Institutional Repository - Research Portal
Dépôt Institutionnel - Portail de la Recherche
researchportal.unamur.be
University of Namur

THESIS / THÈSE

MASTER EN SCIENCES INFORMATIQUES

Contribution à la mise en œuvre d'une couche de logiciel permettant un courrier électronique entre réseaux différents

Gathon, Patrice; Van Gyseghem, Alain

Award date:
1986

Awarding institution:
Universite de Namur

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 03. Jul. 2025

Facultés Notre-Dame
de la Paix
Namur

Année académique 1985-86

Contribution à la mise en oeuvre d'une couche de logiciel permettant un courrier électronique entre réseaux différents

Promoteur : Roland Lesuisse

Mémoire présenté par
Patrice Gathon et Alain Van Gyseghem
en vue de l'obtention du titre
de Licencié et Maître
en Informatique

Institut d'Informatique, rue Grandgagnage, 21 à 5000 Namur

Nous remercions monsieur Lesuisse, promoteur de ce mémoire, pour l'aide qu'il nous a apportée et la liberté qu'il nous a laissée tout au long de cette étude.

Nous remercions tout particulièrement monsieur Van Bastelaer pour ses remarques judicieuses sur l'aspect téléinformatique du travail.

Nous remercions également la société BIM pour les moyens qu'elle a mis à notre disposition, ainsi que tous ceux qui, de près ou de loin, ont contribué à la réalisation de ce travail.

Table des matières.

Chapitre 1 : Introduction.

1.1. Caractérisation générale du problème abordé.

L'objectif du mémoire est d'étudier la mise en oeuvre d'un courrier électronique sur un ensemble de réseaux différents.

Il existe actuellement bon nombre de courriers électroniques qui fonctionnent sur des réseaux locaux et sur des réseaux longue portée. Lorsqu'on désire avoir un seul courrier électronique entre des réseaux différents, deux voies peuvent être suivies :

- la première possibilité est de développer une interface entre des courriers électroniques différents, fonctionnant chacun sur un réseau local distinct [Red.-Whi. 83]. Cette possibilité est intéressante lorsqu'il existe déjà une configuration de réseaux locaux et de courriers électroniques. On permet ainsi aux courriers existant d'étendre leur rayon d'action de manière à obtenir une notion de courrier électronique global.
- la deuxième possibilité est de retenir un seul courrier électronique fonctionnant sur un réseau local et de le modifier de manière à ce qu'il travaille avec plusieurs réseaux locaux moyennant un minimum de changements dans ses fonctionnalités.

Nous allons étudier cette deuxième possibilité parce qu'elle demande une recherche préalable à tout développement et dépasse le cadre d'une implémentation d'interface.

Nous envisagerons donc les extensions et/ou modifications à apporter au courrier électronique fonctionnant en site local pour qu'il puisse prendre désormais en charge, de manière transparente à

l'utilisateur, le courrier interne au réseau local et le courrier interréseaux locaux via un réseau longue distance.

1.2. Les travaux apparentés.

Notre étude se démarque sensiblement des travaux menés actuellement [Def. 86] et qui se concentrent principalement sur les possibilités de relier entre eux plusieurs réseaux (locaux ou longue distance) qu'ils soient identiques ou de types différents; la difficulté résidant essentiellement dans les problèmes de conversion de protocoles.

Par ailleurs, des travaux sont également menés pour la réalisation de courriers électroniques étendus, et non plus locaux (travaux du CERN) [Heag. 86]. Cela, soit par l'extension de réseaux locaux existants, soit par la réalisation de courriers électroniques étendus, vus comme tels dès leur conception. Des problèmes se posent aussi à ce niveau et principalement lors de l'interconnexion de courriers de types différents, car il faut s'assurer à la fois du passage physique des documents mais aussi du passage logique des fonctions, d'une part implémentées différemment sur les divers courriers, mais, plus grave encore, peut-être fonctionnellement différentes, ce qui voudrait dire qu'elles ne réalisent pas exactement les mêmes tâches. D'où les tentatives pour aboutir à une standardisation des "Message Handling Systems", dont le fruit est l'apparition très récente de la norme X.400 [X.400].

Toutes ces études abordent donc le problème du côté de la réalisation pratique d'un courrier électronique. Notre démarche, par contre, attaque le problème dans l'autre sens en envisageant, pour un aspect primordial du courrier électronique qu'est le stockage des documents, quelles sont les différentes options possibles à ce niveau et leurs implications en fonction de critères, à la fois fonctionnels, c'est-à-dire relatifs à la

définition d'un problème, et de performances, c'est-à-dire relatifs aux performances du système. C'est pourquoi notre étude se trouve relativement distante des travaux menés actuellement.

1.3. Identification du contexte.

1.3.1. Du problème général au cas traité.

Le courrier électronique que nous utiliserons sera le noyau de celui qui a été développé par Jean-Marie Bernard et Alain Josis [Ber.-Jos. 85]. Nous reprendrons les concepts et les fonctions de base de ce courrier électronique pour ne pas alourdir notre étude avec des fonctions qui ne sont pas liées directement à la notion de courrier électronique.

Les réseaux locaux qui serviront de support au courrier électronique ne doivent pas être obligatoirement proches les uns des autres. C'est pourqoi il est nécessaire de les connecter au moyen d'un réseau à longue distance. Une manière de faire ceci est d'organiser ce réseau au moyen de lignes louées ou par l'intermédiaire du réseau téléphonique commuté. Une autre manière de faire est d'utiliser le réseau public DCS [Rad. 84]. Cette manière est plus efficace, car elle permet d'éviter les problèmes de routage qui sont résolus au sein de DCS [Flint], et elle simplifie fortement la gestion nécessaire lors de tout ajout ou suppression d'un réseau local dans le courrier électronique. C'est cette dernière solution que nous adopterons.

1.3.2. Définition des concepts.

Pour commencer, voyons ce l'on peut trouver sous la notion de COURRIER ELECTRONIQUE (CE). On appelle Courrier Electronique un système automatique de transmission et de distribution à

des abonnés de documents circulant à l'intérieur d'une organisation.

Mais définissons, à présent, les différents concepts liés à cette notion de courrier électronique. Ces concepts se situent à deux niveaux : tout d'abord au niveau du CE lui-même (concepts logiques : abonnés,documents), ensuite au niveau de la configuration sur laquelle repose le CE (concepts physiques : réseaux,postes,...).

a) les concepts logiques :

* on appelle *abonné* tout individu ou service auquel il est permis de transmettre ou recevoir des documents par le CE. Tout abonné est obligatoirement identifié par un identifiant d'abonné unique pour tout le système.

* on appelle *document*, tout texte transmis d'un abonné à un ou plusieurs autres abonnés; et par texte, on entend notes de service, courrier général, rapports, dossiers, à l'exclusion des photos, graphiques ou messages vocaux ou contenant des caractères spéciaux ... Le document est donc l'unité de communication entre deux abonnés du CE.

Structure d'un document:

chaque document est constitué de :

a. une entête où l'on trouve les informations suivantes :

- identifiant de document reprenant l'identifiant d'abonné de l'expéditeur et la date d'expédition;
- la liste des destinataires définie in extenso au moment de l'expédition.

b. un corps ou texte proprement dit.

* le *créateur* et l'*expéditeur* d'un document ne sont en fait qu'un seul et même abonné qui est à l'origine de l'existence du document et de son envoi vers d'autres abonnés.

* le *destinataire* d'un document est un abonné du CE auquel le document est destiné et envoyé. Un même document peut être envoyé à plusieurs

destinataires; on parlera alors de la liste des destinataires d'un document.

* les *liens* sont des associations établies entre des documents et des abonnés destinataires de ces documents; ainsi, un lien existe entre un abonné et un document donnés si l'abonné est un destinataire du document.

* la *boîte aux lettres* (ou bal) d'un abonné n'est autre que la liste des documents disponibles à un moment donné pour un abonné, c'est-à-dire les documents reçus par cet abonné et qu'il n'a pas encore supprimés.

b) les concepts physiques :

* le *poste de travail* est le moyen qu'a un abonné d'accéder au courrier électronique. Il s'agit d'un poste physique auquel l'abonné peut se présenter et exécuter un travail.

* le *serveur de base de données* a pour tâche la gestion des documents et autres données du courrier électronique.

* le *réseau local Ethernet* est le réseau local auquel sont connectés les différents postes de travail. C'est aussi sur le réseau local Ethernet que l'on trouvera le serveur de base de données.

La réalisation d'un courrier électronique étendu entraîne l'apparition de nouveaux concepts, principalement au niveau physique.

* le *réseau public DCS* est le réseau par lequel doit s'effectuer toute communication entre réseaux locaux Ethernet distincts.

* le *serveur passerelle* est l'interface entre les réseaux locaux du système et le réseau public DCS.

* pour chaque réseau local, on pourra regrouper le serveur de base de données et le serveur passerelle en un *serveur local*, composé donc de deux parties.

* on appelle *réseau de création* d'un document le réseau local sur lequel le document a été créé.

* on appelle *réseau destinataire* d'un document, un réseau local vers lequel on expédie le document.

* un serveur local est appellé *serveur éloigné* s'il est situé sur un réseau local différent du réseau de création.

1.3.3. Les fonctions d'un courrier électronique.

Un courrier électronique digne de ce nom doit pouvoir remplir différentes fonctions. Outre la gestion des documents, qui paraît évidente, il doit permettre notamment une gestion facile des abonnés et, pourquoi pas, offrir des facilités (plus secondaires) comme la gestion d'agendas, des envois en recommandé, des récupérations de documents jetés, etc...

La gestion des documents, sur laquelle nous reviendrons ultérieurement, concerne les accès possibles aux documents, qu'ils soient reçus, envoyés ou seulement créés. La nature de cette gestion sera donc diversifiée : création d'un nouveau document, consultation d'un document, expédition d'un document vers ses destinataires, suppression d'un document devenu inutile, consultation de la liste des documents disponibles (boîte aux lettres),... A ces fonctions de base, peuvent s'ajouter des fonctions secondaires, telles que les envois en recommandé, les mises à la poubelle, les envois avec demande de réponse, l'impression des documents, et l'on peut en imaginer bien d'autres. Néanmoins, dans la suite de notre étude nous ne tiendrons compte que des fonctions de base du système.

A part cette gestion des documents, le CE peut offrir une gestion des abonnés, sous la forme de fonctions administratives (introduction, suppression, localisation d'un abonné) et de fonctions d'aide, à l'expédition de documents par exemple (listes de destinataires prédéfinies,...) Ces fonctions seront également laissées de côté pour la suite de notre étude.

1.4. Articulation du travail.

Après avoir posé les objectifs du mémoire et défini les concepts et les fonctions de base qui seront utilisés, nous explicitons ici la suite de notre travail.

Dans un premier temps, nous nous attachons à identifier et à définir diverses politiques, c'est-à-dire des façons de mettre en oeuvre le courrier électronique sur des réseaux différents. Pour nous permettre d'évaluer les politiques, nous définissons aussi un ensemble de critères, certains étant des critères fonctionnels et d'autres des critères de performance.

Immédiatement après les définitions, nous procédons à une première évaluation des politiques sur base des critères. Cette évaluation est qualitative, c'est-à-dire basée sur un raisonnement théorique, et doit être complétée par une évaluation quantitative.

Afin d'effectuer cette évaluation quantitative, il est nécessaire d'approfondir les traitements à effectuer pour remplir les fonctions du courrier électronique et, ceci, dans chacune des diverses politiques. C'est pourqoi nous donnons une approche intuitive et détaillée des fonctions. Nous formalisons ensuite cette approche dans des diagrammes de flux et nous établissons une architecture logique qui sert de base au programme que nous avons implémenté.

Nous pouvons alors procéder à l'évaluation quantitative des politiques. Cette évaluation est basée sur notre programme et porte sur certains facteurs de temps et de coût intervenant lors de l'utilisation du courrier électronique.

Finalement, nous tirons certaines conclusions quant au choix d'une politique. Nous établissons les limites de notre étude et nous proposons quelques voies supplémentaires qui peuvent être approfondies.

Chapitre 2 : Les politiques et les critères d'évaluation.

Dans ce chapitre, nous commencerons par parcourir toutes les politiques possibles pour stocker les documents du courrier électronique. Nous identifierons donc les cas possibles et les définirons. Après cela, nous définirons les critères fonctionnels et les critères de performances qui nous permettront d'évaluer, par la suite, les diverses politiques.

2.1. Les politiques.

Examinons tout d'abord la figure 2.1 pour reconnaître les différents symboles qui seront utilisés dans les schémas de cette partie.

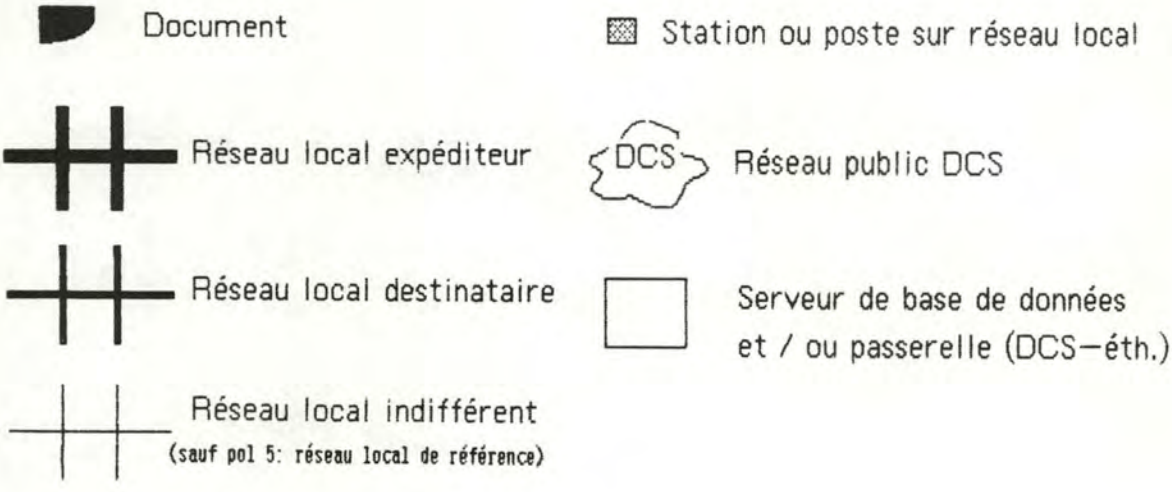

Fig. 2.1 : légende

2.1.1. Identification des politiques.

Nous avons vu, dans l'introduction, que nous allons travailler avec un courrier électronique implémenté sur plusieurs réseaux locaux reliés par le réseau public DCS. Nous savons aussi [Ber.-Jos. 85] [Op.-Dal. 83] qu'un courrier électronique peut être organisé de manière telle qu'il fonctionne avec une *clearinghouse* (fig. 2.2).

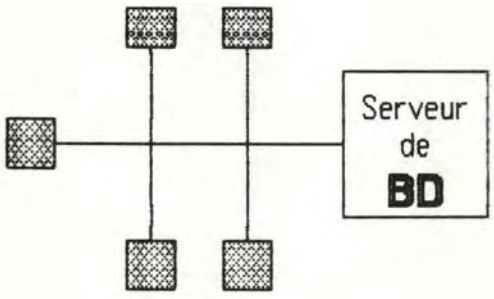

Fig. 2.2 : courrier électronique sur un réseau local, fonctionnant avec une clearinghouse

Cette "clearinghouse" est un serveur qui stocke les documents ainsi que des renseignements nécessaires au fonctionnement du courrier électronique (liste des abonnés, destinataires d'un document, etc...). Notre courrier électronique, en s'étendant (fig. 2.3), n'en a pas moins besoin de ce serveur. Au contraire, la nécessité d'une bonne organisation du stockage devient primordiale, étant donné le plus grand nombre d'utilisateurs du courrier électronique et, donc, l'activité totale plus intense.

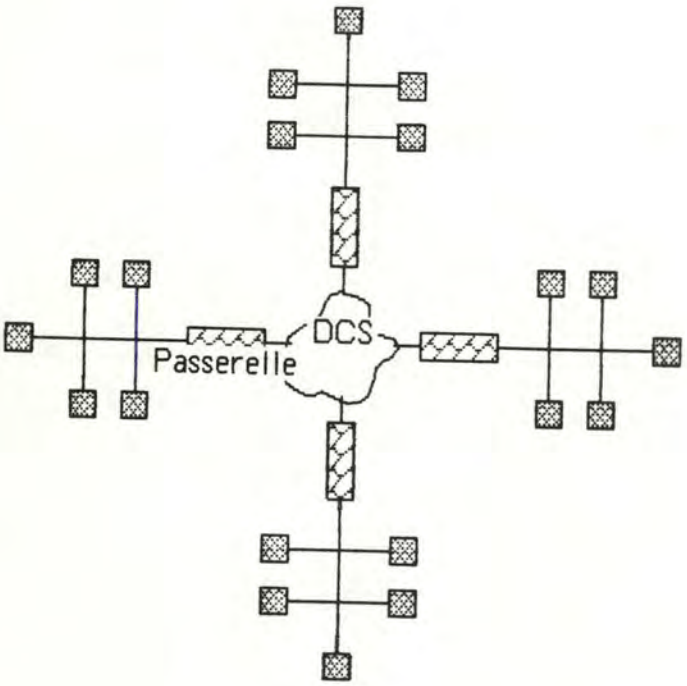

Fig. 2.3 : courrier électronique étendu

Nous avons besoin d'un *serveur de base de données* (clearinghouse) pour tout ce qui concerne le stockage. Ce serveur de base de données peut être unique, auquel cas il y a un seul serveur de base de données pour tout le courrier électronique, mais il peut aussi être multiple, auquel cas il y a un serveur de base données par réseau local. Des variantes sont possibles, par exemple un serveur par région. Voici donc deux possibilités à envisager: unicité ou multiplicité du serveur de base de données.

De même pour les documents stockés dans ce(s) serveur(s), nous pouvons imposer l'unicité de la copie d'un document ou autoriser la multiplicité des copies d'un document. Nous envisagerons donc également l'unicité ou la multiplicité pour un document.

En combinant les possibilités d'unicité et de multiplicté pour le serveur de base de données et pour les documents, nous obtenons quatre optiques de base qui

nous permettront de déterminer les politiques. Ces quatres optiques sont:

a) plusieurs serveurs de base de données et une seule copie par document,
b) plusieurs serveurs de base de données et plusieurs copies par document,
c) un seul serveur de base de données et une seule copie par document,
d) un seul serveur de base de données et plusieurs copies par document.

Se restreindre à un seul serveur de base de données et autoriser plusieurs copies par document (d) est irréaliste. De plus, cette optique entraîne une complexité de traitement plus élevée que l'optique réaliste qui s'en rapproche et qui est celle d'un serveur de base de données et d'une seule copie par document (c).

Nous ne poursuivrons pas l'étude de l'optique (d). Les autres optiques nous amèneront à définir les politiques.

2.1.2. Définition des politiques.

2.1.2.1. Politique 1.

Prenons la première hypothèse selon laquelle nous disposons de plusieurs serveurs de base de données et d'une seule copie par document. Il faut déterminer dans quelle serveur cette copie sera stockée. Plutôt que de la stocker aléatoirement dans un serveur de base de données, nous pouvons choisir de la stocker dans le serveur directement associé au réseau d'où le document est expédié (fig. 2.4).

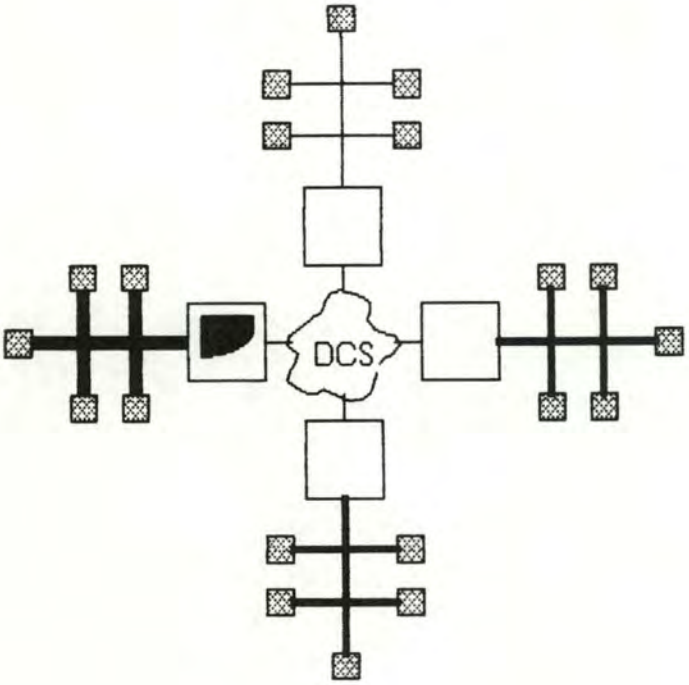

Fig. 2.4 : politique 1

Ce choix est justifiable par l'association logique que l'on peut faire entre le lieu de création ou d'expédition du document et l'endroit où il est stocké.

Cette répartition du stockage des documents sera appelée la *politique 1.* Elle a comme caractéristique essentielle l'unicité du document.

2.1.2.2. Politique 2.

Imaginons à présent que nous avons plusieurs serveurs de base de données et plusieurs copies par document. Il faut donc déterminer dans quels serveurs seront stockées des copies. On peut choisir d'avoir une copie par serveur de base de données ou rechercher le nombre minimum de copies.

Ainsi, si on veut minimiser le nombre de copies, il ne faut en stocker une que quand c'est nécessaire. Alors, si on appelle *réseau destinataire* un réseau où est localisé au moins un abonné destinataire du document, on ne stockera une copie de ce document que dans les serveurs de base de données qui sont associés à des réseaux destinataires (fig. 2.5).

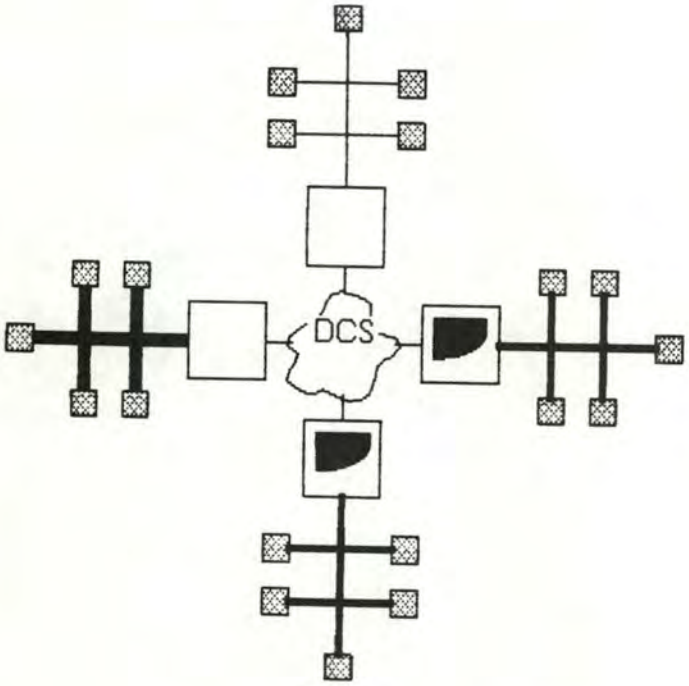

Fig. 2.5 : politique 2

Cette répartition du stockage des documents sera appelée la *politique 2*. Elle a comme caractéristiques essentielles la multiplicité possible du document et, comme nous l'avons sous-entendu dans la définition d'un réseau destinataire, la localisation des abonnés.

Dans cette optique, nous avons examiné le cas du nombre minimum de copies d'un document, mais une autre répartition est possible dans ce cas; elle donnera lieu à la définition de la politique 3.

2.1.2.3. Politique 3.

En nous rappelant la logique qu'il y a d'associer le lieu de création ou d'expédition d'un document et le lieu de stockage de ce document, nous pouvons imaginer le même mécanisme de répartition que pour la politique 2, c'est-à-dire une copie par serveur associé à un réseau destinataire, mais en ajoutant une copie obligatoire dans le serveur de base de données du réseau d'expédition du document (fig. 2.6.).

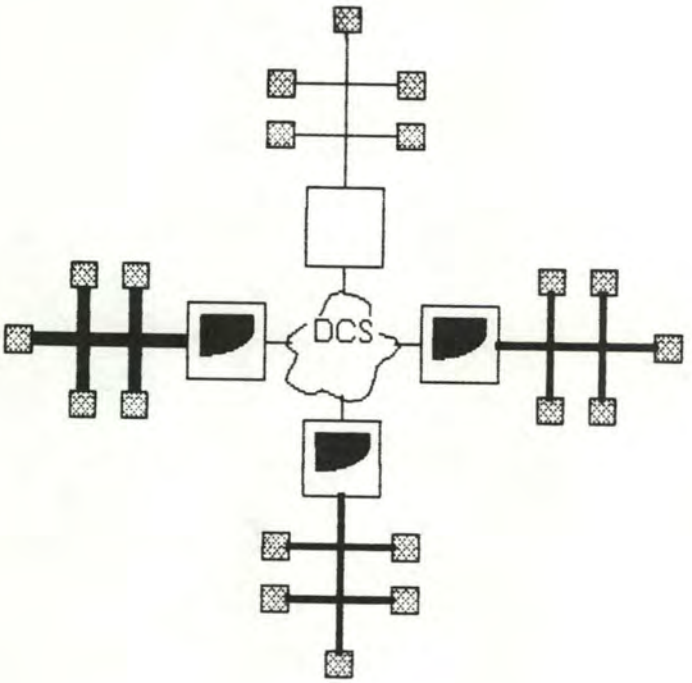

Fig. 2.6 : politique 3

Cette répartition du stockage des documents sera appelée la *politique 3*. Elle a comme caractéristiques principales la multiplicité du document et la localisation des abonnés.

Nous venons d'examiner deux cas de minimisation du nombre de copies d'un document. Il

n'est peut-être pas indispensable de minimiser ce nombre. La politique suivante suit cette idée.

2.1.2.4. Politique 4.

Il est peut-être, en effet, plus intéressant, puisqu'on a plusieurs copies d'un document, de ne pas se limiter au minimum, ceci afin de simplifier le traitement.

Dans ce cas, on peut choisir de stocker toujours une copie du document envoyé dans chaque serveur de base de données du courrier électronique (fig. 2.7).

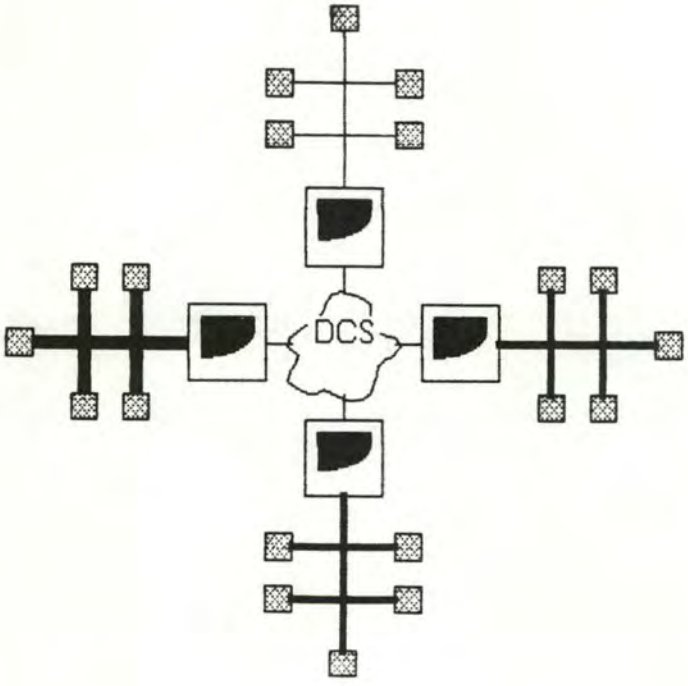

Fig. 2.7 : politique 4

Cette répartition du stockage des documents sera appelée la *politique 4*. Elle a comme caractéristique principale la multiplicité du

document. Remarquons que dans cette politique, la localisation des abonnés n'est plus indispensable.

Une variante de la politique 4 serait d'envoyer une copie du document dans tous les réseaux, et que chaque serveur ne stocke la copie que s'il y a au moins un destinataire sur le réseau. Cette variante nécessiterait la localisation des abonnés et, ainsi, nous ferait perdre un avantage important de la politique 4.

2.1.2.5. Politique 5.

Envisageons maintenant une troisième hypothèse: nous avons un seul serveur de base de données et une seule copie par document. Nous allons donc centraliser les documents dans le serveur unique.

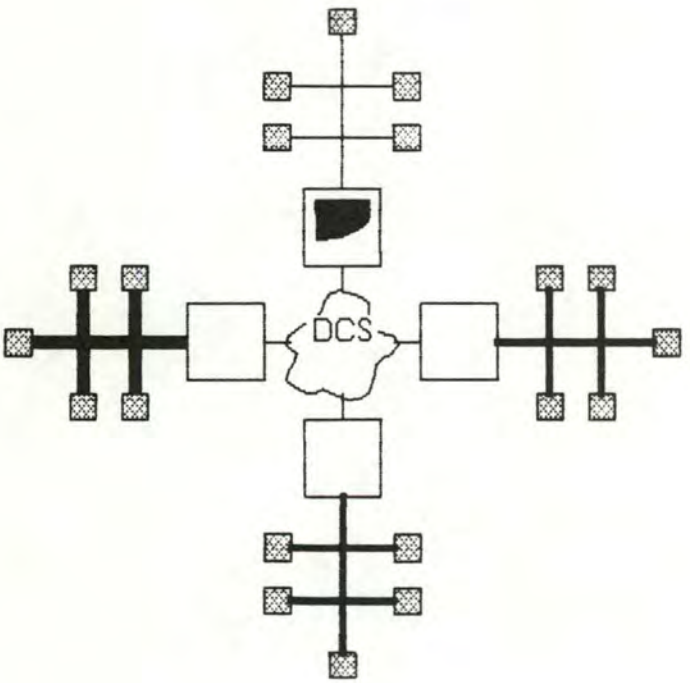

Fig 2.8 : politique 5

Nous stockerons chaque document du courrier électronique dans un seul serveur de base de données qui peut être associé à un réseau local du courrier électronique. Nous appellerons ce réseau *réseau de référence* (fig. 2.8).

Cette méthode de stockage des documents sera appelée la *politique 5*. Elle a comme caractéristiques principales l'unicité du document et la centralisation du stockage.

2.1.2.6. Politique 6.

Le serveur de base de données, unique comme au point précédent, peut être relié directement au réseau public DCS et ne pas être associé à un réseau local du courrier électronique (fig. 2.9).

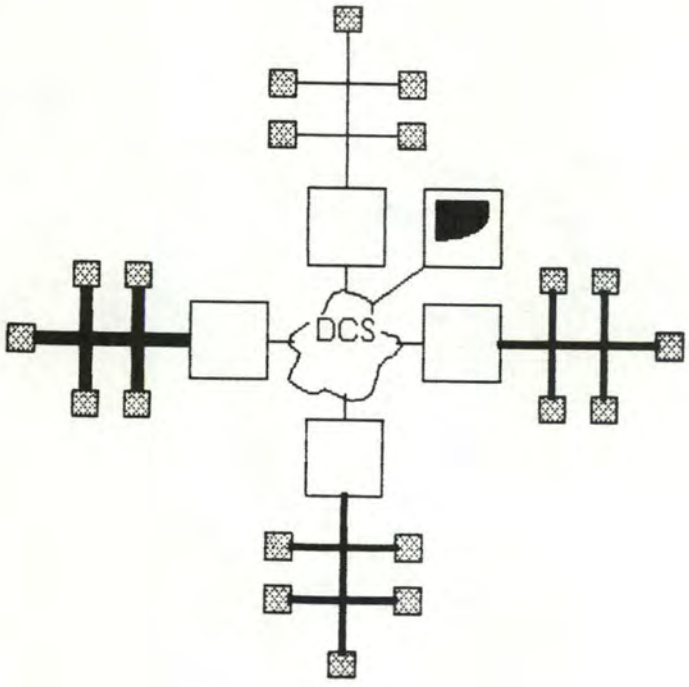

Fig. 2.9 : politique 6

Cette méthode sera appelée la *politique 6*. Elle a comme caractéristiques principales l'unicité du document et la centralisation du stockage.

Nous avons examiné, ici, toutes les possibilités que nous avions identifiées au point 2.1. Il en reste une que nous avions écartée dès le départ, mais que nous citerons.

2.1.2.7. Politique 7.

Les documents peuvent simplement être stockés dans les postes créateurs ou destinataires. Cette politique ne correspond néanmoins pas à l'organisation d'un courrier électronique fonctionnant avec une clearinghouse ou serveur de base de données et qui constitue, pour nous, la référence [Ber.-Jos. 85]. Nous ne poursuivrons donc pas l'étude de cette politique qui, par ailleurs, nous apparaît plus complexe à mettre en oeuvre.

2.2. Les critères d'évaluation.

2.2.1. Identification des critères.

Les critères qui nous permettront d'évaluer les différentes politiques, de situer leurs avantages et leurs points faibles, peuvent être répartis en deux groupes : les critères fonctionnels et les critères de performance. Les critères fonctionnels représentent les contraintes que l'on pourra exiger du système, les fonctions qu'il devrait être en mesure de remplir, d'une façon ou d'une autre.

Les critères de performance représentent les contraintes que le système fournira en retour, pour la prise en compte des critères fonctionnels.

2.2.2. Les critères fonctionnels.

Nous pouvons retenir principalement quatre types de critères fonctionnels, étant donné leur lien avec les fonctions-même du système.

* l'accès aux documents, leur création, leur envoi; leurs consultation et suppression; soit les fonctions essentielles d'un courrier électronique;
* la modification des documents et la gestion des mises à jour qui en découle pour conserver la cohérence de la base de données;
* la notification d'un nouveau document, c'est-à-dire le fait de prévenir un abonné qu'un document lui a été envoyé;
* la confidentialité du système : protection de l'accès au courrier électronique, protection des documents dans les bases de données et lors de leur transfert;
* la localisation des abonnés du courrier électronique sur un site spécifique ou l'accès libre à tout site.

Reprenons chacun de ces critères séparément :

a) l'accès aux documents du courrier électronique regroupe différentes fonctions relatives à la manipulation des documents, tant par le créateur du document que par le ou les destinataires de celui-ci.

* la création du document : la constitution du corps même du document, la spécification des destinataires, l'identification du document, l'intitulé du document,...

* l'envoi d'un document : la prise en charge du document, sa propagation vers le(s) destinataire(s) son stockage, sa duplication,...

* la consultation d'un document par le(s) destinataire(s); si elle est essentielle, par contre, la consultation d'un document par son créateur peut être ou inutile ou primordiale : cela dépend de la nature du document (note personnelle, note d'information à l'intention d'autres personnes,...).

* la suppression d'un document : suppression logique pour un seul abonné, suppression physique, récupération possible d'un document jeté,...

b) la modification des documents : certains documents peuvent être sujets à des modifications à différents moments de leur vie. De ce fait, choisir de dupliquer un document pour en faciliter la consultation peut amener des problèmes lors de possibles modifications. Il faudra faire ici la distinction entre la création d'un nouveau document à partir d'un ancien ou la modification d'un document, ce qui imposerait la propagation de la modification à toutes les copies du document original. Un autre problème qui doit être résolu à ce niveau est celui de la concurrence des accès en modification pour un même document.

c) la notification d'un nouveau document peut comprendre deux points de vue :

* la notification directe : lorsque l'abonné travaille à son poste, le courrier peut l'informer directement de l'arrivée d'un nouveau document;

* la notification indirecte : l'abonné demande explicitement la consultation de sa boîte aux lettres.

d) la confidentialité du courrier électronique peut être abordée à plusieurs niveaux :

* l'accès au courrier par des personnes qui n'y sont pas autorisées, problème qu'il est possible de résoudre par l'utilisation de mots de passe, par exemple;

* la violation du contenu des documents par des personnes étrangères au système, ce qui pose le problème de la protection des fichiers, des directories,... On peut également se poser la question du degré de violation de la base de données, ce qui correspond aux polémiques "centralisation - décentralisation" et "unicité - multiplicité" des documents. Nous savons par ailleurs qu'en cette matière, un seul système est plus facile à protéger qu'une série de systèmes;

* l'attribution d'un document à un faux destinataire ou le fait qu'un abonné puisse accéder à un document dont il n'est pas destinataire. Ce problème relève à la fois d'un routage correct des documents envoyés et d'une limitation des documents disponibles à la consultation;

* l'interception des documents lors de leur transmission, à laquelle il est possible de répondre notamment par des méthodes d'encryptage.

e) la localisation des abonnés : la possibilité d'accéder aux documents doit exister à tout poste du système. Dans ce cadre, deux optiques sont envisageables :

* les abonnés se présentent toujours au même poste, ou, tout du moins, au même réseau : il y a localisation. Dès lors, il est possible de destiner directement le document au poste ou au réseau local où se situe l'abonné.

* les abonnés bougent et ne se présentent pas toujours au même poste (ou réseau) : il n'y a pas de localisation. Dès lors on pensera plus facilement à des méthodes où les documents sont directement propagés à travers tout le système, ou d'autres où ils sont conservés en un seul endroit.

2.2.3. Les critères de performance.

En réponse aux critères fonctionnels définis ci-dessus et au choix d'une politique pour y répondre, le système fera preuve d'une certaine performance traduite au moyen de plusieurs critères. Nous retiendrons principalement quatre critères dont deux peuvent être vus comme des critères de temps de réponse et les deux autres, comme des critères de coût du système:

* temps de réponse : temps nécessaire à l'expédition et à la consultation d'un document;
* coût du système : le volume mémoire utilisé et le volume du trafic via DCS (documents passés d'un site à un autre).

Reprenons chaque critère séparément :

a) le temps nécessaire à l'expédition d'un document. Cette donnée peut être interprétée de deux façons : le temps qui est soustrait au travail possible de l'abonné (le temps écoulé entre le moment où l'abonné demande l'expédition du document et le moment ou il peut demander l'exécution d'une autre opération) et le temps effectivement nécessaire à l'acheminement d'un document à son ou ses destinataires. Notons que le temps soustrait à l'utilisateur peut être réduit au minimum si l'on

envisage l'expédition d'un document parallèlement à d'autres requêtes de l'utilisateur;

b) le temps nécessaire à la consultation d'un document. Il s'agira ici du temps nécessaire à l'acheminement du document, de l'endroit (le plus proche) où il est stocké à l'abonné qui veut le consulter. Ici, suivant les politiques, on pourra retrouver le temps d'expédition du document, décrit en a) ci-dessus (temps d'acheminement);

c) le volume mémoire utilisé : cette donnée est directement liée à la quantité de documents (copies) diffusés dans le système. Même si cet aspect n'est plus très significatif à l'heure actuelle, vu le (relativement) faible coût des moyens de stockage, un courrier électronique à grande échelle pourrait reposer le problème de manière plus aigüe;

d) le volume du trafic via DCS est une donnée relative au nombre de communications nécessaires pour le transfert des documents et au volume de ces transferts; en effet, il nous faudra considérer, de ce point de vue, à la fois les ouvertures de lignes (créations de connexions) et les quantités de données transmises (nombre de paquets) puisque DCS applique des tarifications à ces deux niveaux.

Nous pouvons, pour terminer ce tour des définitions de critères, rappeler, une fois encore, l'interaction entre les deux classes de critères.
Les critères fonctionnels vont situer les fonctions à faire remplir par le courrier électronique. Suite à la définition de ces fonctions, plusieurs politiques peuvent être envisagées, que l'on voit (intuitivement) facilement réalisables. Parmi les politiques choisies (ou parmi toutes les politiques,si on ne désire en écarter aucune a priori), on en conservera une dont les performances sont les plus satisfaisantes. Notons, par ailleurs, que pour deux situations identiques, deux personnes différentes pourraient choisir des politiques différentes. Rien

n'empêche, en effet, d'estimer de façons différentes le niveau de satisfaction d'un même critère.

Donc, dans un premier temps, nous nous attarderons sur une étude qualitative des différentes politiques face aux différents critères mentionnés précédemment; et, dans un deuxième temps, nous aborderons une étude quantitative qui portera sur les réactions des politiques aux seuls critères de performance. Ces deux approches nous permettant de formuler le choix d'une politique bien adaptée aux spécifications définies.

Chapitre 3 : Evaluation qualitative des politiques.

Nous allons envisager, maintenant, les implications des différentes politiques du point de vue des critères que nous avons décrits au chapitre précédent.

Il est important de signaler que l'évaluation que nous faisons à ce stade ne peut être que théorique. Une évaluation quantitative viendra affiner celle-ci par après.

3.1. Politique 1: une seule copie stockée dans le serveur du réseau de création/expédition du document.

a) L'accès au document doit être étudié en deux phases. D'une part, la création et l'envoi d'un document sont simplifiés par le fait que le serveur de base de données doit simplement stocker ce document. D'autre part, la consultation et la suppression d'un document sont plus complexes si la requête émane d'un réseau différent de celui où est stocké le document;

b) La modification des documents, si elle est autorisée, n'entraîne aucune propagation des mises à jour, mais seulement une gestion de la concurrence (par exemple si deux abonnés désirent modifier le même document en même temps);

c) La notification directe d'un nouveau document est difficile à réaliser étant donné qu'aucun renseignement n'arrive aux réseaux destinataires lors de l'expédition. Il reste toujours la notification indirecte par consultation de boîte aux lettres qui nécessite des transferts d'informations par DCS;

d) La confidentialité du courrier électronique est assurée jusqu'à un certain point par le fait qu'un attaquant qui parviendrait à accéder à un serveur de base de données ne disposerait ainsi que d'une partie des documents du courrier. En outre, il faut protéger plusieurs serveurs qui constituent chacun un point d'attaque possible;

e) La localisation des abonnés n'est pas obligatoire. Elle peut exister ou non sans influencer la complexité du système;

f) Le temps d'expédition d'un document ne tient compte que du stockage de celui-ci dans le serveur de base de données du réseau local. Il n'y a aucun passage par DCS à ce moment;

g) Le temps de consultation d'un document est fonction de la localisation de ce document. Si celui-ci est stocké dans le serveur du réseau local qui traite la demande de consultation, le temps sera très court. Sinon, il faut tenir compte de deux passages par DCS pour transmettre la demande et le document en retour;

h) Le volume mémoire utilisé pour le stockage est le minimum possible, puisqu'on n'autorise qu'une seule copie par document. De plus, ce volume est réparti entre les différents serveurs;

i) Le trafic DCS est élevé lors de consultations de documents; en effet, chaque consultation exige deux passages par DCS (cfr point f ci-dessus). De plus, il y a un trafic superflu lorsque plusieurs destinataires sont situés sur un même réseau local éloigné, car, dans ce cas, on transférera plusieurs fois le même document entre les mêmes réseaux;

En résumé, le plus gros obstacle fonctionnel de la politique 1 est la notification directe et les avantages fonctionnels les plus importants sont la modification possible des documents et la délocalisation des abonnés.

3.2. Politique 2: plusieurs copies stockées dans les serveurs des réseaux destinataires.

a) L'accès au document est simplifié en consultation et en suppression parce que ces fonctions ne concernent que le serveur de base de données du destinataire. L'envoi, par contre, est plus complexe à traiter, car il nécessite le routage des copies vers les réseaux destinataires. De plus, la consultation d'un document par son créateur est problématique s'il n'y a pas de destinataire dans le même réseau local;

b) La modification des documents, si elle est autorisée, entraîne une propagation des mises à jour en plus d' une gestion de la concurrence;

c) La notification directe d'un nouveau document est relativement facile à réaliser parce qu'une copie d'un nouveau document est toujours envoyée jusqu'aux serveurs des réseaux destinataires. La notification indirecte, très simple, reste possible;

d) La confidentialité du courrier électronique est, comme dans la politique 1, assurée jusqu'à un certain point par le fait qu'un attaquant qui parviendrait à accéder à un serveur de base de données ne disposerait ainsi que d'une partie des documents du courrier, cette partie étant tout de même plus importante. En outre, il faut protéger plusieurs serveurs qui constituent chacun un point d'attaque possible;

e) La localisation des abonnés est obligatoire. Il n'est pas possible de réaliser cette politique dans une optique de délocalisation des abonnés;

f) Le temps d'expédition d'un document tient compte de la complexité des mécanismes de routage

nécessaires à l'envoi des copies vers les réseaux destinataires;

g) Le temps de consultation d'un document est très court, puisqu'une copie du document est toujours stockée dans le serveur du réseau local destinataire;

h) Le volume mémoire utilisé pour le stockage est plus important; il est cependant optimisé, puisqu'on se limite à autant de copies que de réseaux destinataires pour un document. Ce volume est réparti entre les différents serveurs;

i) Le trafic DCS est élevé lors d'expéditions de documents; en effet, chaque expédition entraîne autant de passages par DCS qu'il y a de réseaux destinataires pour le document envoyé;

En résumé, les plus gros obstacles fonctionnels de la politique 2 sont la modification des documents, la localisation des abonnés et la consultation des documents par leurs créateurs. Les avantages fonctionnels les plus importants sont la consultation des documents et la notification directe.

3.3. Politique 3: plusieurs copies stockées dans les serveurs des réseaux destinataires et du réseau d'expédition.

a) L'accès au document est simplifié en consultation et en suppression parce que ces fonctions ne concernent que le serveur de base de données du destinataire. L'envoi, par contre, est plus complexe à traiter, car il nécessite le routage des copies vers les réseaux destinataires et le stockage dans le serveur du réseau d'expédition;

Comme dans la politique 2:

b) La modification des documents, si elle est autorisée, entraîne une propagation des mises à jour en plus d' une gestion de la concurrence;

c) La notification directe d'un nouveau document est relativement facile à réaliser parce qu'une copie d'un nouveau document est toujours envoyée jusqu'aux serveurs des réseaux destinataires. La notification indirecte, très simple, reste possible;

d) La confidentialité du courrier électronique est assurée jusqu'à un certain point par le fait qu'un attaquant qui parviendrait à accéder à un serveur de base de données ne disposerait ainsi que d'une partie des documents du courrier, cette partie étant tout de même importante. En outre, il faut protéger plusieurs serveurs qui constituent chacun un point d'attaque possible;

e) La localisation des abonnés est obligatoire. Il n'est pas possible de réaliser cette politique dans une optique de délocalisation des abonnés;

f) Le temps d'expédition d'un document tient compte de la complexité des mécanismes de routage nécessaires à l'envoi des copies vers les réseaux destinataires et au stockage dans le serveur du réseau d'expédition;

g) Le temps de consultation d'un document est très court, puisqu'une copie du document est toujours stockée dans le serveur du réseau local destinataire ou expéditeur;

h) Le volume mémoire utilisé pour le stockage est plus important; il est cependant optimisé, puisqu'on se limite à autant de copies que de réseaux destinataires et expéditeur pour un document. Ce volume est réparti entre les différents serveurs;

i) Le trafic DCS est élevé lors d'expéditions de documents; en effet, chaque expédition entraîne autant de passages par DCS qu'il y a de réseaux destinataires pour le document envoyé;

En résumé, les plus gros obstacles fonctionnels de la politique 3 sont la modification des documents et la localisation des abonnés. Les avantages fonctionnels les plus importants sont la consultation des documents et la notification directe.

3.4. Politique 4: plusieurs copies stockées dans tous les serveurs.

a) L'accès au document est simplifié en consultation parce que cette fonction ne concerne que le serveur de base de données du destinataire. La suppression doit être propagée à travers tout le système. L'envoi d'un document est également assez simple, quoique fastidieux puisqu'il nécessite l'envoi de copies vers tous les réseaux du système et le stockage dans le serveur du réseau d'expédition;

Comme dans les politiques 2 et 3 :

b) La modification des documents, si elle est autorisée, entraîne une propagation des mises à jour en plus d' une gestion de la concurrence;

c) La notification directe d'un nouveau document est relativement facile à réaliser parce qu'une copie d'un nouveau document est toujours envoyée jusqu'aux serveurs de tous les réseaux et donc, en particulier, des réseaux destinataires. La notification indirecte, très simple, reste possible;

Par contre :

d) La confidentialité du courrier électronique n'est plus assurée du tout si un attaquant parvient

à accéder à un serveur de base de données; il dispose ainsi de la totalité des documents du courrier. En outre, il faut protéger tous les serveurs du système qui constituent chacun un point d'attaque possible;

e) La localisation des abonnés n'est nullement obligatoire. Il est tout à fait envisageable de réaliser cette politique dans une optique de délocalisation des abonnés;

f) Le temps d'expédition d'un document ne tient compte que de la nécessité d'envoyer une copie vers chaque réseau du système et du stockage dans le serveur du réseau d'expédition;

g) Le temps de consultation d'un document est très court, puisqu'une copie du document est toujours stockée dans le serveur du réseau local;

h) Le volume mémoire utilisé pour le stockage est maximum;

i) Le trafic DCS est élevé lors d'expéditions de documents; en effet, chaque expédition entraîne autant de passages par DCS qu'il y a de réseaux dans le système;

En résumé, le plus gros obstacle fonctionnel de la politique 4 est la modification des documents et les avantages fonctionnels les plus importants sont la consultation des documents, la délocalisation des abonnés et la notification directe.

3.5. Politique 5 : une seule copie stockée dans le serveur d'un réseau local de référence.

a) L'accès au document, tant pour la création et l'envoi que pour la consultation et la suppression, est plus complexe en général, c'est-à-dire si la requête émane d'un réseau différent du réseau local de référence;

b) La modification des documents, si elle est autorisée, n'entraîne aucune propagation des mises à jour, mais seulement une gestion de la concurrence;

c) La notification directe d'un nouveau document est difficile à réaliser étant donné qu'aucun renseignement n'arrive aux réseaux destinataires lors de l'expédition. Il reste toujours la notification indirecte par consultation de boîte aux lettres qui nécessite un transfert d'informations par DCS;

d) La confidentialité du courrier électronique n'est pas assurée du tout si un attaquant parvient à accéder au serveur du réseau de référence; il dispose ainsi de la totalité des documents du courrier. Par contre, il suffit de protéger le serveur du réseau de référence qui constitue le seul point d'attaque possible;

e) La localisation des abonnés n'est pas obligatoire. Elle peut exister ou non sans influencer la complexité du système;

f) Le temps d'expédition d'un document tient compte de l'envoi du document par DCS (si l'on ne se trouve pas sur le réseau de référence) et de son stockage dans le serveur du réseau local de référence;

g) Le temps de consultation d'un document est fonction du réseau local d'où provient la demande : s'il s'agit du réseau de référence, le temps de consultation sera très court; sinon, il faut tenir compte de deux passages par DCS pour transmettre la demande et le document en retour, ce qui est le cas général;

h) Le volume mémoire utilisé pour le stockage est le minimum possible, puisqu'on n'autorise qu'une

seule copie par document. Mais, contrairement à la politique 1, ce volume est centralisé;

i) Le trafic DCS est élevé lors d'expéditions et de consultations de documents; en effet, chacun de ces traitements exige deux passages par DCS;

En résumé, les plus gros obstacles fonctionnels de la politique 5 sont la consultation des documents et la notification directe. Les avantages fonctionnels les plus importants sont la modification des documents et la délocalisation des abonnés.

3.6. Politique 6 : une seule copie stockée dans un serveur indépendant de tout réseau local.

Comme dans la politique 5 :

a) L'accès au document, tant pour la création et l'envoi que pour la consultation et la suppression, est complexe parce qu'il faut chaque fois passer par DCS;

b) La modification des documents, si elle est autorisée, n'entraîne aucune propagation des mises à jour, mais seulement une gestion de la concurrence;

c) La notification directe d'un nouveau document est difficile à réaliser étant donné qu'aucun renseignement n'arrive aux réseaux destinataires lors de l'expédition. Il reste toujours la notification indirecte par consultation de boîte aux lettres qui nécessite un transfert d'informations par DCS;

d) La confidentialité du courrier électronique n'est pas assurée du tout si un attaquant parvient à accéder au serveur indépendant; il dispose ainsi de la totalité des documents du courrier. Par contre, il suffit de protéger le serveur

indépendant qui constitue le seul point d'attaque possible;

e) La localisation des abonnés n'est pas obligatoire. Elle peut exister ou non sans influencer la complexité du système;

f) Le temps d'expédition d'un document tient compte de l'envoi du document par DCS et de son stockage dans le serveur indépendant;

g) Le temps de consultation d'un document tient compte de deux passages par DCS pour transmettre la demande et le document en retour;

h) Le volume mémoire utilisé pour le stockage est le minimum possible, puisqu'on n'autorise qu'une seule copie par document. Ici aussi les documents sont centralisés;

i) Le trafic DCS est élevé lors d'expéditions et de consultations de documents; en effet, chacun de ces traitements exige deux passages par DCS;

En résumé, les plus gros obstacles fonctionnels de la politique 6 sont la consultation des documents et la notification directe. Les avantages fonctionnels les plus importants sont la modification des documents et la délocalisation des abonnés.

3.7. Résumé.

En guise de conclusion à ce chapitre, nous allons synthétiser les différents aspects qui viennent d'être présentés; nous regroupons, à la fois, critères et politiques dans un tableau synoptique ou nous faisons également apparaître des appréciations.

Ces appréciations sont les suivantes :

* on attribuera la cote TB à des politiques qui répondent parfaitement au critère donné;

* la cote B, à des politiques qui répondent bien au critère donné en empruntant toutefois quelques détours par rapport à une solution parfaite;
* la cote M à des politiques qui répondent assez mal au critère donné et
* la cote TM aux politiques dont la solution offerte au critère donné ne pourrait être pire : il s'agira aussi de politiques ne pouvant pas offrir de solution acceptable à ce critère.

	POL 1	POL 2	POL 3	POL 4	POL 5	POL 6
ACCES						
Expédition	TB	M	M	M	B	B
Consultation	M/B	TB	TB	TB	B/TB	B
Consult/créa	TB	TM	TB	TB	B/TB	B
Suppression	B	TB	TB	M	B	B
MODIFICATION	TB	TM	TM	TM	TB	TB
NOTIFICATION						
Automatique	TM	TB	TB	TB	M	M
Sur demande	M	TB	TB	TB	B	B
CONFIDENTIAL.	M/B	M	M	TM	B	TB
LOCALISATION	TB	TM	TM	TB	TB	TB

	POL 1	POL 2	POL 3	POL 4	POL 5	POL 6
TEMPS EXPED.	TB	M	M	TM	B/TB	B
TEMPS CONSULT	M/TB	TB	TB	TB	M/TB	M
VOL MEMOIRE	TB	M	M	TM	TB	TB
TRAFIC DCS	M	B	B	B	M	M

Fig. 3.1 : Tableau synoptique d'évaluation

3.8. D'autres critères.

Outre les critères que nous avons pris en considération, il est possible de tenir compte de critères supplémentaires [Mac.-Gui. 83 p.413]. Ainsi, on peut envisager la stabilité du système, son extensibilité, sa vulnérabilité, le coût de son installation, par exemple. Ce sont ces critères que nos allons brièvement aborder à présent.

A. *La stabilité du système*, c'est-à-dire sa capacité à absorber les pointes d'activité ou de trafic, peut reposer sur deux constatations : tout d'abord, "moins une activité est élevée, moins fortes (dans l'absolu) sont les pointes" et "plus l'activité est répartie sur divers acteurs, moins il y a de risques de surcharges concentrées".

Il est très difficile d'évaluer l'activité-même des abonnés sur un réseau donné; c'est pourquoi nous ne pouvons pas comparer les activités dans les différentes politiques. Néanmoins, il est possible de tenir compte des échanges réalisés entre les réseaux d'un système. En effet, les politiques réagissent différemment au niveau de la quantité d'échanges réalisés, certaines imposant un très fort degré de communication (pol 5) et d'autres un degré nettement moindre (pol 3). Donc, un faible taux d'échanges a pour conséquence de limiter les accroissements de trafic, quand le nombre d'abonnés travaillant au même moment augmente, tandis qu'un taux plus élevé infue directement sur le trafic, quitte à susciter des surcharges, locales ou globales.

La répartition du trafic sur divers serveurs peut influencer le type et l'importance des surcharges; un trafic (et donc une activité) orienté vers un seul et même acteur a pour effet de surcharger celui-ci de tout accroissement pouvant survenir dans le système. De même, un trafic (activité) réparti entre plusieurs serveurs peut être également très gênant si les échanges entre ces serveurs sont nombreux, car une pointe d'activité sur un réseau se répercute directement sur les autres réseaux.

Par contre, des serveurs qui communiquent peu entre eux sont moins capables de transporter un accroissement local d'activité vers les autres réseaux du système. Il faut donc pouvoir différencier les surcharges locales, gênantes pour les abonnés locaux, et les surcharges globales que tous les abonnés doivent subir et qui sont donc nettement plus graves pour la stabilité du système. Parmi les politiques offrant les risques d'une surcharge globale, on retrouve principalement les politiques 5 et 1, pour lequel le trafic est très élevé et lié à la plupart des opérations du courrier électronique. D'un autre côté, on trouve la politique 3 qui ne nécessite que peu de transferts, en outre concentrés sur un seul type d'opérations du courrier électronique.

B. *La vulnérabilité du système*, c'est-à-dire sa résistance lorsqu'un de ses éléments tombe en panne, peut directement se greffer sur le point précédent. La vulnérabilité d'un système comporte deux aspects : la centralisation du système et le niveau de dépendance des différents éléments entre eux. Un système centralisé est totalement mort si l'élément central a une défaillance : dans la politique 5, un tel événement aurait pour effet de rendre le système totalement inutilisable. Un système à forte dépendance (c'est-à-dire où la plupart des opérations nécessitent des échanges) permet la survie des éléments non-défaillants, tout en rendant impossible une utilisation complète des opérations du système : la politique 1 permettrait par exemple d'avoir accès à certains documents de la base de données, en rendant l'accès à d'autres tout à fait impossible. Un système à faible dépendance permet par contre la poursuite quasi normale des activités : les politiques 3 et 4 rendent possible l'accès à tous les documents du système.

Notons néanmoins que, contrairement à la politique 1 qui limite l'accès aux documents et limite les manipulations aux seuls documents accessibles, les politiques 3 et 4 risquent d'entraîner certaines incohérences dans le système, en permettant de travailler

sur des documents qui sont physiquement inaccessibles (suppression de document dans la politique 4, envoi de document dans les pol. 3 et 4).

C. *L'extensibilité du système* peut être vue comme la capacité du système à accepter de nouveaux abonnés, donc un nombre plus élevé de documents, et une augmentation des traitements demandés. Du point de vue du nombre de documents, c'est-à-dire le volume de stockage, il est plus facile d'adapter les capacités localement que globalement car cette adaptation peut refléter beaucoup mieux les besoins réels et donc être quasi optimale. Par ailleurs, toutes les politiques ne nécessitent pas le même volume supplémentaire par abonné : des politiques à copies multiples sont à ce niveau nettement plus exigeantes et donc beaucoup plus coûteuses. Du point de vue du nombre de requêtes à exécuter, là aussi une adaptation locale peut être plus facile, et nettement moins coûteuse qu'une adaptation globlale. En effet, un groupe de "petits CPUs" tend à revenir moins cher qu'un "gros CPU". (cela dépend bien sûr aussi du nombre de petits CPUs nécessaires !)

D. *Le coût de l'installation initiale du système*, c'est-à-dire le coût des CPUs, des liaisons, des volumes mémoire,.... Au niveau des liaisons, nous avons déjà vu qu'un système centralisé revenait moins cher lors de l'installation. Au niveau des CPUs, la tendance actuelle est à l'abaissement global des prix pour des petites machines, de sorte qu'il revient moins cher d'acquérir un ensemble de machines plus petites qu'une seule machine plus grosse; il en va partiellement de même des volumes de stockage car, si le volume de stockage coûte, les périphériques coûtent aussi, et ce serait donc un coût à supporter à chaque réseau du système. Néanmoins, ce critère demande une étude plus détaillée des besoins nécessaires pour chaque situation et des solutions possibles à y apporter.

Remettons ces critères dans un tabeau synoptique : (cotes favorables en gras; cotes défavorables en italique)

	Pol 1	Pol 3	Pol 4	Pol 5
Stabilité	*faible*	**forte**	moyenne	*faible*
Vulnérabilité	moyenne	**faible**	**faible**	*forte*
Cohérence	**forte**	*faible*	*faible*	**forte**
Extensibilité	**forte**	moyenne	moyenne	moyenne
Coût d'instal.	à évaluer concrètement			
total cotes	- / ++	- / ++	- / +	-- / +

Il est donc possible de voir que les critères complémentaires laissent les politiques sur un pied d'égalité : il faudra tenir compte plus précisément d'un ou plusieurs critères pour pouvoir fournir la solution la plus appropriée au problème que l'on s'est posé.

Chapitre 4 : Les traitements nécessaires.

Nous allons étudier à présent les traitements nécessaires pour la mise en oeuvre des fonctions du courrier électronique dans le cadre des différentes politiques. Ces fonctions, comme nous l'avons déjà dit, sont les suivantes : création, expédition, consultation et suppression d'un document. Notons, néanmoins, que tout le travail de création d'un document peut être réalisé localement pour toute politique; nous pourrons donc oublier cette fonction lors de notre étude. Rappelons aussi que la consultation d'un document exigera, dans notre CE, la consultation préalable d'une boîte aux lettres; c'est pourquoi nous étudierons également cette (sous-)fonction.

En ce qui concerne les politiques sujettes à cette étude, nous en conservons quatre qui nous paraissent significatives du point de vue des traitements à effectuer ; il s'agit des politiques 1 (stockage chez le créateur), 3 (stockage chez les destinataires et le créateur), 4 (stockage dans tout le système) et 5 (stockage dans un réseau de référence du système). Les politiques 2 (stockage chez les destinataires seuls) et 6 (stockage dans un réseau de référence externe) peuvent être considérées, de ce point de vue, comme des cas particuliers des politiques 3 et 5 respectivement.

Dans un premier temps, nous donnerons une approche intuitive des fonctions du courrier électonique dans les differentes politiques. Ensuite, nous entrerons plus en détail dans le processus de traitement en décrivant le diagramme de flux de chaque fonction, et ce, pour chaque politique.

4.1. Approche intuitive.

Cette approche se base sur différents symboles (fig. 4.1) que nous pouvons définir comme suit (ou que nous avons déjà défini dans l'introduction) :

* le *poste de travail*; nous ferons la différence ici entre le poste de travail du créateur du document, ceux des destinataires du document et les autres postes de travail du systeme;
* le *réseau local Ethernet*;
* le *réseau public DCS*;
* le *serveur local* est composé de deux parties : un serveur de base de données et un interface Ethernet-DCS (ou passerelle). Le serveur de base de données s'occupe de tous les documents à gérer localement et la passerelle, des communications d'un réseau à l'autre; celle-ci devra donc se situer physiquement entre les deux réseaux Ethernet et DCS.
* le *document*;
* le *fichier de liens* est le témoin des relations (de destinataire) qui existent entre les abonnés et les documents. Notons qu'un fichier de liens peut rendre compte d'une partie seulement des relations, pour tous les abonnés d'un réseau local ou pour tous les documents crées sur ce réseau local, par exemple;
* la *boîte aux lettres*;
* les *demandes de consultation* de boîte aux lettres et de document servent à la communication entre gérants de base de données éloignés;
* la *demande de suppression* de document est également utilisée dans la communication entre serveurs;

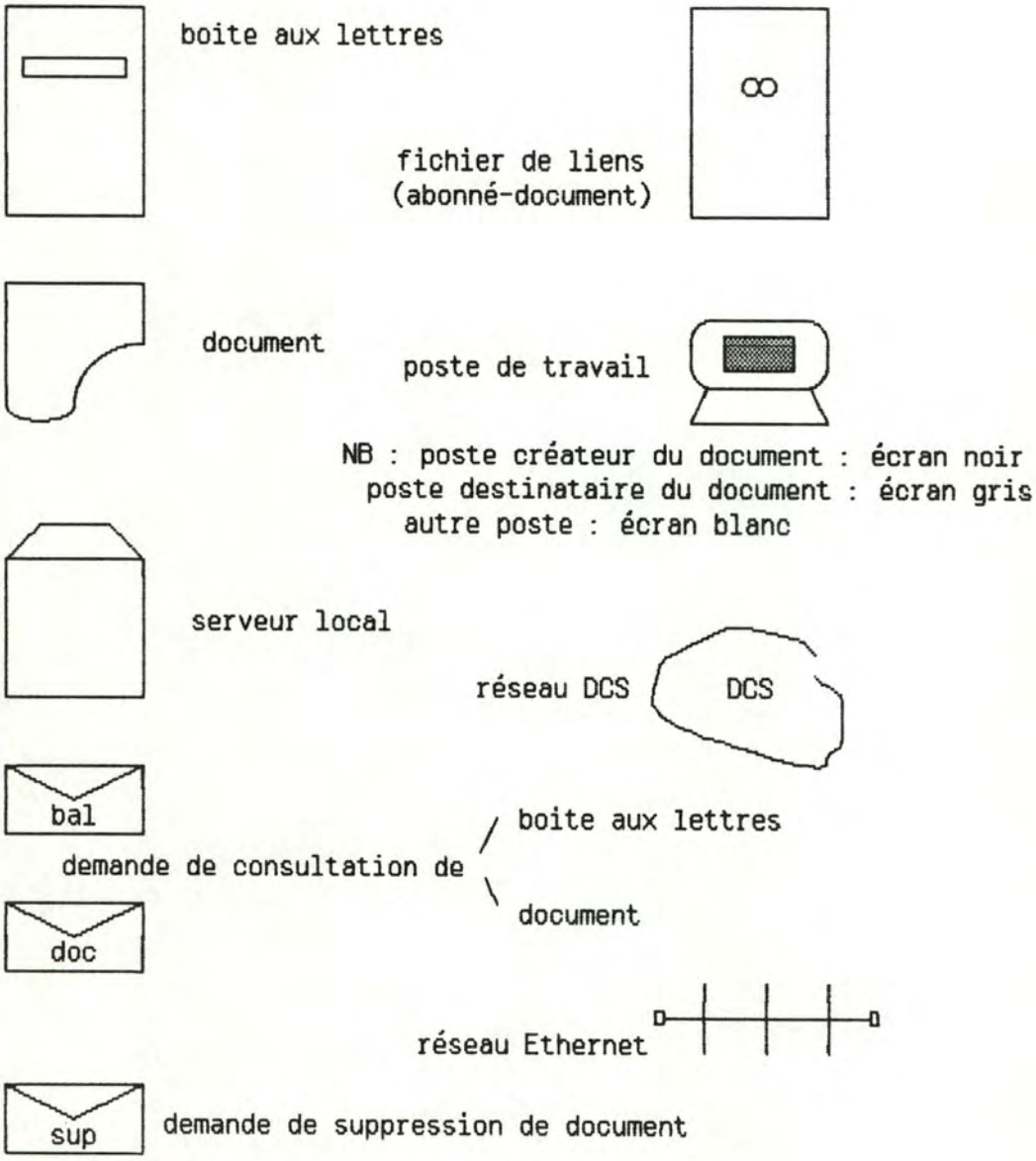

Fig. 4.1 : Légende

4.1.1. L'envoi de documents:

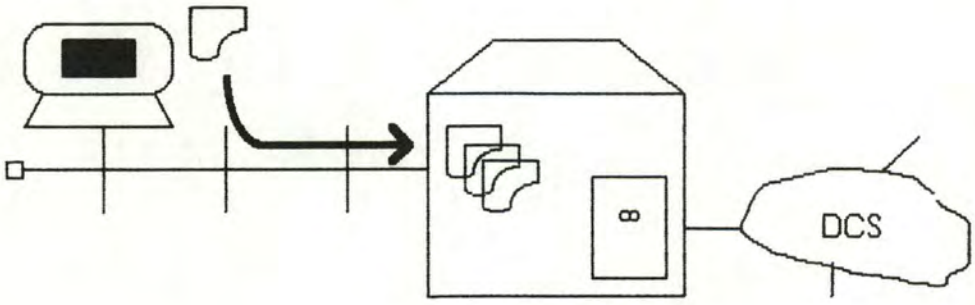

Fig. 4.2 : Envoi dans la politique 1

* pour la politique 1 (fig. 4.2) : comme cette politique est definie par le stockage du document dans le serveur du réseau de création, l'envoi du document consiste uniquement à le stocker au niveau du serveur local et d'y créer les liens nécessaires.

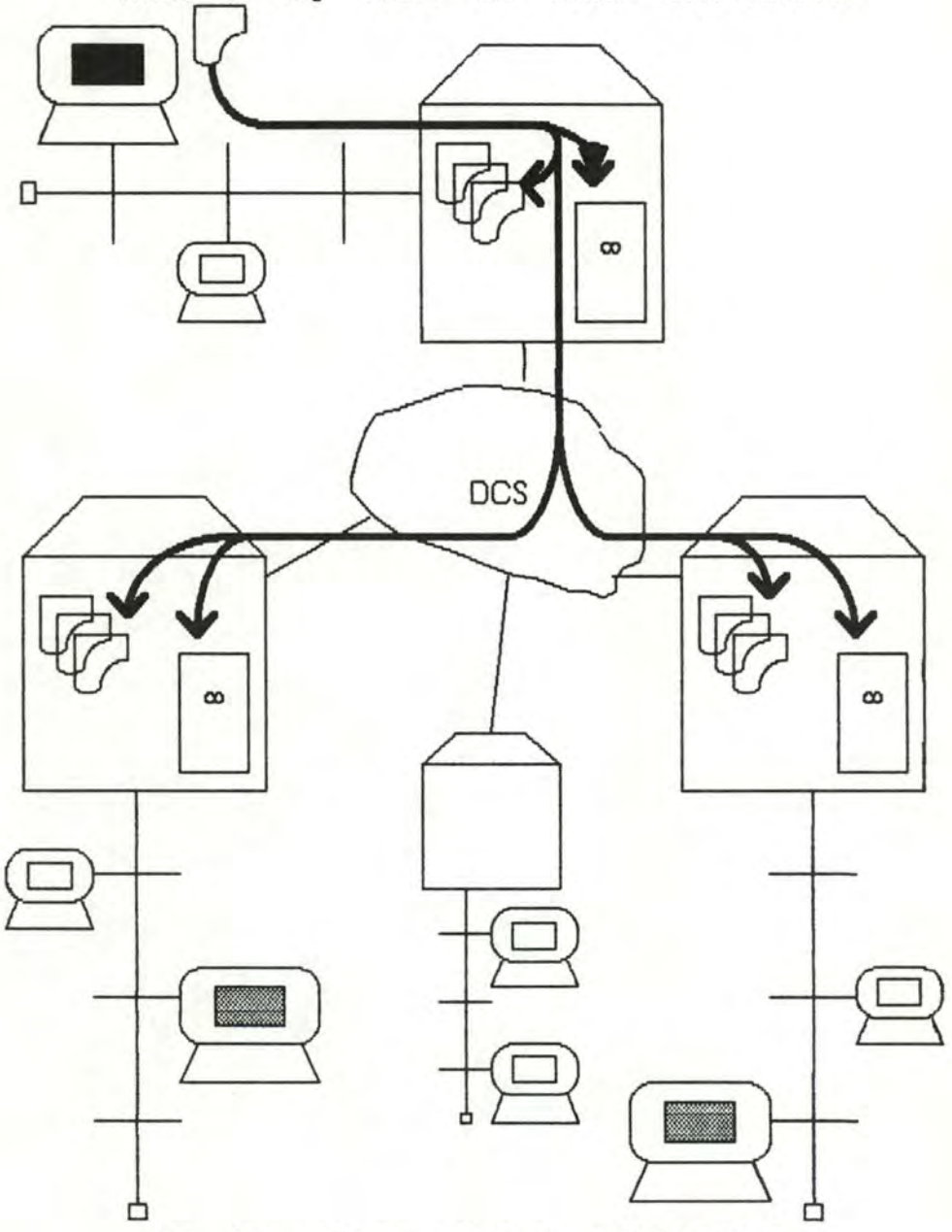

Fig. 4.3 : Envoi dans la politique 3

* pour la politique 3 (fig. 4.3) : il faut ici envoyer le document vers tous les réseaux où se situe au moins un destinataire; dans chacun de ces réseaux, le document devra être stocké et les liens nécessaires, créés.

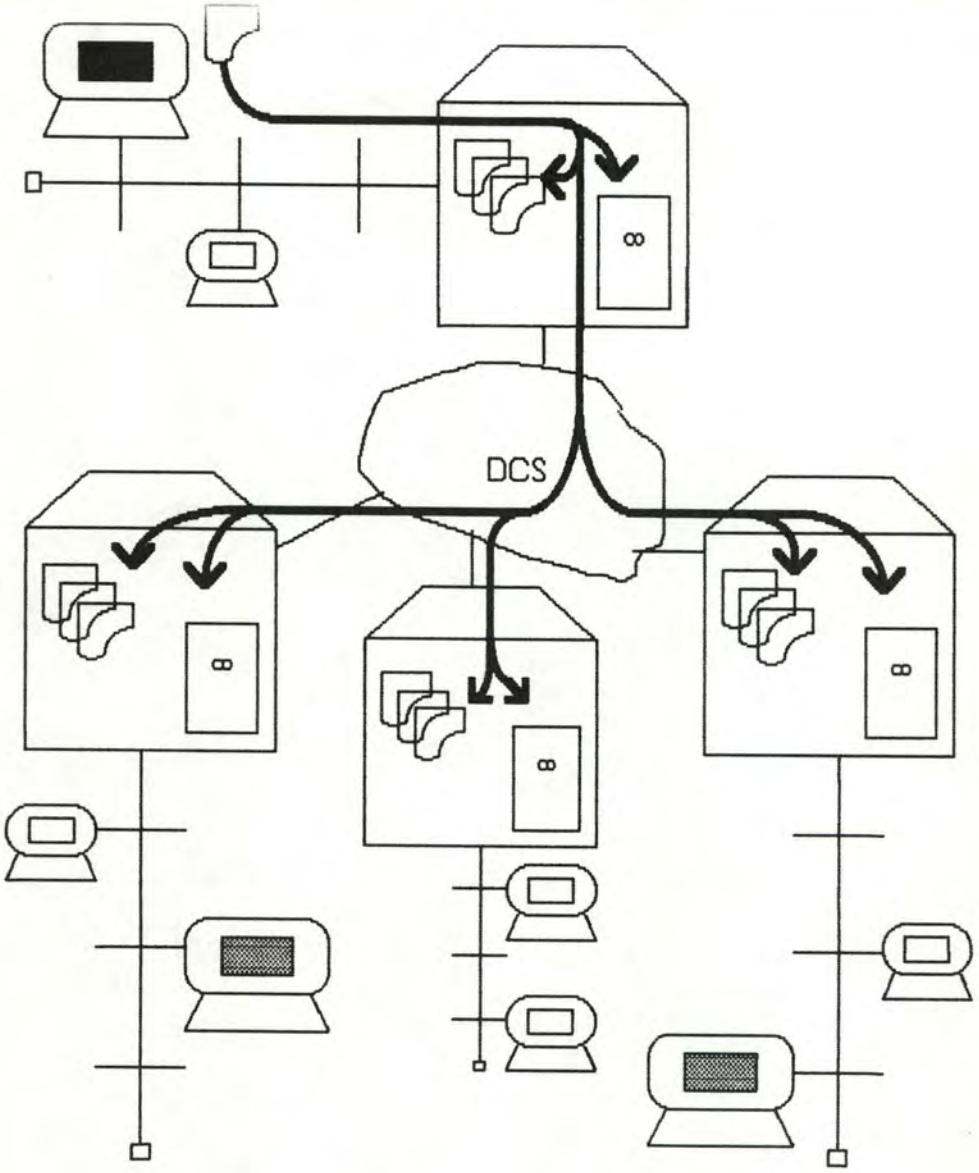

Fig. 4.4 : Envoi dans la politique 4

* pour la politique 4 (fig. 4.4) : cette politique exige la propagation du document à travers tout le système; dans tous les réseaux locaux du système, le document devra être stocké et les liens nécessaires, créés.

* pour la politique 5 (fig. 4.5) : le document est envoyé vers le réseau de référence où il est stocké et où les liens nécessaires sont créés.

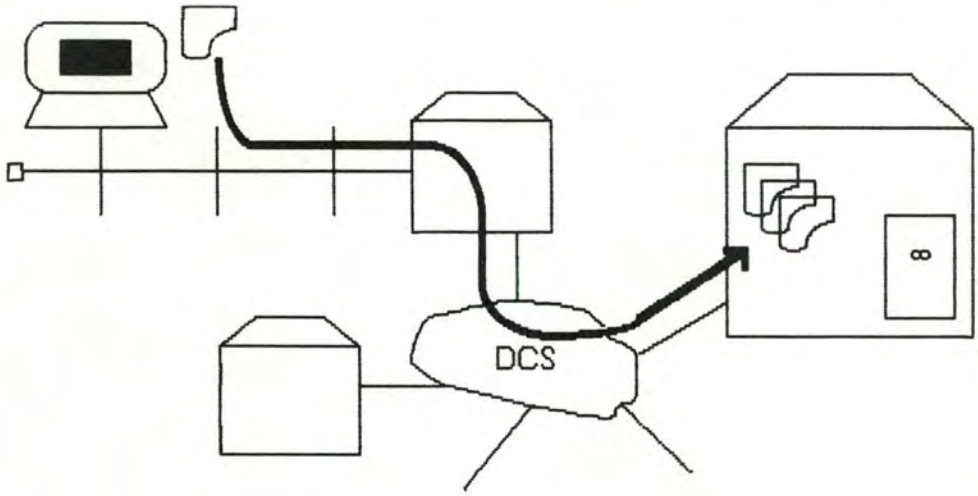

Fig. 4.5 : Envoi dans la politique 5

4.1.2. La consultation de boîte aux lettres:

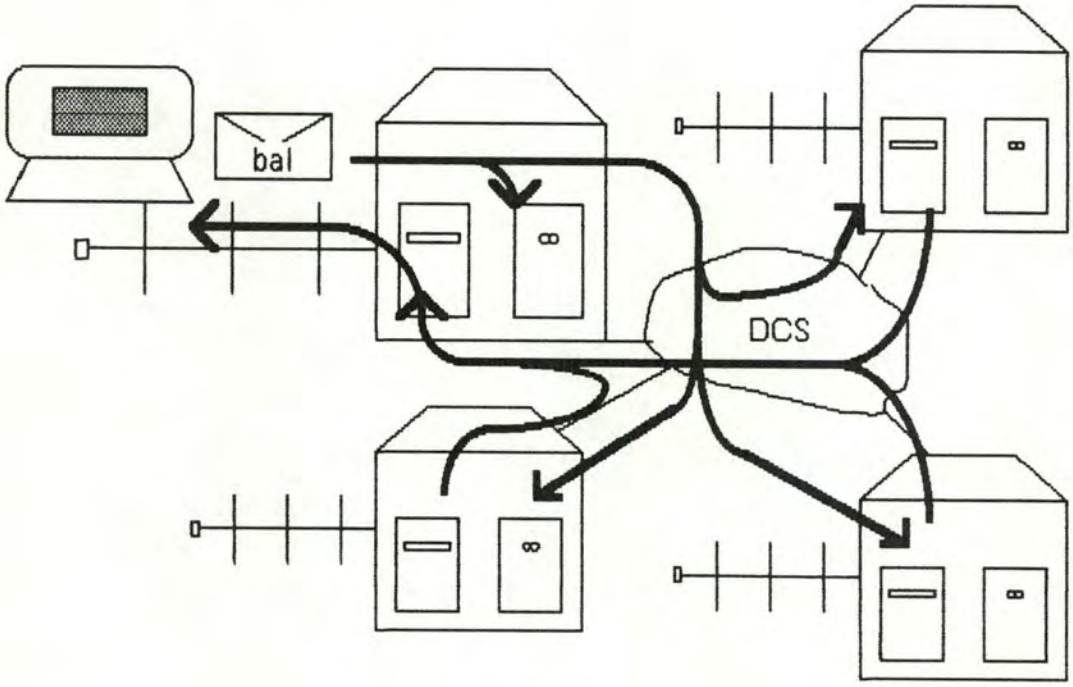

Fig. 4.6 : Consultation bal dans la politique 1

* pour la politique 1 (fig. 4.6) : puisque tout réseau du système peut détenir des documents destinés à un abonné donné, il est nécessaire de rassembler les différents liens (document-abonné donné) dispersés à travers tout le système, afin de constituer la boîte-aux-lettres de cet abonné.

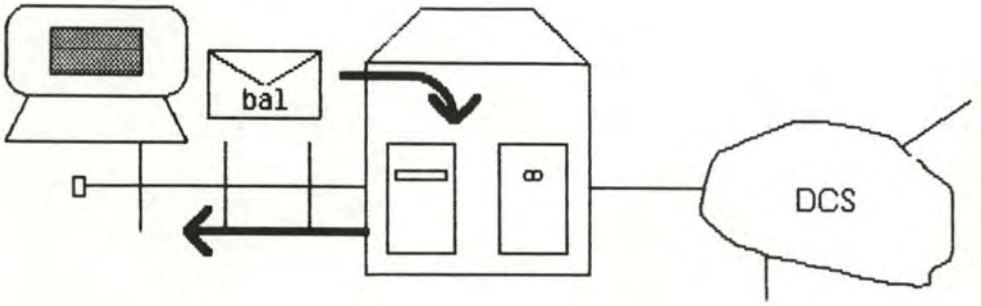

Fig. 4.7 : Consultation bal dans les politiques 3 et 4

* pour la politique 3 (fig. 4.7) : comme tous les documents destinés aux abonnés locaux se trouvent dans le serveur local, la constitution de la boîte aux lettres de l'abonné est directement établie sur base des liens présents au serveur local.

Il en va de même pour la consultation de boîte aux lettres dans la politique 4.

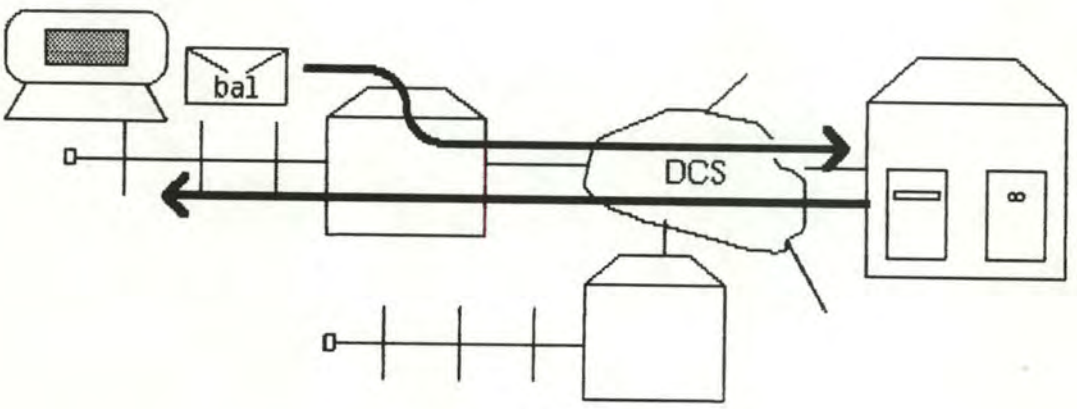

Fig. 4.8 : Consultation bal dans la politique 5

* pour la politique 5 (fig. 4.8) : la boîte aux lettres de l'abonné est établie sur base des liens détenus par le serveur (du réseau) de référence.

4.1.3. La consultation de document:

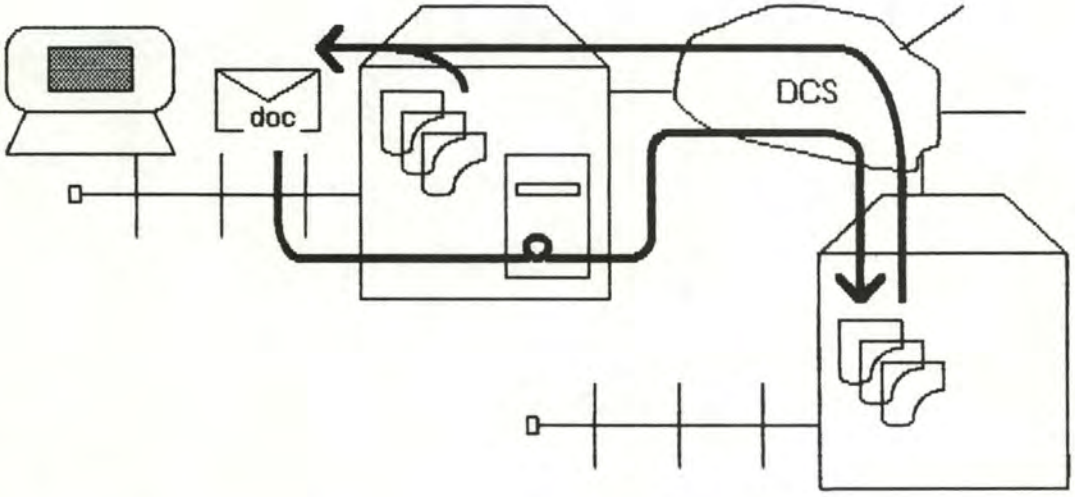

Fig. 4.9 : Consultation document dans la politique 1

* pour la politique 1 (fig. 4.9) : comme les documents sont stockés dans leur réseau de création, il faut situer ce réseau (grâce à la boîte aux lettres) et acheminer le document jusqu'au réseau de l'abonné.

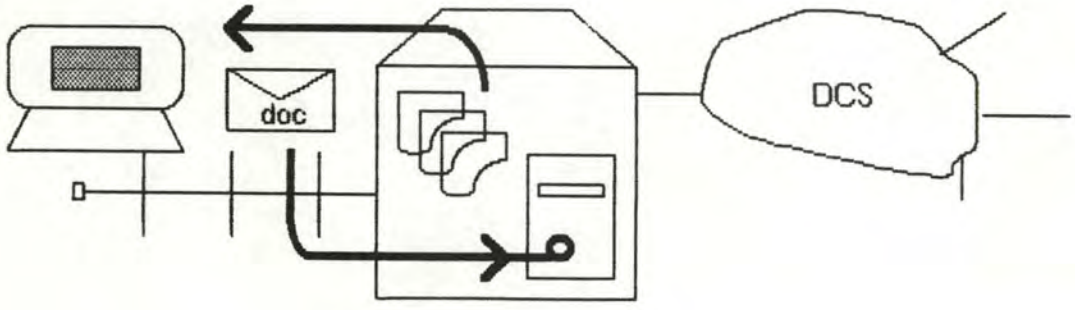

Fig. 4.10 : Consultation document dans les politiques 3 et 4

* pour la politique 3 (fig. 4.10) : tous les documents destinés aux abonnés locaux étant stockés dans le réseau local, un accès local au document est suffisant.

Il en va de même en ce qui concerne la politique 4.

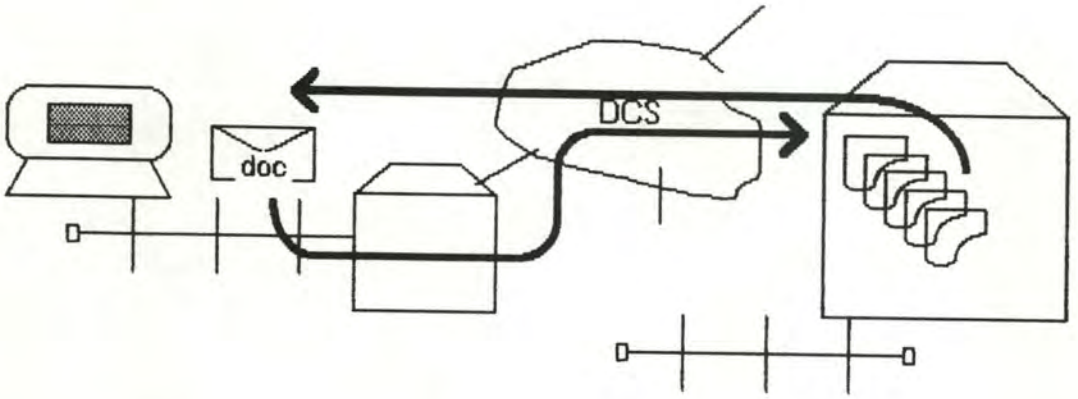

Fig. 4.11 : Consultation document dans la politique 5

* pour la politique 5 (fig. 4.11) : il suffit de demander l'acheminement du document depuis le réseau de référence vers le réseau du destinataire.

4.1.4. La suppresssion de document:

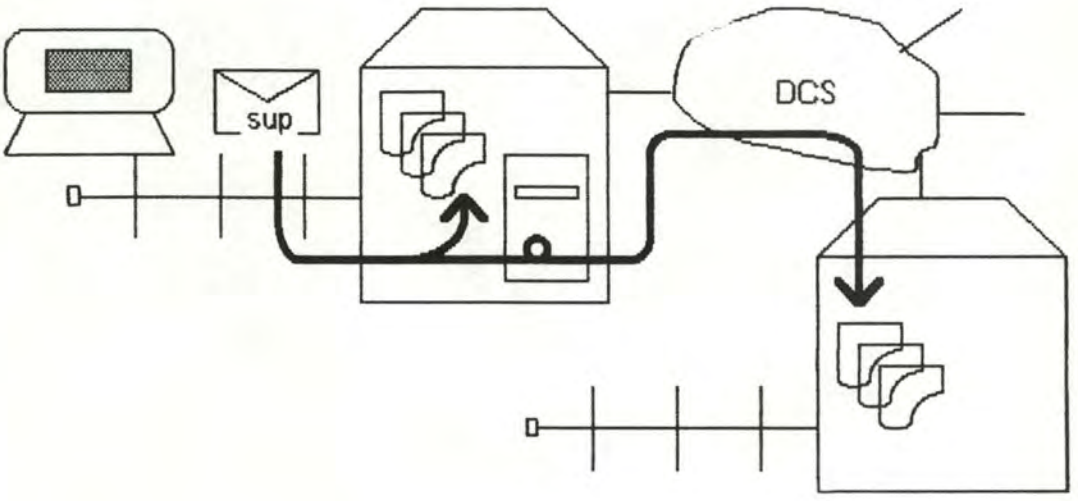

Fig. 4.12 : Suppression dans la politique 1

* pour la politique 1 (fig. 4.12) : il faut supprimer le lien entre l'abonné et le document, ce lien se situant dans le réseau de création du document.

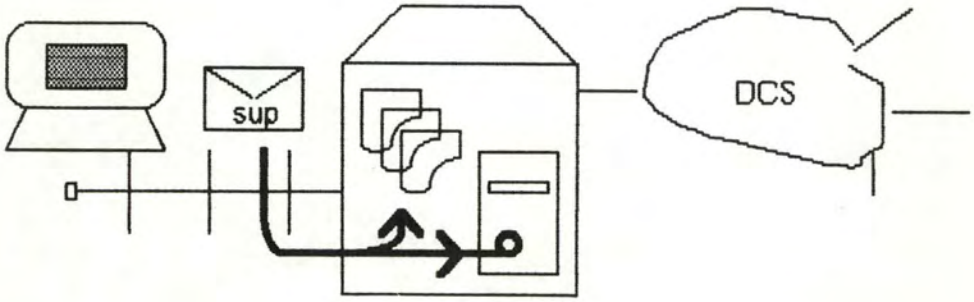

Fig. 4.13 : Suppression dans la politique 3

* pour la politique 3 (fig. 4.13) : il suffit de supprimer le lien qui existe au niveau du serveur local.

* pour la politique 4 (fig. 4.14) : comme dans la politique 3, le lien entre l'abonné et le document doit être supprimé localement, mais aussi dans tous les autres serveurs (des réseaux éloignés).

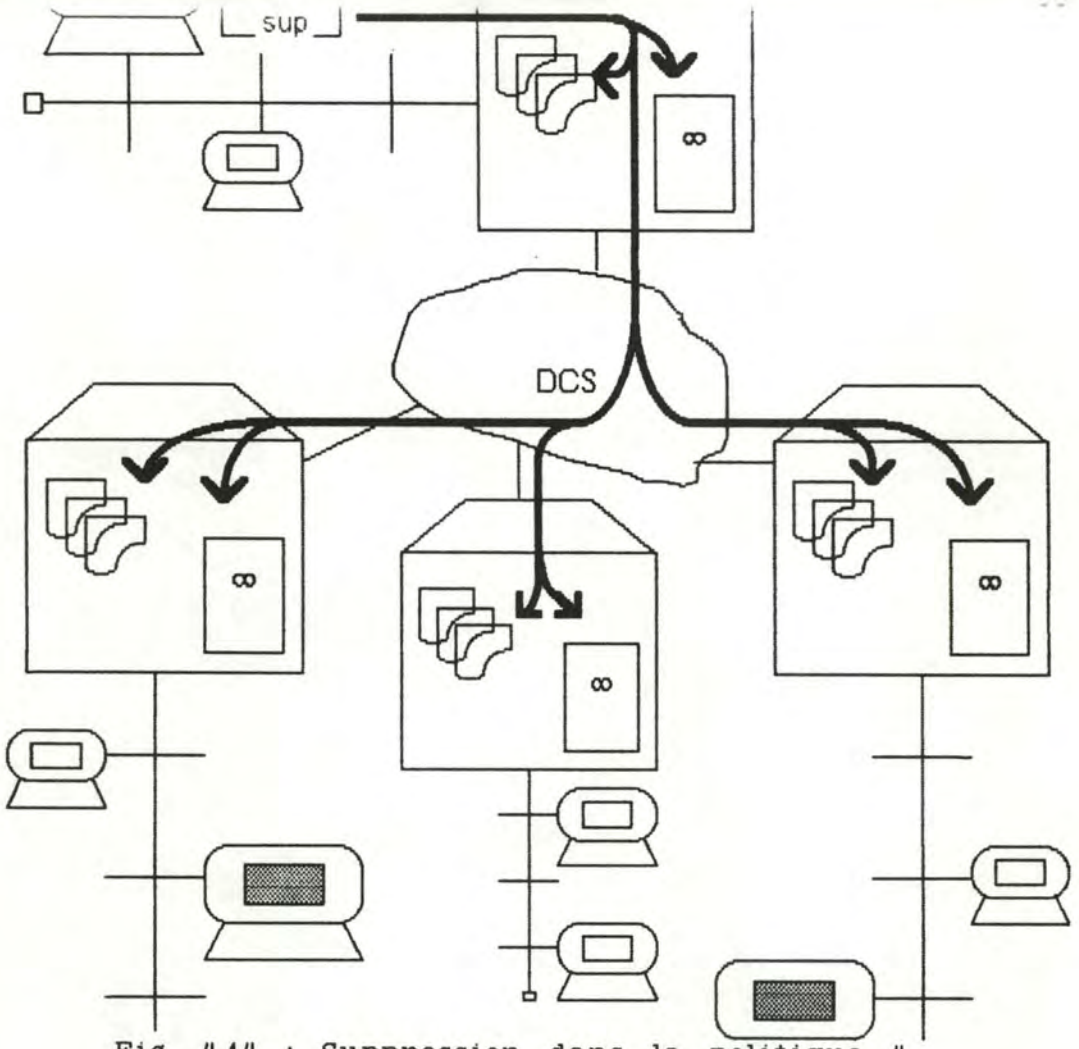

Fig. 4.14 : Suppression dans la politique 4

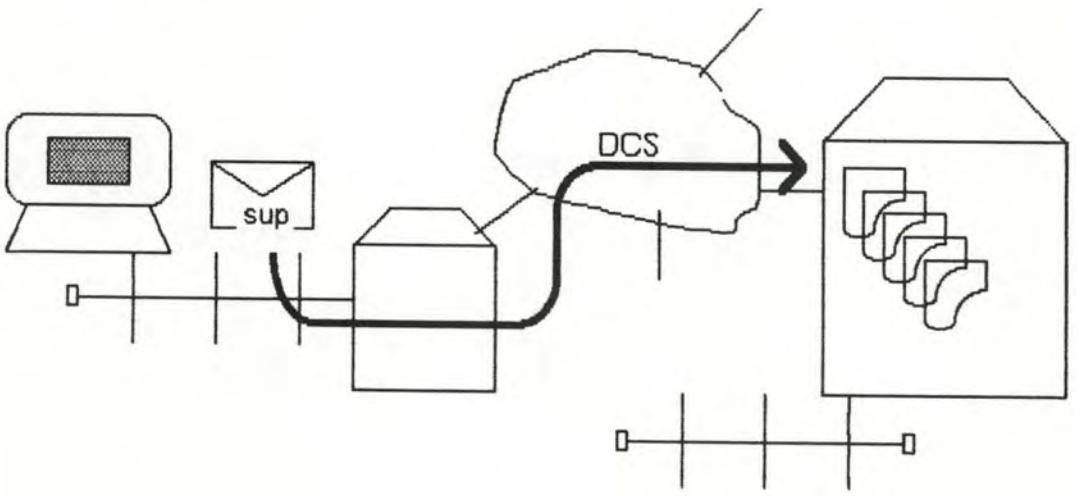

Fig. 4.15 : Suppression dans la politique 5

* pour la politique 5 (fig. 4.15) : le lien qui doit être supprimé se trouve au serveur du réseau de référence.

4.2. Les diagrammes de flux.

L'objectif de cette partie sera, après la description intuitive de chaque fonction vue au point précédent, d'étudier les diagrammes de flux introduits par ces fonctions.

Nous identifierons les différents traitements nécessaires à chaque fonction, à quel endroit ils seront exécutés, les flux de messages entre les différents *processeurs* et la dynamique entre les traitements et les messages. Ceci nous permettra, par la suite, de définir au sein d'une architecture logique, des modules regroupant les traitements identifiés.

Comme dans le point précédent, nous allons envisager séparément les fonctions élémentaires du courrier électronique. Pour chacune, nous détaillerons les diargrammes de flux dans les politiques retenues.

4.2.1. La création/expédition d'un document.

4.2.1.1. Dans la politique 1:

Rappelons que, dans cette politique, les documents sont stockés dans le serveur du réseau de création/expédition.

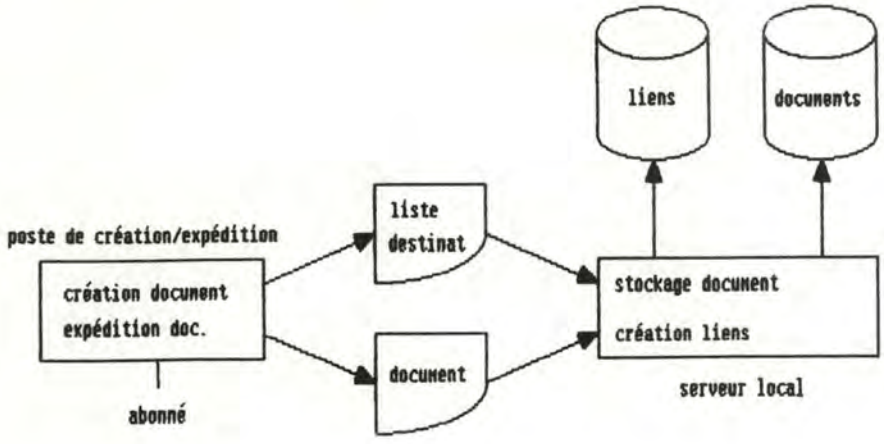

Fig. 4.16 : création/expédition d'un document dans la politique 1

L'abonné (fig. 4.16) crée et expédie un document à partir d'un poste (appelé poste créateur). Le document et sa liste de destinataires sont envoyés au serveur du réseau local (appelé serveur local).

Le serveur local stocke le document dans le fichier des documents et crée les liens abonné destinataire-document dans le fichier des liens.

4.2.1.2. Dans la politique 3:

Dans cette politique, un document est stocké dans le serveur du réseau local de création/expédition ainsi que dans les serveurs des réseaux locaux destinataires.

L'abonné (fig. 4.17) crée et expédie un document à partir du poste créateur. Le document et sa liste de destinataires sont envoyés au serveur local.

Le serveur local stocke le document dans le fichier des documents; il constitue des listes de destinataires par réseau local, en tenant compte de la localisation des abonnés destinataires; il envoie ensuite une copie du document et la liste de destinataires appropriée à chaque réseau local; enfin, sur base de sa propre liste de destinataires, il crée les liens abonné destinataire-document dans le fichier des liens.

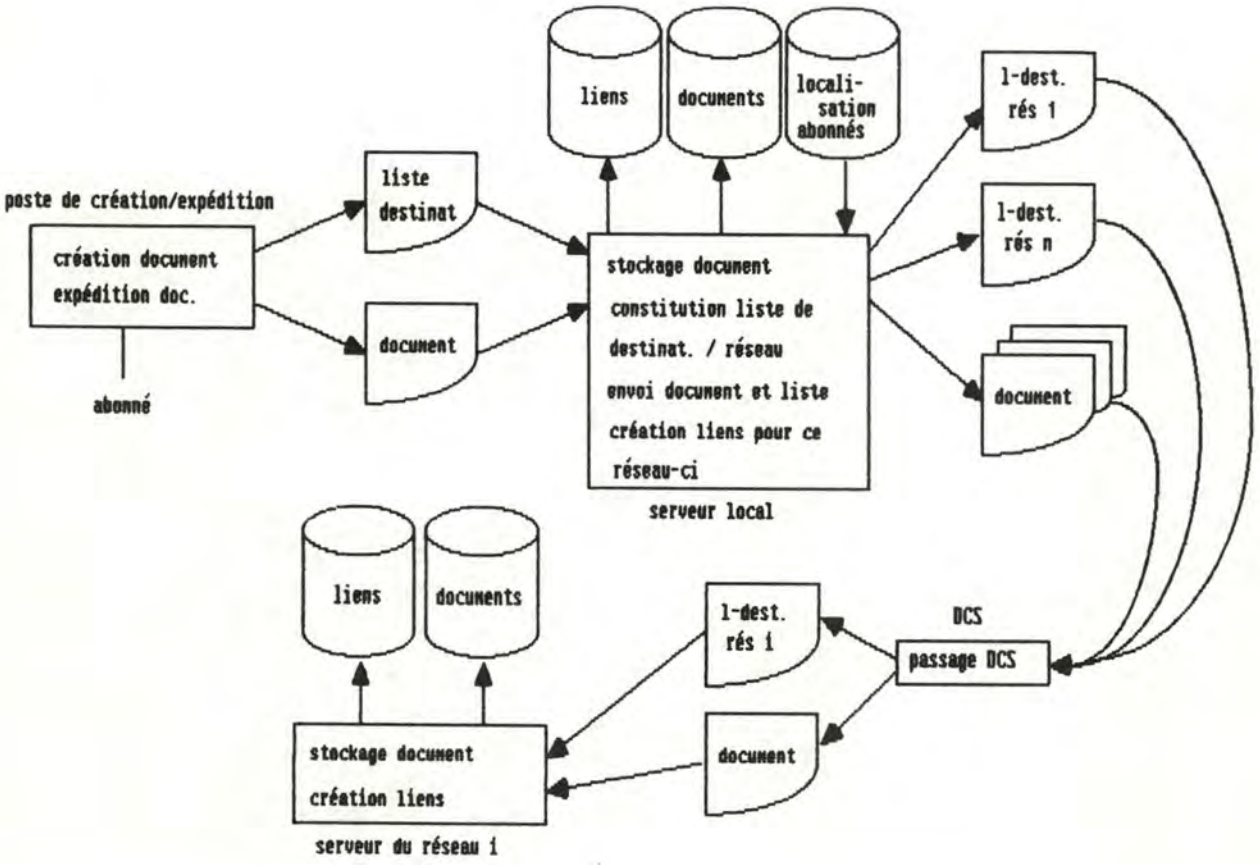

Fig. 4.17 : création/expédition d'un document dans la politique 3

Les copies du document et les listes de destinataires transitent par le réseau public DCS et arrivent par paires aux serveurs des réseaux locaux destinataires.

Chacun de ces serveurs, stocke le document reçu et crée les liens destinataire-document sur base de la liste qui lui a été envoyée.

4.2.1.3. Dans la politique 4:

Dans cette politique, un document est stocké dans chaque serveur de réseau local.

L'abonné (fig. 4.18) crée et expédie un document à partir du poste créateur. Le document et sa liste de destinataires sont envoyés au serveur local.

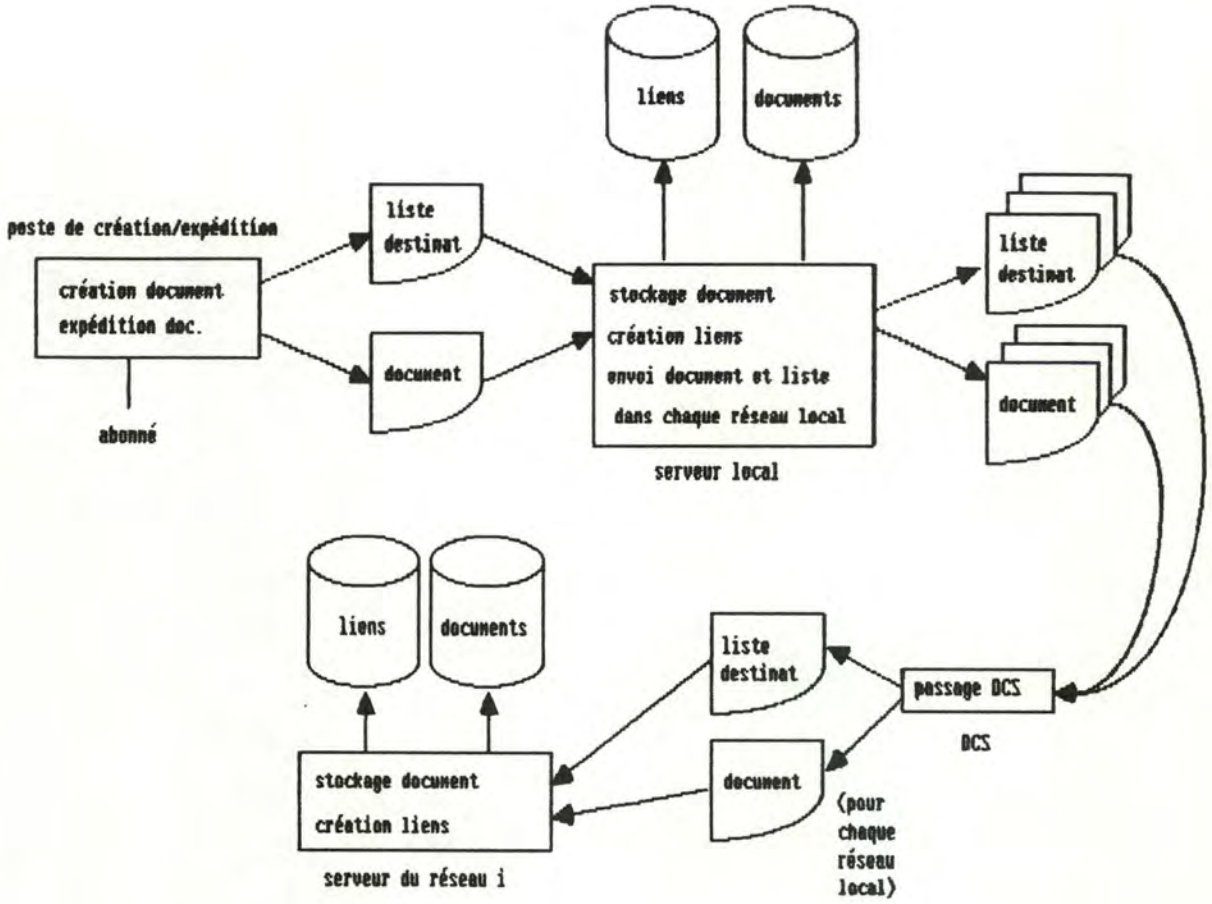

Fig. 4.18 : création/expédition d'un document dans la politique 4

Le serveur local stocke le document dans le fichier des documents; il crée les liens destinataire-document dans le fichier des liens; ensuite, il envoie une copie du document et la liste des destinataires vers chaque réseau local.

Les copies du document et les listes de destinataires transitent par le réseau public DCS et arrivent par paires aux serveurs des réseaux locaux.

Chacun de ces serveurs, stocke le document reçu et crée les liens destinataire-document sur base de la liste qui lui a été envoyée.

4.2.1.4. Dans la politique 5:

Dans cette politique, un document est stocké dans le serveur du réseau local de référence.

L'abonné (fig. 4.19) crée et expédie un document à partir du poste créateur. Le document et sa liste de destinataires sont envoyés au serveur local.

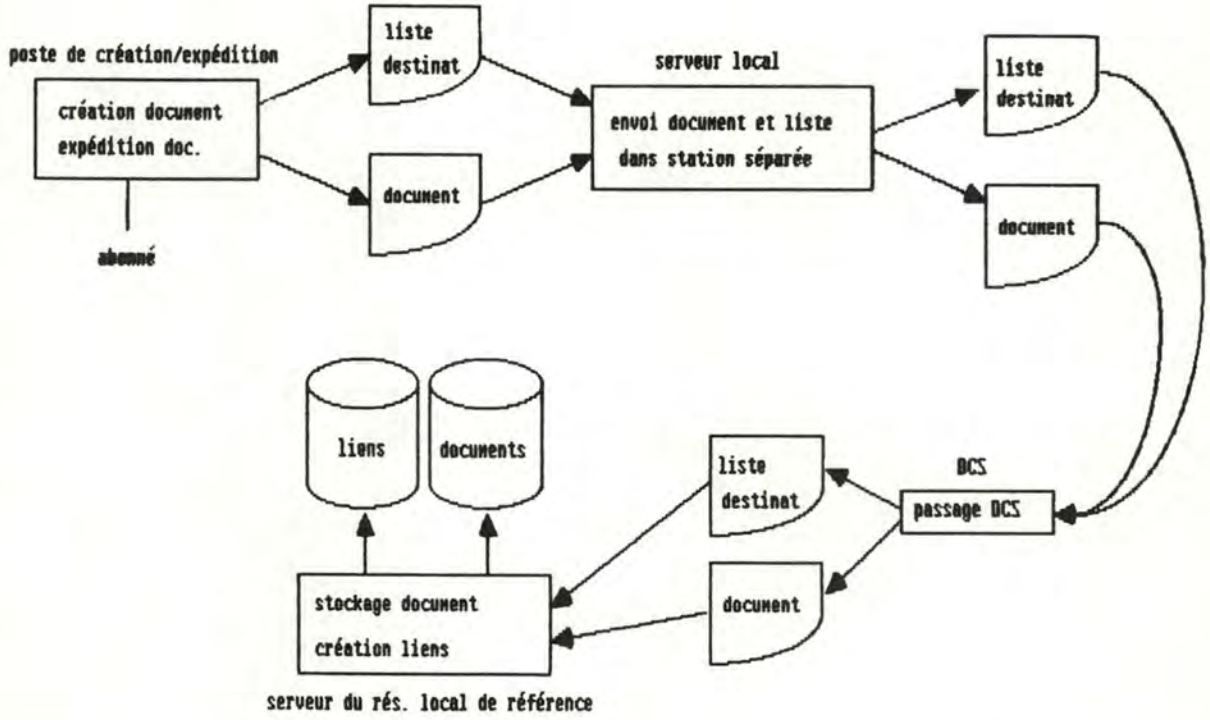

Fig. 4.19 : création/expédition d'un document dans la politique 5

Le serveur local envoie directement le document et la liste des destinataires vers le réseau local de référence.

Le document et la liste de destinataires transitent par le réseau public DCS et arrivent au serveur du réseau local de référence.

Ce serveur stocke le document reçu et crée les liens destinataire-document sur base de la liste qui lui a été envoyée.

4.2.2. La consultation de la boîte aux lettres.

La boîte aux lettres (*BAL*) est construite à partir des liens destinataire-document. Il est en effet possible de retrouver, grâce à ces liens, les différents documents destinés à un abonné donné.

La consultation de la boîte aux lettres se fait en deux temps : l'envoi d'une demande de consultation de BAL dans le(s) serveur(s) où sont stockés des liens relatifs à l'abonné qui consulte, puis la constitution de la BAL avec la(les) réponse(s) du(des) serveur(s).

4.2.2.1. Dans la politique 1:

Les liens sont stockés dans le même serveur que le document auquel ils correspondent, c'est-à-dire dans le serveur du réseau local de création/expédition. Pour un abonné donné, nous pouvons donc avoir des liens dans tout serveur de réseau local: il suffit qu'au moins un document destiné à notre abonné ait été expédié de ce réseau. Nous devrons donc parcourir tous les serveurs lors de la consultation de la BAL.

L'abonné (fig. 4.20) demande la consultation de sa BAL à partir d'un poste (appelé poste consultant). Cette demande est envoyée au serveur local.

Le serveur local envoie une demande de consultation de BAL à chacun des autres serveurs et il crée une BAL provisoire sur base de son propre fichier de liens.

Les demandes de consultation de BAL transitent par le réseau public DCS et arrivent aux serveurs éloignés.

Chaque serveur éloigné, recevant cette demande, crée une BAL provisoire sur base de son propre fichier de liens et envoie la BAL provisoire vers le serveur local.

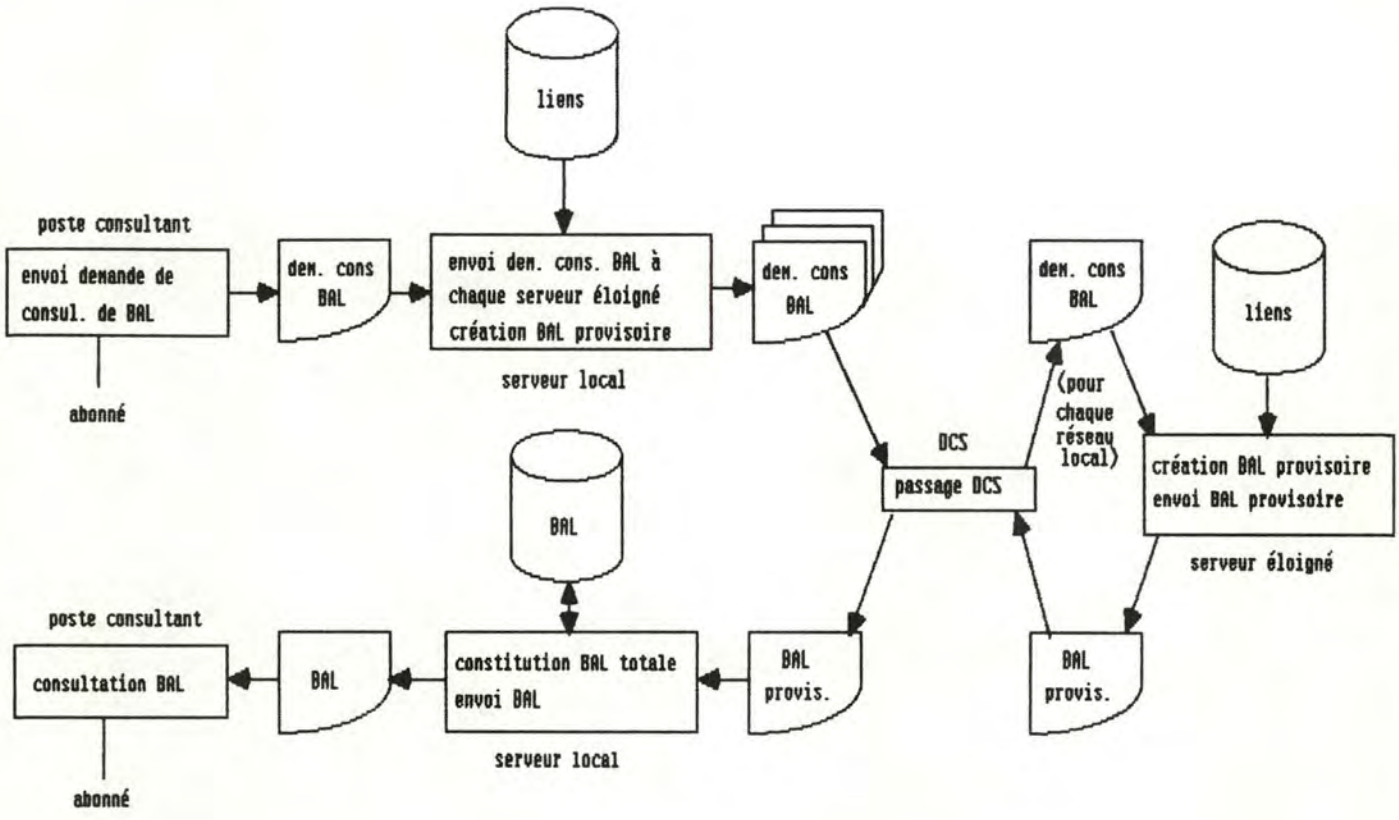

Fig. 4.20 : consultation de la BAL dans la politique 1

La BAL provisoire transite à nouveau par le réseau public DCS et arrive au serveur local.

Le serveur local, chaque fois qu'il reçoit une BAL provisoire d'un serveur éloigné, complète sa BAL. Quand tous les serveurs ont répondu, la BAL est définitive, elle peut être stockée et envoyée vers le poste consultant.

Le poste reçoit la BAL et l'abonné peut la consulter.

4.2.2.2. Dans la politique 3:

Dans cette politique, nous savons que les documents et les liens sont stockés dans les serveurs des réseaux locaux où sont localisés les destinataires. Nous pouvons donc constituer la BAL

complète sur base du fichier de liens du serveur local.

L'abonné (fig. 4.21) demande la consultation de sa BAL à partir du poste consultant. Cette demande est envoyée au serveur local.

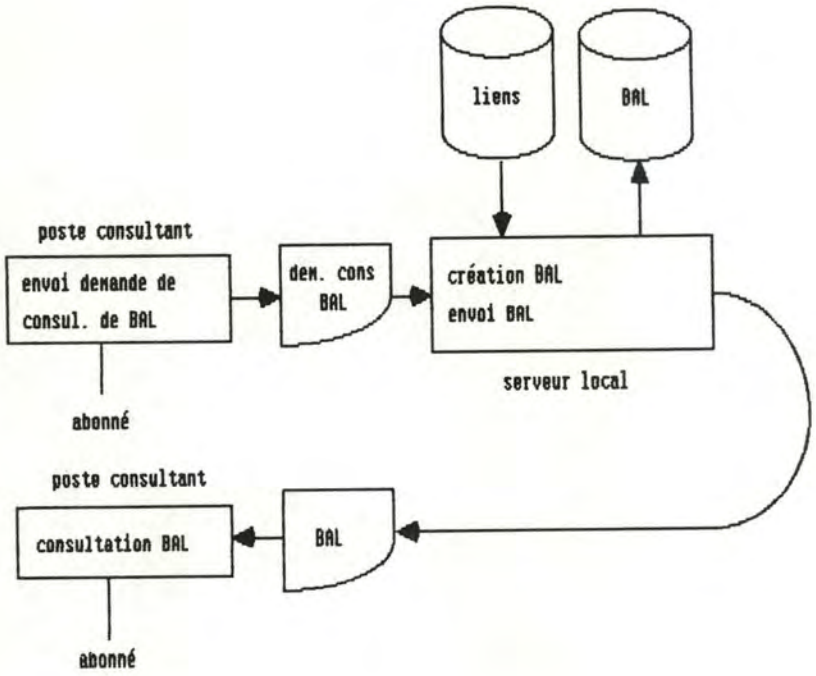

Fig. 4.21 : consultation de la BAL dans la politique 3

Le serveur local crée la BAL sur base de son propre fichier de liens, la stocke et l'envoie vers le poste consultant.

Le poste reçoit la BAL et l'abonné peut la consulter.

4.2.2.3. Dans la politique 4:

Dans cette politique, tous les documents et tous les liens sont stockés dans chaque serveur. Nous pouvons donc, comme dans la politique 3, constituer la BAL complète sur base du fichier de liens du serveur local.

4.2.2.4. Dans la politique 5:

La demande devra parvenir au serveur de référence qui est le seul à connaître les liens et les documents.

L'abonné (fig. 4.22) demande la consultation de sa BAL à partir du poste consultant. Cette demande est envoyée au serveur local.

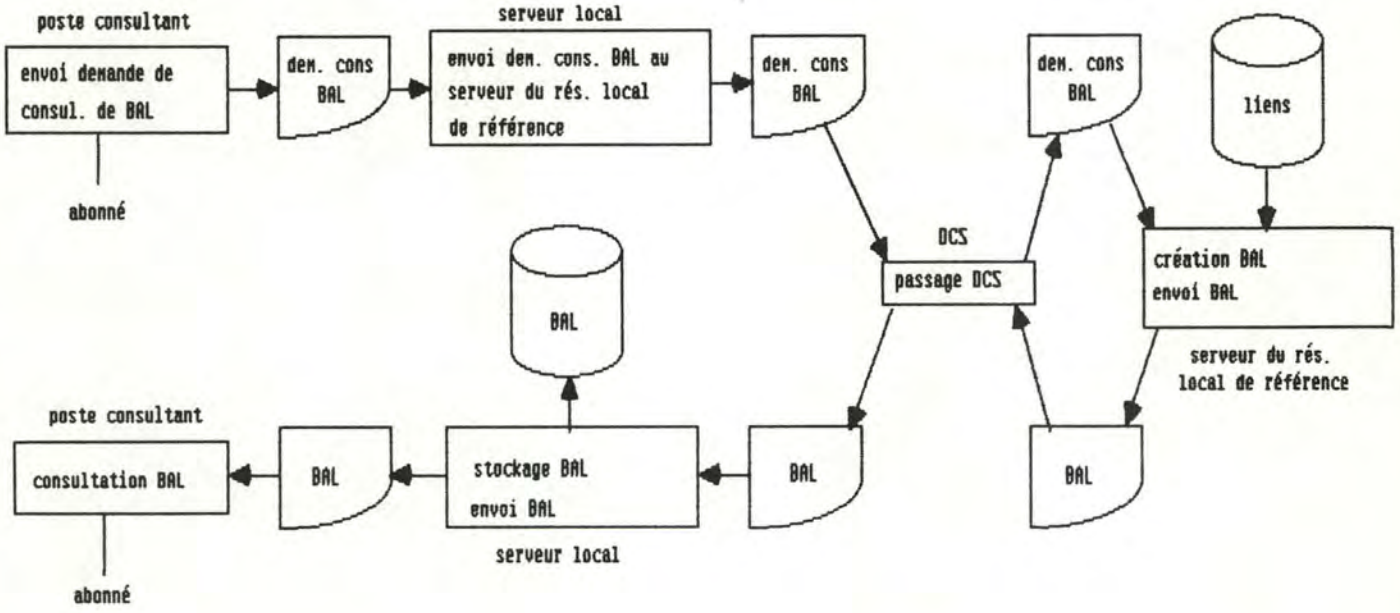

Fig. 4.22 : consultation de la BAL dans la politique 5

Le serveur local envoie une demande de consultation de BAL au serveur de référence.

La demande de consultation de BAL transite par le réseau public DCS et arrive au serveur de référence.

Celui-ci, recevant cette demande, crée la BAL sur base de son propre fichier de liens et l'envoie vers le serveur local.

La BAL transite à nouveau par le réseau public DCS et arrive au serveur local.

Le serveur local stocke la BAL et l'envoie vers le poste consultant.

Le poste reçoit la BAL et l'abonné peut la consulter.

4.2.3. La consultation d'un document.

La consultation d'un document se fait en deux temps: l'envoi d'une demande de consultation de document au serveur le plus proche où est stocké le document désiré, puis la réception et la consultation proprement dite du document.

Le serveur le plus proche où est stocké un document est déterminé par la politique. Si la politique permet les copies multiples, le serveur le plus proche sera certainement le serveur local; nous savons, en effet, que chaque serveur de réseau destinataire au moins a reçu une copie. Si c'est une politique à unicité de document, le serveur le plus proche sera le seul serveur où le document est stocké.

4.2.3.1. Dans la politique 1:

Un document est stocké dans le serveur du réseau local de création/expédition. Nous devons donc connaître l'endroit d'où il a été expédié. Ceci est possible lors de la constitution de la boîte aux lettres: la BAL provisoire qui vient d'un serveur éloigné ne concerne que les documents stockés dans ce serveur. Il faut garder (et ceci peut être caché à l'abonné) une trace de la localisation des documents lors de la constitution de la BAL.

L'abonné (fig. 4.23) demande la consultation d'un document à partir d'un poste (appelé poste consultant). Cette demande est envoyée au serveur local.

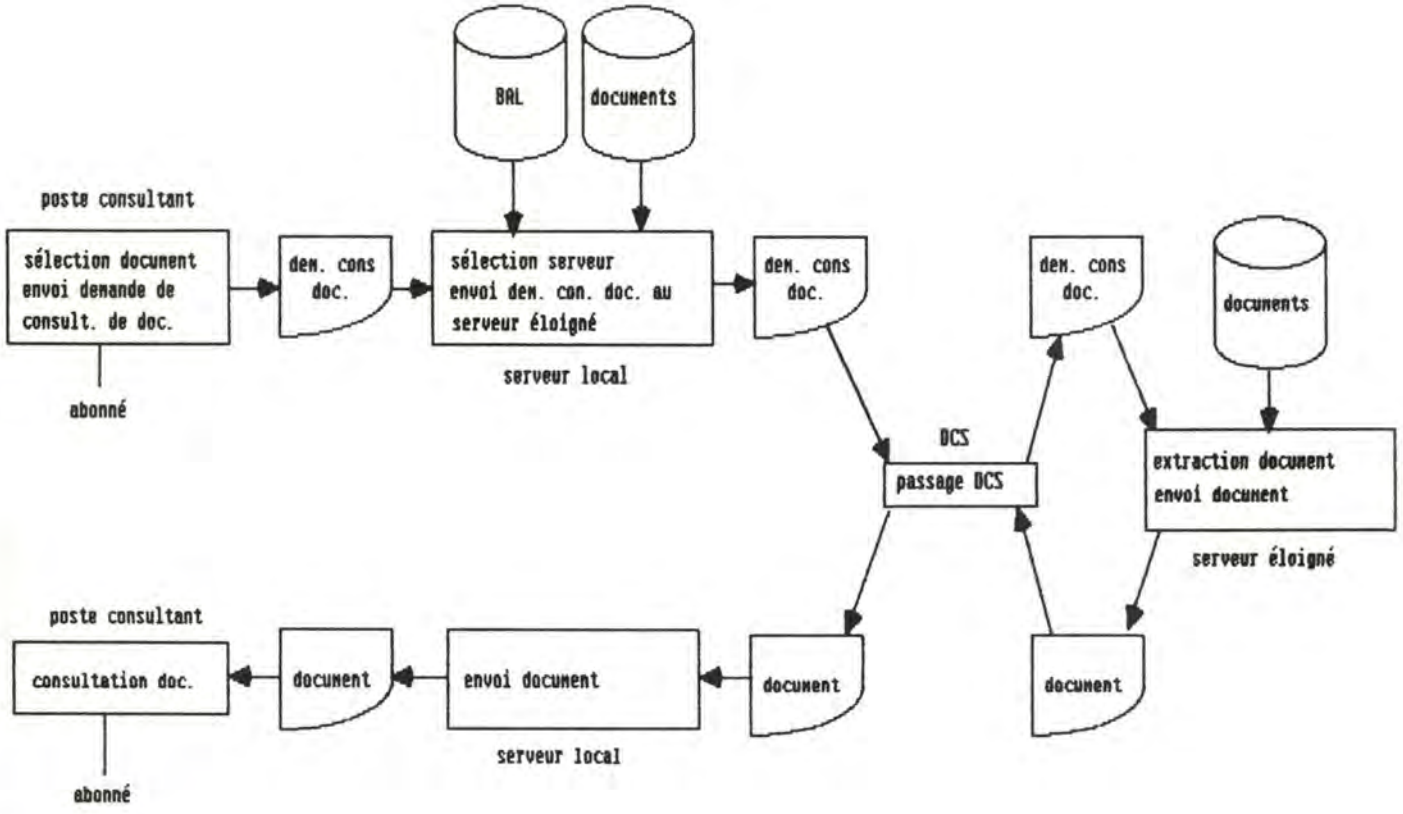

Fig. 4.23 : consultation d'un document dans la politique 1

Le serveur local détermine, dans la BAL, le serveur où le document est stocké. Si le document est stocké dans le serveur local, celui-ci peut l'extraire directement de son fichier des documents; sinon, il envoie une demande de consultation de document au serveur éloigné qui a été sélectionné.

La demande de consultation de document transite par le réseau public DCS et arrive au serveur éloigné.

Le serveur éloigné, recevant cette demande, extrait une copie du document de son fichier des documents et envoie cette copie vers le serveur local.

Le document transite par le réseau public DCS et arrive au serveur local.

Le serveur local envoie le document vers le poste consultant.

Le poste reçoit le document et l'abonné peut le consulter.

4.2.3.2. Dans la politique 3:

Dans cette politique, les documents sont stockés dans les serveurs des réseaux locaux où sont localisés les destinataires. Nous pouvons donc aller chercher le document dans le serveur local.

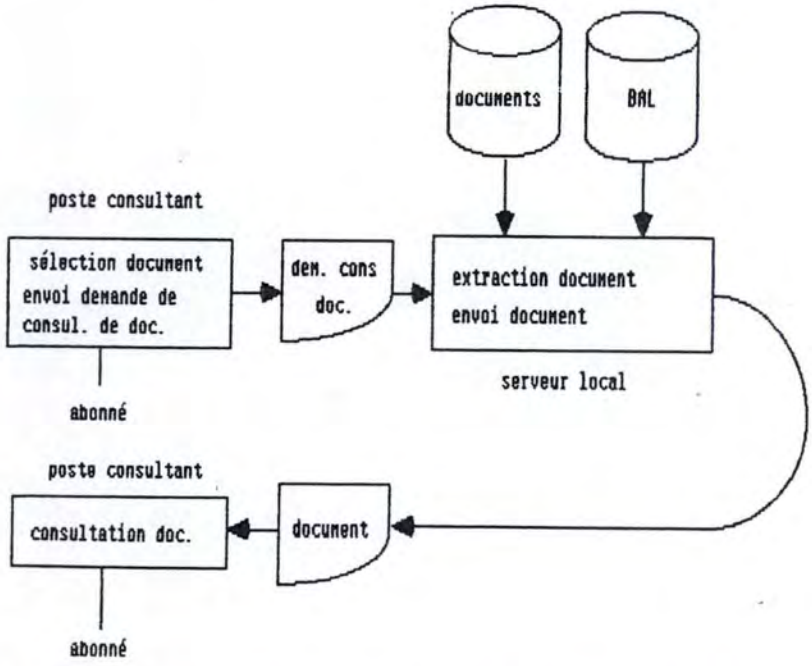

Fig. 4.24 : consultation d'un document dans la politique 3

L'abonné (fig. 4.24) demande la consultation d'un document à partir du poste consultant. Cette demande est envoyée au serveur local.

Le serveur local extrait le document de son fichier de documents et l'envoie vers le poste consultant.

Le poste reçoit le document et l'abonné peut le consulter.

4.2.3.3. Dans la politique 4:

Dans cette politique, tous les documents et tous les liens sont stockés dans chaque serveur. Nous pouvons donc, comme dans la politique 3, aller chercher le document dans le serveur local.

4.2.3.4. Dans la politique 5:

La demande devra parvenir au serveur de référence qui est le seul à connaître les documents.

L'abonné (fig. 4.25) demande la consultation d'un document à partir du poste consultant. Cette demande est envoyée au serveur local.

Le serveur local envoie une demande de consultation de document au serveur de référence.

La demande de consultation de document transite par le réseau public DCS et arrive au serveur de référence.

Celui-ci, recevant cette demande, extrait le document de son fichier de documents et l'envoie vers le serveur local.

Le document transite à nouveau par le réseau public DCS et arrive au serveur local.

Le serveur local envoie le document vers le poste consultant.

Le poste reçoit le document et l'abonné peut le consulter.

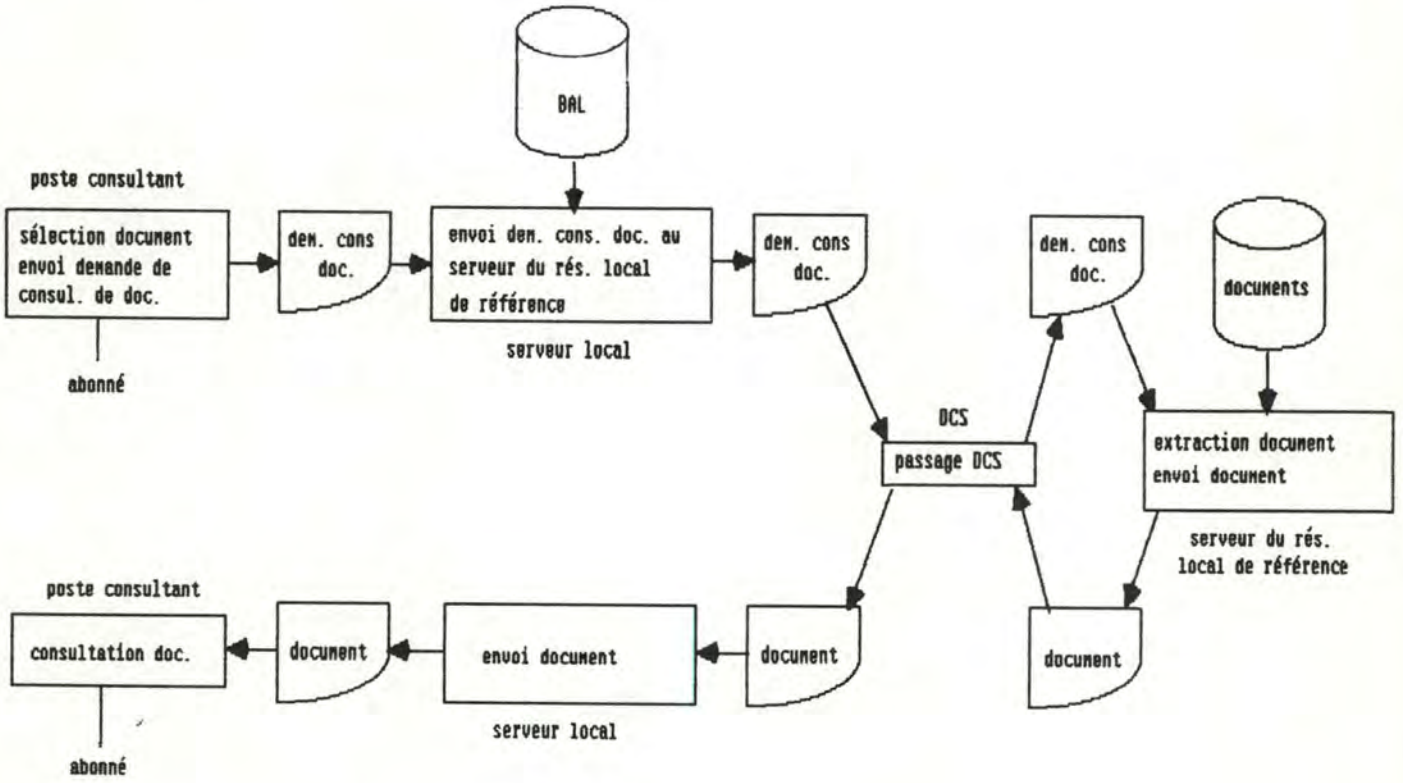

Fig. 4.25 : consultation d'un document dans la politique 5

4.2.4. La suppression d'un document.

La suppression d'un document, comme la consultation, se fait dans le serveur le plus proche où est stocké le document.

La suppression est une suppression logique, puis une suppression physique. Le document supprimé par un abonné n'est plus accessible à cet abonné, il est supprimé logiquement pour cet abonné, le lien destinataire-document concernant cet abonné et ce document est supprimé. Lorsqu'un document est supprimé logiquement pour tous les abonnés auxquels il était accessible, alors ce document est supprimé physiquement de la base de données.

4.2.4.1. Dans la politique 1:

Un document est stocké dans le serveur du réseau local de création/expédition. Nous connaissons l'endroit d'où il a été expédié grâce à la BAL (cfr consultation de document dans la politique 1).

L'abonné (fig. 4.26) demande la suppression d'un document à partir d'un poste (appelé poste supprimant). Cette demande est envoyée au serveur local.

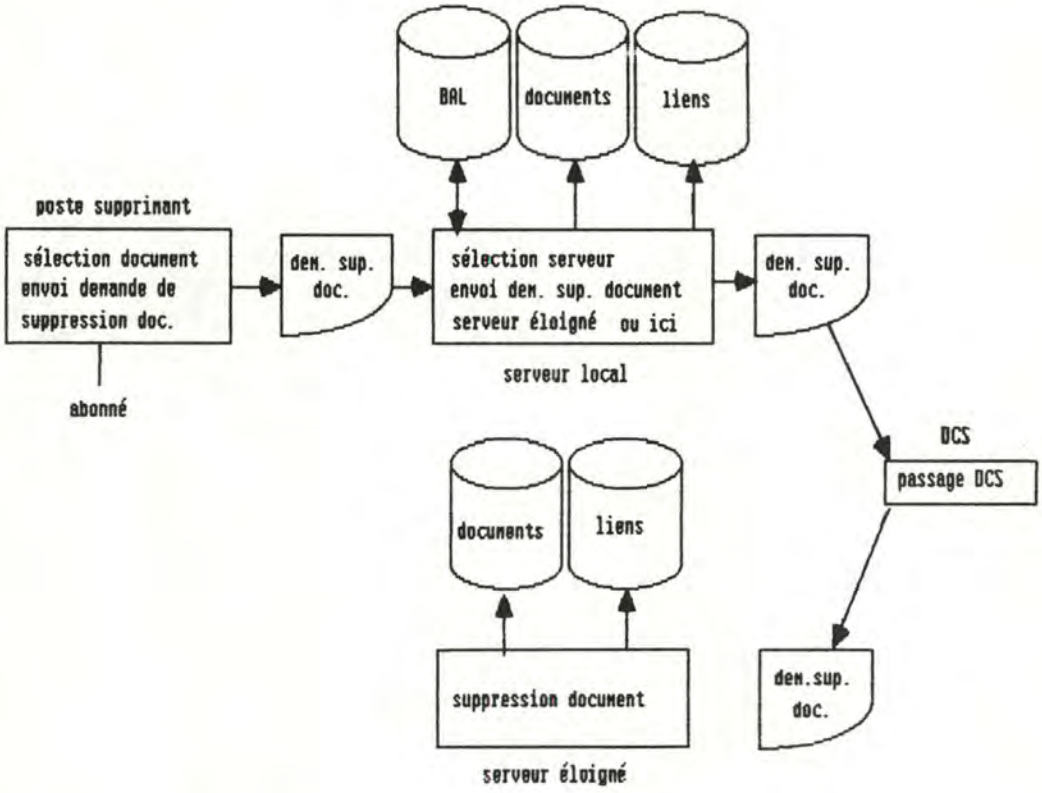

Fig. 4.26 : suppression de document dans la politique 1

Le serveur local détermine, dans la BAL, le serveur où le document est stocké. Si le document est stocké dans le serveur local, celui-ci peut le supprimer directement; sinon, il envoie une demande de suppression de document au serveur éloigné qui a été sélectionné.

La demande de suppression de document transite par le réseau public DCS et arrive au serveur éloigné.

Le serveur éloigné, recevant cette demande, supprime le document pour l'abonné.

4.2.4.2. Dans la politique 3:

Dans cette politique, les documents sont stockés dans les serveurs des réseaux locaux où sont localisés les destinataires. Nous allons donc supprimer le document dans le serveur local.

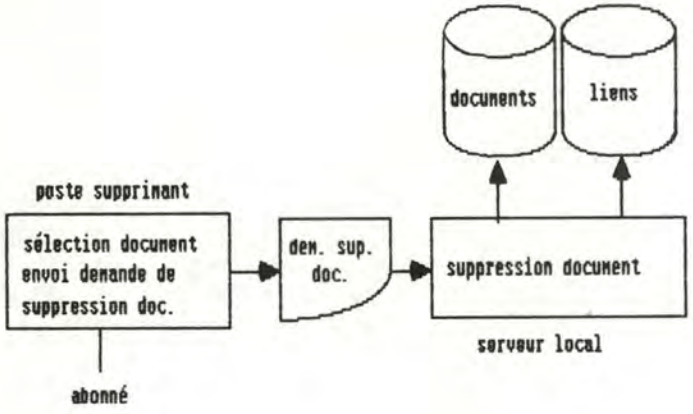

Fig. 4.27 : suppression d'un document dans la politique 3

L'abonné (fig. 4.27) demande la suppression d'un document à partir du poste supprimant. Cette demande est envoyée au serveur local.

Le serveur local supprime le document.

4.2.4.3. Dans la politique 4:

Dans cette politique, tous les documents et tous les liens sont stockés dans chaque serveur. Il faut donc propager la demande de suppression de document dans tous les serveurs.

L'abonné (fig. 4.28) demande la suppression d'un document à partir du poste supprimant. Cette demande est envoyée au serveur local.

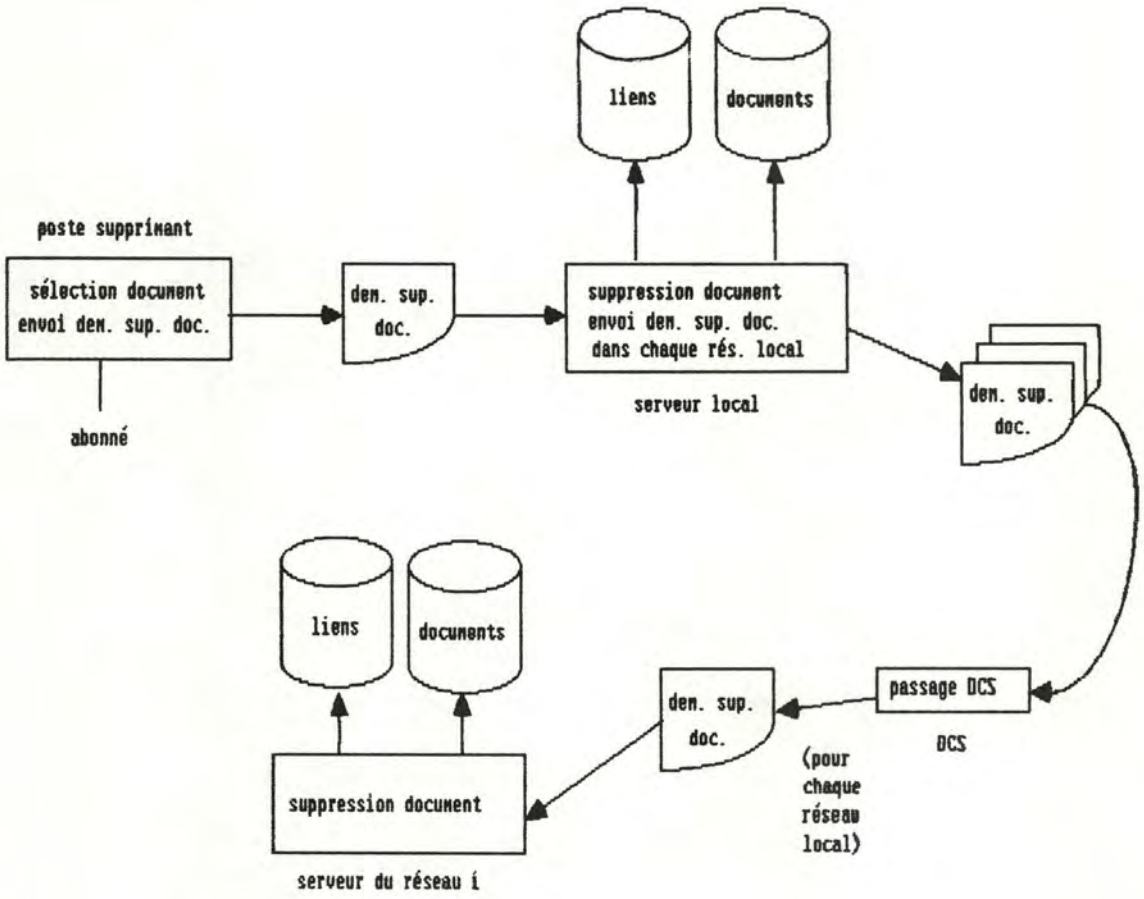

Fig. 4.28 : suppression d'un document dans la politique 4

Le serveur local supprime le document et envoie une demande de suppression de document vers tous les autres serveurs de réseaux locaux du système.

Les demandes de suppression de document transitent par le réseau public DCS et arrivent aux serveurs éloignés.

Chaque serveur éloigné, recevant cette demande, supprime le document pour l'abonné.

4.2.4.4. Dans la politique 5:

La demande devra parvenir au serveur de référence qui est le seul à connaître les documents et les liens.

L'abonné (fig. 4.29) demande la suppression d'un document à partir du poste supprimant. Cette demande est envoyée au serveur local.

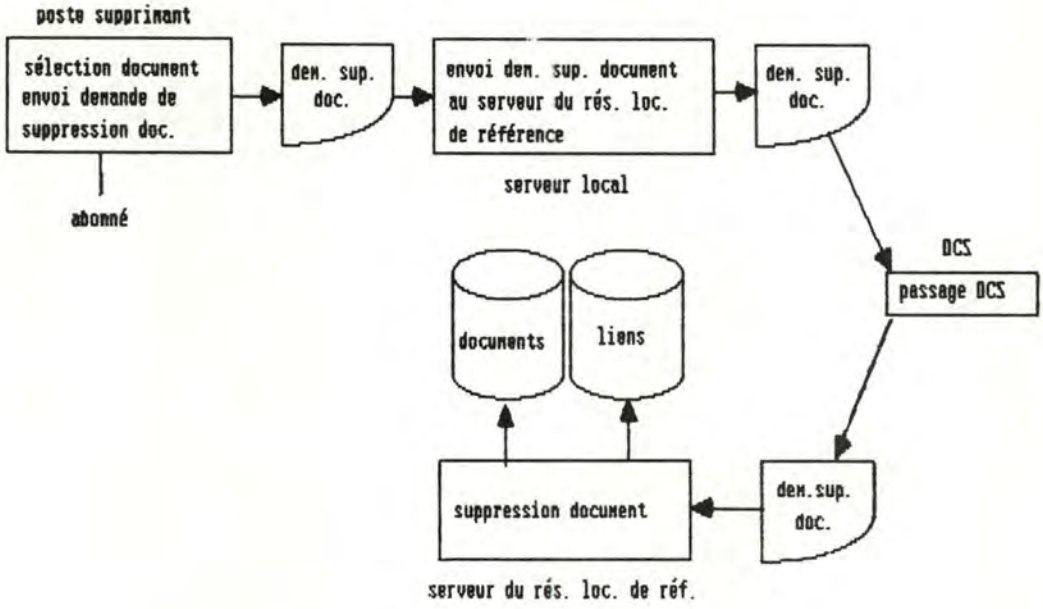

Fig. 4.29 : suppression d'un document dans la politique 5

Le serveur local envoie une demande de suppression de document au serveur de référence.

La demande de suppression de document transite par le réseau public DCS et arrive au serveur de référence.

Celui-ci, recevant cette demande, supprime le document.

4.3. L'architecture logique.

Nous avons déterminé, dans les diagrammes de flux du point précédent, une série de procédures qui doivent être exécutées soit par un poste, soit par un serveur, soit par DCS. Les procédures des serveurs donnent lieu à la définition d'une architecture logique au sein de laquelle elles servent d'interfaces entre les différents modules.

L'architecture logique qui est proposée ici (Fig. 4.30) concerne donc le programme des serveurs de réseaux locaux. Tous les traitements qui ont été vus dans les diagrammes de flux et qui concernent les serveurs locaux ou éloignés sont regroupés dans des modules de l'architecture.

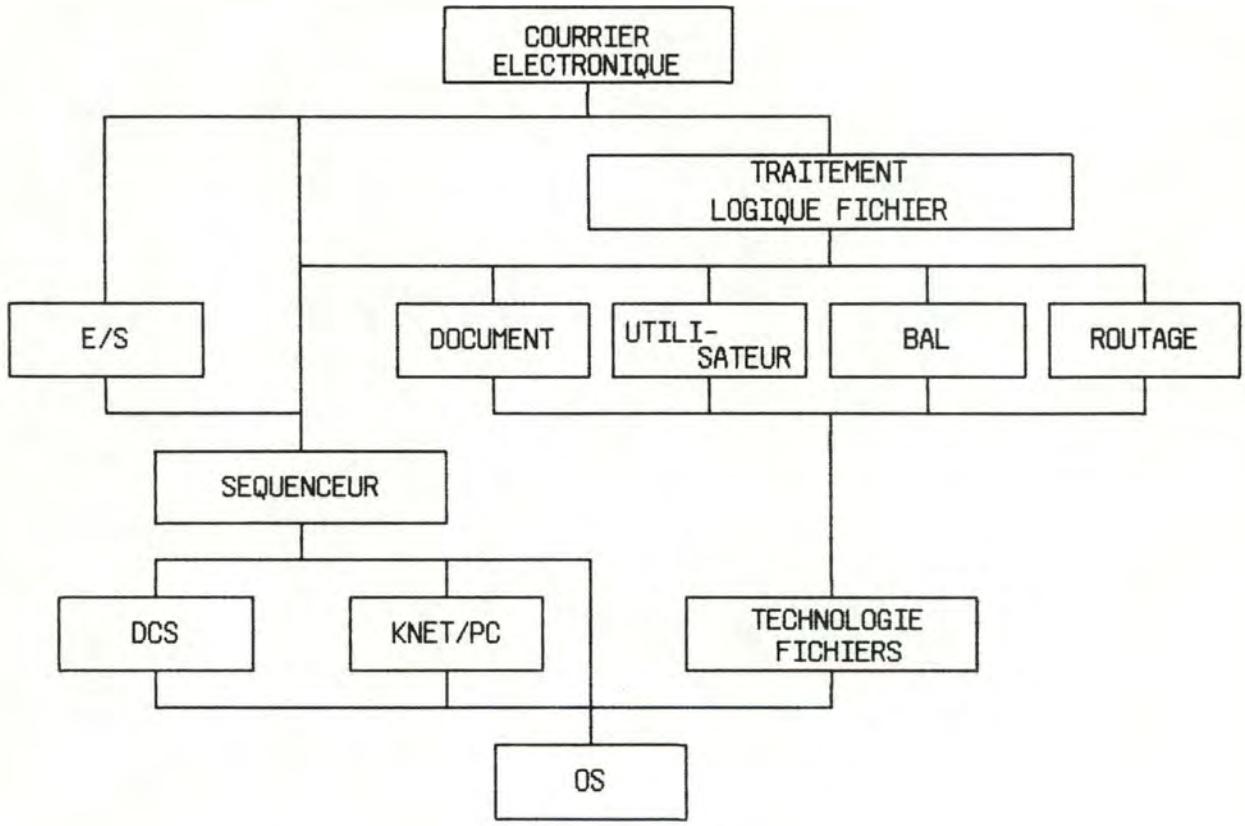

Fig. 4.30 : L'architecture logique d'un serveur

L'architecture que nous proposons est valable quelque soit la politique. Les traitements dépendant directement des politiques sont cachés dans le module TRAITEMENT LOGIQUE FICHIER. L'implémentation est réalisée de manière telle qu'un changement de politique ne nécessite aucun changement du programme ([1]).

4.3.1. Les modules de bas niveau.

Les modules de bas niveau existent, ils ne nécessitent aucune implémentation. Ces modules sont la base de l'architecture logique, car tous les autres modules reposent sur le bon fonctionnement de ceux-ci.

Le module *système d'exploitation (OS)* traite tout ce qui concerne l'exploitation de l'ordinateur d'un point de vue hardware.

Le module *technologie fichiers* prend en charge toutes les fonctions relatives aux fichiers. Ces fonctions sont: ouverture et fermeture d'un fichier, accès séquentiel ou direct à un article, lecture, écriture, suppression d'un article.

Le module *DCS* permet d'établir une liaison avec le réseau public DCS. Il permet d'ouvrir ou de fermer une connexion, d'envoyer ou de recevoir un fichier, d'exécuter une commande à distance.

Le module *KNET/PC* gère les liaisons entre le serveur local et les postes de travail sur le réseau local. Il permet d'envoyer ou de recevoir un fichier, de connecter directement un poste intelligent au serveur local.

4.3.2. Le séquenceur.

Le *séquenceur* est le module charnière entre le bas niveau et les niveaux supérieurs. Ce module, tout comme les modules de niveau supérieur,

([1]) voir à ce sujet le manuel d'utilisation en annexe C.

nécessite une implémentation. Il réalise la synchronisation des fonctions du courrier électronique, il effectue les appels aux traitements nécessaires suivant les fonctions, il est aussi un veilleur lorsqu'une réponse doit venir d'un serveur éloigné.

4.3.3. Les modules de niveau intermédiaire.

Les modules de niveau intermédiaire manipulent chacun un concept particulier du courrier électronique, un module séparé constitue l'interface avec l'utilisateur.

Le module *entrées/sorties (E/S)* est l'interface utilisateur. Il permet d'afficher des menus, des messages, des boîtes aux lettres, des documents; il permet aussi de lire des commandes du couurier, des documents, des identifiants de documents, des listes de destinataires.

Le module *document* gère les documents. Il permet de stocker, de rechercher, de supprimer un document.

Le module *utilisateur* gère les abonnés. Il permet d'identifier et de localiser un abonné, de créer et de supprimer un lien abonné-destinataire, de consulter le(s) lien(s) pour un abonné ou pour un document donné.

Le module *BAL* gère les boîtes aux lettres. Il permet de créer, de compléter, de consulter oude supprimer une boîte aux lettres, de localiser un document, de supprimer un document d'une boîte aux lettres.

Le module *routage* n'est utilisé que dans la politique 3, il remplit les traitements de routage rendus nécessaires par la localisation des abonnés. Il permet de créer, de compléter, de consulter et de supprimer une liste d'abonnés par réseau. Ces

fonctions sont utiles pour créer, par exemple, une liste de destinataires pour un réseau donné.

4.3.4. Les modules de haut niveau.

Les modules de haut niveau réalisent effectivement les fonctions du courrier électronique et manipulent tous les messages des diagrammes de flux.

Le module *traitement logique fichier* gère tout dialogue entre serveurs. Il réalise les envois et les réceptions de boîtes aux lettres, de documents, de demande de consultation de boîte aux lettres ou de document, les expéditions de nouveaux documents, il traite les différentes demandes de consultation. Ce module effectue tous ces traitements en tenant compte de la politique choisie. Il constitue une boîte à outils qui permet de changer de politique sans transformer le programme.

Le module *courrier électronique* réalise les fonctions du courrier électronique: création/expédition de document, consultation de boîte aux lettres, consultation de document, suppression de document, création, connexion, suppression d'abonné.

Chapitre 5 : Evaluation quantitative des politiques.

Le programme du courrier électronique dont l'architecture a été donnée dans la chapitre 4 nous sert à évaluer les différents critères de performance. Dès lors, nous étudions le nombre de documents stockés pour le volume mémoire. Nous évaluons ensuite les coûts imputables à DCS. Enfin, nous abordons le temps de consultation. Nous laissons de côté le temps d'expédition, car, comme nous l'avons dit lors de sa définition dans le chapitre 2, celui-ci peut être caché à l'utilisateur.

5.1. Le nombre de documents.

5.1.1. Introduction.

L'évaluation du nombre de documents pour chacune des politiques implémentées nous permet d'avoir un ordre de grandeur du volume de stockage nécessaire dans le courrier électronique.

Il n'est pas possible de quantifier exactement le volume nécessaire. En effet, celui-ci dépend de plusieurs facteurs dont les principaux sont le nombre de documents et le volume moyen d'un document; ce dernier facteur n'est pas connu. Nous savons cependant que le volume de stockage est directement proportionnel au nombre de documents. Comme les documents sont les éléments principaux à stocker, nous pouvons comparer les diverses politiques par rapport au nombre de documents qu'elles requièrent.

5.1.2. Calcul du nombre moyen de documents.

Nous pouvons déjà affirmer que les politiques qui ont besoin du minimum de volume de stockage sont les politiques à copie unique de document: ce sont les politiques 1 et 5. Il reste donc à évaluer le nombre de documents pour les politiques 3 et 4.

En ce qui concerne la politique 4, le calcul est simple: puisqu'il y a une copie de chaque document dans chaque serveur, le nombre de documents est multiplié par le nombre de serveurs.

Le calcul du nombre de documents dans la politique 3 est plus complexe: nous évaluons le nombre moyen de documents. Avec i étant le nombre de serveurs de réseaux destinataires, le nombre moyen de documents est la somme pour i allant de 1 à n du produit du nombre de documents (pour un i donné) par la probabilité de i.

$$\text{Nbre moyen} = \sum_{i=1}^{n} \text{nbre doc } (i) * \text{Pr } [i]$$

La probabilité de i, c'est-à-dire la probabilité d'avoir i réseaux destinataires est égale à 1/n si tous les réseaux ont la même probabilité d'être destinataires et si le nombre de réseaux destinataires est uniformément réparti entre 1 et n.

Le nombre de documents pour i réseaux destinataires est i+i si tous les réseaux destinataires sont éloignés: une copie pour chaque réseau destinataire plus une pour le réseau d'expédition du document. Si un des réseaux destinataires est le réseau d'expédition, le nombre de documents est i+i-1 = i : une copie pour chaque serveur de réseau destinataire éloigné (i-1) et une copie pour le serveur du réseau d'expédition. Il faut encore calculer la probabilité d'avoir chacun de ces deux états. On y arrive par les combinaisons:

- la probabilité d'avoir i réseaux destinataires éloignés sachant qu'on a i réseaux destinataires parmi n réseaux est :

$$\frac{C_{n-1}^{i}}{C_{n}^{i}} = \frac{n-i}{n}$$

- la probabilité d'avoir i-1 réseaux destinataires éloignés et le réseau d'expédition étant aussi destinataire sachant qu'on a i réseaux destinataires parmi n réseaux est :

$$\frac{C_{n-1}^{i-1}}{C_{n}^{i}} = \frac{i}{n}$$

En résumé, le nombre de documents si on a i réseaux destinataires est :

$$\frac{n-i}{n} * (1+i) + \frac{i}{n} * i = \frac{1}{n} * [(n-1)i + n]$$

Le nombre moyen de documents dans la politique 3 est :

$$\frac{1}{n} \sum_{i=1}^{n} [((n-1)i + n) - \frac{1}{n}]$$

$$= \frac{n+2}{2} - \frac{1}{2n}$$

5.1.3. Interprétation des résultats.

Comparons, dans un tableau, les valeurs du nombre de documents stockés par document expédié dans les diverses politiques, suivant le nombre de réseaux du système.

	pol. 1	pol. 3	pol. 4	pol.5
1 réseau	1	1	1	1
2 réseaux	1	1,75	2	1
3 réseaux	1	2,33	3	1
4 réseaux	1	2,88	4	1
5 réseaux	1	3,40	5	1
6 réseaux	1	3,92	6	1
7 réseaux	1	4,43	7	1
8 réseaux	1	4,94	8	1
9 réseaux	1	5,44	9	1
10 réseaux	1	5,95	10	1

La politique la plus intéressante du strict point de vue du volume de stockage nécessaire est la politique 5, car elle utilise le volume minimum en un seul endroit.

La politique 1 utilise également le volume de stockage minimum, mais elle provoque une perte d'efficacité à cause du caractère aléatoire du lieu de stockage d'un document qui nécessite une importante marge de sécurité dans chaque serveur. Néanmoins, ceci peut être considéré comme un avantage puisque la marge de sécurité plus importante permet plus d'extension de la base de données.

La politique 3 utilise un volume de stockage d'autant plus important que le nombre de réseaux locaux - et donc la possibilité d'avoir des destinataires localisés sur des serveurs éloignés - augmente.

La politique 4 est nettement défavorisée par le critère du volume de stockage, puisqu'elle multiplie la base de données par le nombre de réseaux locaux.

Nous constatons que les conclusions auxquelles nous parvenons sont en accord avec le bon sens; elles

confirment d'ailleurs tout à fait l'évaluation qualitative du volume mémoire.

5.2. Les coûts imputables à DCS.

5.2.1. Introduction.

Les coûts que l'on peut imputer à DCS sont de plusieurs types. Certains sont relatifs à l'installation initiale, permettant la connexion à DCS, ou à des redevances périodiques : ces frais sont fixes. D'autres, par contre, varient en fonction de l'utilisation qui est faite de la connexion. Ainsi, on trouvera tout d'abord des coûts liés au volume des transferts réalisés, c'est-à-dire au nombre de messages échangés, à leur longueur, ainsi qu'au prix du byte transféré; ensuite, on trouvera les coûts liés à l'ouverture des connexions à travers DCS, c'est-à-dire, d'une part, le nombre de ces ouvertures et, d'autre part, leur durée.

5.2.2. Les ouvertures de lignes.

En ce qui concerne les coûts relatifs aux ouvertures de lignes, plusieurs optiques sont envisageables : soit on voudra établir une **connexion permanente** entre deux réseaux (ou les serveurs de ces réseaux) donnés; soit on décidera d'établir cette même **connexion une fois par jour** (ouvrir la connexion en début de journée et la fermer en fin de journée); soit encore on préférera une **ouverture à la demande**, lorsqu'un besoin existe (ouverture et fermeture pour chaque transfert).

Ces trois optiques peuvent être différenciées par leur mode de tarification. Ainsi, l'ouverture permanente revient actuellement à 4800 francs pour deux mois :il s'agit donc d'un montant forfaitaire pour les ouvertures de lignes, montant auquel il faut rajouter les frais relatifs aux volumes transférés. L'etablissement d'une connexion journalière ou à la demande coûte 0.15 franc pour l'ouverture elle-même et 0.1 franc par tranche de

trente secondes où la connexion reste ouverte (soit 12 frs de l'heure). Dès lors, une ouverture journalière sera essentiellement sensible à la durée de l'ouverture, tandis qu'une ouverture ponctuelle sera principalement sensibe au nombre-même d'ouvertures, bien que, dans ce cas, chaque transfert sera allourdi de la taxe minimale de 0.1 fr pour trente secondes (une ouverture à la demande coûtera dès lors 0,15 fr + 0,1 fr pour la durée minimale d'ouverture). Rappelons qu'aucune de ces formules ne considère les volumes transférés; il faudra donc en tenir compte séparément.

Si les trois modes de calcul semblent très différents, il existe néanmoins, entre ces trois optiques, un état d'équilibre : l'ouverture permanente coûte 2400 francs par mois, ce qui correspond à peu près à 200 heures d'utilisation d'une connexion journalière ou encore à 9600 ouvertures sur demandes. Il peut dès lors être intéressant de déterminer les conditions dans lesquelles une optique devient plus avantageuse qu'une autre. C'est ce que nous allons faire.

Nous voyons donc que le problème du coût global d'utilisation de DCS peut déjà porter sur une évaluation des différentes politiques sur base de composants tarifaires liés aux ouvertures de lignes; cette évaluation concerne, entre autres, le calcul du **seuil de rentabilité** d'une ligne dans une optique, relativement aux autres, en fonction du volume moyen des échanges entre deux réseaux.

Pour cela, nous devons tout d'abord déterminer le nombre d'appels nécessaires pour une opération réalisée dans une politique donnée. (Si N est le nombre de réseaux)

La politique 1 se caractérisera donc comme suit :
* expédition d'un nouveau document : aucun échange;
* consultation d'un document : (1-1/N) appel en moyenne car on ne passe pas par DCS si le document est stocké localement
* consultation d'une bal : (N-1) appels;
* suppression d'un document : (1-1/N) appel en moyenne;

La politique 3 se caractérisera comme suit :
Rappelons que, pour la politique 3, nous supposons que le nombre de destinataires est quelconque et qu'il n'y a pas de concentration des destinataires sur un réseau donné. Ceci nous permet de donner une approximation du nombre de documents stockés pour un document envoyé :
nbre doc = ((N+2)/2 - 1/(2*N))
* expédition d'un nouveau document : Ndest
  soit, par approximation : ((N+2)/2 - 1/(2*N))*(N-1)/N
* consultation d'un document : aucun échange;
* consultation d'une bal : aucun échange;
* suppression d'un document : aucun échange;

La politique 4 se caractérisera comme suit :
* expédition d'un nouveau document : (N-1) appels;
* consultation d'un document : aucun échange;
* consultation d'une bal : aucun échange;
* suppression d'un document : (R-1) appels;

La politique 5 se caractérisera comme suit :
* expédition d'un nouveau document : (1-1/N) appel en moyenne;
* consultation d'un document : (1-1/N) appel en moyenne;
* consultation d'une bal : (1-1/R) appel en moyenne;
* suppression d'un document : (1-1/R) appel en moyenne;

Pour la suite de cette étude, nous allons devoir fixer certaines valeurs de manière quelque peu arbitraire. Ces valeurs doivent représenter les probabilités d'exécution des différentes opérations du courrier électronique (de base). Nous considérons dès lors, pour ces valeurs, des pourcentages reflétant une utilisation moyenne a priori ou qui peut, en tout cas, se

révéler tout à fait envisageable. Ainsi, par exemple, la consultation de documents doit au moins être équivalente à la création de documents et même nettement supérieure, puisque, d'une part, bon nombre de documents seront probablement envoyés à plus d'un destinataire et, d'autre part, chaque document sera probablement consulté plus d'une fois avant d'être détruit. La consultation d'une boîte aux lettres doit être réalisée au moins une fois par session de travail, pour constater les arrivées de nouveaux documents et surtout permettre la consultation des documents disponibles; la suppression d'un document doit aussi être plus fréquente que l'expédition puisqu'un même document peut être destiné à plusieurs abonnés, donc détruit (logiquement en tout cas) plusieurs fois.

Nous fixerons donc, comme suit, les différents ratios relatifs à l'utilisation des opérations du CE :

* la consultation de documents comptera pour **50** % des opérations;
* la consultation de bal, pour **10** % des opérations;
* l'expédition de documents, pour **15** % des opérations et
* la suppression de documents, pour **25** % des opérations.

Dès lors, nous pouvons exprimer le nombre moyen d'appels par opération, pour chaque politique selon la formule suivante :

Somme, pour toutes les opérations, du produit (nombre d'appels pour une opération fois probabilité de cette opération)

* dans la politique 1 cela donne :
  Pconsdoc*(1-1/N) + (N-1)*Pconsbal + Psupdoc*(1-1/N)
  soit donc : 0.5*(1-1/N) + (N-1)*0.1 + 0.25*(1-1/N)
  = 0.75*(N-1)/N + 0.1*(N-1)
  = **(0.75 + 0.1*N) * (N-1)/N**;
* pol 3 : (Ndest) * Pexpdoc; soit donc : (Ndest) * 0.15
  cette évaluation peut donc être approchée par la fonction suivante : **0.15 * ((N+2)/2 - 1/(2*N))*(N-1)/N**
* pol 4 : (N-1) * Pexpdoc + (N-1) * Psupdoc
  soit donc : (N-1) * (0.15+0.25) = **(N-1) * 0.4**;

* dans la politique 5 on obtient :

(Pexpdoc + Pconsdoc + Pconsbal + Psupdoc)*(1-1/N)

soit donc : (0.15+0.5+0.1+0.25)*(1-1/N) = **(N-1)/N**;

NB : Nous pouvons remarquer ici que, contrairement à ce que l'on pourrait croire a priori, le nombre de transferts par opération dépend directement, dans chacune des politiques étudiées, du nombre de réseaux dans le système.

Cette évaluation nous permet de dégager le coût relatif à une activité de 100 opérations, pour chaque politique (fig. 5.1) (par modification des formules obtenues ci-dessus, en les multipliant par 25, c'est-à-dire 100*0.25, où 100 est le nombre d'opérations et 0.25, le coût d'un appel sur DCS. Ce coût nous sera utile lors de l'évaluation des politiques en fonction des coûts globaux d'utilisation de DCS.

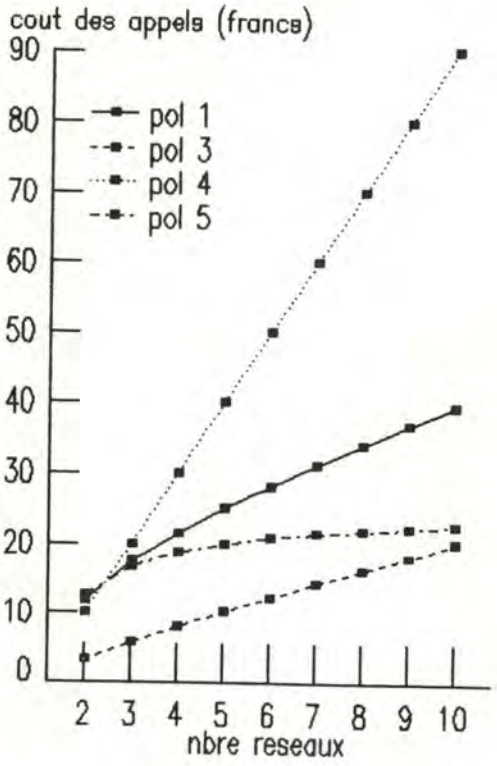

fig. 5.1. : évaluation des coûts d'appels sur DCS.

L'évaluation que nous venons de faire permet de voir que la politique 3 observe un très bon comportement, puisqu'elle reste la meilleure politique même pour un

nombre moyen de réseaux (jusque 10 ou 11). Cette évaluation révèle également un comportement très bon de la politique 5, à partir de 8 ou 9 réseaux; elle n'évolue, pour ainsi dire, plus et devient insensible au nombre de réseaux dans le système. Par contre la politique 1 se montre fort moyenne : elle est constamment deux fois plus chère que la politique 3. Quant à la politique 4, au delà de 3 réseaux, elle devient totalement non-compétitive.

Nous pouvons à présent reprendre la recherche que nous avions commencée précédemment de la formule d'ouverture de connexion la plus avantageuse. Dans cette optique, après avoir déterminé le nombre d'appels nécessaires par opération et par politique, puis en moyenne pour une opération quelconque dans une politique donnée, il nous faut maintenant déterminer la configuration nécessaire au fonctionnement des différentes politiques, afin de pouvoir évaluer le nombre d'échanges que chaque liaison du système supporte pour chaque opération.

Deux types de configurations sont envisageables : la **configuration complète** et la **configuration en étoile**. Notons que nous ne parlons pas ici de la configuration des réseaux locaux, qui sont de type Ethernet, mais de la représentation du système global. La première configuration, complète, consiste à permettre le dialogue entre deux serveurs (de deux réseaux locaux différents) quelconques en imposant l'existence de connexions entre tous les réseaux du système, soit, si le système comporte N réseaux, **(N*(N-1))/2** connexions ! Cette configuration s'applique aux politiques 1, 3 et 4.
La seconde configuration, en étoile, consiste à permettre le dialogue entre un réseau (ou son serveur) donné et tous les autres, en imposant uniquement l'existence de connexions entre chaque réseau du système et le réseau "de référence", comme dans la politique 5; soit, si le système comporte N réseaux, **N-1** connexions !

A partir d'ici, nous pouvons évaluer un aspect essentiel de coût : le seuil de rentabilité d'une ligne dans une politique donnée (et le volume des échanges dans les politiques pour des niveaux d'activité donnés).

Le *seuil de rentabilité* d'une ligne peut être évalué de la façon suivante : dans un premier temps, il faut considérer le nombre moyen d'échanges réalisés par opération, sur une seule ligne d'un système pour une politique donnée; dans un second temps, on doit évaluer le nombre d'opérations (niveau d'activité) nécessaires pour arriver à la barre des 2400 francs mensuels d'une ouverture permanente, car c'est seulement à ce niveau que les trois optiques peuvent être comparées (rappelons que 2400 fr = 9600 appels). Dès lors il sera possible de trouver plus aisément le mode d'ouverture de ligne qui doit être choisi.

Le volume d'échanges par ligne et par opération (fig 5.2) peut être calculé comme suit, pour chaque politique : le nombre moyen de transferts par opération pour une politique donnée doit être divisé par le nombre de liaisons nécessaires dans cette politique :

* pol 1 : ((0.75 + 0.1*N) * (N-1)/N) / ((N-1) * N/2)
  = **(1.5 + 0.2*N) / N^2**;
* pol 3 : (0.15 *((N+2)/2 - 1/(2*N))*(N-1)/N)/((N-1)*N/2)
  (0.3 * ((N+2)/2 - 1/(2*N)) / N^2)
  (appelons cette fonction μ pour les calculs ultérieurs);
* pol 4 : ((N-1) * 0.4) / ((N-1) * N/2)
  = **4 / (5 * N)**;
* pol 5 : ((N-1)/N) / (N-1) = **1 / N**;

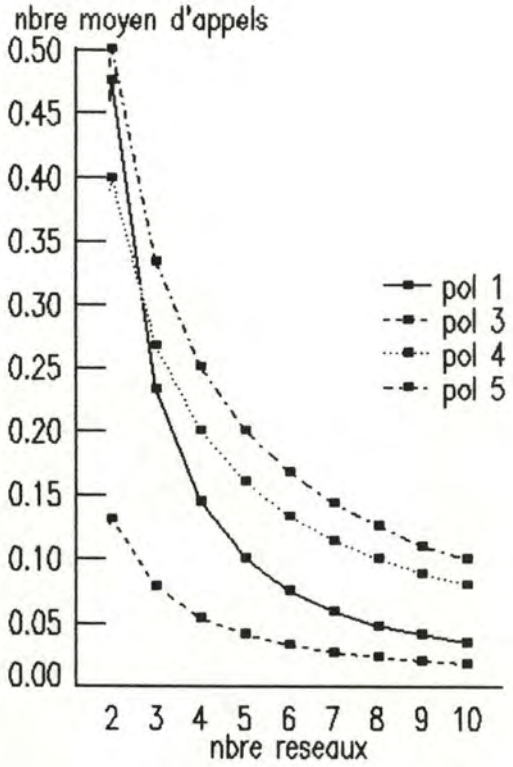

Fig 5.2 : nombre moyen d'appels par ligne par opération

On remarquera aussi que le nombre d'opérations nécessaires par mois pour rendre l'ouverture permanente rentable sera inversément proportionnel au nombre d'échanges moyen par opération, sur une ligne (valeur que nous venons de calculer). En effet :

* Pol 1 : nombre d'opérations nécessaires

$$nbop = 9600 * N^2 / (1.5 + 0.2*N)$$

* Pol 3 : $nbop = 9600 / \mu$
* Pol 4 : $nbop = 9600 * 5 * N / 4 = 12000 * N$
* Pol 5 : $nbop = 9600 * N$

C'est ce nombre minimum d'opérations que l'on appellera seuil de rentabilité, puisque c'est à partir de ce seuil que l'ouverture permanente devient rentable. La figure 5.3 met en évidence le seuil de rentabilité de chaque politique, relativement au nombre de réseaux du système.

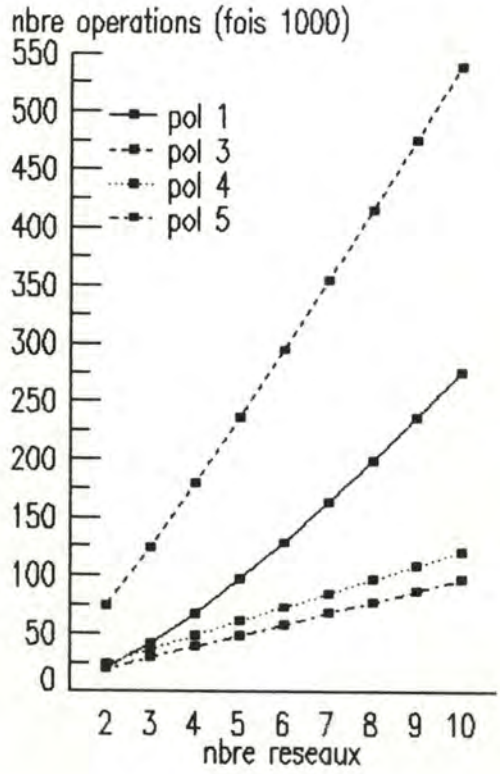

Fig 5.3 : seuil de rentabilité

Il est donc possible, à ce stade, de choisir un mode d'établissement de connexions, en fonction des seuils de rentabilité des lignes et des niveaux minimums d'activité qu'ils engendrent. En effet, nous pouvons remarquer que des quatre politiques étudiées, les politiques 4 et 5 sont celles qui permettent le plus rapidement de préférer une connexion permanente à une connexion temporaire puisqu'il faut très peu d'opérations pour atteindre la barre décisive des 9600 transferts (ou 2400 francs). En ce qui concerne la politique 5, cela se comprend d'ailleurs assez facilement car, pour chaque réseau du système, les échanges sont concentrés sur une seule liaison et chaque opération du CE nécessite un échange sur cette liaison; sauf, bien sûr, pour les opérations demandées à partir du réseau de référence. En ce qui concerne la politique 4, toutes les opérations ne provoquent pas un échange par le réseau public et donc on pourrait s'attendre à avoir globalement moins d'échanges. Néanmoins, les demandes de suppression et les expéditions de nouveaux documents suscitent plusieurs échanges (avec

tous les réseaux du système), ce qui a pour effet d'augmenter globalement le nombre d'échanges et par conséquent de ramener le seuil de rentabilité à un niveau très compétitif.

Par contre, la politique 3, par exemple, se révélera beaucoup plus exigeante au niveau de l'activité nécessaire pour préférer une ouverture permanente à une ouverture ponctuelle : le niveau d'activité nécessaire devra être de deux à trois fois plus élevé que dans la politique 5. Cela s'explique aussi aisément, car la politique 3 exige un grand nombre de liaison d'une part (c'est une configuration complète), et, d'autre part, les liaisons sont en moyenne très peu utilisées par opération. Ceci dit, rien n'empêche de choisir un mode d'ouverture permanente pour une politique telle que celle-ci, s'il se révèle par ailleurs que le niveau d'activité est très important et justifie cette mesure. En effet, une activité minimale justifiant un tel choix dans la politique 3 serait par exemple, en fonction des pourcentages donnés ci-dessus et du nombre d'opérations calculé (soit environ 50000 opérations par réseau), de 5000 consultations de bal, de 7500 nouveaux documents, de 12500 suppressions de documents et de 25000 consultations de documents, pour un mois, pour tous les abonnés venant travailler à ce réseau. Si chacun réalise 20 opérations par jour, cela correspond au travail de 125 abonnés pour un mois. Une telle quantité d'abonné actifs est possible, quoique peu envisageable sur un seul réseau, si l'on considère que, pour les autres politiques, selon un même calcul, on arrive à des ordres de grandeurs nettement moindres. Ainsi, la politique 1 exigerait 68 abonnés, et les politiques 4 et 5 exigeraient seulement 30 et 24 abonnés, respectivement.

Si nous sommes, en définitive, apparemmment capables de choisir entre une solution d'ouverture permanente ou une solution d'ouverture ponctuelle, la difficulté persiste lorsque l'on aborde l'optique de l'ouverture journalière, ou à durée limitée. Signalons tout de suite qu'aucun problème ne se pose si la durée

mensuelle totale des ouvertures journalières dépasse les 200 heures, car de toute facon on lui préférera une ouverture permanente. Là où les choses se corsent, c'est lorsque la durée mensuelle globale est en deça des 200 heures car, alors, le calcul du nombre d'opérations nécessaires doit être revu avec, comme base, le coût relatif à cette durée limitée (nbre d'heures * 12 frs) et non plus aux 2400 francs de l'ouverture permanente. Par exemple, un niveau de 150 heures par mois donnerait donc pour chaque politique les résultats suivants : (150 * 12 = 1800 francs ou 7200 appels)

* Pol 1 : nombre d'opérations nécessaires pour atteindre cette barre : $nbop = 7200 * N^2 / (1.5 + 0.2*N)$
* Pol 3 : $nbop = 7200 / \mu$
* Pol 4 : $nbop = 7200 * 5 * N / 4 = 9000 * N$
* Pol 5 : $nbop = 7200 / N$

ce qui correspond, pour la politique 1 (qui est assez moyenne de ce point de vue) au travail de 51 abonnés par réseau, au lieu des 68 précédemment atteints; la différence n'est pas négligeable mais il est important de voir si le travail de ces 51 abonnés peut être concentré sur les 150 heures offertes, alors que les 68 autres abonnés disposaient, non pas de 200 heures, mais d'un mois complet !

La démarche que nous avons adoptée ci-dessus pour fixer le nombre d'appels engendrés pour chaque opération est critiquable : en effet, si une consultation de document éloigné, par exemple, exige un échange pour l'envoi de la demande de consultation et un échange pour l'envoi du document, elle peut (quoique cela soit moins élégant) susciter deux appels (avec création et fermeture de circuit virtuel) au lieu d'un comme nous l'avons supposé idéalement; dans ce dernier cas, un circuit virtuel est créé et maintenu pour toute la durée des échanges relatifs à cette demande. Inversément, la même consultation de document peut ne susciter elle-même aucun appel en bénéficiant d'un circuit virtuel créé vers le même réseau par une autre opération s'exécutant simultanément. Cela devient une question d'ouverture et

de fermeture de connexions, problème qu'il n'est pas toujours possible de contrôler. C'est pourquoi nous avons voulu l'éviter en considérant un seul appel, par groupes de messages échangés entre deux réseaux, pour une même opération.

Signalons enfin que, bien que le seuil de rentabilité permette de choisir le mode d'ouverture le plus avantageux, et ce, une fois la politique choisie, il ne permet toutefois pas de choisir la (ou les) politique(s) la (les) mieux appropriée(s) du point de vue des appels via le réseau public DCS, c'est-à-dire la politique la moins coûteuse à ce niveau.
Le point suivant doit, de ce point de vue, apporter une solution.

5.2.3. Le nombre de messages qui transitent par DCS.

5.2.3.1. Introduction.

Nous évaluons ici le nombre de messages, transitant par DCS, qui sont nécessaires pour cent exécutions de fonctions du courrier électronique. Cette évaluation est basée sur le programme implémenté conformément à l'architecture vue au chapitre précédent; elle est destinée à permettre, dans le point suivant, de quantifier le volume global des échanges sur DCS; c'est pourquoi nous donnons ici les nombres de messages selon le type de ceux-ci.

Les types de messages sont dus au choix que nous avons fait pour implémenter le dialogue entre les serveurs. Notre choix s'est porté sur les utilitaires de Unix: uucp et uux ([1]). Ces utilitaires sont destinés à lancer des communications en batch, et ne sont donc a priori pas appropriés au courrier électronique. Nous les avons néanmoins choisi pour leur simplicité d'utilisation, et parce que des tests nous ont permis de constater que les délais étaient négligeables dans le cadre d'un programme de simulation. Les types de messages sont : demande de consultation de boîte aux lettres, boîte aux lettres, réception de boîte aux lettres, demande de consultation de document, document, réception de document, liste de destinataires, réception d'un nouveau document, demande de suppression de document.

Avant de présenter la démarche qui sera suivie pour évaluer les nombres de messages sur DCS, posons une hypothèse: l'activité sur le courrier électronique est uniformément répartie. Ceci signifie que:

- à chaque serveur du courrier électronique est associé un même nombre de postes ou d'abonnés;

([1]) l'utilisation de uucp et uux est expliquée en annexe B.

- chaque abonné envoie, en moyenne, le même nombre de documents;

- il n'y a pas de phénomène de concentration sur les réseaux locaux.

Cette hypothèse nous permet de calculer des moyennes du nombre de messages sur DCS. Nous appelons **n** le nombre de serveurs du courrier électronique.

La démarche est la suivante: par politique, nous calculons les nombres moyens de messages échangés sur DCS pour chaque fonction du courrier électronique (consultation de boîte aux lettres, consultation de document, envoi de document, suppression de document).

Nous considérons que l'activité est du même type que celle définie au point 5.2.2., c'est-à-dire que sur 100 exécutions de fonctions du courrier électronique, la consultation de boîte aux lettres est exécutée 10 fois, la consultation de document est exécutée 50 fois, l'expédition de document est exécutée 15 fois et la suppression de document est exécutée 25 fois, en moyenne.

5.2.3.2. Les traitements utilisant DCS.

Les traitements qui nécessitent l'envoi de messages sur DCS sont:

- L'envoi d'une demande de consultation de boîte aux lettres à un serveur éloigné. Il provoque un message de type demande de consultation de boîte aux lettres, suivi d'un message de type boîte aux lettres et d'un autre de type réception de boîte aux lettres.

- L'envoi d'une demande de consultation de document à un serveur éloigné. Il provoque un message de type demande de consultation de document, suivi d'un message de type document et d'un autre de type réception de document.

- L'envoi d'un nouveau document à un serveur éloigné. Il provoque un message de type document, d'un message de type liste de destinataires et d'un autre de type réception de nouveau document.

- L'envoi d'une demande de suppression de document à un serveur éloigné. Il provoque un message de type demande de suppression de document.

A. Dans la politique 1.

La consultation de la boîte au lettres, activée 10 fois, consulte chacun des serveurs éloignés. Il y a donc 10(n-1) messages de type demande de consultation de boîte aux lettres, 10(n-1) messages de types boîte aux lettres et 10(n-1) messages de type réception de boîte aux lettres.

La consultation d'un document, activée 50 fois, n'envoie pas de message si le document se trouve dans le serveur local, sinon il y en a 50 de type demande de consultation de document, 50 de type document et 50 de type réception de document. En moyenne, il y en aura 50(1-1/n) pour chacun des trois types.

L'expédition d'un document, activée 15 fois, ne provoque, dans cette politique, aucun échange de messages sur DCS.

La suppression de document, activée 25 fois, provoque en moyenne 25(1-1/n) messages de type demande de suppression de document.

B. Dans la politique 3.

La consultation de la boîte au lettres, activée 10 fois, ne provoque, dans cette politique, aucun échange de messages sur DCS.

La consultation d'un document, activée 50 fois, ne provoque, dans cette politique, aucun échange de messages sur DCS.

L'expédition d'un document, activée 15 fois, provoque des envois de document vers chaque réseau destinataire. Nous avons vu au point 5.1.2. qu'il y a, en moyenne, [(n+2)/2]-1/2n documents stockés. Ceci signifie qu'il y a en moyenne ce nombre moins un -qui est stocké dans le serveur local- envois de documents vers des réseaux destinataires, ce qui fait (n/2)-1/2n. Il y a donc, en moyenne, 15[(n/2)-1/2n] messages de type document, 15[(n/2)-1/2n] messages de type liste de destinataires et 15[(n/2)-1/2n] messages de type réception d'un nouveau document.

La suppression de document, activée 25 fois, ne provoque, dans cette politique, aucun échange de messages sur DCS.

C. Dans la politique 4.

La consultation de la boîte au lettres, activée 10 fois, ne provoque, dans cette politique, aucun échange de messages sur DCS.

La consultation d'un document, activée 50 fois, ne provoque, dans cette politique, aucun échange de messages sur DCS.

L'expédition d'un document, activée 15 fois, provoque un envoi de nouveau document vers chaque serveur éloigné. Il y a donc, en moyenne, 15(n-1) messages de type document, 15(n-1) messages de type liste de destinataires et 15(n-1) messages de type réception d'un nouveau document.

La suppression de document, activée 25 fois, provoque l'envoi d'une demande de suppression dans chacun des serveurs éloignés. Il y a donc, en moyenne, 25(n-1) messages de type demande de suppression de document.

D. Dans la politique 5.

La consultation de la boîte au lettres, activée 10 fois, consulte le serveur du réseau de référence. Des messages sont échangés sur DCS si le réseau de référence

n'est pas le réseau local où s'effectue la consultation. Il y a donc, en moyenne, 10(1-1/n) messages de type demande de consultation de boîte aux lettres, 10(1-1/n) messages de types boîte aux lettres et 10(1-1/n) messages de type réception de boîte aux lettres.

La consultation d'un document, activée 50 fois, consulte le serveur du réseau de référence. Des messages sont échangés sur DCS si le réseau de référence n'est pas le réseau local où s'effectue la consultation. Il y a donc, en moyenne, 50(1-1/n) messages de type demande de consultation de document, 50(1-1/n) de type document et 50(1-1/n) de type réception de document.

L'expédition d'un document, activée 15 fois, provoque des échanges de messages sur DCS si le réseau de référence n'est pas celui où s'effectue l'expédition. Il y a donc, en moyenne, 15(1-1/n) messages de type document, 15(1-1/n) de type liste de destinataires et 15(1-1/n) de type réception d'un nouveau document.

La suppression de document, activée 25 fois, provoque, en moyenne, 25(1-1/n) messages de type demande de suppression de document.

E. Tableau récapitulatif.

Le tableau présenté ici reprend le nombre de messages échangés sur DCS par politique, par type de message, pour n réseaux locaux et pour 100 exécutions de fonctions du courrier électronique (10 cons. boîte aux lettres, 50 cons. document, 15 exp. document et 25 sup. document).

	Pol. 1	Pol. 3	Pol. 4	Pol. 5
dem cons bal	10(n-1)	0	0	10(1-1/n)
bal	10(n-1)	0	0	10(1-1/n)
rec bal	10(n-1)	0	0	10(1-1/n)
dem cons doc	50(1-1/n)	0	0	50(1-1/n)
document	50(1-1/n)	15[(n/2)-1/2n]	15(n-1)	65(1-1/n)
rec document	50(1-1/n)	0	0	50(1-1/n)
liste dest.	0	15[(n/2)-1/2n]	15(n-1)	15(1-1/n)
rec nouv doc	0	15[(n/2)-1/2n]	15(n-1)	15(1-1/n)
dem sup doc	25(1-1/n)	0	0	25(1-1/n)

Les résultats de ce tableaux sont utilisés dans le point suivant pour calculer le volume des échanges selon les politiques.

5.2.3. Les volumes échangés.

Nous sommes donc maintenant en mesure d'évaluer plus précisément les coûts relatifs aux volumes de données transférées par l'intermédiaire du courrier électronique. Cette évaluation est primordiale, car, après l'évaluation des coûts relatifs aux appels, elle est la deuxième étape dans l'évaluation globale des coûts imputables à DCS.

Pour commencer, il est nécessaire de fixer les ordres de grandeur des différents types de messages échangés.

Ainsi, une consultation de boîte aux lettres va nécessiter 3 traitements : une demande de consultation de boîte aux lettres, un envoi de la boîte aux lettres et une réception de celle-ci. La demande de consultation va consister en un message de **25** bytes d'information utile; l'envoi de la boîte aux lettres, en environ **80** bytes, si une boîte aux lettres compte en moyenne 5 références (on peut estimer à **25** bytes environ la taille d'une boîte aux lettres partielle). La réception d'une boîte aux lettres compte, pour sa part, une **trentaine** de bytes.

De même, une consultation de document va entraîner 3 traitements : tout d'abord, la demande de consultation de document consistant en quelques **40** bytes d'information utile. Ensuite, l'envoi du document, qui risque d'être beaucoup plus long : en effet, si l'on suppose qu'un document a une longueur d'une page de texte (nous nous situons dans une moyenne acceptable), cela correspond environ à 1500 caractères, soit **1500** bytes. Enfin, le troisième traitement, la réception du document, compte également environ **40** bytes.

De plus, lors de l'envoi d'un nouveau document, il faut encore ajouter la liste des destinataires de ce document, c'est-à-dire quelques identifiants d'abonnés; fixons cela à une vingtaine de bytes en moyenne, pour l'envoi de cette liste (une **quinzaine** pour des listes partielles de la politique 3) et à environ **60** bytes la réception d'un nouveau document.

Une demande de suppression de document sera du même calibre puisqu'elle consistera en environ 35 bytes (le détail de tous ces nombres est expliqué en annexe !).

Plusieurs aspects devront être pris en compte pour déterminer exactement les volumes transférés pour chaque politique : outre les volumes relatifs aux données utiles transférées à l'intérieur des paquets et que nous venons de mentionner, il faudra également tenir compte des données nécessaires à la confection des paquets qui seront transférés par DCS, et des données relatives au dialogue entre les réseaux, lors de la création et de la fermeture des circuits virtuels et ainsi que lors de l'utilisation de uucp et uux.

Détaillons donc la composition des volumes échangés sur DCS : la création d'un circuit virtuel via DCS consiste en 2 paquets de dialogue initial (appel et confirmation), soit 2 segments à comptabiliser lorsqu'on effectue un appel. La fermeture de ce circuit coûte 1 paquet de libération (1 segment). L'utilisation de uucp ou uux suscite également des échanges "inutiles" pour l'utilisateur : on compte pour cela 3 échanges séparés de 6 caractères, soit trois paquets (au mieux dans un segment) à comptabiliser; de même, toute trame d'information passée par uucp ou uux comporte une entête de 6 bytes propre à ces utilitaires. Par ailleurs, un paquet de données est également composé de quelques bytes d'emballages, soit 5 bytes environ.

<u>En résumé</u> : la création (et fermeture) d'un circuit virtuel via DCS coûte l'équivalent de 3 segments; l'envoi d'une information grâce à uucp ou uux nécessite l'envoi de 1 segments et tout paquet de données, d'une longueur inférieure à 128 bytes, se voit imputer de 5 bytes "d'emballage" et de 6 bytes pour un header imposé par uucp et uux !
Dès lors, l'envoi d'un paquet de 64 bytes (c'est-à-dire le nombre maximum de bytes pour un segment) par DCS va permettre d'envoyer au plus 53 (64-11) bytes utiles, et l'envoi d'un paquet de 127 bytes, longueur maximale

autorisée d'un paquet sur DCS, au maximum 116 (127-11) bytes utiles.

Il serait possible, à partir d'ici, de détailler les coûts relatifs à chacune des fonctions et ce, pour chaque politique. Néanmoins cela n'offre pas beaucoup d'intérêt si ces coûts ne sont pas rassemblés pour permettre une comparaison des politiques sur leurs volumes globaux échangés. C'est pourquoi, en se basant sur les résultats obtenus au point 5.2.3. ci-dessus, il nous est possible de déterminer le volume total des données échangées dans chaque politique, à la fois pour chaque opération et globalement, cela pour un volume d'activité de 100 opérations.

Notons, pour la compréhension des prochains calculs, que tout appel, lors de sa clotûre, se verra facturer le volume des données échangées par unités de 10 segments de 64 bytes.

Pour la politique 1, nous retenons donc que :

* l'on a 10 consultations de boîtes aux lettres, ce qui nécessite 10*(N-1) appels à des réseaux éloignés, chaque appel comprenant la création du circuit virtuel, l'envoi d'une demande de consultation de boîte aux lettres (uux), l'envoi de la boîte aux lettres (uucp), la réception de celle-ci (uux), et la fermeture du circuit virtuel. En d'autres mots, un appel pour consultation de boîte aux lettres coûte donc : 2 segments + (25+11) bytes + (25+11) bytes+ (30+11) bytes + 1 segment + 3 segments (uucp-uux) = (2+1+1+1+1+3) ou 9 segments. Dans la politique 1 on a donc à payer **10*(N-1)*0,2** fr pour la consultation de boîte aux lettres selon l'activité donnée, puisque 0,2 fr est le coût de 10 segments envoyés.
* l'on a 50 consultations de documents qui exigent 50*(1-1/N) appels avec des réseaux éloignés, chaque appel comprenant la création du circuit virtuel, l'envoi d'une demande de consultation de document (uux), l'envoi du document (uucp), la réception de celui-ci (uux), et la

fermeture du circuit virtuel. En d'autres mots, un appel pour consultation de document donc : 2 segments + (40+11) bytes + (1524+154)* bytes + (40+11) bytes + 1 segment + 3 segments (uucp-uux) = (2+1+27+1+1+3) ou 35 segments. Dans la politique 1 on a donc à payer **50*(1-1/N)*(4*0,2)** fr pour la consultation de documents selon l'activité donnée.

(* : les 1524 bytes du document doivent être répartis dans une série de paquets; il faut donc considérer autant d'entêtes qu'il y faut de paquets; rappelons qu'un paquet peut atteindre une taille maximale de 127 bytes pour DCS)

* l'on a aucune expédition de document nécessitant un appel vers d'autres réseaux;
* l'on a 25 suppressions de documents qui exigent 25*(1-1/N) appels à des réseaux éloignés, chaque appel nécessitant une ouverture de circuit virtuel, l'envoi de la demande de suppression (uux) et la fermeture du circuit virtuel. Soit un volume de : 2 segments + (35+11) bytes + 1 segment + 1 segment (uux) = (2+1+1+1) ou 5 segments. Ce qui entraîne un coût global de **25*(1-1/N)*0,2** fr pour les suppressions de documents relatifs à l'acitvité donnée.

En définitive, le coût global des volumes échangés dans la politique 1 pour une activité de 100 opérations peut être évalué à : 10*(N-1)*0,2 + 50*(1-1/N)*(4*0,2) + 25*(1-1/N)*0,2 = **2*(N-1)+45*(1-1/N)** francs.

Pour la politique 3, nous retenons donc que :

* l'on a aucune consultation de boîte aux lettres engendrant des appels vers des réseaux éloignés;
* l'on a aucune consultation de document engendrant des appels vers des réseaux éloignés;
* l'on a 15 expéditions de documents qui nécessitent 15*[(N/2)-1/2N] appels vers des réseaux éloignés, chaque appel étant composé de l'ouverture d'un circuit virtuel, de l'envoi du document (uucp), de l'envoi d'une liste partielle des destinataires (uucp), de la réception de ces informations (uux) et de la fermeture de la ligne. Cela représente donc un volume équivalent à : 2 segments + (1524+154) bytes + (15+11) bytes + (55+11) bytes + 1

segment + 3 segments (uucp-uux) = (2+27+1+2+1+3) ou 36 segments. Les données envoyées dans la politique 3 pour les expéditions de documents s'élèvent donc à **15*[(N/2)-1/2N] * (4*0,2)** fr;

* l'on a aucune suppression de document qui exige un appel à un réseau éloigné.

En définitive, le coût global des volumes échangés dans la politique 3 pour une activité de 100 opérations peut être évalué à : 15*[(N/2)-1/2N]*(4*0,2) francs ou encore à **12*[(N/2)-1/2N]** francs.

Pour la politique 4, nous pouvons retenir que :

* l'on a aucune consultation de boîte aux lettres engendrant des appels vers des réseaux éloignés;
* l'on a aucune consultation de document engendrant des appels vers des réseaux éloignés;
* l'on a 15 expéditions de documents qui nécessitent 15*(N-1) appels vers des réseaux éloignés, chaque appel étant composé de l'ouverture d'un circuit virtuel, de l'envoi du document, de l'envoi de la liste des destinataires, de la réception de ces informations et de la fermeture de la ligne. Cela représente donc un volume équivalent à : 2 segments + (1524+154) bytes + (20+11) bytes + (55+11) bytes + 1 segment + 3 segments (uucp-uux) = (2+27+1+2+1) ou 36 segments. Les données envoyées dans la politique 4 pour les expéditions de documents s'élèvent donc à **15*(N-1)* (4*0,2)** fr;
* l'on a aucune suppression de document qui exige un appel à un réseau éloigné.

En définitive, le coût global des volumes échangés dans la politique 4 pour une activité de 100 opérations peut être évalué à : 15*(N-1)*(4*0,2) = **12*(N-1)** francs.

Pour la politique 5, nous pouvons retenir que :

* l'on a 10 consultations de boîtes aux lettres, ce qui nécessite 10*(1-1/N) appels à des réseaux éloignés, chaque appel comprenant la création du circuit virtuel, l'envoi d'une demande de consultation de boîte aux lettres, l'envoi de la boîte aux lettres, la réception de celle-ci, et la fermeture du circuit virtuel. En d'autres mots, un appel pour consultation de boîte aux lettres

coûte donc : 2 segments + (25+11) bytes + (25+11) bytes + (30+11) bytes + 1 segment + 3 segments (uucp-uux) = (2+1+1+1+1+3) ou 9 segments. Dans la politique 5 on a donc à payer **10*(1-1/N)*0,2** fr pour la consultation de boîte aux lettres selon l'activité donnée.

* l'on a 50 consultations de documents qui exigent 50*(1-1/N) appels vers des réseaux éloignés : cela donne donc le même résultat que dans la politique 1, à savoir, **50*(1-1/N)*(4*0,2)** fr pour la consultation de documents selon l'activité donnée.

* l'on a 15 expéditions de documents qui nécessitent 15*(1-1/N) appels vers des réseaux éloignés, chaque appel étant composé de 36 segments, comme dans la politique 4. Les données envoyées dans la politique 5 pour les expéditions de documents s'élèvent donc à **15*(1-1/N)*(4*0,2)** fr;

* l'on a, comme pour la politique 1, 25 suppressions de documents qui exigent 25*(1-1/N) appels à des réseaux éloignés. Ce qui entraîne également un coût global de **25*(1-1/N)*0,2** fr pour les suppressions de documents relatifs à l'acitvité donnée.

En définitive, le coût global des volumes échangés dans la politique 5, pour une activité de 100 opérations, peut être évalué à : 10*(1-1/N)*0,2 + 50*(1-1/N)*(4*0,2) + 15*(1-1/N)*(4*0,2) + 25*(1-1/N)*0,2 = (2+40+12+5)*(1-1/N) = **59*(1-1/N)** francs.

Le graphique suivant permet d'ailleurs de comparer les coûts relatifs au volume de données échangées dans les différentes politiques.

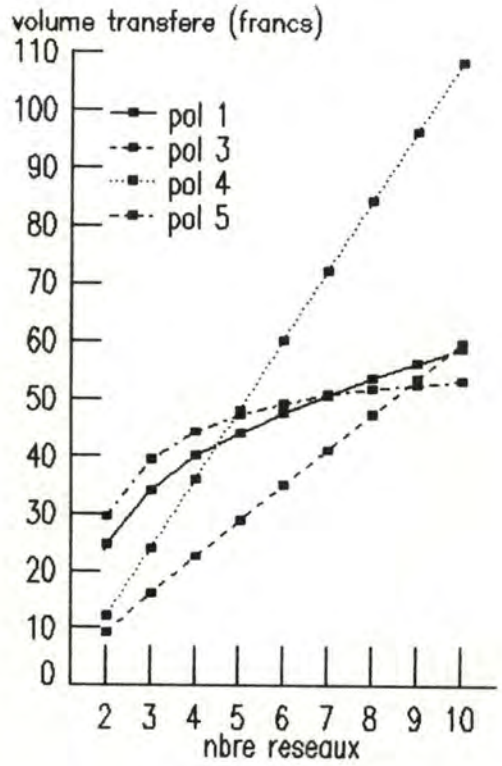

fig 5.4 : les coûts relatifs aux volumes transférés.

Nous pouvons remarquer que, pour les coûts relatifs aux volumes transférés, la politique 3 se révèle particulièrement intéressante puisqu'elle reste pratiquement tout le temps la meilleure; ce n'est qu'avec un nombre élevé de réseaux (à partir de 9) qu'elle est dépassée par la politique 5, puis par la politique 1. Nous pouvons également remarquer deux autres caractéristiques importantes : la première est l'intérêt de la politique 5 dans un système comprenant beaucoup de réseaux, car cette politique est bornée par un coût de 0,59 fr par opération, quel que soit le nombre de réseaux. La deuxième caractéristique est le comportement désastreux de la politique 4 qui, dès l'apparition d'un cinquième réseau, se révèle être définitivement la moins avantageuse des politiques. Enfin, la politique 1 se comporte de façon moyenne par rapport aux autres politiques.

Notons qu'après ce qui vient d'être dit et en se rappelant les propos tenus au chapitre ***, nous pouvons imaginer qu'une technique d'ouverture permanente de

circuit virtuel permettrait de réaliser des gains considérables au niveau des coûts liés aux volumes transférés. Cette économie serait proportionnelle au nombre d'opérations nécessitant des appels vers d'autres réseaux, suivant la politique où l'on se trouve, et au nombre de segments relatifs à l'ouverture et à la fermeture d'un circuit virtuel. Inversément, on pourrait parler de perte supplémentaire liée à la technique d'ouverture ponctuelle et il serait alors pensable de compter cette perte dans le côut-même de l'établissement d'un appel, modifiant aussi, par la même occasion, les seuils de rentabilités obtenus ci-dessus.

5.2.5. Evaluation globale des politiques vis-à-vis des coûts globaux de DCS.

Une évaluation des politiques vis-à-vis des coûts globaux de DCS doit nécessairement comporter un regroupement des coûts relatifs aux appels (et à la durée) et des coûts relatifs aux volumes, tous calculés antérieurement. Pour fournir cette évaluation globale, nous regroupons ces données sous une même activité de 100 opérations.

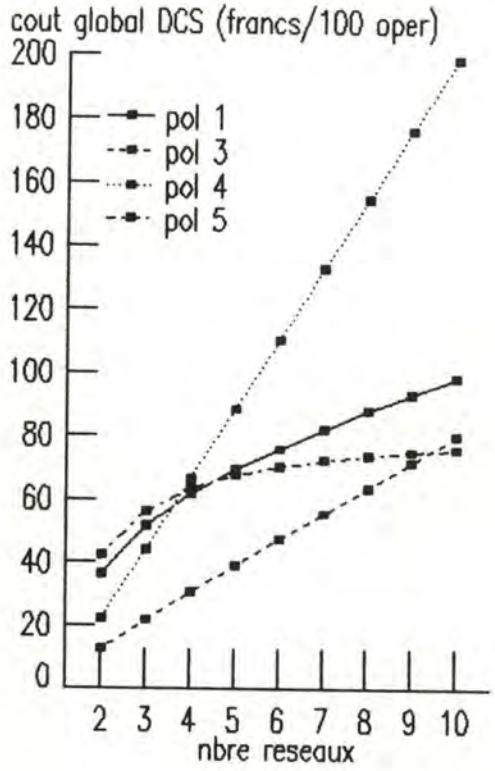

fig. 5.5. : les coûts globaux de DCS.

Pour cette évaluation globale des coûts imputables à DCS, plusieurs points sont à souligner. Tout d'abord le très bon comportement de la politique 3, qui reste la meilleure jusqu'à un système composé de 9 réseaux. Ensuite, le comportement identique et relativement moyen des politiques 1 et 5, jusqu'à un niveau de 5 réseaux, seuil à partir duquel la politique 1 continue une progression normale, tandis que la politique 5 s'améliore considérablement et tend même à suplanter la politique 3, dès l'apparition d'un dixième réseau. Notons, à ce point de vue, que la politique 5 sera bornée supérieurement par un coût de 84 francs (59 pour le volume et 25 pour les appels), quel que soit le nombre de réseaux; ce qui devient très intéressant pour un niveau élevé de réseaux. Enfin, nous pouvons remarquer le comportement désastreux de la politique 4, qui, bien qu'elle soit acceptable pour des systèmes très petits (2-4 réseaux), ne peut être envisagée pour des systèmes plus étendus (à partir de 5 réseaux): nous voyons notamment que, pour 10 réseaux, elle est de 2 à 3 fois plus onéreuse que les autres politiques.

Remarquons enfin, comme c'est généralement le cas, que le coût lié au volume des données transférées représente la majeure partie du coût global relatif à DCS.

5.3. Le temps de consultation d'un document.

Pour évaluer le temps nécessaire à la consultation d'un document, nous commençons par décomposer cette consultation en ses diverses étapes. Nous considérons d'abord la consultation la plus complexe, c'est-à-dire celle qui nécessite le passage par DCS.

1) Le temps d'attente au poste consultant, pour que la demande passe sur le réseau local Ethernet: nous estimons que ce temps est très court, car le temps d'accès à Ethernet est presque immédiat [Sho.-Hupp 80].

2) Le temps de propagation et le temps de transmission sur Ethernet: nous estimons que ces temps sont très courts, étant donné les caractéristiques techniques d'Ethernet.

3) Le temps d'attente pour traitement et le temps de traitement dans le serveur local: nous estimons que ces temps sont très courts, car le serveur est un ordinateur relativement puissant.

4) Le temps d'attente dans le serveur local pour le passage sur DCS: ce temps ne peut être évalué, car le système que nous avons utilisé (uucp-uux) est un sytème orienté batch. Nous ne pouvons, cependant, pas négliger ce temps. En effet, même en utilisant une procédure temps réel, il peut y avoir un délai si, par exemple, il faut établir une connexion.

5) Le temps de propagation à travers DCS: ce temps est très court. On néglige, en général, ce temps pour les liaison terrestres [Mac.-Gui. 83 p.176].

6) Le temps de transmission de la demande sur DCS: la longueur de la demande est, comme nous l'avons vu au point 5.2.4. ci-dessus, de 56 bytes. L'endroit le moins rapide, dans la transmission par DCS est entre l'ordinateur et le noeud d'entrée, la vitesse de transmission y est de 9600 bps. Il faut donc 448/9600 = 0,05 secondes pour transmettre la demande par DCS.

7) Le temps d'attente pour traitement et le traitement de la demande dans le serveur éloigné: nous estimons que ces temps sont très courts, car le serveur est un ordinateur relativement puissant.

8) Le temps d'attente dans le serveur éloigné pour le passage sur DCS: nous faisons ici la même remarque que pour le point 4).

9) Le temps de propagation à travers DCS: nous pouvons négliger ce temps, pour la même raison qu'au point 5).

10) Le temps de transmission du document sur DCS: Si on estime la longueur moyenne d'un document à 1500 bytes, le nombre moyen de bytes nécessaire pour transférer un document par DCS est de 1740 bytes environ, comme nous l'avons vu au point 5.2.4. ci-dessus. A la vitesse de transmission de 9600 bps, il faut 13920/9600 = 1,45 secondes auxquelles nous rajoutons 0,05 secondes pour le message de réception de document qui est de même nature que la demande. Il faut donc, en moyenne, 1,5 secondes pour que le document revienne.

11) Le temps d'attente pour le traitement et le traitement du document par le serveur local: nous estimons que ces temps sont très courts, car le serveur est un ordinateur relativement puissant.

12) Le temps d'attente dans le serveur local, pour que le document passe sur le réseau local Ethernet: nous estimons que ce temps est très court, car le temps d'accès à Ethernet est presque immédiat [Sho.-Hupp 80].

13) Le temps de transmission et de propagation sur Ethernet: nous estimons que ces temps sont très courts, étant donné les caractéristiques techniques d'Ethernet.

14) Le temps de traitement de la réception du début du document par le poste consultant: nous estimons que ce temps est très court, car le poste consultant est un micro-ordinateur totalement dédicacé à cette tâche.

Nous avons observé que lorsque le document est stocké dans le serveur local, ce qui constitue la consultation la plus simple, le temps de réponse est quasi immédiat, c'est-à-dire inférieur à une seconde. Ce temps comprend les temps intermédiaires 1), 2), 3), 11), 12), 13) et 14). Notre estimation selon laquelle ces temps étaient tous très courts a donc été confirmée par l'observation.

En ce qui concerne la consultation plus complexe qui nécessite le passage par DCS, nous constatons que les temps intermédiaires qui sont importants sont 4), 6), 8) et 10). Ces temps sont, au total, d'au moins 1,55 secondes.

En conclusion, nous pouvons dire que les politiques 3 et 4, où un document se trouve toujours dans le serveur local de l'abonné consultant, sont très efficaces du point de vue du temps de consultation d'un document. Les politiques 1 et 5, lorsque le document n'est pas stocké dans le serveur local de l'abonné consultant, peuvent donner un temps de consultation de document tout à fait acceptable; il faut, cependant, rester prudent à cause du délai possible d'accès à DCS.

Chapitre 6 : Conclusion.

6.1. Idées maîtresses.

L'objectif de notre mémoire était d'étudier la mise en oeuvre d'un courrier électronique sur un ensemble de réseaux différents. Nous avons choisi de partir d'un courrier électronique existant sur un réseau local et d'envisager les extensions et les modifications à y apporter de sorte qu'il puisse prendre en charge le courrier interne au réseau local et le courrier entre réseaux locaux via un réseau externe, et ceci de manière transparente à l'utilisateur.

Nous avons opté pour DCS, comme réseau longue portée reliant les différents réseaux locaux. Nous avons défini un courrier électronique simplifié pour orienter notre étude sur les problèmes engendrés par la multiplication des réseaux locaux et par le transport via DCS.

Nous avons ensuite défini un ensemble de politiques relatives à la répartition de la base de données du courrier électronique entre les serveurs des réseaux locaux. Nous avons aussi défini des critères d'évaluation des politiques. Nous avons immédiatement procédé à une première évaluation des politiques; cette évaluation n'était que qualitative.

Pour obtenir une évaluation quantitative, nous devions implémenter nos politiques. C'est pourquoi, après avoir considéré les politiques 2 et 6 comme des cas particuliers des politiques 3 et 5, nous avons déterminé les traitements nécessaires pour exécuter les fonctions principales du courrier électronique dans les politiques 1, 3, 4 et 5. Nous avons alors proposé une architecture logique valable quelle que soit la politique. Le programme que nous avons implémenté à partir de cette

architecture nous a permis d'évaluer quantitativement les facteurs de coût et de temps qui interviennent lors de l'utilisation du courrier électronique étendu.

Les évaluations quantitative et qualitative que nous avons faites ne nous permettent pas de classer les politiques entre elles parce que les tendances dégagées par certains critères sont souvent infirmées par d'autres. Le concepteur d'un courrier électronique étendu doit choisir une politique en fonction de son comportement par rapport à un sous-ensemble de critères qu'il juge les plus importants. C'est pourquoi nous faisons ici une synthèse rapide de chaque politique en reprenant les différents éléments d'évaluation qui ont été abordés.

La *politique 1* met l'accent sur l'unicité des documents; ceci rend possible la modification, et le volume mémoire nécessaire est minimal. Les temps de consultation varient selon l'endroit où est stocké le document recherché: il faut les surveiller. Au niveau DCS, nous considérons cette politique comme "moyenne" tant pour les coûts liés au volume des échanges que pour les coûts liés au nombre d'appels.

La *politique 3* a comme caractéristique principale la multiplicité des documents qui rend les temps de consultation très faibles. Cette politique a comme désavantage fonctionnel la localisation obligatoire des abonnés. Au niveau DCS, par contre, cette politique est tout à fait performante, car elle est la meilleure tant pour les coûts dus au volume des échanges que pour les coûts dus au nombre d'appels, et ceci jusque neuf réseaux locaux.

La *politique 4* est aussi caractérisée par la multiplicité des documents. Cette politique est très simple, car elle ne nécessite aucun traitement de localisation d'abonnés. Au niveau DCS, elle est la plus coûteuse à tout point de vue.

La *politique 5* est une politique de centralisation, avec l'unicité des documents. Les temps de consultations sont toujours les mêmes: faible sur le réseau local de référence et plus important (à surveiller) sur les autres réseaux locaux. Au niveau du trafic DCS, cette politique devient très intéressante lorsque le nombre de réseaux locaux est élevé.

6.2. Les limites de notre travail.

Nous discutons ici de certains points qui nous paraissent être des limites à notre travail. Ces limites sont: l'hypothèse de non concentration des destinataires d'un document sur le réseau local de création, les valeurs de certaines données et l'applicabilité du mémoire dans une application réelle.

L'hypothèse de non-concentration des destinataires d'un document sur le réseau local de création. Cette hypothèse facilite les calculs, mais elle n'est pas nécessairement réaliste. En effet, les documents échangés à travers un courrier électronique peuvent être envoyés vers des abonnés du même réseau local; abonnés avec lesquels on a probablement plus de rapports et donc de messages à échanger. Une certaine concentration est donc très probable. Les politiques 1 et 3 s'en trouveraient directement influencées : la politique 1, lors des consultations et suppressions de documents, et la politique 3, pour les consultations de documents. Cette dernière verrait encore chuter le nombre de ses transferts de documents.

Cependant, il n'est pas possible d'envisager cette hypothèse dans tous les cas : ainsi, dans la politique 4, celle-ci ne change rien et, dans la politique 5, elle concerne uniquement les abonnés du réseau de référence. Par ailleurs, on peut dire que si les abonnés ne sont pas localisés sur un réseau déterminé la répartition des destinataires d'un document sur les différents réseaux du système n'a plus beaucoup de sens.

Les valeurs de certaines données peuvent être, en pratique, sensiblement différentes de celles choisies dans le chapitre 5. Ainsi, pour les pourcentages de répartition des opérations, un nombre plus réduit (ou au contraire plus grand) de destinataires pour un document aurait pour effet de diminuer (accroître) les pourcentages relatifs aux consultations et aux suppressions de documents. Néanmoins, nous avons essayé de refléter une activité moyenne probable d'un courrier électronique tel que le nôtre, sans chercher à rendre plus complexe notre étude par l'introduction d'un paramètre supplémentaire.

L'applicabilité du mémoire dans une application réelle. Nous n'avons pas réalisé un courrier électronique utilisable tel quel. Nous avons étudié les avantages et inconvénients de différentes politiques relatives au stockage des documents en circulation dans un courrier électronique. Ceci peut être utile là où une mémorisation de documents s'avère nécessaire. Notre étude permet, en outre, d'avoir une vue immédiate des implications d'une politique de stockage tant sur des critères fonctionnels, que sur des critères de performance relatifs à un système informatique, en général, et à un courrier électronique, en particulier.

6.3. Les extensions possibles.

Nous pensons qu'un courrier électronique sur plusieurs réseaux crée de nouveaux problèmes de concurrence. Nous avons laissé à Unix le soin de gérer ces problèmes, mais une étude plus détaillée des risques (de deadlock, par exemple) et des remèdes possibles pourrait être très profitable. De même, les accès aux processeurs ne sont pas analysés en détail et ils devraient faire également l'objet d'une étude plus approfondie pour permettre de conforter les conclusions relatives au temps de réponse abordé au chapitre 5.

Les problèmes de sécurité peuvent être aussi abordés plus en détail. On peut cerner plus précisément des questions comme la protection locale des documents,

leur protection lors des transferts, la protection d'accès au courrier électronique, toutes questions auxquelles nous avons peu ou pas répondu.

Il est possible de rendre optimal le dialogue sur DCS. En effet, celui que nous avons implémenté se base principalement sur des utilitaires existants dans Unix (UUX et UUCP) et n'est pas optimal en terme de nombre d'échanges de messages réalisés. Une optimisation de ce dialogue pourrait amener le courrier électronique à réaliser de sérieuses économies de transferts et de ce fait confronter certaines politiques du point de vue de certains critères de performance, comme le temps de réponse ou le coût DCS.

La dernière extension que nous proposons est un système d'aide au choix d'une politique. Nous pensons à un système qui proposerait une politique selon les spécifications fonctionnelles et de performances que pourrait donner le concepteur d'un courrier électronique sur plusieurs réseaux.

Bibliographie .

[Ber.-Jos. 85] Jean-Marie Bernard & Alain Josis (1985), *Réalisation d'un courrier électronique. Implèmentation sur un réseau local*, FNDP, Namur

[Cor. 81] Cornafion, *Systèmes Informatiques Répartis : Concepts Techniques* (p.151-271) Dunod 1981.

[Def. 86] Ph. Defert (6 february 1986), *Communications Software Survey for a European Astronomical Network.*

[Flint] David D. Flint, *The Data Ring Main. An Introduction to Local Area Networks*, Computing Sciences Series.

[Heag. 86] Denise Heagerty (18 february 1986), *A User Guide to Electronic Mail Services at CERN.*

[Mac.-Gui. 83] G. Macchi & J.-F. Guilbert (1983), *Télèinformatique*, Dunod, Paris

[Op.-Dal. 83] Derek D. Oppen & Yogen K. Dalal (July 1983), *"The Clearinghouse: A Decentralized Agent for Locating Named Objects in a Distributed Environment", ACM transactions on Office Information Systems*, (1, 3), pp. 230-253.

[Rad. 84] S. Radicati (July 1984), *"Managing Transient Internetworks Links in the Xerox Internet", ACM Transactions on Office Systems*, (2, 3), pp. 213-225.

[Red.-Whi. 83] David D. Redell & James E. White (September 1983), *"Interconnecting Electronic Mail Systems"*, *IEEE*, pp. 55-63.

[Sho.-Hupp 80] John F. Shoch & Jon A. Hupp (December 1980), *"Measured Performance of an Ethernet Local Network"*, *Communications of the ACM*, (23, 12), pp 711-720.

[X.400] Draft Recommandation X.400 , *Message Handling Systems : System Model-Service Elements*.

Annexe A : notre courrier électronique étendu.

```
/**************************/
 /* Déclarations des types */
 /**************************/


#define LAB 8 /* Longueur maximale d'un abonné */
#define MAXL 80 /* Longueur maximale d'une ligne de document
*/
#define TRUE 1
#define FALSE 0


/* une stucture date est composée d'une année, d'un mois et
d'un jour : cette structure date représente une date */

typedef struct date
     {int year;
      int month;
      int day;
     };


/* la structure time est composée d'une heure, d'une minute et
d'une seconde données : cette structure time représente un
instant donné */

typedef struct time
     {int hour;
      int minute;
      int second;
     };


/* une structure idd est composée d'une structure time, d'une
structure date et d'un identifiant d'abonné : cette structure
idd permet d'identifier un document, l'identifiant d'abonné
référençant le créateur du document */

typedef struct idd
     {struct time crtime;
      struct date crdate;
      int ida;
     };


/* une structure line est composée d'une chaîne de caractères
et d'une référence à la stucture line suivante : cette
structure représente une ligne de document */

typedef struct line
     {char l[MAXL];
      struct line *nextl;
     };


/* une structure lbal est composée d'une structure idd et
d'une référence à la structure lbal suivante : cette structure
représente un document figurant dans une boîte aux lettres */
```

```
typedef struct lbal
     {struct idd iddoc;
      struct lbal *nextiddoc;
     };


/* une structure llnw est composée d'un identifiant d'abonné
et d'une référence à la structure llnw suivante : cette
structure représente un abonné figurant dans une liste de
destinataires */

typedef struct llnw
     {int dest;
      struct llnw *nextdest;
     };


/* une structure lien est composée d'une structure idd et d'un
identifiant d'abonné : cette structure lien permet d'établir
une relation entre un abonné et un document, dont l'abonné est
destinataire */

typedef struct lien
     {struct idd iddoc;
      int ida;
     };


/* une structure local est composée d'un identifiant d'abonné
et d'un identifiant de réseau : cette structure permet de
localiser un abonné sur un réseau */

typedef struct local
     {int ida;
      int idnw;
     };


/* une structure routnw est composée d'une référence à la
première structure llnw, d'une référence à la dernière
structure llnw, d'une structure idd, d'un identifiant de
réseau et d'une référence à la structure routnw suivante :
cette structure représente la liste des destinataires d'un
document donné à envoyer vers un réseau donné */

typedef struct routnw
        {struct llnw *firstdest;
         struct llnw *lastdest;
         struct idd iddoc;
         int idnw;
         struct routnw *nextl;
        };

struct routnw *firstl,*lastl;
FILE *fich,*fopen();
int ida,idab,idabr,idnw;
char nameab[LAB];
struct line *cd;
struct lbal *bal;
struct idd idvide,iddocr;
```

```
                    /**********************/
                     /* MODULE SEQUENCEUR */
                     /**********************/


/*************************************************************
*
Le séquenceur est le programme principal; il synchronise les
appels appropriés aux différentes interfaces.
*************************************************************
/

#include <stdio.h>
#include "declarations.c"
#include "fonclien.c"
#include "util.c"
#include "utilbis.c"
#include "document.c"
#include "e-s.c"
#include "mutilisateur.c"
#include "bal.c"
#include "mroutage.c"
#include "trtlogfi.c"

main ()
{    int fin, choix;

     fin = 0;
     affichmess (1);
     lectida (&ida);
     if (politique()==3)
     if (flocalisation(ida)!=dcsaddr())
     {    affichmess(5);
          fin=1;};
     while (fin == 0)
     {    choix = affichmenu (2);
          switch (choix)
          {    case 1 : consbal ();
                        break;
               case 2 : consdoc ();
                        break;
               case 3 : creexpdoc ();
                        break;
               case 4 : supdocu ();
                        break;
               case 0 : fin = 1;
                        break;
          }
     }
}


/*************************************************************
*
La procédure 'consbal' réalise la consultation de la boîte aux
lettres.
*************************************************************
/

consbal ()
```

```
{    struct lbal bal;

     if ((politique () == 1) || (politique () == 6))
     {    ecrattente (ida);
          envoidcbal (ida);
          attente (ida);
     }
     else envoidcbal (ida);
     bal.iddoc = idvide;
     consultbal (ida, &bal);
     affichbal (bal);
     stop ();
}


/*****************************************************************
*
La procédure 'consdoc' réalise la consultation d'un document.
*****************************************************************
/

consdoc ()
{    struct idd ident;
     struct line corps;
     struct lbal bal;

     bal.iddoc = idvide;
     consultbal (ida, &bal);
     cls();
     affichbal (bal);
     affichmess (2);
     lectiddoc (&ident);
     if ((politique () == 1) || (politique () == 6))
        {       ecrattente (ida);
          envoidcdoc (ident, ida);
          attente (ida);
     }
     else envoidcdoc (ident, ida);
     if (cherchedoc (ident, &corps) == 1)
          affichdoc (ident, corps);
     else affichmess (7);
     stop ();
}


/*****************************************************************
*
La procédure 'supdocu' réalise la suppression d'un document.
*****************************************************************
/

supdocu ()
{    struct idd ident;
     struct lbal bal;

     bal.iddoc = idvide;
     consultbal (ida, &bal);
     cls();
     affichbal (bal);
     affichmess (6);
```

```
     lectiddoc (&ident);
     envoidsup (ident,ida);
}


/*****************************************************************
*
La procédure 'creexpdoc' réalise la création et l'expédition
d'un document.
*****************************************************************
/

creexpdoc ()
{    int fin, choix, corpslu, ldestlu;
     struct idd ident;
     struct llnw ldest;
     struct line corps;

     fin = 0;
     corpslu = 0;
     ldestlu = 0;
     while (fin == 0)
     {    choix = affichmenu (3);
          switch (choix)
          {    case 1 : lectcdoc (&corps);
                        corpslu = 1;
                        break;
               case 2 : lireldest (&ldest, ida);
                        ldestlu = 1;
                        break;
               case 3 : if ((corpslu == 1) && (ldestlu == 1))
                         {   temps (&ident);
                             ident.ida=ida;
                             envoidoc (ident,&ldest, corps, ida);
                         }
                         else {   affichmess (3);
                                  stop ();
                               };
                         break;
               case 0 : fin = 1;
                        break;
          }
     };
}


/*****************************************************************
*
La procédure 'lireldest' permet de lire au terminal une liste
de destinataires.
*****************************************************************
/

lireldest (ldest, idab)
struct llnw *ldest;
int idab;
{    int fin, ida;
     struct llnw *inter, *dernier;

     fin = 0;
```

```
        inter = ldest;
        affichmess (4);
        while (fin == 0)
        {    lectida (&ida);
             if (ida != 0)
             {    inter->dest = ida;
                  dernier = inter;
                  inter->nextdest = malloc (sizeof (*inter));
                  inter = inter->nextdest;
             }
             else {    dernier->nextdest->dest = idab;
                  dernier->nextdest->nextdest = 0;
                  fin = 1;
                  }
        };
}


/*****************************************************************
*
La procédure 'temps' met à jour les parties date et heure d'un
identifiant de document.
*****************************************************************
/

temps (ident)
struct idd *ident;
{    int temps, taffich, annee, mois;

        temps = time (0);
        taffich = temps % 60;
        ident->crtime.second = taffich;
        temps /= 60;
        taffich = temps % 60;
        ident->crtime.minute = taffich;
        temps /= 60;
        temps -= 8;
        taffich = temps % 24;
        ident->crtime.hour = taffich;
        temps /= 24;
        annee = 1970;
        while (temps > 364)
        {    if (annee % 4 != 0)
             temps -= 365;
             else temps -= 366;
             annee++;
        };
        ident->crdate.year = annee;
        mois = 1;
        while (temps > 30)
        switch (mois)
        {    case 1 : m31 (&temps, &mois); break;
             case 2 : if (annee % 4 != 0) temps -= 28;
                       else temps -= 29;
                       mois++; break;
             case 3 : m31 (&temps, &mois); break;
             case 4 : m30 (&temps, &mois); break;
             case 5 : m31 (&temps, &mois); break;
             case 6 : m30 (&temps, &mois); break;
             case 7 : m31 (&temps, &mois); break;
```

```
            case 8 : m31 (&temps, &mois); break;
            case 9 : m30 (&temps, &mois); break;
            case 10 : m31 (&temps, &mois); break;
            case 11 : m30 (&temps, &mois); break;
            case 12 : m31 (&temps, &mois); break;
        };
        if (temps > 29)
        switch (mois)
        {   case 4 : m30 (&temps, &mois); break;
            case 6 : m30 (&temps, &mois); break;
            case 9 : m30 (&temps, &mois); break;
            case 11 : m30 (&temps, &mois); break;
        };
        if (temps > 27) if (mois == 2)
        {   if (annee % 4 != 0) temps -= 28;
            else temps -= 29;
            mois++;
        };
        temps++;
        ident->crdate.month = mois;
        ident->crdate.day = temps;
}

m31 (temps, mois)
int *temps, *mois;
{   *temps -= 31;
    *mois += 1;
}

m30 (temps, mois)
int *temps, *mois;
{   *temps -= 30;
    *mois += 1;
}


/*****************************************************************
*
La procédure 'ecrattente' crée un fichier d'attente pour un
abonné.
*****************************************************************
/

ecrattente (ida)
int ida;
{   char nom [12];
    FILE *fich;

    sprintf (nom, "wait%-d", ida);
    fich = fopen (nom, "w");
    fprintf (fich, "%d", 0);
    fclose (fich);
}


/*****************************************************************
*
La procédure 'attente' attend que le fichier d'attente pour
l'abonné 'ida' revienne à son état initial.
```

```
**************************************************************
/

attente (ida)
int ida;
{     char nom [12], comde [15];
      FILE *fich;
      int temps, num, fin;

      fin = 0;
      while (fin == 0)
      {     temps = time (0);
            temps += 5;
            while (temps > time (0)) {};
            sprintf (nom, "wait%-d", ida);
            fich = fopen (nom, "r");
            fscanf (fich, "%d", &num);
            fclose (fich);
            if (num == 0) fin = 1;
      };
      sprintf (comde, "rm %s", nom);
      system (comde);
}
```

```
                \*****************************\
                 \* MODULE BOITE AUX LETTRES *\
                 \*****************************\


\*****************************************************************
*
 creerbal (ida, idnw, bal):
     données: 'ida', un identifiant d'abonné,
              'idnw', un identifiant de réseau;
     préconditions: 'ida' et 'idnw' sont syntaxiquement et
                    sémantiquement corrects;
     résultats: 'bal', une boîte aux lettres;
     postconditions: 'bal' est relative à 'ida' et à 'idnw';
     fonction: créer la boîte aux lettres relative à 'ida'
dans
                le réseau 'idnw'.
*****************************************************************
\

void creerbal (ida, idnw, bal)
int ida;
int idnw;
struct lbal bal;
{    char nom[12], comde[15];
     FILE *fich, *fopen ();
     struct lbal inter;

     sprintf (nom, "bal%-d.dat", ida);
     if (existbal (ida) == 1)
     {    sprintf (comde, "rm %s", nom);
          system (comde);
     };
     fich = fopen (nom, "w");
     inter = bal;
     while (inter.nextiddoc != 0)
     {    fprintf (fich, "%d %d %d %d %d %d %d %d\n",
           inter.iddoc.crdate.year, inter.iddoc.crdate.month,
           inter.iddoc.crdate.day, inter.iddoc.crtime.hour,

inter.iddoc.crtime.minute,inter.iddoc.crtime.second,
           inter.iddoc.ida, idnw);
          inter = *(inter.nextiddoc);
     };
     if (compareidd (inter.iddoc, idvide) == FALSE)
     fprintf (fich, "%d %d %d %d %d %d %d %d\n",
           inter.iddoc.crdate.year, inter.iddoc.crdate.month,
           inter.iddoc.crdate.day, inter.iddoc.crtime.hour,

inter.iddoc.crtime.minute,inter.iddoc.crtime.second,
           inter.iddoc.ida, idnw);
     fclose (fich);
}


/*****************************************************************
*
complbal (ida, idnw, bal)
     données: 'ida', un identifiant d'abonné,
              'idnw', un identifiant de réseau,
```

```
                'bal', une boîte aux lettres;
     préconditions: 'ida' et 'idnw' sont syntaxiquement et
                    sémantiquement correctes,
                    'bal' est une boîte aux lettres associée à
                    'ida';
     fonction: la boîte aux lettres de 'ida' est complétée
               avec 'bal' venant de 'idnw'.
**************************************************************
/

void complbal (ida, idnw, bal)
int ida;
int idnw;
struct lbal bal;
{    char nom[25];
     FILE *fich, *fopen();
     struct lbal inter;

     sprintf (nom, "/usr/patain/bal%-d.dat", ida);
     fich = fopen (nom, "a");
     inter = bal;
     while (inter.nextiddoc != 0)
     {    fprintf (fich, "%d %d %d %d %d %d %d %d\n",
           inter.iddoc.crdate.year, inter.iddoc.crdate.month,
           inter.iddoc.crdate.day, inter.iddoc.crtime.hour,

inter.iddoc.crtime.minute,inter.iddoc.crtime.second,
           inter.iddoc.ida, idnw);
          inter = *(inter.nextiddoc);
     };
     if (compareidd (inter.iddoc, idvide) == FALSE)
     fprintf (fich, "%d %d %d %d %d %d %d %d\n",
           inter.iddoc.crdate.year, inter.iddoc.crdate.month,
           inter.iddoc.crdate.day, inter.iddoc.crtime.hour,

inter.iddoc.crtime.minute,inter.iddoc.crtime.second,
           inter.iddoc.ida, idnw);
     fclose (fich);
}

/**************************************************************
*
consultbal (ida, bal)
     donnée: 'ida', un identifiant d'abonné;
     précondition: 'ida' est syntaxiquement et sémantiquement
                   correct;
     résultat: 'bal', une boîte aux lettres;
     postcondition: 'bal' est relative à 'ida';
     fonction: fournit dans 'bal' la boîte aux lettres totale
               de 'ida'.
**************************************************************
/

consultbal (ida, bal)
int ida;
struct lbal *bal;
{    char nom[12];
     int rien;
     FILE *fich, *fopen();
     struct lbal *inter, *derniere;
```

```
     sprintf (nom, "bal%-d.dat", ida);
     fich = fopen (nom, "r");
     inter = bal;
     derniere = bal;
     fscanf (fich, "%d %d %d %d %d %d %d %d\n",
       &inter->iddoc.crdate.year,&inter->iddoc.crdate.month,
       &inter->iddoc.crdate.day,&inter->iddoc.crtime.hour,
       &inter->iddoc.crtime.minute,&inter-
>iddoc.crtime.second,
       &inter->iddoc.ida, &rien);
     while (feof (fich) == 0)
     {    inter->nextiddoc = malloc (sizeof (*inter));
          inter = inter->nextiddoc;
          derniere = inter;
          fscanf (fich, "%d %d %d %d %d %d %d %d\n",
          &inter->iddoc.crdate.year,&inter-
>iddoc.crdate.month,
           &inter->iddoc.crdate.day,&inter->iddoc.crtime.hour,
       &inter->iddoc.crtime.minute,&inter-
>iddoc.crtime.second,
           &inter->iddoc.ida, &rien);
     };
     derniere->nextiddoc = 0;
     fclose (fich);
};


\*****************************************************************
*
 supbal (ida):
     donnée: 'ida', un identifiant d'abonné;
     précondition: 'ida' est syntaxiquement correct;
     fonction: supprime la boîte aux lettres relative à 'ida'
               dans le serveur local.
*****************************************************************
\

void supbal (ida)
int ida;
{    char nom[12], comde[20];

     sprintf (nom, "bal%-d.dat", ida);
     sprintf (comde, "rm %s", nom);
     system (comde);
};


\*****************************************************************
*
compidd compare 'ident1' et 'ident2', renvoie 0 s'ils sont
identiques, 1 sinon.
*****************************************************************
\

compidd (ident1, ident2)
struct idd ident1;
struct idd ident2;
{       int trouve;
```

```
        if (ident1.crdate.year == ident2.crdate.year)
        if (ident1.crdate.month == ident2.crdate.month)
        if (ident1.crdate.day == ident2.crdate.day)
        if (ident1.crtime.hour == ident2.crtime.hour)
        if (ident1.crtime.minute == ident2.crtime.minute)
        if (ident1.crtime.second == ident2.crtime.second)
        if (ident1.ida == ident2.ida) return (0);
        else return (1);
};


\*************************************************************
*
supdocbal (ida, ident)
     donnees: 'ida', un identifiant d'abonné,
              'ident', un identifiant de document;
     préconditions: 'ida' et 'ident' sont syntaxiquement
                    corrects;
     fonction: supprime 'ident' de la boîte aux lettres de
               'ida' dans le serveur local.
*************************************************************
\

void supdocbal (ida, ident)
int ida;
struct idd ident;
{    char nom[12], comde[30];
     FILE *fich1, *fich2, *fopen ();
     int rien;
     struct lbal inter;

     sprintf (nom, "bal%-d.dat", ida);
     fich1 = fopen (nom, "r");
     fich2 = fopen ("temporai.bal", "w");
     fscanf (fich1, "%d %d %d %d %d %d %d %d\n",
         &inter.iddoc.crdate.year,&inter.iddoc.crdate.month,
         &inter.iddoc.crdate.day,&inter.iddoc.crtime.hour,

&inter.iddoc.crtime.minute,&inter.iddoc.crtime.second,
         &inter.iddoc.ida, &rien);
     while (feof (fich1) == 0)
     {    if (compidd (ident, inter.iddoc) != 0)
          {    fprintf (fich2, "%d %d %d %d %d %d %d %d\n",
                    inter.iddoc.crdate.year,
                    inter.iddoc.crdate.month,
                    inter.iddoc.crdate.day,
                    inter.iddoc.crtime.hour,
                    inter.iddoc.crtime.minute,
                    inter.iddoc.crtime.second,
                    inter.iddoc.ida, rien);
               fscanf (fich1, "%d %d %d %d %d %d %d %d\n",
                    &inter.iddoc.crdate.year,
                    &inter.iddoc.crdate.month,
                    &inter.iddoc.crdate.day,
                    &inter.iddoc.crtime.hour,
                    &inter.iddoc.crtime.minute,
                    &inter.iddoc.crtime.second,
                    &inter.iddoc.ida, &rien);
          }
          else
```

```
        {    fscanf (fich1, "%d %d %d %d %d %d %d %d\n",
                            &inter.iddoc.crdate.year,
                            &inter.iddoc.crdate.month,
                            &inter.iddoc.crdate.day,
                            &inter.iddoc.crtime.hour,
                            &inter.iddoc.crtime.minute,
                            &inter.iddoc.crtime.second,
                            &inter.iddoc.ida, &rien);
        }
    };
    if (compidd (ident, inter.iddoc) != 0)
        fprintf (fich2, "%d %d %d %d %d %d %d %d\n",
         inter.iddoc.crdate.year,inter.iddoc.crdate.month,
         inter.iddoc.crdate.day,inter.iddoc.crtime.hour,
         inter.iddoc.crtime.minute,
         inter.iddoc.crtime.second,inter.iddoc.ida, rien);
    fclose (fich1);
    fclose (fich2);
    sprintf (comde, "cp temporai.bal %s", nom);
    system (comde);
    system ("rm temporai.bal");
};


\*****************************************************************
*
localdoc (ida, ident, idnw)
    donnée: 'ida', un identifiant d'abonné,
            'ident', un identifiant de document;
    préconditions: 'ida' et 'ident' sont sémantiquement et
                   syntaxiquement corrects;
    résultat: 'idnw', un identifiant de réseau;
    postcondition: 'idnw' correspond au réseau où est stocké
                  le document 'ident';
    fonction: renvoie l'identifiant du réseau où est localisé
              'ident'.
*****************************************************************
\

int localdoc (ida, ident, idnw)
int ida;
struct idd ident;
int *idnw;
{   FILE *fich, *fopen ();
    struct lbal bal;
    char nom [12];
    int trouve;

    sprintf (nom, "bal%-d.dat", ida);
    fich = fopen (nom, "r");
    fscanf (fich, "%d %d %d %d %d %d %d %d\n",
        &bal.iddoc.crdate.year,&bal.iddoc.crdate.month,
        &bal.iddoc.crdate.day,&bal.iddoc.crtime.hour,
        &bal.iddoc.crtime.minute,&bal.iddoc.crtime.second,
        &bal.iddoc.ida, idnw);
    trouve = 0;
    while (feof (fich) == 0 && trouve == 0)
    {  if (compidd (ident, bal.iddoc) == 0) trouve = 1;
       else fscanf (fich, "%d %d %d %d %d %d %d %d\n",
            &bal.iddoc.crdate.year,&bal.iddoc.crdate.month,
```

```
            &bal.iddoc.crdate.day,&bal.iddoc.crtime.hour,
&bal.iddoc.crtime.minute,&bal.iddoc.crtime.second,
            &bal.iddoc.ida, idnw);
     };
     fclose (fich);
     return(trouve);
};


\*****************************************************************
*
existbal teste si la boîte aux lettres de 'ida' a déjà été
créée.
*****************************************************************
\

int existbal (ida)
int ida;
{    char nom[12];
     FILE *fich, *fopen();
     int trouve;

     sprintf (nom, "bal%-d.dat", ida);
     fich = fopen (nom, "r");
     if (fich != 0)
     { trouve = 1;
       fclose (fich);}
     else trouve = 0;
     return (trouve);
};


\*****************************************************************
*
remplirbal remplit 'bal' pour 'ida'.
*****************************************************************
\

void remplirbal (ida, bal)
int ida;
struct lbal *bal;
{    struct lbal *inter;
     struct lbal *dernier;
     struct idd *iddocr;

     inter = bal;
     inter->iddoc = idvide;
     dernier = inter;
     iddocr = docsuiv (idvide, ida);
     while (compareidd(idvide,*iddocr) != TRUE)
     {    inter->iddoc = *iddocr;
          dernier = inter;
          inter->nextiddoc = malloc (sizeof (*inter));
          inter = inter->nextiddoc;
          iddocr = docsuiv (*iddocr, ida);
     };
     dernier->nextiddoc = 0;
};
```

```
                    /*******************/
                    /* MODULE DOCUMENT */
                    /*******************/


/*****************************************************************
*
compiddo compare 'ident1' et 'ident2', renvoie 0 s'il sont les
m@mes, 1 sinon
*****************************************************************
/

compiddo (ident1, ident2)
struct idd ident1;
struct idd ident2;
{    int trouve;

     if (ident1.crdate.year == ident2.crdate.year)
     if (ident1.crdate.month == ident2.crdate.month)
     if (ident1.crdate.day == ident2.crdate.day)
     if (ident1.crtime.hour == ident2.crtime.hour)
     if (ident1.crtime.minute == ident2.crtime.minute)
     if (ident1.crtime.second == ident2.crtime.second)
     if (ident1.ida == ident2.ida) return (0);
     else return (1);
};


/*****************************************************************
*
stockdoc (corps, idennt)
     données: 'corps', un corps de document,
              'ident', un identifiant de document;
     préconditions: 'ident' est syntaxiquement correct,
                    'corps' est le corps du document
                    d'identifiant 'ident';
     fonction: réalise le stockage du document 'ident' de
corps
               'corps' dans la base de données du serveur
               local.
*****************************************************************
/

void stockdoc (corps, ident)
struct line corps;
struct idd ident;
{    char nom[12];

     stocor (corps, nom);
     stocid (ident, nom);
};


void nomsuiv (nom)
char nom[12];
{    int numero;
     FILE *fich, *fopen ();

     fich = fopen ("dernier.dat", "r");
     fscanf (fich, "%d", &numero);
```

```
     fclose (fich);
     numero++;
     fich = fopen ("dernier.dat", "w");
     fprintf (fich, "%d", numero);
     fclose (fich);
     sprintf (nom, "f%-d.dat", numero);
};


void stocor (corps, nom)
struct line corps;
char nom[12];
{    struct line inter;
     FILE *fopen (), *fich;

     nomsuiv (nom);
     fich = fopen (nom, "w");
     inter = corps;
     do
     {    fputs (inter.l, fich);
          if (inter.nextl != 0) inter = *(inter.nextl);
     }
     while (inter.nextl != 0);
     fclose (fich);
};


void stocid (ident, nom)
struct idd ident;
char nom[12];
{    FILE *fopen (), *fich;

     fich = fopen ("document.idx", "a");
     fprintf (fich, "%d %d %d %d %d %d %d %s\n",
        ident.crdate.year,
        ident.crdate.month,ident.crdate.day,ident.crtime.hour,

ident.crtime.minute,ident.crtime.second,ident.ida,nom);
     fclose (fich);
};


/*************************************************************
*
cherchedoc (ident, corps)
     donnée: 'ident', un identifiant de document;
     préconditions: 'ident' est syntaxiquement correct;
                    le document 'ident' se trouve dans le
                    serveur local;
     résultat: 'corps', un corps de document;
     postcondition: 'corps' est le corps du document 'ident';
     fonction: accède au corps du document 'ident' dans la
base
               de données du serveur local.
*************************************************************
/

int cherchedoc (ident, corps)
struct idd ident;
struct line *corps;
```

```
{    char nom [12];

     if (cherchnom (ident, nom) == 0) return (0);
     else {    cherchcor (nom, corps);
          return (1);
          };
};


int cherchnom (ident, nom)
struct idd ident;
char *nom;
{    FILE *fopen (), *fich;
     struct idd inter;
     int trouve;

     fich = fopen ("document.idx", "r");
     fscanf (fich, "%d %d %d %d %d %d %d
%s,&inter.crdate.year,
     &inter.crdate.month,&inter.crdate.day,&inter.crtime.hour,
     &inter.crtime.minute,&inter.crtime.second,&inter.ida,nom)
;
     trouve = 0;
     while (feof (fich) == 0 && trouve == 0)
     {    if (compiddo (ident, inter) == 0) trouve = 1;
          else fscanf (fich, "%d %d %d %d %d %d %d %s",
                    &inter.crdate.year,&inter.crdate.month,
                    &inter.crdate.day,&inter.crtime.hour,

&inter.crtime.minute,&inter.crtime.second,
                    &inter.ida, nom);
     };
     fclose (fich);
     return (trouve);
};


void cherchcor (nom, corps)
char nom[12];
struct line *corps;
{    FILE *fopen (), *fich;
     struct line *inter, derniere;

     fich = fopen (nom, "r");
     inter = corps;
     fgets (inter->l, MAXL, fich);
     while (feof (fich) == 0)
     {    inter->nextl = malloc (sizeof (*inter));
          derniere = *inter;
          inter = inter->nextl;
          fgets (inter->l, MAXL, fich);
     };
     derniere.nextl = 0;
     fclose (fich);
};


/*************************************************************
*
supdoc (ident)
```

```
     donnée: 'ident', un identifiant de document;
     préconditions: 'ident' est syntaxiquement correct,
                    le document 'ident' se trouve dans le
                    serveur local;
     fonction: supprime le document 'ident' de la base de
               données du serveur local.
**********************************************************/

void supdoc (ident)
struct idd ident;
{    char nom [12], comde[20];

     supidd (ident, nom);
     sprintf (comde, "rm %s", nom);
     system comde);
}

void supidd (ident, nom)
struct idd ident;
char *nom;
{    char nominter[12];
     FILE *fopen (), *fich1, *fich2;
     struct idd inter;

     fich1 = fopen ("document.idx", "r");
     fich2 = fopen ("temporai.idx", "w");
     fscanf (fich1, "%d %d %d %d %d %d %d %s",
      &inter.crdate.year,&inter.crdate.month,
      &inter.crdate.day,&inter.crtime.hour,
      &inter.crtime.minute,&inter.crtime.second,&inter.ida,
      nominter);
     while (feof (fich1) == 0)
     {    if (compiddo (ident, inter) != 0)
          {    fprintf (fich2, "%d %d %d %d %d %d %d %s\n",
                     inter.crdate.year,inter.crdate.month,
                     inter.crdate.day,inter.crtime.hour,
                     inter.crtime.minute,inter.crtime.second,
                     inter.ida, nominter);
               fscanf (fich1, "%d %d %d %d %d %d %d %s",
                     &inter.crdate.year,&inter.crdate.month,
                     &inter.crdate.day,&inter.crtime.hour,

&inter.crtime.minute,&inter.crtime.second,
                     &inter.ida, nominter);
          }
          else
          {    sprintf (nom, "%s", nominter);
               fscanf (fich1, "%d %d %d %d %d %d %d %s",
                     &inter.crdate.year,&inter.crdate.month,
                     &inter.crdate.day,&inter.crtime.hour,

&inter.crtime.minute,&inter.crtime.second,
                     &inter.ida, nominter);
          };
     };
     fclose (fich1);
     fclose (fich2);
     system ("cp temporai.idx document.idx");
     system ("rm temporai.idx");
};
```

```
                    /***************/
                    /* MODULE E-S */
                    /***************/


/*************************************************************
affichbal (bal)
     donnée: 'bal', une boîte aux lettres;
     précondition: 'bal' est syntaxiquement correcte;
     fonction:  affiche 'bal' sur la sortie standard jusqu'à ce
                que <return> soit tapé
*************************************************************/
affichbal (bal)
struct lbal bal;
{    struct lbal inter;

     inter = bal;
     cls ();
     printf ("\nBal (heure-minute-seconde-annee-mois-jour-ida)
:\n");
     while (inter.nextiddoc != 0)
     {       printf ("      %2d     %2d     %2d    %4d   %2d
%2d   %d\n",

inter.iddoc.crtime.hour,inter.iddoc.crtime.minute,

inter.iddoc.crtime.second,inter.iddoc.crdate.year,
              inter.iddoc.crdate.month, inter.iddoc.crdate.day,
              inter.iddoc.ida);
              inter = *(inter.nextiddoc);
        };
     if (compareidd (idvide, inter.iddoc) != 1)
     printf ("      %2d     %2d     %2d    %4d   %2d   %2d
%d\n",
             inter.iddoc.crtime.hour,
inter.iddoc.crtime.minute,
             inter.iddoc.crtime.second,
inter.iddoc.crdate.year,
             inter.iddoc.crdate.month, inter.iddoc.crdate.day,
             inter.iddoc.ida);
     printf ("\n");
};


/*************************************************************
affichdoc (ident, corps)
     données: 'ident', un identifiant de document,
              'corps', un corps de document;
     préconditions: 'ident' et 'corps' sont syntaxiquement
                    corrects;
     fonction:   affiche  'ident'  et  'corps'  sur  la  sortie
                 standard jusqu'à ce que <return> soit tapé
*************************************************************/

affichdoc (ident, corps)
struct idd ident;
struct line corps;
{    cls ();
     printf ("\nIdentifiant du document :\n");
     printf ("%d %d %d %d %d %d %d\n", ident.crtime.hour,
```

```
ident.crtime.minute,ident.crtime.second,ident.crdate.year,
     ident.crdate.month, ident.crdate.day, ident.ida);
     printf ("\nCorps du document :\n");
     printcorps (corps);
};


printcorps (corps)
struct line corps;
{       struct line inter;

        inter = corps;
        while (inter.nextl != 0)
        {       printf ("%s", inter.l);
                inter = *(inter.nextl);
        };
        printf ("%s", inter.l);
}


/*****************************************************************
affichmenu (num)
     donnée: 'num', un numéro de menu;
     précondition: 'num' est un numéro de menu existant;
     résultat: 'affichmenu', un numéro de choix;
     postcondition:   le numéro   est correct pour le menu
                      considéré
     fonction:  affiche le  menu numéro  'num'  sur  la  sortie
                standard et lit le choix sur l'entrée standard.
*****************************************************************/

affichmenu (num)
int num;
{    char command[15], fichier[15];
     int res, ok, choix;

     sprintf (command, "cat menu%-d.dat", num);
     system (command);
     sprintf (fichier, "menu%-dch.dat", num);
     fich = fopen (fichier, "r");
     fscanf (fich, "%d", &res);
     ok=0;
     while (ok==0)
     {    printf ("\n votre choix ? : ");
          scanf ("%d", &choix);
          if ((choix)=0) && (choix(=res)) ok=1;
     };
     return (choix);
};


/*****************************************************************
affichmess (num)
     donnée: 'num', un numéro de message;
     précondition: 'num' est un numéro de message existant;
     fonction:  affiche le  message  de  numéro  'num'  sur  la
sortie
                standard jusqu'à ce que (return) soit tapé
*****************************************************************/
```

```
affichmess (num)
int num;
{    switch (num)
     {case 1 : printf ("\nIntroduisez votre numero d'ida\n");
               break;
      case 2 :
         printf ("\nIntroduisez l'identifiant du document ");
         printf ("a consulter\n");
         break;
      case 3 :
         printf ("\nIntroduire d'abord un corps de document
");
         printf ("et une liste de destinataires\n");
         break;
      case 4 :
         printf ("\nIntroduisez la liste des destinataires,
");
         printf ("terminez par ida = 0\n");
         break;
      case 5 :
         printf ("\nConnection interdite sur ce reseau\n");
         break;
      case 6 :
         printf ("\nIntroduisez l'identifiant du document ");
         printf ("a supprimer\n");
         break;
      case 7 : printf ("\nCe document n'existe pas\n");
               break;
     };
};


/*****************************************************************
lectcdoc (corps)
     résultat: 'corps';
     postcondition: 'corps' est syntaxiquement correct;
     fonction: lit 'corps' à partir de l'entrée standard
*****************************************************************/

lectcdoc (corps)
struct line *corps;
{    struct line *inter, derniere;
     int lg=2, i;
     char ligne [MAXL];

     inter = corps;
     cls ();
     getchar ();
     printf ("\nIntroduction du document (terminer par . sur
une ");
     printf ("ligne) :\n");
     while (lg > 1)
     {    lg = getline (ligne);
          if (lg > 1)
          {    for (i=0; i<MAXL; i++) inter->l [i] = ligne
[i];
               inter->nextl = malloc (sizeof (*inter));
               derniere = *inter;
               inter = inter->nextl;
```

```
            }
        };
        derniere.nextl = 0;
};


getline (s)
char s[];
{       int c, i;

        for (i=0; i<MAXL-1 && (s[i]=getchar()) != '\n'; i++);
        s[i+1] = '\0';
        if (s[0]=='.') return (1);
        else return (2);
};


/*****************************************************************
lectiddoc (ident)
     résultat: 'ident', un identifiant de document;
     postcondition: 'ident' est syntaxiquement correct;
     fonction: lit 'ident' à partir de l'entrée standard.
*****************************************************************/

lectiddoc (ident)
struct idd *ident;
{    printf ("\nIdentification du document :\n");
     printf ("heure : ");
     scanf ("%d", &ident->crtime.hour);
     printf ("minute : ");
     scanf ("%d", &ident->crtime.minute);
     printf ("seconde : ");
     scanf ("%d", &ident->crtime.second);
     printf ("annee : ");
     scanf ("%d", &ident->crdate.year);
     printf ("mois : ");
     scanf ("%d", &ident->crdate.month);
     printf ("jour : ");
     scanf ("%d", &ident->crdate.day);
     printf ("ida createur : ");
     scanf ("%d", &ident->ida);
};


/*****************************************************************
lectida (ida)
     résultat: 'ida', un identifiant d'abonné;
     postcondition: ida est syntaxiquement correct;
     fonction: lit 'ida' à partir de l'entée standard.
*****************************************************************/

lectida (ida)
int *ida;
{    printf ("\nIdentification de l'abonne : ");
     scanf ("%d", ida);
};


/*****************************************************************
cls ()
```

```
     fonction: efface l'écran.
*****************************************************************/

cls ()
{    printf
("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
};


/*****************************************************************
stop ()
     fonction: attend qu'on tape return.
*****************************************************************/

stop ()
{    char bidon;

     scanf ("%c", &bidon);
     scanf ("%c", &bidon);
};
```

```
                /*************************/
                /*   Module de routage   */
                /*************************/


/*****************************************************************
appel de la procédure : creerliste(iddoc,idnw);
objectif : initialiser une liste des abonnés situés sur un
réseau donné et qui soient destinataires du document donné;
cette initialisation consiste à créer une liste vide;
entrées : iddoc : un identifiant de document;
          idnw : un identifiant de réseau;
pré-conditions : iddoc et idnw sont syntaxiquement et
sémantiquement corrects;
résultats : / ;
post-conditions : / .
*****************************************************************/

void creerliste(iddoc,idnw)
struct idd iddoc;
int idnw;

{struct routnw *newlist;
 extern struct routnw *firstl,*lastl;

 newlist=malloc(sizeof(*newlist));
 if (firstl==0)
    {firstl=newlist;
     lastl=newlist;}
 else {lastl->nextl=newlist;
       lastl=newlist;};
 newlist->iddoc=iddoc;
 newlist->idnw=idnw;
 newlist->firstdest=0;
 newlist->lastdest=0;
 newlist->nextl=0;
};



/*****************************************************************
appel de la procédure : ajoutliste(idab,iddoc,idnw);
objectif : ajouter l'abonné idab à la liste des abonnés situés
sur le réseau idnw et qui sont destinataires du document
iddoc;
entrées : iddoc : un identifiant de document;
          idab : un identifiant d'abonné;
          idnw : un identifiant de réseau;
pré-conditions : iddoc, idab et idnw sont syntaxiquement et
sémantiquement corrects;
résultats : / ;
post-conditions : / .
*****************************************************************/

void ajoutliste(idab,iddoc,idnw)
struct idd iddoc;
int idab,idnw;

{struct llnw *newdest;
```

```
 struct routnw *newlist, *currentl;
 extern struct routnw *firstl,*lastl;

 newdest=malloc(sizeof(*newdest));
 currentl=firstl;
 while ((compareidd(currentl->iddoc,iddoc)==0)||(currentl-
>idnw!=idnw))
    {currentl=currentl->nextl;};
 if (currentl->firstdest==0)
    {currentl->firstdest=newdest;
     currentl->lastdest=newdest;}
 else {currentl->lastdest->nextdest=newdest;
       currentl->lastdest=newdest;};
 newdest->dest=idab;
 newdest->nextdest=0;
};



/*****************************************************************
appel de la procédure : supprimliste(iddoc,idnw);
objectif : supprimer la liste des destinataires du document
iddoc qui sont situés sur le réseau idnw;
entrées : iddoc : un identifiant de document;
          idnw : un identifiant de réseau;
pré-conditions : iddoc et idnw sont syntaxiquement et
sémantiquement corrects;
résultats : / ;
post-conditions : / .
*****************************************************************/

void supprimliste(iddoc,idnw)
struct idd iddoc;
int idnw;

{struct routnw *currentl,*previousl;
 extern struct routnw *firstl,*lastl;

 currentl=firstl;
 previousl=0;
 while ((compareidd(iddoc,currentl->iddoc)==0)||(currentl-
>idnw!=idnw))
    {previousl=currentl;
     currentl=currentl->nextl;};
 if ((previousl==0)&&(currentl->nextl==0))
    {firstl=0;
     lastl=0;}
 else if (currentl->nextl==0)
         {lastl=previousl;
          previousl->nextl=0;}
      else if (previousl==0) firstl=firstl->nextl;
           else previousl->nextl=currentl->nextl;
};



/*****************************************************************
appel de la fonction : fressuiv(idnw);
objectif : fournir l'identifiant du réseau suivant idnw dans
le système;
```

```
entrées : idnw : un identifiant de réseau;
pré-conditions : idnw est syntaxiquement et sémantiquement
correct; idnw peut être nul;
résultats : fressuiv : un identifiant de réseau;
post-conditions : fressuiv est l'identifiant du réseau suivant
idnw dans le système; fressuiv est l'identifiant du premier
réseau du système si idnw est nul; fressuiv est nul si idnw
est l'identifiant du dernier réseau du système.
**************************************************************/

int fressuiv(idnw)
int idnw;

{int network;

 fich=fopen("fsys.dat","r");
 fscanf(fich,"%d ",&network);
 if (idnw!=0)
    {while (idnw!=network) fscanf(fich,"%d ",&network);
     if (fscanf(fich,"%d ",&network)==EOF) network=0;
    };
 fclose(fich);
 return(network);
};



/**************************************************************
appel de la fonction : consultliste(iddoc,idnw);
objectif : renvoyer la liste des destinataires du document
iddoc qui sont situés sur le réseau idnw;
entrées : iddoc : un identifiant de document;
          idnw : un identifiant de réseau;
pré-conditions : iddoc et idnw sont syntaxiquement et
sémantiquement corrects;
résultats : consultliste : une référence à une liste de
destinataires;
post-conditions : la liste référencée par consultliste
contient tous les abonnés situés sur le réseau idnw et
destinataires du document iddoc.
**************************************************************/

struct llnw *consultliste(iddoc,idnw)
struct idd iddoc;
int idnw;

{struct llnw *currentdest;
 extern struct routnw *firstl;
 struct routnw *currentl;

 currentl=firstl;
 while ((compareidd(iddoc,currentl->iddoc)==0)||(currentl-
>idnw!=idnw))
     {currentl=currentl->nextl;};
 return(currentl->firstdest);
};



/**************************************************************
```

```
appel de la fonction : flistevide(ldest);
objectif : dire si la liste de destinataires ldest est vide ou
non;
entrées : ldest : une structure llnw;
pré-conditions : ldest est syntaxiquement correcte;
résultats : flistevide : entier;
post-conditions : flistevide est une variable booléenne qui
vaut 1 si la liste de destinataires ldest est vide et 0 sinon.
**********************************************************/

int flistevide(ldest)
struct llnw *ldest;

{extern struct routnw *firstl;
 struct routnw *currentl;

 if (ldest==0) return(1);
 else return(0);
};




void lireliste()

/* lireliste lit une liste de destinataires pour un iddoc-idnw
   existant a partir du fichier newlist.dat */

{struct idd iddocr;
 int idnwr,idabr;

 fich=fopen("newlist.dat","r");
 fscanf(fich,"%d %d %d %d %d %d %d %d ", &iddocr.crdate.year,
        &iddocr.crdate.month, &iddocr.crdate.day,
        &iddocr.crtime.hour, &iddocr.crtime.minute,
        &iddocr.crtime.second, &iddocr.ida,&idnwr);
 while (fscanf(fich,"\n %d",&idabr)!=EOF)
   {ajoutliste(idabr,iddocr,idnwr);};
 fclose(fich);
};
```

```
                /*********************************/
                /* Module des fonctions liées à */
                /*        l'utilisateur          */
                /*********************************/


/*****************************************************************
appel de la fonction : fconsultnextlien(iddoc,idab);
objectif : fournir le lien suivant relatif à un document
donné;
entrées : iddoc : un identifiant de document;
          idab : un identifiant d'abonné;
pré-conditions : iddoc et idab sont syntaxiquement et
sémantiquement corrects; idab peut être nul;
résultats : fconsultnextlien : un identifiant d'abonné;
post-conditions : fconsultnextlien est syntaxiquement et
sémantiquement correct; si idab est nul, fconsultnextlien est
l'identifiant du premier abonné lié au document donné;
fconsultnextlien peut être nul s'il n'y a pas d'abonné
suivant.
*****************************************************************/

int fconsultnextlien(iddoc,idab)
struct idd iddoc;
int idab;

{int ok,idabr;
 struct idd iddocr;

 fich=fopen("fliens.dat","r");
 fscanf(fich,"%d %d %d %d %d %d %d %d", &idabr,
        &iddocr.crdate.year, &iddocr.crdate.month,
        &iddocr.crdate.day, &iddocr.crtime.hour,
        &iddocr.crtime.minute, &iddocr.crtime.second,
        &iddocr.ida);
 while (compareidd(iddocr,iddoc)==0)
   {fscanf(fich,"%d %d %d %d %d %d %d %d ", &idabr,
           &iddocr.crdate.year, &iddocr.crdate.month,
           &iddocr.crdate.day, &iddocr.crtime.hour,
           &iddocr.crtime.minute, &iddocr.crtime.second,
           &iddocr.ida);};
 if (idab!=0)  {while
((idab!=idabr)||(compareidd(iddoc,iddocr)==0))
                  {fscanf(fich,"%d %d %d %d %d %d %d %d ",
                     &idabr, &iddocr.crdate.year,
                     &iddocr.crdate.month, &iddocr.crdate.day,

&iddocr.crtime.hour,&iddocr.crtime.minute,
                     &iddocr.crtime.second, &iddocr.ida);
                   };
                 if (fscanf(fich,"%d %d %d %d %d %d %d %d ",
                     &idabr, &iddocr.crdate.year,
                     &iddocr.crdate.month, &iddocr.crdate.day,

&iddocr.crtime.hour,&iddocr.crtime.minute,
                     &iddocr.crtime.second, &iddocr.ida)!=EOF)
                {ok=1;
                 while
((compareidd(iddocr,iddoc)==0)&&(ok==1))
                  {if (fscanf(fich,"%d %d %d %d %d %d %d %d ",
```

```
                    &idabr, &iddocr.crdate.year,
                    &iddocr.crdate.month, &iddocr.crdate.day,

&iddocr.crtime.hour,&iddocr.crtime.minute,
                    &iddocr.crtime.second, &iddocr.ida)==EOF)
                  ok=0;};
                }
                else ok=0;
                if (ok==0) idabr=0;
               };
 return(idabr);
};


/*****************************************************************
appel de la procédure : creerlien(iddoc,idab);
objectif : creer un lien entre l'abonné idab et le document
iddoc;
entrées : iddoc : un identifiant de document;
          idab : un identifiant d'abonné;
pré-conditions : iddoc et idab sont syntaxiquement et
sémantiquement corrects;
résultats : / ;
post-conditions : / .
*****************************************************************/

void creerlien(iddoc,idab)
struct idd iddoc;
int idab;

{

 fich=fopen("fliens.dat","a");
 fprintf(fich,"%d %d %d %d %d %d %d %d\n", idab,
         iddoc.crdate.year, iddoc.crdate.month,
         iddoc.crdate.day, iddoc.crtime.hour,
         iddoc.crtime.minute, iddoc.crtime.second, iddoc.ida);
 fclose(fich);
};


*****************************************************************
appel de la fonction : docsuiv(iddoc,idab);
objectif : fournir le document suivant iddoc et relatif a
idab;
entrées : iddoc : un identifiant de document;
          idab : un identifiant d'abonné;
pré-conditions : iddoc et idab sont syntaxiquement et
sémantiquement corrects; iddoc peut être nul;
résultats : docsuiv : une référence à un identifiant de
document;
post-conditions : l'identifiant de document référencé par
docsuiv est syntaxiquement et sémantiquement correct; cet
identifiant est le premier si iddoc est nul; l'identifiant
peut être nul si iddoc est l'identifiant du dernier document.
*****************************************************************/

struct idd *docsuiv(iddoc,idab)
```

```
int idab;
struct idd iddoc;

{int finfich,idabr,ok;
 extern struct idd idvide,iddocr;

 fich=fopen("fliens.dat","r");
 if (fich==0)
     return(&idvide);
 if (fscanf(fich,"%d %d %d %d %d %d %d %d ", &idabr,
     &iddocr.crdate.year, &iddocr.crdate.month,
     &iddocr.crdate.day, &iddocr.crtime.hour,
     &iddocr.crtime.minute, &iddocr.crtime.second,
     &iddocr.ida)==EOF)
  return(&idvide);
 if (compareidd(iddoc,idvide)==TRUE)
    {finfich=FALSE;
     while ((idab!=idabr)&&(finfich==FALSE))
      {if (fscanf(fich,"%d %d %d %d %d %d %d %d ",&idabr,
                  &iddocr.crdate.year,&iddocr.crdate.month,
                  &iddocr.crdate.day,&iddocr.crtime.hour,
                  &iddocr.crtime.minute,&iddocr.crtime.second,
                  &iddocr.ida)==EOF)finfich=TRUE;
      };
     if (finfich==TRUE)
        return(&idvide);
       return(&iddocr);
    }
 else {while
((idabr!=idab)||(compareidd(iddoc,iddocr)==FALSE))
        {fscanf(fich,"%d %d %d %d %d %d %d %d ",&idabr,
                &iddocr.crdate.year,&iddocr.crdate.month,
                &iddocr.crdate.day,&iddocr.crtime.hour,
                &iddocr.crtime.minute, &iddocr.crtime.second,
                &iddocr.ida);
         };
        if (fscanf(fich,"%d %d %d %d %d %d %d %d ",&idabr,
                  &iddocr.crdate.year,&iddocr.crdate.month,
                  &iddocr.crdate.day,&iddocr.crtime.hour,
                  &iddocr.crtime.minute,&iddocr.crtime.second,
                  &iddocr.ida)==EOF) finfich=TRUE;
        else finfich=FALSE;
        while ((finfich==FALSE)&&(idab!=idabr))
          {if (fscanf(fich,"%d %d %d %d %d %d %d %d ",&idabr,
                    &iddocr.crdate.year,&iddocr.crdate.month,
                    &iddocr.crdate.day,&iddocr.crtime.hour,

&iddocr.crtime.minute,&iddocr.crtime.second,
                    &iddocr.ida)==EOF)
           finfich=TRUE;};
           if (finfich==TRUE)
               return(&idvide);
             return(&iddocr);
              };
};


/*************************************************************
appel de la fonction : flocalisation(idab);
```

```
objectif : donner la localisation d'un abonné;
entrées : idab : un identifiant d'abonné;
pré-conditions : idab est syntaxiquement et sémantiquement
correct;
résultats : flocalisation : un identifiant de réseau;
post-conditions : flocalisation est syntaxiquement et
sémantiquement correct.
**************************************************************/

int flocalisation(idab)
int idab;

 fich=fopen("flocalab.dat","r");
 fscanf(fich,"%d %d",&idabr,&idnwr);
 while (idabr!=idab)
        {fscanf(fich,"%d %d",&idabr,&idnwr);
        };
 return(idnwr);
};



/**************************************************************
appel de la fonction : fexistlien(iddoc);
objectif : dire s'il existe au moins un abonné associé à un
document donné;
entrées : iddoc : un identifiant de document;
pré-conditions : iddoc est syntaxiquement et sémantiquement
correct;
résultats : fexistlien : entier;
post-conditions : fexistlien est une variable booléenne qui
vaut 1 s'il existe au moins un abonné lié à iddoc et 0 sinon.
**************************************************************/

int fexistlien(iddoc)
struct idd iddoc;

{int trouve,idabr;
 struct idd iddocr;

 fich=fopen("fliens.dat","r");
 trouve=0;
 while ((fscanf(fich,"%d %d %d %d %d %d %d %d ",&idabr,
                &iddocr.crdate.year,&iddocr.crdate.month,
                &iddocr.crdate.day,&iddocr.crtime.hour,
                &iddocr.crtime.minute,&iddocr.crtime.second,
                &iddocr.ida)!=EOF)&&(trouve==0))
  {if (compareidd(iddoc,iddocr)==1) trouve = 1;};
printf("FEXISTLIEN-\n");
 return (trouve);
};



/**************************************************************
appel de la procédure : supprimlien(iddoc,idab);
objectif : supprimer le lien existant entre un abonné et un
document donnés;
entrées : iddoc : un identifiant de document;
          idab : un identifiant d'abnné;
```

```
pré-conditions : iddoc et idab sont syntaxiquement et
sémantiquement corrects;
résultats : / ;
post-conditions : / .
*****************************************************************/

void supprimlien(iddoc,idab)
struct idd iddoc;
int idab;

{struct idd iddocr;
 int idabr;
 FILE *fich2;

 fich=fopen("fliens.dat","r");
 fich2=fopen("fliens2.dat","w");
 fscanf(fich,"%d %d %d %d %d %d %d %d ", &idabr,
        &iddocr.crdate.year, &iddocr.crdate.month,
        &iddocr.crdate.day, &iddocr.crtime.hour,
        &iddocr.crtime.minute, &iddocr.crtime.second,
        &iddocr.ida);
 while((compareidd(iddocr,iddoc)==0)||(idab!=idabr))
   {fprintf(fich2,"%d %d %d %d %d %d %d %d \n", idabr,
            iddocr.crdate.year, iddocr.crdate.month,
            iddocr.crdate.day, iddocr.crtime.hour,
            iddocr.crtime.minute, iddocr.crtime.second,
            iddocr.ida);
    fscanf(fich,"%d %d %d %d %d %d %d %d ",&idabr,
           &iddocr.crdate.year, &iddocr.crdate.month,
           &iddocr.crdate.day, &iddocr.crtime.hour,
           &iddocr.crtime.minute, &iddocr.crtime.second,
           &iddocr.ida);
   };
 while(fscanf(fich,"%d %d %d %d %d %d %d %d ",&idabr,
              &iddocr.crdate.year, &iddocr.crdate.month,
              &iddocr.crdate.day, &iddocr.crtime.hour,
              &iddocr.crtime.minute,&iddocr.crtime.second,
              &iddocr.ida)!=EOF)
   {fprintf(fich2,"%d %d %d %d %d %d %d %d \n",idabr,
            iddocr.crdate.year, iddocr.crdate.month,
            iddocr.crdate.day, iddocr.crtime.hour,
            iddocr.crtime.minute, iddocr.crtime.second,
            iddocr.ida);
   };
 fclose(fich);
 fclose(fich2);
 system("cp fliens2.dat fliens.dat");
 system("rm fliens2.dat");
};
```

```
/*************************************************************
appel de la fonction : compareidd(iddoc1,iddoc2);
objectif : dire si deux identifiants de document sont
identiques;
entrées : iddoc1 et iddoc2 : des identifiants de documents;
pré-conditions : iddoc1 et iddoc2 sont syntaxiquement
corrects;
résultats : compareidd : entier;
post-conditions : compareidd est une variable booléenne qui
vaut 1 si iddoc1 et iddoc 2 sont identiques et 0 sinon.
*************************************************************/

int compareidd(iddoc1,iddoc2)
struct idd iddoc1,iddoc2;

{int result;

 if (iddoc1.crdate.year == iddoc2.crdate.year &&
     iddoc1.crdate.month == iddoc2.crdate.month &&
     iddoc1.crdate.day == iddoc2.crdate.day &&
     iddoc1.crtime.hour == iddoc2.crtime.hour &&
     iddoc1.crtime.minute == iddoc2.crtime.minute &&
     iddoc1.crtime.second == iddoc2.crtime.second &&
     iddoc1.ida == iddoc2.ida) result = 1;
 else result = 0;
 return(result);
};
```

```
/*****************************************************************
dcsaddr renvoie l'identifiant du serveur local.
*****************************************************************/

dcsaddr ()
{    FILE *fich;
     int adress;

     fich = fopen ("dcsaddr.dat", "r");
     fscanf (fich, "%d", &adress);
     fclose (fich);
     return (adress);
};

/*****************************************************************
dcsrefer renvoie l'identifiant du réseau de référence
*****************************************************************/

dcsrefer ()
{       FILE *fich;
        int adress;

        fich = fopen ("dcsrefer.dat", "r");
        fscanf (fich, "%d", &adress);
        fclose (fich);
        return (adress);
};

/*****************************************************************
politique renvoie le numéro de la politique en cours
*****************************************************************/

politique ()
{       FILE *fich;
        int polit;

        fich = fopen ("politique.dat", "r");
        fscanf (fich, "%d", &polit);
        fclose (fich);
        return (polit);
};

/*****************************************************************
stationname renvoie le nom DCS qui correspond à 'idnw'
*****************************************************************/

stationname (idnw,name)
char *name;
int idnw;

{
    switch(idnw)
    {   case 1 : sprintf(name,"            ");break;
        case 2 : sprintf(name,"            ");break;
        case 3 : sprintf(name,"            ");break;
        default : sprintf(name,"            ");break;
    };
};
```

```
/*****************************************************************
waitplus ajoute un au fichier d'attente de 'ida
*****************************************************************/

waitplus (ida)
int ida;
{     char nom [12];
      int num;

      sprintf (nom, "wait%-d", ida);
      fich = fopen (nom, "r");
      fscanf (fich, "%d", &num);
      fclose (fich);
      num++;
      fich = fopen (nom, "w");
      fprintf (fich, "%d", num);
      fclose (fich);
};


/*****************************************************************
waitmoins retranche un au fichier d'attente de 'ida'
*****************************************************************/

waitmoins (ida)
int ida;
{       char nom [12];
        int num;

        sprintf (nom, "wait%-d", ida);
        fich = fopen (nom, "r");
        fscanf (fich, "%d", &num);
        fclose (fich);
        num--;
        fich = fopen (nom, "w");
        fprintf (fich, "%d", num);
        fclose (fich);
};
```

```
                /*--------------------------*/
                /*   Module de traitement   */
                /*     logique des fichier  */
                /*--------------------------*/


/*****************************************************************
appel de la fonction : stocker (corps, ident, ldest);
objectif : stocker le corps du document d'identifiant ident et
créer la liste des destinataires de ce document;
entrées : corps : une structure line, un corps de document;
          ident : un identifiant de document;
          ldest : une liste de destinataires;
pré-conditions : corps, ident et ldest sont syntaxiquement
corrects; de plus, corps est le corps du document dont ident
est l'identifiant;
résultats : / ;
post-conditions : / .
*****************************************************************/

stocker (corps, ident, ldest)
struct line corps;
struct idd ident;
struct llnw *ldest;
{
       stockdoc (corps, ident);
        while (ldest->nextdest != 0)
        {       creerlien (ident, ldest->dest);
                ldest = ldest->nextdest;
        };
        creerlien (ident, ldest->dest);
};



/*****************************************************************
appel de la fonction : envoidoc(ident, ldest, corps, ida);
objectif : envoyer le document ident de corps corps aux
destinataires de la liste ldest;
entrées : ident : un identifiant de document;
          corps : un corps de document;
          ldest : une référence à une liste de destinataires;
pré-conditions : ident, corps et ldest sont syntaxiquement et
sémantiquement corrects;  corps est le corps du document
ident;
résultats : / ;
post-conditions : / .
*****************************************************************/

envoidoc (ident, ldest, corps)
struct idd ident;
struct llnw *ldest;
struct line corps;
{    int idnw, idnwr;
     struct llnw *inter;

     switch (politique ())
     {    case 1 : stocker (corps, ident, ldest);
                   break;
```

```
            case 3 : idnw = 0;
                  idnwr = fressuiv (idnw);
                  while (idnwr != 0)
                  {   creerliste (ident, idnwr);
                      idnw = idnwr;
                      idnwr = fressuiv (idnw);
                  };
                  while (ldest->nextdest != 0)
                  {   idnw = flocalisation (ldest->dest);
                      ajoutliste (ldest->dest, ident, idnw);
                      ldest = ldest->nextdest;
                  };
                  idnw = flocalisation (ldest->dest);
                  ajoutliste (ldest->dest, ident, idnw);
                  idnw = 0;
                  idnwr = fressuiv (idnw);
                           while (idnwr != 0)
                  {   inter = consultliste (ident, idnwr);
                                    if (idnwr == dcsaddr ())
                      {    stocker (corps, ident, inter);
                      }
                      else if (flistevide (inter) != 1)
                      envnewdocdcs(ident,corps,*inter,idnwr);
                      supprimliste (ident, idnwr);
                      idnw = idnwr;
                      idnwr = fressuiv (idnw);
                  };
                  break;
            case 4 : idnw = 0;
                  idnwr = fressuiv (idnw);
                  while (idnwr != 0)
                  {   if (idnwr != dcsaddr ())
                      envnewdocdcs(ident,corps, *ldest, idnwr);
                      else stocker (corps, ident, ldest);
                      idnw = idnwr;
                                    idnwr = fressuiv (idnw);
                           };
                           break;
            case 6 : idnw = dcsrefer ();
                  inter = ldest;
                  ldest = malloc (sizeof (ldest));
                           ldest->dest = ident->idab;
                           ldest->nextdest = inter;
                  if (idnw != dcsaddr ())
                  envnewdocdcs (ident, corps, *ldest, idnw);
                  else stocker (corps, ident, ldest);
                  break;
      };
};


/*****************************************************************
appel de la fonction : envoidsup(ident,ida);
objectif : envoyer une demande de suppression du document
ident par l'abonné ida;
entrées : ident : un identifiant de document;
          ida : un identifiant d'abonné;
pré-conditions : ident et ida sont syntaxiquement et
sémantiquement corrects; il existe un lien entre ida et ident;
```

```
résultats : / ;
post-conditions : / .
*****************************************************************/

envoidsup (ident, ida)
struct idd ident;
int ida;

{    int idnw, idar;

     localdoc (ida, ident,&idnw);
     if (idnw == dcsaddr ())
     {    supprimlien (ident, ida);
          if (fexistlien (ident) == 0) supdoc (ident);
     }
     else envdsupdcs (ident, ida, idnw);
};



/*****************************************************************
appel de la fonction : envoidcdoc(iddoc,ida);
objectif : envoi d'une demande de consultation de document par
l'abonné ida vers le réseau où le document est stocké;
entrées : iddoc : un identifiant de document;
          ida : un identifiant d'abonné;
pré-conditions : iddoc et ida sont syntaxiquement et
sémantiquement corrects; il existe un  lien entre ida et
iddoc;
résultats : / ;
post-conditions : / .
*****************************************************************/

envoidcdoc (iddoc,ida)
int ida;
struct idd iddoc;


{int idnw;
 int trouve;

 switch(politique())
 {
   case 1 : trouve = localdoc(ida,iddoc,&idnw);
            if (idnw!=dcsaddr())
            envdcdocdcs(iddoc,idnw);
            break;

   case 3 : break;

   case 4 : break;

   case 6 : if (dcsaddr()!=dcsrefer())
envdcdocdcs(iddoc,dcsrefer());
            break;
  };
};
```

```
/*****************************************************************
appel de la fonction : envoidcbal(ida);
objectif : envoi d'une demande de consultation de boîte aux
lettres pour l'abonné ida;
entrées : ida : un identifiant d'abonné;
pré-conditions : ida est syntaxiquement et sémantiquement
correct;
résultats : / ;
post-conditions : / .
*****************************************************************/

envoidcbal (ida)
int ida;


{int idnw,res;
 struct lbal *bal;

 idnw=dcsaddr();
 bal=malloc (sizeof (*bal));
 switch(politique())
 {
   case 1 : remplirbal(ida,bal);
            creerbal(ida,idnw,*bal);
            res=0;
            res=fressuiv(res);
            while (res!=0)
             { if (res!=idnw) envdcbaldcs(ida,res);
               res=fressuiv(res);
             };
            break;
   case 3 : remplirbal(ida,bal);
            creerbal(ida,idnw,*bal);
            break;

   case 4 : remplirbal(ida,bal);
          creerbal(ida,idnw,*bal);
            break;

   case 6 : envdcbaldcs(ida,dcsrefer());
            break;
 };
};


/*****************************************************************
appel de la fonction : envoibaldcs(ida,fichbal,idnw);
objectif : envoi d'une boîte aux lettres pour l'abonné ida,
dans le fichier fichbal, vers le réseau idnw;
entrées : ida : un identifiant d'abonné;
          fichbal : un nom de fichier;
          idnw : un identifiant de réseau;
pré-conditions : ida et idnw sont syntaxiquement et
sémantiquement correct;
résultats : / ;
post-conditions : / .
*****************************************************************/

envbaldcs (idab,fichbal,idnw)
int idab;
```

```
char fichbal[12];
int idnw;

{  char dest[30],source[30],cmde[60];

   sprintf(dest,"%s/usr/patain/%s",stationname(idnw),fichbal);
   sprintf(source,"/usr/patain/%s",fichbal);
   sprintf(cmde,"uucp %s %s",source,dest);
   system(cmde);
   sprintf(cmde,"uux %s/usr/patain/receptbaldcs %d
/usr/patain/%s %d",
          stationname(idnw),idab,fichbal,dcsaddr());
   system(cmde);
};


/*****************************************************************
appel de la fonction : envoidcbaldcs(idab,idnw);
objectif : envoi vers le réseau idnw d'une demande de
consultation de boîte aux lettres pour l'abonné idab;
entrées : idab : un identifiant d'abonné;
          idnw : un identifiant de réseau;
pré-conditions : idab et idnw sont syntaxiquement et
sémantiquement corrects;
résultats : / ;
post-conditions : / .
*****************************************************************/

envdcbaldcs (idab,idnw)
int idab,idnw;

{ char dest[30],cmde[60],nom[12];

  waitplus(idab);
  stationname(idnw,nom);
  sprintf(dest,"%s/usr/patain/",nom);
  printf("%s\n",dest);
  sprintf(cmde,"uux %sreceptdcbaldcs %d
%d",dest,idab,dcsaddr());
  printf("%s\n",cmde);
  system(cmde);
};


/*****************************************************************
appel de la fonction : envdcdocdcs(iddoc,idnw);
objectif : envoi vers le réseau idnw d'une demande de
consultation du document iddoc;
entrées : iddoc : un identifiant de document;
          idnw : un identifiant de réseau;
pré-conditions : iddoc et idnw sont syntaxiquement et
sémantiquement corrects;
résultats : / ;
post-conditions : / .
*****************************************************************/

envdcdocdcs (iddoc,idnw)
struct idd iddoc;
```

```
int idnw;

{ char dest[30],cmde[60],ident[25],nom[12];

  sprintf(ident,"%d %d %d %d %d %d %d",iddoc.crdate.year,
          iddoc.crdate.month, iddoc.crdate.day,
          iddoc.crtime.hour, iddoc.crtime.minute,
          iddoc.crtime.second, iddoc.ida);
  stationname(idnw,nom);
  sprintf(dest,"%s/usr/patain/",nom);
  sprintf(cmde,"uux %sreceptdcdocdcs %s %d", dest, ident,
          dcsaddr());
  system(cmde);
};


/*****************************************************************
appel de la fonction : envoidocdcs(iddoc,cd,idnw);
objectif : envoi d'un document vers le réseau idnw;
entrées : iddoc : un identifiant de document;
          idnw : un identifiant de réseau;
          cd : un corps de document;
pré-conditions : iddoc idnw et cd sont syntaxiquement et
sémantiquement corrects; cd est le corps du document dont
l'identifiant est iddoc;
résultats : / ;
post-conditions : / .
*****************************************************************/

envdocdcs (iddoc,cd,idnw)
struct idd iddoc;
struct line cd;
int idnw;

{ char dest[30],source[30],cmde[60],fichdoc[12],ident[25];

  sprintf(ident,"%d %d %d %d %d %d %d",iddoc.crdate.year,
        iddoc.crdate.month,iddoc.crdate.day,iddoc.crtime.hour,
        iddoc.crtime.minute,iddoc.crtime.second,iddoc.ida);
   sprintf(dest,"%s/usr/patain/%s",stationname(idnw),fichdoc);
   sprintf(source,"/usr/patain/%s",fichdoc);
   sprintf(cmde,"uucp %s %s",source,dest);
   system(cmde);
   sprintf(cmde,"uux %s/usr/patain/receptdocdcs %d %s
/usr/patain/%s %d",
           stationname(idnw),idab,ident,fichdoc,dcsaddr());
   system(cmde);
};



/*****************************************************************
appel de la fonction : stocorpsnew(corps,fichdoc);
objectif : stocker le corps de document dans le fichier
fichdoc;
entrées : corps : un corps de document;
          fichdoc : un nom de fichier;
pré-conditions : corps et fichdoc sont syntaxiquement et
sémantiquement corrects;
résultats : / ;
```

```
post-conditions : / .
*****************************************************************/

stocorpsnew (corps, fichdoc)
struct line corps;
char fichdoc [12];
{       struct line inter;

        sprintf (fichdoc, "ndoc%-d.tmp", dcsaddr ());
        fich = fopen (fichdoc, "w");
        inter = corps;
        do
        {       fputs (inter.l, fich);
                if (inter.nextl != 0) inter = *(inter.nextl);
        }
        while (inter.nextl != 0);
        fclose (fich);
};



/*****************************************************************
appel de la fonction : stocldest(ldest,fichldest);
objectif : stocker la liste de destinataires ldest dans le
fichier fichldest;
entrées : ldest : une liste de destinataires;
          fichldest : un fichier;
pré-conditions : ldest et fichldest sont syntaxiquement
corrects;
résultats : / ;
post-conditions : / .
*****************************************************************/

stocldest (ldest,fichldest)
struct llnw ldest;
char fichldest[12];
{       struct llnw *inter;

        sprintf (fichldest, "ldest%-d.tmp", dcsaddr ());
        fich = fopen (fichldest, "w");
        inter = &ldest;
        do
        {       fprintf (fich,"%d ",inter->dest);
                inter = inter->nextdest;
        }
        while (inter != 0);
        fclose (fich);
};



/*****************************************************************
appel de la fonction : envoinewdocdcs(iddoc,cd,ldest,idnw);
objectif : envoi vers le réseau idnw d'un nouveau document
d'identifiant iddoc, de corps cd et dont les destinataires
appartiennent à ldest;
entrées : iddoc : un identifiant de document;
          idnw : un identifiant de réseau;
          cd : un corps de document;
          ldest : une liste de destinataires;
```

```
pré-conditions : iddoc, idnw, cd et ldest sont syntaxiquement
et sémantiquement corrects;
résultats : / ;
post-conditions : / .
*****************************************************************/

envnewdocdcs (iddoc,cd,ldest,idnw)
struct idd iddoc;
struct line cd;
int idnw;
struct llnw ldest;

{ char
dest[30],source[30],cmde[60],fichdoc[12],fichldest[12],ident[2
5];
  char nom[12];

   stocorpsnew(cd,fichdoc);
   stocldest(ldest,fichldest);
   sprintf(ident,"%d %d %d %d %d %d %d",iddoc.crdate.year,
           iddoc.crdate.month, iddoc.crdate.day,
           iddoc.crtime.hour, iddoc.crtime.minute,
           iddoc.crtime.second, iddoc.ida);
   stationname(idnw,nom);
   sprintf(dest,"%s/usr/patain/%s",nom,fichdoc);
   sprintf(source,"/usr/patain/%s",fichdoc);
   sprintf(cmde,"uucp %s %s",source,dest);
   system(cmde);
   sprintf(dest,"%s/usr/patain/%s",nom,fichldest);
   sprintf(source,"/usr/patain/%s",fichldest);
   sprintf(cmde,"uucp %s %s",source,dest);
   system(cmde);
   sprintf(cmde,"uux %s/usr/patain/receptdocdcs %s
/usr/patain/%s /usr/patain/%s",
           nom,ident,fichdoc,fichldest);
   system(cmde);
   sprintf (cmde, "rm /usr/patain/%s", fichdoc);
   system (cmde);
   sprintf (cmde, "rm /usr/patain/%s", fichldest);
   system (cmde);
};



/*****************************************************************
appel de la fonction : envdsupdcs(iddoc,idab,idnw);
objectif : envoi vers le réseau d'une demande de suppression
du document iddoc pour l'abonné idab;
entrées : iddoc : un identifiant de document;
          idab : un identifiant d'abonné;
          idnw : un ientifiant de réseau;
pré-conditions : iddoc, idab et idnw sont syntaxiquement et
sémantiquement corrects; il existe un lien entre idab et
iddoc;
résultats : / ;
post-conditions : / .
*****************************************************************/

envdsupdcs (iddoc,idab,idnw)
struct idd iddoc;
```

```
int idab,idnw;

{ char dest[30],cmde[60],ident[25],nom[12];

  sprintf(ident,"%d %d %d %d %d %d %d",iddoc.crdate.year,
          iddoc.crdate.month, iddoc.crdate.day,
          iddoc.crtime.hour, iddoc.crtime.minute,
          iddoc.crtime.second, iddoc.ida);
  stationname(idnw,nom);
  sprintf(dest,"%s/usr/patain/",nom);
  sprintf(cmde,"uux %sreceptdsupdcs %s %d ",dest,ident,idab);
  system(cmde);
};
```

```
/*************************************/
/* Programme de création d'abonné */
/*   et de réseau                    */
/*************************************/


/*****************************************************************
appel du programme : creerab;
objectif  : creerab  permet l'insertion et la localisation d'un
nouvel  abonné sur le courrier électronique et l'insertion d'un
réseau supplémentaire dans le système;
*****************************************************************/

#include <stdio.h>

main ()

{ int out,rep,ok,cont;
  int idab,idnw;

 out=0;
 while (out==0)
  { rep=9;
    while ((rep!=1)&&(rep!=2)&&(rep!=3))
     { printf("voulez-vous :\n");
       printf("1 : creer un nouvel abonne \n");
       printf("2 : creer un nouveau reseau \n");
       printf("3 : sortir \n");
       scanf ("%d",&rep);
     };
    switch (rep)
     {  case 1 : ok=0;
                 cont=1;
                   while ((ok==0)&&(cont==1))
            { printf("\nle nouvel abonne ? : ");
                    scanf("%d",&idab);
              if (existab(idab)==1)
              { printf("\nl'abonne %d existe deja\n",idab);
                cont=0;}
                    else { printf("\nson reseau ? : ");
                       scanf("%d",&idnw);
                 printf("\n");
                      if (existres(idnw)!=0)
                      { ok=1;
                        localab(idab,idnw);}
                          else printf("le reseau %d n'existe
                                       pas\n", idnw);
                  };
            };
              break;
        case 2 : printf("\n nouveau reseau ? : ");
                 scanf("%d",&idnw);
                 if (existres(idnw)==0)
                 localres(idnw);
                 else printf("le reseau %d existe deja \n",
                              idnw);
                 break;
        case 3 : out = 1;
                 break;
```

```
        };
    };
}


/*****************************************************************
appel de la fonction : existres(idnw);
objectif : dire si un réseau donné existe déjà dans le
système;
entrées : idnw : un identifiant de réseau;
pré-conditions : idnw est syntaxiquement correct;
résultats : existres : entier;
post-conditions : existres est une variable booléenne qui vaut
1 si le réseau idnw existe déjà et 0 sinon.
*****************************************************************/

int existres(idnw)
int idnw;

{ int idnwr,existe,cont;
  FILE *fich,*fopen();

  fich = fopen ("fsys.dat","r");
  if (fich == 0)
  existe = 0;
  else { if (fscanf(fich,"%d",&idnwr)==EOF)
         existe = 0;
         else { cont = 1;
                while ((cont==1)&&(idnwr!=idnw))
                { if (fscanf(fich,"%d",&idnwr)==EOF)
                  cont = 0; };
                if (cont == 0)
                existe = 0;
                else existe = 1;
              };
       };
  fclose(fich);
  return(existe);
};
/*****************************************************************
appel de la fonction : existab(idab);
objectif : dire si un abonné donné existe déjà pour le CE;
entrées : idab : un identifiant d'abonné;
pré-conditions : idab est syntaxiquement correct;
résultats : existab : entier;
post-conditions : existab est une variable booléenne qui vaut
1 si l'abonné idab existe dans le CE et 0 sinon.
*****************************************************************/

int existab(idab)
int idab;

{ int idabr,idnwr,existe,cont;
  FILE *fich,*fopen();

  fich = fopen("flocalab.dat","r");
  if (fich==0)
  existe = 0;
  else { if (fscanf(fich,"%d %d",&idabr,&idnwr)==EOF)
```

```
        existe = 0;
        else { cont = 1;
               while ((cont==1)&&(idabr!=idab))
               { if (fscanf(fich,"%d %d",&idabr,&idnwr)==EOF)
                 cont = 0; };
               if (cont == 0)
               existe = 0;
               else existe = 1;
             };
      };
  fclose(fich);
  return(existe);
};


/*****************************************************************
appel de la procédure : localab(idab,idnw);
objectif : permettre l'insertion et la localisation d'un
abonné dans le CE, par son insertion dans le fichier
flocalab.dat.
*****************************************************************/

localab (idab,idnw)
int idab,idnw;

{ FILE *fich,*fopen();

  fich = fopen("flocalab.dat","a");
  fprintf(fich,"%d %d \n",idab,idnw);
  fclose(fich);
};


/*****************************************************************
appel de la procédure : localres(idnw);
objectif : permettre l'insertion d'un réseau dans le système,
par son insertion dans le fichier fsys.dat.
*****************************************************************/

localres (idnw)
int idnw;

{ FILE *fich,*fopen();

  fich = fopen("fsys.dat","a");
  fprintf(fich," %d\n",idnw);
  fclose(fich);
};
```

```
/*************************************************************
*
receptbaldcs (ida, nomfich, idnw)
     données: 'ida', un identifiant d'abonné,
              'nomfich', un nom de fichier,
              'idnw', un identifiant de réseau;
     préconditions: les données sont syntaxiquement correctes;
     fonction:  complète la  boîte aux lettres de 'ida' avec la
                boîte   aux  lettres   reçue  dans   le  fichier
                'nomfich'  et originaire du réseau 'idnw'; met à
                jour le fichier d'attente de 'ida'.
*************************************************************
/

#include <stdio.h>
#include "declarations.c"
#include "utilbis.c"

main (argc, argv)
int argc;
char *argv[];
{    int ida, idnw;
     struct lbal *bal;
     char nomfich [25];

     sscanf (*++argv, "%d", &ida);
     sscanf (*++argv, "%s", nomfich);
     sscanf (*++argv, "%d", &idnw);
     bal = malloc (sizeof (*bal));
     chargbal (nomfich, bal);
     complbal (ida, idnw, *bal);
     waitmoins (ida);
}

chargbal (nomfich, bal)
char nomfich [25];
struct lbal *bal;
{    struct lbal *inter, *derniere;
     int rien;

     fich = fopen (nomfich, "r");
     inter = bal;
     derniere = bal;
     fscanf (fich, "%d %d %d %d %d %d %d\n",
      &inter->iddoc.crdate.year, &inter->iddoc.crdate.month,
      &inter->iddoc.crdate.day, &inter->iddoc.crtime.hour,
      &inter->iddoc.crtime.minute, &inter-
>iddoc.crtime.second,
      &inter->iddoc.ida, &rien);
     while (feof (fich) == 0)
     {    inter->nextiddoc = malloc (sizeof (*inter));
          inter = inter->nextiddoc;
          derniere = inter;
          fscanf (fich, "%d %d %d %d %d %d %d\n",
      &inter->iddoc.crdate.year, &inter->iddoc.crdate.month,
      &inter->iddoc.crdate.day, &inter->iddoc.crtime.hour,
      &inter->iddoc.crtime.minute, &inter-
>iddoc.crtime.second,
      &inter->iddoc.ida, &rien);
```

```
        };
        derniere->nextiddoc = 0;
        fclose (fich);
}


/*** voir spécifications dans le module boîte aux lettres ***/

complbal (ida, idnw, bal)
int ida;
int idnw;
struct lbal bal;
{       char nom[25];
        FILE *fich, *fopen ();
        struct lbal inter;

        sprintf (nom, "/usr/patain/bal%-d.dat", ida);
        fich = fopen (nom, "a");
        inter = bal;
        while (inter.nextiddoc != 0)
        {    fprintf (fich, "%d %d %d %d %d %d %d %d\n",

inter.iddoc.crdate.year,inter.iddoc.crdate.month,
              inter.iddoc.crdate.day,inter.iddoc.crtime.hour,
          inter.iddoc.crtime.minute,
inter.iddoc.crtime.second,
              inter.iddoc.ida, idnw);
             inter = *(inter.nextiddoc);
        }
        if (compareidd (inter.iddoc, idvide) == FALSE)
        fprintf (fich, "%d %d %d %d %d %d %d %d\n",
           inter.iddoc.crdate.year, inter.iddoc.crdate.month,
           inter.iddoc.crdate.day,inter.iddoc.crtime.hour,

inter.iddoc.crtime.minute,inter.iddoc.crtime.second,
           inter.iddoc.ida, idnw);
        fclose (fich);
}


/*** voir spécifications dans le module boîte aux lettres ***/

int compareidd (iddoc1, iddoc2)
struct idd iddoc1,iddoc2;

/* compareidd renvoie la valeur 1 si iddoc1=iddoc2 et la
valeur 0 sinon */

{int result;

 if (iddoc1.crdate.year == iddoc2.crdate.year &&
     iddoc1.crdate.month == iddoc2.crdate.month &&
     iddoc1.crdate.day == iddoc2.crdate.day &&
     iddoc1.crtime.hour == iddoc2.crtime.hour &&
     iddoc1.crtime.minute == iddoc2.crtime.minute &&
     iddoc1.crtime.second == iddoc2.crtime.second &&
     iddoc1.ida == iddoc2.ida) result = 1;
 else result = 0;
 return(result);
}
```

```
/*************************************************************
receptdbaldcs (ida, idnw)
     données: 'ida', un identifiant d'abonné,
              'idnw', un identifiant de réseau;
     préconditions:  les données sont syntaxiquement et
                     sémantiquement correctes;
     fonction:  crée un boîte aux lettres pour 'ida', et la
                renvoie vers 'idnw'.
*************************************************************/

#include <stdio.h>
#include "declarations.c"
#include "utilbis.c"

main (argc, argv)
int argc;
char *argv[];
{    int ida, idnw;
     char nomfich [12], cmde [30];
     struct idd ident;
     struct idd *identr;
     struct lbal *bal;

     sscanf (*++argv, "%d", &ida);
     sscanf (*++argv, "%d", &idnw);
     bal = malloc (sizeof (*bal));
     remplirbal (ida, bal);
     creerbal (ida, idnw, *bal);
     sprintf (cmde, "mv bal%-d.dat bal%-d.tmp", ida, ida);
     system (cmde);
     sprintf (nomfich, "bal%-d.tmp", ida);
     envoibaldcs (ida, nomfich, idnw);
};


/***** voir spécifications dans le module utilisateur ******/

struct idd *docsuiv(iddoc,idab)
int idab;
struct idd iddoc;

/* docsuiv fournit le document suivant iddoc et relatif a ida
*/
/* le resultat est fourni dans la variable globale iddocr */

{int finfich,idabr,ok;
 extern struct idd idvide,iddocr;

 fich=fopen("fliens.dat","r");
 if (fich==0)
   return(&idvide);
 if (fscanf(fich,"%d %d %d %d %d %d %d %d ",
       &idabr,&iddocr.crdate.year,&iddocr.crdate.month,
       &iddocr.crdate.day,&iddocr.crtime.hour,

&iddocr.crtime.minute,&iddocr.crtime.second,&iddocr.ida)
     == EOF)
   return(&idvide);
 if (compareidd(iddoc,idvide)==TRUE)
```

```
    {finfich=FALSE;
     while ((idab!=idabr)&&(finfich==FALSE))
      {if (fscanf(fich,"%d %d %d %d %d %d %d %d ",
             &idabr,&iddocr.crdate.year,&iddocr.crdate.month,
             &iddocr.crdate.day,&iddocr.crtime.hour,
             &iddocr.crtime.minute,&iddocr.crtime.second,
             &iddocr.ida)
           ==EOF)finfich=TRUE;
      };
     if (finfich==TRUE)
         return(&idvide);
       return(&iddocr);
    }
 else {while
((idabr!=idab)||(compareidd(iddoc,iddocr)==FALSE))
        {fscanf(fich,"%d %d %d %d %d %d %d %d
",&idabr,&iddocr.
             crdate.year,&iddocr.crdate.month,&iddocr.crdate.
             day,&iddocr.crtime.hour,&iddocr.crtime.minute,
             &iddocr.crtime.second,&iddocr.ida);
         };
       if (fscanf(fich,"%d %d %d %d %d %d %d %d ",
             &idabr,&iddocr.crdate.year,&iddocr.crdate.month,
             &iddocr.crdate.day,&iddocr.crtime.hour,
             &iddocr.crtime.minute,
             &iddocr.crtime.second,&iddocr.ida)==EOF)
         finfich=TRUE;
       else finfich=FALSE;
       while ((finfich==FALSE)&&(idab!=idabr))
         {if (fscanf(fich,"%d %d %d %d %d %d %d %d ",
                &idabr,&iddocr.crdate.year,&iddocr.crdate.
                month,&iddocr.crdate.day,&iddocr.crtime.hour,
                &iddocr.crtime.minute,&iddocr.crtime.second,
                &iddocr.ida)==EOF)
          finfich=TRUE;};
          if (finfich==TRUE)
              return(&idvide);
            return(&iddocr);
             };
};

int compareidd (iddoc1, iddoc2)
struct idd iddoc1,iddoc2;

/* compareidd renvoie la valeur 1 si iddoc1=iddoc2 et la
valeur 0 sinon */

{int result;

 if (iddoc1.crdate.year == iddoc2.crdate.year &&
     iddoc1.crdate.month == iddoc2.crdate.month &&
     iddoc1.crdate.day == iddoc2.crdate.day &&
     iddoc1.crtime.hour == iddoc2.crtime.hour &&
     iddoc1.crtime.minute == iddoc2.crtime.minute &&
     iddoc1.crtime.second == iddoc2.crtime.second &&
     iddoc1.ida == iddoc2.ida) result = 1;
 else result = 0;
 return(result);
};
```

```
/*****************************************************************
voir spécifications dans le module traitement logique fichier
*****************************************************************/

envoibaldcs (idab, fichbal, idnw)
int idab;
char fichbal[12];
int idnw;

{  char dest[30];
   char source[30];
   char cmde[75];
   char cmde2[75];
   char nom[12];
   char prog[70];

   stationname (idnw, nom);
   sprintf(dest,"%s/usr/patain/r%s",nom,fichbal);
   sprintf(source,"/usr/patain/%s",fichbal);
   sprintf(cmde,"uucp %s %s",source,dest);
   printf("%s\n",cmde);
   system(cmde);
   sprintf(cmde,"
");
   sprintf (prog, "%s /usr/patain/receptbaldcs %d
/usr/patain/%s %d", nom, idab, fichbal, dcsaddr());
   printf ("%s\n", prog);
   sprintf (cmde2, "uux %s", prog);
   printf("%s\n",cmde2);
   system(cmde2);
   sprintf (cmde, "rm /usr/patain/%s", fichbal);
   system (cmde);
};


/*** voir spécifications dans le module boîte aux lettres ***/

creerbal (ida, idnw, bal)
int ida;
int idnw;
struct lbal bal;
{       char nom[12], comde[15];
        FILE *fich, *fopen ();
        struct lbal inter;

        sprintf (nom, "bal%-d.dat", ida);
        if (existbal (ida) == 1)
        {       sprintf (comde, "rm %s", nom);
                system (comde);
        };
        fich = fopen (nom, "w");
        inter = bal;
        while (inter.nextiddoc != 0)
        {       fprintf (fich, "%d %d %d %d %d %d %d %d\n",
           inter.iddoc.crdate.year,inter.iddoc.crdate.month,
           inter.iddoc.crdate.day,inter.iddoc.crtime.hour,

inter.iddoc.crtime.minute,inter.iddoc.crtime.second,
           inter.iddoc.ida, idnw);
```

```
                inter = *(inter.nextiddoc);
        };
        if (compareidd (inter.iddoc, idvide) == FALSE)
        fprintf (fich, "%d %d %d %d %d %d %d %d\n",
           inter.iddoc.crdate.year,inter.iddoc.crdate.month,
           inter.iddoc.crdate.day,inter.iddoc.crtime.hour,

inter.iddoc.crtime.minute,inter.iddoc.crtime.second,
           inter.iddoc.ida, idnw);
        fclose (fich);
};


/*** voir spécifications dans le module boîte aux lettres ***/

int existbal (ida)
int ida;
{       char nom[12];
        FILE *fich, *fopen();
        int trouve;

        sprintf (nom, "bal%-d.dat", ida);
        fich = fopen (nom, "r");
        if (fich != 0)
        { trouve = 1;
          fclose (fich);}
        else trouve = 0;
        return (trouve);
};


/*** voir spécifications dans le module boîte aux lettres ***/

remplirbal (ida, bal)
int ida;
struct lbal *bal;
{       struct lbal *inter;
        struct lbal *dernier;
        struct idd *iddocr;

        inter = bal;
        inter->iddoc = idvide;
        dernier = inter;
        iddocr = docsuiv (idvide, ida);
        while (compareidd(idvide,*iddocr) != TRUE)
        {       inter->iddoc = *iddocr;
                dernier = inter;
                inter->nextiddoc = malloc (sizeof (*inter));
                inter = inter->nextiddoc;
                iddocr = docsuiv (*iddocr, ida);
        };
        dernier->nextiddoc = 0;
};
```

```
/*************************************************************
receptddocdcs (ident, idnw)
     données: 'ident', un identifiant de document,
              'idnw', un identifiant de réseau;
     postconditions:   les  données   sont  syntaxiquement   et
                      sémantiquement correctes;
     fonction:  extrait le document d'identifiant 'ident' et le
                renvoie au réseau 'idnw'.
*************************************************************/

#include <stdio.h>
#include "declarations.c"
#include "utilbis.c"

main (argc, argv)
int argc;
char *argv[];
{    int idnw;
     struct idd ident;
     struct line *corps;
     char *nom;

     sscanf (*++argv, "%d", &ident.crdate.year);
     sscanf (*++argv, "%d", &ident.crdate.month);
     sscanf (*++argv, "%d", &ident.crdate.day);
     sscanf (*++argv, "%d", &ident.crtime.hour);
     sscanf (*++argv, "%d", &ident.crtime.minute);
     sscanf (*++argv, "%d", &ident.crtime.second);
     sscanf (*++argv, "%d", &ident.ida);
     sscanf (*++argv, "%d", &idnw);
     cherchedoc (ident, corps);
     envoidocdcs (ident, *corps, idnw);
}

/*************************************************************
*  voir les  spécifications dans  le module  traitement logique
fichier
*************************************************************
/

envoidocdcs (iddoc, corps, idnw)
struct idd iddoc;
struct line corps;
int idnw;
{ char dest[30],source[30],cmde[60],fichdoc[12],nom[12];

   stocorps(corps, fichdoc);
   stationname (idnw, nom);
   sprintf(dest,"%s!/usr/patain/%s",nom,fichdoc);
   sprintf(source,"/usr/patain/%s",fichdoc);
   sprintf(cmde,"uucp %s %s",source,dest);
   system(cmde);
   sprintf(cmde,"uux %s/usr/patain/receptdocdcs
      %d %d %d %d %d %d %d %d /usr/patain/%s %d",
      nom,idab,iddoc.crdate.year,
      iddoc.crdate.month,iddoc.crdate.day,iddoc.crtime.hour,
      iddoc.crtime.minute,iddoc.crtime.second,iddoc.ida,
      fichdoc,dcsaddr());
   system(cmde);
```

```
    sprintf (cmde, "rm /usr/patain/%s", fichdoc);
    system (cmde);
}


/******* voir spécifications dans le module document *******/

stocorps (corps, fichdoc)
struct line corps;
char fichdoc [12];
{    struct line inter;

     sprintf (fichdoc, "doc%-d.tmp", dcsaddr ());
     fich = fopen (fichdoc, "w");
     inter = corps;
     do
     {    fputs (inter.l, fich);
          if (inter.nextl != 0) inter = *(inter.nextl);
     }
     while (inter.nextl != 0);
     fclose (fich);
}


/******* voir spécifications dans le module document *******/

int cherchedoc (ident, corps)
struct idd ident;
struct line *corps;
{       char nom [12];

        if (cherchnom (ident, nom) == 0) return (0);
        else {  cherchcor (nom, corps);
                return (1);
             };
}


/******* voir spécifications dans le module document *******/

int cherchnom (ident, nom)
struct idd ident;
char *nom;
{       FILE *fopen (), *fich;
        struct idd inter;
        int trouve;

        fich = fopen ("document.idx", "r");
        fscanf (fich, "%d %d %d %d %d %d %d %s",
          &inter.crdate.year,
          &inter.crdate.month, &inter.crdate.day,
          &inter.crtime.hour,
          &inter.crtime.minute, &inter.crtime.second,
          &inter.ida, nom);
        trouve = 0;
        while (feof (fich) == 0 && trouve == 0)
        {       if (compiddo (ident, inter) == 0) trouve = 1;
                else fscanf (fich, "%d %d %d %d %d %d %d %s",
                        &inter.crdate.year,
&inter.crdate.month,
```

```
                        &inter.crdate.day, &inter.crtime.hour,
                        &inter.crtime.minute,
                        &inter.crtime.second,
                        &inter.ida, nom);
        };
        fclose (fich);
        return (trouve);
}


/******* voir spécifications dans le module document *******/

cherchcor (nom, corps)
char nom[12];
struct line *corps;
{       FILE *fopen (), *fich;
        struct line *inter, derniere;

        fich = fopen (nom, "r");
        inter = corps;
        fgets (inter->l, MAXL, fich);
        while (feof (fich) == 0)
        {       inter->nextl = malloc (sizeof (*inter));
                derniere = *inter;
                inter = inter->nextl;
                fgets (inter->l, MAXL, fich);
        };
        derniere.nextl = 0;
        fclose (fich);
}


/******* voir spécifications dans le module document *******/

compiddo (ident1, ident2)
struct idd ident1;
struct idd ident2;
{       int trouve;

        if (ident1.crtime.hour == ident2.crtime.hour)
        if (ident1.crtime.minute == ident2.crtime.minute)
        if (ident1.crtime.second == ident2.crtime.second)
        if (ident1.crdate.year == ident2.crdate.year)
        if (ident1.crdate.month == ident2.crdate.month)
        if (ident1.crdate.day == ident2.crdate.day)
        if (ident1.ida == ident2.ida) return (0);
        else return (1);
}
```

```
/*****************************************************************
*
receptdocdcs (ida, ident, nomfich, idnw)
     données: 'ida', un identifiant d'abonné,
              'ident', un identifiant de document,
              'nomfich', un nom de fichier,
              'idnw', un identifiant de réseau;
     préconditions:  les données sont syntaxiquement et
                     sémantiquement correctes;
     fonction:  stocke le document d'identifiant 'ident' reçu
                dans le fichier 'nomfich' et originaire du
                réseau 'idnw'; met à jour le fichier d'attente
                de 'ida'.
*****************************************************************
/

#include <stdio.h>
#include "declarations.c"
#include "utilbis.c"

main (argc, argv)
int argc;
char *argv[];
{    int ida, idnw;
     struct idd ident;
     char nomfich [25];
     struct line *corps;

     sscanf (*++argv, "%d", &ida);
     sscanf (*++argv, "%d", &ident.crdate.year);
     sscanf (*++argv, "%d", &ident.crdate.month);
     sscanf (*++argv, "%d", &ident.crdate.day);
     sscanf (*++argv, "%d", &ident.crtime.hour);
     sscanf (*++argv, "%d", &ident.crtime.minute);
     sscanf (*++argv, "%d", &ident.crtime.second);
     sscanf (*++argv, "%d", &ident.ida);
     sscanf (*++argv, "%s", nomfich);
     sscanf (*++argv, "%d", &idnw);
     corps = malloc (sizeof (*corps));
     chargerdoc (nomfich, corps);
     stockdoc (*corps, ident);
     waitmoins (ida);
}

chargerdoc (nomfich, corps)
char nomfich [25];
struct line *corps;
{       struct line *inter, derniere;

        fich = fopen (nomfich, "r");
        inter = corps;
        derniere = *corps;
        fgets (inter->l, MAXL, fich);
        while (feof (fich) == 0)
        {       inter->next1 = malloc (sizeof (*inter));
                derniere = *inter;
                inter = inter->next1;
                fgets (inter->l, MAXL, fich);
        };
```

```
        derniere.nextl = 0;
        fclose (fich);
}


/****** voir les spécifications dans le module document
******/

stockdoc (corps, ident)
struct line corps;
struct idd ident;
{       char nom[12];

        stocor (corps, nom);
        stocid (ident, nom);
}


/****** voir les spécifications dans le module document
******/

stocor (corps, nom)
struct line corps;
char nom[12];
{       struct line inter;
        FILE *fopen (), *fich;

        nomsuiv (nom);
        fich = fopen (nom, "w");
        inter = corps;
        do
        {       fputs (inter.l, fich);
                if (inter.nextl != 0) inter = *(inter.nextl);
        }
        while (inter.nextl != 0);
        fclose (fich);
}


/****** voir les spécifications dans le module document
******/

stocid (ident, nom)
struct idd ident;
char nom[12];
{       FILE *fopen (), *fich;

        fich = fopen ("document.idx", "a");
        fprintf (fich, "%d %d %d %d %d %d %d %s\n",
           ident.crdate.year, ident.crdate.month,
           ident.crdate.day, ident.crtime.hour,
           ident.crtime.minute, ident.crtime.second,
ident.ida,
           nom);
        fclose (fich);
}


/****** voir les spécifications dans le module document
******/
```

```
nomsuiv (nom)
char nom[12];
{       int numero;
        FILE *fich, *fopen ();

        fich = fopen ("dernier.dat", "r");
        fscanf (fich, "%d", &numero);
        fclose (fich);
        numero++;
        fich = fopen ("dernier.dat", "w");
        fprintf (fich, "%d", numero);
        fclose (fich);
        sprintf (nom, "f%-d.dat", numero);
}
```

```
/*****************************************************************
receptnewdocdcs (ident, ida, fichdoc, fichdest)
     données: 'ident', un identifiant de document,
              'ida', un identifiant d'abonné,
              'fichdoc', un fichier contenant un document,
               'fichdest', un fichier contenant une liste de
              destinataires;
     préconditions:  les données sont syntaxiquement et
                     sémantiquement correctes;
     fonction: stocke le document contenu dans le fichier
              'fichdoc' et d'identifiant 'ident'; crée les
              liens entre 'ident' et les abonnés de la liste
              contenue dans 'fichdest', plus l'expéditeur
              'ida'.
*****************************************************************
/

#include <stdio.h>
#include "declarations.c"

main (argc, argv)
int argc;
char *argv[];
{    struct idd ident;
     struct line *corps;
     struct llnw *ldest;
     char fichdoc [25], fichdest [25];

     sscanf (*++argv, "%d", &ident.crdate.year);
     sscanf (*++argv, "%d", &ident.crdate.month);
     sscanf (*++argv, "%d", &ident.crdate.day);
     sscanf (*++argv, "%d", &ident.crtime.hour);
     sscanf (*++argv, "%d", &ident.crtime.minute);
     sscanf (*++argv, "%d", &ident.crtime.second);
     sscanf (*++argv, "%d", &ident.ida);
     sscanf (*++argv, "%s", fichdoc);
     sscanf (*++argv, "%s", fichdest);
     corps = malloc (sizeof (*corps));
     chargerdoc (fichdoc, corps);
     ldest = malloc (sizeof (*ldest));
     chargerdest (fichdest, ldest);
     stocker (*corps, ident, ldest);
}

chargerdoc (nomfich, corps)
char nomfich [25];
struct line *corps;
{       struct line *inter, derniere;
     char cmde [25];

        fich = fopen (nomfich, "r");
     inter = corps;
     derniere = *corps;
        fgets (inter->l, MAXL, fich);
        while (feof (fich) == 0)
        {       inter->nextl = malloc (sizeof (*inter));
                derniere = *inter;
                inter = inter->nextl;
                fgets (inter->l, MAXL, fich);
```

```
        };
        derniere.nextl = 0;
        fclose (fich);
    sprintf (cmde, "rm /usr/patain/%s", nomfich);
    system (cmde);
}

chargerdest (fichdest, ldest)
char fichdest [25];
struct llnw *ldest;
{   struct llnw *inter, *dernier;
    char cmde [25];

    fich = fopen (fichdest, "r");
    inter = ldest;
    dernier = ldest;
    fscanf (fich, "%d", &inter->dest);
    while (feof (fich) == 0)
    {   inter->nextdest = malloc (sizeof (*inter));
        dernier = inter;
        inter = inter->nextdest;
        fscanf (fich, "%d", &inter->dest);
    };
    dernier->nextdest = 0;
    fclose (fich);
    sprintf (cmde, "rm /usr/patain/%s", fichdest);
    system (cmde);
}


/*****************************************************************
*  voir les  spécifications dans  le module  traitement logique
fichier.
*****************************************************************
/

stocker (corps, ident, ldest)
struct line corps;
struct idd ident;
struct llnw *ldest;
{   stockdoc (corps, ident);
    while (ldest->nextdest != 0)
    {   creerlien (ident, ldest->dest);
        ldest = ldest->nextdest;
    };
    creerlien (ident, ldest->dest);
}


/**** voir les spécifications dans le module utilisateur ***/

creerlien (iddoc, idab)
struct idd iddoc;
int idab;
{   fich = fopen ("fliens.dat", "a");
    fprintf (fich, "%d %d %d %d %d %d %d %d\n", idab,
    iddoc.crdate.year, iddoc.crdate.month, iddoc.crdate.day,
    iddoc.crtime.hour,iddoc.crtime.minute,iddoc.crtime.second
,
    iddoc.ida);
```

```
        fclose (fich);
}


/****** voir les spécifications dans le module document ****/

stockdoc (corps, ident)
struct line corps;
struct idd ident;
{       char nom[12];

        stocor (corps, nom);
        stocid (ident, nom);
}


/****** voir les spécifications dans le module document *****/

stocor (corps, nom)
struct line corps;
char nom[12];
{       struct line inter;
        FILE *fopen (), *fich;

        nomsuiv (nom);
        fich = fopen (nom, "w");
        inter = corps;
        do
        {       fputs (inter.l, fich);
                if (inter.nextl != 0) inter = *(inter.nextl);
        }
        while (inter.nextl != 0);
        fclose (fich);
}


/****** voir les spécifications dans le module document *****/

stocid (ident, nom)
struct idd ident;
char nom[12];
{       FILE *fopen (), *fich;

        fich = fopen ("document.idx", "a");
        fprintf (fich, "%d %d %d %d %d %d %d %s\n",
         ident.crdate.year,
         ident.crdate.month, ident.crdate.day,
         ident.crtime.hour,
         ident.crtime.minute, ident.crtime.second, ident.ida,
         nom);
        fclose (fich);
}


/****** voir les spécifications dans le module document *****/

nomsuiv (nom)
char nom[12];
{       int numero;
        FILE *fich, *fopen ();
```

```
        fich = fopen ("dernier.dat", "r");
        fscanf (fich, "%d", &numero);
        fclose (fich);
        numero++;
        fich = fopen ("dernier.dat", "w");
        fprintf (fich, "%d", numero);
        fclose (fich);
        sprintf (nom, "f%-d.dat", numero);
}
```

```
/*************************************************************
receptdsupdcs (ident, ida)
     données: 'ident', un identifiant de document,
              'ida', un identifiant d'abonné;
     préconditions:   les données sont syntaxiquement et
                      sémantiquement correctes;
     fonction:  supprime le lien entre 'ida' et 'ident' dans le
                fichier des liens; si nécessaire, supprime le
                document d'identifiant 'ident'.
*************************************************************
/

#include <stdio.h>
#include "declarations.c"

main (argc, argv)
int argc;
char *argv[];
{    int ida;
     struct idd ident;

     sscanf (*++argv, "%d", &ident.crdate.year);
     sscanf (*++argv, "%d", &ident.crdate.month);
     sscanf (*++argv, "%d", &ident.crdate.day);
     sscanf (*++argv, "%d", &ident.crtime.hour);
     sscanf (*++argv, "%d", &ident.crtime.minute);
     sscanf (*++argv, "%d", &ident.crtime.second);
     sscanf (*++argv, "%d", &ident.ida);
     sscanf (*++argv, "%d", &ida);
     supprimlien (ident, ida);
     if (fconsultnextlien (ident, ida) == 0)
     supdoc (ident);
}


/*** voir les spécifications dans le module utilisateur ****/

supprimlien (iddoc, idab)
struct idd iddoc;
int idab;

/* supprimlien supprime le lien existant entre iddoc et idab
*/

{struct idd iddocr;
 int idabr;
 FILE *fich2;

 fich=fopen("fliens.dat","r");
 fich2=fopen("fliens2.dat","w");
 fscanf(fich,"%d %d %d %d %d %d %d %d ",
   &idabr,&iddocr.crdate.year,

&iddocr.crdate.month,&iddocr.crdate.day,&iddocr.crtime.hour,
   &iddocr.crtime.minute,&iddocr.crtime.second,&iddocr.ida);
 while((compiddo(iddocr,iddoc)==0)||(idab!=idabr))
   {fprintf(fich2,"%d %d %d %d %d %d %d %d ",
      idabr,iddocr.crdate.year,
```

```
iddocr.crdate.month,iddocr.crdate.day,iddocr.crtime.hour,
      iddocr.crtime.minute,iddocr.crtime.second,iddocr.ida);
    fscanf(fich,"%d %d %d %d %d %d %d %d ",
      &idabr,&iddocr.crdate.year,

&iddocr.crdate.month,&iddocr.crdate.day,&iddocr.crtime.hour,
   &iddocr.crtime.minute,&iddocr.crtime.second,&iddocr.ida);
   };
 while(fscanf(fich,"%d %d %d %d %d %d %d %d ",
        &idabr,&iddocr.crdate.
        year,&iddocr.crdate.month,&iddocr.crdate.day,&iddocr.

crtime.hour,&iddocr.crtime.minute,&iddocr.crtime.second,
        &iddocr.ida)!=EOF)
   {fprintf(fich2,"%d %d %d %d %d %d %d %d ",
      idabr,iddocr.crdate.year,

iddocr.crdate.month,iddocr.crdate.day,iddocr.crtime.hour,
      iddocr.crtime.minute,iddocr.crtime.second,iddocr.ida);
   };
 fclose(fich);
 fclose(fich2);
 system("cp fliens2.dat fliens.dat");
 system("rm fliens2.dat");
}


/**** voir les spécifications dans le module utilisateur ****/

int fconsultnextlien (iddoc, idab)
struct idd iddoc;
int idab;

/* consultnextlien fournit l'abonne suivant pour un document
donne*/

{int ok,idabr;
 struct idd iddocr;

 fich=fopen("fliens.dat","r");
 fscanf(fich,"%d %d %d %d %d %d %d %d ",
   &idabr,&iddocr.crdate.year,

&iddocr.crdate.month,&iddocr.crdate.day,&iddocr.crtime.hour,
   &iddocr.crtime.minute,&iddocr.crtime.second,&iddocr.ida);
 while (compiddo(iddocr,iddoc)==0)
   {fscanf(fich,"%d %d %d %d %d %d %d %d ",
      &idabr,&iddocr.crdate.year,

&iddocr.crdate.month,&iddocr.crdate.day,&iddocr.crtime.hour,

&iddocr.crtime.minute,&iddocr.crtime.second,&iddocr.ida);};
 if (idab!=0)  {while ((idab!=idabr) ||
(compiddo(iddoc,iddocr)
                  == 0))
                 {fscanf(fich,"%d %d %d %d %d %d %d %d ",
                   &idabr,&iddocr.crdate.year,
                   &iddocr.crdate.month, &iddocr.crdate.day,
                   &iddocr.crtime.hour,&iddocr.
```

```
                    crtime.minute,&iddocr.crtime.second,
                    &iddocr.ida);
                  };
                if (fscanf(fich,"%d %d %d %d %d %d %d %d ",
                    &idabr,&iddocr.crdate.year,
                    &iddocr.crdate.month, &iddocr.crdate.
                    day, &iddocr.crtime.hour,
                    &iddocr.crtime.minute,
                    &iddocr.crtime.second,&iddocr.ida)!=EOF)
                {ok=1;
                 while ((compiddo(iddocr,iddoc)==0) &&
(ok==1))
                  {if (fscanf(fich,"%d %d %d %d %d %d %d %d ",
                       &idabr, &iddocr.crdate.year,
                       &iddocr.crdate.month,
                       &iddocr.crdate.day, &iddocr.crtime.
                       hour,&iddocr.crtime.minute,&iddocr.
                       crtime.second,&iddocr.ida)==EOF)
                   ok=0;};
                }
                else ok=0;
                if (ok==0) idabr=0;
               };
 return(idabr);
}


/******* voir spécifications dans le module document *******/

supdoc (ident)
struct idd ident;
{       char nom [12], comde[20];

        printf ("Entree dans supdoc\n");
        supidd (ident, nom);
        printf ("%s\n", nom);
        sprintf (comde, "rm %s", nom);
        system (comde);
        printf ("Sortie de supdoc\n");
}


/******* voir spécifications dans le module document *******/

supidd (ident, nom)
struct idd ident;
char *nom;
{       char nominter[12];
        FILE *fopen (), *fich1, *fich2;
        struct idd inter;

        fich1 = fopen ("document.idx", "r");
        fich2 = fopen ("temporai.idx", "w");
        fscanf (fich1, "%d %d %d %d %d %d %d %s",
          &inter.crdate.year,
          &inter.crdate.month, &inter.crdate.day,
          &inter.crtime.hour,
          &inter.crtime.minute, &inter.crtime.second,
          &inter.ida, nominter);
        while (feof (fich1) == 0)
```

```
        {       if (compiddo (ident, inter) != 0)
                {       fprintf (fich2, "%d %d %d %d %d %d %d
%s\n",
                        inter.crdate.year, inter.crdate.month,
                        inter.crdate.day, inter.crtime.hour,
                        inter.crtime.minute,
                        inter.crtime.second,
                        inter.ida, nominter);
                        fscanf (fich1, "%d %d %d %d %d %d %d
%s",
                        &inter.crdate.year,
                        &inter.crdate.month,
                        &inter.crdate.day, &inter.crtime.hour,
                        &inter.crtime.minute,
                        &inter.crtime.second,
                        &inter.ida, nominter);
                }
                else
                {       sprintf (nom, "%s", nominter);
                        fscanf (fich1, "%d %d %d %d %d %d %d
%s",
                        &inter.crdate.year,
                        &inter.crdate.month,
                        &inter.crdate.day, &inter.crtime.hour,
                        &inter.crtime.minute,
                        &inter.crtime.second,
                        &inter.ida, nominter);
                };
        };
        fclose (fich1);
        fclose (fich2);
        system ("cp temporai.idx document.idx");
        system ("rm temporai.idx");
}


/******* voir spécifications dans le module document *******/

compiddo (ident1, ident2)
struct idd ident1;
struct idd ident2;
{       int trouve;

        if (ident1.crtime.hour == ident2.crtime.hour)
        if (ident1.crtime.minute == ident2.crtime.minute)
        if (ident1.crtime.second == ident2.crtime.second)
        if (ident1.crdate.year == ident2.crdate.year)
        if (ident1.crdate.month == ident2.crdate.month)
        if (ident1.crdate.day == ident2.crdate.day)
        if (ident1.ida == ident2.ida) return (0);
        else return (1);
}
```

Annexe B : le dialogue entre serveurs du système.

I. Pour permettre la communication entre plusieurs serveurs situés sur des réseaux distincts, il faut disposer de plusieurs choses : de messages typés, de modules permettant l'envoi et la réception de ces messages, de modules permettant le traitement des demandes contenues dans les messages. Il nous faudra donc à la fois, d'une part, des primitives de communication et des activités prédéfinies sur chaque site, et d'autre part des veilleurs capables d'utiliser les primitives de communication et des serveurs capables de répondre aux requêtes qui arrivent. [Cor 81]

Notre système comporte la plupart de ces éléments à leur forme initiale : tout d'abord, chaque réseau local dispose d'un serveur capable de gérer les communications (uucp, uux, fichier compteur,...) et le traitement des requêtes (bd locale,...). Ensuite, ce même serveur dispose de toutes les primitives de communication nécessaires (uucp, uux) et des activités prédéfinies sont situées sur chaque site (activités de réception de messages).

Par ailleurs, il y a aussi le mode de dialogue existant entre les serveurs, c'est-à-dire la séquence dans laquelle les processus s'enchaînent. Nous allons voir quelles sont les possibilités d'agencement des processus, à l'envoi et à la réception des messages.

Prenons tout d'abord le côté de l'émetteur : le processus émetteur peut envoyer lui-même le message ou créer un processus fils qui s'en chargera. S'il envoie lui-même le message, il pourra, après l'envoi, soit continuer (ou s'arrêter) soit attendre. La continuation est bien entendu plus plaisante car elle permet le parallélisme et donc moins de perte de temps s'il faut réaliser des échanges en série (ceux-ci étant réalisés quasi simultanément). S'il confie à un processus fils la tâche de l'émission du message, le parallélisme est aussi

très facilement applicable, dans la mesure où la création du fils n'entraîne pas le blocage du père (pour que le fils ait terminé).

Au niveau de la réception des messages, on peut disposer de deux choses : de veilleurs et de serveurs. Les veilleurs ont pour rôle de guetter l'arrivée de messages, qu'ils identifient pour les confier aux serveurs compétents. Les veilleurs peuvent tourner continuellement en attente de l'arrivée d'un message ou être réveillés par l'arrivée d'un message. Les serveurs ont pour rôle de traiter les requêtes qui leur sont confiées. Par ailleurs, les serveurs peuvent être lancés directement par un poste éloigné, les messages consistant dès lors en une commande à faire exécuter à distance, et les veilleurs servant uniquement à permettre l'exécution de la commande transmise.

Pour notre part, c'est cette implémentation que nous avons choisie, car elle nous permettait de disposer d'utilitaires existants, en nous évitant de reconstruire ce qui était déjà fait. Notre système dispose donc d'un serveur par réseau local, ce serveur étant capable de réaliser les fonctions du CE, de gérer les communications, d'exécuter les requêtes formulées par des serveurs éloignés, et ce grâce aux utilitaires uux et uucp, le premier permettant justement l'exécution d'une commande à distance et le second permettant la copie (le transfert) d'un fichier, d'un serveur à (vers) un autre.

La façon dont les processus de notre programme dialoguent entre eux, par l'usage de uux et uucp, est la suivante : si un processus PA1 situé sur un site A désire consulter un document situé sur un site B, PA1 va envoyer une demande de consultation de document au site B en utilisant uux pour lancer, à distance, un processus PB1 dont la fonction est de traiter cette demande. Dès lors PB1 va devoir envoyer le document, en utilisant, pour ce faire, uucp, puis susciter la réception de ce document, en utilisant uux pour lancer, sur le site A, un processus PA2, dont la fonction est de gérer la réception du

document et sa consultation. Le même principe a été adopté pour les autres dialogues entre serveurs de réseaux différents.

II. Une autre question, dont nous devons parler à présent, est le mode de calcul des différentes informations, dites utiles, passées dans les paquets transmis par DCS.

Fixons tout d'abord les longueurs des différents objets manipulés dans le courrier électronique :

* un identifiant d'abonné, constitué d'un entier, ne compte que 2 bytes;
* de même un identifiant de réseau, est limité à 2 bytes;
* un identifiant de document, constitué d'une date (6 bytes), d'une heure (6 bytes) et d'un identifiant d'abonné, compte donc 14 bytes;
* un document est constitué arbitrairement de 1500 bytes : composant le corps (environ 1500 caractères).

Parmi les autres données nécessaires au dialogue, on trouve :

* les noms de fichiers, fixés à 10 caractères (bytes);
* les noms des procédures (employées par l'intermédiaire de uux) : (si idab est un identifiant d'abonné, idnw est un identifiant de réseau, nf_ est un nom de fichier et iddoc, un identifiant de document, alors ...)

  ** pour la (réception d'une) demande de consultation de boîte aux lettres : RECEPTDCONSBALDCS(idab,idnw), soit 20 caractères plus deux fois deux bytes; d'où 24 bytes.

  ** pour la réception de la boîte aux lettres : RECEPTBALDCS(idab,idnw,nf_bal), soit 16 caractères plus 14 bytes; d'où un total de 40 bytes;

  ** pour la (réception d'une) demande de consultation de document : RECEPTDCONSDOCDCS (idab,idnw,iddoc), soit 20 caractères plus 18 bytes; d'où un total de 38 bytes;

  ** pour la réception d'un document : RECEPTDOCDCS (iddoc,nf_doc), soit 15 caractères plus 24 bytes, pour un total de 39 bytes;

** pour la réception d'un nouveau document : RECEPTNEWDOCDCS(iddoc,idab,nf_doc,nf_dest), soit 20 caractères plus 34 caractères, pour un total de 54 bytes;

** pour la (réception d'une) demande de suppression de document : RECEPTDSUPDOCDCS(iddoc,idab), soit 19 caractères plus 4 bytes; d'où un total de 23 bytes.

* les envois réalisés avec uucp :

** l'envoi d'une boîte aux lettres : le contenu de la boîte aux lettres (5 identifiants de documents), soit 5*14 = 70 bytes, et le nom du fichier où se trouvera la boîte aux lettres, soit 10 bytes; un total donc de 80 bytes;

** l'envoi d'un document : un document étant plus long que la taille maximale acceptée (127 bytes), pour un paquet dans DCS, il faut répartir le contenu du document sur plusieurs paquets; dès lors, si, pour chaque paquet tranmis par uucp, on a 11 bytes "inutiles" (cfr infra), la longueur maximale de données utiles transmissibles sur un paquet est de 127-11 = 116 bytes. Or pour transmettre 1500 bytes (de corps plus 10 bytes pour le nom de fichier, plus 14 bytes pour l'identifiant, soit 1524 bytes),il faut donc envoyer 13 paquets de 127 bytes et 1 paquet de 27 bytes. Ce qui revient donc à facturer 27 segments.

** l'envoi d'une liste de destinataire doit comprendre l'envoi du nom du fichier où trouver cette liste (10 bytes) et le contenu de cette liste (5 identifiants d'abonnés = 10 bytes), ce qui donne une information de 20 bytes environ.

NB : les données considérées dans les boîtes aux lettres et listes de destinataires partielles sont quelque peu arbitraires mais un calcul plus fin (tenant compte du nombre de réseaux du système) aurait été superflu, vu leur taille relativement petite.

Il nous faut également parler des parties de données inutiles dans les paquets transférés sur DCS. Comme nous l'avons déjà mentionné ci-dessus, cette partie

inutile comporte 11 bytes; voyons à présent comment ces 11 bytes se répartissent.

Tout d'abord, il faut tenir compte des bytes relatifs au niveau transport, soit 2 bytes et 6 bytes exigés par uucp pour la constitution d'un header; puis, des bytes du niveau réseau, soit 3 bytes (2 pour le Numéro de Voie Logique et 1 pour les numéros de fenêtres); nous ne descendons pas plus bas puisque DCS facture les données envoyées au paquet : hors, les données du niveau 2 (niveau liaison de données) s'expriment en trames et non plus en paquets, ces derniers devenant les données utiles transmises dans les trames. Nous arrivons bien, dès lors, à (2+6+3 ou) 11 bytes d'information inutile par paquet transmissible via uucp ou uux.

III. Notons que parmi les hypothèses de calcul des coûts relatifs aux volumes transférés sur DCS, nous utilisons le coût de 0.2 francs pour 10 segments transmis; cela parce que nous considérons, dans la configuration du système, que chaque réseau local dispose de son propre PAD et donc que chaque serveur dispose d'un accès direct au réseau DCS !

IV. Signalons également que le mode de communication que nous avons choisi revient assez cher, notamment par l'utilisation d'ouvertures ponctuelles qui suscitent des dépenses supplémentaires en volumes de données échangées, par l'utilisation aussi de uucp et uux, pour les mêmes raisons, et par l'utilisation de DCS.

Annexe C: Manuel d'utilisation.

Le manuel d'utilisation correspond au programme présenté dans l'annexe A. Il comprend trois parties: le mode d'emploi du programme, les fichiers nécessaires au programme et l'utilitaire de création d'abonné et d'ajout de réseau local.

C.1. Mode d'emploi du programme.

Après s'être connecté au système via telnet et le réseau Ethernet, l'utilisateur lance le courrier électronique par CE, puis return.

Le programme demande l'identifiant d'abonné: c'est la procédure d'identfication. Remarquons que le seul contrôle implémenté ici est, lorsqu'on travaille avec la politique 3, de vérifier le bon respect de la localisation de l'abonné.

Ensuite, le menu principal est affiché. Il permet de choisir une des quatre fonctions du courrier électronique ou de stopper le programme:

- consulter la boîte aux lettres;
- consulter un document;
- créer/expédier un document;
- supprimer un document;
- quitter le courrier électronique.

Pour exécuter une fonction, il faut introduire le numéro correspondant au choix, puis return.

Consulter la boîte aux lettres: le programme construit la boîte aux lettres de l'abonné de la manière qui est déterminée par la politique. La boîte aux lettres est affichée jusqu'à ce que l'abonné tape return.

Consulter un document: le programme affiche la boîte aux lettres qui a été préalablement construite lors d'une consultation, et demande l'identifiant du document à consulter. Le programme détermine le lieu de stockage du document suivant la politique et l'achemine vers l'affichage. L'abonné peut ainsi le consulter avant de taper return pour revenir au menu.

Créer/expédier un document: un menu est affiché à l'abonné; il est conseillé de suivre les étapes dans l'ordre de présentation au menu:

- création d'un document;
- création d'une liste de destinataires;
- expédition du document;
- retour au menu principal.

La *création d'un document* permet d'éditer un nouveau document. On termine la création du document en introduisant un point comme premier et unique caractère de la dernière ligne.

La *création d'une liste de destinataires* demande à l'abonné d'introduire les identifiants des abonnés destinataires. On termine la crréation de la liste en introduisant un identifiant nul. L'identfiant de l'abonné créateur est automatiquement ajouté à la liste, de manière à ce que celui-ci puisse consulter par après tout document qu'il a expédié.

L'*expédition du document* ne peut être exécutée qu'après les deux premières étapes. L'expédition est réalisée par le programme conformément à la politique en cours.

Le *retour au menu précédent* a pour effet de perdre le document et la liste s'ils n'ont pas été expédiés.

Supprimer un document: la phase de détermination du document à supprimer est identique

à celle de la consultation. Ensuite, le programme se charge de la suppression et revient au menu principal.

Quitter le courrier électronique commande l'arrêt du programme.

C.2. Les fichiers nécessaires au programme.

Les fichiers qui sont décrits ici sont indispensables pour le bon fonctionnement du programme. Ils peuvent être créés, dans l'environnement Unix, avec un éditeur de texte.

Le fichier *dcsaddr.dat* contient l'adresse DCS du serveur local. L'adresse que l'on stocke dans ce fichier est un mnémonique qui doit être compris par les utilitaires de la famille UUCP (1).

Le fichier *dcsrefer.dat* contient l'adresse DCS du réseau local de référence. Ce fichier n'est nécessaire que lorsqu'on utilise la politique 5. L'adresse doit aussi être comprise par les utilitaires UUCP.

Le fichier *politique.dat* contient un numéro correspondant à la politique choisie. Les numéros possibles sont 1, 3, 4 ou 5. Ils correspondent aux politiques 1, 3, 4 ou 5. Ce seul fichier est, pour le programme, l'indicateur de la politique selon laquelle on travaille.

C.3. L'utilitaire de création d'abonné et d'ajout de réseau local.

Il existe un utilitaire qui permet de créer de nouveaux abonnés et d'ajouter de nouveaux réseaux locaux au courrier électronique. Cet utilitaire s'appelle "creerab.c". Il présente un

(1) consulter le manuel de Unix pour l'installation de UUCP.

menu principal permettant de choisir entre la création d'un abonné et l'ajout d'un réseau local.

La création d'un nouvel abonné se fait à partir d'un identifiant d'abonné non encore existant.

L'ajout d'un nouveau réseau se fait par l'introduction d'une nouvelle adresse DCS. Cette adresse est du même type que celle des fichiers dcsaddr.dat et dcsrefer.dat vus au point précédent.