are used in a similar way to the Run Time Library. The
```

```
   Interface Routines follow the VMS Procedure Calling Standard.
}
[EXTERNAL]
   FUNCTION mrif$start_assemble(VAR msgctxid : UNSIGNED;
                                    flag : UNSIGNED := %IMMED 0)
                                :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$select_assemble(msgctxid : UNSIGNED;
                                 msgtype  : UNSIGNED := %IMMED 0)
                                :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$end_assemble(msgctxid : UNSIGNED;
                              %DESCR contents:VARYING[a1] OF CHAR:=%IMMED 0;
                              %DESCR envelope:VARYING[b1] OF CHAR:=%IMMED 0)
                                :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$start_put(msgctxid : UNSIGNED;
                           class    : UNSIGNED;
                           item     : UNSIGNED) :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$end_put(msgctxid : UNSIGNED) :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_string(msgctxid : UNSIGNED;
                            item     : UNSIGNED;
                            %DESCR string : VARYING[a1] OF CHAR)
                                :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_integer(msgctxid : UNSIGNED;
                             item     : UNSIGNED;
                             %REF number : ARRAY[u1..u2:INTEGER] OF $UBYTE;
                             size : $UWORD := %IMMED 0)
                                :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_flags(msgctxid : UNSIGNED;
                           item     : UNSIGNED;
                           flags    : UNSIGNED) :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_date(msgctxid : UNSIGNED;
                          item     : UNSIGNED;
                          date     : $QUAD) :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_text_record(msgctxid : UNSIGNED;
                         %DESCR string : VARYING[a1] OF CHAR;
                         flags : UNSIGNED := %IMMED 0) :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_text_file(msgctxid : UNSIGNED;
                         %DESCR string : VARYING[a1] OF CHAR)
                                :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$start_nbs_put(msgctxid : UNSIGNED;
                               nbstype  : UNSIGNED;
                               nbsqual  : UNSIGNED) :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$end_nbs_put(msgctxid : UNSIGNED) :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$put_nbs_record(msgctxid : UNSIGNED;
                                %DESCR string : VARYING[a1] OF CHAR)
                                       :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$connect(VAR lnkctx : UNSIGNED;
            %DESCR node : VARYING[a1] OF CHAR := %IMMED 0)
                                :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$disconnect(lnkctx : UNSIGNED) :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$identify(lnkctx:UNSIGNED;
            %DESCR mailbox : VARYING[a1] OF CHAR;
            %DESCR password : VARYING[b1] OF CHAR := %IMMED 0)
```

```
                              :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$start_disassemble(VAR msgctxid : UNSIGNED;
            %DESCR contents : VARYING[a1] OF CHAR := %IMMED 0;
            %DESCR envelope : VARYING[b1] OF CHAR := %IMMED 0)
                              :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$select_disassemble(msgctxid:UNSIGNED;
                               msgtype:UNSIGNED:=%IMMED 0)
                              :UNSIGNED;EXTERNAL;
[EXTERNAL]
   FUNCTION mrif$end_disassemble(msgctxid:UNSIGNED)
                              :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$find(msgctxid : UNSIGNED;
                      VAR class : UNSIGNED;
                      VAR item  : UNSIGNED;
                      range : UNSIGNED;
                      VAR length : UNSIGNED)
                          :UNSIGNED;EXTERNAL;
[EXTERNAL]
   FUNCTION mrif$start_get(msgctxid:UNSIGNED;
                      VAR class:UNSIGNED;
                      VAR item:UNSIGNED)
                              :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$end_get(msgctxid:UNSIGNED):UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$get_string(msgctxid:UNSIGNED;
                            item:UNSIGNED;
                            %DESCR value:VARYING[a1] OF CHAR;
                            VAR length : $UWORD := %IMMED 0)
                            :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$get_integer(msgctxid:UNSIGNED;
                             item:UNSIGNED;
                             %REF number:ARRAY[u1..u2:INTEGER] OF $UBYTE;
                             VAR length:$UWORD:=%IMMED 0)
                             :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$get_flags(msgctxid:UNSIGNED;
                           item:UNSIGNED;
                           VAR value :UNSIGNED)
                          :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$get_date(msgctxid : UNSIGNED;
                          item:UNSIGNED;
                          VAR value :$QUAD)
                          :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$get_text_record(msgctxid:UNSIGNED;
                          %DESCR value:VARYING[a1] OF CHAR;
                          options:UNSIGNED:=%IMMED 0;
                          VAR length : $UWORD := %IMMED 0)
                         :UNSIGNED;EXTERNAL;


[EXTERNAL]
   FUNCTION mrif$start_nbs_get(msgctxid:UNSIGNED;
                           VAR nbstype:UNSIGNED;
                           VAR nbsqual:UNSIGNED) :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$end_nbs_get(msgctxid:UNSIGNED):UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$get_nbs_record(msgctxid:UNSIGNED;
                              %DESCR value:VARYING[a1] OF CHAR;
                              VAR length : $UWORD := %IMMED 0)
                             :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$isprim(nbstype : UNSIGNED):BOOLEAN;EXTERNAL;

[EXTERNAL]
```

```
   FUNCTION mrif$fetch(lnkctx:UNSIGNED;
        %DESCR content:VARYING[a1] OF CHAR;
        %DESCR envelope:VARYING[b1] OF CHAR := %IMMED 0;
        flags:UNSIGNED := %IMMED 0)
           :UNSIGNED;EXTERNAL;

[EXTERNAL]
   FUNCTION mrif$query(lnkctx : UNSIGNED;
                       VAR number : UNSIGNED)
           :UNSIGNED;EXTERNAL;

{ Declare Run Time Library that wil be needed. }

[EXTERNAL]FUNCTION LIB$STOP(%IMMED sts:UNSIGNED):UNSIGNED;EXTERNAL;

[EXTERNAL]FUNCTION LIB$SIGNAL(%IMMED sts:UNSIGNED):UNSIGNED;EXTERNAL;

{ Declare help library (cfr MRXTESTER$BUILD.PAS) }

[EXTERNAL]PROCEDURE mrxtester$get_help(item:INTEGER);EXTERNAL;

PROCEDURE check (status_code : UNSIGNED);

{
  Procedure to determine whether the status returned by another routine
  was a VMS success or error status. If the routine returned an error,
  the program doesn't terminates, but indicates which MRIF error has occured.

  INPUT : status_code - a 32 bit VMS status returned by another routine
  Implicit Input : msgctx,out
}

BEGIN
IF NOT ODD(status_code)          { Bottom bit indicates success }
THEN
   BEGIN

writeln(out,'%MRXTESTER-E-MRIFERR, Error returned by MRIF at line',line);
        CASE int(status_code) OF
mrif$_confused : writeln(out,'MRIF$_CONFUSED, Internal Error.');
mrif$_badnbsfile:writeln(out,'MRIF$_BADNBSFILE, NBS File Has Invalid Syntax.');
mrif$_filnotacc:writeln(out,'MRIF$_FILNOTACC, Cannot Access File.');
mrif$_invhere:writeln(out,'MRIF$_INVHERE, Routine Is Invalid In This State.');
mrif$_invmsgctx:writeln(out,'MRIF$_INVMSGCTX, Invalid Message Context Identifier.');
mrif$_invitem:
writeln(out,'MRIF$_INVITEM, Invalid Item Code, Class Or Combination Of Item Code And Class');
mrif$_invrange:writeln(out,'MRIF$_INVRANGE, Invalid Range Specified.');
mrif$_nomore:writeln(out,'MRIF$_NOMORE, No More Elements Of This Item Code And Class Remain.');
mrif$_processed:writeln(out,'MRIF$_PROCESSED, Current Element Is Already Fully Processed.');
mrif$_wrongpos:writeln(out,'MRIF$_WRONGPOS, Not Positionned On Specific Item.');
mrif$_outrange:writeln(out,'MRIF$_OUTRANGE, Integer Is Out Of Valid Range.');
mrif$_overflow:writeln(out,'MRIF$_OVERFLOW, Integer Does Not Fit Into The Specified Buffer.');
mrif$_parprocess:writeln(out,'MRIF$_PARPROCESS, Call Is Invalid When Element Is Partly Processed.');
mrif$_corrupt:writeln(out,'MRIF$_CORRUPT, Message Cannot Be Parsed.');
mrif$_truncate:writeln(out,'MRIF$_TRUNCATE, Element Too Big For Buffer, Element Partly Processed.');
mrif$_invarg:writeln(out,'MRIF$_INVARG, Flags Longword Has One Or More Undefined Bits Set.');
{mrif$_eot:writeln(out,'MRIF$_EOT, No Further Records.');
lib$_insvirmen:writeln(out,'LIB$_INSVIRMEM, Insufficient Virtual Memory.');}
mrif$_baddate:writeln(out,'MRIF$_BADDATE, Date Is Invalid.');
mrif$_duplicate:writeln(out,'MRIF$_DUPLICATE, Element Already Added To Message And Duplicate Not Allowed.');
mrif$_invmsgtyp:writeln(out,'MRIF$_INVMSGTYP, Element Is Not Allowed On This Message Part.');
mrif$_invfft:writeln(out,'MRIF$_INVFFT, File Format Is Invalid.');
mrif$_strnull:writeln(out,'MRIF$_STRNULL, String Length Is Less Than One.');
mrif$_invselection:writeln(out,'MRIF$_INVSELECTION, Invalid Message Part Selected.');
mrif$_selected:writeln(out,'MRIF$_SELECTED, Message Part Already Selected.');
mrif$_paramno:writeln(out,'MRIF$_PARAMNO, Too Few Parameters Specified.');
    OTHERWISE
writeln(out,'MRIF$_UNDEFINED, Undefined MRIF error code :',status_code,'.');
     END;
   END;
END;

FUNCTION str_to_int(s : str_type):INTEGER;

{ Function to translate a char string to an integer. }

VAR n,
    i   : INTEGER;
    c   : CHAR;
    ch  : str_type;

BEGIN

n:=0;
```

```
FOR i:=1 TO length(s) DO
        BEGIN
        ch:=substr(s,i,1);
        readv(ch,c);
        n:=n * 10 + (ORD(c) - ORD('0'));
        END;
str_to_int:=n;

END;

FUNCTION conv(s:str_type):INTEGER;

{
  Function to return the mrif code of an element in the message,
  given its name.

  INPUT : s - readable description of mrif$k_item
  OUTPUT : item - mrif$k_item
}

VAR item : INTEGER;

BEGIN

item:=0;

if index(s,'amdname') <> 0 then item:=mrif$k_amdname
else if index(s,'app_message_id') <> 0 then item:=mrif$k_app_message_id
else if index(s,'arrivaldate') <> 0 then item:=mrif$k_arrivaldate
else if index(s,'author') <> 0 then item:=mrif$k_author
else if index(s,'autoforward') <> 0 then item:=mrif$k_autoforward
else if index(s,'bcc') <> 0 then item:=mrif$k_bcc
else if index(s,'bodypart') <> 0 then item:=mrif$k_bodypart
else if index(s,'cc') <> 0 then item:=mrif$k_cc
else if index(s,'complete_message') <> 0 then item:=mrif$k_complete_message
else if index(s,'contenttypes') <> 0 then item:=mrif$k_contenttypes
else if index(s,'content') <> 0 then item:=mrif$k_content
else if index(s,'enddate') <> 0 then item:=mrif$k_enddate
else if index(s,'country') <> 0 then item:=mrif$k_country
else if index(s,'date') <> 0 then item:=mrif$k_date
else if index(s,'ddif') <> 0 then item:=mrif$k_ddif
else if index(s,'deferred') <> 0 then item:=mrif$k_deferred
else if index(s,'deliverdate') <> 0 then item:=mrif$k_deliverdate
else if index(s,'diagnostic') <> 0 then item:=mrif$k_diagnostic
else if index(s,'encodedtypes') <> 0 then item:=mrif$k_encodedtypes
else if index(s,'enddate') <> 0 then item:=mrif$k_enddate
else if index(s,'envelope') <> 0 then item:=mrif$k_envelope
else if index(s,'explicconv') <> 0 then item:=mrif$k_explicconv
else if index(s,'extensionid') <> 0 then item:=mrif$k_extensionid
else if index(s,'forwarded') <> 0 then item:=mrif$k_forwarded
else if index(s,'freeform') <> 0 then item:=mrif$k_freeform
else if index(s,'from') <> 0 then item:=mrif$k_from
else if index(s,'g3fax') <> 0 then item:=mrif$k_g3fax
else if index(s,'generation') <> 0 then item:=mrif$k_generation
else if index(s,'generated') <> 0 then item:=mrif$k_generated
else if index(s,'givenname') <> 0 then item:=mrif$k_givenname
else if index(s,'hopcount') <> 0 then item:=mrif$k_hopcount
else if index(s,'ia5text') <> 0 then item:=mrif$k_ia5text
else if index(s,'info') <> 0 then item:=mrif$k_info
else if index(s,'initials') <> 0 then item:=mrif$k_initials
else if index(s,'inreplyto') <> 0 then item:=mrif$k_inreplyto
else if index(s,'listname') <> 0 then item:=mrif$k_listname
else if index(s,'list') <> 0 then item:=mrif$k_list
else if index(s,'location') <> 0 then item:=mrif$k_location
else if index(s,'message_id') <> 0 then item:=mrif$k_message_id
else if index(s,'msgclass') <> 0 then item:=mrif$k_msgclass
else if index(s,'name') <> 0 then item:=mrif$k_name
else if index(s,'obsoletes') <> 0 then item:=mrif$k_obsoletes
else if index(s,'odif') <> 0 then item:=mrif$k_odif
else if index(s,'orgname') <> 0 then item:=mrif$k_orgname
else if index(s,'orgunit') <> 0 then item:=mrif$k_orgunit
else if index(s,'pdate') <> 0 then item:=mrif$k_pdate
else if index(s,'perlistflg') <> 0 then item:=mrif$k_perlistflg
else if index(s,'permsgflg') <> 0 then item:=mrif$k_permsgflg
else if index(s,'perrecflg') <> 0 then item:=mrif$k_perrecflg
else if index(s,'prdname') <> 0 then item:=mrif$k_prdname
else if index(s,'precedence') <> 0 then item:=mrif$k_precedence
else if index(s,'reason') <> 0 then item:=mrif$k_reason
else if index(s,'references') <> 0 then item:=mrif$k_references
else if index(s,'replyby') <> 0 then item:=mrif$k_replyby
else if index(s,'replyreq') <> 0 then item:=mrif$k_replyreq
else if index(s,'replytousers') <> 0 then item:=mrif$k_replytousers
else if index(s,'reportflg') <> 0 then item:=mrif$k_reportflg
else if index(s,'rmsfile') <> 0 then item:=mrif$k_rmsfile
```

```
else if index(s,'route') <> 0 then item:=mrif$k_route
else if index(s,'sender') <> 0 then item:=mrif$k_sender
else if index(s,'sensitivity') <> 0 then item:=mrif$k_sensitivity
else if index(s,'sfd') <> 0 then item:=mrif$k_sfd
else if index(s,'subject') <> 0 then item:=mrif$k_subject
else if index(s,'surname') <> 0 then item:=mrif$k_surname
else if index(s,'telephone') <> 0 then item:=mrif$k_telephone
else if index(s,'teletex') <> 0 then item:=mrif$k_teletex
else if index(s,'telex') <> 0 then item:=mrif$k_telex
else if index(s,'terminalid') <> 0 then item:=mrif$k_terminalid
else if index(s,'text') <> 0 then item:=mrif$k_text
else if index(s,'tif0') <> 0 then item:=mrif$k_tif0
else if index(s,'tif1') <> 0 then item:=mrif$k_tif1
else if index(s,'to') <> 0 then item:=mrif$k_to
else if index(s,'uacontid') <> 0 then item:=mrif$k_uacontid
else if index(s,'uniqueuaid') <> 0 then item:=mrif$k_uniqueuaid
else if index(s,'userid') <> 0 then item:=mrif$k_userid
else if index(s,'vendor1') <> 0 then item:=mrif$k_vendor1
else if index(s,'vendor2') <> 0 then item:=mrif$k_vendor2
else if index(s,'vendor3') <> 0 then item:=mrif$k_vendor3
else if index(s,'vendor4') <> 0 then item:=mrif$k_vendor4
else if index(s,'vendor5') <> 0 then item:=mrif$k_vendor5
else if index(s,'vendor6') <> 0 then item:=mrif$k_vendor6
else if index(s,'vendor7') <> 0 then item:=mrif$k_vendor7
else if index(s,'vendor8') <> 0 then item:=mrif$k_vendor8
else if index(s,'vendor9') <> 0 then item:=mrif$k_vendor9
else if index(s,'vendor10') <> 0 then item:=mrif$k_vendor10
else if index(s,'vendor11') <> 0 then item:=mrif$k_vendor11
else if index(s,'vendor12') <> 0 then item:=mrif$k_vendor12
else if index(s,'vendor13') <> 0 then item:=mrif$k_vendor13
else if index(s,'vendor14') <> 0 then item:=mrif$k_vendor14
else if index(s,'vendor15') <> 0 then item:=mrif$k_vendor15
else if index(s,'vendor16') <> 0 then item:=mrif$k_vendor16
else if index(s,'videotex') <> 0 then item:=mrif$k_videotex
else if index(s,'voice') <> 0 then item:=mrif$k_voice
else if index(s,'voicenotif') <> 0 then item:=mrif$k_voicenotif
else if index(s,'wpsplus') <> 0 then item:=mrif$k_wpsplus
else if index(s,'x121address') <> 0 then item:=mrif$k_x121address
else if index(s,'text_record') <> 0 then item:=text_record
else if index(s,'text_file') <> 0 then item:=text_file;
conv:=item;

END;

FUNCTION conv_flag(s:str_type):UNSIGNED;

{
  Function to return the MRIF code of an flag element in the message,
  given its name.

  INPUT : s - readable description of mrif$m_item
  OUTPUT : flag item - mrif$m_item
}

VAR u : UNSIGNED;

BEGIN

if index(s,'action') <> 0 then u:=mrif$m_action
else if index(s,'mr_basic') <> 0 then u:=mrif$m_mr_basic
else if index(s,'mr_confirmed') <> 0 then u:=mrif$m_mr_confirmed
else if index(s,'ua_basic') <> 0 then u:=mrif$m_ua_basic
else if index(s,'ua_confirmed') <> 0 then u:=mrif$m_ua_confirmed
else if index(s,'discloserec') <> 0 then u:=mrif$m_discloserec
else if index(s,'convprohib') <> 0 then u:=mrif$m_convprohib
else if index(s,'altrecip') <> 0 then u:=mrif$m_altrecip
else if index(s,'contretreq') <> 0 then u:=mrif$m_contretreq
else if index(s,'expandedlist') <> 0 then u:=mrif$m_expandedlist
else if index(s,'recnotif') <> 0 then u:=mrif$m_recnotif
else if index(s,'nonrecnotif') <> 0 then u:=mrif$m_nonrecnotif
else if index(s,'returnmsg') <> 0 then u:=mrif$m_returnmsg
else if index(s,'rmsfile') <> 0 then u:=mrif$m_rmsfile
else if index(s,'wpsplus') <> 0 then u:=mrif$m_wpsplus
else if index(s,'ddif') <> 0 then u:=mrif$m_ddif
else if index(s,'voicenotif') <> 0 then u:=mrif$m_voicenotif
else if index(s,'text') <> 0 then u:=mrif$m_text
else if index(s,'odif') <> 0 then u:=mrif$m_odif
else if index(s,'tif0') <> 0 then u:=mrif$m_tif0
else if index(s,'tif1') <> 0 then u:=mrif$m_tif1
else if index(s,'sfd') <> 0 then u:=mrif$m_sfd
else if index(s,'voice') <> 0 then u:=mrif$m_voice
else if index(s,'videotex') <> 0 then u:=mrif$m_videotex
else if index(s,'teletex') <> 0 then u:=mrif$m_teletex
else if index(s,'g3fax') <> 0 then u:=mrif$m_g3fax
```

```
else if index(s,'ia5text') <> 0 then u:=mrif$m_ia5text
else if index(s,'telex') <> 0 then u:=mrif$m_telex
else if index(s,'undefined') <> 0 then u:=mrif$m_undefined;

conv_flag := u;

END;


PROCEDURE init;

{
  Procedure to search for MRXTESTER message description language keywords
  in the current line.

  Implicit Input : string, the current line handled by the message assembler
  Output : the place of the MRXTESTER mdl keywords in the line (if any).
}

BEGIN

sa := index(string,'start_assemble');
if (sa=0) then sa:=index(string,'START_ASSEMBLE');

st := index(string,'start');
if (st=0) then st:=index(string,'START');
if(sa<>0) then st:=0;

ef := index(string,'end_flag');
if (ef=0) then ef:=index(string,'END_FLAG');

fl := index(string,'flag');
if (fl=0) then fl:=index(string,'FLAG');
if(ef<>0) then fl:=0;

ea := index(string,'end_assemble');
if (ea=0) then ea:=index(string,'END_ASSEMBLE');

en := index(string,'end');
if (en=0) then en:=index(string,'END');
if ((ea<>0) or (ef<>0)) then en:=0;

pg := index(string,'(');
pd := index(string,')');
vi := index(string,',');
pv := index(string,';');
pe := index(string,':=');

END;

PROCEDURE put(item : INTEGER; value : str_type);

{
  Procedure to add the specified message item and value to the message.

  Input : item - the MRIF code of the message item
          value - the value to be added
  Implicit Input : msgctx, the message context identifier
}

BEGIN

CASE item OF

mrif$k_hopcount,
mrif$k_contenttypes,
mrif$k_precedence,
mrif$k_extensionid,
mrif$k_explicconv,
mrif$k_autoforward,
mrif$k_sensitivity,
mrif$k_replyreq : BEGIN
                  buf[1]:=str_to_int(value);
                  ok:=mrif$put_integer(msgctx,item,buf);
                  check(ok);
                  END;
mrif$k_pdate,
mrif$k_deferred,
mrif$k_date,
mrif$k_replyby,
mrif$k_enddate :BEGIN
                $bintim(ascii_date,date_value);
                ok:=mrif$put_date(msgctx,item,date_value);
                check(ok);
```

```
                END;

mrif$k_message_id,
mrif$k_uacontid,
mrif$k_userid,
mrif$k_route,
mrif$k_x121address,
mrif$k_terminalid,
mrif$k_orgname,
mrif$k_orgunit,
mrif$k_surname,
mrif$k_givenname,
mrif$k_initials,
mrif$k_generation,
mrif$k_telephone,
mrif$k_location,
mrif$k_country,
mrif$k_amdname,
mrif$k_prdname,
mrif$k_uniqueuaid,
mrif$k_listname,
mrif$k_app_message_id,
mrif$k_subject,
mrif$k_inreplyto,
mrif$k_obsoletes,
mrif$k_references,
mrif$k_msgclass,
mrif$k_freeform         :BEGIN
                         ok:=mrif$put_string(msgctx,item,value);
                         check(ok);
                         END;
text_record             :BEGIN
                         ok:=mrif$put_text_record(msgctx,value);
                         check(ok);
                         END;
text_file               :BEGIN
                         ok:=mrif$put_text_file(msgctx,value);
                         check(ok);
                         END;
        END;

END;

PROCEDURE start_assemble;

{
  Procedure to start the assembly of the specified message part.
  Message part can be mrif$k_envelope, mrif$k_content OR
  mrif$k_complete_message.

}

BEGIN

if ((pg<pd) and (pd<pv) and ((sa+13)<pg) and (pe=0))
then BEGIN
     item_str:=substr(string,(pg+1),pd-(pg+1));
     item:=conv(item_str);

     ok:=mrif$select_assemble(msgctx,item);
     check(ok);
     complete:=false; envelope:=false; content:=false;
     if (item=mrif$k_complete_message) then complete:=true
     else if (item=mrif$k_envelope) then envelope:=true
     else if (item=mrif$k_content) then content:=true;
     END
else writeln('Syntax error at line :',line,' in ',inpfile);

END;


PROCEDURE flags;

{ Procedure to start the handling of a flag element.
}

BEGIN

if (((fl+3)<pg) and (pg<pd) and (pd<pv) and (pe=0))
then BEGIN
     item_str:=substr(string,pg+1,pd-(pg+1));
     flag:=0;
     flag_item:=conv(item_str);
     on_flag:=true;
```

```
     END
else writeln('Syntax error at line :',line,' in ',inpfile);

END;

PROCEDURE start;

{ Procedure to start the handling of a composite message element.
  class can be : mrif$k_name,
                 mrif$k_list,
                 mrif$k_bodypart.
}

BEGIN

if (((st+4)<pg) and (pg<vi) and (vi<pd) and (pd<pv) and (pe=0))
then BEGIN
     item_str:=substr(string,vi+1,pd-(vi+1));
     class_str:=substr(string,pg+1,vi-(pg+1));
     ok:=mrif$start_put(msgctx,conv(class_str),conv(item_str));
     check(ok);
     END
else writeln('Syntax error at line :',line,' in ',inpfile);

END;

PROCEDURE end_assemble;

{
 Procedure to stop the assembly of the message part being handled.
}

BEGIN

if ((ea<pv) and (pe=0))
then BEGIN
     if complete then ok:=mrif$end_assemble(msgctx,cntfile,envfile)
     else if envelope then ok:=mrif$end_assemble(msgctx,,envfile)
     else if content then ok:=mrif$end_assemble(msgctx,cntfile)
     else writeln('Invalid context at line :',line,' in ',inpfile);
     check(ok);
     END
else writeln('Syntax error at line :',line,' in ',inpfile);

END;

PROCEDURE end_start;

{
  Procedure to stop the assembly of the specified composite element.
}

BEGIN

if ((en<pv) and (pe=0)) then BEGIN
                               ok:=mrif$end_put(msgctx);
                               check(ok);
                             END
else writeln('Syntax error at line :',line,' in ',inpfile);

END;

PROCEDURE end_flag;

{
 Procedure to stop the assembly of the flag element being handled and
 add it to the message.
}

BEGIN

if ((ef<pv) and (pe=0))
then BEGIN
     ok:=mrif$put_flags(msgctx,flag_item,flag);
     check(ok);
     on_flag:=false;
     END
else writeln('Syntax error at line :',line,' in ',inpfile);

END;


FUNCTION no_blank (s:str_type):str_type;
```

```
{ Function to remove the blanks from the beginning of a line }

VAR s1 : str_type;

BEGIN
       s1 := s;
       while (substr(s1,1,1) = ' ') do s1 := substr(s1,2,length(s1)-1);
       no_blank := s1;
END;

PROCEDURE mrif$k_item;

VAR a : INTEGER;

BEGIN

     string:=no_blank(string);
     pe := index(string,':=');
     pv := index(string,';');
     item_str:=substr(string,1,pe-1);
     item:=conv(item_str);
     value:='';
     if (pv<>0) then BEGIN
               if (pe<pv) then BEGIN
                         value:=substr(string,pe+2,pv-(pe+2));
                         put(item,value);
                              END
                         else writeln('Syntax error')
                    END
               else if on_string
                    then BEGIN
                       value:=substr(string,pe+2,length(string)-(pe+2));
                       REPEAT
                          readln(inp,string);
                          a:=index(string,';');
                          if (a<>0) then value:=value+substr(string,1,a-1)
                                    else value:=value+string;
                       UNTIL (a<>0);
                       put(item,value);
                       END
                    else writeln('Missing ;');

END;


PROCEDURE mrif$m_item;

VAR u : UNSIGNED;

BEGIN

string:=no_blank(string);
pv := index(string,';');
item_str:=substr(string,1,pv-1);
u:=conv_flag(item_str);

if (u = mrif$m_action) then flag:=UOR(flag,mrif$m_action);
if (u = mrif$m_mr_basic) then flag:=UOR(flag,mrif$m_mr_basic);
if (u = mrif$m_mr_confirmed) then flag:=UOR(flag,mrif$m_mr_confirmed);
if (u = mrif$m_ua_basic) then flag:=UOR(flag,mrif$m_ua_basic);
if (u = mrif$m_ua_confirmed) then flag:=UOR(flag,mrif$m_ua_confirmed);
if (u = mrif$m_discloserec) then flag:=UOR(flag,mrif$m_discloserec);
if (u = mrif$m_convprohib) then flag:=UOR(flag,mrif$m_convprohib);
if (u = mrif$m_altrecip) then flag:=UOR(flag,mrif$m_altrecip);
if (u = mrif$m_contretreq) then flag:=UOR(flag,mrif$m_contretreq);
if (u = mrif$m_expandedlist) then flag:=UOR(flag,mrif$m_expandedlist);
if (u = mrif$m_recnotif) then flag:=UOR(flag,mrif$m_recnotif);
if (u = mrif$m_nonrecnotif) then flag:=UOR(flag,mrif$m_nonrecnotif);
if (u = mrif$m_returnmsg) then flag:=UOR(flag,mrif$m_returnmsg);
if (u = mrif$m_rmsfile) then flag:=UOR(flag,mrif$m_rmsfile);
if (u = mrif$m_wpsplus) then flag:=UOR(flag,mrif$m_wpsplus);
if (u = mrif$m_ddif) then flag:=UOR(flag,mrif$m_ddif);
if (u = mrif$m_voicenotif) then flag:=UOR(flag,mrif$m_voicenotif);
if (u = mrif$m_text) then flag:=UOR(flag,mrif$m_text);
if (u = mrif$m_odif) then flag:=UOR(flag,mrif$m_odif);
if (u = mrif$m_tif0) then flag:=UOR(flag,mrif$m_tif0);
if (u = mrif$m_tif1) then flag:=UOR(flag,mrif$m_tif1);
if (u = mrif$m_sfd) then flag:=UOR(flag,mrif$m_sfd);
if (u = mrif$m_voice) then flag:=UOR(flag,mrif$m_voice);
if (u = mrif$m_videotex) then flag:=UOR(flag,mrif$m_videotex);
if (u = mrif$m_teletex) then flag:=UOR(flag,mrif$m_teletex);
if (u = mrif$m_g3fax) then flag:=UOR(flag,mrif$m_g3fax);
if (u = mrif$m_ia5text) then flag:=UOR(flag,mrif$m_ia5text);
if (u = mrif$m_telex) then flag:=UOR(flag,mrif$m_telex);
```

```
if (u = mrif$m_undefined) then flag:=UOR(flag,mrif$m_undefined);

END;

PROCEDURE analyse;

BEGIN

if (sa<>0) then start_assemble
else if (fl<>0) then flags
else if (st<>0) then start
else if (ea<>0) then end_assemble
else if (en<>0) then end_start
else if (ef<>0) then end_flag
else if (pe<>0) then mrif$k_item
else if (pv<>0) then mrif$m_item
else ;

END;


{ PROCEDURE Main }

BEGIN


REPEAT
        write('Input file             > ');
        readln(inpfile);
        if (length(inpfile)=0) then $EXIT(ss$_normal);
        if (inpfile='?') then mrxtester$get_help(mrxtester$k_assinput);
UNTIL (inpfile<>'?');

open(inp,inpfile,readonly);
reset(inp);

REPEAT
        write('Result file [optionna] > ');
        readln(resfile);
        if (resfile='?') then mrxtester$get_help(mrxtester$k_assresult);
        if (length(resfile)=0) then resfile:='TT:';
UNTIL (resfile<>'?');

open(out,resfile);
rewrite(out);

REPEAT
        write('Output message file    > ');
        readln(outfile);
        if (outfile='?') then mrxtester$get_help(mrxtester$k_assoutput);
UNTIL ((outfile<>'?') and (length(outfile)<>0));

envfile:=outfile+'$env';
cntfile:=outfile+'$cnt';

ok:=mrif$start_assemble(msgctx,0);
check(ok);

line:=1;
WHILE (not(eof(inp))) DO
        BEGIN
        readln(inp,string);
        writeln(out,line:4,': ',string);
        init;
        analyse;
        line:=line+1;
                END;

END.
```

ANNEXE 8.6 :

MANUEL D'UTILISATION

MRXTESTER USER GUIDE

V 1.0

Marc ELOY
Daniel REUVIAUX

****
*** ***
*** **MRXTESTER USER GUIDE** ***
*** ***
****

[1] INTRODUCTION

Ce rapport s'essaye à présenter le produit développé par Marc ELOY et Daniel REUVIAUX lors d'un stage à l'I.I.H.E. effectué du 17 Aout 1987 au 22 Janvier 1988.

Le produit MRXTESTER est un User Agent permettant de tester le Message Router X.400 GATEWAY (MR et MRX) de Digital.
Il comporte un certain nombre d'utilitaires qui permettent à l'utilisateur de créer, d'envoyer et de lire un message.
Grace à un générateur automatique de cas de tests d'O/R names, on peut éffectuer une série de tests d'adressage sur une destination donnée.
Le produit est pourvu d'un système de help "on-line" qui guide l'utilisateur pas-à-pas dans sa demarche.

Ce guide s'adresse à toute personne intéressée par le X.400 et les tests de conformité. Nous conseillons vivement aux utilisateurs potentiels de se référrer à la norme X.400 du C.C.I.T.T., ainsi qu'à la documentation Digital des produits concernés (voir annexe).

Enfin, ce produit ne constitue pas un User Agent "utilisateur", dans le sens que ce produit ne comporte pas toutes les facilités offertes par un U.A. (classement, édition de messages ...), et que, de plus, il permet la création de messages syntaxiquement faux (du point de vue de la norme X.400), ceci en vue de tester les réactions du système.

[2] VOCABULAIRE

Avant de commencer la présentation du produit MRXTESTER et de ses composantes, il convient de se définir un vocabulaire de base :

ASSEMBLER UN MESSAGE :
Rendre un ensemble de chaînes de caractères, d'entiers, de dates, de booléens dans un format tel qu'il soit compréhensible pour le Message Router.

DESASSEMBLER UN MESSAGE :
Rendre un message assemblé compréhensible pour l'utilisateur.

POSTER UN MESSAGE :
Donner un message assemblé au Message Router pour qu'il en éffectue la délivraison.

BOITE AUX LETTRES :
Moyen de communication entre l'utilisateur et le Message Router.
(Voir "Comment créer une boîte aux lettres")

LIRE UN MESSAGE :
Relever une boîte aux lettres, prendre un message de sa boîte aux lettres, le désassembler et le ranger dans un fichier. Il faut remarquer qu'à partir du moment où un message a été enlevé d'une boîte, le Message Router n'en est plus responsable, donc que ce message n'existera plus pour lui.

[3] SOFTWARE DISTRIBUTION LIST

MRXTESTER se compose d'une série de fichiers situés dans la directory :

DUA0:[MRMANAGER.MRXTESTER]

MRXTESTER.COM :
Procédure écrite en langage DCL permettant de gérer l'ensemble des produits. Cette procédure se lance en tapant @[MRMANAGER.MRXTESTER]MRXTESTER

MRXTESTER$HELP.HLP :
Procédure DCL permettant l'affichage des helps "on-line" de la procedure mrxtester.com

MRXTESTER$BUILD.PAS :
Programme PASCAL permettant la création et l'assemblage de messages suivant un format ou template. Pour chaque élément de message, un help "on-line" est disponible.

MRXTESTER$READ.PAS :
Programme PASCAL permettant à un utilisateur de relever une boîte aux lettres.

MRXTESTER$POST.PAS :
Programme PASCAL permettant à un utilisateur de poster un message assemblé.

MRXTESTER$ORNAMES.PAS :
Programme PASCAL permettant une génération automatique de messages de tests d'ORNAMES pour un destinataire donné.

MRXTESTER$ORPOST.PAS :
Programme PASCAL permettant de poster les messages généres par MRXTESTER$ORNAMES.

MRXTESTER$HELP.PAS :
Module PASCAL permettant l'affichage des helps "on-line" pour MRXTESTER$BUILD.

[4] CREATION D'UN UTILISATEUR DU SYSTEME DE MESSAGERIE X.400

Soit un utilisateur à créer : Bernard THIRY.
Cet utilisateur possède un USERID sur le système : [BTHIRY]

[4.1] Creation d'une boîte aux lettres Message Router

```
$run sys$system:mrman
This is MRMAN V2.0
MRM> ADD THIRY/OWNER=BTHIRY/DEFAULT/BEEP
%Added THIRY,     Owner=BTHIRY Beep Default
MRM> exit
```

/OWNER=BTHIRY : lien entre entre la boîte aux lettres THIRY et le userid [BTHIRY].

/DEFAULT : les messages envoyés depuis cette directory le seront par cette boîte aux lettres (sauf avis contraire).

/BEEP : un message sera affiché au terminal de l'utilisateur quand le Message Router aura mis un message dans sa boîte aux lettres.

[4.2] Création d'un abonné X.400

Pour pouvoir recevoir et/ou envoyer du courrier X.400, un utilisateur doit être reconnu par le MRX Gateway.

```
$run sys$system:mrxman
       This is MRXMAN V1.1-024

MRXMAN> ADD
_what: SUBSCRIBER
_MRADDRESS: THIRY/surname=THIRY/given=BERNARD/initials=BTH
%MRX-I-MANSUBCRE,  Subscriber THIRY entry created

MRXMAN> list subscriber thiry/full


Message Router address :  THIRY
PERSONAL DETAILS       - Surname    : THIRY
                         Initials   : BTH
                         Given name: BERNARD
                         Generation:
ORGANIZATION DETAILS   - Organization name :
                                 Unit name :

MRXMAN> exit
```

La 'Message Router address' = la boîte aux lettres de l'abonné. C'est par ce lien que MRX fait le rapport entre les O/R names et les userid du système. On constate que les paramêtres Given name, Generation, Organization name et Unit name sont laissés vide dans notre example.

[5] Utilisation de MRXTESTER

[5.1] Execution du logiciel

MRXTESTER se lance en tapant : **@[MRMANAGER.MRXTESTER]MRXTESTER**

Les commandes MRXTESTER sont les suivantes :

```
CREATE   : Create the MRXTESTER mailbox and subscriber.
BUILD    : Construct a X.400 message following a template.
GENERATE : Generates X.400 messages for O/R names tests.
POST     : Post a X.400 message constructed by BUILD.
ORPOST   : Post X.400 messages constructed by GENERATE.
READ     : Read message from Message Router mailboxes.
EXIT     : Return to VMS.
HELP     : Display this text.
```

[5.2] BUILD : construire un message suivant un template

[5.2.1] Explications

L'utilisateur doit d'abord donner un nom de fichier où seront rangés les fichiers assemblés.
BUILD permet de créer quatre sortes de messages :

- une ENVELOPE seule; celle-ci pourra être postée ultérieurement avec un contenu déja assemblé.

- un CONTENU seul; celui-ci pourra être posté ultérieurement avec une envelope déja assemblée.

- une ENVELOPE et un CONTENU.

- un COMPLETE_MESSAGE : cette forme de message permet à l'utilisateur de créer un contenu de message et le Message Router se chargera de créer lui-même l'envelope à partir des informations contenues dans le <user_header>.

Dans les cas ENVELOPE & CONTENU et COMPLETE_MESSAGE, l'utilisateur a la possibilité de poster son message dés la fin de l'assemblage.

Un nom de fichier de message assemblé aura toujours la forme :

file_name[$env]|[$cnt].NBS

Les extensions $env et $cnt sont utilisées dans les cas de création ENVELOPE & CONTENU et COMPLETE_MESSAGE et correspondent à l'envelope et au contenu respectivement.

[5.2.2] Exemple de creation d'un message COMPLETE_MESSAGE :

```
$ @[MRMANAGER.MRXTESTER]MRXTESTER

        MRXTESTER - MRX X.400 Conformance Testing Procedure V1.0

                (c) Drx & Me jan'88.


MRXTESTER> build

CREATION OF PATTERN BEGINS ...
Output File : exemple

Enter the kind of message you wish to create :
  complete [M]essage.
  [E]nvelope only.
  [C]ontent only.
  [B]oth envelope and content.
Message type : m

<user_content>-------------------------------------------------
[ <header> ] (Y/N) ? Y

[ <user_header> ] (gr)-----------------------------------------
[ <app_message_id> ]     (str) :
[ <subject> ]            (str) : Exemple de creation de complete msg
[ <inreplyto> ]          (str) :
{ <obsoletes> }          (str) :
{ <references> }         (str) :
[ <msgclass> ]           (str) :
[ <precedence> ]         (int) : ?
A simple element that describes how urgent a message is :
0 - high priority (express mail),
1 - normal priority (first class mail),
2 - low priority (second class mail).
On the envelope, PRECEDENCE indicates the priority with which the
message travels.
On the content, it indicates the importance of the message.

[ <precedence> ]         (int) :
[ <autoforward> ]        (int) :
[ <sensitivity> ]        (int) : ?
A simple element that describes the sensitivity of the message. This
can be :
1 - personal,
2 - private,
3 - company-confidential.
How this is implemented is at the discretion of the User Agent.

[ <sensitivity> ]        (int) : 1
[ <pdate> ]              (Y/N) ?
[ <enddate> ]            (Y/N) ?
[ <replyby> ]            (Y/N) ?
[ <create_date> ]        (Y/N) ? Y
[ <create_date> ]        (Y/N) ? dd-MMM-yyyy hh:mm:ss.cc :22-JAN-1988
[ <from> ]               (Y/N) ? ?
A composite element that is the name of the originator of the msg.
This is the person to whom any replies should be sent, unless
REPLYTOUSERS appears on the content. FROM is usually, but not
always, the same person as the SENDER on the envelope.
```

```
[ <from> ]                  (Y/N) ? Y
<descriptive_name>
<freeform>                  (str) :
[ <orname> ]
<X400_orname>          (1,2,3,4) : ?
Choose one of the following :
<orname_1>::=<x121address>+[ <terminalid> ]+<mradress>
<orname_2>::=<country>+<amdname>+<uniqueuaid>+<mraddress>+{<domains
<orname_3>::=<country>+<amdname>+<x121address>+<mraddress>+{<domains
<orname_4>::=<country>+<amdname>+<mraddress>+<choice>

<X400_orname>          (1,2,3,4) : 4
[ <country> ]               (str) : be
[ <amdname> ]               (str) : rtt
<mr_address>
<userid>                    (mbx) : reuviaux
{ <route> }                 (str) :
[ <CHOICE> : At least one of the followings. ]
[ <personnal_name> ]

<surname>                   (str) : reuviaux
[ <givenname> ]             (str) : daniel
[ <initials> ]              (str) : drx
[ <generation> ]            (str) :
[ <prdname> ]               (str) : iihe
[ <orgname> ]               (str) :
[ <orgunit> ]               (str) :
[ <domainspecific> ]

[ <telephone> ]             (str) :
[ <location> ]              (str) :
{ <author> }                (Y/N) :
{ <to> }                    (Y/N) ? y
[N]ame or [L]ist            (N/L) : n
<descriptive_name>
<freeform>                  (str) :
[ <orname> ]
<X400_orname>          (1,2,3,4) : 4
[ <country> ]               (str) : be
[ <amdname> ]               (str) : rtt
<mr_address>
<userid>                    (mbx) : speltens
{ <route> }                 (str) :
[ <CHOICE> : At least one of the followings. ]
[ <personnal_name> ]

<surname>                   (str) : speltens
[ <givenname> ]             (str) : marc
[ <initials> ]              (str) :
[ <generation> ]            (str) :
[ <prdname> ]               (str) : iihe
[ <orgname> ]               (str) :
[ <orgunit> ]               (str) :
[ <domainspecific> ]

[ <telephone> ]             (str) :
[ <location> ]              (str) :
[ <reportflg> ]             (flg) ?
[ <replyreq> ]              (int) :
```

```
{ <to> }                    (Y/N) ?
{ <cc> }                    (Y/N) ?
{ <bcc> }                   (Y/N) ?
{ <replytousers> }          (Y/N) ?


{ <body> }                  (Y/N) ? ?
<body>::=<text_bodypart>

{ <body> }                  (Y/N) ? y
We just limit ourselves to <text_bodypart> for the moment.
<text_bodypart> (file) : first.txt
%RMS-E-FNF, file not found
%TRACE-E-TRACEBACK, symbolic stack dump follows
module name     routine name     line       rel PC    abs PC

MRXTESTER$BUILD USER_BODY        1586       000000E7  00006787
MRXTESTER$BUILD USER_CONTENT     1647       00000133  00006F4F
MRXTESTER$BUILD MRXTESTER$BUILD  1697       0000031F  0000727F
{ <body> }                  (Y/N) ? Y
<text_bodypart> (file) : [eloy.textes]first.txt
{ <body> }                  (Y/N) ? N

The creation is now complete.

Do you want to post this message NOW [N] ? Y
Connecting to node BVX82
Identified to mailbox MRXTESTER
Message posted successfully with message_id : O51404122108891/...
```

A partir de ce moment, le Message Router a accepté le message et en endosse la responsabilité.

[5.2.3] Quelques remarques :

- Les éléments d'un message peuvent être obligatoires <item>,
  optionnels [<item>],
  répétitifs { <item> }.

- Un élément peut être simple ou composé.

- Si l'on souhaite la présence d'un élément simple dans le message, il suffit de rentrer la valeur que l'on souhaite qu'il prenne. Si l'on ne veut pas de la présence d'un élément, il suffit de taper <Return>.

- Un élément composé est introduit par (Y|N) : il s'agit des flags, des noms, des dates ...

- De l'aide sur un élément peut être obtenue en tapant '?<Return>'.

- Certaines erreurs sont gérées par le programme et ne cause pas l'arret de celui-ci.

- Pour éviter des erreurs à l'encodage des chaînes de caractères (pour des tests de longueurs par exemple), le logiciel intègre une bibliothèque de composants pré-définis.

  Quand on veut utiliser un tel composant, on tape '@' suivi de son nom, MRXTESTER ira le rechercher dans la table.

  Exemple : Subject (str) : **@SJ1**
  ira rechercher l'élément 'SJ1' dans la table.

On trouvera en annexe, un autre exemple de création de message qui utilise cette possibilité ainsi que la définition complète de la table.

[5.3] POST : faire accepter au Message Router un message assemblé

[5.3.1] Explications

Le programme essaye dans un premier temps de se connecter au noeud local (c-a-d : réserver de la place memoire pour travailler).
Il se peut que la tentative de connection échoue si la charge du système est trop importante. Dans ce cas, il suffit généralement de réessayer ultérieurement. Si ces tentatives échouent, consultez votre system manager.

Si la connection réussi, le programme essayera d'utiliser la boîte aux lettres MRXTESTER pour communiquer avec le Message Router, si cette boîte n'existe pas, il vous demandera de lui fournir un nom de boîte aux lettres M.R. existante ainsi que son mot de passe eventuel.
En tapant <Return> à la question MAILBOX >, on demande au Message Router d'utiliser la "default mailbox" associée à l'utilisateur. (cfr. plus haut)
Si vous ne possédez pas de boîte aux lettres, ou n'en connaissez aucune, consultez votre M.R. manager.
Un utilisateur peut créer la boîte aux lettres MRXTESTER en utilisant la commande CREATE.

[5.4] READ : Relever une boîte aux lettres

[5.4.1] Explications

Un utilisateur peut demander de consulter sa boîte aux lettres. Pour se faire, le programme doit d'abord se connecter au Message Router et s'identifier à une boîte au lettres existante. Les remarques définies pour POST restent valables ici. Ceci fait, l'utilisateur peut demander à MRXTESTER de désassembler son (ses) message(s) et de les ranger dans un (des) fichier(s). Un nom de fichier aura toujours la forme : mailbox_name+number.mhs, ou number est le numéro d'ordre du message dans la boîte aux lettres.

Une fois qu'un message est enlevé de la boîte aux lettres, il n'existe plus pour le Message Router : il est sous la responsabilité de l'utilisateur.

[5.4.2] Exemple d'utilisation de READ pour le message créé plus haut

```
MRXTESTER> read

Connecting to node BVX82
Mailbox [default mailbox]          > speltens
Password [optional]                >
Mailbox [SPELTENS] contains   1 NEW message.
Do you wish to read it/them ? [Y]
Output file [SPELTENS01.MHS] (TT:) > tt:
```

Remarque : "tt:" comme nom de fichier permet un affichage à l'ecran.

```
Message nr  1 fetched successfully.
-----------------------------------------------
MESSAGE NR  1 :

MESSAGE-ENVELOPE
   MESSAGE_ID          : 51404122108891/378@BVX82
   PDATE               : 22-JAN-1988 14:04:15.00
   TO
      COUNTRY          : be
      AMDNAME          : rtt
      PRDNAME          : iihe
      SURNAME          : speltens
      GIVENNAME        : marc
      USERID           : SPELTENS
      PERRECFLG        : %X000000A8
         ACTION        : Y
         MRBASIC       : Y
         MRCONFIRMED   : N
         UABASIC       : Y
         UACONFIRMED   : N
   SENDER
      USERID           : MRXTESTER
   HOPCOUNT            : %X00000001
   CONTENTTYPES        : %X00000001
```

```
MESSAGE-CONTENT
   SUBJECT              : Exemple de création de complete msg
   SENSITIVITY          : %X00000001
   DATE                 : 22-JAN-1988 14:00:29.00
   FROM
      COUNTRY           : be
      AMDNAME           : rtt
      PRDNAME           : iihe
      SURNAME           : reuviaux
      GIVENNAME         : daniel
      INITIALS          : drx
      USERID            : reuviaux
   TO
      COUNTRY           : be
      AMDNAME           : rtt
      PRDNAME           : iihe
      SURNAME           : speltens
      GIVENNAME         : marc
      USERID            : speltens
   TEXT

     This is the first X.400 message send from the
     Inter-university Institutes for High Energies (Brussels)
     to the Facultes Universitaires Notre-Dame de la Paix
     in Namur.

-----------------------------------------------

MRXTESTER> exit
End of MRXTESTER
```

[5.5] CREATE :

Cette commande permet à l'utilisateur de créer la boîte aux lettres MRXTESTER, ainsi qu'un abonné X.400 MRXTESTER. Cette création ne doit avoir lieu qu'une fois. Certains privilèges sont requis pour exécuter MRXMAN.

[5.6] GENERATE :

Cette fonction permet de générer automatiquement une liste de test d'O/R names pour une destination donnée.

Ces tests couvrent les formats 1.1, 1.2, 1.3 et 1.4 des formats d'O/R names X.400.

Pour chaque format, sont générés les messages suivants :

¤01 ORname vide
¤02 ORname correct minimum (Elts obligatoires seulement)
¤03 ORname correct maximum (Elts obligat. et optionnels)
¤04 ORname errone

Les Messages de test sont stockés dans des fichiers appelés

'ORTEST(name)(1-4)(01-04)$ENV.NBS' pour l'enveloppe
et 'ORTEST(name)(1-4)(01-04)$CNT.NBS' pour le contenu

'name' étant le nom du test
le premier chiffre le numéro du format testé
1 - format 2
2 - format 1.2
3 - format 1.3
4 - format 2
les 2 derniers chiffres le type du test

[5.7] ORPOST :

Cette fonction permet de poster un message qui a été généré par GENERATE.

Il établi la connection avec votre noeud local (s'il est disponible), vous identifie au Message Router et vous demande le nom du message ORTEST à poster.

[6] Maintenance du logiciel MRXTESTER

[6.1] Conseils préliminaires

1. Veillez à modifier les sources originales se trouvant dans la directory [MRMANAGER.MRXTESTER].
2. Gardez toujours une version originale.
3. Tenez la documentation le plus à jour possible.

[6.2] Edition et mise à jour

Par l'éditeur de texte **EDIT**.

Exemple : EDIT MRXTESTER$READ.PAS

[6.3] Compilation

Par la commande **PASCAL prog_name**
en ajoutant pour le MRXTESTER$READ, l'option **/NOOPTIMIZE**
pour des raisons de compatibilité de versions du PASCAL.

[6.4] Link

Par la commande **LINK prog_name [,mrxtester$help], opt/options**

opt.opt est un fichier
contenant la phrase "SYS$SHARE:MRIFSHARE/SHARE"

[,mrxtester$help] est un programme à linker au programme MRXTESTER$BUILD uniquement.
Il s'occupe du HELP ON-LINE (Option '?').

[6.5] Destruction des anciennes versions

Par la commande **PURGE**

[6.6] Execution

Par la commande **@[MRMANAGER.MRXTESTER]MRXTESTER**

[7] Exemple d'utilisation du MRXTESTER

[7.1] Composition

```
ROOT>@mrxtester

      MRXTESTER - MRX X.400 Conformance Testing Procedure V1.0

      (c) Drx & Me jan'88.

MRXTESTER> help

MRXTESTER Utilities

 CREATE   : Create the MRXTESTER mailbox and subscriber.
 BUILD    : Construct a X.400 message following a template.
 GENERATE : Generates X.400 messages for O/R names tests.
 POST     : Post a X.400 message constructed by BUILD.
 ORPOST   : Post X.400 messages constructed by GENERATE.
 READ     : Read message from Message Router mailboxes.
 EXIT     : Return to VMS.
 HELP     : Display this text.

Topic ? build


  BUILD

  This facility allows you to create a X.400 message file following
  a template. This message will be stored in a file in order to be
  posted by the POST utility.

 ********* WARNING ! *********

  This utility is test utility.
  Users should know how to write a X.400 message before using it.
  This utility allows the user to enter wrong messages in order to
  test the MTA's reaction to them.

 DO NOT USE THIS UTILITY AS A COMMON USER AGENT

  unless you are sure of what you are doing.

    Additionnal informations available :

  Message_item   Message_type   Data_type


Build subtopic : message_item

      Message item

  A message item can be mandatory : <item>
                       optionnal : [ <item> ]
                      repetitive : { <item> } or { [ <item> ] }
```

Build subtopic : **message_type**

Message type.

You can build a complete message, an envelope and a content, an envelope only, or a content only.

You create 'ENVELOPE ONLY' messages in order to use them with the same content.
You create 'CONTENT ONLY' messages in order to use them with the same envelope.
If want to construct both the envelope and the content, use the 'BOTH' option.
'COMPLETE MESSAGES' are messages where only the content is enter by the user, the envelope is added by the Message Router, following the elements it finds in the 'user header'.

Build subtopic : **data_type**

Data type.

Data in a message may be :

(str) : strings,
(int) : integers,
(flg) : flags,
(date) : dates,
(gr) : groups of data,
(rec) : recursion,
(mbx) : mailbox.

Refer to the Message Router guide 'How to Write a Message Router Application' for the use of flags in a message.

Build subtopic :
Topic ?

MRXTESTER> **build**

CREATION OF PATTERN BEGINS ...
Output File : **msg**

Enter the kind of message you wish to create :
complete [M]essage.
[E]nvelope only.
[C]ontent only.
[B]oth envelope and content.
Message type : **B**

<user_envelope> ----------

[<message_id>] (sys:always)
[<uacontid>] (str) : **jksdfnvkjnadkfjbna**
[<hopcount>] (int) : **1**
[<contenttypes>] (int:0,1,2) : **1**
[<env_precedence>] (int:0,1,2) : **1**
[<env_pdate>] (sys:always)
[<deferred>] (dat) ? dd-MMM-yyyy hh:mm:ss.cc : **01-FEB-19**

```
[ <encodedtypes> ]          (flg)    ? Y
WARNING : This program adds only TEXT bodyparts to a message.
   <text>                   ? Y
   [ <voice> ]              ? ?
A bit defined in ENCODEDTYPES. When set, it indicates a voice bodypart
in the message. A voice bodypart contains a bit string representing
digitized voice. This is not yet implemented. (!)

Enter a value for this item :
   [ <videotex> ]           ?
   [ <teletex> ]            ?
   [ <tif0> ]               ?
   [ <tif1> ]               ? ?
A bit defined in ENCODEDTYPES. When set, it indicates that the message
content contains at least one TIF1 bodypart in the message. A TIF1 bodypart
contains a document whose structure conforms to Text Interchange Format 0
application rules and is defined in Recommendation S.a,
Document interchange protocol for telematic services.

Enter a value for this item :
   [ <g3fax> ]              ?
   [ <ia5text> ]            ?
   [ <telex> ]              ?
   [ <undefined> ]          ?
   [ <sfd> ]                ?
   [ <rmsfile> ]            ?
   [ <wpsplus> ]            ?
   [ <ddif> ]               ?
   [ <voicenotif> ]         ? ?
A bit defined in ENCODEDTYPES. When set, it indicates that the message
content contains at least one voice notification bodypart. A voice
notification bodypart tells recipients that voice mail has been sent to
them.

Enter a value for this item :
   [ <odif> ]               ?
[ <permsgflg> ] (flg)    ? Y
   Up to now, MR v.2.0 acts as follows :
   <discloserec> & <convprohib> : always set, <altrecip> ignored
   <contretreq> O.K.
   [ <contretreq> ]         ? Y
   [ <altrecip> ]           ?
   [ <convprohib> ]         ? Y
   [ <discloserec> ]        ? Y
[ <sender_name> ] (grp) ? Y
<X400_orname_[1]>, <X400_orname_[2]>, <X400_orname_[3]>, <X400_orname_[4]>
Format : 4
   <country> (str)          : BE
   <amdname> (str)          : RTT
   <mr_address> (gr)
     <userid> (mbx)         : ELOY
     Do you want to add a route [N] ?
   [ <CHOICE> : At least one of the followings. ]
    [ <personnal_name> ]
      <surname> (str)             : MARC
      [ <givenname> ] (str)       : ELOY
      [ <initials> ] (str)        : ME
      [ <generation> ] (str)      :
    [ <prdname> ] (str)           : IIHE
    [ <orgname> ] (str)           :
    [ <orgunit> ] (str)           :
```

<domainspecific> (gr) ?

```
[ <telephone> ]          :
[ <location> ]           :
```

```
<recipient> { <recipient> }
<env_to>          ? : Y
<env_to>
<N>ame or <L>ist   ? N
<env_name>
<orname>
<X400_orname_[1]>, <X400_orname_[2]>, <X400_orname_[3]>, <X400_orname_[4]>
Format : 4
   <country> (str)              : BE
   <amdname> (str)              : RTT
   <mr_address> (gr)
     <userid> (mbx)             : REUVIAUX
     Do you want to add a route [N] ?
   [ <CHOICE> : At least one of the followings. ]
    [ <personnal_name> ]
      <surname> (str)           : Daniel
      [ <givenname> ] (str)     : Reuviaux
      [ <initials> ] (str)      :
      [ <generation> ] (str)    :
    [ <prdname> ] (str)         :
    [ <orgname> ] (str)         :
    [ <orgunit> ] (str)         :
    <domainspecific> (gr)   ?
      [ <telephone> ]           :
      [ <location> ]            :
<perrecflg> (flg)               ? Y
   Here are the permitted combinations :
   none, action, ua_basic, ua_confirmed,
   action & ua_basic, action & ua_confirmed.
   <action>                     ? Y
   [ <ua_basic> ]               ?
   [ <ua_confirmed> ]           ? Y
   [ <mr_basic> ]               ?
   [ <mr_confirmed> ]           ?
[ <extensionid> ] (int)         :
[ <explicconv> ]  (int)         :
<env_cc>   ? : Y
<env_cc>
<env_cc_[N]ame> or <env_cc_[L]ist>  ? N
<env_name>
<orname>
<X400_orname_[1]>, <X400_orname_[2]>, <X400_orname_[3]>, <X400_orname_[4]>
Format : 1
   <x121address> (str)          : 28160000
   [ <terminalid> ] (str)       : terminal_x121
   <mr_address> (gr)
     <userid> (mbx)             : SYSTEM
     Do you want to add a route [N]
<perrecflg> (flg)  ? Y
   Here are the permitted combinations :
   none, action, ua_basic, ua_confirmed,
   action & ua_basic, action & ua_confirmed.
   <action>                     ? N
   [ <ua_basic> ]               ?
   [ <ua_confirmed> ]           ?
   [ <mr_basic> ]               ?
   [ <mr_confirmed> ]           ?
[ <extensionid> ] (int)         :
[ <explicconv> ]  (int)         :
<env_bcc>   ? : N
{ <recipient> }  ? N
```

```
<user_content> ------------------------------------------------------------
Do you want to create a user_header [Y] ? Y

[ <user_header> ] (gr) ----------------------------------------------------
[ <app_message_id> ] (sys)   : @TB2
[ <subject> ] (str)          : @SJ1
[ <in_reply_to> ] (str)      : texte encode a la main...
[ <obsoletes> ] (str)        : @TB1
[ <references> ] (str)       : texte provenant de la librairie de composant
[ <msgclass> ] (str)         :
[ <cnt_precedence> ] (int)   :
[ <autoforward> ] (int)      :
[ <sensitivity> ] (int)      :
[ <cnt_pdate> ] (date)       ? Y dd-MMM-yyyy hh:mm:ss.cc     :
[ <end_date> ] (date)        ? Y dd-MMM-yyyy hh:mm:ss.cc     : 23-JAN-1988
[ <reply_by> ] (date)        ? N
[ <create_date> ] (date)     ? N
[ <from_name> ] (gr)         ? N
{ <author_name> } (gr)       ? N
{ <cnt_to> } (gr)            ? N
{ <cnt_cc> } (gr)            ? N
{ <cnt_bcc> } (gr)           ? N
{ <reply_to_user> } (gr)     ? N

Do you want to create a user_body [Y]    ? y

{ body } ------------------------------------------------------------------
We just limit ourselves to <text_bodypart> for the moment.
<text_bodypart> (file) : first.txt
The creation is now complete.

Do you want to post the message NOW [N]   ? n
```

[7.2] ENVOI

```
MRXTESTER> help

MRXTESTER Utilities

 CREATE   : Create the MRXTESTER mailbox and subscriber.
 BUILD    : Construct a X.400 message following a template.
 GENERATE : Generates X.400 messages for O/R names tests.
 POST     : Post a X.400 message constructed by BUILD.
 ORPOST   : Post X.400 messages constructed by GENERATE.
 READ     : Read message from Message Router mailboxes.
 EXIT     : Return to VMS.
 HELP     : Display this text.

Topic ?  post


  POST
  This utility allows you to post a message which has been
  assembled before with the BUILD utility.
  It establishes the connection with your local node (if it is
  available), identify yourself to a Message Router mailbox,
  and them prompts you for the message you wish to post.

  Additionnal information available :

   Mailbox   Message_type   Node


Post subtopic : mailbox

     Mailbox.

It is the way the Message Router identifies a user and makes the
relation between a user name and an user account on the system.
By default the procedure uses the MRXTESTER mailbox (if any).
If this mailbox doesn't exist you can create it (see CREATE) or
enter your mailbox name and password. Each user would have a
default mailbox, which is identifiable by typing <Return> to the
Mailbox and Password prompts.


Post subtopic : message_type

       Message type.

  You can build a complete message, an envelope and a content, an
  envelope only, or a content only.

  You create 'ENVELOPE ONLY' messages in order to use them with the
  same content.
  You create 'CONTENT ONLY' messages in order to use them with the
  same envelope.
  If want to construct both the envelope and the content, use the
  'BOTH' option.
  'COMPLETE MESSAGES' are messages where only the content is enter
  by the user, the envelope is added by the Message Router,
  following the elements it finds in the 'user header'.
```

```
Post subtopic : node

     Node.

It is the name of the local node where the Message Router is
running. There can be local or remote nodes (see the Message Router
Manager's Guide for details). The procedure uses your local node
name as a default.


Post subtopic : erreur
MRXTESTER-W-NOINFAVBL, No information available for item ERREUR.
Post subtopic :
Topic ?

MRXTESTER> post
Connecting to node BVX82
Identified to mailbox MRXTESTER
Complete message file :
Envelope message file : msg$env
Content message file  : msg$cnt
Message posted successfully with message_id : O14636102108891/321@BVX82

Do you want to post another message [Y] ? N

New service message on BVX82 for ELOY
MRXTESTER>
```

[7.3] LECTURE

```
MRXTESTER> help

MRXTESTER Utilities

 CREATE   : Create the MRXTESTER mailbox and subscriber.
 BUILD    : Construct a X.400 message following a template.
 GENERATE : Generates X.400 messages for O/R names tests.
 POST     : Post a X.400 message constructed by BUILD.
 ORPOST   : Post X.400 messages constructed by GENERATE.
 READ     : Read message from Message Router mailboxes.
 EXIT     : Return to VMS.
 HELP     : Display this text.

Topic ?  read

     READ

  This utility allows you to read the messages in a mailbox.
  It establishes the connection with your local node and asks you
  for a mailbox name and password to identify to.
  It fetches this mailbox and warns you if it has found message(s)
  in it. If you want to read them, it will prompts you for a file
  name where the message wil be translate. A message can only be
  read once. When it has been read the Message Router assumes no
  more responsability for it.

     Additionnal information available :

     Mailbox   Node


Read subtopic :
Topic ?

MRXTESTER> read

Connecting to node BVX82
Mailbox [default mailbox]        > ELOY
Password [optional]              >
Mailbox [ELOY] contains   1 NEW message.
Do you wish to read it/them ? [Y]
Output file [ELOY01.MHS] (TT:)   >
Message nr  1 fetched successfully.

MRXTESTER> exit
End of MRXTESTER
```

```
ROOT>type eloy01.mhs
------------------------------------------------
MESSAGE NR  1 :

SERVICE-MESSAGE-ENVELOPE
   MESSAGE_ID     : 54636102108891/322@BVX82
   PDATE : 20-JAN-1988 16:36:45.00
   TO
      COUNTRY         : BE
      AMDNAME         : RTT
      PRDNAME         : IIHE
      SURNAME         : MARC
      GIVENNAME       : ELOY
      INITIALS        : ME
      USERID: ELOY
   HOPCOUNT        : %X00000001
SERVICE-MESSAGE-CONTENT
   MESSAGE_ID     : 14636102108891/321@BVX82
   UACONTID        : jksdfnvkjnadkfjbna
   GENERATED       : BVX82

------------------------------------------------
```

[8] Exemple d'utilisation du GENERATEUR AUTOMATIQUE de cas de test pour O/Rnames (Naming & Adressing)

[8.1] GENERATE

```
ROOT>@mrxtester

       MRXTESTER - MRX X.400 Conformance Testing Procedure V1.0

       (c) Drx & Me jan'88.


MRXTESTER> help

MRXTESTER Utilities

 CREATE   : Create the MRXTESTER mailbox and subscriber.
 BUILD    : Construct a X.400 message following a template.
 GENERATE : Generates X.400 messages for O/R names tests.
 POST     : Post a X.400 message constructed by BUILD.
 ORPOST   : Post X.400 messages constructed by GENERATE.
 READ     : Read message from Message Router mailboxes.
 EXIT     : Return to VMS.
 HELP     : Display this text.

Topic ?  generate


  GENERATE
         This utility allows you to generate automatically a list
         of O/R names test cases for a specified destination.
         These tests covers form 1.1, 1.2, 1.3 and 2 of the X.400
         O/R names formats.
         Messages files are 'ORTEST(name)(1-4)(01-04)$ENV.NBS'
                        and 'ORTEST(name)(1-4)(01-04)$CNT.NBS'


Topic ? orpost

  ORPOST
  This utility allows you to post a message which has been
  assembled before with the GENERATE utility.
  It establishes the connection with your local node (if it is
  available), identify yourself to a Message Router mailbox,
  and them prompts you for the message you wish to post.


Topic ?
```

```
MRXTESTER> generate

This program AUTOMATICALY generates suites of O/R naming and adressing tests
patterns which will have to be used with the X400 Message Tester
                                                        (c) Me & Drx 1987
Results files are called "ORTESTname(1-4)(1-n)$ENV.NBS" (envelope)
                     for "ORTESTname(1-4)(1-n)$CNT.NBS" (contents)

Enter name of TEST [] :BXL_NAM

(ORIGINATOR is always MRXTESTER)

Please enter DATA on RECIPIENT :
x121adr                    : 2816000
terminalid                 : nom du terminal
country      [BE]          :
amdname      [RTT]         :
uniqueuaid                 : unique_UA_ID
surname                    : ALAIN
givenname                  : Vankerm
initials                   : AVK
generation                 :
PRDname                    : FNDP
userid       [MRXTESTER]   : FNDP
ORGname                    :
orgunit                    :
telephone                  :
location                   :
GENERATION OF PATTERN BEGINS ...

TEST built with filenames : ORTEST_BXL_NAM101$env & ORTEST_BXL_NAM101$cnt
TEST built with filenames : ORTEST_BXL_NAM102$env & ORTEST_BXL_NAM102$cnt
TEST built with filenames : ORTEST_BXL_NAM103$env & ORTEST_BXL_NAM103$cnt
TEST built with filenames : ORTEST_BXL_NAM104$env & ORTEST_BXL_NAM104$cnt
TEST built with filenames : ORTEST_BXL_NAM201$env & ORTEST_BXL_NAM201$cnt
TEST built with filenames : ORTEST_BXL_NAM202$env & ORTEST_BXL_NAM202$cnt
TEST built with filenames : ORTEST_BXL_NAM203$env & ORTEST_BXL_NAM203$cnt
TEST built with filenames : ORTEST_BXL_NAM204$env & ORTEST_BXL_NAM204$cnt
TEST built with filenames : ORTEST_BXL_NAM301$env & ORTEST_BXL_NAM301$cnt
TEST built with filenames : ORTEST_BXL_NAM302$env & ORTEST_BXL_NAM302$cnt
TEST built with filenames : ORTEST_BXL_NAM303$env & ORTEST_BXL_NAM303$cnt
TEST built with filenames : ORTEST_BXL_NAM304$env & ORTEST_BXL_NAM304$cnt
TEST built with filenames : ORTEST_BXL_NAM401$env & ORTEST_BXL_NAM401$cnt
TEST built with filenames : ORTEST_BXL_NAM402$env & ORTEST_BXL_NAM402$cnt
TEST built with filenames : ORTEST_BXL_NAM403$env & ORTEST_BXL_NAM403$cnt
TEST built with filenames : ORTEST_BXL_NAM404$env & ORTEST_BXL_NAM404$cnt

END OF GENERATION
```

[8.2] ORPOST

```
MRXTESTER> orpost

Connecting to node BVX82
Identified to mailbox MRXTESTER
Name of ORTEST message file : ORTEST_BXL_NAM403
Envelope message file : ORTEST_BXL_NAM403$ENV
Content message file  : ORTEST_BXL_NAM403$CNT
Message posted successfully with message_id : O34046102108891/323@BVX82

Do you want to post another message [Y] ? Y

Name of ORTEST message file : ORTEST_BXL_NAM404
Envelope message file : ORTEST_BXL_NAM404$ENV
Content message file  : ORTEST_BXL_NAM404$CNT
Message posted successfully with message_id : O85046102108891/324@BVX82

Do you want to post another message [Y] ?

Name of ORTEST message file : ORTEST
Do you want to post another message [Y] ? N

MRXTESTER> exit
End of MRXTESTER
```

[9] TABLES DES COMPOSANTS

Cette table contient une bibliotheque de composants pré-définis, la 'SPAG Component Library' définie dans le 'SPAG Interworking Guide' (pages 6.1-4)

[9.1] TABLES DES NOMS ET TYPES

TYPE	NOM	REM	DESCRIPTION
Typed Body Components			
	TB1	L_IA5text_1	Small IA5 string
	TB11		Full IA5 character set
	TB2X		1500 characters IA5 string
	TB3		ISO6937 body
	TB4X		1500 characters ISO6937
	TB5		Forwarded bodypart (Test1 V->R)
	TB6		Forwarded bodypart (Test2 R->V)
IP-Message-ID Components			
	IP1	L_IPMessa_ID_0	small prt
	IP1X		max 64 prt
	IP2		small prt
	IP2X		max 64 prt
	IP21		small prt
	IP21X		max 64 prt
	IP3		small prt
	IP3X		max 64 prt
	IP4		small prt
	IP4X		max 64 prt
	IP5		small prt
	IP5X		max 64 prt
	IP6		small prt
	IP6X		max 64 prt
Message_ID components			
	MS1		vendor specific
	MS2X		max 16 prt
Subject Components			
	SJ1	L_T61string_4	Small T61 string
	SJ11		Long T61 string
	SJ2X		Max 128 T61 string
Replying-IP-Message-ID Components			
	RI1	L_IPMessagIP_1	small prt
	RI2X		marx 64 prt
Printable Strings			
	ps1		length 51
	ps2		length 23
	ps3		length 23
	ps4		length 23
	ps5		length 23
	ps6		length 32
	ps7		length 6
	ps8		length 16
	ps9		length 25
	psA		length 53
T.61 Strings			
	T610		length 51

[9.2] CONTENUS

REMARQUE : Les contenus entre parenthèses ne sont pas implémentés.

```
TB1   : component='ABCDEFGHIJLKM0123456789'
TB11  : component='ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnop
                   qrstuvwxyz !"#$%''(*+,-./:;<=>?@[\]^_`{|}~'
TB2X  : component='(1500 char IA5String)'
TB3   : component='A!"%&()*+,-./A:;<=>?END'
TB4X  : component='(1500 char ISO6937)'
TB5   : component='(Forwarded bodypart (test1 Vendor to Reference))'
TB6   : component='(Forwarded bodypart (test2 Reference to Vendor))'

IP1   : component='Test number 1.'
IP1X  : component='Test number 1.Maximum length of 64 Printable string
                   .........END.'
IP2   : component='Test number 2.'
IP2X  : component='Test number 2.Maximum length of 64 Printable string
                   .........END.'
IP21  : component='Suite of test number 2.'
IP21X : component='Suite of test number 2.Maximum length of 64
                  Printable StringEND.'
IP3   : component='Test number 3.'
IP3X  : component='Test number 3.Maximum length of 64 Printable
                   string.........END.'
IP4   : component='Test number 4.'
IP4X  : component='Test number 4.Maximum length of 64 Printable
                   string.........END.'
IP5   : component='Test number 5.'
IP5X  : component='Test number 5.Maximum length of 64 Printable
                   string.........END.'
IP6   : component='Test number 6.'
IP6X  : component='Test number 6.Maximum length of 64 Printable
                   string.........END.'

MS1   : component='(vendor defined)'
MS2X  : component='ABCDEFGHIJKLMNOP'

SJ1   : component='Subject number 1.'
SJ11  : component='ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'
SJ2X  : component='(max 128 T61 string)'

RI1   : component='ABCDEFGH.'
RI2X  : component='ABCDEFGHIJKLMNOPQRSTUVWXYZ''()+,-./=?0123456789abcde
                   fghijklmnop'

ps1   : component='ABCDEFGHIJKLMNOPQRSTUVWXYZ ''()+,-./=?0123456789abc'
ps2   : component='PQRSTUVWXYZ ''()+,-./=?'
ps3   : component='0123456789abcdefghijklm'
ps4   : component='nopqrstuvwxyzABCDEFGHIJ'
ps5   : component='MNOPQRSTUVWXYZ ''()+,-./'
ps6   : component='defghijklmnopqrstuvwxyzABCDEFGHI'
ps7   : component='JKLMNO'
ps8   : component='=?0123456789abc'
ps9   : component='defghijklmnopqrstuvwxyzAB'
psA   : component='CDEFGHIJKLMNOPQRSTUVWXYZ''()+,-./=?
                   0123456789abcdefgh'

T610  : component='(T.61 string (51 char))'
```