MASTER EN SCIENCES INFORMATIQUES

Les protocoles de transfert de fichiers

Daene, Bertrand

Award date:
1985

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix (Namur)

Institut d'Informatique

Année académique 1984 - 1985

LES PROTOCOLES

DE TRANSFERT DE FICHIERS

Mémoire présenté par

BERTRAND DAENE

en vue de l'obtention
du titre de
licencié et maître en informatique

Promoteur Ph. Van BASTELAER

Remerciements
==========

Je voudrais tout d'abord remercier Monsieur Philippe Van Bastelaer, et surtout Béatrice Scoyer pour sa patience et sa remarquable disponibilité. Merci aussi à Patrick Geurts, pour les moments passés ensemble à maîtriser les caprices du PDP-11.

Ma gratitude va également à ma famille, ainsi qu'aux membres et amis du Centre Religieux Universitaire pour les joyeuses années de vie communautaire.

Merci enfin à tous ceux qui ont contribués directement ou indirectement à ce mémoire.

TABLE DES MATIERES

TABLE DES MATIERES

INTRODUCTION

0.1. Contexte

Les prochaines années promettent une large expansion des réseaux d'ordinateurs dans lesquels des équipements de différents constructeurs et utilisateurs seront connectés un niveau national et international. De tels réseaux permettront de partager des ressources onéreuses entre de nombreux utilisateurs, et de transférer données et programmes rapidement et à peu de frais. Des réseaux d'ordinateurs de différentes tailles existent déjà. En Belgique, par exemple, le réseau DCS est de plus en plus disponible. Autre exemple: l'expérience Minitel en France, qui permet l'accès à des bases de données. Dans le futur, il est probable que la plupart des grandes compagnies vont établir leur propre réseau afin de fournir le support dont elles ont besoin pour l'automatisation de leur administration et de leur production. A leur tour, ces réseaux nécessiteront probablement des connexions aux réseaux régionaux, nationaux et internationaux pour accéder à des services et des données non disponibles localement. Les entreprises les plus avancées dans ce domaine sont certainement les banques.

Les utilisations des transferts dans les réseaux publics ou à longue distance sont nombreuses: diffusion des données (par exemple, des tables de barèmes), centralisation des données (par exemple, la collecte de données statistiques), accès à des bases de données comme source de documentation ou d'information (par exemple, des bases publiques de données statistiques ou économiques), transfert de logiciel et distribution de documments (par exemple, rapports ou articles journalistiques). En outre, le transfert de fichiers sert de base aux applications suivantes: courrier électronique, bases de données réparties, transfert de travaux (job transfer) et systèmes d'exploitation répartis.

Etablir un lien de communication entre les divers équipements ne constitue qu'une partie du problème; pour gérer ces communications, des logiciels sont nécessaires. Actuellement, les transferts se font entre deux ordinateurs d'une même entreprise ou d'un même constructeur ou, en tous cas, entre deux points définis à l'avance, ce qui permet à la même personne ou à la même équipe d'écrire le logiciel aux deux extrémités du transfert. Mais avec l'accès aux réseaux publics, le nombre de correspondants croît dans des proportions énormes et on ne peut plus déterminer à l'avance les correspondants potentiels. A partir de ce moment il est nécessaire de définir des conventions indépendantes de tout constructeur ou utilisateur particulier.

C'est le rôle des protocoles standards qui fournissent une convention entre utilisateurs. Un utilisateur peut alors écrire un logiciel qui permettra de communiquer avec tous les correspondants qui auront suivi ce protocole. Il n'est donc plus nécessaire que la même personne ou la même équipe écrive les logiciels de chaque ordinateur en vue de la communication avec les autres. Outre ces avantages, les protocoles permettent une grande diminution des efforts de programmation et offrent la possibilité d'utiliser des produits 'clés en mains'.

Afin de rendre possible la communication entre tous les utilisateurs, il s'impose de définir une norme internationale. C'est le rôle des instituts de normalisation tels que l'I.S.O., le C.C.I.T.T. ou l'E.C.M.A. On trouvera une description de ces différents instituts au chapitre 4 du mémoire de Didier de GHELLINCK [DdG]. Je me suis particulièrement intéressé dans ce mémoire aux Protocoles de Transfert de Fichiers qui permettent de déplacer ou de copier des ensembles d'informations, appelés fichiers, entre différents ordinateurs via des moyens de télécommunication.

0.2. Objectifs

La démarche de ce mémoire consiste, en un premier temps, dans l'étude des protocoles de transferts de fichiers standards existants, puis en un second temps,dans la mise en oeuvre d'un de ces protocoles basée sur une couche transport telle que définie par les normes ISO.

L'étude des protocoles vise à définir un modèle général. En effet, les protocoles, parce que complexes et très rigoureux dans leurs définitions, présentent de grande difficultés delecture et de compréhension; ils sont en outre très longs (plus d'une centaine de pages). De plus, chaque protocole a sa terminologie propre. L'intérêt d'un modèle général est donc de rassembler les concepts communs et de les présenter d'une manière concise. Il suffit alors d'une correspondance terminologique pour évaluer leurs avantages et inconvénients respectifs et les comparer entre eux. Le lecteur trouvera peut-être ardus les deux premiers chapitres de ce travail. Qu'il veuille bine m'en excuser car il est difficile de décrire avec rigueur des normes et des protocoles sans employer un style 'protocolaire'.

Le but de la mise en oeuvre est de maîtriser les différents concepts et de donner un aperçu des difficultés de l'implémentation. L'idéal aurait été de pouvoir réaliser un logiciel assurant la communication avec d'autres ordinateurs conformes au protocole choisi. Une telle réalisation pouvait

s'imaginer entre deux ordinateurs des Facultés, mais, vu l'absence de moyens physiques, et le peu de temps qu'il me restait, il m'a semblé préférable de me contenter d'une maquette simulant le transfert de fichiers suivant le protocole choisi. Cependant, cette maquette, toute limitée qu'elle soit, m'a permis de rencontrer les objectifs de la mise en oeuvre c'est-à-dire la maîtrise des concepts et l'évaluation des difficultés. En outre, l'architecture de la maquette est telle que le processus implémentant le protocole peut être réutilisé dans une implémentation complète; il suffit de définir les couches inférieures et de réécrire éventuellement l'interface avec l'utilisateur.

0.3. Plan du mémoire

Comme expliqué ci-dessus, la première partie porte sur l'étude des protocoles. Le premier chapitre rappelle les concepts du Modèle de Référence défini par l'I.S.O. et dressant le contexte de la normalisation des Protocoles de Transfert de Fichiers. Le deuxième chapitre décrit le Modèle Général de Protocole de Transfert de Fichiers que j'ai défini, tandis que le chapitre 3 donne trois exemples de protocoles en établissant le lien avec le Modèle Général et en les comparant entre eux.

La deuxième partie concerne la mise en oeuvre: le chapitre 4 décrit les objectifs et les concepts de la maquette, tandis que le chapitre 5 traite davantage des problèmes d'implémentation. Les chapitres 4 et 5 sont courts car j'ai trouvé inutile d'ennuyer le lecteur avec les détails de l'implémentation puisque en annexe se trouve la description complète de l'architecture de la maquette qui constitue un des résultats importants du mémoire.

PARTIE 1

ETUDE

Chapitre 1 : NORMES ISO

Ce chapitre décrit le Modèle de Référence produit par l'ISO, ou International Standard Organisation (Organisation Internationale de Normalisation). Il introduit ainsi la terminologie, les concepts et les conventions de service utilisés tout au long de ce mémoire. La description complète de ce qui suit se trouve dans la norme ISO/IS7498 traitant du modèle de référence [ISO 1] et dans l'Annexe de la norme ISO/DIS8072 traitant de la couche transport [ISO 2].

1.1. Objet et Domaine d'Application

1.1.1. Pourquoi cette norme ?

L'objectif de la Norme Internationale Modèle de Référence de l'Interconnexion des Systèmes Ouverts est de fournir une base commune de coordination pour l'élaboration de normes portants sur l'interconnexion des systèmes, tout en permettant de situer les normes existantes par rapport à ce Modèle de Référence dans son ensemble. Celui-ci est conçu de manière suffisamment souple pour s'adapter aux progrès technologiques et à l'extension des demandes des utilisateurs.

L'expression Interconnexion de Systèmes Ouverts, notée OSI (de l'anglais Open Systems Interconnection), qualifie une famille de normes d'échanges d'informations entre systèmes qui sont "ouverts" à cet effet les uns aux autres du fait de leur utilisation commune des normes appropriées.

Le fait qu'un système soit ouvert à d'autres n'implique aucune réalisation ou technologie particulière de systèmes, ni de moyens d'interconnexion particuliers, mais exprime l'acceptation mutuelle de normes appropriées, ainsi que la conformité à ces dernières.

1.1.2. Normes OSI

D'autres travaux sont en cours au sein de l'Organisation Internationale de Normalisation, visant à l'élaboration de normes OSI portant sur les domaines suivants:

a) protocoles de terminaux virtuels;
b) protocoles de transfert, d'accès et de manipulation de fichiers;
c) protocoles de transfert et de manipulation de travaux;
d) services et protocoles de la Couche Session;
e) services et protocoles de la Couche Transport;
f) services et protocoles de la Couche Réseau;

g) services et protocoles de la Couche Liaison de Données;
h) services et protocoles de la Couche Physique;
j) protocole de gestion de l'OSI;

Les trois premiers points sont relatifs aux Couches Application et Présentation du Modèle de Référence; l'expérience acquise au cours de l'élaboration des normes correspondantes doit conduire à des normes générales de services et de protocoles de la Couche Présentation.

Actuellement, seul le Modèle de Référence constitue une norme ISO (IS). Les niveaux session et transport en sont au niveau de projet international (DIS) et la couche présentation est à l'état d'avant-projet (DP). Au niveau application seul le protocole pour le transfert des fichiers (FTAM) est au niveau d'avant-projet. En ce qui concerne les trois couches inférieures, le protocole X.25 du C.C.I.T.T. est la norme généralement admise.

1.1.3. L'environnement de l'Interconnexion des Systèmes Ouverts

Définitions

système : ensemble comprenant un ou plusieurs ordinateur(s), le logiciel associé, des périphériques, des terminaux, des opérateurs humains, des processus physiques, des moyens de transfert d'informations, etc... et constituant un tout autonome capable d'effectuer des traitements et/ou des transferts d'informations.

système_ouvert : système dont les communications avec d'autres systèmes sont effectuées conformément aux normes OSI (d'Interconnexion de Systèmes Ouverts). Sauf indication contraire, le terme système sera utilisé comme synonyme de système ouvert.

processus_d'application : élément d'un système effectuant un traitement d'information pour une application particulière.

Description

L'OSI concerne les échanges d'informations entre systèmes ouverts (et non le fonctionnement interne de chacun des systèmes ouverts). L'OSI ne concerne pas seulement le transfert d'informations entre systèmes, c'est-à-dire la transmission, mais également l'aptitude de ces systèmes à unir leurs efforts pour réaliser une tâche commune (répartie). En d'autres termes, l'OSI concerne la coopération entre systèmes, telle qu'elle est impliquée par l'expression "interconnexion de systèmes".

Les processus d'application sont des traitements effectués sur ces systèmes. Ils peuvent être des processus manuels, des processus informatisés ou des processus physiques.

1.2. Architecture de l'OSI

1.2.1. Principes de la structuration en couches

Définitions

sous-système_(N) : élément d'une division hiérarchique d'un système n'ayant d'interaction qu'avec les éléments des niveaux immédiatement supérieur et inférieur de cette division.

couche_(N) : subdivision de l'architecture d'OSI, constituée de sous-systèmes de même rang (N).

entité_(N) : élément actif d'un sous-système (N).

entités_homologues : entités appartenant à une même couche.

sous-couche : subdivision d'une couche.

service_(N) : capacité que possèdent la couche (N) et les couches inférieures à celle-ci, fournie aux entités (N+1) à la frontière entre la couche (N) et la couche (N+1).

fonction_(N) : élément de l'activité d'entités (N).

point_d'accès_à_des_services_(N) : frontière entre un service (N) et l'entité (N+1) utilisant ce service.

Description

La technique de structuration de base du Modèle de Référence d'OSI est la structuration en couches. Selon cette technique, on considère que chaque système est logiquement composé d'un ensemble ordonné de sous-systèmes qu'on représente pour la commodité dans l'ordre vertical, comme indiqué sur la figure 1.1. Les sous-systèmes adjacents communiquent à travers leur frontière commune. L'ensemble des sous-systèmes de même rang (N) constitue la couche (N) du Modèle de Référence d'OSI. Un sous-système (N) est constitué d'une ou plusieurs entités (N).

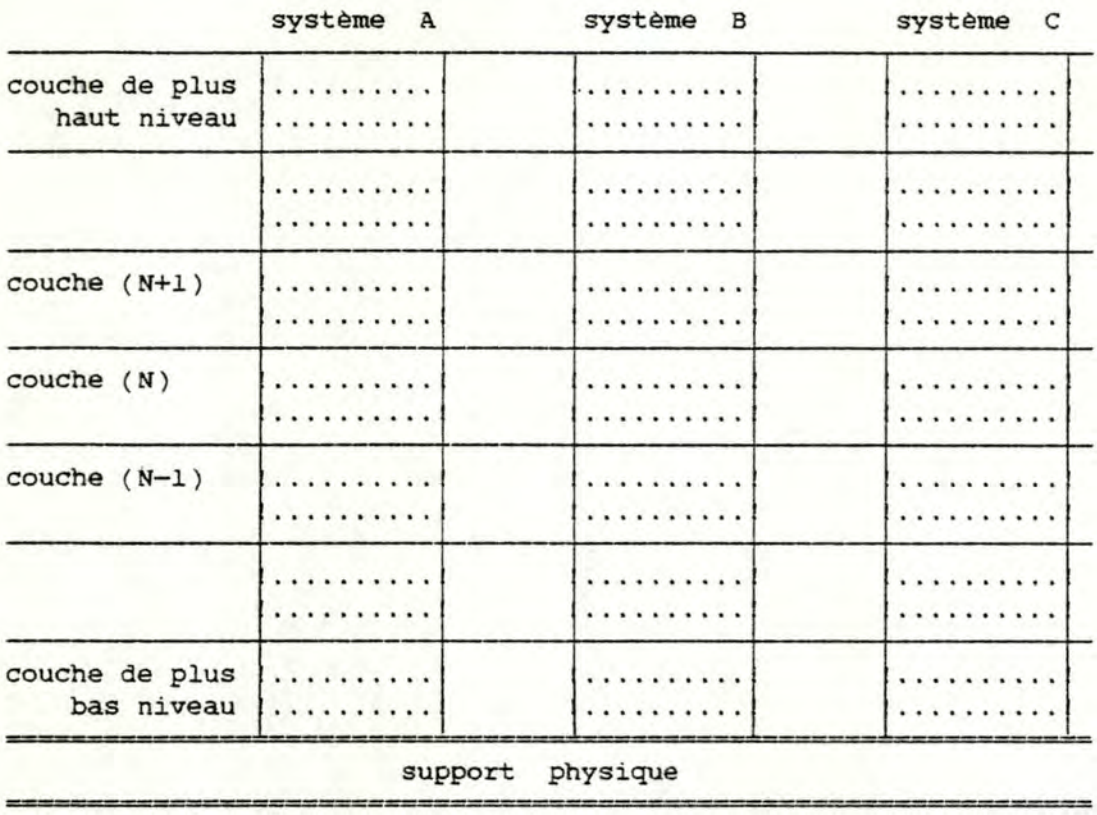


Figure 1.1 - Structuration en couches de systèmes ouverts coopérants.

1.2.2. Communication entre entités homologues

Définitions

connexion_(N) : Association établie par la couche (N) entre deux ou plusieurs entités (N+1) pour le transfert de données.

extrémité_de_connexion_(N) : Terminaison d'une connexion (N) en un point d'accès à des services (N).

entités_correspondantes : entités (N) reliés par une connexion (N).

protocole_(N) : ensemble de règles et de formats (sémantiques et syntaxiques) déterminant les caractéristiques de communication des entités (N) lorsqu'elles effectuent les fonctions (N).

contrôle_de_flux : Fonction contrôlant le flux de données au sein d'une couche ou entre couches adjacentes.

transmission_de_données_(N) : Service (N) transportant des unités de données du service (N) d'une entité (N+1) à une ou plusieurs entités (N+1) via des connexions (N).

transmission_duplex_(N) : Transmission de données (N) dans les deux sens à la fois.

transmission_semi-duplex_(N) : Transmission de données (N) dans un sens ou dans l'autre, le sens de transmission étant contrôlé par une entité (N+1).

transmission_simplex_(N) : Transmission de données (N) dans un seul sens fixé à l'avance.

réinitialisation : Fonction mettant les entités (N) correspondantes en un état défini à l'avance, avec possibilité de perte ou de duplication de données.

Description

Pour pouvoir échanger des informations entre deux ou plusieurs entités (N+1), il faut établir entre elles une association dans la couche (N), en suivant un protocole (N). Cette association est appelée une connexion (N).

Les connexions (N) sont établies par la couche (N) entre au moins deux points d'accès à des services (N). La terminaison d'une connexion (N) à un point d'accès à des services (N) est appelée extrémité de connexion (N). Une connexion comportant plus de deux extrémités de connexion est appelée connexion multipoint. Des entités (N) reliées par une connexion sont appelées entités (N) correspondantes.

Les entités correspondantes synchronisent leur dialogue à partir d'un point bien défini. Suite à des perturbations de la connexion, il peut y avoir une perte de synchronisation entre les entités. Dans ce cas, une réinitialisation est nécessaire.

1.3. Description des couches OSI

1.3.1. Couches spécifiques

Le Modèle de Référence comporte sept couches:

a) la Couche Application (couche 7);
b) la Couche Présentation (couche 6);
c) la Couche Session (couche 5);
d) la Couche Transport (couche 4);
e) la Couche Réseau (couche 3);
f) la Couche Liaison de Données (couche 2);
g) la Couche Physique (couche 1);.

Ces couches sont représentées Figure 1.2. La couche de niveau le plus élevé est la Couche Application. Elle contient les entités d'application qui coopèrent dans l'environnement OSI. Les couches de niveaux inférieurs fournissent les services par l'intermédiaire desquels les entités d'application coopèrent.

Les couches 1 à 6, ainsi que le support physique OSI contribuent à l'élaboration progressive des services de communication. A chacune des frontières séparant les couches successives, correspond un niveau d'élaboration des services pour lequel est définie une norme de services OSI. Le fonctionnement des couches est, quant à lui, régi par des normes de protocoles OSI.

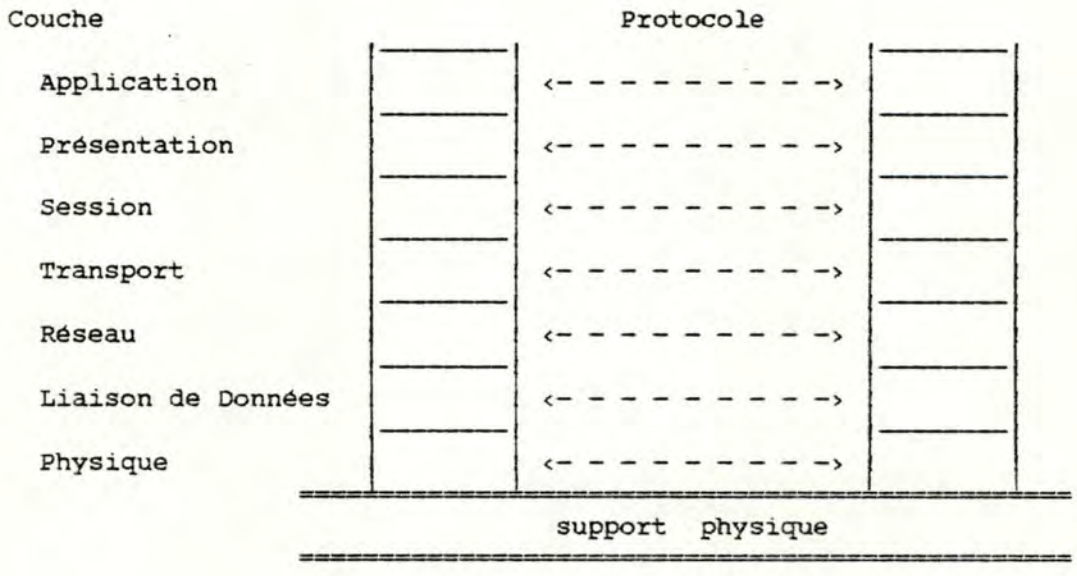


Figure 1.2 - Modèle de Référence à sept couches

1.3.2. Principes appliqués pour déterminer les sept couches du Modèle de Référence

Pour déterminer les sept couches du Modèle de Référence, les principes suivants ont été appliqués. Ceux-ci semblent également utiles pour orienter des décisions ultérieures dans l'élaboration de normes OSI.

- P1 : Ne pas créer un nombre trop grand de couches, afin de simplifier l'intégration des différentes couches.

- P2 : Créer des frontières là où la description des interfaces est concise, et le nombre d'interactions au travers de cette frontière est réduit.

- P3 : Créer des couches séparées pour prendre en charge des fonctions qui diffèrent manifestement par le traitement effectué ou par la technologie mise en jeu.

- P4 : Regrouper des fonctions similaires dans une même couche.

- P5 : Choisir des frontières là où l'expérience passée a prouvé qu'un tel choix donnait de bons résultats.

- P6 : Créer une couche avec des fonctions faciles à repérer, de telle sorte que la conception de la couche puisse être entièrement revue et ses protocoles modifiés de façon importante sans avoir à modifier les services attendus des couches adjacentes ou fournis à celles-ci.

- P7 : Créer une frontière là où il peut être utile, à un moment ou à un autre, de normaliser l'interface correspondante.

- P8 : Créer une couche là où il y a besoin de distinguer un niveau d'abstraction de manipulation de données.

- P9 : Permettre d'effectuer dans une couche des modifications de fonctions ou de protocoles sans affecter les autres couches.

- P10 : Pour chaque couche, ne créer des frontières qu'avec les couches des niveaux immédiatement supérieurs et inférieurs.

Notez le rapport avec les principes vus au cours de méthodologie de développement de logiciel (AVL): Le principe de cacher de l'information se retrouve dans les principes P8 et P9, le principe d'avoir une forte cohésion interne dans les modules est couvert par les principes P3 et P4 et le principe de maintenir un faible degré de couplage entre les modules est repris par les principes P2 et P6.

1.3.3. Rôle des différentes couches

a. La Couche Application

En tant que couche la plus élevée du Modèle de Référence, la Couche Application donne au processus d'application le moyen d'accéder à l'environnement OSI. Le rôle de la Couche Application est de servir de fenêtre entre entités d'application correspondantes utilisant l'OSI pour échanger des informations significatives. Chaque processus d'application est vu par son homologue à travers une entité d'application. Chaque fois que des communications ont lieu dans l'environement OSI, c'est au travers de la Couche d'Application que tous les paramètres nécessaires concernant un processus d'application sont portés à la connaissance de l'environement OSI.

b. La Couche Présentation

La Couche Présentation se charge de la représentation des informations que des entités d'applications se communiquent, ou auxquelles elles se réfèrent au cours de leur dialogue. Elle fait en sorte qu'une représentation commune des informations soit utilisée entre les entités d'application. Chaque entité d'application peut utiliser n'importe quelle syntaxe, car la Couche Présentation assure la transformation entre cette syntaxe et la syntaxe commune nécessaire à la communication entre entités d'application.

c. La Couche Session

La Couche Session fournit aux entités de présentation coopérantes les moyens nécessaires pour organiser et synchroniser leur dialogue et pour gérer leur échange de données. A cet effet, la Couche Session fournit les services nécessaires à l'établissement d'une connexion de session entre deux entités de présentation et la prise en charge harmonieuse des interactions d'échanges de données.

d. La Couche Transport

Le service de Transport assure un transfert de données transparent entre entités de session en les déchargeant complètement des détails d'exécution d'un transfert de données fiable et d'un bon rapport qualité/prix. La Couche Transport optimise l'utilisation des services de réseau disponibles afin d'assurer au moindre coût les performances requises par chacune des entités session. Tous les protocoles définis dans cette couche ont une signification de bout-en-bout.

e. La Couche Réseau

La Couche Réseau fournit d'une part les moyens d'établir, de maintenir et de libérer des connexions de réseau entre des systèmes contenant des entités d'application devant communiquer, et d'autre part les moyens fonctionnels et procéduraux d'échanger entre entités de transport des unités de données du service réseau sur des connexions de réseau. Elle rend les entités de transport indépendantes des problèmes de routage et de relais liés à l'établissement et au fonctionnement d'une connexion de réseau donnée. Elle masque aux entités de transport la façon dont des resources des couches de niveaux inférieurs, comme les liaisons de données, sont utilisées pour fournir des connexions de réseau.

f. La Couche Liaison de Données

La Couche Liaison de Données fournit les moyens fonctionnels et procéduraux nécessaires à l'établissement, à la maintenance et à la libération des connexions de liaison de données entre entités de réseau, ainsi qu'au transfert des unités de données du service de liaison de données. Elle détecte et corrige, dans la mesure du possible, les erreurs pouvant se produire dans la Couche Physique.

g. La Couche Physique

La Couche Physique fournit les moyens mécaniques, électriques, fonctionnels et procéduraux nécessaires à l'activation, au maintien et à la désactivation des connexions physiques destinées à la transmission de bits entre entités de liaison de données. Une connexion physique peut mettre en jeu plusieurs systèmes intermédiaires, relayant chacun la transmission des bits dans la Couche Physique. Les entités de la Couche Physique sont interconnectées au moyen d'un support physique.

1.4. Conventions de Services

La description des services et des protocoles de chaque couche est basée sur un certain nombre de conventions appelées Conventions de Service. Ces conventions seront utilisées principalement dans la deuxième partie de ce mémoire, pour définir les interfaces entre les différentes couches, permettant ainsi une compréhension plus aisée par rapport au modèle général OSI.

1.4.1. Définitions

service_d'une_couche : Service fourni par une couche du Modèle de Référence.

utilisateur_d'un_service : Représentation abstraite de la totalité des entités d'un système déterminé qui utilisent un service.

fournisseur_d'un_service : Machine abstraite constituant un modèle de comportement de la totalité des entités fournissant le service telles que vues par un utilisateur du service.

primitive : Elément abstrait, et indépendant de la réalisation du modèle, d'une interaction entre un utilisateur d'un service et le fournisseur de ce service. On distingue quatre types de primitives: les primitives de requête, les primitives d'indication, les primitives de réponse et les primitives de confirmation.

requête : Primitive émise par un utilisateur de services pour demander l'exécution d'une certaine procédure.

indication : Primitive émise par le fournisseur de services soit pour demander l'exécution d'une certaine procédure, soit pour indiquer que cette exécution a été demandée par un utilisateur du service.

réponse : Primitive émise par un utilisateur d'un service pour achever, au niveau d'un point d'accès particulier à ce service, une certaine procédure préalablement lancée par une indication reçue à ce point d'accès au service.

confirmation : Primitive émise par un fournisseur d'un service pour achever, au niveau d'un point d'accès particulier à ce service, une certaine procédure préalablement lancée par une requête reçue à ce point d'accès au service.

1.4.2. Description

Toutes les interactions sont définies entre deux utilisateurs du service situés dans la couche (N+1), auprès de points d'accès au service séparés. Les utilisateurs du service communiquent via le fournisseur du service se trouvant dans la couche (N), en utilisant tous les services des couches de niveaux inférieurs à la couche (N+1). Les informations sont échangées entre un utilisateur du service et le fournisseur du service à l'aide de primitives.

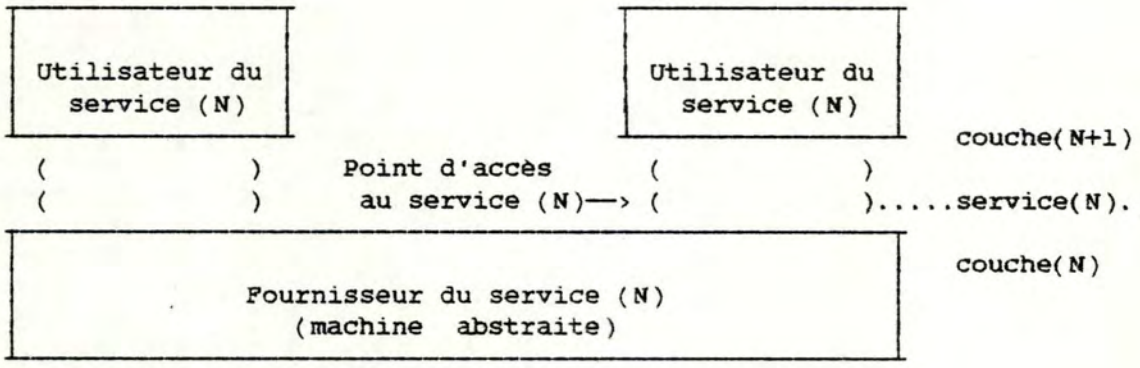


Figure 1.3 - Modèle d'un service

Chapitre 2 : MODELE GENERAL

Ce chapitre décrit la terminologie et les concepts spécifiques aux protocoles et aux services de transfert de fichiers. Il servira de modèle aux protocoles étudiés dans les chapitres suivants.

2.1. Terminologie

donnée : toute représentation à laquelle est ou peut être attribué une signification (e.g. des caractères).

information : une combinaison de données et la signification qui y est attachée.

fichier : une collection d'informations nommée de façon non ambigüe et ayant un ensemble commun d'attributs.

système_de_fichiers : une collection organisée de fichiers, y compris leurs attributs et leur nom, se situant dans un système ouvert particulier.

transfert_de_fichier : une fonction qui transfère des fichiers entiers entre systèmes.

accès_à_un_fichier : la lecture, la modification ou le remplacement d'une partie du contenu d'un fichier.

gestion_de_fichier : la création et l'effaçage d'un fichier, et la lecture ou la manipulation des attributs associés à un fichier en tant que tout.

service_de_transfert_de_fichier : service fourni par la couche application assurant le transfert de fichiers entre systèmes de fichiers.

protocole_de_transfert_de_fichier_(ou_FTP) : protocole utilisé par les entités de la couche application fournissant le service de transfert de fichier pour assurer ce service. On y trouvera également les protocoles utilisés par les entités des couches présentation et session qui contribuent à l'élaboration du service de transfert de fichiers.

attribut : un élément de donnée, ayant une signification déterminée, ainsi que l'ensemble des valeurs qu'il peut prendre.

attributs_d'un_fichier : le nom et d'autres propriétés identifiables d'un fichier. Les attributs se réfèrent au fichier comme un tout ou à sa structure interne. La même valeur d'un attribut de fichier est observée à un moment donné par tous les utilisateurs du service, même si plus d'un utilisateur est actif à ce moment là.

attributs_d'activité : Les attributs décrivants l'activité d'utiliser le service de transfert de fichier. Les attributs sont locaux à une connexion du service de transfert de fichier.

phase : une période de temps pendant laquelle les messages de protocoles ont un but particulier et une représentation constante.

régime : une période de temps pendant laquelle une certaine collection d'informations contextuelles est valide.

initiateur : entité qui prend l'initiative de la demande de transfert.

répondeur : entité qui reçoit la demande de transfert et répond à celle-ci.

expéditeur : entité qui se trouve du côté du système de fichiers qui mémorise initialement le fichier et qui l'envoie.

destinataire : entité vers qui le fichier est tranféré.

2.2. Objectifs des protocoles de transfert de fichier

Le but de la standardisation d'un Service de Transfert de Fichier est de permettre l'interconnexion ouverte d'utilisateurs qui veulent transférer des fichiers entre des systèmes qui se comportent comme s'ils stockaient des fichiers. Tout ce qui apparaît comme un système dans le but de l'interconnexion, et qui se conforme aux protocoles de transfert de fichiers spécifiés dans le rôle de fournisseur d'un système de fichiers, est considéré comme fournissant un système de fichiers.

Un Protocole de Transfert de Fichier ou FTP définit les services et spécifie les protocoles pour le transfert de données, sous forme de fichiers, entre ordinateurs à travers un réseau utilisant des moyens de télécommunication. Il ne s'applique donc pas à l'échange de fichiers par des moyens physiques tels que bandes ou disques. De même, le but n'est pas de fournir la coordination de différentes zones de stockages de fichiers pour fournir une apparence unique et unifiée, bien qu'un tel support est une extension future possible.

Parce qu'un FTP est destiné à être implémenté par des utilisateurs très divers, il doit être construit de manière très flexible; en particulier, des implémentations très simples doivent être possibles. En concevant et spécifiant le protocole, un certain nombre de considérations d'importance à court et à long terme doivent être prises en compte. Ils incluent:

- La possibilité de différence de sophistication et de taille entre les sytèmes locaux et distants.
- La possibilité d'augmentation en sophistication des implémentations.
- La nécessité de travailler avec des sytèmes existants. Il n'est pas question d'effectuer des changements majeurs dans les O.S. existants.

Il existe peu de tels systèmes bien que certaines architectures de réseaux permettent de transférer des fichiers entre des équipements du même constructeur. Une facilité plus commune est la possibilité de transférer des ensembles "anonymes" d'informations, en simulant un système spécifique, tel qu'un terminal interactif. Le désavantage de tels systèmes cependant est que chaque transfert doit faire des hypothèses sur la nature et la taille du transfert à effectuer, et il n'y a pas moyen de modifier le transfert en conséquence ou de le définir en termes des données en question.

Un FTP suppose en général la disponibilité d'un service de transport, c'est-à-dire un service implémentant les quatres couches inférieures du modèle de référence, qui fournit un moyen de communication indépendant du réseau. Les exigences du FTP vis-à-vis de ce service de transport sont les suivantes:

a. L'exigence de base du FTP est la transmission dans chaque sens d'un flux d'octets entre les deux entités qui implémentent le service de transfert de fichier. Chaque flux doit être synchronisé aux deux bouts, de façon à être décodé correctement en une suite d'enregistrements, et la paire de flux doit aussi être synchronisée. Le service de transport assure la fourniture d'un point de départ bien défini pour chaque flux. Il doit aussi réguler le flux de données de façon à limiter les demandes au destinataire.

b. Le service de transport est responsable de la détection d'erreurs et du recouvrement d'erreurs dépendantes du réseau. Idéalement, le FTP préférerait éviter tout problème d'erreurs, car le service de transport devrait être 'error free'. De façon plus réaliste, il est demandé que les erreurs non détectées soient suffisament rares (suivant une certaine 'qualité de service' choisie par l'utilisateur) et que les erreurs détectées mais non corrigées par le service de transport soient communiquées aux deux processus du FTP d'une manière qui permet la réinitialisation. Un tel événement est appelé un_reset du service de transport. Après un reset du service de transport, les deux parties sont synchronisées, et il n'y a plus de données en transit. Cependant, quelques données envoyées précédemment peuvent être perdues. Si un reset ne peut être donné les deux parties doivent être averties que la communication s'est interrompue de manière irrecouvrable et la connexion fermée.

c. Le service de transport peut permettre à un processus d'effectuer une commande de resynchronisation (un reset), qui a le même effet qu'une erreur détectée du service transport.

d. Le service de transport doit permettre à un processus de fermer la connexion à tout moment. Un échec catastrophique du service de transport peut aussi résulter en une fermeture spontanée. La fermeture peut provoquer la perte des données encore en transit.

2.3. Procédure générale de transfert

Le service de transfert de fichier et le protocole qui le supporte créent, en une série d'étapes, un environement de travail dans lequel les activités désirées par l'utilisateur peuvent prendre place. Le dialogue doit, tour à tour:

- permettre à l'utilisateur et au fournisseur de système de fichiers d'établir l'identité du partenaire,
- identifier le fichier qui est nécessaire,
- établir les attributs décrivant la structure du fichier qui doit être accédé à cette occasion,
- engager le transfert de données.

Ces étapes mettent en place différentes parties du contexte opérationel. La période pour laquelle une partie de l'information contextuelle est valide est appelée un régime . Lorsque progressivement plus de détails contextuels sont établis, une imbrication des régimes correspondants est construite.

Une période de temps pendant laquelle les échanges du protocole ont un but particulier, tel que établir, utiliser ou abandonner un contexte opérationel est appelé une phase . Pour chaque phase, un ensemble de messages valides est défini en termes d'états de transition. A chaque instant une entité est dans une phase précise; les phases ne peuvent être imbriquées.

Les phases sont introduites dans les paragraphes suivants, dans l'ordre dans lequel elles apparaîtraient durant une utilisation typique du service de transfert de fichier. La figure 2.1 montre l'imbrication des régimes.

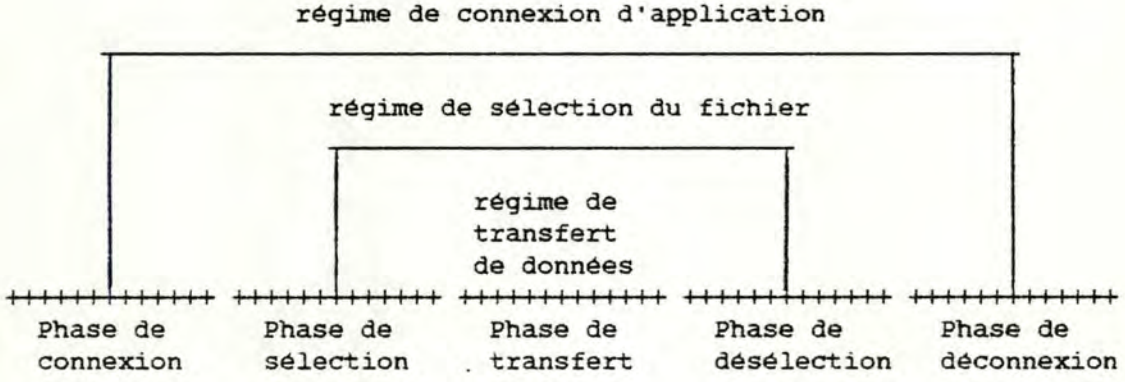


Figure 2.1 Phases et imbrication des régimes

2.3.1. Phase d'établissement de connexion d'application

Cette phase n'est pas spécifique aux protocoles de transfert de fichiers, mais est une phase commune qui établit le régime de connexion d'application; elle établit également l'autorisation et les informations de comptabilité nécessaires pour effectuer des opérations sur le système de fichiers.

2.3.2. Phase de sélection du fichier

Cette phase démarre l'activité du service de transfert de fichiers et établit le régime de sélection du fichier; elle identifie ou crée un fichier unique auquel les opérations des phases suivantes vont s'appliquer. La sélection continue jusqu'à ce qu'elle soit interrompue explicitement par une phase de désélection ou une phase de terminaison. Les opérations effectuées dans les phases suivantes se réfèrent donc au fichier sélectionné et ne contiennent pas d'identification explicite du fichier.

Cette phase détermine également les propriétés du fichier et les détails du format dans lequel il doit être transféré, par la sélection et la négociation des valeurs d'attributs. La phase de sélection peut entrainer la création d'un nouveau fichier avec les propriétés spécifiées. La sélection du fichier peut se faire en termes d'un nom de fichier ou elle peut être exprimée comme une série de contraintes sur les autres attributs du fichier dont les valeurs servent à identifier le fichier demandé. Une demande de sélection de fichier peut inclure des informations de contrôle spécifiques aux droits d'accès au fichier.

2.3.3. Phase de transfert de données

Cette phase inclut le transfert proprement dit et les échanges administratifs pour le réguler. Elle est définie en termes de l'expéditeur et du destinataire du fichier, comme établit durant la phase de sélection, plutôt qu'en termes des processus initiateurs et répondeurs.

2.3.4. Phase de désélection du fichier

La phase de désélection abandonne le contexte opérationel établi par la phase de sélection. Elle arrête le dialogue et indique l'état final du transfert. L'activité du service de transfert de fichier est alors terminée.

La phase de désélection peut inclure des séquences standards de comptabilisation. Elle laisse effective toute authentification ou identification de compte établi avant la phase de sélection du fichier. La phase de désélection peut entrainer la suppression du fichier sélectionné.

2.3.5. Phase de terminaison de connexion d'application

Cette phase dissout l'association entre l'utilisateur et le fournisseur du système de fichiers, abandonnant tout régime d'authentification ou de comptabilisation. Il n'y a donc plus de contexte opérationel restant; l'activité de service de transfert de fichier est terminée.

2.4. Concepts de système virtuel de fichiers et attributs

2.4.1. Le besoin d'un modèle de système de fichiers

La façon dont un système de fichiers est implémenté varie considérablement parmi les systèmes existants. Les différents systèmes sont d'une grande variété dans la description de la mémorisation de données et des moyens d'y accéder. Un FTP est basé sur le présupposé que tout fichier ou système de fichier peut être décrit dans les termes d'une représentation standard unique. Cette représentation, appelée le Système Virtuel des Fichiers, peut alors être utilisée par le protocole pour exprimer les transfert de fichier, chaque participant faisant la correspondance entre les descriptions standards et les resources locales. Ce modèle commun à utiliser dans la description des fichiers et de leurs attributs doit donc être établi avant que les services, les protocoles et les procédures de transfert de fichiers puissent être utilisés d'une façon ouverte.

La force du protocole vient de sa capacité de définir n'importe quel fichier en termes d'un ensemble de caractéristiques standards, et d'indiquer les moyens par lesquels le fichier devrait être transféré et mémorisé. Une telle définition permet d'absorber les différences de styles et de spécifications en une fonction locale de conversion, et ainsi n'importe quel système peut communiquer avec un autre système en termes qui peuvent être mutuellement compris. En masquant les détails d'un O.S. local pour l'utilisateur d'une connexion externe, le besoin de modifier le système est réduit et donc aussi le coût initial pour travailler en système ouvert.

Exprimer le dialogue en termes du Système Virtuel de Fichiers permet l'interconnexion d'une large gamme de systèmes de complexités différentes. La définition d'un nombre de sous-ensembles optionnels dans la définition du système virtuel de fichiers rend possible des façons de travailler où un système plus simple communique avec un système plus sophistiqué; par exemple, un ordinateur complexe peut communiquer avec la mémoire auxiliaire d'un terminal intelligent, ou avec un périphérique à enregistrement par blocs tel qu'un lecteur de bandes ou de disques. Le protocole ne sert pas seulement à concilier les différences de styles entre des catégories similaires de mémoires, mais aussi résout les différences de sophistication ou de type de mémoire. Cependant il permet à une implémentation d'insister sur des exigences à propos de n'importe qu'elle propriété nécessaire, telles que les niveaux de sécurité.

2.4.2. Conversion dans la définition du système virtuel de fichiers

Pour utiliser les services et les protocoles de transfert de fichiers, une implémentation doit relier les éléments de la définition du système virtuel de fichiers au système de stockage local. La façon dont un fichier est traduit de ou vers le Système Virtuel de Fichiers variera considérablement de système à système, mais les échanges d'informations seront toujours exprimés en termes des attributs standards. Un implémenteur utilisant ce standard met au point une correspondance entre d'une part les actions et les éléments de données représentant le contenu du fichier et les attributs du système virtuel de fichiers dans l'Environement d'Interconnexion de Systèmes Ouverts (OSIE) et d'autre part les ressources de l'Environnement du Système Local (LSE).

Lorsque les messages du protocoles sont reçus par l'entité d'application, ils sont interprétés en termes des correspondances établies localement entre les éléments de données et les aspects de l'environnement du système local liés au stockage d'information. La conversion est donc un ensemble de correspondances établies par le concepteur du système local.

2.4.3. La forme du système virtuel de fichiers

Le système virtuel de fichiers est décrit par un ensemble d'attributs identifiants ou définissants les fichiers à transférer.

La définition du Système Virtuel de Fichiers forme un schéma pour la description des informations mises en fichiers. Dans cette description un fichier est une entité ayant:

- un nom de fichier, qui permet de le référencer sans ambiguïté,
- d'autres attributs descriptifs qui expriment les propriétés communes du fichier telles que la taille, les informations de budget, l'historique, les protections d'accès, etc.,
- des attributs décrivant la structure logique et les dimensions des données stockées dans le fichier,
- toute donnée formant le contenu du fichier.

Ce sont toutes des informations pouvant être observées par n'importe quel initiateur autorisé. Si deux observateurs font la même demande à propos de ces aspects d'un fichier, ils obtiendront la même information sur ses propriétés. Certains de ces attributs (comme le nom du fichier) ont des valeurs constantes durant toute la vie du fichier, et sont appelés les attributs_de_stockage.

Il y a aussi des attributs décrivant les relations entre le fichier et un initiateur particulier, concernant des choses telles que l'identification, l'avancement d'une activité, les coûts cumulés, etc.; il y a un ensemble distinct de valeurs pour ces attributs pour chaque activité en cours. Cet ensemble est créé lorsque la connexion au système de fichiers est établie, et

détruit lorsqu'elle est coupée. D'autres attributs spécifient les conditions du transfert et peuvent ne plus avoir de signification une fois que le transfert est terminé; ils sont appelés les attributs_de_transfert.

Finalement, il y a les attributs_d'information qui ne sont pas négociés mais passent des informations supplémentaires. Par exemple, certains attributs du fichier décrivent la structure du contenu du fichier. Cette structure est préservée durant la vie du système. Cependant, tous les utilisateurs accédant au fichier ne sont pas concernés par ceux-ci en toute généralité. Ils peuvent, par exemple, avoir besoin d'accéder à un fichier hiérarchique complexe comme s'il était plat pour construire des résumés, ou bien il n'est peut-être pas nécessaire d'accéder à chaque fois, isolément, aux plus petites unités de structure d'un fichier. En plus des attributs utilisés dans la création et la gestion d'un fichier pour décrire la structure permanente du fichier, il y a, pour chaque activité, un attribut du contexte d'accès indiquant le sous-ensemble de la structure du fichier actuellement utilisé.

2.4.4. Négociation des valeurs d'attributs

La phase de sélection du fichier négocie les valeurs d'attributs appropriées au transfert. Chaque paramètre réfère un attribut et contient soit une valeur et un opérateur relationnel associé, ou une indication du motif pour lequel aucune valeur n'est donnée (soit attribut inconnu, soit aucune valeur disponible). Lorsqu'un paramètre dans la requête de transfert a une valeur, l'opérateur relationnel indique la contrainte que l'initiateur impose à la liberté du répondeur de spécifier une valeur différente sur la réponse. Si pour un certain attribut particulier, la requête de transfert ne donne pas de paramètre, ou si le paramètre ne contient pas de valeur, alors le répondeur peut spécifier toute valeur souhaitée dans la réponse. Si ni l'initiateur (dans la requête de transfert) ni le répondeur (dans la réponse) ne fournit une valeur pour un attribut, la valeur par défaut est supposée. Dans ces circonstances, s'il n'y pas de valeur par défaut, aucune hypothèse ne peut être faite sur la valeur utilisée par l'autre partie.

Pour les attributs qui sont négociables, les valeurs sont contraintes par les capacités de l'implémentation et, plus spécialement pour certains attributs, par ce que l'utilisateur ou l'implémenteur exige pour un essai particulier de transfert. Ces contraintes sont référées comme les capacités ou les exigences, respectivement, des processus FTP appropriés. Les capacités et exigences pour un paramètre particulier peuvent être distinguées dans la requête de transfert par l'opérateur relationel utilisé. Une capacité sera généralement une borne supérieure sur la gamme de valeurs possibles (ou une affirmation qu'il n'y pas de limite supérieure), tandis qu'une exigence sera généralement une borne inférieure. Si une valeur spécifique est nécessaire, l'opérateur d'égalité sera utilisé et la distinction entre capacités et exigences n'a pas d'importance.

2.5. Protocole de contrôle de transfert de fichier

2.5.1. Le contrôle de l'activité du fichier

Dans la section précédente, il est à noter que le protocole ne définit pas l'initiative qui provoque l'initiateur à démarrer un transfert de fichier. Le protocole s'occupe uniquement des informations qui permettent au répondeur d'envoyer ou de recevoir le fichier, pas des raisons de l'initiateur de demander un transfert. La fonction principale du protocole de contrôle sera de définir le transfert de fichier et ses résultats. Donc le protocole de contrôle pourrait être très similaire en structure aux parties correspondantes du FTP.

Un tel protocole est cependant rarement défini. En fait, si l'implémentation du FTP déplacera des fichiers à la demande d'un certain utilisateur du service de transfert de fichier, la façon dont un tel utilisateur spécifie ses besoins dépend en beaucoup de points de l'implémentation locale. Une grande variété dans la façon dont le service de transfert de fichiers est utilisé doit être prise en compte. Le service de transfert de fichier peut, par exemple, être réalisé comme:

- un utilitaire exécuté par un seul utilisateur;
- un programme d'application écrit par l'utilisateur;
- un sous-système servant une queue de demandes de plusieurs utilisateurs;
- un composant de l'O.S. interfacé au langage de contrôle de travaux local.

L'utilisateur du transfert de fichier peut également effectuer un protocole de niveau plus élevé étroitement lié à l'implémentation du transfert de fichiers. Dans tous ces cas, les messages du protocole utilisés dans le transfert seront les-mêmes.

2.5.2. Protocole de contrôle et protocole de transfert

La façon dont les activités du fichier sont contrôlées doit être expliquée pour clarifier les buts de ce standard. Considèrons le transfert de fichier; pour chaque transfert, il y a trois entités concernées: une entité qui prend l'intiative du contrôle, une entité qui accède au fichier virtuel source et une entité qui accède au fichier virtuel destination. Depuis le contrôleur, il y a deux flux d'informations (cfr figure 2.2):

1. des informations, concernant la spécification du fichier virtuel source et les contraintes sur la façon dont le transfert doit être fait, envoyées à l'entité accédant le fichier source,
2. des informations, concernant la spécification du fichier virtuel destination et les contraintes sur la façon dont le transfert doit être effectué, envoyées à l'entité accédant le fichier destination.

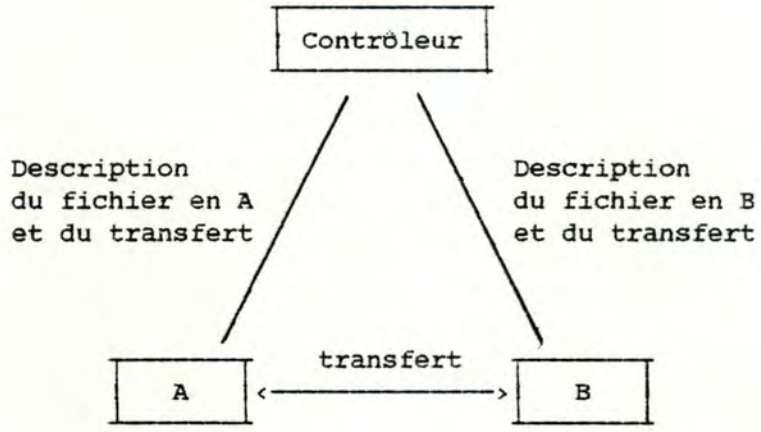


Figure 2.2. Flux depuis le contrôleur

Il n'est pas nécessaire que l'utilisateur et un des systèmes de fichier soient dans le même espace physique; ils pourraient être connectés par un quelconque moyen de télécommunication, donnant ainsi un transfert de tiers. Mais pour simplifier la coordination et le contrôle du transfert, il est supposé que le contrôleur conduira, en géneral, ces flux d'informations via un des deux systèmes de fichiers, qui agira comme son agent en effectuant le transfert (cfr Figure 2.3). Dans beaucoup de cas, cela viendra naturellement du fait que le contrôleur et un des sytèmes de fichiers seront dans le même système.

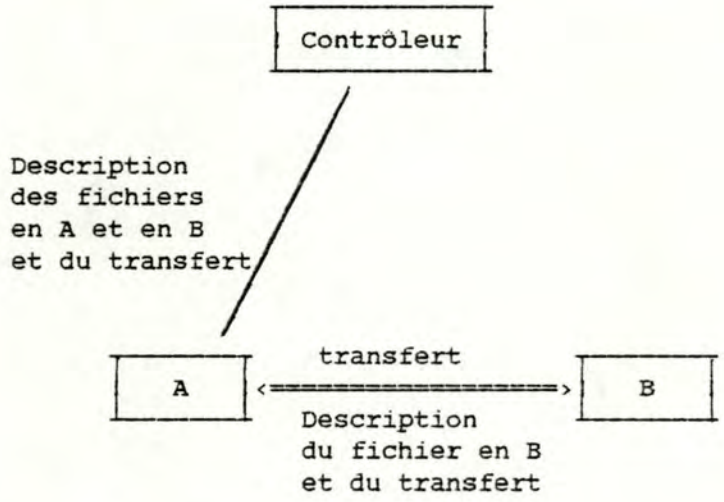


Figure 2.3. Flux dirigé via un agent

Il y a donc deux protocoles en fonction:

- un protocole de contrôle entre l'utilisateur et le système de fichiers qui agira comme son agent,
- un protocole de transfert entre le système de fichiers initiateur et un autre sytème de fichiers répondeur.

En décrivant le protocole de contrôle, le système de fichiers agent et le système de fichiers répondeur sont considérés comme une seule entité, tandis qu'en décrivant le protocole de transfert, l'utilisateur et le système de fichiers initiateur sont considérés comme une entité. (cfr figure 2.4)

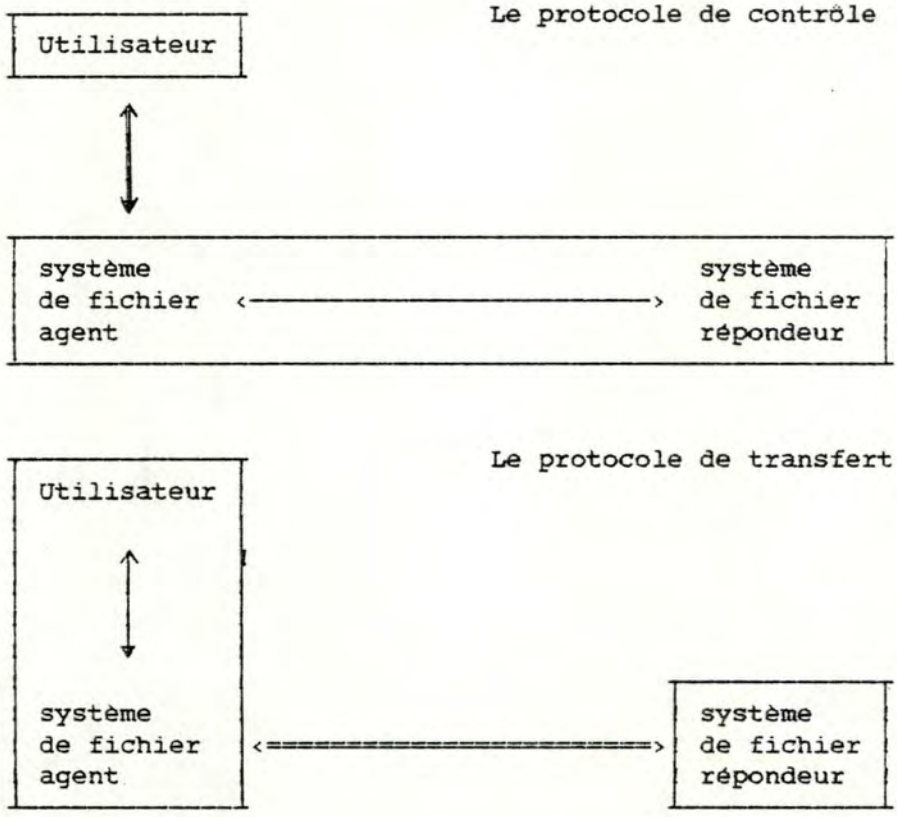


Figure 2.4 Protocole de contrôle et de transfert

2.5.3. Asymétrie du dialogue

Les actions à effectuer par les protocoles de fichier montrent quelques asymétries importantes qui sont reflétées dans la structure du service et du protocole. Les asymétries prennent la forme de relations maître-esclave.

Premièrement, suite à la canalisation des informations décrites ci-dessus, chaque activité est démarrée par un utilisateur du service de transfert de fichier (l'initiateur, A dans la figure 2.3), qui a certains buts à atteindre. La partie associée au système de fichiers (le répondeur, B dans la figure 2.3) réagit simplement à cette initiative dans un rôle passif. En effet, les protocoles ne doivent pas transmettre l'information à propos du système de fichiers de l'initiateur, mais seulement sur celui du répondeur. L'acte de transfert peut être considéré comme effectué par une application de copie qui a un accès local à un fichier et un accès à distance à l'autre.

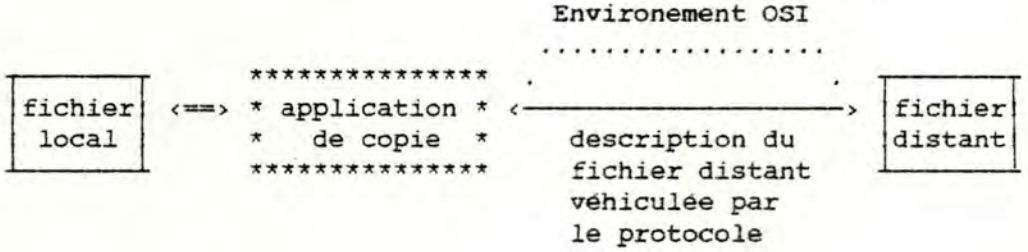


Figure 2.5. Asymétrie du dialogue

La seconde asymétrie est le fait plus fondamental que, lors d'un transfert de données, un côté particulier est l'expéditeur du fichier de données, et l'autre est le destinataire; c'est à dire qu'à tout instant il y a une direction privilégiée du flux de données.

2.5.4. La localisation des fonctions de contrôle d'erreur

La communication identifiée entre l'initiateur et le répondeur dans la clause précédente, peut elle-même être divisée d'une façon modulaire, conduisant à deux services distincts. Ceux-ci sont:

1. un service de transfert de fichier sûr, dans lequel l'utilisateur n'a pas à s'inquiéter de la détection et du recouvrement d'erreur, mais laisse de telles considérations au fournisseur de service une fois que sont définies les exigences de qualité de service.
2. un service de transfert de fichier à correction par

l'utilisateur, qui inclut des primitives donnant à l'utilisateur de service des facilités pour le recouvrement des erreurs et la gestion du transfert.

La spécification du protocole qui relie le service de transfert de fichier sûr au service à correction consiste en un ensemble standard de procédures pour la détection et le recouvrement d'erreurs.

2.6. Fonctions additionnelles

Le Protocole de Transfert de Fichiers peut fournir différents services optionnels qui peuvent être sélectionnés durant la phase de sélection en utilisant un attribut, appelé 'Facilités', permettant à des implémentations ayant différents degrés de sophistication de travailler ensemble. La sophistication d'une implémentation peut alors être facilement exprimée en positionnant les bits appropriés dans cet attribut Facilités. Cette section décrit les différents services possibles.

Les implémentations permettant l'emploi de services optionnels devront être préparées à un refus de ces services, et être capable de travailler à un niveau plus bas. En particulier, le protocole doit décrire un niveau minimum auquel toute implémentation destinée à l'utilisation en environnement de système ouvert doit être prête à travailler.

2.6.1. Fonctions de la couche application

a. types_de_protocoles

Différents types de protocoles ont été identifiés. Les protocoles_de_transfert_de_fichiers s'occupent uniquement de déplacer un fichier d'un système de fichier vers un autre, tandis que les protocoles_d'accès_au_fichier permettent à un processus sur une machine d'accéder et de modifier des morceaux de données sur une autre. Finalement, il y a des protocoles_de_gestion_de_fichier qui permettent des opérations plus étendues, telles que lister un directory et renommer ou effacer un fichier à distance.

b. types_de_fichiers

Le FTP assure toujours le transfert de caractères ou d'octets; il permet parfois en outre d'envoyer le fichier par_enregistrement. Les fichiers à accès_par_clé et les fichiers relatifs sont parfois aussi pris en compte, surtout pour les protocoles d'accès au fichier.

c. wilcards

Certains protocoles permettent le transfert de groupes de fichiers par l'emploi de wilcards, c'est-à-dire des noms de fichier contenant des caractères de remplacement.

2.6.2. Fonctions du service présentation

a. choix_du_code

Les FTP permettent en général de transférer les caractères sous le code ASCII et également de choisir un autre code standard tel que l'EBCDIC ou le sixbit. En outre certains protocoles permettent la définition de codes_privés appelés par un nom convenu entre les deux entités correspondantes.

b. transfert_binaire

La plupart des FTP permettent outre le transfert de fichiers de textes, de transférer des données binaires en représentant les mots machines sur un ou plusieurs octets. L'option mixage permet de mélanger les données binaires et les données textes dans le même transfert. Lorsque les délimiteurs des mots machines ne correspondent pas avec ceux des octets, l'option compactage permet de faire se suivre les mots à l'intérieur des octets d'où un gain de place. Finalement, l'option ordre_modifié permet d'envoyer le bit de poids le plus faible en tête plutôt que le bit de poids le plus fort.

c. compression

L'option compression, appelée aussi 'repeat count', indique si des données textes ou binaires peuvent être transférées en format compressé c'est-à-dire en remplaçant les suites de données identiques par un nombre et le code de la donnée.

d. groupage

L'option groupage permet la non-préservation des délimiteurs c'est-à-dire permet de regrouper plusieurs enregistrements ou de scinder un enregistrement en plusieurs parties.

e. formatage_de_texte

L'option permet de préserver autant que possible la disposition et le formatage des fichiers de texte.

f. encriptage

Une autre option possible est l'encriptage pour assurer le secret des données mais elle n'est pas inclue dans les protocoles car elle peut se faire en dehors du protocole.

g. conversions

Certains protocoles utilisés avec un service transport moins puissant permettent la transformation des octets en code_sur_7_bits par une méthode de préfixage ou encore aux seuls_caractères_écrivables, par exemple lorsque certains caractères de contrôle ont une signification pour les couches inférieures et ne peuvent donc pas être employés.

2.6.3. Fonctions du service session

a. contrôle_de_flux

L'expéditeur spécifie des points de synchronisation en insérant des marques de resynchronisation numérotées dans les données transmises. Le destinataire renvoie un signal d'acceptation (acknowledge) contenant le numéro de la marque pour indiquer qu'il a bien reçu les données jusqu'au point de synchronisation.

b. réinitialisation

Le protocole assure le recouvrement des données perdues ou mal transmises, en retransmettant des parties du fichier pendant la phase de transfert de données. Lorsque le destinataire constate que des données ont été perdues ou sont erronées, il envoie une requête de réinitialisation pour demander la retransmission depuis une marque particulière.

c. fenêtre

La technique de la fenêtre ou window permet d'envoyer plusieurs points de synchronisation successifs avant d'attendre le signal d'acceptation, ce qui rend possible un envoi continu des données.

d. block_check

Le protocole peut assurer le contrôle d'erreur en insérant après chaque enregistrement un code de contrôle. Il se sert alors de la réinitialisation pour assurer la correction des erreurs détectées.

e. time-out

Une perte de communication ou l'échec d'un processus peut se produire à tout moment, et un recouvrement automatique peut ne pas être faisable ou réussi. Pour aider dans de telles situations le protocole peut inclure la notion de time-outs, c'est-à-dire une interruption après un temps fixé. Si un time-out a lieu, le processus qui détecte le time-out termine unilatéralement le transfert. Le service de transport peut également fournir des time-outs pour des recouvrement d'erreurs non destructives. Si un time-out a lieu dans la phase de sélection ou de désélection, la connexion du service de transport peut être fermée, en fournissant des informations de diagnostic. Si un time-out a lieu durant la phase de transfert de données, une forme de requête d'abandon devrait être envoyée comme convenue. Les processus entrent alors en phase de désélection.

Un attribut Minimum Time-out spécifie l'intervalle utilisé. Cette valeur peut être guidée ou ajustée par le processus initiateur, et est utilisée pour limiter en temps toute situation où une réponse, une interaction ou une continuation est demandée. Dans tous les cas, la valeur par défaut de l'attribut Minimum Time-out devrait être choisie

pour prendre en compte tous les délais acceptables du réseau. Il est parfois possible de choisir une valeur différente pour chaque direction de la communication.

f. reprise

Le protocole permet aussi à un fichier d'être envoyé en plusieurs parties, en utilisant la reprise dans le nouveau service de transfert. Cela peut être utilisé soit parce que le premier transfert s'est terminé de façon catastrophique (une reprise non prévue), soit que le transfert initial a été interrompu délibérément pour des raisons opérationnelles ou de gestion. Un même fichier peut être envoyé en autant de parties que désiré. Le service doit être spécifié dans le transfert original, de façon à ce que les processus puissent maintenir les enregistrements convenables. Le mécanisme permet en particulier de s'assurer que le transfert n'ait lieu qu'une et une seule fois, comme par exemple, lorsqu'on transmet une queue de travaux à exécuter.

g. interruption

Une dernière option est la possibilité de suspendre temporairement un transfert, quand le destinataire trouve qu'il y a des conflits de demandes de ressources.

Chapitre 3 : EXEMPLES DE PROTOCOLES

Ce chapitre décrit des exemples de protocoles de transfert de fichiers existants, en les comparant au modèle général donné au chapitre précédent. Je me suis principalement intéressé aux protocoles qui ne sont pas liés à un réseau d'un constructeur particulier et je décris ici trois protocoles qui peuvent être considérés comme des standards dans leur domaine.

3.1. NIFTP ou Network Independent File Transfer Protocol

Ce protocole est aussi appelé Blue Book. On en trouvera la description dans le document de référence [NIFTP].

3.1.1. Terminologie

processus_P (P process) : Il s'agit du processus initiateur du transfert.

processus_Q (Q process) : Il s'agit du processus répondeur.

processus_S (S process) : Il s'agit du processus envoyeur (sender en anglais).

processus_R (R process) : Il s'agit du processus receveur (receiver en anglais).

Virtual_Filestore (Système virtuel de fichier) : La représentation standard d'ensembles organisés de fichiers.

Record (Enregistrement) : Une unité utilisée pour structurer les données. Cellle-ci est découpée en subrecords pour le transfert des données.

Subrecord : L'unité en laquelle les données et les commandes sont divisées pour la transmission.

T-attributs (T attribute) : Les attributs de transfert. Ce sont les informations propres au transfert du fichier.

Q-attributs (Q attribute) : Les attributs de stockage. Ce sont les informations concernant la façon dont le fichier est ou sera stocké au site répondeur (Q).

I-attributs (I attribute) : Les attributs d'information. Ils fournissent des informations complémentaires pour le terminal utilisateur ou pour un opérateur.

Tansfer_docket (Docket de transfert) : Un ensemble d'informations que l'envoyeur et le receveur doivent maintenir pour permettre la reprise correcte d'un transfert arrêté. Il contient les contraintes courantes sur les valeurs des attributs.

Paramètre (Parameter) : Donnée accompagnant les commandes d'initialisation permettant de spécifer des contraintes sur les valeurs des attributs lors de la négociation.

Monitor_Flag : Drapeau associé à un paramètre des commandes d'initialisation pour indiquer que l'initiateur désire connaître la valeur attribuées par le répondeur à l'attribut spécifié.

IA5 : Autre nom pour le code standard international ASCII. Une table est donnée à la page 155 du texte de la norme NIFTP.

Code_Privé (Private Code) : Code non standard convenu entre deux sites pour la représentation des caractères.

Mot_Binaire (Binary Word) : Ensemble de bits servant d'unité pour les transferts de fichiers contenant autre chose que du texte. Il correspond généralement à l'unité de mémorisation des ordinateurs en communication.

Padding : Remplissage avec des bits à zéro lorsque les mots binaires ne sont pas un multiple de l'octet.

Tab_stops (Tabulations) : Positions dans une ligne donnant le déplacement à effectuer lors de la rencontre du caractère de tabulation (code 09 en ASCII).

Delimiter_Preservation (Préservation des délimiteurs) : Un attribut de transfert indiquant si les positions des limites des enregistrements transmis ainsi que des tabulations doivent être préservées dans le fichier du receveur et donc dans le transfert.

Capability (Capacité) : La capacité d'une implémentation de transfert de fichier d'exécuter une certaine fonction.

Facilities (Facilités ou Options) : Un attribut de transfert dont la valeur indique quelles options sont utilisées pendant le transfert des données.

Hold_facility (Capacité d'interruption) : La capacité du receveur de demander à l'envoyeur de suspendre le transfert de fichier.

Hold_state (Etat interrompu) : L'envoyeur est à l'état interrompu après avoir accepté une demande d'interruption. Dans cet état aucune donnée ne peut être transmise.

Restart (Resynchronisation ou Réinitialisation) : Une méthode utilisée lors de la perte de données ou d'autres conditions d'erreur, en retransmettant une partie du fichier durant la même opération de transfert de fichier.

Restart_mark (Marque de restart ou Point de synchronisation) : Une marque placée dans le flot de données pour fournir des positions connues à partir desquelles une resynchronisation ou une reprise peut avoir lieu.

Initial_restart_mark (Marque initiale) : Un attribut de transfert indiquant à quel point de synchronisation le transfert doit démarrer.

Mark_Acknowledgement (Acceptation d'une marque) : La confirmation de la réception et du sauvetage des données jusqu'à une marque de restart spécifiée.

Acknowledgement_Window (Fenêtre) : Un attribut de transfert qui définit le nombre maximum de marques de restart que l'envoyeur peut transmettre avant de recevoir une acceptation du receveur.

Resumption (Reprise) : La méthode où plusieurs opérations de transfert sont combinées pour transmettre un fichier complet.

Push : Une requête au service de transport pour assurer la délivraison des messages sans en attendre d'autres, c'est-à-dire une demande de vider les buffers intermédiaires.

Transfert_control_command (Commandes de contrôle de transfert) : Ces commandes sont utilisées pendant la phase de transfert de données pour indiquer les erreurs, les marques de synchronisation, etc.

3.1.2. Objectifs par rapport au modèle général

Le protocole NIFTP a déjà quelques années derrière lui. Il a été défini dès 1978 et a été revu depuis ce qui fait que la plupart des problèmes et des ambiguïtés ont été éclaircis. Il couvre les trois couches supérieures du Modèle de Référence ISO, mais comme sa définition a précédé la définition du Modèle, les différentes fonctions ne sont pas classifiées dans les trois niveaux. Il spécifie un protocole qui permet le transfert de fichier sur n'importe quel type de réseau de communication, public ou privé, indépendamment des caractéristiques hardware donc indépendamment de tout constructeur. En l'absence de standard international, il définit un protocole qui peut être utilisé pendant l'intérim. La CEE a d'ailleurs reconnu ce protocole pour ses échanges en attendant la définition de la norme ISO.

Ce protocole ne définit que le transfert de fichier et non pas la gestion ni l'accès aux éléments du fichier. Cependant, il inclut certaines options dans le mode d'accès permettant par exemple de détruire le fichier après copie, ou de l'envoyer vers une file pour exécution. D'autres protocoles sont définis par la même source pour assurer le transfert de travaux ou les applications de courrier électronique.

Son exigence de base vis-à-vis du niveau transport est la fourniture de deux flux d'octets (un dans chaque sens) entre les deux sites qui implémentent le service de transfert de fichier. Outres les exigences décrites dans le modèle général, le service transport doit fournir une fonction, appelée push, qui a pour but de forcer la livraison de toute donnée bufferisée. Bien qu'en principe le niveau transport devrait être 'error free', NIFTP prévoit un mécanisme de restart en cas d'erreur ou de perte de données.

3.1.3. Procédures de transfert

Dans NIFTP, le transfert se fait en trois phases: la phase d'initialisation, la phase de transfert des données et la phase de terminaison. La phase d'initialisation correspond à la phase de sélection du modèle général et la phase de terminaison correspond à la phase de désélection. Le protocole NIFTP ne parle quasiment pas de l'établissement et de la terminaison de la connexion. Il démarre par la réception du signal d'ouverture de connexion; il suppose donc que la demande de connexion se fait en dehors de l'activité de transfert de fichier.

Les phases d'intialisation et de terminaison ont lieu entre un initiateur P et un répondeur Q qui se transforment en envoyeur S et en receveur R pendant la phase de transfert des données. Cela fait que pendant cette phase de transfert, il n'y a aucune

différence entre un envoi ou une réception de fichier, seuls les rôles des deux entités changent.

3.1.4. Attributs et fichier virtuel

Le protocole NIFTP se base sur une définition de fichier virtuel appelé Virtual Filestore. Il est vu comme une suite de records qui sont eux-même des suites de caractères ou de mots binaires. Ses attributs possibles sont: le nom du fichier considéré, son mode d'accès, sa taille et la grandeur maximum de ses records, la taille du mot binaire ou son code et éventuellement le nom du code privé utilisé, un nom d'utilisitateur, un nom de compte (account), un mot de passe du fichier, un mot de passe de l'utilisateur et un mot de passe du compte. Les noms du fichier, de l'utilisateur et du compte ainsi que leur mot de passe sont considérés comme des suites de caractères. En outre les attributs 'Output Device Type' et 'Device Type Qualifier' permettent de caractériser le système de destination quand il s'agit, par exemple, d'envoyer le fichier vers une imprimante.

Au niveau des attributs de transfert, il est possible de négocier les codes de transfert ou les formats binaires, différents modes de formatages ainsi que la préservation des délimiteurs et des tabulations.

NIFTP se distingue par un mécanisme de négociation fort complet permettant une grande variété de transferts. Le lecteur trouvera une description de ces différents attributs à la section 2.7.3.2. de l'annexe au mémoire.

3.1.5. Le contrôle du transfert

Comme il veut rester très général et pouvoir ainsi être utilisé d'une grande variété de façons, le protocole NIFTP ne définit pas les relations avec l'utilisateur. Il s'occupe uniquement des informations qui permettent d'envoyer ou de recevoir un fichier, pas des raisons de l'initiateur de demander le transfert. L'implémentation déplacera des fichiers à la demande de l'utilisateur du service de transfert de fichier. La façon dont un tel utilisateur spécifie ses besoins dépend en beaucoup de points de l'implémentation locale.

Cependant, un certain nombre de facilités sont prévues telles que la possibilité d'interrompre momentanément le transfert, la possibilité d'arrêter un transfert et de le reprendre plus tard, la possibilité de transmettre des messages d'information à un utilisateur ou des messages pour demander certaines actions à un opérateur telles que monter une bande.

Pendant le transfert, le contrôle du transfert se fait via des commandes de contrôle de transfert qui s'insèrent entre les subrecords de données. Ces commandes permettent de signaler les erreurs, de demander l'arrêt ou l'interruption du transfert, de changer de code en cours de transfert et d'assurer la synchronisation. Ces contrôles sont gérés comme si l'utilisateur se trouvait au site P, il n'est pas explicitement prévu que le contrôle puisse se faire à partir d'un troisième site. Mais comme il est expliqué dans le modèle général, cela ne constitue pas une réelle limitation.

3.1.6. Fonctions additionnelles

Au niveau des options possibles, ce protocole est très riche. Pour le service présentation, il permet le choix du code avec code privé, le transfert binaire mélangé avec le texte, le compactage et l'ordre modifié en mode binaire, la compression en modes binaire et texte, la préservation ou non des délimiteurs et un formatage de texte très détaillé. Seules les options f. et g. du modèle général ne sont pas prévues. Au niveau de la couche session, toutes les options indiquées dans le deuxième chapitre sont comprises.

Comme on le voit, NIFTP est sophistiqué du côté des options possibles. La volonté de généralité y est pour beaucoup. Le revers de cette sophistication est que cela le rend complexe à comprendre et implémenter.

3.2. FTAM ou File Transfer, Access and Management

Le protocole FTAM est appelé à devenir une norme internationnale ISO. Il est actuellement développé par le Working Group 95 pour le Subcommittee 16 du Task Committee 97 portant sur l'interconnexion des systèmes ouverts. Je me suis basé pour cette étude sur l'avant-projet ISO/DP 8571 [FTAM].

3.2.1. Terminologie

Filestore (Système de fichier) : Un ensemble organisé de fichiers, incluant leurs attributs et leur nom, résidant dans un système ouvert particulier.

Filestore_action (Action sur un système de fichier) : La définition d'un type d'action qui a une signification en termes de système de fichier.

Commitment_unit (Transaction) : Une unité de traitement qui doit réussir ou échouer complètement. Si l'unité réussit, ses effets sont visibles pour les autres processus, tandis que si elle échoue, les effets du traitement sont annulés.

FADU_ou_File_access_data_unit : Unité de donnée d'accès au fichier.

PDU_ou_Protocol_data_unit : Unité de donnée du protocole.

3.2.2. Objectifs par rapport au modèle général

Le but de cette norme est la manipulation d'ensembles identifiables d'informations, telles que celles mémorisées dans les systèmes de fichiers, ou passées comme un tout entre des processus d'application en communication. Outre le transfert de fichier, elle permet l'accès aux possibilités de stockage d'informations de telle façon que les données puissent être ajoutées, effacées ou modifiées, sans qu'il y ait besoin de connaître la façon dont le système de fichier est fourni. Elle permet également la manipulation des fichiers comme un tout c'est-à-dire de le créer, de l'effacer, de changer son nom et d'autres attributs.

3.2.3. Procédures de transfert

Par rapport aux phases du modèle général, FTAM décompose la phase de sélection en trois phases : La sélection du fichier, la manipulation du fichier et son ouverture. De même, la désélection est découpée en une phase de fermeture et une phase de désélection. De plus, une phase de gestion du système de fichier est insérée entre la phase d'établissement de connexion et la phase de sélection.

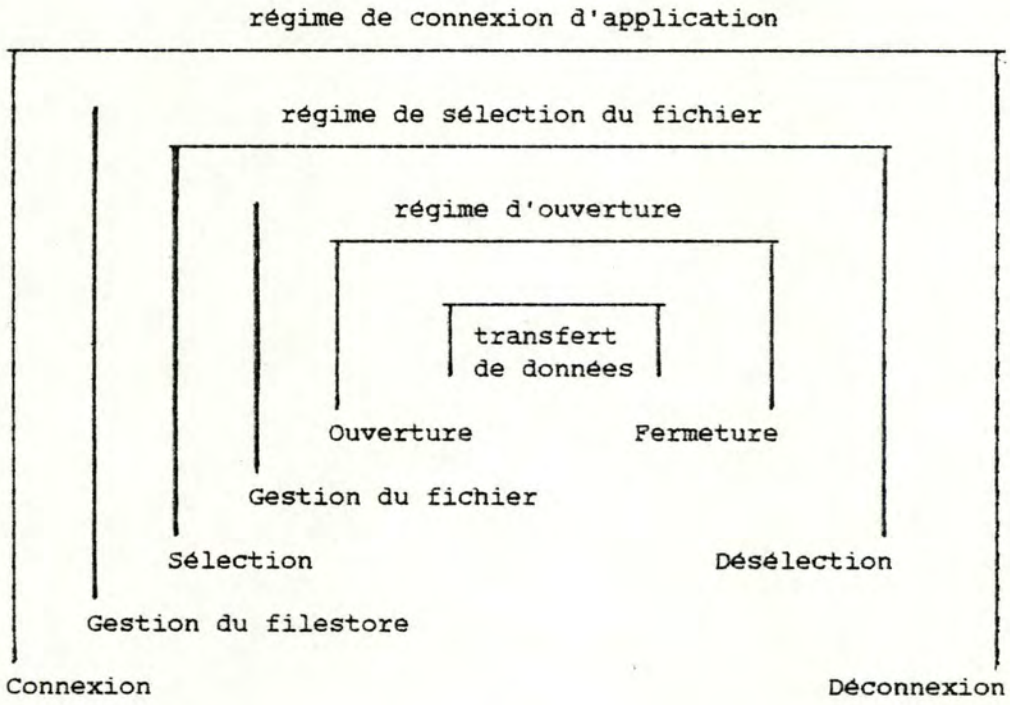


Figure 3.1 Imbrication des régimes dans FTAM

3.2.4. Attributs et fichier virtuel

Le fichier virtuel est défini comme une entité ayant: 1) un nom, qui permet de la référencer sans ambiguïté, 2) d'autres attributs qui expriment des propriétés communes du fichier telles que la taille, l'historique,etc. , 3) des attributs décrivant la structure logique et la dimension des données stockées dans le fichier, 4) toute donnée formant le contenu du fichier. Outre ces attributs utilisés pour décrire la structure permanente du fichier, il y a, pour chaque activité, un attribut du contexte d'accès indiquant quel sous-ensemble du fichier est actuellement utilisé.

Un fichier contient une ou plusieurs unités de données appelées DU ou Data Units qui peuvent être logiquement reliées. En général ces relations seront séquentielles, hiérarchiques, en réseaux ou relationelles. Ce modèle de fichier virtuel est basé sur la structure d'arbre pour représenter les relations entre les Data Units afin de les accéder et de les identifier. Cette structure est appelée la structure d'accès. Chaque noeud qui n'est pas une feuille contient une ou zéro Data Unit, et chaque feuille en contient exactement une. Un sous-arbre de la structure d'accès est appelé une FADU ou File Access Data Unit qui est un objet typé. C'est l'unité sur laquelle les opérations peuvent être exécutées. Deux cas particuliers de cette structure d'accès sont 'Unstructured', c'est-à-dire une racine et une Data Unit, ce qui correspond à un fichier séquentiel, et 'Flat', qui consiste en deux niveaux, une racine sans Data Unit et des Data Units uniquement aux feuilles, ce qui correspond à un fichier d'enregistrements.

Outre le nom du fichier, il existe des attributs pour le mode d'accès et les protections correspondantes, pour un compte, pour les heures et dates de création, de modification et d'accès, pour l'identité du créateur, du dernier modifieur et du dernier lecteur, pour le contexte de présentation, pour une clé d'encriptage, pour la structure d'accès, pour la taille présente et future, etc. Il est même prévu un attribut pour une qualification légale, lorsque les lois seront sorties. A cela il faut encore ajouter une série d'attributs d'activité. On peut difficilement imaginer plus !

Tous ces attributs peuvent être de trois types: les attributs scalaires, qui contiennent une seule valeur, les attributs vecteurs, qui contiennent une liste de zéro ou plusieurs élements qui ont chacun une valeur, les attributs ensembles, qui sont des ensembles non ordonnés d'éléments. Les valeurs d'attributs peuvent être des séquences de caractères, des séquences d'octets, des entiers, l'adresse au point d'accès au service de fichier, une date et une heure ou un élément d'un ensemble défini de valeurs. Comme on le voit, FTAM a une syntaxe très riche pour les attributs.

3.2.5. Le contrôle du transfert

Les spécifications du service de fichier n'indiquent pas quand et comment une connexion du service session est établie, car ces détails dépendent des aspects de la gestion des systèmes en communication. La disponibilité d'une connexion est considérée comme un aspect de l'environnement dans lequel le service de fichier est défini.

Le contrôle du transfert et de l'accès se fait par des primitives qui sont au même niveau que les demandes de données.

3.2.6. Fonctions additionnelles

Au niveau des applications, outre le transfert de fichier, FTAM assure la création et l'effaçage des fichiers, la lecture et la modification des attributs du fichier, la localisation, la lecture, l'insertion, le remplacement, l'extension et l'effacement des unités de données. L'accès peut être direct ou séquentiel et en lecture, en écriture ou les deux à la fois. Tout ceci est contrôlé par des permissions d'accès qui peuvent être fonction de l'identité de l'utilisateur, d'un mot de passe et/ou de la localisation de l'entité initiatrice.

Les aspects du transfert qui ont une signification pour les applications sont exprimés en termes d'une syntaxe abstraite qui comprend la déclaration des types de données à communiquer. Les transformations effectuées par la couche présentation doivent être restreintes de façon à préserver tous les aspects des types de données définis. La couche présentation s'occupe également de la transformation du nom du fichier qui est défini comme un vecteur de séquences de caractères, les délimiteurs étant définis localement.

Pour ce qui est des options, toutes celles du modèle général sont prévues et encore bien d'autres puisque le protocole définit également l'accès au fichier; l'esprit de FTAM étant de permettre de faire un maximum de choses.

3.3. KERMIT

Le protocole KERMIT a été développé au départ à l'université de Columbia à New-York. Il est distribué gratuitement et demande très peu de moyens, ce qui fait qu'il est très répandu surtout pour la connexion entre un ordinateur hôte et un micro-ordinateur travaillant en tant que terminal.

3.3.1. Terminologie

Local : Lorsque deux machines sont connectées, la machine "local" est celle avec laquelle vous interagissez directement et qui a le contrôle du terminal. Le "KERMIT local" est celui qui tourne sur la machine "local".

Remote : La machine "remote" est celle qui est du côté éloigné de la connexion et avec laquelle vous devez parler via la machine "local". Le "KERMIT remote" tourne sur la machine "remote".

Server (Serveur) : Une implémentation du KERMIT remote qui peut acccepter des commandes d'un programme KERMIT local, plutôt que directement de l'utilisateur.

User (Utilisateur) : Outre son sens habituel qui dénote la personne qui utilise un système ou un programme, "utilisateur" est également utilisé en se référant au programme KERMIT local, lorsque le KERMIT remote est un serveur.

Printable_ASCII_character (Caractère ASCII écrivable) : Un caractère dont le code ASCII est compris entre 32 (espace) et 126 (tilde).

Packet (Paquet) : Un string de caractères clairement délimité, y compris des champs de contrôle mis autour des données. Les champs de contrôle permettent au programme KERMIT de déterminer si les données ont été transmises correctement et complètement. Le paquet est la plus petite unité de transmission dans le protocole KERMIT.

ACK : Abbréviation de "Acknowledge". Un ACK est un paquet qui est envoyé pour accepter la réception d'un autre paquet.

NAK : Abbréviation pour "Negative Acknowledege". Un NAK est un paquet pour dire qu'un paquet corrompu ou incomplet est arrivé, qu'un mauvais paquet est arrivé ou que le paquet attendu n'est pas arrivé.

3.3.2. Objectifs par rapport au modèle général

Le protocole de transfert de fichier KERMIT est destiné à des transmissions orientées caractère, et cela sur des lignes de communication sérielles. Il est relativement simple et emploie ce qui est disponible c'est-à-dire les lignes vers les terminaux. Son utilisation typique est le transfert entre un ordinateur hôte et un micro-ordinateur qui émule un terminal.

3.3.3. Procédures de transfert

Voici la séquence normale des paquets à envoyer. On y reconnaîtra les phases de sélection, de transfert, de désélection et de terminaison de la connexion. Par rapport au modèle général, il y a une adaptation pour prendre en compte le transfert de fichiers groupés.

1. L'envoyeur envoie le paquet Send-Initiate (S) pour spécifier ses paramètres.
2. Le receveur renvoie un ACK (Y) avec ses propres paramètres.
3. L'envoyeur transmet un paquet File-Header (F) qui contient le nom du fichier et le receveur renvoie un ACK.
4. L'envoyeur envoie le contenu du fichier dans des paquets de données (D). Chaque paquet de données doit être acccepté (ACK) avant de pouvoir transmettre le suivant.
5. Lorsque tout le fichier est transmis, l'envoyeur envoie un paquet End-Of-File (Z) et le receveur l'accepte (ACK).
6. S'il y a un autre fichier à transmettre, le processus est répété à partir de la troisième étape.
7. Lorsqu'il ne reste plus de fichier à transférer, l'envoyeur transmet un paquet End-Of-Transmission (B) et le receveur l'accepte. Cela termine la transaction et ferme la connexion logique (la connexion physique reste ouverte).

Lorsqu'un paquet est mal reçu, le receveur renvoie un NAK et le paquet est retranmis. De plus, toute condition qui empêchera la transmission réussie du fichier est appelée une erreur fatale. Les paquets Error (E) fournissent le moyen, lorsqu'une telle erreur est détectée, d'arrêter le transfert gracieusement, en fournissant les informations pertinentes à l'utilisateur.

Le protocole KERMIT couvre toutes les couches ISO depuis la couche de liaison de données jusqu'à la couche application. Il ne demande que de pouvoir transmettre des caractères ASCII encodés sur 7 ou 8 bits sur une connexion physique RS-232. Il peut même se restreindre à l'ensemble des caractères écrivables plus un caractère de contrôle.

3.3.4. Attributs et fichier virtuel

L'objectif de KERMIT est le transfert de fichiers séquentiels uniquement. La version standard ne prévoit aucun attribut autre que le nom du fichier et les particularités physiques de la ligne. Mais optionnellement, l'échange d'attributs du fichier est prévu. Le nom du fichier est composé de deux strings séparés par un point permettant de spécifier un nom et un type; tout le reste de la structure d'accès (directory, volume, etc.) est oubliée.

3.3.5. Le contrôle du transfert

Les communications entre systèmes sont établies par le simple utilisateur. Typiquement, l'utilisateur tourne KERMIT sur un micro-ordinateur, entre en émulation de terminal et se connecte à un ordinateur remote, fait le login, exécute KERMIT sur l'ordinateur remote, et lance les commandes pour lancer le transfert, puis "s'échappe" de retour vers le micro et lance les commandes de telle façon que le KERMIT local démarre son côté du transfert de fichier, les fichiers pouvant être transférés seuls ou groupés.

Les versions plus avancées simplifient l'interface utilisateur en permettant au KERMIT sur l'ordinateur remote de tourner comme un serveur, qui peut transférer des fichiers dans les deux sens sur commande du KERMIT de l'utilisateur. Le serveur peut fournir également des facilités pour la gestion des fichiers.

3.3.6. Fonctions additionnelles

Outre le transfert de fichier, KERMIT prevoit l'émulation de terminal et également en option la gestion du fichier y compris le changement de direcory et le listing de celui-ci.

KERMIT a pour but principal le transfert de fichiers textes mais il transmet aussi des fichiers binaires par mots de 8 bits comme s'ils étaient des suites de caractères.

Ce protocole prévoit le renvoi des paquets erronés mais pas la gestion de la fenêtre. Tout le transfert se fait donc à l'alternat.

Finalement, afin de demander très peu de ressources, KERMIT permet la transformation de 8 bits en 7 bits par préfixage et également la restriction à l'ensemble des caractères écrivables. Il permet en outre la compression par repeat count.

3.4. Comparaison

Nous avons donc vu trois protocoles qui peuvent être considérés comme des standards dans leur domaine. Je ne pense pas qu'on puisse dire que l'un est meilleur que l'autre. Chacun d'eux a ses objectifs propres et dans ce domaine, il est le mieux conçu. Il me reste donc à indiquer le pour et le contre de ces protocoles et rappeler les domaines auxquels ils se destinent.

NIFTP se caractérise par la négociation des attributs ainsi que par un assez grand nombre d'options de présentation, ce qui le rend assez complet. Cependant, il ne cherche pas masquer l'ordinateur distant. Il suppose la connaissance de la structure des fichiers au site distant ainsi que la façon de les nommer. Il a aussi l'avantage d'exister et d'être déjà implémenté sur un certain nombre de machines. Cependant, Paul Van Binst, professeur à la VUB et utilisateur de NIFTP, m'indiquait lors d'une rencontre que toutes les implémentations qu'il avait vues étaient des versions minimum de NIFTP.

FTAM est caractéristique des normes ISO, c'est-à-dire qu'il est le fruit de longues négociations entre un grand nombre de parties. Il est donc extrêmement complet mais aussi très complexe et difficile à comprendre. Outre le transfert de fichier, il a pour but la gestion des fichiers et l'accès aux enregistrements. Tout cela sans devoir connaître la structure des fichiers distants ni la façon de les nommer et quelque soit la forme possible du fichier. Il a par conséquent un niveau présentation très poussé avec de nombreux attributs. Pas un seul ne manque. Cependant lorsque je vois déjà la difficulté d'implémenter NIFTP (cfr deuxième partie du mémoire et l'annexe), je me demande s'il sera possible d'implémenter un tel protocole.

Comparé aux deux autres, KERMIT est très simple dans sa version de base, ce qui permet de l'implémenter sur micro-ordinateur; mais les implémentations avec serveur peuvent grâce aux options devenir assez complètes. Son principal avantage est le peu de moyens de communications nécessités; ils sont d'ailleurs souvent déjà disponibles. Mais le revers de la médaille est qu'il est peu efficace pour des communications entre gros ordinateurs puisqu'il ne prévoit pas de mécanisme de fenêtre, qu'il travaille avec de petits paquets et que le système de préfixage est assez lourd.

Pour conclure, voici les domaines propres de chaque protocole. KERMIT me semble principalement destiné aux communications entre nicro-ordinateurs ou entre un ordinateur et un micro-ordinateur émulant un terminal. Je propose NITFP pour les transfert de fichiers entre gros ordinateurs considérés comme des entités séparées. Et finalement FTAM pour un réseau d'ordinateurs vu comme un système unique, tel qu'il en existe dans les banques ou les administrations, c'est-à-dire comme point de départ pour une base de donnée répartie.

PARTIE 2

MISE EN OEUVRE

Chapitre 4 : LA MAQUETTE

La deuxième partie porte sur la réalisation d'une maquette. Ce chapitre 4 expose les objectifs et l'environement de cette maquette, tandis que le chapitre 5 portera plus sur l'aspect implémentation.

4.1. Objectifs

Le but de la deuxième partie du mémoire était initialement l'écriture d'un programme assurant le transfert de fichier entre machines hétérogènes, respectant un protocole choisi et se basant sur une couche transport fournie.

Suite à l'analyse des trois protocoles, mon choix s'est porté sur le protocole NIFTP. Il y a trois motifs à ce choix. D'abord du point de vue de l'intérêt, NIFTP est un protocole qui est de plus en plus reconnu comme pouvant servir de standard; son choix par la CEE en témoigne. De plus, il est relativement complet et permet de saisir les mécanismes de la négociation des attributs. Le deuxième motif est le fait qu'il soit clairement expliqué et que, vu son expérience, il risque de donner peu de surprise par des problèmes inattendus. Le dernier motif est la procédure par élimination. Le protocole FTAM est très complexe et obscur, et par surcroît non encore définitivement spécifié. KERMIT, pour sa part, englobe les niveaux inférieurs ce qui oblige à écrire des routines de bas niveau et offre peu d'intérêt pour la compréhension des mécanismes propres aux couches supérieures. Il ne restait donc plus que NIFTP.

Ayant choisi le protocole, il me fallait choisir la machine et le langage d'implémentation. J'ai choisi UNIX et le langage C pour plusieurs raisons. D'abord, ils sont tous deux très 'up to date' et de plus en plus employés, huit constructeurs européens ne viennent-ils pas en début d'année de choisir UNIX comme leur standard commun ? De plus UNIX offre un environnement de programmation apportant pas mal de facilités pour faire du développement. Un autre argument est la nécessité de pouvoir gérer des interruptions ainsi que plusieurs processus simultanés, ce qui est généralement loin d'être évident sur d'autres systèmes si pas impossible. Finalement, UNIX offrait le système des pipes entre les processus ce qui permettait une bonne simulation de l'échange entre deux ordinateurs.

Au vu des circonstances, il m'a semblé préférable de réaliser une maquette, c'est-à-dire une simulation, plutôt qu'une implémentation réelle entre deux machines. D'abord, du point de vue pratique, je ne disposais pas d'une implémentation des couches inférieures ni d'une connexion physique. Il m'aurait également fallu faire deux implémentations sur des machines différentes; ce qui représente un travail supplémentaire même en veillant à un maximum de portabilité. D'autre part, le programme de transfert de fichier selon la norme NIFTP est déjà disponible pour le PDP11 sous UNIX. Par contre, une maquette montrant le fonctionnement d'un transfert de fichier peut être utile dans le cadre du cours de téléinformatique.

4.2. Architecture

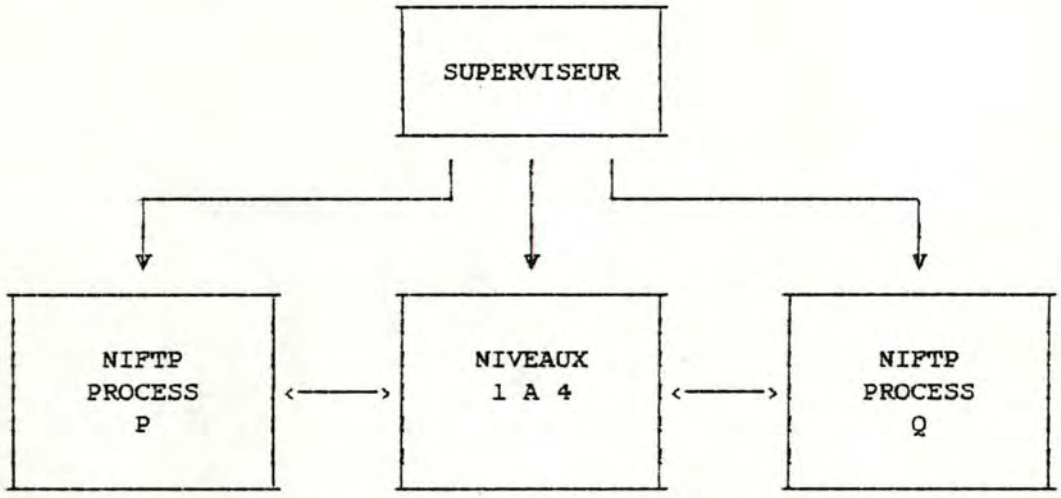


Figure 4.1 - Architecture Générale.

La maquette est construite autour de plusieurs processus. D'abord, il y a deux processus implémentant le protocole NIFTP, l'un jouant le rôle d'initiateur et l'autre le rôle de répondeur. Ils doivent avoir chacun leur écran de communication avec l'utilisateur. Ensuite, pour assurer le transfert, il faut un processus simulant le niveau transport et les niveaux inférieurs. Celui-ci doit afficher sur un écran le contenu des différents paquets qui passent par la connexion transport. Finalement, comme UNIX ne permet des communications par des pipes qu'entre des processus créés par un même père, j'ai rajouté un processus SUPERVISEUR chargé d'initialiser les trois autres processus et les pipes qui les relient. De plus pour éviter la monopolisation

de trois terminaux, le processus SUPERVISEUR s'occupe de rassembler les trois écrans sur un seul écran physique tout en permettant à chaque processus de travailler comme s'il avait son propre écran.

Comme une maquette pour le niveau 3 avait déjà été réalisé dans le cadre d'un laboratoire de deuxième licence [LABO], je me suis arrangé pour permettre une intégration future possible. C'est pourquoi, j'ai pris pour les pipes communiquant avec le niveau transport la même définition que celle prise dans ce laboratoire. Ainsi on pourra facilement écrire un niveau quatre pour faire le lien entre les deux.

Du côté des processus implémentant le protocole NIFTP, je les ait définis de telle manière qu'ils puissent être réutilisables dans d'autres maquettes ou dans un système réel lorsqu'il existera une ligne physique et que les couches inférieures seront implémentées. Ainsi pour les réutiliser, il suffira de redéfinir les primitives de base du module INTERFACE NIVEAU 4 en fonction de l'implémentation de la communication avec le niveau 4 et également la partie gérant d'écran du module UTILITAIRES en fonction de l'interface avec l'écran. La maquette montre une utilisation interactive du protocole de transfert mais on peut facilement imaginer que le transfert soit utilisé par un programme de plus haut niveau. Dans ce cas, il suffit de redéfinir l'interface avec l'écran ou d'envoyer sur le pipe les messages correspondant aux fonctions demandées. Cette modularité se retrouve également à l'intérieur du processus NIFTP qui est décomposé en sept modules, répartis suivant les couches du Modèle de Référence ISO. Les modules TRANSFERT et NEGOCIATEUR se trouvent dans la couche application, le module PRESENTATION se trouve dans la couche 6 comme sont nom l'indique, le module FORMATEUR DE RECORD correspond à la couche session, comme son nom ne l'indique pas, le module INTERFACE NIVEAU 4 est là pour assurer une indépendance par rapport à l'implémentation de la couche 4, les deux autres modules n'implémentent pas des fonctions du protocole mais fournissent les élements de base (événements, mémoire, time-outs, écran, ...) et le séquencement.

Il me reste encore à indiquer les place des différents éléments du système de transfert réel. Les deux couches inférieures, physique et liaison de données, se trouveront dans l'O.S., si pas sous forme de hardware spécialisé. Les couches 3 et 4 seront typiquement dans l'O.S. Les trois couches supérieures, session, présentation et application seront fournis comme des routines d'une librairie, tandis que le programme utilisateur exécutera soit un programme de plus haut niveau, soit jouera le rôle du processus SUPERVISEUR, c'est-à-dire assurera la conversion entre les commandes reçues à l'écran et l'appel des routines de la librairie.

Chapitre 5 : L'IMPLEMENTATION

5.1. Fonctionnalités

Les fonctions offertes par le processus de transfert de fichiers sont celles correspondantes aux messages définis dans le gérant d'écran à la page 17 de l'annexe. Il y a d'abord la demande d'envoi ou de réception d'un fichier en indiquant le nom de départ et le nom d'arrivée. Différentes options peuvent être ajoutées pour indiquer, d'une part, le nom de l'utilisateur et les différents mots de passe nécessaires pour accéder au fichier et, d'autre part, le mode d'accès, le code du texte, le format binaire, le formatage de texte, le type de données ou la taille de la fenêtre. Pour ces dernières options, il serait possible de définir des macros telles que 'COBOL', 'FORTRAN', 'OBJET', 'LPT', etc. qui détermineraient ces options en fonction du type de fichier à envoyer ou recevoir. Il y a aussi la possibilité d'arrêter le transfert ou de l'interrompre momentanément.

Les services attendus de la couche transport sont ceux définis dans le document de Béatrice Scoyer intitulé 'layer4' [LAYER4]. On en trouvera une liste détaillée à partir de la page 33 de l'annexe. Il s'agit essentiellement de pouvoir ouvrir, fermer ou attendre une connexion, de pouvoir envoyer ou recevoir des données, de permettre à la couche 4 de prévenir de la survenance d'un événement et, finalement, des contrôles de flux entre les deux couches.

5.2. Difficultés d'implémentation

Lors de l'implémentation, j'ai rencontré différentes difficultés qui m'ont obligé à revoir plusieurs fois l'architecture du programme avant d'aboutir à l'architecture décrite en annexe. En voici les principales.

Il y a d'abord les caractéristiques propres d'un protocole comme NIFTP. L'emploi du mécanisme de la fenêtre qui permet d'envoyer plusieurs records de données sans en attendre l'acceptation, fait que les messages renvoyés par le receveur arrivent d'une manière asynchrone c'est-à-dire sans être explicitement attendus. Une autre source d'asynchronisme est la possibilité d'interrompre ou d'arrêter le transfert à tout

moment. Un autre événement qui peut arriver sans être attendu est le time-out. J'ai donc décidé de gérer cela par des interruptions qui génèrent des événements; il me fallait donc implémenter un gérant de la file des événements qui sont traités l'un après l'autre par le séquenceur. De plus grâce à ce système le séquenceur ne doit pas attendre les statuts renvoyés par le niveau transport; il peut continuer à traiter les autres événements, s'il y en a, et a l'arrivée du statut celui-ci est mis dans la file des événements. De même lorsqu'il n'y a pas d'événements à traiter, il peut, plutôt que de boucler pour rien, se mettre en pause et il sera alors réveillé à la première interruption. Il libère ainsi le processeur pour les autres processus.

La deuxième difficulté est que NIFTP assigne des priorités aux événements. Par exemple, en cas de fermeture de la connexion (TS_close) on ne peut plus rien envoyer même pas un message d'erreur. Cet événement est donc prioritaire par rapport aux autres. Un autre exemple est la détection de la fin du fichier qui passe avant la demande d'interruption puisque le transfert des données est terminé. Il faut encore ajouter à cela qu'un paquet de données arrivé du niveau transport doit d'abord passer par le module FORMATEUR DE RECORD avant de passer au module PRESENTATION et au module TRANSFERT ou NEGOCIATEUR, ce qui implique aussi des priorités.

La difficulté principale est la suivante. Selon la description du protocole, les commandes de contrôle de transfert son précédées d'une entête à zéro et les subrecords ont toujours une entête différente de zéro. Ce mécanisme doit permettre, en principe, de les reconnaître à tout moment. Cependant comme rien n'empêche d'avoir des octets à zéro dans le corps des subrecords, il ne suffit pas de repérer les octets nuls dans le flot d'octet; il faut donc repérer chaque subrecord. Ici intervient un deuxième problème: dans le cas des fichiers textes en format non compressé, il suffit de regarder le champ 'compte' pour savoir la longueur en octets du subrecord et donc faire la décomposition en subrecords. Mais dans le cas du format compressé et, ce qui est encore plus compliqué, dans le cas de données binaires dont la longueur de mot n'est pas un octet, il faut refaire une partie du travail du module PRESENTATION pour décomposer en subrecords.

La difficulté suivante vient du séquenceur. Le fait d'avoir quatre modes différents (P, Q, S et R), oblige quasiment à avoir quatre séquenceurs différents qui acceptent des événements différents et dont il faut prévoir les transitions de l'un à l'autre. De plus étant donné le nombre d'événements et d'états différents, on obtient des pages et des pages de routines de sélection. J'ai personnellement choisi d'écrire par état une routine qui fait un 'case' en fonction de l'événement. Mais cela produit plus de dix pages de code plus un tableau d'adresse de routines pour y accéder en fonction de l'état courant.

Une dernière difficulté sur laquelle j'ai buté est le négociateur. La vérification des contraintes mais surtout la gestion des dépendances sont assez complexes.

Plusieurs de ces problèmes auraient pu être évités en diminuant les fonctionnalités de l'implémentation. C'est ce qui a été fait au niveau présentation, mais mon objectif étant d'écrire un programme qui soit facilement extensible jusqu'à la version maximale, je ne pouvais pas supprimer beaucoup de choses sans mettre en défaut cet objectif.

Finalement, à cause de tous ces problèmes rencontrés je n'ai pu arriver au bout de l'implémentation, mais j'ai pu décrire une architecture du programme qu'il me semble possible d'implémenter.

CONCLUSION

La première partie de ce mémoire portait sur l'étude des protocoles de tranfert de fichiers. En allant du général au particulier, nous avons d'abord défini un modèle général de protocole, puis nous avons vu trois exemples de différents niveaux. L'élaboration d'un protocole universel, c'est-à-dire non lié à un utilisateur ou à un constructeur particulier, nous est apparue comme une tâche complexe.

Pour limiter le travail fastidieux de lecture et de compréhension, nous n'avons pas étudié les protocoles liés à un constructeur particulier, bien que certains d'entre-eux peuvent passer pour des standards, tels SNA et DECNET.

La seconde partie du travail a permis de cerner les différents problèmes liés à l'implémentation d'un protocole de fichier, même si elle n'a pu être menée jusqu'au bout. Le résultat principal en est l'architecture décrite en annexe.

En résumé, les trois résultats principaux sont: d'abord un modèle général de protocole de transfert de fichier, qui peut servir de point de référence pour la description des protocoles particuliers, ensuite une description et une comparaison de trois protocoles qui ont ou prendront une place importante dans les transferts de fichiers, et enfin l'architecture d'un programme conforme au protocole NIFTP.

Pour prolonger la partie analytique de ce mémoire, il faudrait comparer les protocoles associés à SNA et à DECNET vis-à-vis du modèle général. En ce qui concerne la programmation, la suite de ce mémoire consisterait à achever l'implémentation de la maquette puis à l'intégrer dans un logiciel assurant toutes les couches du transfert.

BIBLIOGRAPHIE

[AVL] A. VAN LANSWERDE (1984),
"Méthodologie de Développement de Logiciels",
Notes de cours, FUNDP-Namur (non publié).

[DdG] D. de GHELLINCK (1984),
"Contribution à une réalisation
d'interconnexion de systèmes ouverts",
travail de fin d'études, FUNDP-Namur.

[FTAM] ISO/DP 8571 (1984),
"Information Processing Systems - Open Systems
Interconnection - File Transfer Access and Management",
Organisation internationale de standardisation
Document ISO/TC 97/SC 16 N 1669.

[ISO1] ISO/IS 7498,
"Systèmes de traitement de l'information -
Interconnexion des systèmes ouverts -
Modèle de référence de base de
l'interconnexion de systèmes ouverts",
Organisation internationale de standardisation.

[ISO2] ISO/DIS 8072 (1984),
"Systèmes de traitement de l'information -
Interconnexion des systèmes ouverts -
Définition du service transport",
Organisation internationale de standardisation.

[ISO3] ISO/DIS 8326 rev. (1983),
"Systèmes de traitement de l'information -
Interconnexion des systèmes ouverts -
Définition du service session de base
en mode connexion",
Organisation internationale de standardisation,
Document ISO/TC 97/SC 16 N 1742.

[ISO4] ISO/DIS 8327 rev. (1983),
"Systèmes de traitement de l'information -
Interconnexion des systèmes ouverts -
Spécification du protocole session
de base en mode connexion",
Organisation internationale de standardisation,
Document ISO/TC 97/SC 16 N 1743.

[ISO5] "Information Processing Systems - Open Systems
Interconnection - Addendum to ISO/DP 8326
covering Session Symetric Synchronisation
for the Session Service",
Organisation internationale de standardisation,
Document ISO/TC 97/SC 16 N 1687.

[ISO6] "Information Processing Systems - Open Systems
Interconnection - Addendum to ISO/DP 8327
covering Session Symetric Synchronisation
for the Session Protocol",
Organisation internationale de standardisation,
Document ISO/TC 97/SC 16 N 1688.

[ISO7] "Information Processing Systems - Open Systems
Interconnection - Presentation Protocol Specification",
Organisation internationale de standardisation'
Document ISO/TC 97/SC 16 N 1667.

[KERMIT1] F. DA CRUZ and B. CATCHINGS (1984),
"KERMIT Protocol Manual",
4th edit. rev. 1, Columbia University Center
for Computing Activities, New York.

[KERMIT2] F. DA CRUZ, D. TZOAR and B. CATCHINGS (1984),
"KERMIT User Guide",
4th edit., Columbia University Center
for Computing Activities, New York.

BIBLIOGRAPHIE

[LAYER4] B. SCOYER (1984),
"Layer4 - Résumé des Fonctions d'Interface entre les Niveaux 4 et 5",
Document de travail, FUNDP-Namur (non publié).

[LABO] P. DUNON, E. EVRARD, N. DACHOUFFE, D. DAZARD, P-Y. DESWERT et S.LIZIN (1985),
"Réalisation d'une maquette du niveau 3 de X25",
Rapport de laboratoire, FUNDP-Namur (non publié).

[MACCHI] C. MACCHI et JF. GUILBERT (1979),
"Téléinformatique",
Bordas et C.N.E.T.-E.N.S.T., Paris.

[NIFTP] (1981),
"A Network Independent File Transfer Protocol (FTP-80)"
prepared by the High Level Protocol Group as revised by the File Transfert Protocol Implementors Group of the Data Communication Protocols Unit,
National Physical Laboratory,
Teddington (Middlesex TW 11 OLW).

[TAN] AS. TANENBAUM (1980),
"Computer Networks",
Prentice-Hall Inc., Engelwood Cliffs, NJ 07632.

ANNEXE

ARCHITECTURE DU PROGRAMME

Annexe : ARCHITECTURE DU PROGRAMME

Le but de ce programme est de fournir une maquette de FTP utilisant le protocole NIFTP.

1. Architecture générale

1.1. Schéma

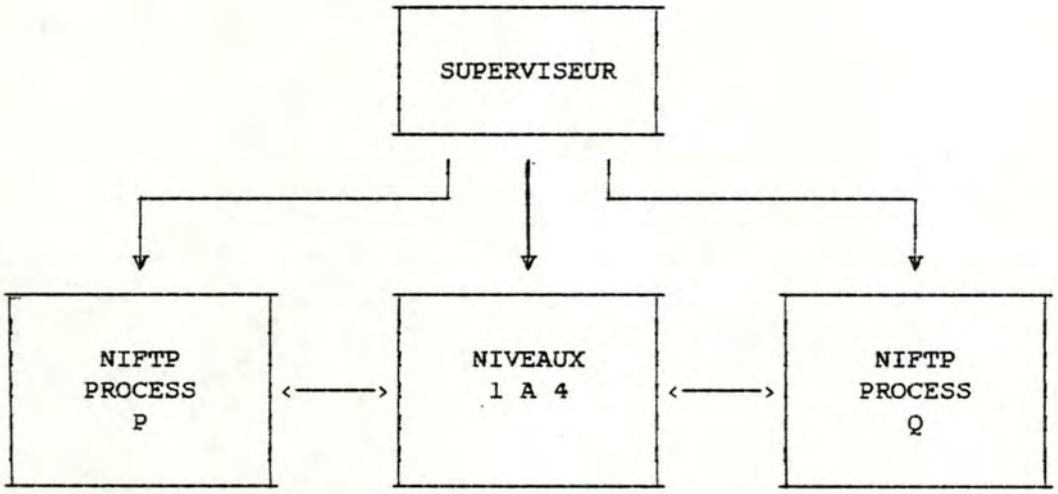


Figure A.1 - Architecture Générale.

La maquette est composée de quatre processus:

- Le processus SUPERVISEUR qui assure la gestion de l'écran et initialise les trois autres processus. Il permet à ces autres processus de travailler comme s'ils avaient chacun leur écran.

- Le processus INITIATEUR qui agit comme processus P du protocole NIFTP. C'est par lui que l'utilisateur lance les transferts.

- Le processus REPONDEUR qui agit comme processus Q du protocole NIFTP. Il travaille en tant que serveur c'est-à-dire en répondant aux demandes du processus initiateur.

- Le processus NIVEAUX_1_A_4 qui simule un service de transport c'est-à-dire assure la communication entre les processus P et Q en affichant ce qui passe sur la ligne de transmission.

La communication entre les différents processus se fait via des pipes.

2. Architecture des processus NIFTP

2.1. Structure générale

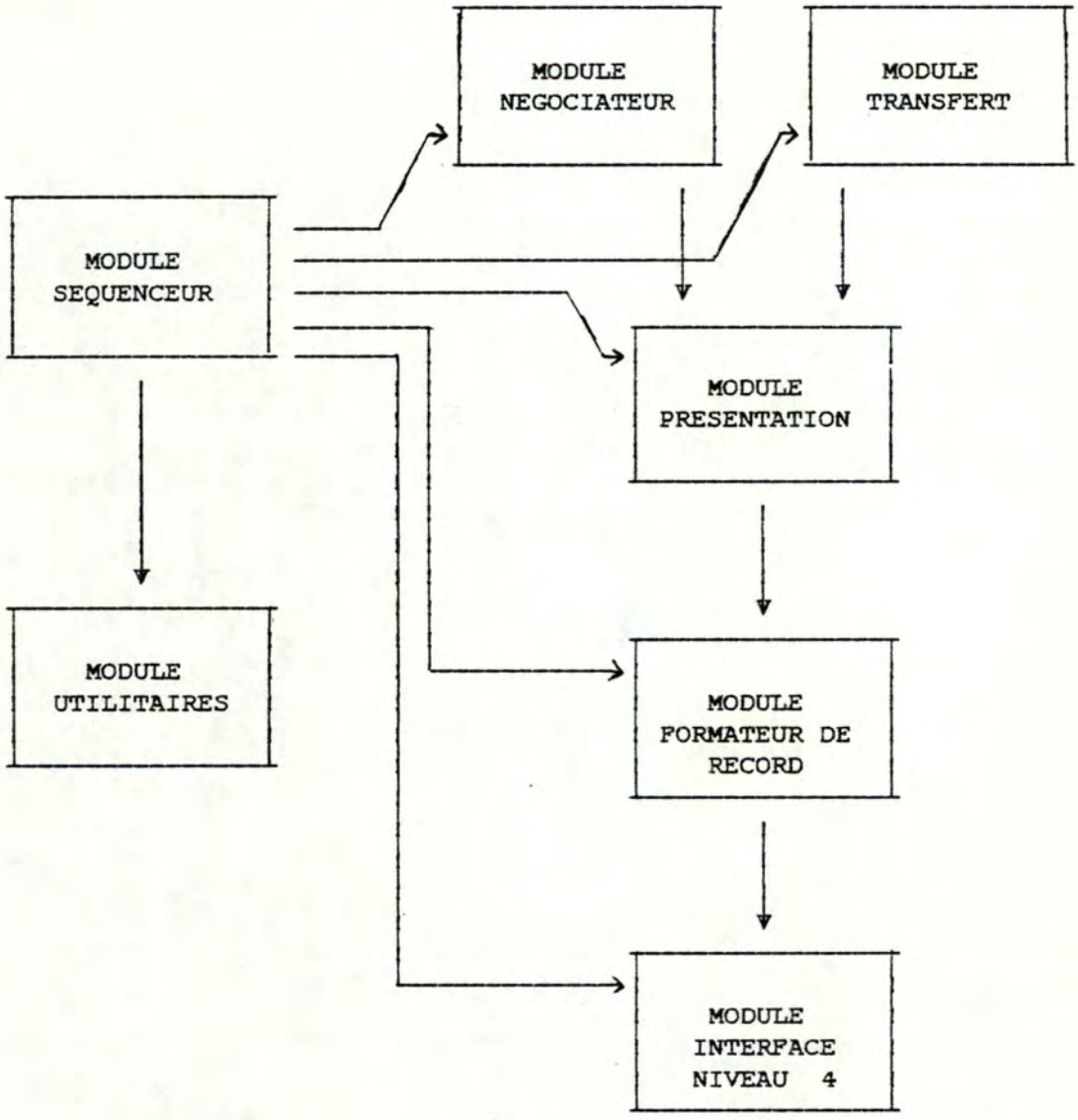


Figure A.2 - Architecture des modules NIFTP

Le processus assurant le transfert de fichier devant fonctionner dans un environnement asynchrone, je l'ai construit autour du concept d'événement. Un événement est un message venant du niveau transport, un message venant de l'écran, un timeout ou

un événement interne. Les différents modules peuvent, comme résultat de leur activation, créer des événements internes. Les événements en attente de traitement sont rangés dans une file d'attente par ordre de priorité puis par ordre chronologique. Ainsi le processus fonctionnera comme un automate qui lira les événements dans la file d'attente et les traitera en exécutant un traitement spécifique en fonction du type d'événement et de l'état dans lequel il se trouve. Ce traitement est défini dans les tables d'état du protocole.

L'architecture des processus NIFTP est construite autour des cinq modules suivants:

- Le module UTILITAIRES qui gère la file des événements, les timeouts, les erreurs et l'allocation mémoire, et qui comprend aussi l'interface avec l'écran.

- Le module SEQUENCEUR qui active les fonctions des différents modules en se basant sur la file des événements et suivant les tables d'états.

- Le module NEGOCIATEUR qui assure la négociation des différents attributs.

- Le module TRANSFERT qui assure l'accès aux fichiers ainsi que la gestion de la fenêtre.

- Le module PRESENTATION qui assure les conversions syntaxiques correspondantes à la couche présentation: conversion de code, compression, formattage de texte, etc.

- Le module FORMATEUR_DE_RECORD qui assure l'assemblage des subrecords ainsi que l'insertion des points de synchronisation et les commandes de contrôle du transfert.

- Le module INTERFACE_NIVEAU_4 qui assure l'interfaçage avec le niveau transport.

2.2. Le module UTILITAIRES

2.2.1. Le gérant de la file des événements

2.2.1.1. Structure des événements

Les événements sont représentés par la structure suivante:

```
struct event
  { evt_type      type;      /* type d'événement            */
    int           priorite;  /* priorité de l'événement     */
    time_t        temps;     /* heure d'arrivée             */
    char         *donnees;   /* pointeur vers les données   */
    int           longueur;  /* longueur des données        */
    BOOL          permanent; /* permanence des données      */
    struct event *next;      /* pointeur vers l'evt suivant */
  };

  £define EVT      struct event
```

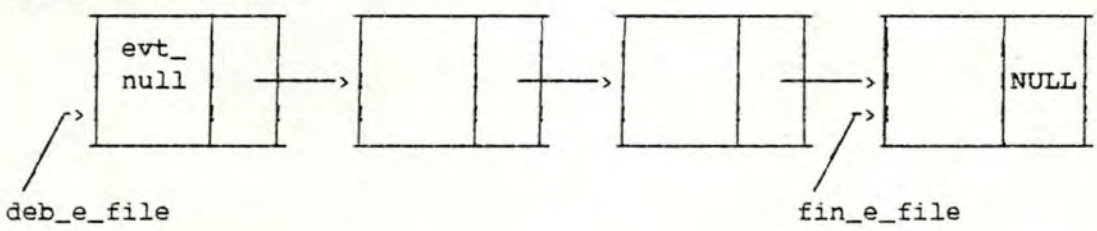
Les champs 'priorite' et 'temps' servent à ordonner la file des événements. Le champ 'donnees' est un pointeur vers une zone contenant les données liées à l'événement. Le champ 'longueur' contient la longueur de cette zone et le champ 'permanent' indique si les données doivent être détruites lorsque l'on détruit l'événement. Finalement, le champ 'next' permet de chaîner les événemts entre-eux pour construire la file.

2.2.1.2. Politique de gestion de la file des événements

Pour gérer la file on se sert d'un pointeur de début et d'un pointeur de fin et d'un élément nul. La file est construite comme une liste unidirectionnelle en servant du champ 'next'.

```
EVT evt_null = { 0, 0, 0, NULL, 0, TRUE, NULL }; /* evt vide */
EVT *deb_e_file = &evt_null;     /* debut de la file */
EVT *fin_e_file = &evt_null;     /* fin de la file   */
```

liste avec 3 éléments



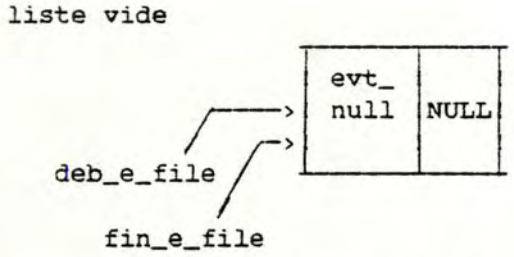


Figure A.3 - File des événements

Principes de gestion de la file des événements

- Les événements sont classés par ordre de priorité, puis pour ceux de même priorité par ordre chronologique sur le champ 'temps' et enfin, pour ceux qui ont même priorité et même champ 'temps', par ordre d'insertion.

- On retire toujours le premier élement de la file, c'est-à-dire celui qui suit l'événement null.

- L'événement null existe toujours et permet aux pointeurs deb_e_file et fin_e_file de toujours pointer vers un événement.

- La file est vide si deb_e_file et fin_e_file sont égaux.

2.2.1.3. Primitives du gérant de la file des événements

```
EVT *evt_creer(type,donnees,longueur,permanent)
---------------
    evt_type type;
    char     *donnees;
    int      longueur;
    BOOL     permanent;
```

Primitive pour créer un événement. Le champ 'priorite' est initialisé en fonction du type de l'événement. Le champ 'temps' est assigné à la valeur courante de l'horloge. Et le champ 'next' est initialisé à NULL.

Si la primitive se déroule correctement, elle renvoie un pointeur vers l'événement créé, sinon elle provoque une erreur fatale c'est-à-dire un message d'erreur à l'écran et l'arrêt du processus.

evt_detruire(p_evt)

```
EVT *p_evt;
```

Primitive pour détruire l'événement pointé par p_evt. La zone de données est également désalouée si le champ 'permanent' est à FALSE.

evt_inserer(p_evt)

```
EVT     *p_evt;
```

Primitive pour insérer l'événement pointé par p_evt dans la file suivant la politique de gestion de la file.

EVT *evt_retirer( )

Primitive pour retirer le premier élément de la file. Si la file est vide, elle renvoie le pointeur NULL, sinon elle renvoie un pointeur vers l'événement qui a été retiré de la file.

2.2.1.4. Liste des types d'événements

type	priorité et mode où l'evt peut avoir lieu	signification	données (voir plus bas)
niveau transport			
TS_data	15 PQRS	arrivée de data du niv4	*
TS_open	11 PQ	evt open de la liaison niv4	*
TS_close	1 PQRS	evt close de la liaison niv4	*
TS_reset	4 PQRS	evt reset de la liaison niv4	*
écran			
Start_FTP	12 P	démarrage du transfert	*
End	10 P	arrêt du transfert	
Set_Code	20 S	changement de code	*
Hold_up	16 R	interruption	
End_hold_up	16 R	fin d'interruption	

internes
formateur

SS_	12	R	Start of Data	*
MS_	20	R	Mark Point	*
CS_	20	R	Code Select	*
ESa	13	R	End of Data - abort	
ESe	13	R	End of Data - erreur	
ESh	13	R	End of Data - hold	
ESok	13	R	End of Data - ok	
RR_	12	S	Restart Request	*
MR_	16	S	Mark Acknowledge	*
QRa	13	S	Quit - abort	
QRe	13	S	Quit - erreur	
QRh	13	S	Quit - hold	
QRok	13	S	Quit - ok	
ERe	14	S	End Acknowledge - erreur	
ERh	14	S	End Acknowledge - hold	
ERok	14	S	End Acknowledge - ok	
SS_data	20	PQR	Subrecord de data	*

module presentation

Data	17	PQR	Données retransformées	*
Data_NOK	10	R	Données invalides	

module transfert

Ack_W_End	17	S	La limite de la fenêtre est atteinte	
Data_End	10	S	Fin des données	
Data_OK	20	S	Données prêtes à être envoyées	
End_OK	14	R	La fin du fichier est sauvée	
Mark_Pnt	20	S	Point de synchronisation atteint	
OK_for_Ack	11	R	Un Mark Acknowledge peut être envoyé	
Rest_Pnt	11	S	Le point de reprise est atteint (après RR)	

module négociateur

STOP	14	Q	Request Termination	*
GO	14	Q	Start Data Transfer	
RPOS	14	P	Positive Reply	*
RNEG	14	P	Negative Reply	*
SFT	14	Q	Start File Transfer	*
STOPACK	14	P	Acknowledge Termination	
RPOS_OK	13	P	RPOS valide	
RPOS_NOK	13	P	RPOS non valide	
RPOS_EOF	13	P	RPOS en fin de fichier	
SFT_OK	13	Q	SFT valide	
SFT_NOK	13	Q	SFT non valide	

```
module séquenceur
 Entry        3  P            Généré automatiquement après
                                  l'état P_OK ou P_FAIL

module utilitaires
 Timed_out    5 PQRS          Un timeout a expiré              *
 Error        2 PQRS          Erreur de protocole              *
```

Signification des données attachées aux événements

- TS_data : paquet de données
- TS_open : message à l'ouverture
- TS_reset : motif du reset
- Start_FTP : Send ou Receive + les fichiers + options
- Set_Code : Le code à employer
- SS_, MS_, RR_ et MR_ : Le numéro du point de synchronisation
- CS_ : Le numéro du code à utiliser
- SS_data : Le contenu du subrecord
- Data : Les données à ecrire dans le fichier
- STOP, RPOS et SFT : La liste des paramètres
- RNEG : Les paramètres fautifs
- Timed_out : Le motif du timeout
- Error : Le motif de l'erreur

La liste des priorités

mode	P	Q	R	S
1	TS_close	TS_close	TS_close	TS_close
2	Error	Error	Error	Error
3	Entry			
4	TS_reset	TS_reset	TS_reset	TS_reset
5	Timed_out	Timed_out	Timed_out	Timed_out
10	End		Data_NOK	Data_End
11	TS_open		OK_for_Ack	Rest_Point
12	Start_FTP	TS_open	SS	RR
13	Cmd_traitée	Cmd_traitée	ES	QR
14	SS_data	SS_data	End_OK	ER
15	TS_data	TS_data	TS_data	TS_data
16			Hold_up/End_hold_up	MR
17			Data	Ack_W_End
20			CS	Set_code
			SS_data	Data_OK
			MS	Mark_Pnt

2.2.2. Le gérant des timeouts

2.2.2.1. Structure des timeouts

Les timeouts sont représentés par la même structure que les événements:

```
struct event
  { evt_type      type;      /* type de timeout
    int           priorite;  /* toujours = à 5 pour les timeouts
    time_t        temps;     /* temps d'expiration
    char         *donnees;   /* pointeur vers les données
    int           longueur;  /* longueur des données
    BOOL          permanent; /* permanence des données (TRUE)
    struct event *next;      /* pointeur vers le timeout suivant
  };

  #define TIMEOUT  struct event
```

Les timeouts ont la même structure que les événements pour pouvoir facilement les insérer dans la file des événements lors de leur expiration. Le champ 'temps' sert à ordonner la file des timeouts. Le champ 'donnees' est un pointeur vers une zone contenant les données liées au timeout. Il s'agit d'un string donnant le motif du timeout. Le champ 'longueur' contient la longueur de cette zone et le champ 'permanent' est à TRUE puisqu'il s'agit d'un message fixe en mémoire. Finalement, le champ 'next' permet de chaîner les timeouts entre-eux pour construire la file.

2.2.2.2. Politique de gestion de la file des timeouts

La file est construite de la même façon que celle des événements, c'est-à-dire que pour gérer la file on se sert d'un pointeur de début, d'un pointeur de fin et d'un élément nul, et la file est construite comme une liste unidirectionnelle en se servant du champ 'next'.

```
EVT to_null = { 0, 0, 0, NULL, 0, TRUE, NULL }; /* to vide */
EVT *deb_t_file = &to_null; /* debut de la file */
EVT *fin_t_file = &to_null; /* fin de la file   */
```

cfr figure A.3

Principes de gestion de la file des timeouts

- Les timeouts sont classés par ordre chronologique sur le champ 'temps', c'est-à-dire en fonction de leur moment d'expiration. Ceux qui ont même champ 'temps', sont classés par ordre d'insertion.

- A chaque mouvement de la file (insertion, expiration ou effacement), on fait expirer les timeouts dont le temps d'expiration est dépassé et on remet l'alarme à l'heure d'expiration du premier timeout de la liste.

- L'événement null existe toujours et permet aux pointeurs deb_t_file et fin_t_file de toujours pointer vers un timeout.

- La file est vide si deb_t_file et fin_t_file sont égaux.

2.2.2.3. Primitives du gérant de la file des timeouts

to_inserer(type)

```
to_type type;
```

Primitive pour créer un timeout et l'insérer dans la file. Le champ 'type' est initialisé par le paramètre type. Le champ 'temps' initialisé à la valeur courante de l'horloge augmentée de la valeur de délai correspondante au type de timeout. Les champs 'donnees' et 'longueur' sont initialisés par un string en fonction du type de timeout. Le champ 'permanent' est initialisé à TRUE. Puis le champ 'next' est utilisé pour insérer le timeout dans la file en suivant la politique définie ci-dessus.

STATUS to_effacer(type)

```
evt_type        type;
```

Primitive pour effacer les timeouts d'un type donné lorsque l'événement attendu est arrivé.

to_expirer( )

Primitive appelée lors d'une interruption produite par l'expiration de l'alarme et qui transfère les timeouts qui ont expirés dant la file des événements.

set_alarm( )

Primitive appelée par les autres primitives pour remettre à jour l'alarme après toute manipulation de la file.

2.2.2.4. Liste des types de timeouts

type	signification	durée
mode P ou Q		
W_P_REPLY	attente de GO ou STOP	?
W_Q_REPLY	attente de RPOS ou RNEG	?
W_SFT	attente de SFT	?
W_STOPACK	attente de STOPACK	?
W_STOP	attente de STOP	?
W_Start_FTP	attente de Start_FTP	?
W_TS_open	attente de TS_open	?
mode S ou R		
W_DATA	attente de Data	?
W_ERe	attente de ER(E)	?
W_ERh	attente de ER(H)	?
W_ERok	attente de ER(OK)	?
W_ESe	attente de ES(E)	?
W_ESh	attente de ES(H)	?
W_ESh_or_SS	attente de ES(H) ou SS	?
W_ESok	attente de ES(OK)	?
W_MR	attente de MR	?
W_RR_or_QR	attente de RR ou QR( )	?
W_SS	attente de SS	?
W_Hold_up	attente de End_Hold_up	?

2.2.3. Le gérant d'erreur

Toutes les erreurs détectées dans les autres modules que le module UTILITAIRES font appel à la fonction suivante:

```
E_error(e_string,e_longueur,e_permanence)
--------

    char    *e_string;
    int      e_longueur;
    BOOL     e_permanence;
```

Elle a pour effet de générer un événement 'Error' dont les données sont le message 'e_string' de longueur 'e_longueur'. Les erreurs rencontrées dans le module UTILITAIRES ne peuvent être traitées de cette façon et provoquent plutôt un message à l'écran suivit de l'arrêt du processus.

2.2.4. Le gérant de la mémoire

2.2.4.1. Objectifs

L'allocation dynamique de mémoire est utilisée d'une part lors de la création d'événements et de timeouts et d'autre part pour stocker les données et messages. La création d'événements et de timeouts nécessite l'allocation de structures event donc de blocs de taille fixe, tandis que pour les données, on alloue des zones de caractères de longueur variable.

2.2.4.2. Primitives du gérant de la mémoire

```
char *malloc(size)
-----------
    unsigned size;
```

La primitive malloc alloue une zone d'au moins size bytes commencant à une limite de mot et retourne un pointeur vers cette zone. S'il ne reste plus assez de place libre contiguë, malloc retourne le pointeur NULL.

```
free(ptr)
----
    char *ptr;
```

La primitive free libère une zone de mémoire allouée par malloc et pointée par ptr. Cette zone pourra être realloué par malloc. Il est indispensable que ptr soit un pointeur alloué par malloc.

remarque:

La routine malloc allouant des zones de mémoire en un seul morceau, la fonction pour faire pointer ptr sur le caractère suivant est simplement ptr++ .

2.2.4.3. Remarque

Lorsqu'on a atteint la limite de la mémoire, on doit tout arrêter car on ne plus allouer d'événements. Pour éviter celà on pourrait imaginer que lorsque la mémoire utilisée atteint une certaine limite, on prenne des mesures de sauvegarde telles que bloquer l'arrivée de messages du niveau transport jusqu'au moment où ayant traité un certain nombre d'événements, suffisament de mémoire ait été libérée. D'autre part comme les routines malloc et free ne font pas de compactage de la mémoire, celle-ci n'est pas utilisée très efficacement. On pourrait remplacer l'allocation de zones de longueur variable par l'allocation de suites de blocs de taille fixe. Mais il faudrait alors écrire une routine qui puisse donner le caractère suivant d'un caractère donné.

2.2.5. Le gérant d'écran

2.2.5.1. Objectifs

Le gérant d'écran a deux fonctions, d'une part l'envoi de messages à l'écran et d'autre part la réception des commandes de l'utilisateur. L'écriture à l'écran se fait dans des zones prédéfinies qui sont:

Z_INFO	zone des messages d'information
Z_ERROR	zone des messages d'erreur
Z_ETAT	zone indiquant l'état courant
Z_MODE	zone indiquant le mode courant
Z_EVT	zone indiquant l'événement courant
Z_STAT	zone indiquant le nombre de marques envoyées
Z_PROMPT	zone où on affiche le prompt

Divers paramètres sont également disponibles, tels que la longueur de chaque zone ainsi que la largeur et la longueur de l'écran.

2.2.5.2. Primitives du gérant d'écran

SC_write(zone,string)

```
ZONE    zone;
char    *string;
```

Cette primitive a pour effet d'afficher à l'écran le message string dans la zone indiquée. Si le string est trop long il est tronqué à la longueur de la zone et si la zone n'existe pas, rien n'est affiché à l'écran.

SC_read( )

Cette primitive est appelée chaque fois qu'un message arrive de l'écran. Les messages ont la forme d'une ligne de commande. Les commandes acceptées sont les suivantes:

```
En mode P
  Send    filename1 filename2 options    Envoi d'un fichier
  Receive filename1 filename2 options    Réception d'un fichier
  End                                    Arrêt du transfert

En mode Q
  End                                    Arrêt du transfert

En mode R
  Hold                                   Interruption du transfert
  End-hold                               Fin d'interruption
```

filename1 est le nom du fichier existant
filename2 est le nom de la copie à créer

options est une liste d'options commencant chacune par /

les options disponibles sont:

/USER-NAME:
/ACCOUNT:
/FILE-PASSW:
/USER-PASSW:
/ACCOUNT-PASSW:
/ACCES-MODE:
/TTCODE:
/TXT-FORMAT:
/BIN-FORMAT:
/ACK-WINDOW:
/DATA-TYPE:

2.2.5.3. Remarque

L'interface avec l'écran se fait dans la maquette à travers le processus superviseur qui est relié à ce processus-ci via un pipe. Dans le sens venant de l'écran les messages ont la forme d'une suite de caractères terminée par un zéro. Dans l'autre sens, les messages ont la forme d'un octet représentant la zone suivi d'une suite de caractères à afficher terminée par un zéro.

2.3. Le module SEQUENCEUR

2.3.1. Description

Le module SEQUENCEUR est essentielement construit autour de tables d'états. Il lit les événements un à un hors de la file des événements puis en fonction du type d'événement et de l'état dans lequel il se trouve, il active différentes fonctions des autres modules et change éventuellement d'état. Les tables d'états indiquent, en fonction de l'événement et de l'état courant, les actions à effectuer et le nouvel état. Il y a quatre modes pour le séquenceur et quatre tables qui y correspondent. Ces quatre modes sont P pour initialiser le transfert en tant que initiateur, Q pour initialiser le transfert en tant que répondeur, S pour la phase de transfert en tant qu'envoyeur (Sender) et R pour la phase de transfert en tant que receveur (Receiver). Si le processus est lancé en tant qu'initiateur, il passe succesivement dans les modes P-S-P s'il s'agit d'envoyer un fichier et P-R-P s'il s'agit de le recevoir. Mais si le processus est lancé en tant que répondeur alors il passe dans le premier cas par Q-S-Q et dans le second par Q-R-Q.

Certains événements n'entrent pas directement dans les tables d'états mais doivent d'abord être transformés. Il s'agit des événements TS_data, SS_data et TS_close. L'événement TS_close provoque un message d'erreur à l'écran puis le passage direct à l'état P_IDLE ou Q_IDLE puisque la connexion n'est plus disponible. L'événement TS_data est quant à lui passé au module FORMATEUR DE RECORD qui le décompose en les événements suivants: SS_, MS_, CS_, ESa, ESe, ESh, SEok, RR_, MR_, QRa, QRe, QRh, QRok, ERe, ERh, ERok et SS_data. L'événement SS_data est passé au module PRESENTATION pour en faire un événement Data qui dans les modes P ou Q, est passé directement au module NEGOCIATEUR.

2.3.2. Tables d'états

2.3.2.1. Mode P

Liste des états

P_IDLE Pas de connexion au niveau transport. Seul l'événement TS_open est reconnu.

P_OPEN La connexion au niveau transport est ouverte. Le transfert est possible dans les deux directions. La première chose attendue est l'envoi du SFT. La liaison peut être fermée si une erreur survient ou si elle n'est plus nécessaire. Cet état est normalement réentré après la fin de la phase de terminaison de façon à ce que la connexion puisse être réutilisée si nécessaire.

P_SFT Un SFT a été envoyé, une réponse est attendue et un timeout peut être lancé.

P_GO C'est un état de transition marquant la fin de la phase d'initialisation, il est transformé en S_GO ou R_GO en entrant dans la phase de transfert.

P_STOP Un STOP a été envoyé, un STOPACK est attendu et un timeout peut être lancé.

P_OK Le transfert s'est terminé correctement avec ER(ok). L'événement Entry a lieu immédiatement à l'entrée dans cet état, à moins qu'il n'y ait un événement Error.

P_FAIL Le transfert a échoué, la dernière commande étant un ER(e), ES(a) ou QR(a). L'événement Entry a lieu immédiatement à l'entrée dans cet état, à moins qu'il n'y ait un événement Error.

Liste des actions

SFT Envoyer une commande SFT.

GO Envoyer une commande GO.

STOPt Envoyer une commande STOP avec l'état du transfert indiquant 'terminated' si nécessaire.

STOPr Envoyer une commande STOP avec l'état du transfert indiquant 'rejected' si nécessaire.

STOPa Envoyer une commande STOP avec l'état du transfert indiquant 'aborted' si nécessaire.

CLOSE Fermer la connexion du service transport.

	P_IDLE	P_OPEN	P_SFT	P_STOP	P_OK	P_FAIL
RPOS_OK	*	*	GO P_GO	*	*	*
RPOS_NOK	*	*	STOPr P_STOP	*	*	*
RPOS_EOF	*	*	STOPt P_STOP	*	*	*
RNEG	*	*	STOPr P_STOP	*	*	*
STOPACK	*	*	*	- P_OPEN	*	*
Start_FTP	- -	SFT P_SFT	- -	*	*	*
TS_open	- P_OPEN	- -	- -	- -	- -	- -
End	- -	CLOSE P_IDLE	CLOSE P_IDLE	CLOSE P_IDLE	CLOSE P_IDLE	CLOSE P_IDLE
Entry	- -	*	*	- -	STOPt P_STOP	STOPa P_STOP
Inv_Com	- P_IDLE	CLOSE P_IDLE	CLOSE P_IDLE	CLOSE P_IDLE	- -	- -
TS_reset	- -	- P_OPEN	CLOSE P_IDLE	CLOSE P_IDLE	- -	- -
Timed_out	- -	- -	CLOSE P_IDLE	CLOSE P_IDLE	- -	- -
Error	- -	CLOSE P_IDLE	CLOSE P_IDLE	CLOSE P_IDLE	STOPa P_STOP	CLOSE P_IDLE

Inv_Com = Commande Invalide c'est-à-dire tout autre événement.

* = L'occurence de cet événement dans cet état doit être traitée comme une Inv_Com.

2.3.2.2. Mode Q

Liste des états

Q_IDLE Pas de connexion au niveau transport. Seul l'événement TS_open est reconnu.

Q_OPEN La connexion au niveau transport est ouverte. Une commande SFT est attendue et un timeout peut être lancé. Cet état est normalement réentré après la fin de la phase de terminaison de façon à ce que le processus P puisse répondre à la commande STOPACK avec un autre SFT ou en fermant la connexion du service de transport.

Q_RPOS Un RPOS a été envoyé, une commande GO ou STOP est attendue et un timeout peut être lancé.

Q_RNEG Un RNEG a été envoyé, une commande STOP est attendue et un timeout peut être lancé.

Q_GO C'est un état de transition marquant la fin de la phase d'initilisation, il est transformé en S_GO ou R_GO en entrant dans la phase de transfert.

Q_OK Le transfert s'est terminé correctement avec un ER(ok).

Q_FAIL Le transfert a échoué, la dernière commande étant ER(e), ES(a) ou QR(a).

Liste des actions

RPOS Envoyer une commande RPOS.

RNEG Envoyer une commande RNEG.

STOPACK Envoyer une commande STOPACK.

CLOSE Fermer la connexion du service transport.

	Q_IDLE	Q_OPEN	Q_RPOS	Q_RNEG	Q_OK	Q_FAIL
SFT_OK	*	RPOS Q_RPOS	*	*	*	*
SFT_NOK	*	RNEG Q_RNEG	*	*	*	*
GO	*	*	- Q_GO	*	*	*
STOP	*	*	STOPACK Q_OPEN	STOPACK Q_OPEN	STOPACK Q_OPEN	STOPACK Q_OPEN
TS_open	- Q_OPEN	- -	- -	- -	- -	- -
Inv_Com	- Q_IDLE	CLOSE Q_IDLE	CLOSE Q_IDLE	CLOSE Q_IDLE	CLOSE Q_IDLE	CLOSE Q_IDLE
TS_reset	- -	CLOSE Q_IDLE	CLOSE Q_IDLE	CLOSE Q_IDLE	CLOSE Q_IDLE	CLOSE Q_IDLE
Timed_out	- -	CLOSE Q_IDLE	CLOSE Q_IDLE	CLOSE Q_IDLE	CLOSE Q_IDLE	CLOSE Q_IDLE
Error	- -	CLOSE Q_IDLE	CLOSE Q_IDLE	CLOSE Q_IDLE	CLOSE Q_IDLE	CLOSE Q_IDLE

2.3.2.3. Mode S

Liste des états

S_GO Cet état est l'entrée en venant de la phase d'initialisation.

S_DATA Les données peuvent être envoyées.

S_RR Une commande RR a été reçue.

S_HOLD1 Un ES(h) a été envoyé en réponse à un QR(h), la commande ER(h) est attendue et un timeout peut être lancé.

S_HOLD2 Idem à S_HOLD1 mais le QR(h) est arrivé alors qu'on était dans l'état S_ESok.

S_HORR Un ES(h) a été envoyé et une commande RR reçue, la commande ER(h) est attendue et un timeout peut être lancé.

S_MR On a atteint la limite de la fenêtre (Acknowledgement Window), une commande MR est attendue et un timeout peut être lancé.

S_ESok Un ES(ok) a été envoyé, une commande ER(ok) est attendue et un timeout peut être lancé.

S_WAIT En attente après un reset du service transport, Une commande RR ou QR est attendue et un timeout peut être lancé.

S_ESe Un ES(e) a été lancé, une commande ER(e) est attendue et un timeout peut être lancé.

S_OK C'est un état de transition marquant la fin du transfert après la réception d'un ER(ok), il est transformé en P_OK ou Q_OK en entrant dans la phase de terminaison.

S_FAIL C'est un état de transition après l'échec du transfert (un ER(e) ou un QR(a) a été reçu ou un ES(a) a été envoyé), il est transformé en P_FAIL ou Q_FAIL en entrant dans la phase de terminaison.

Liste des actions

SS Envoyer une commande SS.

CS Envoyer une commande CS.

MS Envoyer une commande MS.

ESok Envoyer une commande ES(ok).

ESh Envoyer une commande ES(h).

ESe Envoyer une commande ES(e).

ESa Envoyer une commande ES(a).

AWT Avancer la limite inférieure de la fenêtre (last mark acknowledged).

AWL Avancer la limite supérieure de la fenêtre (last mark send).

RWL Rétablir la limite supérieure de la fenêtre (last mark send) et reposionner le pointeur dans le fichier.

DATA Envoyer le subrecord de données suivant.

	S_GO	S_DATA	S_RR	S_HOLD1	S_HOLD2	S_HORR
MR_	*	AWT S_DATA	AWT S_RR	AWT S_HOLD1	AWT S_HOLD2	AWT S_HORR
RR_	*	RWL S_RR	*	RWL S_HORR	RWL S_HORR	*
QRa	- S_FAIL	- S_FAIL	- S_FAIL	- S_FAIL	- S_FAIL	- S_FAIL
QRe	ESe S_ESe	ESe S_ESe	ESe S_ESe	ESe S_ESe	ESe S_ESe	ESe S_ESe
QRh	SS+ESh S_HOLD1	ESh S_HOLD1	ESh S_HORR	- S_HOLD1	- S_HOLD2	- S_HORR
QRok	SS+ESok S_ESok	ESok S_ESok	*	- S_HOLD1	- S_HOLD2	*
ERe	*	*	*	*	*	*
ERh	*	*	*	- S_DATA	ESok S_ESok	- S_RR
ERok	*	*	*	*	*	*
Data_End	- -	ESok S_ESok	- -	- -	- -	- -
Mark_Pnt	SS S_DATA	MS+AWL S_DATA	- -	- -	- -	- -
Rest_Pnt	- -	- -	SS S_DATA	- -	- -	- HORR
Ack_W_End	- -	- MR	- -	- -	- -	- -
Data_OK	- -	DATA S_DATA	- -	- S_HOLD1	- -	- -
Set_Code	- -	CS DATA	- -	- -	- -	- -

	S_GO	S_DATA	S_RR	S_HOLD1	S_HOLD2	S_HORR
Inv_Com	ESe S_ESe	ESe S_ESe	ESe S_ESe	ESe S_ESe	ESe S_ESe	ESe S_ESe
TS_reset	- S_WAIT	- S_WAIT	- S_WAIT	- S_WAIT	- S_WAIT	- S_WAIT
Timed_out	- -	- -	- -	ESa S_FAIL	ESa S_FAIL	ESa S_FAIL
Error	ESe S_ESe	ESe S_ESe	ESe S_ESe	ESe S_ESe	ESe S_ESe	ESe S_ESe

	S_MR	S_ESok	S_WAIT	S_ESe
Inv_Com	ESe S_ESe	ESe S_ESe	ESe S_ESe	- S_ESe
TS_reset	- S_WAIT	- S_WAIT	- S_WAIT	- S_WAIT
Timed_out	ESa S_FAIL	ESa S_FAIL	ESa S_FAIL	ESa S_FAIL
Error	ESe S_ESe	ESe S_ESe	ESe S_ESe	- S_ESe

	S_MR	S_ESok	S_WAIT	S_ESe
MR_	AWT S_DATA	AWT S_ESok	AWT S_WAIT	– S_ESe
RR_	RWL S_RR	RWL S_RR	RWL S_RR	– S_ESe
QRa	– S_FAIL	– S_FAIL	– S_FAIL	– S_FAIL
QRe	ESe S_ESe	ESe S_ESe	ESe S_ESe	– S_ESe
QRh	ESh S_HOLD1	ESh S_HOLD2	ESh S_HOLD1	– S_ESe
QRok	ESok S_ESok	– S_ESok	ESok S_ESok	– S_ESe
ERe	*	*	*	– S_FAIL
ERh	*	*	*	– S_ESe
ERok	*	– S_OK	*	– S_OK
Data_End	– –	– –	– –	– S_ESe
Mark_Pnt	– S_MR	– –	– –	– S_ESe
Rest_Pnt	– –	– –	– –	– S_ESe
Ack_W_End	– –	– –	– –	– S_ESe
Data_OK	– S_MR	– –	– S_WAIT	– S_ESe
Set_Code	– –	– –	– –	– S_ESe

2.3.2.4. Mode R

Liste des états

R_GO — Cet état est l'entrée de la phase d'initialisation. Une commande SS est attendue et un timeout peut être lancé.

R_DATA — Des données sont attendues et un timeout peut être lancé.

R_RR — Une commande SS est attendue, normalement parce que une commande RR a été envoyée, et un timeout peut être lancé.

R_PEND — Un QR(h) a été envoyé, une commande ES(h) est attendue et un timeout peut être lancé.

R_HOLD — Une commande ES(h) a été reçue en réponse à un QR(h). Aucun input n'est attendu.

R_PERR — Les commandes ES(h) et SS sont attendues, les commandes QR(h) et RR ont étées envoyées et un timeout peut être lancé.

R_HORR — La commande ES(h) a été reçue et une commande SS est attendue, normalement après l'envoi d'un RR, mais aucun input n'est attendu dans cet état (même le SS).

R_ESok — Une commande ES(ok) a été reçue. Aucun input n'est attendu. On attend l'écriture de la fin du fichier.

R_QRok — Un QR(ok) a été envoyé, une commande ES(ok) est attendue et un timeout peut être lancé.

R_QRe — Un QR(e) a été envoyé, un ES(e) est attendu et un timeout peut être lancé.

R_OK — C'est un état de transition après l'envoi d'un ER(ok), il est transformé en P_OK ou Q_OK en entrant dans la phase de terminaison.

R_FAIL — C'est un état de transition après l'échec du transfert (un ER(e) ou un QR(a) a été envoyé ou un ES(a) a été reçu), il est transformé en P_FAIL ou Q_FAIL en entrant dans la phase de terminaison.

Liste des actions

MR — Envoyer une commande MR et avancer la limite inférieure de la fenêtre si nécessaire.

RR — Envoyer une commande RR.

QRok	Envoyer une commande QR(ok).
QRh	Envoyer une commande QR(h).
QRe	Envoyer une commande QR(e).
QRa	Envoyer une commande QR(a).
ERok	Envoyer une commande ER(ok).
ERh	Envoyer une commande ER(h).
ERe	Envoyer une commande ER(e).
Save	Enregistrer le code.
Keep	Vérifier les données si nécessaire et les mémoriser.
AWL	Avancer la limite supérieure de la fenêtre (last mark receveid) et noter la position de la marque dans les données.
RWL	Rétablir la limite supérieure de la fenêtre (last mark received) et reposionner le pointeur dans le fichier.

	R_DATA	R_RR	R_PEND	R_HOLD	R_PERR	R_HORR
SS_	* R_DATA	RWL R_DATA	*	*	RWL R_PEND	*
MS_	AWL R_DATA	– R_RR	AWL R_PEND	*	– R_PERR	*
CS_	Save R_DATA	– R_RR	Save R_PEND	*	– R_PERR	*
ESa	– R_FAIL	– R_FAIL	– R_FAIL	– R_FAIL	– R_FAIL	– R_FAIL
ESe	ERe R_FAIL	ERe R_FAIL	ERe R_FAIL	ERe R_FAIL	ERe R_FAIL	ERe R_FAIL
ESh	– R_DATA	– R_RR	– R_HOLD	– R_HOLD	– R_HORR	– R_HORR
ESok	– R_ESok	– R_RR	– R_PEND	*	– R_PERR	*
Data	Keep R_DATA	– R_RR	Keep R_PEND	*	– R_PERR	*
OK_for_Ack	MR R_DATA	MR R_RR	MR R_PEND	MR R_HOLD	MR R_PERR	MR R_HORR
Hold_up	QRh R_PEND	QRh R_PERR	– –	– –	– –	– –
End_hold_up	– –	– –	ERh R_DATA	ERh R_DATA	ERh R_RR	ERh R_RR
End_OK	QRok R_QRok	– –	ERh+QRok R_QRok	ERh+QRok R_QRok	– –	– –
Data_NOK	RR R_RR	– –	RR R_PERR	RR RHORR	– –	– –
Inv_Com	QRe R_QRe	QRe R_QRe	QRe R_QRe	QRe R_QRe	QRe R_QRe	QRe R_QRe
TS_reset	MR+RR R_RR	MR+RR R_RR	MR+RR R_RR	MR+QRh R_PEND	MR+RR R_RR	MR+RR R_RR
Timed_out	QRa R_FAIL	QRa R_FAIL	QRa R_FAIL	– –	QRa R_FAIL	– –
Error	QRe R_QRe	QRe R_QRe	QRe R_QRe	QRe R_QRe	QRe R_QRe	QRe R_QRe

	R_ESok	R_QRok	R_QRe	R_GO
SS_	*	– R_QRok	*	– R_DATA
MS_	*	– R_QRok	*	*
CS_	*	– R_QRok	*	*
ESa	– R_FAIL	– R_FAIL	– R_FAIL	– R_FAIL
ESe	ERe R_FAIL	ERe R_FAIL	ERe R_FAIL	ERe R_FAIL
ESh	– R_ESok	– R_QRok	*	*
ESok	– R_ESok	ERok R_OK	*	*
Data	–	– R_QRok	*	*
OK_for_Ack	MR R_ESok	– R_QRok	*	– –
Hold_up	QRh R_PEND	– R_QRok	– –	QRh R_PERR
End_hold_up	– –	– –	– –	– –
End_OK	ERok R_OK	– –	*	QRok R_QRok
Data_NOK	RR R_RR	– –	*	– –
Inv_Com	QRe R_QRe	QRe R_QRe	– R_QRe	QRe R_QRe
TS_reset	QRok R_QRok	QRok R_QRok	QRe R_QRe	RR R_RR
Timed_out	– –	QRa R_FAIL	QRa R_FAIL	QRa R_FAIL
Error	QRe R_QRe	QRe R_QRe	– R_QRe	QRe R_QRe

2.3.3. Changement d'état

La plupart du temps lors du changement d'état, il faut faire une action spécifique, entre autre lancer des timeouts.

On lance un timeout en entrant dans les états suivant:

```
P_SFT   -> timeout de type W_Q_REPLY
P_STOP  -> timeout de type W_STOPACK
Q_OPEN  -> timeout de type W_SFT
Q_RPOS  -> timeout de type W_P_REPLY
Q_RNEG  -> timeout de type W_P_REPLY
S_HOLD1 -> timeout de type W_ERh
S_HOLD2 -> timeout de type W_ERh
S_HORR  -> timeout de type W_ERh
S_MR    -> timeout de type W_MR
S_ESok  -> timeout de type W_ERok
S_WAIT  -> timeout de type W_RR_or_QR
S_ESe   -> timeout de type W_ERe
R_DATA  -> timeout de type W_DATA
R_RR    -> timeout de type W_SS
R_PEND  -> timeout de type W_ESh
R_PERR  -> timeout de type W_ESh_or_SS
R_QRok  -> timeout de type W_ESok
R_QRe   -> timeout de type W_ESe
R_GO    -> timeout de type W_SS
```

En outre, on peut lancer un timeout en entrant dans les états suivants:

```
P_IDLE  -> timeout de type W_TS_open
P_OPEN  -> timeout de type W_Start_FTP
Q_IDLE  -> timeout de type W_TS_open
Q_OK    -> timeout de type W_STOP
Q_FAIL  -> timeout de type W_STOP
R_HOLD  -> timeout de type W_Hold_up
R_HORR  -> timeout de type W_Hold_up
```

Cas particuliers:

```
P_OK    -> créer un événement Entry
P_FAIL  -> créer un événement Entry
S_DATA  -> lire un bloc de données du fichier
```

Finalement, les états du type GO, OK et FAIL servent de transition entre les modes. Ainsi l'entrée dans l'état P_GO ou Q_GO provoque un changement de mode vers S ou R et le passage à l'état S_GO ou R_GO. De même l'entrée dans l'état S_OK ou R_OK (respec. S_FAIL ou R_FAIL), provoque un changement de mode vers P ou Q et le passage à l'état P_OK ou Q_OK (respec. P_FAIL ou Q_FAIL).

2.4. Le module INTERFACE NIVEAU 4

2.4.1. Description

Le but du module INTERFACE NIVEAU 4 est de fournir un interface standard avec le niveau 4 c'est-à-dire de cacher pour le reste du processus NIFTP l'implémentation de l'interface avec le niveau transport. De cette façon, il pourra facilement être inséré dans d'autres maquettes ou dans un système complet. Les fonctions logiques de l'interface sont basées sur la définition faite dans le document de Béatrice Scoyer intitulé 'layer4' [LAYER4]. Du point de vue de l'implémentation de la maquette l'interface se fait via un pipe. Dans le but de faciliter une intégration avec la maquette implémentant le niveau 3 faite par le labo de 2e licence [LABO], j'ai pris la même définition pour les messages qui transitent dans le pipe.

2.4.2. Spécification logique des fonctions d'interface avec le niveau 4

2.4.2.1. Interface 4 - 5

Les fonctions pour l'interface 5 - 4 nécéssitées par la réalisation de cette maquette sont les mêmes que celles définies dans le document de Béatrice Scoyer dans lesquelles on a supprimé ce qui n'est pas utilisé dans la maquette c'est-à-dire :

- la notion de qualité de service.
- le service de données expresses.
- l'attente et réception de données (wait and receive)

Ce choix a entrainé la supression des fonctions :

TR_SNDXD
TR_RCVXD
TR_WRCVND
TR_WRCVXD

et des valeurs de status:

ErrQosLoc
ErrQosRem
ErrQos
ErrNotAvail

De plus, la fonction TR_INF_EVENT a été modifiée en y ajoutant l'événement data available qui remplace ainsi la fonction TR_INF_DTAVAIL. Il faut encore noter que le contrôle de flux dans le sens 4 vers 5 n'est pas utilisé dans cette version de la maquette mais pourrait l'être dans une version ultérieure (cfr la gestion de la mémoire en 2.2.4.3).

2.4.2.2. Résumé des fonctions interface

1. couche_5 —> couche_4 : services

- status = TR_CONNECT (loctrad, remtrad, qos_param, msg, lnmsg
  var exp_data, var idtrcon)
  [open connection]

- status = TR_CLOSE (idtrcon)
  [close connection]

- status = TR_SNDND (idtrcon, msg, lnmsg)
  [send normal data]

- status = TR_RCVND (idtrcon, var msg, var lnmsg)
  [receive normal data]

- status = TR_WCONN (loctrad, var remtrad, qos_param, maxtime,
  var msg, var lnmsg, var exp_data, var idtrcon)
  [wait for a connection establishement request]

2. couche_5 <— couche_4 : services

- TR_INF_EVENT (idtrcon, event, info)
  [inform of occurrence of event]

3. couche_5 —> couche_4 : contrôle de flux

- TR_WR_ENABLE (idtrcon)
- TR_WR_DISABLE (idtrcon)
  [enable/disable the transfer from 4 to 5]

4. couche_5 <— couche_4 : contrôle de flux

- TR_RD_ENABLE (idtrcon)
- TR_RD_DISABLE (idtrcon)
  [enable/disable the transfer from 5 to 4]

```
status = TR_CONNECT ( loctrad, remtrad, qos_param, msg, lnmsg
                      var exp_data, var idtrcon )
```

La couche supérieure demande à la couche Transport d'établir une connexion transport entre le site local d'adresse LOCTRAD [local transport address] et le site distant d'adresse REMTRAD [remote transport address].

Elle attend un service dont la qualité est définie dans QOS-PARAM [quality of service parameters] et demande ou non la présence du service de transfert de données expresses EXP-DATA [expedited data service].

> Dans cette maquette QOS-PARAM aura une valeur fixe et EXP-DATA indiquera toujours l'absence de données expressses.

Un message MSG [message] de longueur LNMSG [length of message] est à transmettre lors de l'établissement de la connexion.

Si la connexion est établie, la fonction renvoie dans STATUS la valeur "NoErr", et dans IDTRCON [identifier of transport connection] l'identifiant de la connexion ainsi établie.

EXP-DATA [expedited data service] constitue le point d'accord entre les deux sites en ce qui concerne le service de transfert de données expresses.

Si l'établissement de la connexion échoue, la fonction renvoie dans STATUS une valeur explicative :

a. rejet par le site local

   - ErrFullLoc : no more connections at local site

b. rejet par le site distant

   - ErrReject : request is rejected by the remote site

c. pas de réponse

   - ErrTimeout : no answer during a time interval fixed by the system

d. problème de transmission sur le réseau

- ErrNet : cannot transmit something on the network

e. problème de protocole

- ErrProt : incorrect messages exchange for protocol

f. paramètres de la fonction

- ErrLocAd : invalid local address
- ErrRemAd : invalid remote address
- ErrLnMsg : invalid message length

---

status = TR_CLOSE (idtrcon)

La couche supérieure demande à la couche Transport de fermer la connexion Transport IDTRCON [identifier of transport connection].

Si la fermeture se passe correctement, la fonction renvoie dans STATUS la valeur "NoErr".

Sinon elle renvoie dans STATUS une valeur explicative :

a. rejet par le site local

- ErrNotOpen : the connection is not open

d. problème de transmission sur le réseau

- ErrNet : cannot transmit something on the network

e. problème de protocole

- ErrProt : incorrect messages exchange for protocol

f. paramètres de la fonction

- ErrIdCon : invalid connection identifier

---

status = TR_SNDND (idtrcon, msg, lnmsg)

La couche supérieure demande à la couche Transport de transmettre le message de données normales MSG [message] de longueur LNMSG [length of message] sur la connexion transport IDTRCON [identifier of transport connection].

Si l'envoi se passe correctement, la fonction renvoie dans STATUS la valeur "NoErr".

Sinon elle renvoie dans STATUS une valeur explicative :

a. rejet par le site local

- ErrNotOpen : the connection is not open

d. probleme de transmission sur le réseau

- ErrNet : cannot transmit something on the network

e. problème de protocole

- ErrProt : incorrect messages exchange for protocol

f. paramètres de la fonction

- ErrIdCon : invalid connection identifier
- ErrLnMsg : invalid message length

Remarque.

Le contrôle de flux de l'interface lié à ce service est réalisé via les deux procédures

TR_RD_ENABLE
TR_RD_DISABLE

---

status = TR_RCVND (idtrcon, var msg, var lnmsg)

La couche supérieur demande à la couche Transport de voir si un message de données normales, de longueur maximum LNMSG [length of message], est arrivé sur la connexion IDTRCON [identifier of transport connection].

Si un message est présent, il est fourni dans MSG [message] ainsi que sa longueur dans LNMSG [length of message]. Et la fonction renvoie dans STATUS la valeur "NoErr".

Si aucun message ne peut être fourni, la fonction renvoie dans STATUS une valeur explicative :

a. rejet par le site local

- ErrNotOpen : the connection is not open

c. pas de réponse

- ErrNoMsg : no message is available

e. problème de protocole

- ErrProt : incorrect messages exchange for protocol

f. paramètres de la fonction

- ErrIdCon : invalid connection identifier

---

status = TR_WCONN (loctrad, var remtrad, var qos_param, maxtime,
var msg, var lnmsg, var exp_data, var idtrcon)

La couche supérieure demande à la couche Transport d'attendre et de répondre, pendant un temps maximum MAXTIME [maximum wait time] à une demande d'établissement d'une connexion Transport avec le site local d'adresse LOCTRAD [local transport address] émise par le site distant d'adresse REMTRAD [remote transport address].

Si la connexion est établie, la fonction renvoie dans STATUS la valeur "NoErr", dans IDTRCON [identifier of transport connection] l'identifiant de la connexion ainsi établie et dans REMTRAD [remote transport address] l'adresse du site ayant émis la requête.

EXP-DATA [expedited data service] constitue le point d'accord entre les deux sites en ce qui concerne le service de transfert de données expresses. QOS-PARAM [quality of service parameters] constitue le point d'accord entre les deux sites en ce qui concerne la qualité des services durant la connexion.

> Dans cette maquette, EXP-DATA doit indiquer qu'on utilise pas de données expresses et QOS-PARAM doit être à la valeur fixe de l'implémentation.

MSG [message] contient un message éventuellement reçu lors de l'établissement de la connexion et LNMSG [length of message] sa longueur.

Si l'établissement de la connexion échoue, la fonction renvoie dans STATUS une valeur explicative :

a. rejet par le site local

- ErrFullLoc : no more connections at local site

c. pas de réponse

- ErrTimeout : no request during the time interval fixed by the system

d. problème de transmission sur le réseau

- ErrNet : cannot transmit something on the network

e. problème de protocole

- ErrProt : incorrect messages exchange for protocol

f. paramètres de la fonction

- ErrLocAd : invalid local address
- ErrRemAd : invalid remote address

TR_INF_EVENT (idtrcon, event, info)

La couche Transport informe la couche supérieure que l'événement EVENT [event] est arrivé sur la connexion transport IDTRCON [identifier of transport connection].

INFO [information] contient des informations complémentaires suivant le type d'évènement.

Valeurs possibles de EVENT - contenu de INFO

connect	loctrad, remtrad
close	-
reset	originator
data available	

Remarque.

Le contrôle de flux lié à ce service pour les événements data available est réalisé via les deux procédures

TR_WR_ENABLE
TR_WR_DISABLE

TR_WR_ENABLE (idtrcon)
TR_WR_DISABLE (idtrcon)

La couche supérieure indique à la couche Transport si elle est prête ou non à recevoir des messages sur la connexion transport IDTRCON [identifier of transport connection].

Remarque.

Ces procédures contrôlent le flux de l'interface pour les événements data available du service TR_INF_EVENT.

TR_RD_ENABLE (idtrcon)
TR_RD_DISABLE (idtrcon)

La couche Transport indique à la couche supérieure si elle est prête ou non à recevoir des messages sur la connexion transport IDTRCON [identifier of transport connection].

Remarque.

Ces procédures contrôlent le flux de l'interface pour le service TR_SNDND.

2.4.3. Primitives et événements du module INTERFACE NIVEAU 4

Ce paragraphe décrit les interactions avec les autres modules du processus NIFTP, c'est-à-dire les primitives et les événements. Les primitives permettent au processus d'agir sur le niveau transport tandis que les événements permettent au niveau transport d'agir sur le processus.

Evénements

TS_data Cet événement indique qu'un paquet de données est arrivé du niveau transport. Il a comme données les données reçues.

TS_open Cet événement indique qu'une connexion est ouverte au niveau transport. Il a comme données l'adresse du site distant connecté et le message reçu lors de la connexion.

TS_close Cet événement indique que la connexion est fermée ou peut être considérée comme fermée (en cas d'erreur). Il a comme données le motif de la fermeture.

TS_reset Cet événement indique qu'un reset du niveau transport a eu lieu. Il a comme données le motif du reset.

Primitives

tr_connect(loctrad,remtrad,lmsgin,msgin)

```
ADDRESS loctrad,remtrad;
int     lmsgin;
char    *msgin;

status -> TS_open(remtrad,lmsgout,msgout)
          TS_close(lmotif,motif)
```

Cette primitive a pour effet de demander au niveau transport de créer une connexion entre le site local d'adresse LOCTRAD et le site distant d'adresse REMTRAD en transmettant le message de connexion MSGIN de longueur LMSGIN. Si la connexion a pu être établie, la primitive répond en créant un événement TS_open ayant comme donnée le message MSGOUT reçu lors de l'établissement de la connexion et de longueur LMSGOUT, ainsi que l'adresse REMTRAD du site distant connecté. En cas d'échec, la primitive répond en créant un événement TS_close avec comme donnée le motif MOTIF de l'échec et de longueur LMOTIF.

tr_close( )

```
status -> TS_close(lmotif,motif)
```

Cette primitive a pour effet de demander au niveau transport de fermer la connexion. Elle répond toujours par un événement TS_close ayant comme donnée, en cas d'échec, le motif de cet échec. En cas d'échec de la fermeture la connexion n'est plus utilisable et est donc considérée comme fermée.

tr_sndnd(msg,lmsg)

```
char    *msg;
int     lmsg;

status -> -
          TS_close(lmotif,motif)
```

Cette primitive permet de demander au niveau transport de transmettre le message MSG de longueur LMSG. En cas d'échec, la primitive répond par un événement TS_close ayant pour donnée le motif MOTIF de l'échec de longueur LMOTIF. Si la primitive se déroule correctement elle ne renvoie aucun événement, c'est le processus NIFTP distant qui se charge de répondre.

tr_wconn(loctrad)

```
ADDRESS loctrad;

status -> TS_open(remtrad,lmsgout,msgout)
          TS_close(lmotif,motif)
```

Cette primitive a pour effet de demander au niveau transport d'attendre une demande de connexion avec le site local d'adresse LOCTRAD. Si une connexion a pu être établie, la primitive répond en créant un événement TS_open ayant comme donnée le message MSGOUT reçu lors de l'établissement de la connexion et de longueur LMSGOUT ainsi que l'adresse REMTRAD du site distant connecté. En cas d'échec, la primitive répond en créant un événement TS_close avec comme donnée le motif MOTIF de l'échec et de longueur LMOTIF.

2.4.4. Structure des pipes

L'interface est matérialisée par deux pipes. Dans chacun de ces pipes peuvent circuler des événements qui sont soit des fonctions interfaces, soit des status qui annoncent la terminaison de l'exécution d'une fonction interface.

- pipe 5 —> 4 : fonctions interfaces de 5 vers 4
- pipe 4 —> 5 : status (des fonctions interfaces de 5 vers 4)
  fonctions interfaces de 4 vers 5

Nous allons donc définir pour chaque pipe une structure de données telle qu'elle contienne toutes les informations nécessaires à chaque type d'événement qui peut passer dans ce pipe. Ensuite, il faudra définir pour chaque fonction interface ainsi que pour le status de chaque fonction interface, quels sont les champs de la structure qu'il faut garnir et avec quelles valeurs il faut les garnir.

Le premier élément de chaque structure est un identificateur d'événement. La valeur de ce champ permettra d'identifier le type d'événement dont il s'agit.

Puisqu'on veut avoir pour un pipe une structure qui soit toujours la même (et donc de longueur fixe), on doit utiliser un pointeur vers les zones de longueur variable (par exemple : le message dans le TR_SNDND). Or, Unix ne permet pas que deux processus, même apparentés, utilisent une zone de données commune. Donc tout doit passer par le pipe. C'est pourquoi, lorsqu'il y a un message à transmettre avec l'événement, ce message sera écrit dans le pipe juste après la structure décrivant l'événement. De cette façon, celui qui reçoit l'événement doit procéder de la façon suivante :

- lire dans le pipe la structure (dont il connait la description),
- consulter dans la structure le champ qui donne la longueur de ce message,
- si la longueur n'est pas nulle, lire dans le pipe un message de cette longueur.

Pour terminer nous définissons les primitives que le module INTERFACE offre aux modules de niveau supérieur.

2.4.4.1. Définition des messages voyageant dans les pipes

Ces messages ont la même forme que ceux définis pour le niveau 3 par le labo de 2e licence [LABO], afin de permettre une intégration ultérieure des deux maquettes.

pipe 5 —> 4

```
struct elt_p54 {
     int    evt_p54;     identifier of event on pipe 54
     long   clk_p54;     birth time of event       (not used)
     int    idc_p54;     identifier of transport connection
     long   loc_p54;     local transport address
     long   rem_p54;     remote transport address
     char  *msg_p54;     pointer to message        (not used)
     int    lms_p54;     length of message
     long   mti_p54;     maximum time of wait      (infinite)
     int    mxl_p54;     maximum length of receive (infinite)
               };
```

evt_p54 : valeurs possibles et leur signification

01	TR_CONNECT	[open connection]
02	TR_CLOSE	[close connection]
03	TR_SNDND	[send normal data]
04	TR_RCVND	[receive normal data]
06	TR_WCONN	[wait for a connection establishment request]

pipe 4 --> 5

```
struct elt_p45 {
     int    evt_p45;     identifier of event on pipe 45
     int    idc_p45;     identifier of transport connection
     long   loc_p45;     local transport address
     long   rem_p45;     remote transport address
     char  *msg_p45;     pointer to message     (not used)
     long   lms_p45;     length of message
     int    eve_p45;     event of nw_inf_event
               };
```

evt_p45 : valeurs possibles et leur signification

fonctions interfaces:

12	TR_INF_EVENT	[inform of occurrence of event]
17	TR_RD_ENABLE	[enable transfert from 5 to 4]
18	TR_RD_DISABLE	[disable transfert from 5 to 4]

status:

40	NoErr	[no error]
41	ErrFullLoc	[no more connections at local site]
42	ErrDisable	[the connection is not open]
43	ErrReject	[request is rejected by the remote site]
44	ErrTimeout	[no answer during a time interval]
45	ErrNet	[cannot transmit something on the network]
46	ErrProt	[incorrect messages exchange for protocol]
47	ErrNotOpen	[the connection is not open]
48	ErrNoMsg	[no message is available]

status de syntaxe:

50	ErrLocAd	[invalid local address]
51	ErrRemAd	[invalid remote address]
52	ErrLnMsg	[invalid message length]
53	ErrIdCon	[invalid connection identifier]

eve_p45 : valeurs possibles et leur signification

01	connect
02	close
03	reset
04	data available

2.4.4.2. Définition des fonctions interfaces par rapports aux pipes

fonction interface		status correspondant
nom	structure et champs utilisés	structure et champs utilisés
TR_CONNECT	elt_p54 evt_p54 = 01 clk_p44 loc_p54 rem_p54 msg_p54 lms_p54	elt_p45 evt_p45 = 40,41,43, 44,45,46, 50,51,52 idc_p45 msg_p45 lnmsg_p45
TR_CLOSE	elt_p54 evt_p54 = 02 clk_p54 idc_p54	elt_p45 evt_p45 = 40,45,46, 47,53 idc_p45
TR_SEND	elt_p54 evt_p54 = 03 clk_p54 idc_p54 msg_p54 lms_p54	elt_p45 evt_p45 = 40,42,45, 46,47,52 53 idc_p45
TR_RCVND	elt_p54 evt_p54 = 04 clk_p54 idc_p54 mxl_p54 (infini)	elt_p45 evt_p45 = 40,42,46, 47,48,53 idc_p45 msg_p45 lms_p45
TR_WCONN	elt_p54 evt_p54 = 06 clk_p54 loc_p54 mti_p54 (infini)	elt_p45 evt_p45 = 40,41,44, 45,46,50, 51 idc_p45 rem_p45 msg_p45 lms_p45

fonction interface		status correspondant
nom	structure et champs utilisés	structure et champs utilisés
TR_INF _EVENT	elt_p45 evt_p45 = 12 idc_p45 loc_p45 rem_p45 eve_p45	
TR_RD_ENABLE	elt_p45 evt_p45 = 17 idc_p45	
TR_RD _DISABLE	elt_p45 evt_p45 = 18 idc_p45	

2.4.4.3. Primitives d'accès au pipe

rd_pipe(buf,msg,lmsg)

```
int     lmsg;
char    *buf, *msg;
```

Cette primitive lit dans le pipe un buffer BUF contenant un elt_p45 et lit également un message MSG dont la longueur est LMSG.

wr_pipe(buf,msg,lmsg)

```
int     lmsg;
char    *buf, *msg;
```

Cette primitive écrit dans le pipe un buffer BUF et écrit également un message MSG dont la longueur est LMSG. Après écriture dans le pipe, la primitive ne libère ni le buffer, ni la suite de blocs mémoire pointée par MSG.

2.4.5. Gestion du dialogue avec le niveau 4

Pour chaque envoi d'une fonction d'interface dans le pipe de 5 vers 4, un status doit revenir dans le pipe de 4 vers 5. Comme on rend tout de suite la main au module appelant pour lui eviter de perdre son temps en attente, il est possible que d'autres demandes d'envoi arrivent avant le retour du status. Si on statisfait directement ces envois (on dira qu'on fait du multitraitement), il ne sera plus possible de faire le lien entre les envois et les status. Pour eviter cela on ne fera pas de nouvel envoi tant que le status correspondant au premier n'est pas revenu c'est-à-dire qu'il faudra gérer une file des demandes d'envoi. D'autre part, il faut tenir compte du fait que le contrôle de flux peut empêcher les envois sur le pipe de 5 vers 4. Cela nous amène à définir une file des éléments elt_p54 en attente d'envoi, l'elt_p54 en attente de réponse et deux primitives de gestion de la file.

```
snd_pipe(buf,lmsg,msg)
---------
    int     lmsg;
    char    *buf, *msg;
```

Cette primitive est appelée à chaque demande d'envoi dans le pipe. Elle a pour effet de mettre la demande dans la file puis s'il n'y a pas d'envoi en attente de réponse et si le pipe n'est pas bloqué, elle envoie la demande dans le pipe et indique qu'un envoi est en attente de réponse.

```
rcv_pipe(buf,lmsg,msg)
---------
    int     lmsg;
    char    *buf, *msg;
```

Cette primitive est appelée chaque fois que le niveau transport signale qu'il a écrit quelque chose dans le pipe de 4 vers 5. Elle a pour effet de lire un elt_p45 du pipe ainsi que le message associé.

S'il s'agit d'un TR_RD_ENABLE ou d'un TR_RD_DISABLE, elle indique l'état bloqué ou débloqué du pipe de 5 vers 4.

S'il s'agit d'un TR_INF_EVENT indiquant open, close ou reset, elle génère l'événement correspondant.

S'il s'agit d'un TR_INF_EVENT indiquant data available, elle effectue une demande d'envoi de TR_RCVND.

S'il s'agit d'un status indiquant NoErr, elle génère l'événement correspondant à l'elt_p54 en attente de réponse. Puis détruit cet elt_p54 et envoie l'elt_p54 suivant si la file n'est pas vide, ou bien indique qu'il n'y a plus d'envoi en attente si la file est vide.

S'il s'agit d'un status indiquant une erreur, elle génère un événement TS_close. Puis met à jour la file comme dans le cas précédent.

2.5. Le module FORMATEUR DE RECORD

2.5.1. Description

Le but du module FORMATEUR DE RECORD est d'assurer les services de la couche session d'une part en assemblant les subrecords de données et d'autre part en insérant des Commandes de Contrôle de Transfert, CCT en abrégé.

Le contrôle de flux est géré en indiquant le point de départ du transfert (par une CCT SS) et les points de synchronisation, encore appelés 'mark' en anglais (par des CCT MS), tandis que le receveur (R) des données envoie des signaux d'acceptation des points de synchronisation (mark acknowledge) (par des CCT MR).

En cas de détection d'une erreur de données ou d'un reset du niveau transport, le receveur (R) envoie un signal de réinitialisation (par une CCT RR) et l'envoyeur (S) répond en indiquant un nouveau point de départ (par une CCT SS).

Les CCT permettent également de gérer la fin du transfert en cas de déroulement correct (par la séquence QR(ok), ES(ok) et ER(ok)), en cas d'échec avec fin négociée (par la séquence QR(e), ES(e) et ER(e)) ou en cas d'échec avec fin brutale (par les CCT ES(a) et ER(a)). Il y a également moyen de gérer les interruptions de transfert (par la séquence QR(h), ES(h) et ER(h)).

Pour terminer une CCT (la CCT CS) permet au niveau présentation de sélectionner le code de transfert.

Pour être complet, il faut encore signaler que la gestion de la fenêtre se fait dans le module TRANSFERT (cfr 2.8.3.) lors de l'accès au fichier et non à ce niveau-ci.

2.5.2. Mécanismes de gestion par les Commandes de Contrôle de Transfert

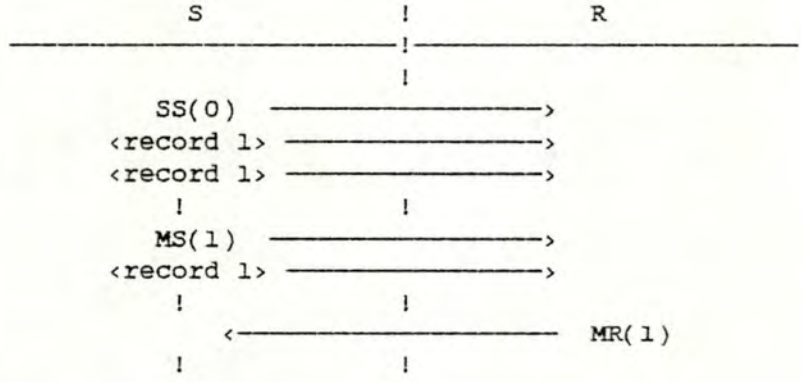
2.5.2.1. Points de synchronisation

La gestion des points de synchronisation se fait par les CCT suivantes:

SS(num)	Start of Data
MS(num)	Mark Point
MR(num)	Mark Acknowledge

où num est le numéro du point de synchronisation.

L'échange commence par l'envoi d'une CCT SS avec normalement 'num' égal à zéro (sauf dans le cas d'une reprise d'un transfert antérieur). Ensuite l'envoyeur envoie les records de données entremêlées avec des CCT MS marquant les points de synchronisation et ayant des valeurs de 'num' croissantes. Et au fur et à mesure de la réception des points de synchronisation le receveur renvoie des signaux d'acceptation de points de synchronisations par des CCT MR.



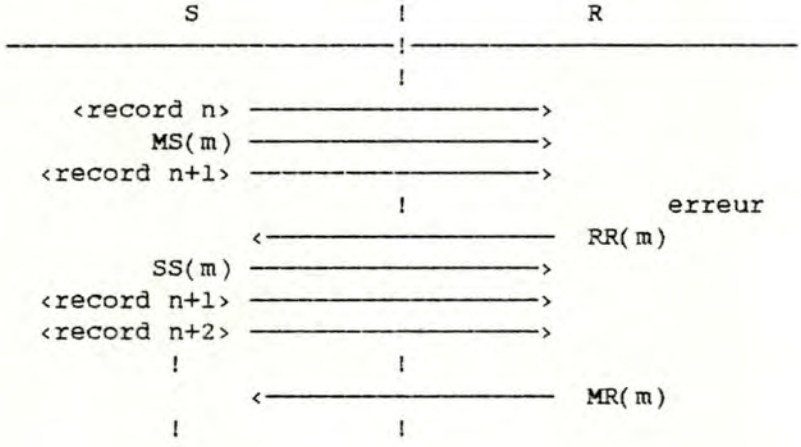
2.5.2.2. Réinitialisation

La demande de réinitialisation se fait par la CCT suivante:

RR(num)	Restart Request

où num est le numéro du point de synchronisation à partir duquel il faut recommencer.

Lorsque le receveur détecte une erreur de données ou un reset du niveau transport, il envoie une CCT RR indiquant le dernier point de synchronisation reçu avant l'erreur ou le reset. Puis l'envoyeur envoie une CCT SS avec ce même numéro et le transfert reprend.



2.5.2.3. Fin de transfert et interruption

La gestion de la fin du transfert et des interruptions se fait par les trois CCT suivantes:

QR(param)	Quit
ES(param)	End of Data
ER(param)	End Acknowledge

où param prend une des valeurs suivantes:

ok	[00]	en cas de terminaison normale
h	[10]	en cas d'interruption du transfert
e	[2x]	en cas d'erreur
a	[3x]	pour une terminaison immédiate après un timeout

ces paramètres sont encore précisés par

ES(e), ER(e) et QR(e)

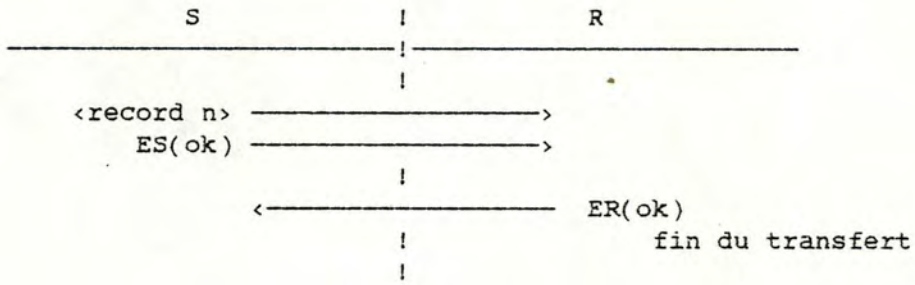
[20]	erreur du receveur avec récupération possible
[21]	erreur du receveur sans récupération possible
[22]	erreur de protocole détectée par le receveur

ES(e) et ER(e)

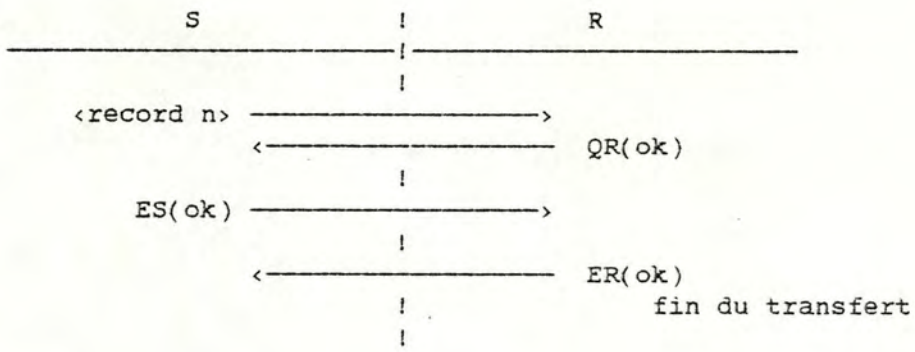
[28]	erreur de l'envoyeur avec récupération possible
[29]	erreur de l'envoyeur sans récupération possible
[2A]	erreur de protocole détectée par l'envoyeur

```
ES(a)
    [30]    En attendant un MR
    [31]    En attendant un RR après un reset
    [32]    En attendant un ER(ok), après l'envoi d'un ES(ok)
    [33]    En attendant un ER(e), après l'envoi d'un ES(e)
    [34]    En attendant un ER(h), après l'envoi d'un ES(h)

QR(a)
    [30]    En attendant des données
    [31]    En attendant un SS, après  l'envoi d'un RR
    [32]    En attendant un ES(ok), après l'envoi d'un QR(ok)
    [33]    En attendant un ES(e), après l'envoi d'un QR(e)
    [34]    En attendant un ES(h), après l'envoi d'un QR(h)
    [35]    En attendant le premier SS, après le GO
    [36]    En attendant un ES(h) et un SS,
                        après l'envoi d'un QR(h) et d'un RR

ER(a) n'existe pas.
```

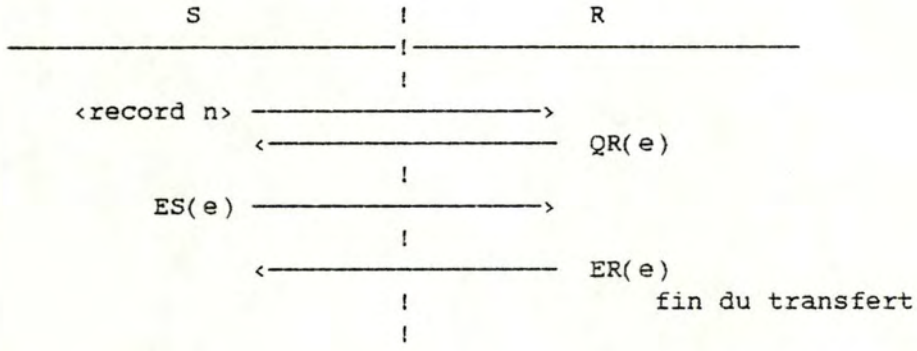
Dans le cas où l'envoyeur détecte la fin de fichier, il envoie ES(ok) et le receveur répond par ER(ok).



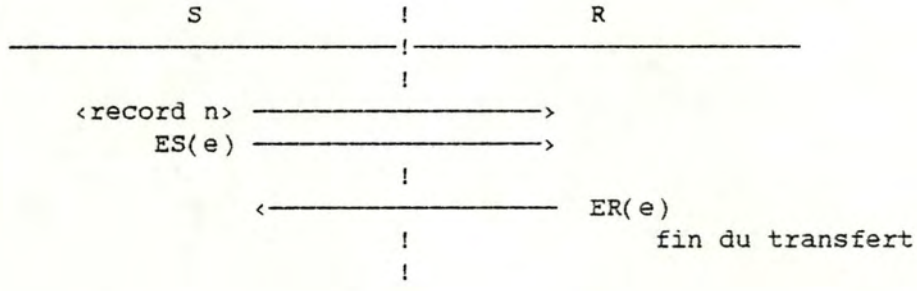
Dans le cas où le receveur détecte la fin de fichier, il envoie un QR(ok), l'envoyeur répond par ES(ok) et le transfert se termine par l'envoi par le receveur du ER(ok).



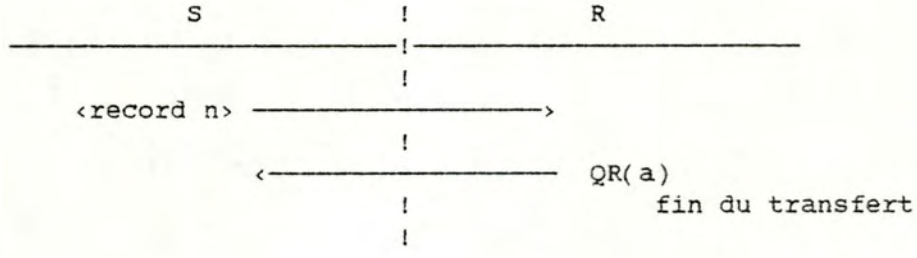
Dans le cas où une erreur est détectée par le receveur, il envoie un QR(e), l'envoyeur répond par un ES(e) et le transfert se termine par l'envoi par le receveur du ER(e).



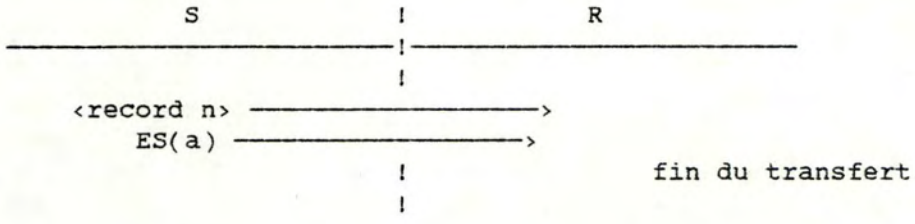
Dans le cas où une erreur est détectée par l'envoyeur, il envoie un ES(e) et le receveur répond par un ER(e).



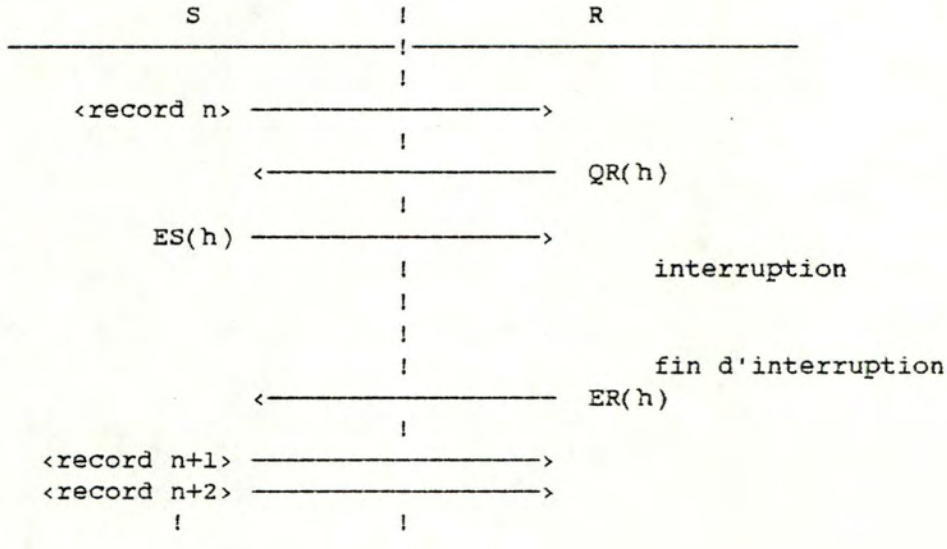
Dans le cas où un timeout à lieu chez le receveur, il envoie un QR(a) et le transfert est terminé.



Dans le cas où un timeout à lieu chez l'envoyeur, il envoie un ES(a) et le transfert est terminé.



Si le receveur veut interrompre le transfert, il envoie un QR(h) et l'envoyeur accepte l'interruption en envoyant un ES(h). L'interruption se termine lorsque le receveur envoie un ER(h).

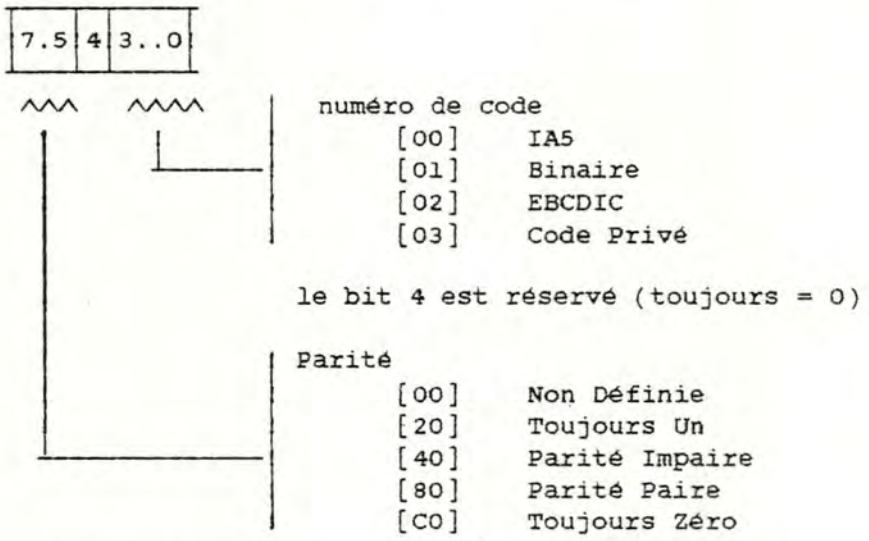


2.5.2.4. Sélection du code

La dernière CCT permet au module PRESENTATION de sélectionner le code de transfert.

CS(code) Code Select

où code a la signification suivante:



La CCT CS est insérée entre les subrecords pour indiquer un changement de code. Juste après le SS le code sélectionné est IA5 avec la parité Non Définie. Si le code de transfert est différent de IA5, un CS doit suivre le SS. Le positionnement du CS et le choix du code est limité par l'attribut Data Type Attribute (cfr le module NEGOCIATEUR en 2.7.3.2.).

2.5.2.5. Précédence entre les Commandes de Contrôle de Transfert

Beaucoup de CCT apparaissent comme faisant parties de courtes séquences où la réception d'une CCT provoque la transmission d'une autre. Dans un tel cas, la première CCT est dite pendante jusqu'à ce que la CCT suivante est envoyée où reçue. Les CCT ont une précédence (souvent en fonction de leur argument), qui détermine le résultat lorsque d'autres CCT sont reçues tandis qu'une CCT est pendante. La nouvelle CCT peut soit rendre la CCT précédente inutile, soit être ignorée parce que moins importante. En particulier parmis les CCT ES, QR et ER, le groupe OK est surpassé par le groupe H, H par E et E par A. La forme A termine unilatéralement le transfert et ne peux donc être

pendante. Une CCT est dite correspondre avec sa suivante si elles sont du même groupe.

Dans tous les cas, une CCT ES surpasse toute celles pendantes de précédence plus petite (plus faible) ou tout QR plus faible. Le receveur doit ignorer un ES s'il y a un QR pendant de plus haute précédence (plus fort). Une CCT ES est pendante jusqu'à la réception d'une CCT ER correspondante, ou jusqu'à ce qu'elle soit surpassée par un QR plus fort ou détruite par un reset. Un ES(ok) pendant est détruit par une CCT RR. Un ES(ok) qui a été détruit par un RR ou un QR(h) doit être retransmis si le transfert doit se terminer avec succès.

Une CCT RR peut être détruite par un reset du service transport, tout QR (sauf le QR(h)), un ES(e) ou un ES(a).

Un QR surpasse un QR plus faible, et détruit tout ES plus faible. Il reste pendant jusqu'à la réception d'un ES correspondant, ou jusqu'à ce qu'il soit détruit par un ES plus fort ou un reset. Un QR reçu tandis qu'un ES correspondant ou plus fort est pendant doit être ignoré. Un QR(h) pendant est surpassé par un ER(h). Un QR(h) peut être utilisé pour rétablir un état interrompu après un reset du service transport. Un ES(h) en réponse est toujours attendu dans ce cas.

Un ES pendant est satisfait par la réception d'un ER correspondant. Un ER(h) surpasse un QR(h). Une CCT ER termine une séquence donc elle ne peut être pendante. Un receveur peut envoyer un RR avant un ER(h) s'il veut qu'une partie du transfert soit répété après que l'interruption soit terminée. Un ER(ok) ou ER(e) valides terminent le transfert et donc surpassent toute CCT pendante.

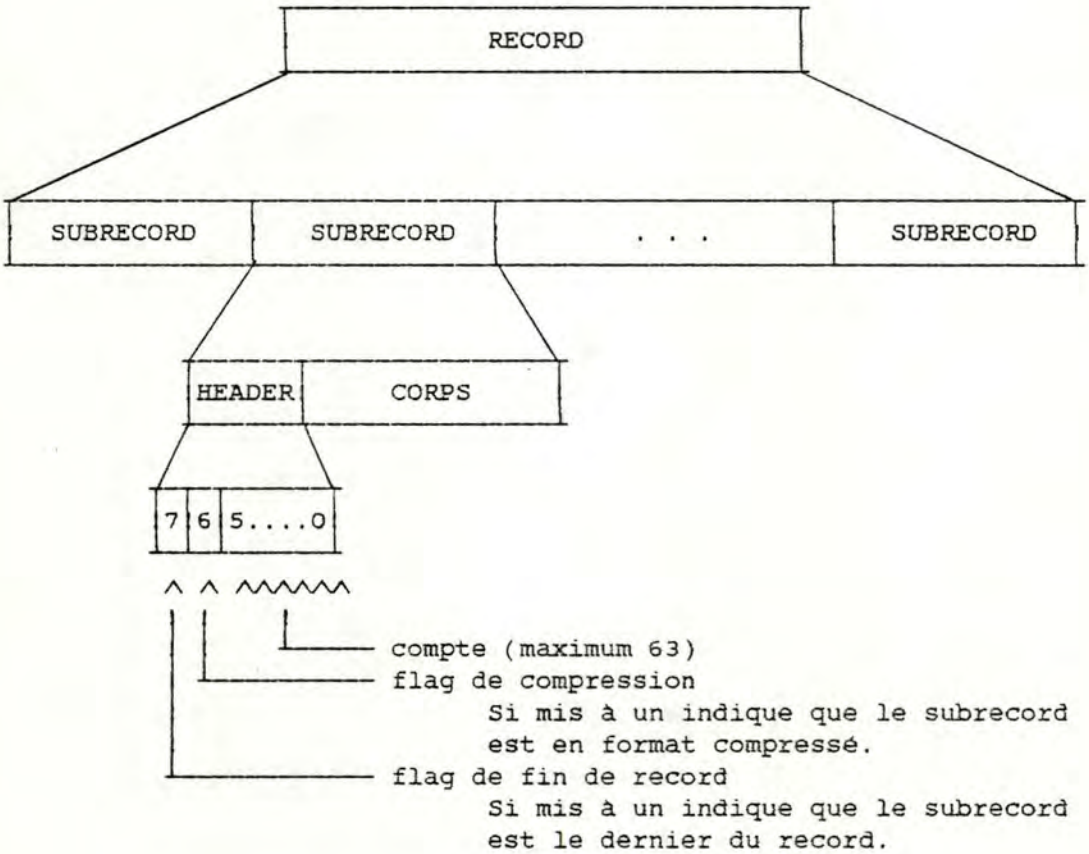
Remarque

> Les tables d'états du transfert c'est-à-dire pour les modes S et R (cfr 2.3) sont en grande partie construites à partir de ces règles de précédence. Les états sont déterminés à partir des CCT pendantes, en y ajoutant l'état initial GO et l'état DATA où aucune CCT n'est pendante. Les transitions sont fonction des précédences. Un bon exercice de compréhension est de reconstruire les tables à partir de ces règles en sachant qu'il faut prévoir un état où à la fois un ES(h) (ou QR(h)) et un RR sont pendants, de même qu'un état où on est en fin de fichier et qu'un ES(h) est pendant.

2.5.3. Formats

2.5.3.1. Structure de record NIFTP

Les commandes d'initialisation et de terminaison ainsi que les données sont transmises dans NIFTP comme des records de longueur arbitraire; pour la transmission un record est décomposé en subrecords, chacun d'eux est précédé d'un octet d'entête appelé header octet. Les Commandes de Contrôle de Transfert peuvent être insérées aux limites des subrecords. Cette structure de record permet à la fois des records de longueur arbitraire et la compression des données. La structure du record est la suivante:



L'interprétation du champ 'compte' et du flag de compression dépend du type de donnée dans le subrecord. Une entête de record avec seulement le flag de compression ne doit jamais être uitlisée tandis que l'entête d'un record de longueur zéro aura toujours le bit de fin de record à un, ainsi l'octet sera non nul. Une entête de subrecord avec un compte à zéro et le bit 7 à zéro est expressément réservée. Cela permet d'utiliser l'octet zéro comme entête spéciale pour introduire des CCT, qui sont toujours de longueur fixe (2 octets) et non compressés.

Selon la description du protocole, ce mécanisme doit permettre, en principe, de reconnaître les Commandes de Contrôle de Transfert à tout moment. Cependant comme rien n'empêche d'avoir des octets à zéro dans le corps des subrecords, il ne suffit pas de repérer les octets nuls dans le flot d'octet; il faut donc repérer chaque subrecord. Ici intervient un deuxième problème: dans le cas des fichiers textes en format non compressé, il suffit de regarder le champ 'compte' pour savoir la longueur en octets du subrecord et donc faire la décomposition en subrecords. Mais dans le cas du format compressé et, ce qui est encore plus compliqué, dans le cas de données binaires dont la longueur de mot n'est pas un octet, il faut refaire une partie du travail du module PRESENTATION pour décomposer en subrecords.

Dans les phases d'initialisation et de terminaison, toute l'information est transférée sous forme de commandes (cfr module NEGOCIATEUR 2.7.1.). Ces commandes sont encodées de la même manière que les records de données textes, chaque commande occupant exactement un record. Les commandes ne peuvent pas être compressées, même si la compression a étée convenue entre les deux parties, mais elles peuvent être décomposées en subrecords si nécessaires. Les commandes sont toujours écrites en code IA5.

2.5.3.2. Corps des Subrecords de Données

Subrecords de données textes

Un subrecord texte contient des caractères de texte. La signification du champ 'compte' dépend de la position du flag de compression. Si ce flag n'est pas à un alors le 'compte' donne le nombre de caractères qui forment le subrecord. La taille maximum d'un subrecord est 63 caractères. Le compte n'inclut pas le header octet. Si le flag de compression est à un, alors le compte donne le nombre de fois que le caractère unique suivant l'entête doit être inséré en reconstruisant le subrecord compressé. La correspondance des caractères avec les octets dépend du code employé. Pour les codes standards, IA5 et EBCDIC, chaque caractère occupe un octet. Pour les codes privés, la correspondance est définie par un accord privé.

Texte non compressé

caractère 1	caractère 2	. . .	caractère n

où n est la valeur du champ 'compte'

Texte compressé

caractère

Subrecords de données binaires

Un subrecord binaire contient des mots binaires, chacun d'eux est un string de bits de longueur définie par la valeur de l'attribut Binary Word Size (cfr module PRESENTATION en 2.6.2.). La signification du champ 'compte' dépend du positionnement du flag de compression. Si ce flag est à zéro, le compte donne le nombre de mots binaires contenus dans le subrecord. La correspondance des mots avec les octets pour la transmission est définie par l'attribut Binary Format (cfr module PRESENTATION en 2.6.2.). La taille d'un subrecord, en octets, est le nombre minimum d'octets complets nécessaires pour représenter les mots binaires transmis avec l'algorithme de correspondance choisi. Si l'attribut Binary Word Size est à 8, les mots correspondent directement aux octets. Un subrecord binaire peut au maximum contenir 63 mots. Si le flag de compression est à un, le compte donne le nombre de fois que l'unique mot binaire suivant l'entête doit être inséré en reconstruisant le subrecord compressé. Le nombre d'octets dans le subrecord est le nombre minimum nécessaire pour un seul mot binaire avec l'agorithme de correspondance choisi. Le padding (remplissage) est fait avec des bits à zéro jusqu'au prochain octet entier.

Binaire aligné

mot 1 (+pad)	mot 2 (+pad)	. . .	mot n (+pad)

où n est la valeur du champ 'compte'
(+pad) indique un padding

Binaire compacté

mot 1	mot 2	. . .	mot n-1	mot n (+pad)

où n est la valeur du champ 'compte'
(+pad) indique un padding

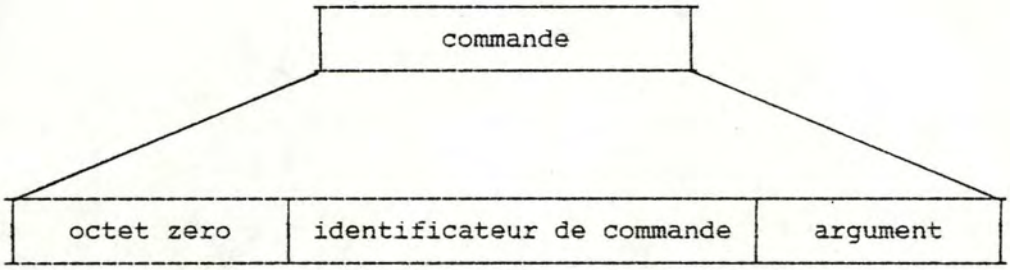
Binaire compressé

mot (+pad)

où (+pad) indique un padding

2.5.3.3. Commandes de Contrôle de Transfert

Les Commande de Contrôle de transfert sont représentées par deux octets précédés d'un header octet nul. L'identificateur de commande et l'argument sont chacun d'un octet. Ces commandes sont insérées entre les subrecords durant la phase de transfert. Cela permet de placer les marques arbitrairement dans le fichier et ainsi de transférer des fichiers sans structure de record en évitant d'imposer des limites de record arbitraires.



2.5.4. Primitives d'assemblage

L'assemblage des subrecords se fait dans un buffer de taille fixe déterminée par la taille maximum des messages que l'on peut envoyer sur le niveau transport. Cette taille doit être plus grande que la longueur du plus grand subrecord possible c'est-à-dire 63 caractères ou 63 mots binaires plus un octet pour l'entête. Chaque fois qu'un subrecord ou une CCT ferait déborder le buffer, on envoie le contenu du buffer puis on introduit le subrecord ou la CCT à partir du début du buffer. On pourrait envoyer séparément chaque subrecord et chaque CCT mais il n'est pas toujours très efficace de transmettre des petits paquets. Le vidage du buffer pourra également être forcé par une primitive snd_record ou bien après certaines CCT. De plus une primitive end_record permetrra d'indiquer la fin du record. Il faudra donc retenir la position du header octet du dernier subrecord pour pouvoir y ajouter le flag de fin de record. Si le record est terminé par une CCT ou s'il est vide, on y ajoutera un subrecord de longueur zéro dont le flag de fin de record sera à un. Les primitives sont les suivantes:

subrecord(lsrec,srec)

```
int     lsrec;
char    *srec;
```

Cette primitive a pour effet d'insérer le subrecord SREC de longueur LSREC dans le buffer. S'il reste moins de LSREC octets libres dans le buffer, il est vidé avant l'insertion.

end_record( )

Cette primitive permet de signaler au formateur de record que la fin de record est atteinte. Elle a pour effet de mettre à un le flag de fin de record du header octet du dernier subrecord, ou d'ajouter un subrecord de longueur zéro ayant le flag de fin de record positionné à un.

snd_record( )

Cette primitive demande d'envoyer le contenu du buffer à travers le niveau transport et d'assurer que tout ce qui peut être envoyé arrivera de l'autre côté. Cette action est encore appelée dans le protocole "faire un push" . Elle doit être efectuée après la fin des records de commande en phases d'initialisation et de terminaison (cfr module NEGOCIATEUR en 2.7.1.), ainsi qu'après la CCT MS lorsque la limite de la fenêtre est atteinte (cfr module TRANSFERT en 2.8.3.)

```
SS(num)
MS(num)
CS(code)
ES(type)
MR(num)
RR(num)
ER(type)
QR(type)
--------
    short   num;    numéro de point de synchronisation
    short   code;   code de transfert
    tc_type type;   type de commande

    typedef tc_type short
    £define OK      00
    £define H       10
    £define E       20
    £define A       30
```

Ces primitives permettent d'insérer les CCT correspondantes dans le buffer. Les commandes ES, MR, RR, QR et ER demandent de faire un push après leur insertion. De même la commande MS demande un push si la limite de la fenêtre est atteinte (cfr module TRANSFERT en 2.8.3.) mais le push pour cette dernière est demandé par le module TRANSFERT.

2.5.5. Primitives de désassemblage

Lorsqu'un événement TS_DATA est passé au module FORMATEUR DE RECORD, la structure des données est examinée pour en retirer la découpe en subrecords et les CCT. Chaque subrecord donne lieu à un evénement SS_DATA, tandis que pour chaque CCT est généré un événement correspondant. Les événements sont produits dans l'ordre où sont rencontrées les données qui les ont provoqués. Le module PRESENTATION recevra ainsi les subrecords dans l'ordre et pourra donc détecter la fin de record lorsqu'il examinera le header octet du dernier subrecord.

```
desassemble(lmsg,msg)
---------
    int     lmsg;
    char    *msg;
```

Cette primitive assure le désassemblage du message MSG de longueur LMSG.

2.6. Le module PRESENTATION

2.6.1. Description

Le but de ce module est d'assurer les services de la couche présentation comme son nom l'indique. J'ai décrit au chapitre 2 du mémoire les différentes fonctions du service présentation (cfr chap 2 par 6.2). Parmis ces fonctions, le protocole NIFTP décrit le choix de code de transfert, avec possibilité de code privé, le transfert binaire avec compactage et choix de l'ordre des octets, la compression en mode texte et en mode binaire, le formatage de texte et la possibilité de grouper et de scinder les records.

2.6.2. Options et Attributs

Le module offre essentiellement trois primitives: 'encode' pour transformer un record du format interne au format de transfert, 'decode' pour faire la transformation inverse et 'setmode' pour sélectionner les différentes options possibles. C'est sur cette dernière que je m'attarderai le plus pour décrire les différents modes possibles et ce qu'ils signifient.

encode(lrec,rec)

```
int     lrec;
char    *rec;
```

Cette primitive a pour effet de transformer le record REC de longueur LREC du format interne au format de transfert. Elle produit des subrecords qu'elle passe au module FORMATEUR DE RECORD.

decode(lsrec,srec)

```
int     lsrec;
char    *srec;
```

Cette primitive est appelée à la réception d'un événement SS_data produit par le module FORMATEUR DE RECORD. Elle transforme les subrecords en format interne et les rassemble pour reformer le record. A la fin de chaque record elle produit un événement Data ayant pour donnée le contenu du record. Si une erreur est détectée elle produit l'événement Data_NOK.

set_code(mode,val)

```
P_MODE  mode;
int     val;

typedef short P_MODE;
```

Cette primitive a pour effet d'assigner la valeur VAL au mode MODE. MODE peut prendre une des valeurs suivantes:

TRAN_CODE	choix du code de transfert
TXT_COMP	compression de texte
BIN_COMP	compression binaire
BIN_CONC	compactage binaire
FORMAT	préservation du formatage
DELIM_PRES	préservation des délimiteurs et de la position des tabs
PRIV_CODE	nom du code privé
STOR_CODE	code de stockage en Q
PRIV_S_CODE	nom du code privé de stockage
BIN_W_SIZE	taille du mot binaire
PAGE_WIDTH	largeur de page
PAGE_LENGTH	longueur de page
TAB	positions des tabulations

remarque: val dans le cas des modes PRIV_CODE, PRIV_S_CODE et TAB est un pointeur vers une chaîne de caractères.

Le mode TRAN-CODE permet le choix du code de transfert, il peut prendre les valeurs IA5 (00), BINARY (01), EBCDIC (02) ou PRIVATE (03) (note 1). Ces valeurs sont déterminées par l'attribut Text Transfer Code [02] (Note 2). La possibilité de choisir les modes texte ou binaire et de les mélanger, est fonction de l'attribut Data Type [20]. Dans le cas du choix BINARY, le mode binaire est sélectionné et la longueur du mot

---

1. Les chiffres indiqués entre ( ), sont les valeurs correspondantes à donner au paramètre val. Elles sont exprimées en hexadécimal.
2. Les attributs sont définis et négociés par le module NEGOCIATEUR pendant la phase d'initialisation (cfr module NEGOCIATEUR 2.7.3.). Ces attributs ne s'appliquent que pendant la phase de transfert, l'initialisation et la terminaison se faisant toujours en mode texte de code IA5 non compressé et non formaté. Une description détaillée de chaque attribut se trouve dans le texte de la norme NIFTP à la section 7 [NIFTP]. Les chiffres indiqués entre [ ] sont les codes des attributs; ils facilitent la référence aux attributs.

binaire est sélectionnée par le mode BIN-W-SIZE qui est déterminé par l'attribut Binary Word Size [24]. Dans le cas du choix PRIVATE, le code est déterminé par la valeur du mode PRIV-CODE qui est lui-même déterminé par l'attribut Private Transfert Code Name [09]. Les modes STOR-CODE et PRIV-S-CODE permettent de déterminer les transformations de code nécessaires, STOR_CODE prenant les mêmes valeurs que TRAN_CODE et PRIV_S_CODE étant le nom du code privé si STOR_CODE a la valeur PRIVATE. STOR_CODE est déterminé par l'attribut Text Storage Code [22] et PRIV_S_CODE par l'attribut Private Storage Code Name [29]. En outre on peut ajouter à la valeur de TRAN_CODE un choix de parité en lui additionant une des valeurs suivantes: P_ANY (00), P_ONE (20), P_ODD (40), P_EVEN (80) ou P_ZERO (C0). La valeur de TRAN_CODE additionnée de la valeur de parité sert à déterminer l'argument de la CCT CS. Sa valeur par défaut est (00) c'est-à-dire IA5 avec P_ANY.

Les modes TXT-COMP et BIN-COMP sélectionnent respectivement la compression de texte et la compression binaire. Ils peuvent prendre les valeurs binaires: M_UNKNOW (0) si la compression n'est pas implémentée, ou bien NO (1) ou YES (2). Le choix de la compression est limité par l'attribut Facilities [0E].

En mode binaire, le mode BIN-CONC sélectionne le compactage et l'ordre des octets. Il prend pour valeur une des deux valeurs NOGAPS (0001), pour le binaire compacté, ou WORD_BOUNT (0002), pour le binaire aligné, additionée à une des deux valeurs MOST_FIRST (4000), pour faire précéder l'octet de poids le plus fort, ou LEAST_FIRST (8000), pour faire précéder l'octet de poids le plus faible. Ces valeurs sont déterminées par l'attribut Binary Format [04]. La valeur par défaut est (8002) c'est-à-dire le mode binaire aligné avec l'octet de poids faible en tête.

Le mode FORMAT permet en mode texte de sélectionner l'effet du formatage. Le mécanisme du formatage et les actions à prendre sont définis dans le texte de la norme NIFTP à la section 3.2 [NIFTP]. Le mode FORMAT peut prendre les valeurs NL_NO (0001), FORTRAN (0002), NL_ALL (0004), NP_NL (0008), NP_ALL (0010), NO_NL (0020), NO_ALL (0040) ou NO_NO (0080). Les deux premières lettres indiquent l'action en fin de record: NL pour newline, NP pour newpage et NO pour aucune action. Les deux dernières lettres indiquent les caractères pris en considération pour le formatage: NO pour aucun, NL pour uniquement <NL> et ALL pour les caractères <CR>, <LF>, <NL>, <BS>, <FF> et <NP>. La valeur FORTRAN indique que chaque caractère commence par un caractère de contrôle ANSI c'est-à-dire ' ' pour newline, '+' pour carriage return, '1' pour newpage et '0' pour deux newline; aucune autre action n'est effectuée. Le mode FORMAT est déterminé par l'attribut Text Formating [03]. Sa valeur par défaut est (0001) c'est-à-dire que la fin des records impliquent un newline et qu'aucune autre action n'est prise, ce qui convient à un ficher texte qui est destiné à l'écran où à l'imprimante. Les modes PAGE-WIDTH et PAGE-LENGTH permettent de définir la largeur et la longueur de page pour les actions du formatage et sont déterminés par les attributs Page Width [26] et Page Length [27].

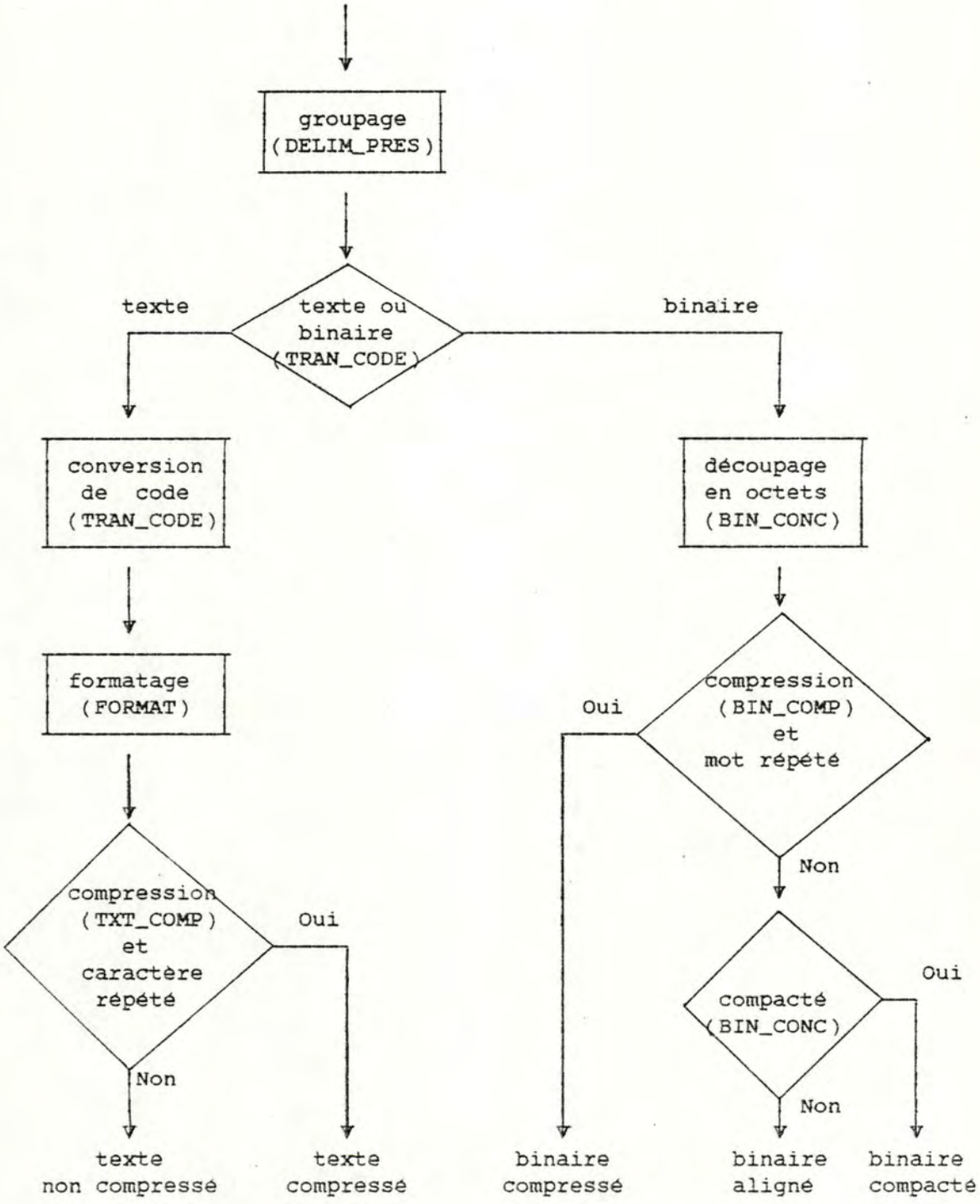
Le mode DELIM-PRES prend pour valeur la somme de R_BOUND (0001), si les délimiteurs de fin de record doivent être préservés, et TAB_CHAR, s'il faut préserver les tabulations. La valeur de ce mode est déterminée par l'attribut Delimiter Preservation [21]. Les délimiteurs de fin de record doivent être préservés s'ils ont une signification pour le formatage (cfr attribut Text Formating [03]). La non préservation des délimiteurs permet de scinder ou de grouper des records pour satisfaire aux attributs Maximum Transfert Record Size [05] et Maximum Storage Record Size [25]. Si on ne demande pas la préservation des tabulations elles peuvent être remplacés par des blancs suivant la transformation définie par le mode TAB qui fournit un string avec un caractère non blanc à chaque tabulation. Les tabulations sont déterminés par l'attribut Horizontal Tabs [23] et l'interprétation détaillée des tabulations peut être trouvée dans le texte de la norme NIFTP à la section 3.2 [NIFTP].

2.6.3. Version minimum

Dans un premier temps la maquette pourra ne comprendre qu'une version minimum pour le module PRESENTATION. Cette version minimum n'autoriserait que le tranfert de texte IA5 sans compression ni formatage. Elle ne devrait donc qu'assurer le découpage en subrecord. A noter que sous cette forme, elle pourrait assurer le transfert binaire pour des mots de 8 bits. Dans un deuxième stade on pourrait ajouter la compression puis le formatage. Ensuite pour le binaire, il faudrait implémenter la découpe de mots de plus de 8 bits, puis la compression et finalement le compactage. Pour compléter le module, il faudrait encore y ajouter la possibilité de scinder ou de regrouper des records.

2.6.4. Enchaînement des actions

Voici un diagramme d'enchaînement possible des différentes actions, avec entre parenthèses les modes intervenant.



2.7. Le module NEGOCIATEUR

2.7.1. Description

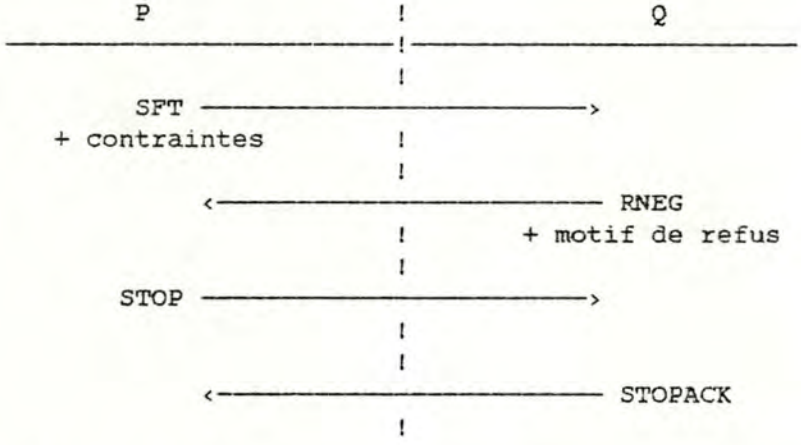
Le module NEGOCIATEUR a pour but de gérer les phases d'intialisation et de terminaison, c'est-à-dire toute la négociation des attributs du fichier et du transfert. Pour cela six commandes sont définies, elles sont:

nom	signification	envoyé par
STOP	Demande de terminaison	P
GO	Démarrage du transfert de données	P
RPOS	Réponse positive	Q
RNEG	Réponse négative	Q
SFT	Démarrage du transfert de fichier	P
STOPACK	Acceptation de la terminaison	Q

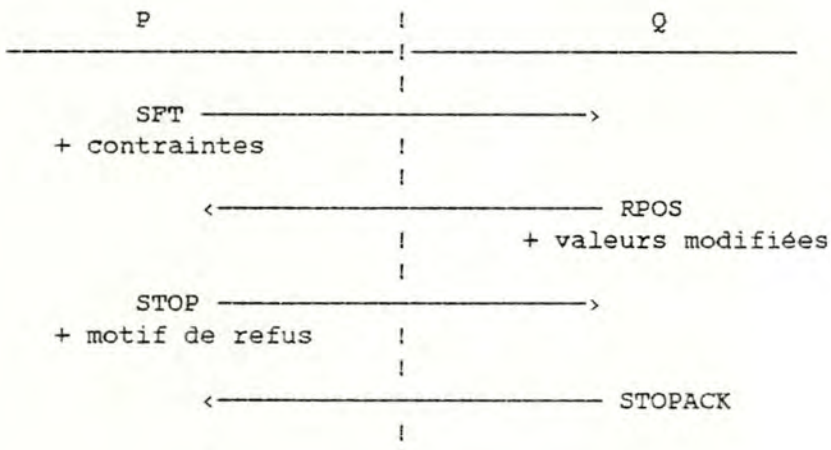
Les commandes SFT, RPOS et RNEG ont des paramètres qui déterminent des contraintes sur les attributs du fichier et du transfert. On appelle docket un ensemble de contraintes définissant pour chaque attribut un opérateur et une valeur. Chaque site maintient un docket appelé transfer_docket. Le but de la négociation est de faire en sorte que les deux dockets contiennent les mêmes valeurs.

2.7.2. Schéma de la négociation

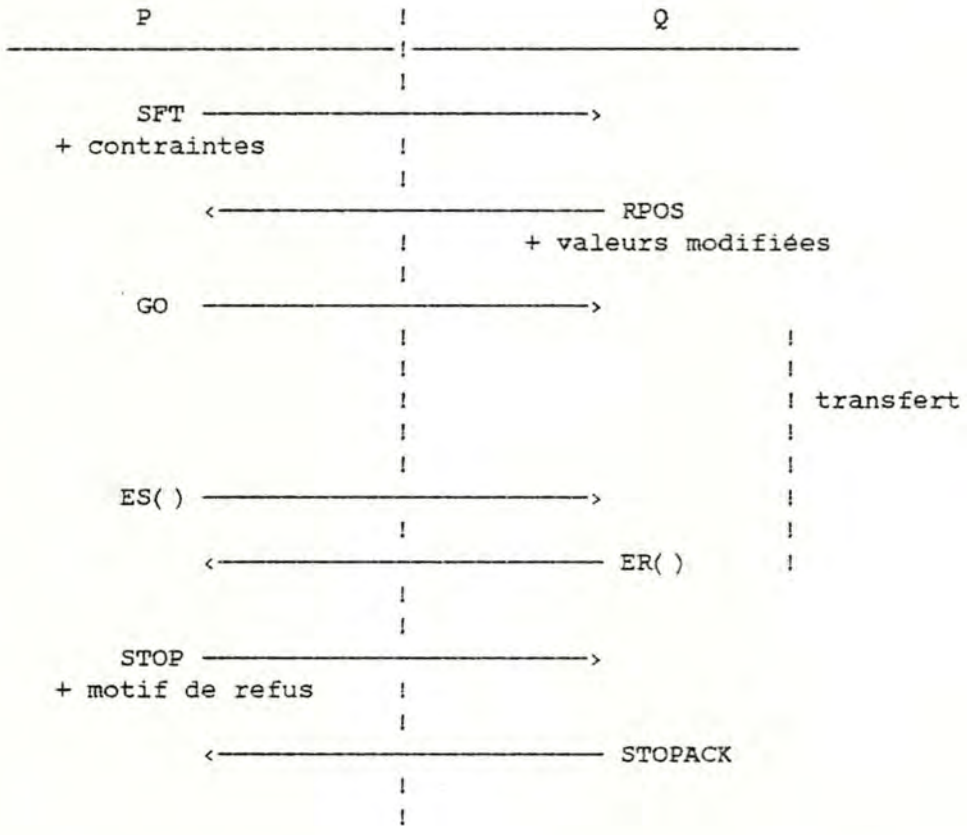
Pour démarrer un transfert de fichier, le processus P envoie une commande SFT avec les paramètres associés vers le processus Q. Si le processus Q choisi de rejeter le transfert de fichier, il renvoie une réponse négative RNEG, avec les paramètres associés indiquant le motif du rejet. Le processus P envoie alors une commande STOP que le processus Q accepte avec une commande STOPACK pour terminer le transfert.



Si, après réception du SFT, le processus Q accepte le transfert, il renvoie la commande RPOS, avec éventuellement des valeurs d'attributs ajustées. Il doit fournir un paramètre pour chaque attribut demandé par le processus P en utilisant le monitor flag. Si cependant le processus P trouve que les valeurs modifiées sont inadéquates, il renvoie une commande STOP au processus Q avec des paramètres donnant la raison du refus. Notez que les paramètres doivent au moins comprendre les attributs State of Transfer et Information Message. Le processus Q accepte la terminaison du transfert en renvoyant une commande STOPACK.



Si le processus P accepte les valeurs fournies par la commande RPOS, il provoque l'entrée dans la phase de transfert de données en envoyant une commande GO. A la fin du transfert lorsque l'envoyeur a envoyé la commande de contrôle ES et que le receveur a répondu par la commande de contrôle ER, le processus P envoie une commande STOP pour terminer le transfert ce que le processus Q confirme par une commande STOPACK.



Dans certaines circonstances, s'il y a une importante coupure du réseau, le transfert peut être complètement interrompu. D'autre part, il est parfois désirable pour des questions de gestion, d'arrêter un transfert de fichier, par exemple si une machine a prévu une interruption de service. Cette facilité est appelée une reprise d'un transfert. Dans ce cas, de l'information doit être mémorisée entre deux transferts de fichier et l'utilisation de la facilité doit être convenue durant la phase de négociation du transfert original en mettant à un le bit 1 de l'attribut Facilities [0E]. Le protocole spécifie quel participant à la responsabilité de la reprise et quelles valeurs d'attributs doivent être employées. Ces exigences demandent la mémorisation des dockets. Cette gestion des dockets mémorisés étant compliquée, j'ai décidé de ne pas l'inclure dans la maquette. La reprise se fera en retransmettant l'entièreté du fichier.

2.7.3. Attributs

2.7.3.1. Règles de négociation

Il y a trois classes d'attributs: ceux qui réfèrent le système de fichier en Q, appelés attributs_de_stockage ou Q-attributs, ceux qui réfèrent aux détails du transfert réel, appelés attributs_de_transfert ou T-attributs, et ceux qui ne sont pas négociés mais passent des informations supplémentaires, appelés attributs_d'information ou I-attributs. Le but de la négociation est de se mettre d'accord sur les valeurs des Q- et T-attributs. Il n'y a pas de négociation des I-attributs, mais ils peuvent contribuer au rejet du transfert.

La phase d'initialisation négocie les attributs appropriés au transfert. Chaque attribut ne peut être référencé que par au plus un paramètre à chaque commande, à l'exception des Action et Information Messages. Cette restriction est imposée pour permettre des implémentations simples du processus Q. Chaque paramètre contient soit une valeur et l'opérateur relationnel associé, ou une indication sur le motif pour lequel aucune valeur n'est donnée (soit que l'attribut est inconnu ou qu'aucune valeur n'est disponible). Lorsqu'un paramètre dans la commande SFT a une valeur, l'opérateur relationnel indique la contrainte que P place sur la liberté de Q de spécifier une autre valeur dans la commande RPOS. Si pour un certain attribut, la commande SFT ne donne pas de paramètre, ou si le paramètre ne contient pas de valeur, alors Q peut spécifier toute valeur désirée dans la commnde RPOS. Si ni P (dans la commande SFT), ni Q (dans la commande RPOS) ne fournit une valeur pour l'attribut, la valeur par défaut est supposée. Dans ce cas, s'il n'y a pas de valeur par défaut, aucune hypothèse ne peut être faite sur la valeur utilisée par l'autre partie.

Pour les attributs qui sont négociables, les valeurs sont contraintes par les capacités de l'implémentation et, plus spécialement pour certains attributs, par ce que l'utilisateur ou l'implémenteur exige pour un essai particulier de transfert. Ces contraintes sont appelées les capacités ou les exigences, respectivement, des processus FTP appropriés. Puisque le protocole impose une limite d'un paramètre par attribut et par commande (pour les Q- et T-attributs), P est obligé de choisir entre spécifier les capacités ou les exigences. Si une valeur spécifique est nécessaire, l'opérateur d'égalité sera utilisé et la distinction entre capacités et exigences n'a pas d'importance.

Une fois que le processus P a spécifié ses capacités ou ses exigences dans la commande SFT, le processus Q peut comparer celles-ci avec ses propres capacités et exigences. Le transfert peut etre accepté si une valeur satisfaisant ces contraintes peut être trouvée pour chaque Q- et T-attribut. Lorsque le processus Q a encore le choix entre un domaine de valeurs possibles pour un attribut, il en choisit une et la renvoie dans la commande RPOS.

Si le processus Q omet un paramètre dans la commande RPOS, il accepte implicitement la valeur donnée par la commande SFT. Le processus P peut exiger que le processus Q inclue dans la commande RPOS une valeur pour un paramètre donné. Cela permet à P de déterminer la valeur prise par Q sans imposer lui-même une contrainte.

Si un des paramètre de la commande SFT réfère un attribut qui n'est pas implémenté par le processus Q, celui-ci doit répondre avec un paramètre indiquant "unknow attribute" plutôt que de rejeter le transfert. Q supposera que toutes les contraintes pour cet attribut sont satisfaites, bien que P puisse décider du contraire et rejeter le transfert. De même, si P reçoit dans un RPOS un paramètre qu'il ne connaît pas, il peut choisir de continuer ou non. Ces situations peuvent arriver parce que des implémentations simples ne doivent pas inclure tous les attributs actuellement définis. Des implémentation permettant l'emploi de faciltés optionnelles doivent être prêtes au refus de ces facilités et être capables de travailler à un niveau plus simple. En particulier toute implémentation qui doit être utilisée dans un environnement ouvert doit être prête à travailler au niveau minimum décrit dans l'Appendice III du texte de la norme NIFTP [NIFTP].

Une stratégie d'implémentation est décrite à la page 111 du texte de la norme NIFTP [NIFTP]. Dans cette implémentation chaque site maintient deux dockets: LOCAL, pour les capacités et les exigences de l'implémentation, et REMOTE, pour l'état courant des valeurs d'attributs convenues entre les deux parties. P_LOCAL est initialisé par les contraintes de l'implémentation de P puis par les exigences de l'utilisateur. Ensuite P_REMOTE est mis à jour en fonction de P_LOCAL et cela produit les paramètres de la commande SFT. A la réception du SFT, Q vérifie les paramètres et leurs dépendances, puis met à jour Q_REMOTE. Ensuite il compare Q_REMOTE et Q_LOCAL, puis fixe une valeur qu'il met dans Q_LOCAL. Après cela, Q_REMOTE est mis à jour en fonction de Q_LOCAL et cela produit les paramètres de RPOS. A la réception du RPOS, P vérifie les paramètres et leurs dépendances, puis met à jour P_REMOTE et P_LOCAL. A ce moment les quatres dockets contiennent les mêmes valeurs et P peut envoyer la commande GO.

2.7.3.2. Liste des attributs

Voici la liste des attributs, avec un premier tableau donnant l'identifiant de l'attribut, le type d'attribut (T, Q ou I), son utilisation et son nom anglais. J'ai réparti les attributs en quatres classes d'utilisation: 'p', pour les attributs définissant les fonctions du module PRESENTATION (cfr 2.6.8.), 'f', pour les attributs donnant les caractéristiques du fichier, ils servent à définir les actions à effectuer lors de la sélection, l'ouverture et la fermeture du fichier, 'w' pour les attributs concernant la gestion de la fenëtre (cfr module TRANSFERT 2.8.3.) et 'n' pour les attributs ne servant qu'à la négociation, la signification de ces attributs sera expliquée après les tableaux. Le deuxième tableau donne le type de valeur, la valeur par défaut et la valeur pour l'implémentation. Une description détaillée de chaque attribut peut être trouvée dans la section 7. du texte de la norme NIFTP [NIFTP].

notation	identifiant, type et utilisation			nom en anglais
PROT_IDENT	00	I	n	Protocol Identification
ACCES_MODE	01	T	f	Mode of Access
T_TRA_CODE	02	T	p	Text Transfer Code
TXT_FORMAT	03	T	p	Text Formatting
BIN_FORMAT	04	T	p	Binary Format
MAX_TR_REC	05	T	p	Maximum Transfer Record Size
TRAN_LIMIT	06	T	f	Transmission Limit
DATA_ESTIM	07	I	f	Data Estimate
TRANSF_IDE	08	T	n	Transfer Identifier
PRI_T_CODE	09	T	p	Private Transfer Code Name
ACK_WINDOW	0A	T	w	Acknowledgement Window
INIT_RMARK	0B	T	w	Initial Restart Mark
MIN_TIME_O	0D	T		Minimum Timeout
FACILITIES	0E	T	p	Facilities
TRAN_STATE	0F	I	n	State of Transfer
DATA_TYPE	20	T	p	Data Type
DELIM_PRES	21	T	p	Delimiter Preservation
T_STO_CODE	22	Q	p	Text Storage Code
HORIZ_TABS	23	T	p	Horizontal Tabs
B_WORD_SIZ	24	T	p	Binary Word Size
MAX_ST_REC	25	Q	p	Maximum Storage Record Size
PAGE_WIDTH	26	T	p	Page Width
PAGE_LENGT	27	T	p	Page Length
PRI_S_CODE	29	Q	p	Private Storage Code Name
FILENAME	40	Q	f	Filename
USERNAME	42	Q	f	Username
USER_PASSW	44	Q	f	Username Password
FILE_PASSW	45	Q	f	File Password
ACCOUNT	4A	Q	f	Account
ACCO_PASSW	4B	Q	f	Account Password
OUT_DEVICE	50	Q	f	Output Device Type
DEV_TYPE_Q	51	Q	f	Device Type Qualifier
FILE_SIZE	60	Q	f	File Size
ACTION_MSG	70	I	n	Action Message
INFORM_MSG	71	I	n	Information Message
SPECIAL_OP	80	*		Special Options

notation	type de valeur	implémentation opérat.	valeur	valeur par défaut
PROT_IDENT	bits	EQ	0100	-
ACCES_MODE	bits	EQ	....	none
T_TRA_CODE	bits	LE	000D	0001
TXT_FORMAT	bits	LE	00FF	0001
BIN_FORMAT	bits	LE	C003	8002
MAX_TR_REC	entier	LE	FFFF	FFFF
TRAN_LIMIT	entier	LE	FFFF	FFFF
DATA_ESTIM	entier	A	....	none
TRANSP_IDE	entier	Unknow		none
PRI_T_CODE	string	Unknow		""
ACK_WINDOW	entier	LE	00FF	00FF
INIT_RMARK	entier	EQ	0000	0000
MIN_TIME_O	entier	GE	....	0258
FACILITIES	bits	LE	005F	0000
TRAN_STATE	bits	-		-
DATA_TYPE	bits	LE	000F	0001
DELIM_PRES	bits	LE	8001	0000
T_STO_CODE	bits	LE	000D	none
HORIZ_TABS	string	A	-	"X"
B_WORD_SIZ	entier	GE	0001	0008
MAX_ST_REC	entier	A	-	none
PAGE_WIDTH	entier	LE	FFFF	none
PAGE_LENGT	entier	LE	FFFF	none
PRI_S_CODE	string	Unknow		""
FILENAME	string	EQ	....	none
USERNAME	string	EQ	....	none
USER_PASSW	string	EQ	....	none
FILE_PASSW	string	Unkonw		none
ACCOUNT	string	Unknow		none
ACCO_PASSW	string	Unknow		none
OUT_DEVICE	string	A	-	LP
DEV_TYPE_Q	string	A	-	""
FILE_SIZE	entier	EQ	....	none
ACTION_MSG	string	-		-
INFORM_MSG	string	-		-
SPECIAL_OP	*	Unknow		-

- signifie qu'une valeur par défaut n'a pas de sens
none signifie qu'il n'y a pas de valeur disponible
.... signifie que la valeur dépend du transfert demandé

L'attribut PROT-IDENT sert à identifier le protocole utilisé. Cette version-ci porte le numéro [00xx] les deux derniers chiffres étant un numéro de version de l'implémentation. Ce paramètre doit toujours figurer avec l'opérateur d'égalité.

L'attribut TRANSF-IDE identifie un transfert particulier pour la reprise d'un transfert arrêté. Comme la reprise n'est pas définie dans cette maquette, cet attribut n'est pas utilisé.

L'attribut MIN-TIME-O spécifie l'intervale minimum en secondes après lequel le transfert peut être arrêté unilatéralement. Il entre dans la détermination des valeurs de timeout. La valeur par défaut [0258] correspond à dix minutes.

L'attribut TRAN-STATE est utilisé pour fournir de l'information sur l'état courant du transfert, en particulier pour indiquer si un transfert qui a échoué peut être utilement réessayé. Il peut prendre les valeurs 'Viable', 'Rejetected', 'Terminated' ou 'Aborted'. Pour les différentes valeurs voir le texte de la norme NIFTP aux pages 28 et 74 [NIFTP].

L'attribut ACTION-MSG permet d'envoyer un message à un opérateur spécifiant quelle intervention est nécessaire avant que le transfert puisse avoir lieu. Ce paramètre doit toujours avoir l'opérateur EQ et une valeur explicite. Le monitor flag ne peut pas être utilisé.

L'attribut INFORM-MSG permet d'envoyer de messages à un terminal utilisateur, une boîte au lettres ou un fichier log. Ce paramètre doit également toujours avoir l'opérateur EQ et une valeur explicite. Le monitor flag ne peut pas non plus être utilisé.

L'attribut SPECIAL-OP permet de spécifier des options spéciales non couvertes par les attributs standards. Il n'est pas utilisé dans cette maquette.

2.7.3.3. Dépendances entre attributs

Les valeurs de certains attributs peuvent mofifier l'importance d'autres attributs et la validité de leurs valeurs. Ces dépendances sont notées en commentaires de la section 7. du texte de la norme NIFTP [NIFTP]. Les attributs les plus importants pour contrôler les dépendances fixes sont:

- Data Type [20]
- Mode of Access [01]

- Facilities [0E]
- Delimiter Preservation [21]

En outre certaines dépendances sont spécifiques à une implémentation ou à une façon d'utiliser le protocole. Elles impliquent généralement les attributs suivants:

- Mode of Access [01]
- Delimiter Preservation [21]
- Horizontal Tabs [23]
- Text Formatting [03]

2.7.3.4. La gestion des dockets

Ce paragraphe résume quand les dockets sont créé et quand ils sont détruits. P crée un transfer docket avant d'envoyer la commande SFT initiale et ce docket est utilisé jusqu'à ce qu'un des événements suivant survient:

a. Un STOPACK est reçu indiquant que le transfert est terminé avec succès.
b. Un RNEG est reçu indiquant que Q ne retient pas de docket.
c. Un STOPACK est reçu indiquant qu'une erreur s'est produite pour laquelle aucun nouvel essai n'est possible.
d. Un RNEG est reçu en réponse à une demande de reprise alors que Q n'a pas retenu l'ancien docket. (Ce sera toujours le cas ici puisqu'on ne prévoit pas la reprise)

Q crée un transfer docket s'il répond au SFT initial par un RPOS. Il reste jusqu'à la réception d'un STOP indiquant que le transfert est terminé avec succès ou avec échec.

A noter que l'ouverture du fichier se fait lors de la création du docket et la fermeture lors de la destruction du docket.

2.7.4. Primitives et actions sur les dockets

2.7.4.1. Primitives

Le module NEGOCIATEUR offre quatres primitives: deux pour l'envoi des commandes, P_envoi et Q_envoi, et deux pour la reception, P_reception et Q_reception.

P_envoi(cmd)

```
COMMANDE  cmd;
```

Cette primitive assure l'envoi des commandes STOP, GO et SFT. Elle crée et met à jour les dockets et construit un record contenant la commande qu'elle passe au module PRESENTATION pour être formaté et envoyé.

Q_envoi(cmd)

```
COMMANDE  cmd;
```

Cette primitive assure l'envoi des commandes RPOS, RNEG et STOPACK. Elle crée et met à jour les dockets et construit un record contenant la commande qu'elle passe au module PRESENTATION pour être formaté et envoyé.

P_reception(lmsg,msg)

```
int     lmsg;
char    *msg;
```

Cette primitive est appelée à la réception d'un événement Data en phase d'initialisation ou de terminaison. Elle reçoit les commandes RPOS, RNEG et STOPACK. Elle compare les paramètres reçus avec le docket et met celui-ci à jour ou le détruit. Elle produit les événements RNEG et STOPACK et, pour la commande RPOS, elle produit les événements RPOS_OK ou RPOS_NOK suivant que les paramètres sont acceptés ou refusés. Lorsque la facilité de reprise est implémentée, l'événement RPOS_EOF indique une reprise acceptable mais en fin de fichier.

Q_reception(lmsg,msg)

```
int     lmsg;
char    *msg;
```

Cette primitive est appelée à la réception d'un événement Data en phase d'initialisation ou de terminaison. Elle reçoit les commandes STOP, GO et SFT. Elle compare les paramètres reçus avec le docket et met celui-ci à jour ou le détruit. Elle produit les événements STOP et GO et, pour la commande SFT, elle produit les événements SFT_OK ou SFT_NOK suivant que les paramètres sont acceptés ou refusés.

2.7.4.2. Actions

Voici les différentes actions qui peuvent être effectuées sur les dockets.

create_docket( )

Cette primitive crée les docket REMOTE et LOCAL. Elle initialise LOCAL aux valeurs de l'implémentation et REMOTE aux valeurs par défaut.

delete_docket( )

Cette primitive a pour effet de détruire les dockets REMOTE et LOCAL.

init_docket( )

Cette primitive met à jour le docket LOCAL en fonction des exigences de l'utilisateur.

recep_att( )

A la réception d'une commande SFT ou RPOS, cette primitive vérifie la correction des paramètres puis met à jour le docket REMOTE. Ensuite, elle compare REMOTE et LOCAL puis vérifie les dépendances entre attributs. Finalement, elle met à jour LOCAL à partir de REMOTE avec des opérateurs EQ, choissisant une valeur particulière lorsqu'il reste un domaine de valeurs possibles. La primitive retourne un status OK ou NOK suivant que les paramètres sont acceptables ou non.

update_att( )

Cette primitive met à jour le docket REMOTE à partir du docket LOCAL en produisant les paramètres pour la commande SFT ou RPOS.

liste_att( )

Cette primitive produit la liste des attributs qui ont étés rejetés pour l'envoi d'une commande RNEG.

2.7.4.3. Algorithmes

Voici les actions à effectuer lors de l'envoi ou de la réception de chaque commande.

```
P_envoi
    STOP     : -
    GO       : -
    SFT      : create_docket; f_open;
                   init_docket; update_att;
Q_envoi
    RPOS     : update_att;
    RNEG     : liste_att;
    STOPACK  : -

P_reception
    RPOS     : recep_att;
    RNEG     : delete_docket; f_close;
    STOPACK  : delete_docket; f_close;

Q_reception
    STOP     : delete_docket; f_close;
    GO       : -
    SFT      : f_open; create_docket; recep_att;
```

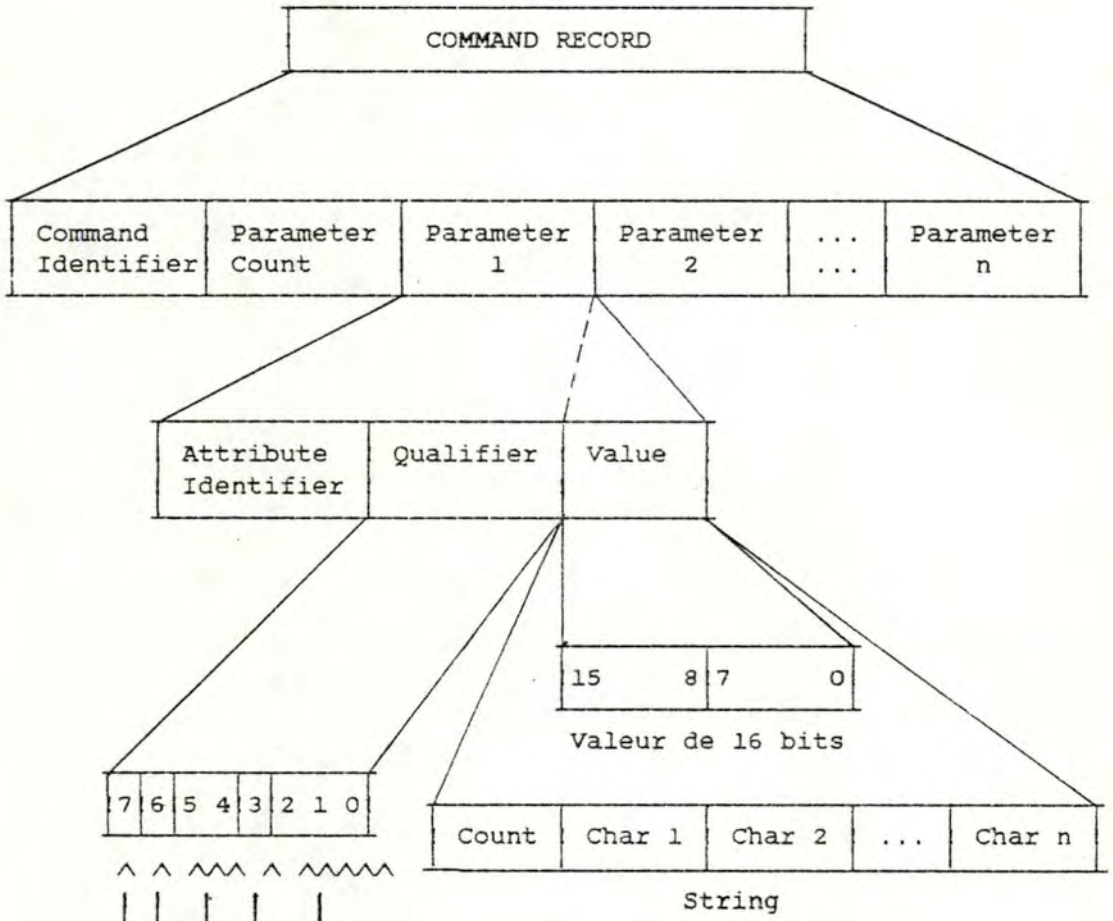
2.7.5. Formats

Les commandes sont encodées de la même manière que les records de données textes, chaque commande occupant exactement un record. Les commandes ne peuvent pas être en format compressé, mais elle peuvent être découpées en subrecords si nécessaire. Une commande est faite des composants suivants: un octet contenant l'identifiant de la commande, un octet donnant le nombre de paramètres dans la commande (de 0 à 255) et une liste de paramètres. Les identifiants des commandes sont les suivants:

STOP	00
GO	01
RPOS	02
RNEG	03
SFT	04
STOPACK	05

Les paramètres sont composés d'un octet donnant l'identifiant de l'attribut (cfr 2.7.3.), d'un octet qualificateur donnant la fonction spécifique du paramètre (il est décrit au paragraphe suivant) et une valeur dont la longueur dépend du champ 'format' du qualificateur. Cette longueur est nulle s'il n'y a pas de valeur, elle prend deux octets pour un entier ou un champ de bits et dans le cas d'un string elle est composé d'une suite de caractères précédés d'un octet indiquant sa longueur.

Le qualificateur est utilisé de la façon suivante: le monitor flag est utilisé par P pour demander à Q d'inclure une valeur explicite pour cet attribut dans la commande RPOS. Le champ 'format' spécifie s'il y a une valeur associée au paramètre et sa forme, comme suit: soit le champ 'valeur' est omit, soit la valeur est représentée par 16 bits, l'octet contenant les bits 15-8 étant transmis en premier, soit la valeur est un string c'est-à-dire un octet de longueur suivi de ce nombre de caractères IA5 de parité quelconque. L'opérateur avec la valeur associée au paramètre définit une contrainte qui doit être satisfaite par la valeur finale de l'attribut. Les détails de l'interprétation des paramètres est donné à la section 4.5 du texte de la norme NIFTP [NIFTP].



→ Opérateur
- [02] EQ Egal
- [03] LE Plus petit ou égal
- [05] NE Pas égal
- [06] GE Plus grand ou égal
- [07] ANY N'importe quelle valeur

→ Le bit 3 est réservé (= 0)

→ Format
- [00] Attribut inconnu (valeur absente)
- [10] Pas de valeur disponible (")
- [20] 16 bits (entier ou champ de bits)
- [30] String

→ Le bit 6 est réservé (= 0)

→ Monitor flag
- [00] Pas besoin de retour
- [01] Besoin d'un retour

2.8. Le module TRANSFERT

2.8.1. Description

Le module transfert a un double objectif: d'une part la gestion de l'accès au fichier et d'autre part le gestion de la fenêtre. Concernant le fichier, il s'occupe de l'ouvrir, de le fermer et d'en donner les attributs ainsi que d'écrire les données et les lire.

La fenêtre est utilisée par la méthode de recouvrement des erreurs par retransmission d'une partie du fichier durant le transfert. Cette méthode est appelée le mécanisme de restart . Le mécanisme de restart peut être employé lorsque l'un des deux terminaux, ou la ligne de communication entre eux, n'est pas suffisamment sûr. Il permet d'assurer de bout en bout la livraison et la mémorisation des données. Pour que le mécanisme fonctionne, les données ne doivent pas être acceptées (acknowledged) tant qu'elles ne sont pas mémorisées de façon sûre. L'application particulière et les caractéristiques de l'appareil de stockage détermineront quand les données peuvent être considérées comme mémorisées de façon sûre.

Pour utiliser le mécanisme de restart, il faut envoyer et accepter des points_de_synchronisation, appelés aussi marques_de_restart. L'envoyeur doit maintenir un index donnant, pour chaque marque depuis la dernière acceptée à la dernière envoyée, les détails sur l'endroit ou les données correspondantes sont stockées. La fenêtre détermine le nombre de marques qui peuvent être ainsi retenues, c'est-à-dire la taille de cet index. Dans certains cas, l'envoyeur ne devra pas retenir cet index explicitement, mais pourra calculer les positions demandées, par exemple lorsque les marques correspondent aux divisions de la structure du fichier.

Les caractéristiques locales des appareils employés influencera le positionement des points de synchronisation et la taille de la fenêtre. Les caractéristiques de découpage en blocs de l'appareil d'arrivée placent également des contraintes sur les points de synchronisation. Par exemple, un receveur stockant ses données sur disque n'acceptera les marque que quand les données seront effectivement sur le disque. Il mémorisera donc les marques reçues et ne renvera une acceptation de marque que quand le buffer sera plein. Celà peut poser des problèmes si la limite de la fenêtre est atteinte alors que le buffer n'est pas plein. La valeur par défaut de la fenêtre est 255 pour réduire l'occurence d'un tel problème. Des recommandations pour le placement des marques et le choix de la taille de la fenêtre se trouvent à la page 114 du texte de la norme NIFTP [NIFTP]. Dans le cas où la reprise de transfert est prévue, les marques sont numérotées de 0 à 65534 mais seulement les 8 bits de poids faible sont envoyés dans les commandes de contrôle MS, MR et RR.

Les actions sur la fenêtre sont: avancer la limite supérieure de la fenêtre (dernière marque envoyée) lors de l'envoi d'une marque, avancer la limite inférieure de la fenêtre (dernière marque acceptée) lors de l'acceptation d'une marque et reculer la limite supérieure de la fenêtre lorsqu'on a détecté une erreur et qu'on reprend une partie du transfert. Les algorithmes pour la gestion de la fenêtre sont décrits à la page 115 du texte de la norme NIFTP [NIFTP].

2.8.2. Primitives d'accès au fichier

Voici les primitives permettant l'accès au fichier: les trois premières sont utilisées durant les phases d'initialisation et de terminaison, les quatre dernières durant la phase de transfert.

STATUS f_open(filename,type)

```
char    *filename;
ACC_TYPE    type;
```

Cette primitive a pour effet d'ouvrir le fichier de nom FILENAME avec le mode d'accès TYPE. Le TYPE peut être 'r' pour une ouverture en lecture, 'w' pour une ouverture en écriture et 'a' pour une ouverture en append. Si le fichier a pu être ouvert, la primitive retourne la valeur 1 comme STATUS mais si l'ouverture a échoué, soit parce que les protections d'écriture et de lecture ne le permettent pas, soit parce que le fichier n'existe pas alors qu'on a demandé une lecture, alors la valeur de STATUS est 0. Notez que la référence du fichier est une variable globale du module et est initialisée par cette primitive si l'ouverture réussi.

STATUS f_close(type)

```
CLO_TYPE  type;
```

Cette primitive permet de refermer le fichier en fin de transfert. TYPE donne le mode de fermeture: 'n' pour une fermeture du fichier sans autre action, 'd' pour effacer le fichier à la fermeture, 'i' pour transférer le fichier dans une file batch pour exécution, 'o' pour le retirer d'une file de fichiers de sortie.

VAL f_att(att);

```
ATTRIBUT   att;
```

Cette primitive est employée par le module négociateur pour connaître différents attributs du fichier. Elle fournit dans VAL la valeur de l'attribut ATT. Les différentes attributs qui peuvent être demandés sont EXIST pour demander si un fichier existe, PROTECT pour connaître les protections d'un fichier et SIZE pour connaître sa taille.

f_read(lbloc,bloc)

```
int     lbloc;
char    *bloc;
```

Cette primitive lit, dans le buffer pointé par BLOC, un bloc de données de longueur LBLOC. Elle retourne dans LBLOC le nombre de caractères effectivement lus. Si ce nombre est 0, celà veut dire que la fin de fichier est atteinte.

f_write(lbloc,bloc)

```
int     lbloc;
char    *bloc;
```

Cette primitive écrit le bloc de données de longueur LBLOC contenu dans le buffer pointé par BLOC. Elle retourne dans LBLOC le nombre caractères effectivement écrits.

f_fin( )

Cette primitive demande de vider le buffer et de marquer la fin de fichier.

f_seek(pos)

```
long    pos;
```

Cette primitive a pour effet de repositionner le pointeur de fichier de telle façon que la prochaine lecture ou écriture se fasse à la position POS.

2.8.3. Primitives de gestion de la fenêtre

Voici les primitives qui gèrent la fenêtre et le positionnement des points de synchronisation. Ces primitives sont désactivées si lors de la négociation, le mécanisme de restart n'est pas convenu entre les deux sites.

awl(num)

```
int  num;
```

Cette primitive fait avancer la limite supérieure de la fenêtre jusqu'à la marque NUM. Si la limite de la fenêtre est atteinte et que le processus est en mode S c'est-à-dire comme envoyeur, elle provoque l'événement Ack_W_End, tandis qu'en mode R, c'est-à-dire en tant que receveur elle provoque l'événement Ok_for_Ack.

awt(num)

```
int  num;
```

Cette primitive fait avancer la limite inférieure de la fenêtre jusqu'à la marque NUM. En mode R, c'est-à-dire en tant que receveur, elle provoque l'événement Ok_for_Ack.

rwl(num)

```
int  num;
```

Cette primitive est appelée lors de la détection d'une erreur et recule la limite supérieure de la fenêtre à la position NUM, si cette position est comprise entre la limite supérieure et la limite inférieure de la fenêtre. Si NUM est plus petit que la limite inférieure, alors la limite supérieure est ramenée à la limite inférieure tandis que si NUM est plus grand que la limite supérieure, rien n'est modifié. Dans tous les cas, NUM est remis à la valeur de la limite supérieure après modification.

n_read(nb)

```
int  nb;
```

Cette primitive indique au gérant de la fenêtre qu'un bloc de longueur NB a été lu. Elle provoque l'événement Mark_Pnt si un point de synchronisation a été atteint.

n_write(nb)

```
int  nb;
```

Cette primitive indique au gérant de la fenêtre qu'un bloc de longueur NB a été écrit. Elle provoque l'événement OK_for_Ack si un point de synchronisation a été atteint.

2.8.4. Primitives pour le transfert

Voici les primitives qui sont appelées par le module séquenceur pendant le transfert.

access(lbloc,bloc)

```
int     lbloc;
char    *bloc;
```

Cette primitive lit un record dans le buffer BLOC et dont la longueur est LBLOC. Elle fait appel à f_read et crée un événement Data_End si la fin du fichier est atteinte et Data_OK sinon. Puis elle fait appel à n_read pour mettre à jour la fenêtre.

keep(lbloc.bloc)

```
int     lbloc;
char    *bloc;
```

Cette primitive écrit un record contenu dans le buffer BLOC et dont la longueur est LBLOC. Elle fait appel à f_write, puis elle fait appel à n_write pour mettre à jour la fenêtre.

rewind(pos)

```
int     pos;
```

Cette primitive est appelée pour reprendre une partie du transfert lors de la détection d'une erreur. Elle fait appel à f_seek puis crée un événement Rest_Pnt.

f_termine( )

Cette primitive est appelée à la fin du transfert pour assurer l'écriture du dernier bloc de donnée. Elle fait appel à f_fin puis crée l'événement End_OK.

3. Les autres processus

3.1. Le processus NIVEAUX 1 A 4

Le but du processus NIVEAUX 1 A 4 est de simuler le service de transport entre les deux processus NIFTP. Il assure la communication entre les processus P et Q en affichant ce qui passe sur la ligne de transmission. En outre, il offre la possibilité d'intercepter des messages ou de les laisser passer un à un.

Le processus est connecté aux autres processus par six pipes:

P4	pipe de P	vers	NIVEAUX 1 A 4
4P	pipe de NIVEAUX 1 A 4	vers	P
Q4	pipe de Q	vers	NIVEAUX 1 A 4
4Q	pipe de NIVEAUX 1 A 4	vers	Q
S4	pipe de SUPERVISEUR	vers	NIVEAUX 1 A 4
4S	pipe de NIVEAUX 1 A 4	vers	SUPERVISEUR

Les pipes P4 et 4Q forment la connexion de P à Q et les pipes Q4 et 4P forment le retour, c'est-à-dire la connexion de Q à P. Chacune de ces deux connexions est traitée séparément et provoque des messages à l'écran dans des parties bien distinctes. Elles peuvent chacune travailler en mode automatique ou non. En mode automatique les messages sont passés directement d'un pipe à l'autre, en affichant les données du message, tandis qu'en mode non automatique une confirmation est demandée pour chaque message arrivant. Les pipes P4, 4P, Q4 et 4Q répondent aux spécifications définies à la section 2.4.4. dans le module INTERFACE NIVEAU 4. Le processus doit donc pouvoir répondre aux différents messages reçus. Par exemple un message TR_SNDND venant de P provoque, si une connexion est ouverte, un message TR_INF_EVENT indiquant data available en direction de Q et un message de status vers P.

Le pipe 4S permet d'écrire des messages à l'écran tandis que le pipe S4 permet de recevoir des commandes. Les commandes reçues de l'écran sont sous la forme d'une chaîne de caractères suivie d'un zéro. Les commandes suivantes sont comprises:

auto\0	pour passer au mode automatique ou en revenir
\0	pour accepter un message (un simple <return>)
kill\0	pour tuer un message
modif\0	pour insérer des parasites dans un message
reset\0	pour effectuer un reset de la connexion
close\0	pour causer une fermeture de la connexion
open\0	pour causer l'ouverture d'une connexion

L'écriture à l'écran via le pipe 4S se fait dans des zones de l'écran prédéfines qui sont:

Pour le sens de P vers Q
- Z_PQ_MSG
- Z_PQ_PROMPT
- Z_P_IN
- Z_Q_OUT

Pour le sens de Q vers p
- Z_QP_MSG
- Z_QP_PROMPT
- Z_Q_IN
- Z_P_OUT

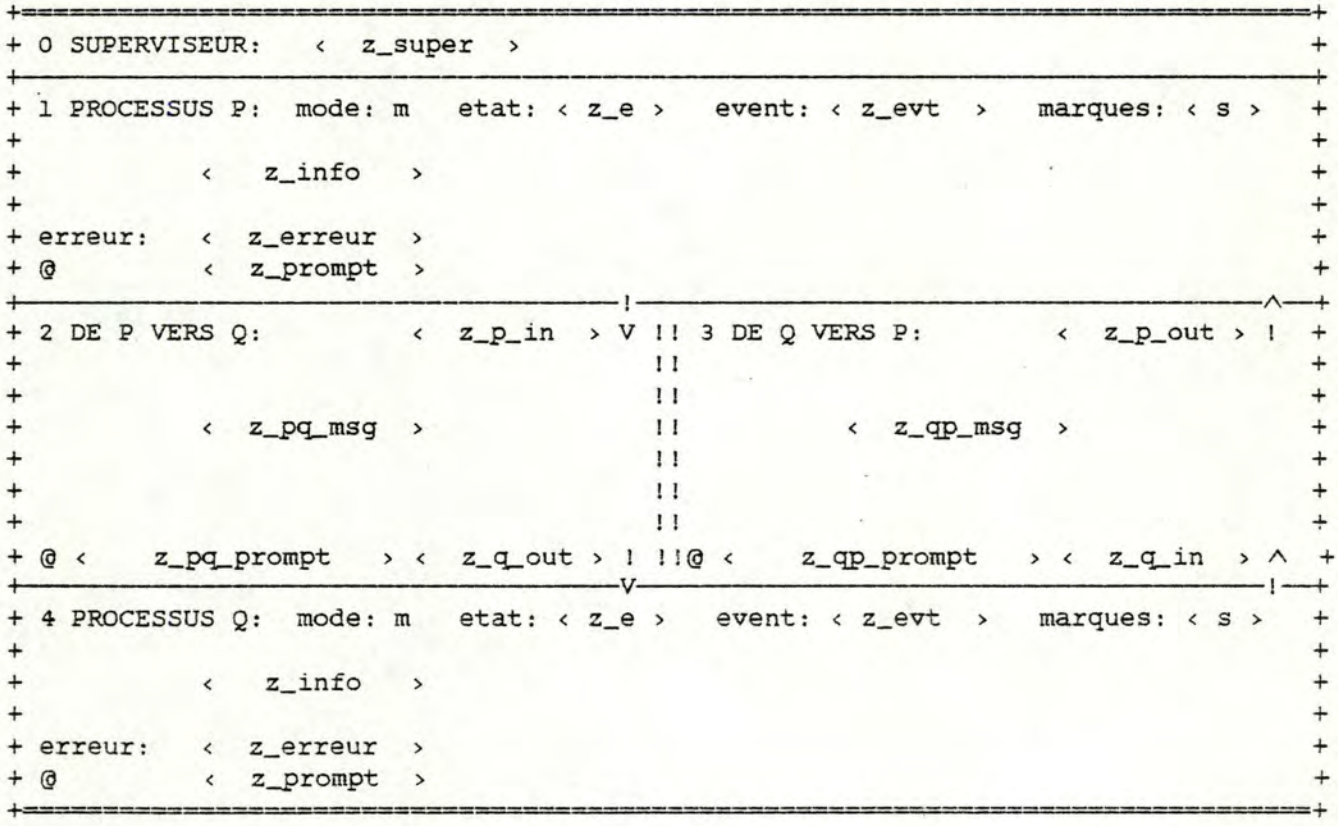
Les messages envoyés sur le pipe vers le superviseur ont la forme d'un octet représentant la zone suivi d'une suite de caractères à afficher terminée par un zéro.

3.2. Le processus SUPERVISEUR

Le processus SUPERVISEUR a pour but d'initialiser la maquette et d'assurer la gestion de l'écran. Il permet aux trois autres processus de travailler comme s'ils avaient chacun leur écran.

Lors du démarrage de la maquette, le processus SUPERVISEUR crée les trois autres processus comme étant ses fils. Ensuite il crée les différents pipes indiqués sur la figure A.1. Comme dans le système Unix il n'est possible de créer des pipes qu'entre des processus frères ou entre un processus père et un processus fils, tous les processus sont créé comme des fils du même processus SUPERVISEUR.

Pour assurer la gestion de l'écran celui-ci est divisé en cinq parties. La partie supérieure sert d'écran au processus SUPERVISEUR. Ensuite vient la partie réservée au processus P, puis deux zones pour le processus NIVEAUX 1 A 4, et finalement une zone pour le processus Q. La disposition proposée pour un écran de 24 lignes sur 80 colonnes est la suivante:



Les différentes zones sont repérées par rapport à la partie d'écran dans laquelle ils se trouvent et on ajoute un offset pour obtenir la place réelle. Ainsi les définitions pour les zones des processus P et Q sont les mêmes, seul l'offset diffère. Voici les coordonées des différentes zones sous la forme (ligne de départ, colonne de départ, nombre de lignes, nombre de colonnes):

```
processus SUPERVISEUR ( offset (0,0) )
        z_super         ( 0,16, 1,63)

processus P ( offset (2,0) ) et Q ( offset (18,0) )
        z_mode          ( 0,23, 1, 1)
        z_etat          ( 0,33, 1, 7)
        z_evt           ( 0,50, 1,10)
        z_stat          ( 0,72, 1, 5)
        z_info          ( 1, 1, 3,78)
        z_erreur        ( 4, 9, 1,70)
        z_prompt        ( 5, 3, 1,76)

processus NIVEAUX 1 A 4 ( offset (9,0) )
        z_pq_msg        ( 1, 1, 6,38)
        z_pq_prompt     ( 7, 3, 1,20)
        z_p_in          ( 0,24, 1,12)
        z_q_out         ( 7,24, 1,12)
        z_qp_msg        ( 1,41, 6,38)
        z_qp_prompt     ( 7,43, 1,20)
        z_q_in          ( 7,64, 1,12)
        z_p_out         ( 0,64, 1,12)
```

Le passage d'une partie de l'écran à l'autre, pour changer d'écran fictif, se fait par une interruption suivie d'un chiffre. Ainsi:

```
C-\ 1  déplace le curseur en <z_prompt> du processus P
C-\ 2  déplace le curseur en <z_pq_prompt>
C-\ 3  déplace le curseur en <z_qp_prompt>
C-\ 4  déplace le curseur en <z_prompt> du processus Q
C-\ 0  déplace le curseur en <z_super>
```

Le processus SUPERVISEUR lit alors une ligne de commande et l'envoie au processus correspondant. Les commandes acceptées par le superviseur sont: 'end' pour terminer la simulation et 'nettoie' pour raffraîchir l'écran.