MASTER EN SCIENCES INFORMATIQUES

Conception d'un logiciel C.A.O. de configuration d'un système informatique distribue

Gonze, Michel

Award date:
1986

Awarding institution:
Universite de Namur

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX , NAMUR

INSTITUT D'INFORMATIQUE

Année académique 1985-1986

CONCEPTION D'UN LOGICIEL C.A.O. DE CONFIGURATION D'UN SYSTEME INFORMATIQUE DISTRIBUE

*

Promoteur de mémoire :

Monsieur le Professeur

Ph. VAN BASTELAER

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

par Michel GONZE

Nous tenons à exprimer nos plus vifs remerciements à Monsieur le Professeur Ph. Van Bastelaer et aux assistants Mademoiselle B. Scoyer et Monsieur P. Geurts pour l'aide efficace et le soutien qu'ils nous ont apportés à la réalisation de ce travail, ainsi qu'à Madame S. Rucquoy pour la dactylographie.

P L A N

page

P L A N D E T A I L L E

page

page

page

A N N E X E S

I N T R O D U C T I O N

Depuis plusieurs années, l'informatique connaît une période de croissance très rapide, aussi bien dans les entreprises que chez les particuliers.

En parallèle au besoin premier de disposer de technologies de traitement de l'information, est apparue la nécessité d'assurer la communication entre divers systèmes informatiques hétérogènes. Certes, il existait déjà des réseaux de communication, mais limités à des cercles restreints. A l'heure actuelle, des efforts de standardisation des techniques de la télé-informatique ont permis d'envisager de développer des réseaux de taille élevée et pouvant accueillir une large gamme de systèmes informatiques.

Cependant, la forte croissance de la télé-informatique et l'importance des coûts d'utilisation ont conduit à se poser le problème du rapport coût/performance des réseaux. Un aspect économique se greffe à un problème purement technologique. Cet aspect a suscité de nombreuses études dans le domaine de la conception des systèmes répartis, domaine qui regroupe de très nombreuses spécialités dont le design d'un réseau, l'analyse de fiabilité, la communication entre des processus éloignés...

Parmi ces sujets, des recherches assez poussées ont été effectuées sur deux problèmes fondamentaux relatifs au design d'une configuration de réseau : l'allocation des fichiers aux noeuds du réseau et l'allocation de capacités aux lignes de transmission, sous certaines contraintes de performance.

La première optique envisagée en vue de résoudre ces deux problèmes a été de développer des algorithmes automatiques optimaux ou heuristiques. Cependant, comme le mentionnent Pichot et Detalle dans leur travail (PIC.84), dans la somme des travaux théoriques réalisés, peu se sont révélés aptes à

fournir des solutions sur des cas réels, avec de bonnes performances quant au temps calcul.

Face à cette faillite des techniques de recherche automatique, est apparue l'opportunité de développer une nouvelle méthodologie de résolution : la conception assistée par ordinateur (C.A.O.). Cette technique est déjà très appréciée dans certaines disciplines telle l'ingénierie. Son principal avantage est évidemment de permettre au concepteur d'être associé à la démarche de conception et de ne plus être dépendant d'un logiciel sur lequel il n'a aucune prise et qui peut ne pas être totalement adapté à ses besoins.

Le présent document contient la synthèse d'une étude sur la mise au point d'un logiciel de conception assistée par ordinateur d'une configuration de réseau et sur son implémentation dans un environnement déterminé. Après avoir précisé, dans le chapitre 1, le problème à résoudre et motivé notre méthodologie dans le chapitre 2, le troisième chapitre présente les spécifications des diverses fonctionnalités que doit recouvrir le logiciel. Ensuite, le chapitre 4 regroupe les éléments fondamentaux sur lesquels est basée l'implémentation issue de ces spécifications. Enfin, étant conscient que notre travail peut supporter quelques insuffisances, un ensemble d'améliorations et d'extensions potentielles est proposé dans le chapitre 5.

◊

CHAPITRE 1

POSITION DU PROBLEME

1.1. DESCRIPTION D'UN RESEAU INFORMATIQUE

1.2. DESCRIPTION DES CONTRAINTES IMPOSEES AU RESEAU

1.3. DEFINITION DU PROBLEME DE LA CONCEPTION D'UNE CONFIGURATION DE RESEAU

1.4. CONCLUSION

Ce chapitre a pour but d'énoncer de manière précise le problème de la conception d'une configuration de réseau.
Dans un premier temps, nous allons définir toutes les informations nécessaires à l'explicitation d'un modèle de description d'un réseau informatique, de ses paramètres d'utilisation et de ses contraintes.
Ensuite, nous préciserons les aspects du problème de configuration étudiés dans le cadre de ce mémoire.

1.1. DESCRIPTION D'UN RESEAU INFORMATIQUE

(PIC.84)

1.1.1. Les composantes d'un réseau

D'un point de vue général, les composantes d'un réseau informatique peuvent être regroupées en quatre catégories :

(i) la topologie du réseau qui consiste en des stations, des noeuds et des lignes les reliant;

(ii) les fichiers qui doivent être distribués;

(iii) les capacités de transmission allouables aux lignes du réseau;

(iv) la description du trafic d'informations à travers le réseau.

Classiquement, le coeur de la topologie d'un réseau est constitué des noeuds et des lignes de communication qui les relient. Son but est de mettre en relation des systèmes informatiques comme par exemple des ordinateurs ou des terminaux. De tels systèmes sont appelés stations.

Une topologie peut se schématiser comme à la figure 1.1. (TAN.81).

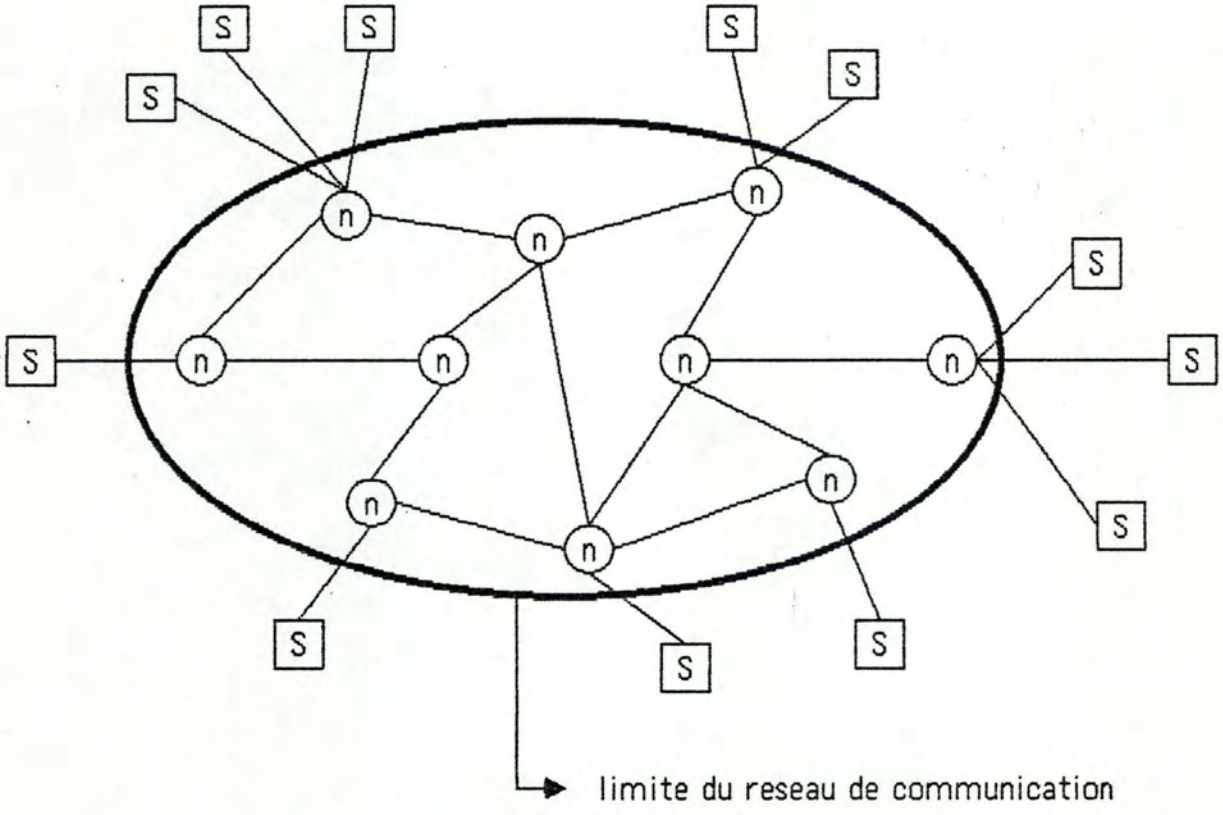


fig 1.1 Topologie d'un reseau informatique

Cette distinction entre les noeuds et les stations permet de différencier l'aspect "communication" de l'aspect "application".

Les trois autres composantes d'un réseau informatique sont plus liées à l'utilisation qui est faite de la topologie.

1.1.2. Description d'un noeud

Un noeud constitue un point d'entrée au réseau de communication proprement dit ou un intermédiaire sur le chemin reliant deux autres noeuds (cfr fig. 1.1.).

Du point de vue transfert d'informations, il représente une étape dans le cheminement des informations entre deux stations.

Dans notre approche, un noeud est décrit par les caractéristiques suivantes :

- un nom qui permet de l'identifier
- la valeur maximale du nombre de bits qu'il peut émettre par seconde (peut être infinie)
- la valeur maximale du nombre de bits qu'il peut percevoir par seconde (peut être infinie)
- la probabilité que la communication entre le noeud et le reste du réseau soit coupée.

Aucune considération technique n'est envisagée.

1.1.3. Description d'une station

Une station constitue un système informatique qu'un utilisateur désire connecter au réseau, via un noeud. Son but est d'effectuer des traitements sur des informations qu'elle possède ou qu'elle acquiert auprès d'autres stations via le réseau de communication.

La nature d'une station peut être diverse :

- un terminal
- un ordinateur
- un contrôleur relié à des terminaux
- un concentrateur relié à des terminaux et à des ordinateurs.

La figure 1.2. schématise quelques configurations potentielles d'une station reliée à un noeud.

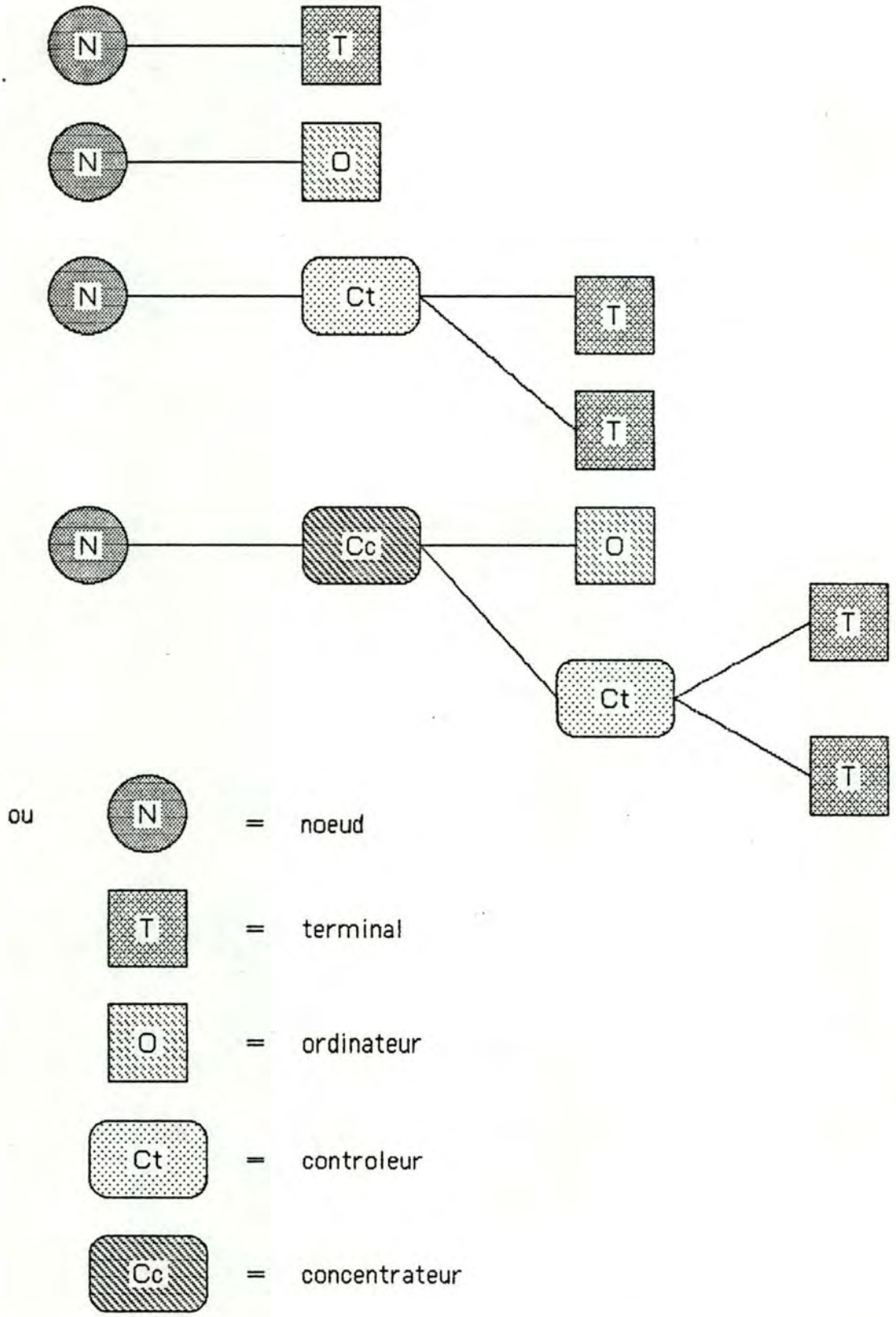


fig 1.2 Exemples de stations

Dans notre étude, nous avons élargi le concept de station en admettant que certaines stations jouent également le rôle de noeud, c'est-à-dire qu'elles sont intégrées au réseau de communication. Néanmoins, seules les stations ont une capacité de stockage des informations de l'utilisateur à l'inverse des noeuds stricts qui ne sont que des relais.

Dans notre approche, une station est décrite par les caractéristiques suivantes :

- un nom qui permet de l'identifier aussi bien parmi les stations que parmi les noeuds
- une capacité de stockage en nombre de bits (peut être infinie)
- un tarif mensuel par bit stocké.
- la valeur maximale du nombre de bits qu'elle peut émettre par seconde (peut être infinie)
- la valeur maximale du nombre de bits qu'elle peut recevoir par seconde (peut être infinie)
- la probabilité que la communication entre la station et le reste du réseau soit coupée.

Aucune considération technique, telle la configuration de la station, n'est envisagée.

1.1.4. Description d'une ligne

Une ligne constitue une voie de communication entre deux noeuds, deux stations ou un noeud et une station. Cependant, une telle voie de communication peut utiliser, plutôt qu'une ligne physique, un réseau de communication existant. Dans ce cas, nous pouvons considérer que deux noeuds, deux stations ou un noeud et une station, sont reliés par une ligne virtuelle que nous appelons ligne non louée. Par opposition une ligne physique est appelée ligne louée.

De plus, il existe également deux types de lignes, différentes par leur manière de transmettre l'information :

- les lignes unidirectionnelles
- les lignes bidirectionnelles simultanées.

Dans notre approche, toutes les lignes d'une topologie sont soit unidirectionnelles, soit bidirectionnelles simultanées.

Une description pertinente d'une ligne comporte les informations suivantes :

- le nom de la station (ou noeud) origine
- le nom de la station (ou noeud) extrémité
- la longueur de la ligne en kilomètres
- dans le cas où la ligne est louée, si son coût est fonction de sa longueur
- le tarif pour une minute de communication lorsque la ligne est non louée.

Aucune considération technique, telle la qualité de la ligne, n'est envisagée.

Il nous semble important de signaler que nous ne permettons pas que deux lignes différentes relient deux stations (ou noeuds), hormis dans le cas d'une ligne unidirectionnelle et de sa réciproque.

De plus, les lignes doivent être telles que la topologie du réseau peut se schématiser sous la forme d'un graphe connexe au niveau des stations, c'est-à-dire que toute paire de stations doivent pouvoir être mises en relation dans les deux sens.

1.1.5. Exemples de topologie

Notations : dans la suite nous appelerons

- NBS, le nombre de stations définies dans la topologie.

- NBN, le nombre de noeuds définis dans la topologie.
- NBL, le nombre de lignes unidirectionnelles ou le nombre de lignes bidirectionnelles multiplié par deux.

Exemple 1

NBS = 3 ; NBN = 4 ; NBL = 18
Toutes les lignes sont bidirectionnelles

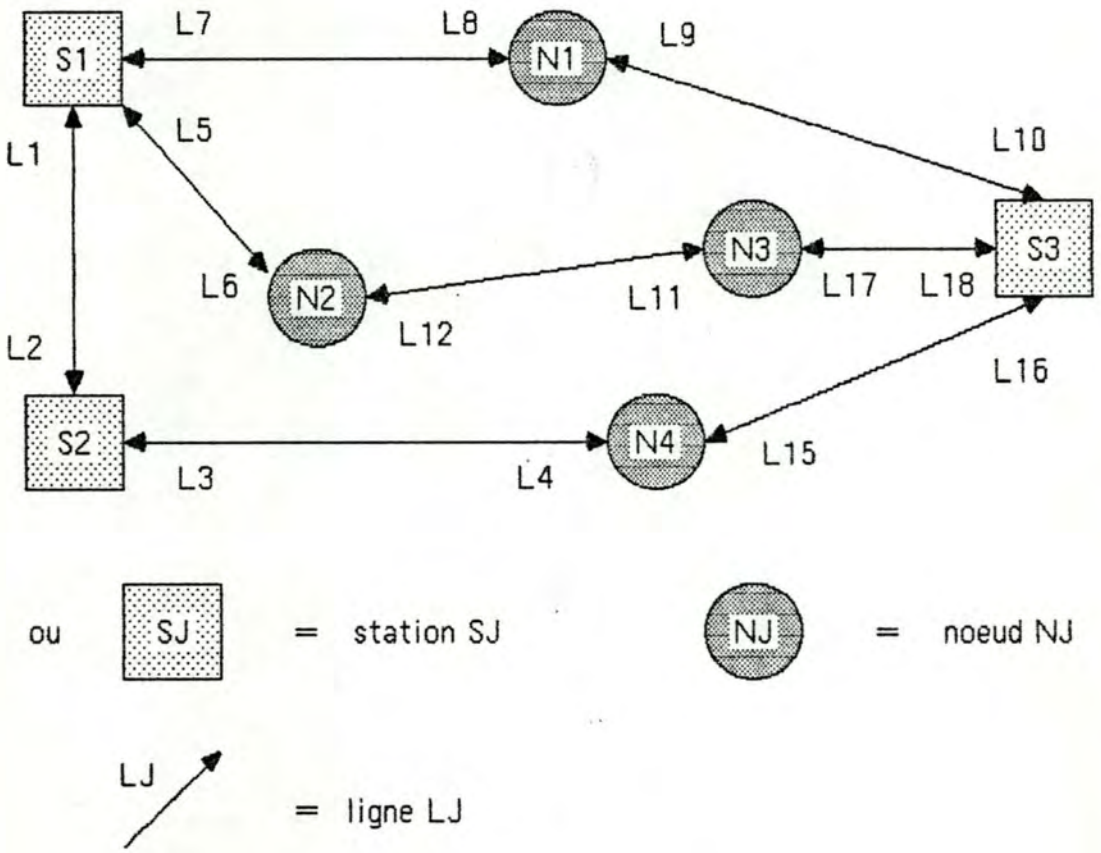


fig 1.3 Exemple de topologie n°1

Exemple 2 :

NBS = 4 ; NBN = 3 ; NBL = 12.
Toutes les lignes sont unidirectionnelles

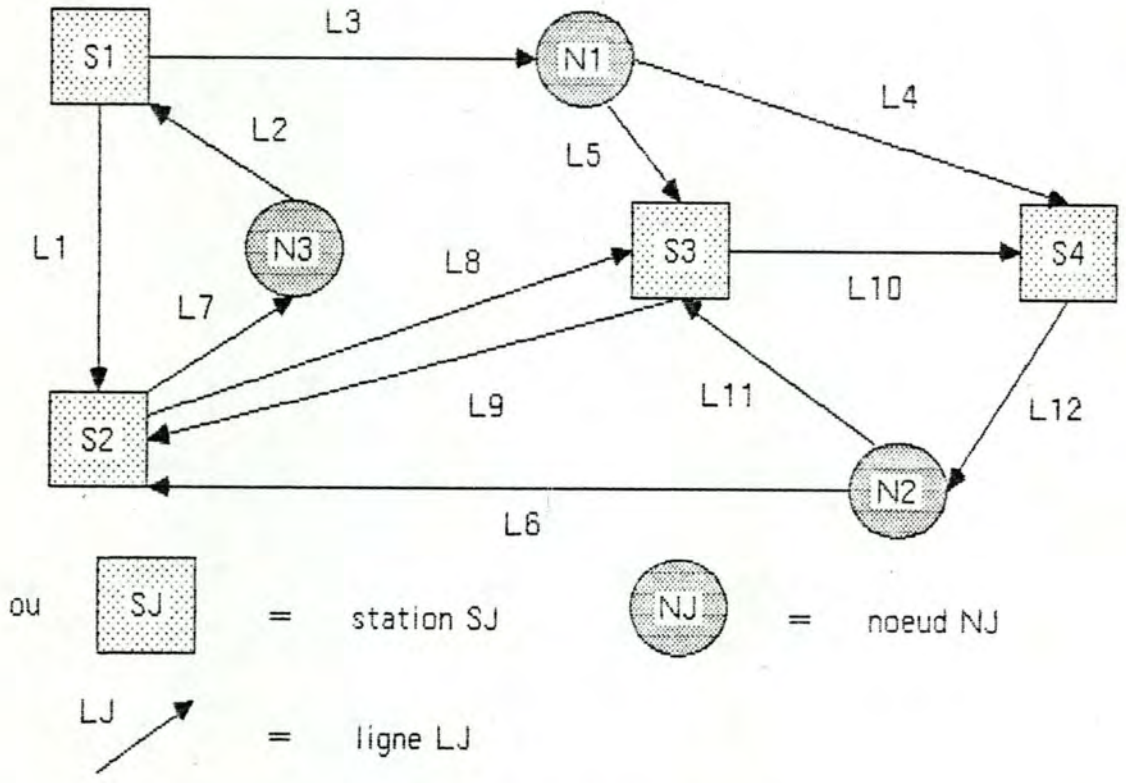


fig 1.4 Exemple de topologie n°2

1.1.6. Description d'un fichier

Nous considérons un fichier comme une collection d'informations au sens classique du terme.

Un fichier est présent dans le réseau sous la forme d'une ou plusieurs copies qui sont stockées dans des stations. Dans un souci de cohérence nous imposons les deux contraintes suivantes :

- un fichier doit au moins être présent en une station
- une station ne peut contenir qu'une seule copie d'un même fichier.

Dans notre approche, un fichier est décrit par les caractéristiques suivantes:

- le nom qui permet de l'identifier
- la longueur du fichier en bits
- la disponibilité souhaitée c'est-à-dire la probabilité souhaitée qu'un des exemplaires du fichier soit accessible lorsqu'un accès à ce fichier est demandé par une station quelconque.

  Cette notion de disponibilité apparaît vu que la fiabilité des lignes et des noeuds n'est pas parfaite.

Notation : Dans la suite, le nombre de fichiers présents dans le réseau est noté NBF.

1.1.7. Description d'une capacité de transmission

(VB.84)

Une capacité de transmission allouée à une ligne est la vitesse maximale de transmission permise sur la ligne.

De manière pratique, il existe une liste discrète de valeurs de capacité disponibles et chaque ligne d'un réseau se voit allouer une valeur de capacité qui est déterminée en fonction du flux d'informations transitant sur cette ligne.

Il existe plusieurs manières très différentes de réaliser une liaison entre deux équipements informatiques :

a) utilisation d'une ligne louée
b) utilisation d'une ligne non louée classique
c) utilisation d'une ligne louée avec commutation par paquets
d) utilisation d'une ligne non louée avec commutation par paquets.

Dans notre approche, nous nous limitons à ces quatre variétés.

La différence entre ces quatre types de liaison se marque, dans ce travail, par la manière de tarifier mensuellement leur utilisation.

1.1.7.1. Ligne louée classique

Le coût de l'allocation d'une valeur de capacité C à une ligne L est une fonction linéaire de la longueur de la ligne ou une constante dépendant de la ligne et de la valeur de capacité allouée, c'est-à-dire

tarif (L, C, "ligne louée") =

soit A (C) et B (C) * longueur (L)

où A (C) et B (C) sont des constantes qui dépendent de C.

soit E (L, C)

où E (L, C) est une constante qui dépend de L et de C.

1.1.7.2. Ligne non louée classique

Le coût de l'allocation d'une valeur de capacité C à une ligne L vaut une constante dépendant de C plus un coût proportionnel à la durée de la communication, c'est-à-dire

tarif (L, C, "ligne non louée classique") =

C (C)
+ prix à la minute de communication sur L
* ENTIER-SUP (temps d'utilisation de L)

où . C (C) est une constante qui dépend de C;

. ENTIER-SUP (x) désigne le plus petit entier supérieur au réel x;

. le prix de la minute de communication est exprimé en francs par minute;

. le temps d'utilisation de L est exprimé en minutes et vaut en première approximation :

durée d'un mois * (flux sur L/C)
avec la durée d'un mois en minutes et le flux sur L en bits par seconde.

1.1.7.3. Ligne louée avec commutation par paquets

Le coût de l'allocation d'une valeur de capacité C à une ligne L vaut une constante plus un coût proportionnel au flux d'informations sur L c'est-à-dire

tarif (L, C, "ligne louée avec commutation par paquets") =

C (C)
+ prix au segment transmis
* ENTIER-SUP[(flux sur L/longueur d'un segment)
* durée d'un mois]

où . C (C) est une constante qui dépend de C;

. ENTIER-SUP (x) désigne le plus petit entier supérieur au réel x;

. le prix par segment transmis est exprimé en francs par segment;

. le flux sur L est donné en bits par seconde;

. la longueur d'un segment est donnée en bits;

. la durée d'un mois est donnée en secondes. •

1.1.7.4. Ligne non louée avec commutation par paquets

Le coût de l'allocation d'une valeur de capacité C à une ligne L vaut une constante plus un coût proportionnel au flux d'informations sur L et à la durée de la communication, c'est-à-dire

tarif (L, C, "ligne non louée avec commutation par paquets") =

C (C)
+ prix au segment transmis
* ENTIER-SUP[(flux sur L/longueur d'un segment)
* durée d'un mois]
+ prix à la minute de communication sur L
* ENTIER-SUP (temps d'utilisation de L)

où . C (C) est une constante qui dépend de C;

. ENTIER-SUP (x) désigne le plus petit entier supérieur au réel x;

. le prix au segment transmis est exprimé en francs par segment;

. le flux sur L est donné en bits par seconde;

. la longueur d'un segment est donnée en bits;

. la durée d'un mois est donnée en secondes;

. le prix à la minute de communication est exprimé en francs par minute;

. le temps d'utilisation de L est exprimé en minutes et vaut en première approximation

durée d'un mois * (flux sur L/C)

avec la durée d'un mois exprimée en minutes.

Donc, nous disposons de quatre listes (éventuellement vides) de valeur de capacité disponibles.

1.1.7.5. Quelques observations

a) Dans le cas des lignes non louées classiques ou avec commutation par paquets, nous ne faisons pas intervenir le coût dû aux appels pour l'ouverture de la communication.
En effet, il est très difficile de définir une technique permettant d'en tenir compte car nous ne connaissons du flux d'informations que sa valeur moyenne par seconde. Rien n'est précisé quant aux variations de ce flux au cours d'une journée.
De ce fait, les coûts obtenus ne seront que des approximations des coûts réels. Cependant, dans l'optique d'une conception d'un réseau informatique, les répercussions de ces approximations sont mineures bien qu'une étude de ce problème pourrait s'avérer intéressante.

b) Dans le même ordre d'idée que pour la première observation, nous faisons une approximation pour le calcul du coût proportionnel au flux.
En effet, nous obtenons le nombre de segments transmis en divisant le flux total transmis par la longueur d'un segment. De ce fait, nous ne pouvons exprimer le fait que certains segments peuvent ne pas être remplis complètement et donc que le nombre de segments transmis peut être plus élevé.

Après avoir décrit les quatre listes de valeurs de capacité, il convient d'allouer à chaque ligne du réseau une valeur de capacité d'un des quatre types. Nous appelons cela le problème de l'allocation de capacités aux lignes et de détermination d'un principe de tarification. Ce problème sera étudié ultérieurement.

Notations.

Dans la suite, nous utiliserons les notations suivantes :

1) Une capacité allouée à une ligne louée classique est appelée capacité louée.

   NBCAPLOUE = nombre de capacités louées disponibles.

2) Une capacité allouée à une ligne non louée classique est appelée capacité non louée.

   NBCAPNLOUE = nombre de capacités non louées disponibles.

3) Une capacité allouée à une ligne louée avec commutation par paquets est appelée capacité DCS louée.

   NBDCSLOUE = nombre de capacités DCS louées.

4) Une capacité allouée à une ligne non louée avec commutation par paquets est appelée capacité DCS non louée.

   NBDCSNLOUE = nombre de capacités DCS non louées.

1.1.8. Description du trafic dans un réseau

1.1.8.1. Les types de trafic

Les stations s'échangent, à travers le réseau, de l'information codée que nous appelons messages.
L'accès aux fichiers par les différentes stations se traduit par un trafic de messages au travers du réseau, trafic dont nous pouvons distinguer quatre composantes :

(i) le "query traffic" = trafic résultant d'une demande de consultation de tel fichier par telle station.

Nous noterons u_{ij} le "query traffic" de la station i pour le fichier j et le μ-query l'inverse de la longueur moyenne d'un message de ce type.

Note : i et j sont des numéros attribués aux stations et fichiers dans un réseau fixé.

(ii) le "update traffic" = trafic résultant d'une demande de mise à jour de tous les exemplaires de tel fichier par telle station.
Nous noterons v_{ij} le "update traffic" de la station i pour le fichier j et μ-update l'inverse de la longueur moyenne d'un message de ce type.

(iii) le "query return traffic" = trafic de retour, vers la station origine de la demande, correspondant au "query traffic".

Nous noterons u'_{ij} le "query return traffic" de la station i pour le fichier j et μ-ret-query l'inverse de la longueur moyenne d'un message de ce type.

(iv) le "update return traffic" = trafic de retour, vers la station origine de la demande, correspondant au "update traffic".

Nous noterons v'_{ij} le "update return traffic" de la station i pour le fichier j et μ-ret-update l'inverse de la longueur moyenne d'un message de ce type.

Pour chaque paire (station, fichier), nous devons préciser la valeur de chacun de ces quatre types de trafic et cela en nombre de messages émis par seconde. Les valeurs à indiquer sont des valeurs moyennes et constantes.

1.1.8.2. Régime permanent et régime jour-nuit

Nous pouvons encore envisager le trafic sous un autre point de vue.
En effet, dans un certain nombre de cas, le trafic n'est pas uniforme tout au long d'une journée.
Dans notre approche, nous proposons deux optiques :

- le trafic se déroule en régime permanent :
  l'utilisation des fichiers est caractérisée par un seul jeu de valeurs de trafic, valable en permanence
- le trafic se déroule en régime jour-nuit :
  l'utilisation des fichiers est différente selon que nous sommes durant une période définie comme la journée ou celle définie comme la nuit. Elle se caractérise dans ce cas par deux jeux de valeurs de trafic.

  Par exemple, un utilisateur peut désirer que les mises à jour des fichiers n'aient lieu que la nuit.

De plus, tout le trafic est soit en régime permanent, soit en régime jour-nuit. Il n'est pas possible de différencier le type de régime pour chaque fichier.
Nous sommes obligés d'imposer cette contrainte car dans le cas contraire, la formule de calcul du flux à travers le réseau est probablement assez complexe.
Seule une étude très approfondie, que nous n'avons pas réalisée, permettrait de lever cette contrainte.

1.1.8.3. La procédure de routage

La communication entre deux stations s'effectue au travers du réseau selon un chemin non aléatoire. La règle définissant, pour chaque paire de stations, le chemin que doivent suivre les messages est appelée procédure de routage.

Les procédures de routage sont adaptatives ou déterministes selon qu'elles tiennent compte ou non des caractéristiques dynamiques du réseau, par exemple le flux sur les lignes et la défaillance de noeuds.
Dans notre approche, nous ne considérons que des procédures de routage déterministes.

Parmi les procédures déterministes, nous distinguons deux types différents :

- le routage entre deux stations, au travers du réseau, est déterminé selon la technique du chemin le plus court, en terme de kilomètres à parcourir (cfr. exemple 3).
- le routage entre deux stations, au travers du réseau, est déterminé par l'utilisateur du réseau qui le fixe au moyen d'une table de routage. (cfr. exemple 3).

Exemple 3

Soit un réseau composé de :

- trois stations S1, S2 et S3;
- deux noeuds N1 et N2;
- six lignes bidirectionnelles c'est-à-dire de douze voies simples L1, L2, L3, L4, L5, L6, L7, L8, L9, L10, L11, L12,

et schématisé par la figure 1.5.

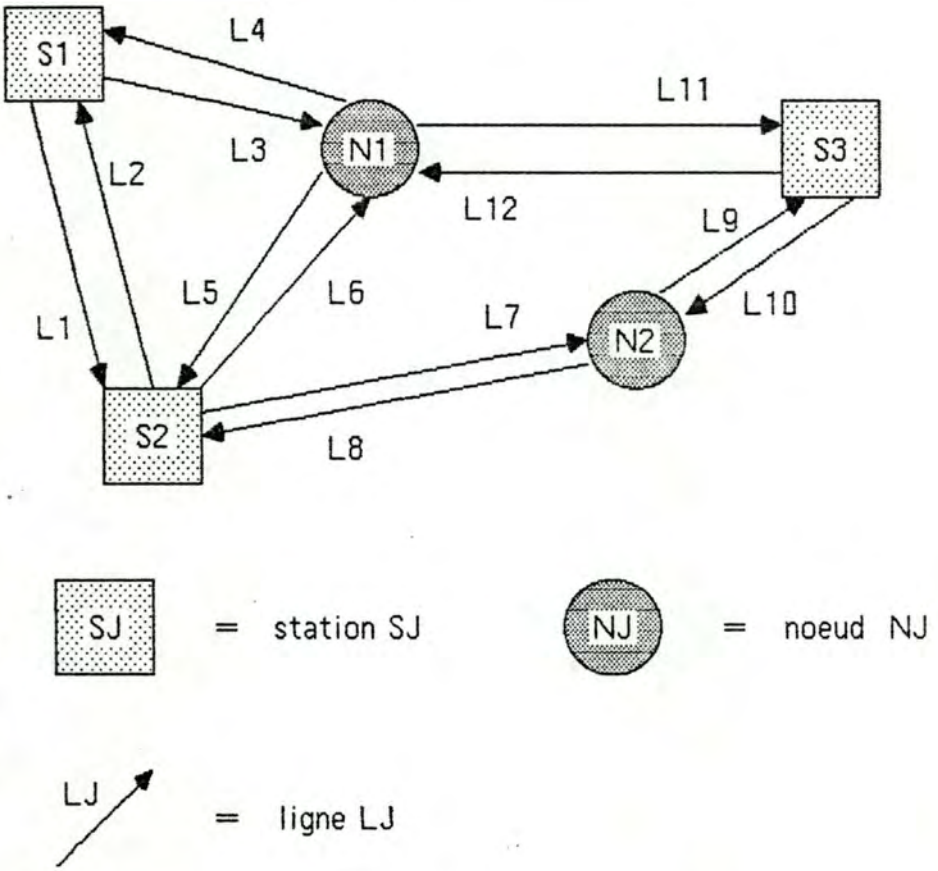


fig 1.5 Exemple de topologie n°3

Supposons que les lignes aient les longueurs suivantes :

- 100 km pour L1 et L2
- 45 km pour L3 et L4
- 55 km pour L5 et L6
- 70 km pour L7 et L8
- 40 km pour L9 et L10
- 120 km pour L11et L12.

La procédure de routage selon la technique du chemin le plus court nous donne les chemins suivants :

- de S1 vers S2 : L1 de longueur égale à 100 km
- de S1 vers S3 : L3 - L11 de longueur égale à 165 km
- de S2 vers S1 : L2 de longueur égale à 100 km
- de S2 vers S3 : L7 - L9 de longueur égale à 110 km
- de S3 vers S1 : L12 - L4 de longueur égale à 165 km
- de S3 vers S2 : L10 - L8 de longueur égale à 110 km

L'utilisation du réseau peut décider de déterminer une partie de la procédure de routage et il peut ainsi redéfinir les chemins suivants :

- de S3 vers S2 : L12 - L5 de longueur égale à 175 km
- de S2 vers S1 : L6 - L4 de longueur égale à 110 km
- les autres chemins restant identiques à ceux définis ci-dessus.

En fait, la deuxième technique de définition de la procédure de routage permet à l'utilisateur de tenir compte de certaines caractéristiques fixes du réseau, telle la définition du trafic, mais pas de la dynamique du réseau.

1.1.8.4. Le flux d'informations à travers le réseau

1) Grâce à la définition du trafic (1.1.8.1.) et de la procédure de routage (1.1.8.3.), nous sommes à même de calculer le flux d'informations circulant sur chaque ligne du réseau.

Nous pouvons envisager deux formules de calcul du flux, selon la manière d'envisager le "query traffic". En effet, pour ce dernier, nous pouvons utiliser l'une des deux techniques suivantes :

a) Si un exemplaire du fichier f est stocké à la station s, alors cette station utilise cet exemplaire.
Sinon, elle utilise indifféremment l'un des exemplaires de ce fichier et cela de manière uniforme.

b) La station s utilise l'exemplaire le plus proche.

Notation : le flux sur la ligne L est noté λ_L.

Définissons les deux formules de calcul du flux.

a) Formule du flux numéro 1

$$\lambda_L = \sum_{(i,k) \in S(L)} \gamma_{ik}$$

où . $S(L) = \{(i,k) \in (1,NBS) \times (1,NBS)$: le chemin, déterminé par le routage, joignant la station S_i à la station S_k passe par la ligne L $\}$

. $$\gamma_{ik} = \sum_{j=1}^{NBF} (u_{ij}/q_j) * (1 - x_{ij}) + v_{ij} * x_{kj} + (u'_{ij}/q_j) * (1 - x_{kj}) + v'_{kj} * x_{ij}$$

où . q_j désigne le nombre d'exemplaires du fichier j

. x_{ij} = 1 si le fichier j est stocké à la station i
= 0 sinon.

γ_{ik} désigne le nombre moyen de messages engendrés par seconde dont la station origine est S_i et la station destination est S_k.

b) Formule du flux numéro 2

$$\lambda_L = \sum_{(i,k) \in S(L)} \gamma_{ik}$$

où . $S(L) = \{(i,k) \in (1,NBS) \times (1,NBS)$: le chemin, déterminé par la table de routage, joignant la station S_i à la station S_k passe par la ligne L $\}$

$$\cdot\ \gamma_{ik} = \sum_{j=1}^{NBF} u_{ij} * z_{ijk}\,(1-x_{ij}) + v_{ij} * x_{kj} + u'_{ij} * z_{kji}\,(1-x_{hj}) + v'_{kj} * x_{ij}$$

où . z_{ijk} = 1 si la station S_k est, parmi les stations contenant un exemplaire du fichier j, la plus proche de la station S_i.
= 0 sinon.

2) Il est à noter que le flux est calculé en terme de messages transitant sur la ligne L par seconde. Il convient par conséquent de calculer la longueur moyenne d'un message sur la ligne L, à partir des longueurs moyennes définies pour chaque type de trafic.

Soit μ_L la longueur moyenne d'un message sur la ligne L.

Nous obtenons cette valeur de la relation suivante :

$$\lambda_L/\mu_L = (\lambda\text{- query})_L/\mu\text{-query} + (\lambda\text{-ret-query})_L/\mu\text{-ret-query} + (\lambda\text{-update})_L/\mu\text{-update} + (\lambda\text{-ret-update})_L/\mu\text{-ret-update.} \quad (1.7.)$$

où . $(\lambda\text{-query})_L$ = le flux sur la ligne L provenant du "query traffic", en nombre de messages par seconde

. $(\lambda\text{-ret-query})_L$ = le flux sur la ligne L provenant du "query return traffic".

. $(\lambda\text{-update})_L$ = le flux sur la ligne L provenant du "update traffic".

. $(\lambda\text{-ret-update})_L$ = le flux sur la ligne L provenant du "update return traffic".

1.2. DESCRIPTION DES CONTRAINTES IMPOSEES AU RESEAU

Dans le paragraphe 1.1., nous avons décrit tous les concepts nécessaires à la description complète d'un réseau informatique.

Il est évident qu'il n'est pas pensable de concevoir un réseau informatique de manière quelconque. En effet, une brève analyse nous conduit de suite à définir un certain nombre de contraintes qui ont soit une justification matérielle, soit une justification du point de vue de la qualité du service.

Dans notre approche, nous envisageons six types de contraintes :

- contraintes de capacité
- contrainte de délai
- contraintes de disponibilité
- contraintes de flux
- contraintes de stockage
- contraintes sur l'emplacement des fichiers.

1.2.1. Contraintes de capacité

Pour toute ligne L, la valeur de capacité qui est allouée à la ligne L, doit être strictement supérieure au flux sur cette ligne.

Le caractère strict de l'inégalité est lié à la formule utilisée pour le calcul du délai (cfr. 1.2.2.).

1.2.2. Contrainte de délai

Le délai de bout-en-bout d'un message est le temps total qu'il passe dans le réseau, entre le moment où il est émis par la station origine et où il est reçu par la station destination.

La contrainte de délai s'exprime de la manière suivante:

$$\overline{T} \leqslant T_{max}$$

où . T_{max} est la valeur fixée comme maximale pour le délai

. $\overline{T}$ est la valeur du délai moyen de transmission d'un message.

Pour le calcul du <u>délai moyen</u>, nous utilisons la formule proposée par Kleinrock (KLE.70) :

$$\overline{T} = \sum_{i=1}^{NBL} (\lambda_i / \gamma) * (1/(\mu_i * C_i - \lambda_i)) + \text{délai - réseau} * \delta_i$$

où . λ_i = le flux sur la ligne i, en nombre de messages par seconde (1.1.8.4.)

. γ = le flux total au travers du réseau

$$= \sum_{i=1}^{NBL} \lambda_i$$

. μ_i = l'inverse de la longueur moyenne d'un message sur la ligne i.

. C_i = la valeur de la capacité allouée à la ligne i.

. délai-réseau = le délai assuré par le réseau pour le transfert d'un message si la ligne i est non louée. Le délai est constant.

. δ_i = 1 si la ligne i est non louée
= 0 si la ligne i est louée.

Nous signalons que, si le régime jour-nuit est choisi (1.1.8.2.) alors la contrainte de délai n'est effective que le jour.

Par notre formule, nous définissons un délai moyen de transmission, comme nous avons envisagé en 1.1.8.1. un trafic moyen.
Nous faisons dans ce cas l'hypothèse que le phénomène est stationnaire et de moyenne parfaitement constante. Il nous est donc impossible d'envisager le problème des pointes de trafic et de leur évidente répercussion sur le délai réel instantané. Il convient donc de ne pas choisir un T_{max} trop petit qui serait trop fortement dépassé lors des pointes de trafic.

1.2.3. Contraintes de disponibilité

La disponibilité effective d'un fichier F est la probabilité effective qu'au moins un de ses exemplaires soit accessible lorsqu'un accès à ce fichier F est demandé par une station quelconque.

Pour son calcul, en plus de la probabilité, notée F_i, que la communication entre une station i et le reste du réseau soit coupée (1.1.2. et 1.1.3.), nous devons connaître la probabilité, notée r_{ik}, de succès d'une communication entre les stations i et k, utilisant le chemin, déterminé par le routage, reliant la station i à la station k.

La valeur de la disponibilité effective du fichier j s'obtient à partir de la formule suivante (PIC.84) :

$$D_j = \left(\sum_{i=1}^{NBS} w_{ij}\, a_{ij}\right) \Big/ \sum_{i=1}^{NBS} w_{ij}$$

où . $w_{ij} = u_{ij} + v_{ij} + u'_{ij} + v'_{ij}$

. $a_{ij} = \left(1 - \sum_{k=1}^{NBS} (1 - r_{ik} * x_{kj})\right) * (1 - F_i * (1 - x_{ij}))$

où x_{ij} = 1 si le fichier j est stocké à la station i
= 0 sinon.

1.2.4. Contraintes de flux

Comme nous l'avons déjà signalé (1.1.2. et 1.1.3.), un noeud ou une station a une capacité maximale de réception et d'émission d'informations. Les contraintes ci-dessous nous expriment ce fait.

1.2.4.1. Soit une station p avec $p \in (1,NBS)$

$$\sum_{L \in E_p} \lambda_L / \mu_L \leqslant$$ valeur maximale du nombre de bits que la station p peut émettre

où $E_p = \{$lignes L dont la station origine est p$\}$

$$\sum_{L \in F_p} \lambda_L / \mu_L \leqslant$$ valeur maximale du nombre de bits que la station p peut recevoir.

où $F_p = \{$lignes L dont la station destination est p$\}$

1.2.4.2. Soit un noeud p avec $p \in (1,NBN)$

$$\sum_{L \in E_p} \lambda_L / \mu_L \leqslant$$ valeur maximale du nombre de bits que le noeud p peut émettre

où $E_p = \{$lignes L dont le noeud origine est p$\}$

$$\sum_{L \in F_p} \lambda_L / \mu_L \leqslant$$ valeur maximale du nombre de bits que le noeud p peut recevoir.

où $F_p = \{$lignes L dont le noeud destination est p$\}$

1.2.5. Contraintes de stockage

Pour des raisons strictement matérielles, il peut arriver qu'une station ne puisse stocker autant de fichiers que ne le désirerait un utilisateur du réseau. A cet effet, nous avons défini dans la description d'une station (1.1.3.) la possibilité de limiter une fois pour toutes la capacité de stockage d'une station.

De ce fait, apparaissent des contraintes de stockage qui peuvent s'énoncer de la façon suivante :

pour toute station i avec i E (1,NBS)

$$\sum_{j=1}^{NBF} x_{ij} * \text{longueur du fichier } j \leqslant \text{capacité de stockage de la station } i$$

où x_{ij} = 1 si le fichier j est stocké à la station i
= 0 sinon.

1.2.6. Contraintes sur l'emplacement des fichiers

En plus des contraintes de stockage (1.2.5.), d'autres restrictions, plus seulement matérielles, peuvent être imposées à l'allocation des exemplaires d'un fichier aux stations du réseau.

En effet, un utilisateur peut souhaiter qu'un exemplaire d'un fichier donné soit stocké obligatoirement dans une ou plusieurs stations ou qu'au contraire, aucun exemplaire ne soit alloué à une ou plusieurs stations.

Ce type de contrainte peut survenir si des problèmes d'accès immédiat ou de sécurité , par exemple, se posent.

1.2.7. Notations

Par la suite, lorsque nous ferons référence à un de ces six types de contraintes, nous utiliserons les appellations suivantes :

a) nous dirons que le réseau est cfaisable si les contraintes de capacité sont respectées

b) nous dirons que le réseau est tfaisable si la contrainte de délai est respectée

c) nous dirons que le réseau est dfaisable si les contraintes de disponibilité sont respectées

d) nous dirons que le réseau est ffaisable si les contraintes de flux sont respectées

e) nous dirons que le réseau est sfaisable si les contraintes de stockage sont respectées

f) nous dirons que le réseau est efaisable si les contraintes sur les emplacements des fichiers sont respectées.

g) Le réseau est faisable s'il est cfaisable, tfaisable, dfaisable, ffaisable, sfaisable et efaisable.

1.3. DEFINITION DU PROBLEME DE LA CONCEPTION D'UNE CONFIGURATION DE RESEAU

Dans les paragraphes 1.2. et 1.3., nous avons développé un modèle de description d'un réseau informatique, de ses paramètres d'utilisation et de ses contraintes d'utilisation.

C'est dans ce cadre que nous allons étudier le problème de la conception d'une configuration de réseau.
Ce problème regroupe en son sein de nombreux aspects dont voici les principaux :

1) Design du réseau de communication
   - conception de la topologie du réseau
   - allocation des capacités aux lignes de transmission
   - allocation des fichiers aux stations.

2) Design d'une station
   - emplacement des concentrateurs
   - allocation des capacités aux lignes
   - étude de réseaux multipoints.

3) Contrôle de flux.

4) Contrôle du routage des informations.

Pour une approche globale du problème, nous vous suggérons de consulter le travail de synthèse réalisé par Pichot et Detalle (PIC.84).

L'extrême complexité de chacun de ces sous-problèmes nous a amené à nous limiter à étudier les deux aspects suivants :

a) l'allocation des fichiers aux stations

b) l'allocation des capacités aux lignes de transmission.

Notre choix a été guidé par leur importance dans le problème global, comme tend à le démontrer l'abondante littérature à leur sujet, et par l'opportunité d'y développer une nouvelle méthodologie de résolution.

Il est à signaler que ce choix nous a influencé lors du développement d'un modèle de description d'un réseau et que par conséquent, notre modèle peut se révéler incomplet pour l'étude d'un autre aspect du problème de conception d'une configuration de réseau.

•

Ce qui suit définit de manière précise l'objectif visé par les deux aspects sus-mentionnés.

1.3.1. Quelques définitions

a) Nous appelons allocation de fichiers, la localisation des fichiers à travers les stations du réseau.

Nous pouvons représenter une allocation de fichiers par une matrice X de dimension NBS * NBF telle que :

X_{ij} = 1 si un exemplaire du fichier j est stocké à la station i

= 0 sinon.

b) Nous appelons allocation de capacités, la liste des valeurs de capacité à allouer aux lignes de transmission et des principes de tarification à leur appliquer. (1.1.7.)

Nous pouvons représenter une allocation de capacités par un vecteur double (C, TYPE) de dimension NBL tel que

- la première composante égale la valeur de capacité à allouer
- la deuxième composante indique quel principe de tarification lui appliquer

1.3.2. Objectif

Soit un réseau informatique décrit selon le modèle explicité dans les paragraphes 1.2. et 1.3.

Notre objectif est de déterminer :

1) une allocation de fichiers X

2) une allocation de valeurs de capacité et de principes de tarification (C, TYPE),

qui minimise la fonction de coût :

$$d = \sum_{L=1}^{NBL} \text{tarif (L, TYPE (L), C (L))}$$

$$+ \sum_{i=1}^{NBS} \sum_{j=1}^{NBF} \text{(tarif au bit stocké pour la station i} * X_{ij} * \text{longueur du fichier j),}$$

sous les contraintes de capacité, de délai, de disponibilité, de stockage; de flux et d'emplacement sur les fichiers.

Nous essayons donc de résoudre les deux problèmes d'allocation simultanément et cela se répercute sur la fonction du coût qui comporte deux parties.

En fait, nous ne nous intéressons qu'à une partie des coûts de fonctionnement mensuels. Nous n'envisageons pas les coûts d'installation par exemple, bien que ceux-ci dépendent également des allocations à déterminer. Une autre définition de la fonction de coût nous amènerait probablement à étendre notre modèle d'un réseau informatique.

1.4. CONCLUSION

Dans ce chapitre, nous avons développé un modèle de description d'un réseau informatique et de ses paramètres d'utilisation.

Ce modèle a été choisi en fonction des aspects qu'il nous intéressait d'aborder dans le cadre de ce mémoire, c'est-à-dire l'allocation de fichiers aux stations et l'allocation de capacités aux lignes de transmission.

Notre problème étant posé de manière précise, nous allons, dans les chapitres suivants, tenter d'y apporter une solution qui soit à la fois raisonnable et de qualité.

◊

CHAPITRE 2

MOTIVATION

DE LA CONCEPTION ASSISTEE

PAR ORDINATEUR

Les deux problèmes qui nous occupent, c'est-à-dire l'allocation de fichiers aux stations et l'allocation de capacités aux lignes de transmission, ont été abondamment abordés dans la littérature.

Dans un premier temps, les recherches se sont axées sur la mise au point d'algorithmes garantissant une solution optimale, algorithmes basés sur des techniques de la programmation mathématique ou sur des recherches exhaustives.

Cependant, il est vite apparu que les algorithmes optimaux nécessitaient des temps calcul très importants lorsqu'ils étaient testés sur des données réelles.
Il a même été démontré que nos deux problèmes étaient NP - complets, c'est-à-dire qu'aucun algorithme optimal dont le temps calcul croît polynomialement avec la taille du problème, n'est réalisable.

L'étude simultanée des deux problèmes accroît encore leur complexité et les rend insoluble de manière optimale en des temps raisonnables, seuls de petits exemples académiques pouvant être traités.

Devant ce constat, une deuxième approche du problème fut envisagée : les méthodes heuristiques.

L'objectif des méthodes heuristiques est d'accélérer les traitements en réduisant l'espace de recherche grâce à des outils définis par la programmation mathématique et cela au prix de la perte de l'optimalité de la solution.

Bien que des critères permettant de guider la limitation de l'espace de recherche aient été développés, le coût calcul de ces heuristiques reste prohibitif sur des cas réels. Seules des configurations très particulières peuvent être traitées avec de bonnes performances.

Il est également apparu que l'ajout de contraintes au réseau, telles celles énoncées au paragraphe 1.2., accroît la complexité du problème et rend invalides certains critères permettant de limiter l'espace de recherche.

Malgré que de remarquables travaux théoriques sur la limitation de l'espace de recherche aient été réalisés, leurs implémentations sur ordinateur se sont révélées décevantes, lors de leur application sur des données réelles. En effet, le nombre de cas envisageables reste très important. Pourtant, le concepteur d'un réseau aurait probablement éliminé de lui-même un certain nombre de cas, de par sa connaissance du problème et de son expérience. Comme ces indices sont incommunicables à une technique analytique, il serait opportun d'associer le concepteur à la démarche de recherche en lui permettant d'établir de manière progressive, une solution acceptable.

Cette approche nécessite un logiciel lui permettant de gérer un modèle de son réseau et de proposer des solutions.

La tâche d'un tel logiciel consisterait à apporter une aide technique au concepteur. Cette aide peut prendre plusieurs formes telles la validation des solutions proposées en fonction des contraintes du problème; le calcul des coûts et performances et la guidance du concepteur dans sa démarche.

Il est à signaler que l'utilisation d'une telle méthodologie permet non seulement de traiter nos deux problèmes de manière simultanée, mais également d'ajouter certains degrés de liberté à la description d'un réseau, par exemple la modifiabilité de la topologie, chose irréalisable en cours de recherche dans les méthodes analytiques.

Dans la suite de notre travail, nous allons proposer une première ébauche de réalisation d'un tel logiciel, de même que des améliorations et extensions potentielles, mises en évidence par cette expérience.

◊

CHAPITRE 3

SPECIFICATIONS FONCTIONNELLES D'UN LOGICIEL DE CONCEPTION ASSISTEE PAR ORDINATEUR

3.1. INTRODUCTION DU RESEAU INITIAL ET DE SES CARACTERISTIQUES

3.2. RECHERCHE D'UNE SOLUTION

3.3. CONCLUSION

Dans les deux premiers chapitres, nous avons exposé le problème objet de ce mémoire, ainsi que la motivation de la technique préconisée en vue de le résoudre, c'est-à-dire la conception assistée par ordinateur.

Dans ce troisième chapitre, nous allons développer les fonctionnalités que devrait recouvrir un logiciel de conception assistée par ordinateur d'une configuration de réseau.

3.1. INTRODUCTION DU RESEAU INITIAL ET DE SES CARACTERISTIQUES

Tout travail de conception a pour objet la construction d'un système à partir d'un ensemble de "briques" de départ, d'éléments de base indispensables.
Dans le cadre de la conception d'une configuration de réseau, nous disposons des éléments décrits dans notre modèle exposé dans le chapitre 1.

L'analyse de notre modèle nous conduit à la définition des fonctions nécessaires à l'introduction d'une description d'un réseau, fonctions dont voici la liste :

- introduction des stations
- introduction des noeuds
- introduction des lignes
- introduction du routage entre les stations
- introduction du catalogue des capacités disponibles et calcul partiel du coût global
- introduction des paramètres d'utilisation du réseau
- introduction du trafic
- introduction de la fiabilité du réseau
- introduction des contraintes d'emplacement sur les fichiers
- sauvetage des informations introduites.

Dans l'ensemble, la technique d'introduction est similaire pour toutes ces fonctions. Pour cette raison, la logique de l'introduction ne sera complètement explicitée que dans le cas des stations. Pour les autres fonctions, nous ne spécifierons que les caractéristiques qui leur sont propres.

Il est à signaler que les spécifications énoncées dans la suite ignorent la possibilité de disposer d'outils graphiques. Une réflexion sur une telle extension sera présentée dans le chapitre 5.

3.1.1. Introduction des stations

La fonction "Introduction des stations" a pour but de saisir au terminal la description des stations du réseau.

Elle se déroule en trois phases :

1) Saisie au terminal

Le concepteur, c'est-à-dire l'utilisateur du logiciel, introduit au terminal les six informations décrivant chaque station (1.1.3.). Une station est identifiée par son nom. Donc, lors de l'introduction, un contrôle de non-existence antérieure est effectué. Ce caractère identifiant du nom implique une grande importance de l'orthographe.

La saisie s'effectue par écran complet et un signe conventionnel, '*' par exemple, permet d'en signaler la fin. Le stockage sur table est effectué à ce stade.

Il nous semble clair qu'un réseau cohérent doit au moins posséder deux stations car les transferts de messages ne peuvent avoir lieu qu'entre deux stations. Concrètement, le logiciel annulera toute introduction concernant un nombre de stations inférieur à 2. Il offrira au concepteur le choix de quitter le programme ou de recommencer l'introduction des stations.

2) Proposition de modification

Le logiciel demande au concepteur s'il désire visualiser son introduction et la modifier.

3) Visualisation des valeurs introduites et possibilité de modification

Le concepteur reçoit la possibilité de modifier les stations introduites lors de la première phase. Il ne lui est pas possible, à ce niveau, de supprimer ou d'ajouter des stations; ce type de manipulation est réalisable à un stade ultérieur (3.2.4.2.).

Le logiciel visualise les stations par groupe de cinq. Il propose alors trois options :

a) continuer la visualisation
b) quitter la troisième phase
c) modifier une station.

Dans l'option c), il suffit au concepteur de retaper la ligne qu'il désire modifier. Dans un souci de cohérence, le logiciel ne permet de modifier que les stations dont la description figure sur l'écran.
Le concepteur peut changer le nom d'une station mais tout en gardant son caractère identifiant, le logiciel intervenant si ce n'est pas le cas.
Après que le concepteur ait retapé la description de la station, le logiciel lui demande de confirmer la modification. Dans l'affirmative, le logiciel met à jour tables et écran et repropose les trois options, le concepteur ayant toujours l'état courant des tables devant les yeux.

3.1.2. Introduction des noeuds

La fonction "Introduction des noeuds" a une structure tout à fait analogue à celle de la fonction "Introduction des stations".

Un point doit être précisé : l'identifiant. Un noeud possède le même type d'identifiant qu'une station : le nom.

De plus, cet identifiant est commun aux deux concepts. Cette caractéristique signifie qu'une station et un noeud ne peuvent avoir le même nom et cela pour une question de bon sens : le concepteur doit pouvoir distinguer un noeud d'une station.

3.1.3. Introduction des lignes

La fonction "Introduction des lignes" a une structure analogue à la fonction "Introduction des stations" mais dédoublée car une ligne peut être unidirectionnelle ou bidirectionnelle.

Préalablement à l'introduction des lignes proprement dites, le concepteur doit spécifier au logiciel si les lignes de son réseau seront toutes unidirectionnelles ou bidirectionnelles et cela irrévocablement. Ce choix irrévocable est motivé par la complexité des opérations de mise à jour des tables que nécessiterait une modification du caractère unidirectionnel ou bidirectionnel des lignes.

Dans le cas des lignes, l'identifiant est constitué du couple (nom de la station ou noeud origine, nom de la station ou noeud destination). De ce fait, le logiciel doit effectuer trois contrôles lors de l'introduction :

a) contrôle de l'existence des stations ou noeuds origine et extrémité de la ligne
b) contrôle de la non-identité des stations ou noeuds origine et extrémité
c) contrôle de la non-existence antérieure de la ligne.

3.1.4. Introduction du routage entre les stations

La fonction "Introduction du routage entre les stations" permet de déterminer le routage du trafic à travers le réseau, c'est-à-dire le chemin suivi par les messages entre les stations.

Ce processus se déroule en deux étapes :

a) Dans un premier stade, le logiciel vérifie si le graphe qui schématise le réseau est connexe. Ce contrôle est obligatoire pour respecter notre modèle d'un réseau informatique (1.1.4.). En parallèle, le logiciel détermine un routage de "poids" minimal où le "poids" s'exprime en termes de distance. Une technique serait d'adopter l'algorithme d'Hamilton.

Si le graphe schématisant le réseau n'est pas connexe, le logiciel le signale au concepteur et lui propose de corriger son réseau. Si le concepteur le désire, le logiciel lui fournit la liste des inaccessibilités, c'est-à-dire les couples de stations qui ne sont reliés par aucun chemin et lui permet d'ajouter des lignes pour rendre connexe la topologie de son réseau.

Remarque : le concepteur ne disposant pas d'un écran graphique permettant de visualiser la topologie de son réseau, nous lui conseillons de dessiner sur une feuille de papier un graphe la schématisant.

b) Dans le cas où le graphe est connexe, même si le logiciel fournit un routage de longueur minimale, le concepteur peut souhaiter en fixer lui-même une partie, voire même la totalité. Il faut donc lui en offrir la possibilité.

La technique d'introduction est la suivante :

1) Le concepteur ne peut modifier à la fois qu'un chemin entre deux stations. Il doit donc, dans un premier temps, spécifier le nom des stations origine et destination de la connexion à déterminer.

2) Le logiciel propose alors au concepteur le routage existant. Il peut s'agir du routage de poids minimal ou d'un routage déjà fixé par le concepteur.

3) Le concepteur doit alors confirmer s'il désire toujours déterminer le routage de la connexion.

4) Dans l'affirmative, il lui est demandé d'introduire successivement le nom des stations ou noeuds intermédiaires en terminant par le nom de la station destination. Il est clair qu'une série de contrôles sont effectués :

   - existence des noeuds ou stations intermédiaires
   - pas de bouchage sur un noeud ou une station
   - pas de cycle dans le routage
   - existence des connexions entre deux noeuds ou stations successifs.

5) Dans le cas où les lignes sont bidirectionnelles, il est demandé au concepteur si le même routage doit être appliqué à la liaison inverse, c'est-à-dire de la destination vers l'origine sus-nommée.

Cette technique est applicable itérativement, tant que le concepteur le désire, et celui-ci peut redéfinir plusieurs fois un même routage.

Lorsque le concepteur est en train de déterminer un nouveau routage, il doit pouvoir annuler l'introduction en cours et dans ce cas, l'ancien routage demeure inchangé.

3.1.5. Introduction des fichiers à distribuer

Le module "Introduction des fichiers à distribuer" a une structure analogue à celle de la fonction "Introduction des stations".

L'identifiant d'un fichier est constitué de son nom.

3.1.6. Introduction du catalogue des capacités disponibles et calcul partiel du coût global

3.1.6.1. Catalogue des capacités disponibles

Le concepteur a la possibilité d'introduire un catalogue des capacités diponibles pour l'allocation de valeurs de capacité aux lignes du réseau en fonction du flux.

Dans notre modèle, nous avons envisagé quatre types de liaisons :

a) ligne louée classique (1.1.7.1.)
b) ligne non louée classique (1.1.7.2.)
c) ligne louée avec commutation par paquets (1.1.7.3.)
d) ligne non louée avec commutation par paquets (1.1.7.4.).

Pour chaque type de liaisons, le concepteur introduit les valeurs de capacité dont il dispose, de même que les valeurs des constantes présentes dans les fonctions de coût et qui dépendent des valeurs de capacité.

Exemple de catalogue

a) Ligne louée classique

valeur	constante A	constante B
1200	8000	450
2400	10000	500

b) Ligne non louée classique

valeur	constante C
300	5000
600	7500

c) Ligne louée avec commutation par paquets

valeur	constante C
600	5000
1200	7500
2400	10000
4800	12000

d) Ligne non louée avec commutation par paquets

valeur	constante C
300	4000
600	5000

L'identifiant d'une capacité est constitué du couple (valeur de la capacité, type de la ligne).

Pour chaque type de ligne, l'introduction des valeurs de capacité suit une logique analogue à celle de l'introduction des stations.

3.1.6.2. Calcul partiel du coût global

Lors de la détermination par le logiciel de l'allocation de capacité aux lignes de transmission, il est nécessaire de connaître pour chaque ligne le coût que représente, pour chaque valeur de capacité de chaque type, son allocation à la ligne.

Dès ce stade, nous pouvons déterminer ces coûts pour le cas des lignes louées classiques car leur principe de tarification ne fait pas intervenir la valeur du flux d'informations transitant sur la ligne (1.1.7.1.).

En effet, dans ce cas, le coût est

- soit une fonction linéaire de la longueur de la ligne et donc le logiciel est capable de calculer le coût pour chaque ligne et pour chaque valeur de capacité,
- soit une constante dépendant de la ligne et de la valeur de capacité, constante que le concepteur introduira à ce niveau.

Pour les trois autres types de lignes dont le coût d'une allocation de capacité dépend du flux d'informations, nous devons attendre de connaître ce flux sur chaque ligne. En effet, ce flux ne peut être connu à ce stade et, de plus, varie en fonction du routage de l'information et de la topologie du réseau.

3.1.7. Introduction des paramètres d'utilisation du réseau

Une fois la topologie de son réseau décrite, le concepteur doit préciser quelques caractéristiques de fonctionnement de ce réseau.

Ces caractéristiques sont :

- le type d'accès aux fichiers
- le délai maximal autorisé
- le type de régime
- le délai occasionné par le réseau
- la longueur d'un segment dans la commutation par paquets
- le prix pour l'envoi d'un segment dans la commutation par paquets.

3.1.7.1. L'accès aux fichiers

Un accès à un fichier pour consultation peut se réaliser de deux manières :

- tous les exemplaires d'un fichier ont la même probabilité d'être accédés;
- on accède à l'exemplaire du fichier le plus proche.

Le choix d'un de ces deux types d'accès se marque dans la formule de calcul du flux d'informations (1.1.8.4.).

Le concepteur doit préciser le type d'accès qui est d'application dans son réseau.

3.1.7.2. Délai maximal autorisé

Le concepteur doit préciser la valeur du délai maximal qui est d'application dans la contrainte de délai (1.2.2.).

3.1.7.3. Type de régime

Dans le paragraphe 1.1.8.2., nous avons présenté deux types de définition du trafic :

- le régime permanent
- le régime jour-nuit.

Le concepteur doit préciser le type de régime applicable pour son réseau.

3.1.7.4. Délai occasionné par le réseau

Dans le cas des lignes non louées, le réseau occasionne un délai pour le transfert.

Dans notre formule du calcul du délai moyen de transmission (1.2.2.), nous tenons compte de ce retard causé par le réseau lui-même. Le concepteur doit en préciser la valeur pour son réseau.

3.1.7.5. Longueur d'un segment

Dans la technique de la commutation par paquets, la tarification de l'utilisation d'une valeur de capacité fait intervenir le nombre de segments transmis sur une ligne (1.1.7.3. et 1.1.7.4.). Or, le flux sur la ligne est donné en bits par seconde. Il est donc nécessaire que le concepteur communique au logiciel la longueur en bits d'un segment.

3.1.7.6. Prix d'un segment

Dans la technique de la commutation par paquets, la tarification de l'utilisation d'une valeur de capacité nécessite de connaître le prix de la transmission d'un segment. Le concepteur doit donc en préciser la valeur.

3.1.8. Introduction du trafic

L'accès aux fichiers depuis une station se traduit par un trafic de messages au travers du réseau (1.1.7.1.).

La fonction "Introduction du trafic" permet au concepteur de quantifier, pour chaque couple (station, fichier) les quatre composantes du trafic :

- "query traffic"
- "update traffic"
- "query-return traffic"
- "update-return traffic"

ainsi que la longueur moyenne d'un message pour chacun de ces quatre types de trafic.

3.1.9. Introduction de la fiabilité du réseau

Dans le paragraphe 1.2.3., nous avons énoncé des contraintes de disponibilité. Dans leur expression, apparaissent les probabilités de succès d'une communication entre deux stations r_{ik} que nous appelons valeurs de fiabilité de la liaison.

Pour chaque paire de stations, le concepteur doit préciser la valeur de fiabilité de leur liaison.

3.1.10 Introduction des contraintes d'emplacement sur les fichiers

Comme énoncé au paragraphe 1.2.6., le concepteur peut désirer spécifier des contraintes sur l'emplacement des fichiers.

Plusieurs techniques d'introduction de ces contraintes sont envisagées :

- pour chaque station, le concepteur spécifie les fichiers objets d'une contrainte et le type de cette contrainte;
- pour chaque fichier, le concepteur spécifie les stations objets d'une contrainte et le type de cette contrainte;
- le concepteur spécifie lui-même les triplets (station objet de la contrainte, fichier objet de la contrainte, type de la contrainte) qu'il souhaite;
- pour tout couple (station, fichier), le concepteur spécifie si ce couple est l'objet d'une contrainte et si oui de quel type.

Après l'introduction, le concepteur peut visualier son introduction et la modifier

- soit en parcourant les stations et leurs contraintes
- soit en parcourant les fichiers et leurs contraintes
- soit en spécifiant une contrainte particulière en indiquant la station et le fichier objet de la contrainte.

3.1.11 Sauvetage des informations introduites

Dans les paragraphes précédents, nous venons de définir les fonctions d'introduction d'une description d'un réseau. Ces fonctions, activées les unes après les autres, peuvent

durer assez longtemps selon l'ampleur du réseau. De ce fait, le concepteur désirerait probablement envisager la recherche d'une solution lors d'une session ultérieure. Par conséquent, nous devons lui offrir la possibilité de sauver ses données sur un fichier.

En allant plus loin dans notre raisonnement, nous allons permettre au concepteur de suspendre l'introduction de son réseau en plusieurs endroits. Dans un premier temps, le concepteur pourra interrompre sa session à chaque terminaison d'une fonction d'introduction. Le logiciel doit alors lui assurer la sauvegarde des informations déjà introduites sur un fichier dont le concepteur aura préalablement déterminé le nom.

Lors du sauvetage sur fichier, le logiciel ajoute aux données un indicateur qui précise ce que contient le fichier.

Lors d'une reprise de session de travail, le concepteur spécifie au logiciel le nom du fichier où existe une description partielle d'un réseau. Grâce à l'indicateur, le logiciel sait déterminer ce qui a déjà été introduit et donc activer, si besoin, les fonctions nécessaires à la continuation de l'introduction du réseau.

Cette démarche de sauvetage impose un séquencement unique des fonctions d'introduction. Cependant, comme cette contrainte doit être de toute façon imposée à certaines fonctions, par exemple il faut introduire les stations avant les lignes, cet inconvénient nous paraît mineur.

Dans notre optique, le concepteur utilise toujours le même fichier tout au long de l'introduction complète du réseau. Ce n'est qu'ultérieurement (3.2.) qu'il peut utiliser plusieurs fichiers pour conserver différentes versions de son réseau.

3.1.12. Conclusion

Dans ce paragraphe 3.1., nous venons de spécifier toutes les fonctions nécessaires à l'introduction de la description d'un réseau informatique. Il nous reste à définir les fonctions qui nous permettent de rechercher une solution acceptable aux problèmes qui nous occupent et que nous avons posés au paragraphe 1.3.2., et cela dans le cadre d'une conception assistée par ordinateur.

3.2. RECHERCHE D'UNE SOLUTION

3.2.1. Stratégie globale

Une fois son réseau complètement décrit au moyen des fonctions décrites dans le paragraphe 3.1., le concepteur doit rechercher une solution aux problèmes de l'allocation de fichiers aux stations et de capacités aux lignes de transmission.

Dans l'optique de la conception assistée par ordinateur, il est naturel d'éliminer dans une large mesure les fonctions de calcul ayant recours à des algorithmes d'optimisation.

Dans notre cadre, le concepteur définira lui-même, avec sa connaissance du problème, l'allocation de fichiers.
En ce qui concerne l'allocation de capacités, il nous paraît préférable que ce soit le logiciel qui la détermine et ce pour deux raisons :

- l'allocation de capacités est limitée par des contraintes extérieures, à l'inverse de l'allocation de fichiers plus souple;
- il est possible de déterminer une allocation de capacités "pas trop mauvaise" avec des algorithmes relativement simples.

Au fil de la recherche, le concepteur peut s'apercevoir que la topologie de son réseau n'est pas suffisante ou que certains paramètres sont inadéquats. De cette constatation est née l'idée de permettre la modification de certaines caractéristiques du réseau.

Cependant, si nous permettons au concepteur de changer son réseau, il peut souhaiter conserver la version courante pour pouvoir éventuellement y revenir. De ce fait, nous avons décidé d'autoriser la possibilité de sauver le réseau courant tel que le décrivent ses différents paramètres, sur un fichier, que nous appelons fichier de données. Bien évidemment, il est nécessaire de prévoir la fonction inverse, c'est-à-dire la reconstitution d'un réseau dans l'état où il était défini, à partir d'un fichier de données.

En résumé, voici les fonctions que nous offrons au concepteur :

1) Sauvetage du réseau courant sur un fichier de données.
2) Reconstitution d'un réseau à partir d'un fichier de données.
3) Visualisation et modification du réseau courant.
4) Affichage de la solution courante et du coût du réseau courant.
5) Tests de faisabilité, détermination de l'allocation de capacités et calcul du coût du réseau courant.
6) Introduction d'une allocation de fichiers.

Ces six fonctions sont accessibles par le concepteur via un coordinateur qui les présente sous forme d'un menu.

Rappelons que, dans le cadre de notre travail, nous ne disposons pas d'outils graphiques. De ce fait, lorsque nous désirons parler des informations à afficher par le logiciel, il s'agit toujours de tableaux ou de messages décrivant la situation rencontrée. Une présentation des extensions graphiques sera proposée dans le chapitre 5.

3.2.2. Sauvetage du réseau courant sur un fichier de données

La fonction "sauvetage du réseau courant sur un fichier de données" permet au concepteur de sauver la version courante du réseau s'il lui semble que celle-ci représente une étape intéressante dans son processus de conception, notamment parce qu'il compte reprendre plus tard son raisonnement à partir de cette étape. Les informations sauvées sont de plusieurs types :

- la description du réseau courant, comme décrite dans le paragraphe 1.1.
- l'allocation de fichiers courante si elle a été définie.
- l'allocation de capacités si elle a pu être calculée et les coûts et performances en découlant.

Le concepteur a le loisir d'appeler cette fonction au niveau d'un coordinateur du module "Recherche d'une solution". De même, lorsqu'il désire terminer sa session de travail, il lui sera possible de faire appel une dernière fois à cette fonction pour sauvegarder l'état de son réseau.

Notre logiciel ne gère en aucune façon les fichiers de données créés par le concepteur. Celui-ci doit donc prévoir un répertoire des fichiers créés lors de ses différentes sessions de travail. Néanmoins, des contrôles seront effectués pour éviter au concepteur d'écraser des fichiers existants.

3.2.3. Reconstitution d'un réseau à partir d'un fichier de données

Si nous permettons la sauvegarde sur fichier de différentes versions du réseau, il est normal de donner au concepteur la possibilité de reconstituer une ancienne version en lieu et place du réseau courant.

Comme il a déjà été mentionné, le logiciel ne réalise aucune gestion des fichiers de données, tout au plus effectue-t-il un contrôle d'existence avant la lecture.

A priori, le logiciel ne sait pas ce qu'il va trouver dans un fichier de données. De ce fait, lors de la fonction de sauvetage, le logiciel va insérer un indicateur lui permettant, à la reconstitution, de savoir ce qu'il doit lire.

3.2.4. Visualisation et modification du réseau courant

La fonction "Visualisation et modification du réseau courant" constitue une partie importante du logiciel pour laquelle il y a lieu d'être prudent.

Si la visualisation du réseau ne pose aucun problème, il n'en va pas de même pour la modification. En effet, toute modification d'une donnée introduite initialement peut avoir des répercussions sur une partie des autres informations décrivant le réseau courant. Par conséquent, nous devons être attentif à rester cohérent dans tout ce que nous faisons et à bien déterminer toutes les implications d'une quelconque modification du réseau courant.

D'un point de vue pratique, vu leur étroite connexion, nous avons réuni la visualisation et la modification en une seule fonction. De plus, pour éviter des redondances, nous allons spécifier de manière complète la fonction relative aux stations. Pour les autres données, la structure est analogue et nous précisons, si nécessaire, les quelques différences qui surviennent.

3.2.4.1. Optique générale

Le concepteur peut modifier à sa guise le réseau, sans référence à ses essais antérieurs. La flexibilité maximale qui lui est ainsi accordée se paie : le logiciel ne garde pas de trace du raisonnement de la conception. Le concepteur doit gérer lui-même son cheminement en stockant des

versions-clés de son réseau sur des fichiers de données. En outre, cette flexibilité autorise une évolution permanente de l'énoncé même du problème au fil des essais. Par exemple, les contraintes peuvent être assouplies.

Il va de soi que le concepteur peut négliger les possibilités offertes s'il est tenu à une description de réseau fixe.

Il est bon de faire observer que toute modification du réseau nécessite un contrôle de cohérence sur l'ensemble des données, ce qui peut se traduire par d'importants temps de calcul.

3.2.4.2. Les stations

Sur chaque type de données, il est possible de réaliser trois opérations :

a) ajouter
b) modifier
c) supprimer.

a) Ajouter une station

Le concepteur reçoit la possibilité d'introduire la description d'une nouvelle station et de l'insérer dans le réseau courant.

Il est trivial qu'un contrôle de non-existence antérieure sera exécuté et que la fonction sera itérative pour permettre l'ajout de plusieurs stations.

Cependant, il est incohérent de laisser comme cela les stations : il faut les relier au reste du réseau. De ce fait, il doit être fait appel à une fonction permettant d'ajouter des lignes.

L'insertion d'une station dans le réseau consiste aussi à la faire apparaître dans toutes les tables où les stations apparaissent :

- table de description des stations
- tables décrivant les connexions entre les stations
- tables décrivant le routage entre les stations
- tables décrivant le trafic
- table spécifiant la fiabilité des communications entre les stations
- table spécifiant les contraintes d'emplacement des fichiers.

Il est à signaler que la présente fonction peut modifier de manière importante la topologie du réseau. De ce fait, le routage entre toutes les stations du réseau est susceptible d'être modifié. Cet aspect sera envisagé au paragraphe 3.2.4.5.

b) Modifier une station

Le concepteur a la possibilité de modifier la description d'une station. La technique de modification est analogue à celle décrite lors de l'exposé de la fonction "Introduction des stations" (3.1.1.).

Il convient de signaler que si le nom de la station est modifié, il faut aller mettre à jour la table de description des lignes car dans cette table, le nom de la station y est explicitement mentionné.

c) Supprimer une station

Le concepteur a la possibilité de supprimer une station du réseau. Cette fonction a de grandes répercussions sur le réseau. En effet, la suppression d'une station a pour effet de supprimer les lignes qui lui sont incidentes.

Les mêmes tables que lors de l'ajout d'une station, doivent être mises à jour. D'un point de vue pratique, la suppression d'une entrée implique leur compression. De plus, bon nombre de fonctions se servent du numéro d'ordre dans la table de description des stations pour repérer une

station particulière. Donc, une mise-à-jour très prudente s'impose.

Comme dans le cas de l'ajout d'une station, le routage entre les stations peut être affecté. Cet aspect sera étudié au paragraphe 3.2.4.5.

3.2.4.3. Les noeuds

Sur ce type de données, nous retrouvons les mêmes opérations que pour les stations. Cependant leur impact est moins grand et se limite essentiellement au routage entre les stations du réseau car le rôle d'un noeud se limite à permettre la communication entre les stations. L'impact sur le routage sera étudié au paragraphe 3.2.4.5.

3.2.4.4. Les lignes

L'ajout et la suppression d'une ligne influenceront essentiellement le routage des messages entre les stations (3.2.4.5.).

Nous apportons une restriction à la modification de la description d'une ligne : le concepteur ne peut changer le nom des stations ou noeuds extrémités de la ligne. En effet, nous assimilons ce type de modification à la création d'une nouvelle ligne. De ce fait, s'il désire apporter cette modification, le concepteur doit supprimer la ligne et puis l'ajouter avec les modifications voulues en faisant appel aux fonctions ad hoc.

Toute manipulation des lignes rend invalide la solution courante.

3.2.4.5. Le routage dans le réseau

Comme nous l'avons spécifié dans notre modèle de description d'un réseau, toute communication entre deux stations suit un chemin non aléatoire. Lors de

l'introduction de la topologie du réseau initial, le logiciel détermine un routage de poids minimal entre chaque paire de stations et le concepteur peut modifier une partie ou la totalité du routage (1.1.8.3.).

Dans certaines situations, le concepteur peut être amené à modifier le routage défini initialement. Trois types de situations peuvent être rencontrés :

a) modification volontaire

b) modification suite au viol d'une contrainte

c) modification suite à un changement de la topologie comme présenté dans les paragraphes 3.2.4.2., 3.2.4.3. et 3.2.4.4.

a) Modification volontaire

Le concepteur a la possibilité de modifier le routage défini précédemment entre n'importe quel couple de stations. Il est seul juge de l'opportunité de réaliser cette opération.

b) Modification suite au viol d'une contrainte

Dans le paragraphe 1.2., nous avons présenté les contraintes qui sont imposées à notre réseau. Comme il sera explicité au paragraphe 3.2.6., le viol de certaines de ces contraintes peut être éliminé par une modification du routage entre certaines stations.

Exemple : Considérons une partie d'un réseau (fig. 3.1.)

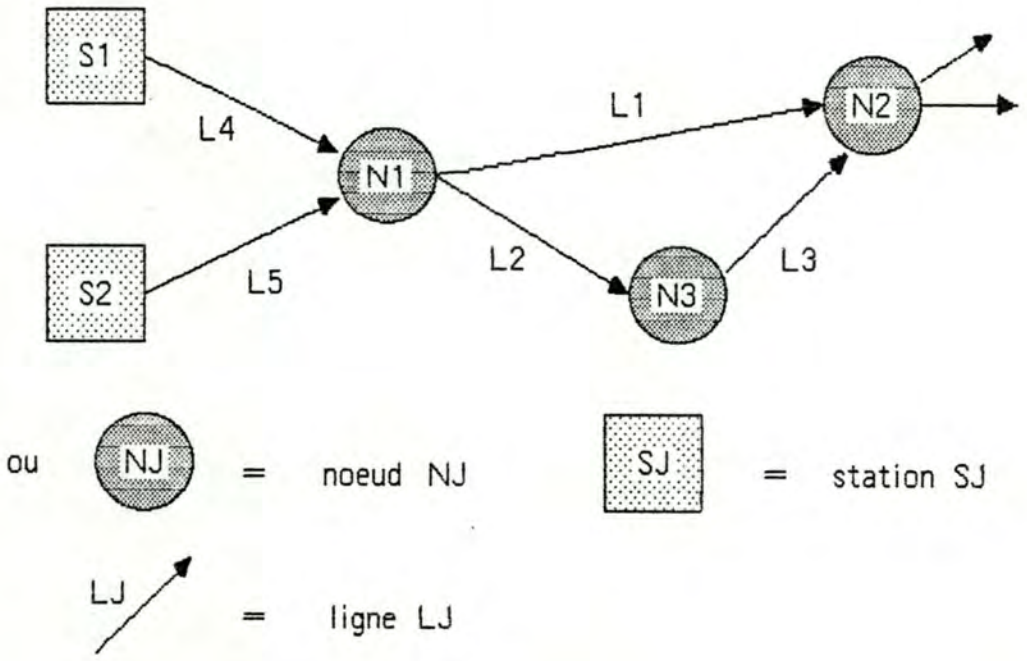


fig 3.1 Partie d'un reseau

Supposons que des stations S1 et S2 portent un flux d'informations qui transite par, respectivement L4 - L1 et L5 - L1.

Imaginons que la contrainte de capacité soit violée par la ligne L1 c'est-à-dire qu'il n'existe pas de valeur de capacité de transmission suffisante pour supporter le flux d'informations transitant sur L1.

Un remède à ce problème serait de modifier le routage du flux d'informations issu d'une des deux stations S1 ou S2.
Par exemple, le flux issu de S2 pourrait suivre le chemin L5 - L2 - L3 et ainsi éliminer le viol de la contrainte de capacité.

Pour résoudre des cas similaires à l'exemple ci-dessus, le concepteur doit avoir la possibilité de modifier une partie du routage. Cependant, comme la topologie du réseau n'est pas modifiée, le concepteur peut récupérer les anciens routages qui étaient définis précédemment et ne modifier que ceux qui lui sont nécessaires.

c) Modification suite à un changement de la topologie

Nous pouvons distinguer deux cas. Le premier consiste en l'ajout d'un noeud, d'une station ou d'une ligne. Dans ce cas, à la demande du concepteur, le logiciel recalcule un routage de poids minimal pour tirer parti de la nouvelle topologie. Ensuite, le concepteur a la possibilité de forcer une partie du routage soit en reprenant les anciens chemins forcés, soit en modifiant seulement quelques chemins.

Le second cas consiste en la suppression d'un noeud, d'une station ou d'une ligne. Dans ce cas, le logiciel peut avoir plusieurs réactions :

1) le logiciel recalcule un routage de poids minimal et propose au concepteur de redéfinir certains chemins, l'ancien routage étant ignoré;

2) le logiciel détecte tous les chemins utilisant les lignes supprimées et soit demande au concepteur les nouveaux chemins qui les remplacent, soit recalcule automatiquement le routage sur base du poids minimal;

3) le logiciel recalcule automatiquement le routage de poids minimal en tenant compte des modifications et demande au concepteur ce qu'il doit faire pour tout chemin modifié par ce calcul.

Il est à signaler que seul le cas a) est accessible par une demande explicite du concepteur. Les cas b) et c) sont envisagés seulement en cas de problème ou de modification du réseau courant.

Toute modification du routage rend invalide la solution courante.

3.2.4.6. Les fichiers

Les fichiers constituent le type d'informations dont la gestion est la plus simple. L'ajout ou la suppression d'un fichier se marque par une modification du flux sur les lignes de communications, flux qui doit être recalculé. Il faut également être attentif au problème des capacités de stockage des stations qui peut apparaître. Dans tous les cas, la solution courante est déclarée invalide.

Dans le cas de l'ajout d'un fichier, pour respecter notre modèle, une allocation de ce fichier aux stations du réseau doit être spécifiée par le concepteur. Au minimum un exemplaire du fichier ajouté doit être alloué. De même le trafic vers ce fichier depuis les stations doit être précisé par le concepteur.

3.2.4.7. Les capacités de transmission

Nous devons tout d'abord signaler que seul le catalogue des capacités disponibles peut être modifié et qu'il est impossible d'ajouter ou de modifier ou de supprimer un principe de tarification.

Il n'est possible de modifier que la valeur des constantes A, B ou C apparaissant dans le principe de tarification. Si le concepteur désire modifier une valeur de capacité, il doit supprimer l'ancienne valeur et insérer la nouvelle.

Dans le cas où le concepteur supprime ou modifie une capacité intervenant dans l'allocation de capacités calculée par le logiciel, l'allocation de capacité courante est déclarée invalide.

3.2.4.8. Le trafic

En ce qui concerne le trafic, la seule opération concevable est de modifier les valeurs du trafic. En effet, tout ajout ou suppression d'une station ou d'un fichier active de suite une fonction de mise à jour de la matrice du trafic. De ce fait, le concepteur visualise toujours une matrice de trafic cohérente.

Le concepteur peut désirer :

- redéfinir complètement le trafic
- visualiser le trafic courant et le modifier
- modifier ponctuellement le trafic; dans ce cas, le concepteur précise la station et le fichier concernés
- modifier les longueurs moyennes d'un message pour chaque type de trafic.

Toute modification du trafic rend invalide l'allocation de capacité calculée par le logiciel car le flux d'informations sur les lignes est déduit du trafic et la détermination de l'allocation de capacités est effectuée en fonction de ce flux.

3.2.4.9. Les paramètres

Comme dans le cas du trafic, le concepteur ne peut que modifier les paramètres du réseau. Toute modification d'un paramètre se réalise sous l'entière responsabilité du concepteur car une telle modification peut rendre invalide la solution courante.

Les paramètres "Type d'accès aux fichiers" et "Type de régime" sont tels que leur modification rend invalide la solution courante car ils influencent fortement le calcul du flux sur les lignes.

Le paramètre "Délai maximal autorisé" peut être diminué sans problème tant qu'il reste supérieur au délai moyen calculé par le logiciel c'est-à-dire tant que la contrainte de délai (1.2.2.) est respectée. Si ce n'est pas le cas, la solution courante est déclarée invalide.

Les paramètres "Longueur d'un segment" et "Prix d'un segment" n'invalident pas l'allocation de capacités courante mais seulement la valeur du coût global car ils interviennent dans les principes de tarification des capacités DCS (1.1.7.3.).

3.2.4.10. La fiabilité du réseau

La modification de la fiabilité des liaisons entre chaque couple de stations peut avoir pour conséquence de rendre invalide l'allocation de fichiers courante.

Dès qu'une valeur de fiabilité d'une liaison est modifiée vers le bas, le logiciel doit contrôler si la contrainte de disponibilité (1.2.3.) est toujours respectée. Si cette contrainte n'est plus vérifiée, la solution courante n'est plus valide.

3.2.4.11. Les contraintes d'emplacement

Toute modification des contraintes d'emplacement doit être répercutée sur l'allocation de fichiers. Si les nouvelles contraintes d'emplacement n'obligent pas le logiciel à modifier automatiquement l'allocation de fichiers, l'allocation courante reste valide ainsi que la solution courante. Dans le cas contraire, la solution courante est invalidée.

3.2.4.12. Remarque

Il est à signaler que le concepteur peut modifier le réseau aussi longtemps qu'il le désire avant d'activer les tests de faisabilité et le calcul de l'allocation de capacités et des coûts et performances.

Lors des modifications, le logiciel se contente de signaler au concepteur si la solution précédente reste valide ou pas. De plus, si le réseau a été déclaré une fois invalide, il le reste quelles que soient les modifications ultérieures. Seule l'activation des fonctions de calcul (3.2.6.) peut reconstruire une solution valide.

3.2.5. Affichage de la solution courante et du coût du réseau courant

Cette fonction permet au concepteur de visualiser la solution courante, déterminée par le logiciel en ce qui concerne l'allocation des capacités et des coûts et performances.

Elle peut être activée de deux manières :

a) après l'activation du module comportant les tests de faisabilité, la détermination de l'allocation de capacités et le calcul des coûts et performances (3.2.6.). Cela se fait de manière automatique.

b) à tout moment à partir du coordinateur de la recherche d'une solution.
Dans ce cas, si le réseau a été modifié de telle manière que la solution courante est déclarée invalide, le logiciel précise au concepteur que la solution courante n'est pas valable pour le réseau courant et visualise la solution si le concepteur le demande explicitement.

Les informations affichées consistent en :

- l'allocation de fichiers
- l'allocation de capacités :
  ligne - valeur de la capacité allouée - type
- le coût de la solution (1.3.2.)
- le délai moyen du réseau

3.2.6. Tests de faisabilité, détermination de l'allocation de capacités et calcul du coût du réseau courant

Comme nous l'avons déjà précisé (3.2.1.), alors que le concepteur détermine lui-même une allocation de fichiers, nous confions au logiciel le soin de déterminer une allocation de capacités. En plus de cette tâche, le logiciel doit effectuer les tests de faisabilité et le calcul des coûts et performances.

L'ensemble de ces tâches peut être ordonné par un coordinateur selon le processus suivant :

1) vérification des contraintes de stockage des stations (1.2.5.)
2) calcul de la disponibilité des fichiers et vérification des contraintes de disponibilité (1.2.3.)
3) calcul du flux sur les lignes du réseau (1.1.8.4.)
4) vérification des contraintes de flux (1.2.4.)
5) vérification des contraintes de capacité (1.2.1.)
6) vérification du caractère réalisable de la contrainte de délai
7) calcul d'une allocation de capacités de coût minimal
8) si l'allocation calculée en 7) ne vérifie pas la contrainte de délai (1.2.2.), calcul d'une allocation de capacité la respectant
9) calcul du coût du réseau (1.3.3.).

En cas de violation d'une contrainte, le processus est arrêté. Dans ce cas, le logiciel visualise au concepteur le problème rencontré. Il est bon de signaler que la vérification d'une contrainte n'est pas arrêtée dès qu'il y a violation. Par exemple, envisageons les contraintes de stockage. Le logiciel vérifie la contrainte pour toutes les stations, même si dès la première station un problème est apparu. Ceci permet de mettre en évidence tous les problèmes

attenants à la contrainte. Le concepteur peut alors essayer de tout corriger en une seule fois s'il le désire.

Si le processus est arrêté avant la phase 6), après exposition des problèmes rencontrés, le logiciel amène l'utilisateur au niveau du coordinateur du module "Recherche d'une solution".

3.2.6.1. Vérification des contraintes de stockage

Le but de ce contrôle est d'assurer qu'en chaque station, les fichiers qui y sont alloués n'excèdent pas la capacité de stockage.

Supposons que plusieurs stations violent cette contrainte. Le logiciel mémorise les noms de ces stations et les visualise au concepteur.

Dans une seconde phase, le concepteur peut demander au logiciel de lui préciser la nature des problèmes. Dans ce cas, le logiciel présente pour chaque station violant la contrainte les informations suivantes :

- le nom de la station
- la capacité de stockage de la station
- la différence entre la somme des longueurs des fichiers alloués à la station et la capacité de stockage
- la liste des fichiers alloués à la station ainsi que leur longueur, en précisant ceux qui sont soumis à une contrainte d'emplacement obligatoire.

Avec ces informations, le concepteur peut se rendre facilement compte du remède à apporter pour vérifier les contraintes de stockage, c'est-à-dire éliminer certains exemplaires de fichier.

3.2.6.2. Calcul de la disponibilité des fichiers et vérification des contraintes de disponibilité

Le but de cette fonction est d'effectuer une mesure de la disponibilité moyenne de chaque fichier. Le concepteur ayant défini pour chaque fichier une borne minimale, le logiciel doit déterminer les fichiers dont la disponibilité est inférieure à cette borne.

Le viol de la contrainte de disponibilité d'un fichier résulte probablement d'un nombre trop faible d'exemplaires de ce fichier. Dans ce cas, le logiciel doit visualiser l'allocation courante du fichier incriminé et proposer de l'étendre.

3.2.6.3. Vérification des contraintes de flux

Comme il est précisé dans notre modèle d'un réseau informatique, les stations et noeuds du réseau ont une capacité maximale de réception et d'émission du flux d'informations (1.1.1. et 1.1.2.).

Ayant préalablement calculé le flux d'informations circulant dans le réseau, le logiciel doit vérifier si ces contraintes sont respectées.

Supposons que le flux entrant à la station X soit supérieur à la capacité de réception de celle-ci. Une solution est de dévier une partie du flux par d'autres stations ou noeuds, ce qui nécessite une modification du routage. Ce point a été étudié au paragraphe 3.2.4.5.

Il nous semble clair que le logiciel ne peut fournir la modification à apporter au routage, mais il peut apporter des éléments d'informations pour guider le concepteur. Ces éléments sont :

- les lignes arrivant à la station X
- le flux sur ces lignes
- l'origine de ces flux, c'est-à-dire les liaisons utilisant ces lignes.

C'est alors au concepteur de tirer parti de ces informations et de découvrir le remède à apporter.

3.2.6.4. Vérification des contraintes de capacité

A un instant donné, le concepteur dispose d'un catalogue des valeurs de capacités. Mais il n'est pas sûr que ces valeurs de capacités soient suffisantes pour absorber le flux d'informations sur les lignes du réseau. Il convient donc que le logiciel s'en assure avant d'essayer de déterminer une allocation de capacité.

En cas de problème, le logiciel visualise les lignes où le flux est trop élevé pour les valeurs de capacité disponibles ainsi que les valeurs des flux transitant sur ces lignes.

Le concepteur peut alors réagir de deux manières :

- il ajoute une valeur de capacité suffisamment élevée pour supporter tous les flux sur les lignes
- il dérive une partie du flux transitant sur les lignes incriminées. Pour ce faire, il peut demander au logiciel de lui fournir des informations complémentaires sur la composition des flux c'est-à-dire sur les liaisons les composant.

3.2.6.5. Vérification du caractère réalisable de la contrainte de délai

Avant de calculer une allocation de capacité, le logiciel s'assure qu'il y a moyen de définir une allocation de capacité qui respecte la contrainte de délai.

Si cette contrainte ne peut en aucun cas être respectée, le logiciel peut déterminer les lignes où le problème se pose et le signaler au concepteur. Celui-ci peut alors réagir de manière analogue à celle présentée dans le cas de la violation des contraintes de capacité.

Cette démarche est rendue possible par le choix qui a été fait quant à la formule de calcul du délai moyen, c'est-à-dire la formule de Kleinrock (1.2.2.) où apparaît la contribution de chaque ligne au délai moyen.

3.2.6.6. Calcul d'une allocation de capacités.

Le logiciel se charge de déterminer automatiquement une allocation de capacités. Par le contrôle des contraintes de capacités (3.2.6.4.), nous sommes assurés que c'est réalisable.

Dans un premier temps, le logiciel détermine une allocation de coût minimal. Cependant, il se peut que cette allocation ne satisfasse pas la contrainte de délai. Or, par le contrôle (3.2.6.5.), nous sommes assurés que cette contrainte peut être vérifiée. Donc dans le cas où la contrainte de délai n'est pas respectée, le logiciel calcule une allocation de capacité vérifiant la contrainte de délai, mais sans s'occuper du problème du coût de cette allocation.

Les algorithmes nécessaires à ces calculs sont extraits du programme réalisé par Pichot et Detalle dans le cadre de l'étude de notre problème par une technique heuristique (PIC.84).

3.2.7. Introduction d'une allocation de fichiers

Le concepteur reçoit la possibilité d'introduire une proposition de solution au problème de l'allocation de fichiers.

Dans le cas où le concepteur définit pour la première fois une allocation et dans celui où il désire la redéfinir complètement, le logiciel lui propose deux techniques d'introduction :

- pour chaque station, le concepteur cite les fichiers qu'il désire y allouer
- pour chaque fichier, le concepteur cite les stations où doit être alloué un exemplaire de ce fichier.

Il est à signaler que le contrôle des contraintes d'emplacement est réalisé en cours d'introduction :

- préalablement à toute manipulation du concepteur, le logiciel alloue les exemplaires de fichiers obligatoires
- si le concepteur veut allouer un fichier interdit à une station, le logiciel lui en refusera l'introduction.

Si une allocation de fichiers existe déja, le concepteur peut la modifier ponctuellement. Pour cela, le logiciel lui visualise l'allocation courante soit par station, soit par fichier. Le concepteur peut alors ajouter ou retirer une allocation d'un exemplaire d'un fichier à une station, le logiciel contrôlant les contraintes d'emplacement.

Nous tenons à rappeler que dans notre modèle d'un réseau, au moins un exemplaire de chaque fichier doit être alloué dans le réseau.

3.3. CONCLUSION

Dans ce chapitre, nous avons présenté les spécifications fonctionnelles d'un logiciel de conception assistée par ordinateur d'une configuration de réseau. Il ne s'agit évidemment pas d'une démarche unique. En effet, le simple fait de ne pas disposer d'outils graphiques conditionne la manière d'envisager le dialogue entre le logiciel et le concepteur.

Dans cette étude, nous avons mis l'accent sur les tâches que doit réaliser le logiciel pour guider efficacement le concepteur dans sa démarche.

Dans la suite de ce travail, nous allons proposer une implémentation de ces fonctionnalités dans un environnement classique.

◊

CHAPITRE 4

ANALYSE ORGANIQUE

4.1. UNE ARCHITECTURE LOGIQUE

4.2. L'ENVIRONNEMENT LOGICIEL ET MATERIEL

4.3. LE DEVELOPPEMENT DES ALGORITHMES

4.4. CONCLUSION

Après avoir décrit les diverses fonctionnalités que doit recouvrir un logiciel de conception assistée par ordinateur d'une configuration de réseau, une analyse organique a été effectuée en vue de permettre leur implémentation dans un environnement réel. Le chapitre quatre constitue un dossier de programmation qui retrace la démarche adoptée. Après avoir exposé une architecture logique, une présentation des contraintes matérielles et logicielles rencontrées est réalisée, ainsi que des répercussions sur l'architecture physique. De plus, certains algorithmes un peu complexes sont explicités pour en permettre une compréhension aisée.

4.1. UNE ARCHITECTURE LOGIQUE

4.1.1. Principe

La décomposition en modules logiques est basée sur la différenciation des types de traitements à réaliser. Une simple lecture des diverses fonctionnalités du logiciel induit de manière naturelle une architecture logique simple, à laquelle vient se greffer un module reprenant différentes fonctions d'utilité générale.

Il est à signaler que la découpe proposée est libre de toutes contraintes matérielles. Son but est de permettre le développement des fonctions de manière indépendante, le seul lien entre ces différents modules se constituant de la description des structures de données(annexes).

4.1.2. Les modules logiques

Le principe de décomposition énoncé au 4.1.1. conduit aux modules décrits ci-après.

1) Module 1 : coordinateur du logiciel

Le module 1 a pour objectif de coordonner l'activation des différentes fonctions présentes dans les autres modules et cela selon un scénario déterminé.

2) Module 2 : définition d'un réseau

Le module 2 se compose des différentes fonctions réalisant l'introduction de la description d'un réseau (3.1.). Pour toutes les fonctions de calcul, le module "bloc numérique" est utilisé.

3) Module 3 : recherche d'une solution

Le module 3 regroupe les diverses fonctions réalisant la recherche d'une solution (3.2.). Pour toutes les fonctions de calcul, le module "bloc numérique" est utilisé.

4) Module 4 : stockage et reconstitution d'un réseau sur un fichier de données

Le module 4 a pour objectif d'accéder aux fichiers de données en vue de réaliser soit le stockage de la description d'un réseau sur un fichier de données, soit la reconstitution d'un réseau à partir d'un fichier de données.

5) Module 5 : bloc numérique

Le module 5 regroupe toutes les fonctions de calcul qui figurent dans le logiciel :

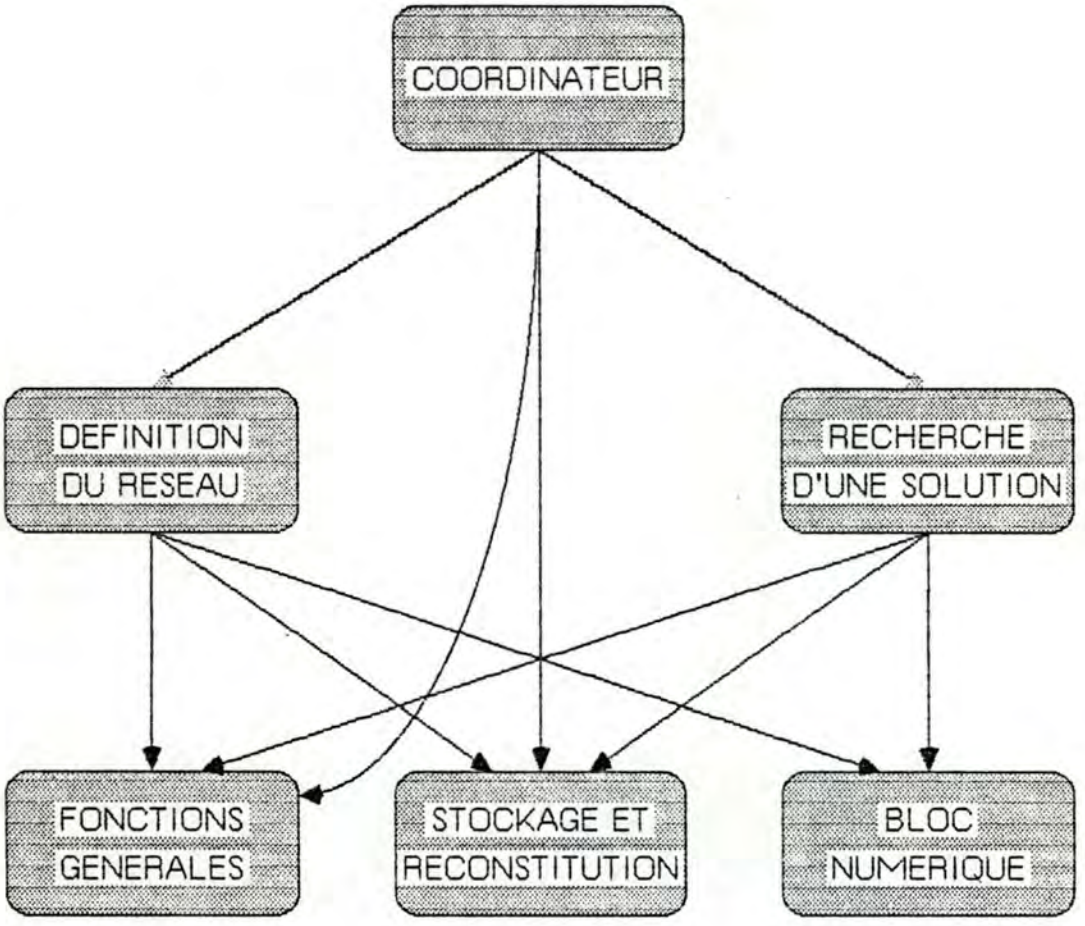
- vérification de la connexité de la topologie
- détermination d'un routage de poids minimal à travers le réseau
- contrôle de la faisabilité d'une solution : contraintes de stockage, contraintes de disponibilité, contraintes de flux, contraintes de capacité, contrainte de délai
- détermination de l'allocation de capacité
- calcul des coûts et performances.

6) Module 6 : fonctions d'utilité générale

Le module 6 regroupe une série de fonctions qui sont utilisées très fréquemment à travers le logiciel. Ces fonctions s'occupent principalement de :

- la gestion d'écran : lecture d'une chaîne de caractères, d'un entier, impression de messages bien définis, positionnement du curseur, effacement d'une partie de l'écran,...
- la vérification de l'existence d'une entité (station, ligne,...)
- la détermination du numéro d'ordre d'une entité.

La figure 4.1. représente l'architecture logique où les relations entre les modules, sont les relations "utilise" et "active".



ou ↓ = relation "utilise"

↓ = relation "active"

fig 4.1 Architecture logique

4.2. L'ENVIRONNEMENT LOGICIEL ET MATERIEL

4.2.1. L'environnement

Lors de l'implémentation des différents modules, nous avons travaillé dans un environnement matériel et logiciel qui nous a imposé certaines contraintes quant à la programmation et à la découpe physique du logiciel.

Le matériel utilisé est l'ordinateur PDP-11 de l'Institut d'Informatique des F.U.N.D.P. à Namur, doté de terminaux de type VT100 de Digital Equipment Corporation.
Quant à l'environnement logiciel, il est constitué du système d'exploitation UNIX version 7 (UNI.79) et du langage de programmation C avec les librairies standards STDIO.H, CTYPE.H et SGTTY.H. (KER.78)

L'utilisation de cet environnement nous a révélé un certain nombre de problèmes qui doivent être résolus avant d'entamer l'implémentation.

- limitation du code à 64 KB auxquels s'ajoutent 64 KB pour les données
- inexistence de fonctions standards de gestion d'écran dont principalement le positionnement du curseur à l'écran
- fonctionnement synchrone du terminal type VT100.

4.2.2. Limitation du code

4.2.2.1. Etude du problème

Cette contrainte quant à la taille du code, oblige à abandonner l'idée initiale de réaliser le logiciel sous la forme d'un ou deux gros programmes. Il est donc nécessaire de déterminer une découpe plus fine des fonctionnalités du logiciel pour les réaliser sous la forme d'un grand nombre de programmes disjoints.

Heureusement, le système UNIX offre les moyens pour réaliser la coordination de programmes distincts et cela au moyen des fonctions fork(), exec() et wait(), utilisables dans un programme écrit en C. La figure 4.2. représente une procédure écrite en C, permettant l'exécution d'un programme, en l'occurrence "station", à partir d'un autre. Chaque programme est alors appelé processus.

```
station()

{
int id,status;
extern char input[12];

id=fork();
if (id==0)
  execl("station","station","0",input,0);
wait(&status);
}
```

fig. 4.2. Coordination de processus.

La démarche est la suivante :

- le processus P1 exécute l'appel de la procédure station().
- l'instruction fork() crée un deuxième processus P2 identique à P1 et lui attribue le numéro 0.
- le processus P1 dont le numéro est différent de 0 va à l'instruction wait (status) où il attend un signal de P2 pour être débloqué.
- le processus P2 rencontre l'instruction execl(...) et il fait démarrer l'exécution du programme dont le nom est le premier paramètre de l'instruction. Ceci crée un sous-processus.
- à la fin de l'exécution du sous-processus déclenché par P2, le processus P2 est détruit et un signal de déblocage est envoyé à P1 qui reprend son cours normal.

Pour tout autre détail, il faut consulter (UNI.79).

Cependant, un problème subsiste encore : les données présentes en mémoire centrale dans un processus ne se communiquent pas à ses sous-processus et inversément. Le seul moyen de permettre à deux processus de communiquer est d'utiliser des fichiers. Dans notre cas, les processus s'échangeront un fichier de données où sont stockées toutes les données relatives au réseau et utilisées par un grand nombre de fonctions.

4.2.2.2. Application au logiciel

Il reste à définir une décomposition de notre logiciel en sous-processus. Encore une fois, ce sont les spécifications fonctionnelles du chapitre 3 qui vont déterminer la découpe.

Le processus principal est le coordinateur du logiciel. C'est lui qui contient les différents menus des fonctionnalités et qui active les sous-processus pour réaliser les diverses fonctions demandées par l'utilisateur.

A chacune des fonctions ci-dessous correspond un sous-processus :

- introduction des stations
- introduction des noeuds
- introduction des lignes
- introduction du routage entre les stations
- introduction des fichiers à distribuer
- introduction du catalogue des capacités disponibles
- introduction des paramètres d'utilisation du réseau
- introduction du trafic
- introduction de la fiabilité du réseau
- introduction des contraintes d'emplacement sur les fichiers
- sauvetage du réseau courant sur un fichier de données
- modification des stations

- modification des noeuds
- modification des lignes
- modification du routage entre les stations
- modification des fichiers à distribuer
- modification du catalogue des capacites disponibles
- modification des paramètres d'utilisation du réseau
- modification du trafic
- modification des contraintes d'emplacement
- affichage de la solution courante
- introduction d'une allocation de fichiers
- contrôle des contraintes de capacité
- contrôle de la contrainte de délai
- contrôle des contraintes de disponibilité
- contrôle des contraintes de flux
- contrôle des contraintes de stockage
- calcul de l'allocation des capacités et du coût de la solution.

Au début de chacun de ces sous-processus, une lecture des informations relatives à la description du réseau, est effectuée sur un fichier de données, fichier dont le nom est fourni par le processus coordinateur. A la fin du sous-processus, une écriture des mêmes informations, peut-être modifiées, ainsi que de nouvelles informations le cas échéant, est réalisée sur le même fichier. Ce fichier de données est le lien entre les divers sous-processus.

4.2.3. Gestion d'écran

Dans l'environnement auquel nous avons été confronté, il n'existe pas de librairie regroupant des fonctions de gestion d'écran. Certes, il n'est pas difficile de développer une version rudimentaire de certaines fonctions telles la lecture d'une chaîne de n caractères ou la lecture d'un nombre entier. Cependant, le plus grave est l'inexistence d'une fonction de positionnement du curseur à l'écran. Sans cela, il est exclu de proposer à l'utilisateur un

interface un tant soit peu agréable, chose inacceptable pour un logiciel de conception assistée par ordinateur.

Grâce à l'amabilité d'un assistant de l'Institut d'Informatique, une procédure de sa conception a pu être utilisée pour le positionnement du curseur, avec comme inconvénient qu'elle n'est valable que sur un terminal de type VT100. L'utilisation d'une telle procédure a donc pour conséquence de limiter la portabilité du logiciel. Cette limitation est acceptée car la procédure permet d'améliorer sensiblement la convivialité du logiciel. Dans le paragraphe 5.2.4., une solution sera proposée pour pallier à ce défaut.

Pour les autres fonctions de gestion d'écran, des versions rudimentaires sont mises au point et spécifiées dans l'annexe 1 sous la rubrique "Fichier TECH.C".

4.2.4. Fonctionnement du terminal VT100

Lors des tests de compréhension du langage C, il est apparu que le terminal de type VT100 fonctionne en mode synchrone, c'est-à-dire qu'il n'envoit les caractères introduits vers l'ordinateur que lorsque le caractère "RETURN" est introduit.

Comme il a été mentionné au paragraphe 4.2.3., en l'absence de fonctions standardisées, des procédures de lecture à l'écran ont dû être développées. Or, pour de telles fonctions, il est impératif de pouvoir contrôler un caractère dès qu'il a été introduit au clavier. D'où la nécessité de modifier le mode du terminal pour le rendre asynchrone.

A partir de la documentation sur le système UNIX (UNI.79) une procédure d'initialisation du terminal en mode asynchrone a été développée. La figure 4.3. représente le texte de cette procédure.

```
init_vt100()

{
extern struct sgttyb etat1,etat2;

gtty(0,&etat1);
etat2=etat1;
etat2.sg_flags |= CBREAK;
stty(0,&etat2);

}
```

fig. 4.3. Initialisation en mode asynchrone du terminal.

Cependant, ce changement de mode du terminal produit des effets de bord désagréables dont le principal est que le caractère DEL de recul d'une position du curseur à l'écran, n'a plus aucun effet. De ce fait, les procédures de lecture à l'écran doivent en tenir compte et gérer elles-mêmes ce caractère très important. Cette gestion consiste en les étapes suivantes :

- il faut détecter le caractèe DEL
- si au moins un caractère, autre que DEL, a déjà été introduit et non annulé par un DEL, il faut annuler le dernier caractère introduit et faire reculer le curseur d'une position à l'écran.

La figure 4.4. illustre un exemple de cette gestion, appliqué à une fonction de lecture d'une chaîne de caractères :

```
getline(s,lim)

char s[];
int lim;

{
  int c,i1;

    i1=0;
    while (i1<lim-1 && (c=getchar())!=EOF && c!='\n')
      { if (c== DEL && i1>0)
          { printf("\b \b");
            --i1;
          };
        if (c!=DEL)
          { s[i1]=c;
            ++i1;
          }
      }
  if (c=='\n')
    { s[i1]=c;
      ++i1;
    }
  s[i1]='\0';
  return(i1);
}
```

Fig. 4.4. Gestion du caractère DEL

De plus, cet effet de bord subsiste à la fin de l'exécution du logiciel. Il est donc nécessaire de rendre au terminal ses anciennes caractéristiques et donc son mode synchrone.
La procédure, illustrée à la figure 4.5., effectue cette fonction, pour peu que l'état initial du terminal ait été mémorisé (ici dans etat1). Pour plus de détails, il faut consulter (UNI.79).

```
retab_vt100()

{
extern struct sgttyb etat1;

stty(0,&etat1);

}
```

Fig. 4.5. Rétablissement du mode du terminal.

Il est à signaler que l'applicabilité des procédures illustrées aux figures 4.3. et 4.5. ne se limitent pas aux terminaux de type VT100.

4.2.5. Conclusion

Dans ce paragraphe 4.2., les contraintes dues à l'environnement logiciel et matériel ont été présentées ainsi que les conséquences qu'elles ont induites, conséquences dont la principale est la décomposition de l'architecture physique (4.2.2.). C'est dans ce contexte qu'ont été développés les algorithmes réalisant les fonctionnalités du logiciel, algorithmes qui vont être abordés dans le paragraphe 4.3.

4.3. LE DEVELOPPEMENT DES ALGORITHMES

Après avoir développé une architecture logique et jeté les bases d'une architecture physique, l'étape suivante de notre démarche consiste à concevoir les algorithmes de chaque fonction. Dans le paragraphe 4.3., nous nous limiterons à présenter les algorithmes de notre conception. En effet, les algorithmes des fonctions de calcul sont extraits d'un logiciel réalisé dans le cadre de leur mémoire, par Pichot et Detalle (PIC.84).
Pour ces fonctions, notre travail s'est limité à une traduction en langage C et à quelques modifications mineures.

4.3.1. Philosophie de la programmation

Dans le développement des modules de notre logiciel, un objectif est d'apporter une grande lisibilité et facilité de compréhension à nos programmes. Dans ce but, certains principes généraux sont sélectionnés :

a) Dans la mesure du possible, à une manipulation des données ou à un menu correspond une procédure.
Exemples :
. à la manipulation "Introduction de la description des stations" correspond une procédure
. menu général du coordinateur du module "Recherche d'une solution".

b) Lorsqu'une même action peut être effectuée sur plusieurs objets, des sous-procédures sont créées et cela dans un souci de clarté.
Exemple :
soit la fonction "modification des valeurs du trafic d'une station envers un fichier en régime jour-nuit". Comme huit valeurs différentes peuvent être modifiées dans la description du trafic, il existe une procédure de coordination des modifications et huit procédures de saisie des modifications.

c) Les textes et messages apparaissant à l'écran figurent dans le corps des procédures concernées, sous la forme d'instructions d'écriture explicites.
Même si cela alourdit un peu les procédures, une telle pratique peut aider à comprendre la logique d'un algorithme, en visualisant les scénarios possibles.

d) Chaque procédure et variable possède un nom qui est le plus explicite possible quant à sa fonction. Comme l'architecture physique est basée sur un système de processus disjoints, certaines procédures risquent de

porter le même nom.

e) Les procédures réalisant des fonctions similaires respectent la même logique.

Ces quelques principes de programmation étant admis, la majorité des procédures développées sont simples à comprendre à la lecture, pour peu que le lecteur connaisse le langage C ou à la rigueur le langage PASCAL. De plus, en annexe de ce travail, figurent des commentaires sur chacune des variables globales du logiciel, ainsi que les spéficifications des objectifs et paramètres de chacune des procédures implémentées.

Dans ce qui va suivre, nous allons présenter quelques aspects un peu plus complexes du logiciel, ainsi que quelques algorithmes de structure assez générale.

4.3.2. La notion de niveau d'introduction

Lors de l'introduction de la description d'un réseau, le concepteur a l'opportunité d'interrompre sa session de travail pour la reprendre ultérieurement (3.1.11.). Pour réaliser cette fonction, lors du sauvetage des informations sur un fichier de données, le logiciel ajoute aux données proprement dites, un indicateur qui précise le contenu du fichier, le niveau d'introduction atteint. Dans le logiciel, cet indicateur niveau est représenté par une variable entière. Ci-dessous, figure la table de codification : à chaque valeur de la variable correspondent les types des informations introduites.

niveau 0 : aucune donnée
niveau 1 : les stations du réseau
niveau 2 : les données du niveau 1
plus les noeuds du réseau
niveau 3 : les données du niveau 2
plus les lignes du réseau

niveau 4 : les données du niveau 3
plus le routage entre les stations
niveau 5 : les données du niveau 4
plus les fichiers à distribuer
niveau 6 : les données du niveau 5
plus les capacités disponibles
niveau 7 : les données du niveau 6
plus les paramètres d'utilisation
niveau 8 : les données du niveau 7
plus le trafic à travers le réseau
niveau 9 : les données du niveau 8
plus la fiabilité du réseau
niveau 10 : les données du niveau 9
plus les contraintes d'emplacement sur les fichiers
niveau 11 : les données du niveau 10
plus une allocation de fichiers.

Cette variable niveau devient indispensable dans notre logiciel, dans la mesure où il est composé d'un système de processus disjoints ne pouvant communiquer que via des fichiers (4.2.2.). En effet, cette variable est le seul moyen pour un sous-processus de communiquer au processus coordinateur s'il a bien rempli sa fonction. Par exemple, lors de la détermination du routage à travers le réseau, un processus vérifie le caractère connexe de la topologie. Si la topologie est connexe, ce processus gère la fonction d'introduction du routage et signale au processus coordinateur, via la variable niveau qui est mise à 4, que tout va bien. Si la topologie n'est pas connexe et si le concepteur ne veut pas la corriger, le processus se termine de suite et signale au processus coordinateur qu'il n'a pu réaliser sa fonction entièrement, et cela via la variable niveau qui vaut alors 3. C'est alors au processus coordinateur de réagir.

Cet aspect "communication entre processus" de la variable niveau a été étendu à d'autres fonctions du logiciel : la vérification des contraintes de faisabilité. En effet, à chaque vérification d'une contrainte correspond un sous-

processus et ces divers sous-processus sont activés dans un ordre bien déterminé par le processus coordinateur. En cas de violation d'une contrainte, le sous-processus correspondant le communique au coordinateur comme précédemment, via la variable niveau. Pour ces fonctions, la table de codification est la suivante :

niveau 12 : les contraintes de stockage sont vérifiées
niveau 13 : les contraintes de stockage, et de disponibilité sont vérifiées
niveau 14 : les contraintes de stockage, de disponibilité et de flux sont vérifiées
niveau 15 : les contraintes de stockage, de disponibilité, de flux et de capacité sont vérifiées
niveau 16 : les contraintes de stockage, de disponibilité, de flux, de capacité et de délai sont vérifiées
niveau 17 : toutes les contraintes de faisabilité sont vérifiées et une allocation de capacités est déterminée.

L'utilisation de cette variable niveau permet également d'éviter de recontrôler inutilement une contrainte. Par exemple, une modification du trafic à travers le réseau invalide toutes les contraintes de faisabilité sauf les contraintes de stockage. Dans ce cas, la variable niveau est portée à la valeur 12 et le coordinateur sait ainsi qu'il n'est pas nécessaire d'activer le processus relatif aux contraintes de stockage. Ceci permet d'alléger la charge en temps calcul.

4.3.3. Quelques algorithmes

Comme il a déjà été mentionné, la majorité des procédures développées sont simples à comprendre et ne nécessitent pas de détails complémentaires autres que les spécifications présentées en annexe de ce travail. En effet, elles consistent en des gestions d'écran et des manipulations élémentaires de données. Cependant, une présentation de la logique

de quelques algorithmes va être réalisée. La sélection de ces algorithmes est basée sur deux critères :

- soit leur logique est utilisée dans de nombreux contextes
- soit leur développement a demandé un peu de réflexion.

Dans la présentation, chaque algorithme est identifié par le nom de la procédure l'implémentant et le nom du fichier contenant cette procédure, la spécification de la procédure étant explicitée en annexe 1.

Note

Aucun algorithme de calcul n'est abordé ici. Pour une présentation de leur logique, il faut consulter (PIC.84).

a) Fichier MAITRE.C : introduction()

Cette procédure met en pratique l'aspect "communication entre processus" de la variable niveau. Son but est d'activer successivement les fonctions permettant l'introduction d'un réseau, avec possibilité d'interrompre la session de travail et de la reprendre ultérieurement.

Logique

Pas 1 : Lecture dans le fichier de données, du niveau atteint pour l'introduction et mémorisation de cette valeur dans la variable niveau. Si le fichier mentionné par l'utilisateur n'existe pas, la variable niveau est mise à zéro. Aller au pas 2.

Pas 2 : Soit n la valeur de la variable niveau. Activation de la fonction d'introduction permettant d'amener la variable niveau à la valeur n + 1. Aller au pas 3.

Pas 3 : Si la fonction peut ne pas se terminer correctement (niveau = 0,2,3,4 ou 5), contrôle que

la fonction s'est bien achevée (si oui, ok = 1).
Aller au pas 4.

Pas 4 : Si la fonction s'est bien achevée, la variable niveau prend la valeur n + 1 et proposition de continuer l'introduction ou d'arrêter.
Sinon l'utilisateur doit arrêter (ok = 0).
Aller au pas 5.

Pas 5 : Si l'utilisateur veut continuer (ok = 1) et si la variable niveau est strictement inférieure à 10, aller au pas 2.
Sinon, aller au pas 6.

Pas 6 : Fin de la procédure.

b) Fichier MEMALLOC.C : all_stat()

Cette procédure réalise la définition par l'utilisateur, de l'allocation de fichiers aux stations du réseau, et cela par énumération des stations.

Logique

Pas 1 : Mémorisation de l'allocation de fichiers, déclarée obligatoire par les contraintes d'emplacement.
Positionnement à la première station.
Aller au pas 2.

Pas 2 : Visualisation des fichiers dont l'allocation à la station est obligatoire.
Aller au pas 3.

Pas 3 : Proposition de prolongation de l'allocation de fichiers à la station.
Aller au pas 4.

Pas 4 : Saisie au terminal du nom du fichier à allouer à la station (avec contrôle de cohérence) par la procédure lec_fich().
Aller au pas 5.

Pas 5 : Si le retour de la procédure lec_fich signale la fin de l'introduction, aller au pas 6.
Sinon, mémorisation de l'allocation ainsi définie et aller au pas 4.

Pas 6 : Si des stations non encore envisagées existent, positionnement à la station suivante et aller au pas 2. Sinon aller au pas 7.

Pas 7 : Fin de la procédure.

Cette logique peut être adaptée pour réaliser la procédure all_fich().

c) Fichier MEMCONTR.C : ll_contrainte()

Cette procédure réalise l'introduction des contraintes d'emplacement, en procédant par énumération des stations.

Logique

Pas 1 : Affichage d'un message d'introduction.
Aller au pas 2.

Pas 2 : Visualisation du nom d'une station, non encore visualisée. Proposition d'introduction de contraintes d'emplacement pour cette station.
Aller au pas 3.

Pas 3 : Saisie du choix. Si l'utilisateur désire introduire des contraintes, aller au pas 4.
Sinon aller au pas 9.

Pas 4 : Saisie au terminal du nom du fichier auquel doit être imposée une contrainte d'emplacement.
Aller au pas 5.

Pas 5 : Contrôle du nom introduit.
Si le nom vaut '*', aller au pas 7.
Si le nom est celui d'un fichier inexistant, aller au pas 4.
Si le nom est celui d'un fichier existant, aller au pas 6.

Pas 6 : Proposition de trois types de contraintes.
Saisie du code symbolisant le type de contraintes à appliquer au fichier.
Mémorisation de la contrainte introduite.
Aller au pas 7.

Pas 7 : Proposition d'introduction d'une nouvelle contrainte pour la station.
aller au pas 8.

Pas 8 : Saisie du choix. Si l'utilisateur désire introduire une nouvelle contrainte, aller au pas 4.
Sinon, aller au pas 9.

Pas 9 : S'il existe une station non encore visualisée,
Aller au pas 2.
Sinon, aller au pas 10.

Pas 10 : Fin de la procédure.

Cette logique peut être adaptée pour réaliser la procédure 12-contrainte().

d) Fichier MEMPARA.C : modl_regime(ch)

Cette procédure permet de modifier la valeur du type de régime applicable au trafic : permanent ou jour-nuit.

Selon le type de régime courant, le nombre de paramètres utilisés dans la description, varie. De ce fait, il existe, au niveau supérieur, deux procédures différentes : mod_perm_option() et mod_jn_option(), pour modifier les paramètres.
Si l'utilisateur modifie le type de régime, il est nécessaire de mettre à jour l'écran car les informations sont différentes d'un régime à l'autre. De plus, l'utilisateur doit toujours avoir la possibilité de modifier les paramètres. Donc, le module adéquat doit être activé. Par exemple, envisageons que le type de régime soit permanent. L'utilisateur, par le module mod_perm_option(), active la procédure modl_régime(ch) pour que le régime devienne jour-nuit. A la fin de cette procédure, l'utilisateur doit garder la possibilité de modifier les paramètres et donc la procédure mod_jn_option() doit être activée. En bref, la modification du type de régime produit une cascade d'appels "récursifs" des deux procédures de niveau supérieur. De plus, quand l'utilisateur désire arrêter de modifier les paramètres, il faut que cet ordre remonte la cascade d'appels. Cela s'effectue en renvoyant une variable de niveau en niveau.

Logique

Pas 1 : Visualisation des deux choix possibles : 0 pour le régime permanent et 1 pour le régime jour-nuit.
Aller au pas 2.

Pas 2 : Saisie au terminal du choix, avec contrôle de validité.
Aller au pas 3.

Pas 3 : Si le nouveau régime choisi est du type permanent, aller au pas 7.
Sinon, aller au pas 4.

Pas 4 : Mémorisation du changement de régime.
Initialisation arbitraire des paramètres propres au régime jour-nuit.
Mise à jour de l'écran.
Aller au pas 5.

Pas 5 : Activation de la procédure mod_jn_option().
Aller au pas 6.

Pas 6 : Mise à zéro de la variable CH pour signifier que la procédure de niveau supérieur, mod_perm_option(), doit également s'arrêter.
Aller au pas 7.

Pas 7 : Fin de la procédure.

Cette logique est adaptable à la procédure mod2_régime(ch).

e) Fichier MEMROUTE.C : ent_route()

Cette procédure coordonne la détermination automatique d'un routage de poids minimal à travers le réseau et une modification du routage courant par l'utilisateur.

Logique

Pas 1 : Détermination automatique d'un routage de poids minimal et vérification du caractère connexe de la topologie du réseau (rech_hamilton()).
Aller au pas 2.

Pas 2 : Si la topologie du réseau est connexe, affichage d'un message à l'écran et aller au pas 7.
Sinon, aller au pas 3.

Pas 3 : Proposition de correction de la topologie ou de quitter le logiciel.
Aller au pas 4.

Pas 4 : Saisie du choix. Si le choix est de quitter, aller au pas 9. Sinon, aller au pas 5.

Pas 5 : Activation de la procédure de visualisation et de correction ent2-route().
Aller au pas 6.

Pas 6 : Si le retour de la procédure signale que l'utilisateur veut quitter le logiciel,
aller au pas 9.
Sinon, aller au pas 1.

Pas 7 : Proposition de modifier le routage déterminé par le logiciel.
Si l'utilisateur désire modifier le routage,
aller au pas 8.
Sinon, aller au pas 9.

Pas 8 : Activation de la procédure permettant de modifier le routage (enr_table_routage()).
Aller au pas 9.

Pas 9 : Fin de la procédure.

f) Fichier MEMROUTE.C : enr_table_routage()

Cette procédure permet de modifier le routage courant à travers le réseau.

Logique

Pas 1 : Affichage d'un message d'introduction.
Aller au pas 2.

Pas 2 : Saisie au terminal du nom de la station origine du routage à modifier (avec contrôle d'existence). Si le nom saisi diffère du symbole d'annulation '*', aller au pas 3.
Sinon, aller au pas 13.

Pas 3 : Saisie au terminal du nom de la station destination du routage à modifier (avec contrôle d'existence).
Si le nom saisi diffère du symbole d'annulation '*', aller au pas 4.
Sinon, aller au pas 13.

Pas 4 : Proposition de visualisation du routage courant entre les deux stations.
Aller au pas 5.

Pas 5 : Saisie du choix. Si l'utilisateur désire visualiser le routage, aller au pas 6.
Sinon, aller au pas 7.

Pas 6 : Visualisation du routage entre les deux stations dont les noms ont été introduits.
Aller au pas 7.

Pas 7 : Demande de confirmation de la volonté de modifier le routage entre les deux stations.
Aller au pas 8.

Pas 8 : Saisie du choix. Si l'utilisateur confirme, aller au pas 9.
Sinon, au pas 13.

Pas 9 : Activation de la procédure permettant l'introduction du nouveau routage (avec divers contrôles de cohérence).
Aller au pas 10.

Pas 10 : Si le retour de la procédure signale que tout s'est bien passé (ok = 1), mémorisation du nouveau routage entre les deux stations et aller au pas 11. Sinon, aller au pas 13.

Pas 11 : Si les lignes du réseau sont bidirectionnelles, proposition d'appliquer le même routage pour la liaison inverse et aller au pas 12.
Sinon, aller au pas 13.

Pas 12 : Mémorisation du nouveau routage pour la liaison inverse.
Aller au pas 13.

Pas 13 : Proposition de modification du routage entre deux stations.
aller au pas 14.

Pas 14 : Saisie du choix. Si l'utilisateur désire modifier un routage, aller au pas 2.
Sinon, aller au pas 15.

Pas 15 : Fin de la procédure.

g) Fichier MEMROUTE.C : visu_route()

Cette procédure permet de visualiser le routage entre deux stations de numéro n1 et n2.
La logique se décompose en deux phases :

- construction d'une table "passe" qui contient dans ses k premières positions, la liste des numéros des k stations participant, dans cet ordre, au routage entre les stations de numéros n1 et n2. Cette phase est réalisée par les pas 1, 2, 3, 4 et 5.
- visualisation de la liste, ordonnée par la première phase, des stations participant au routage entre les stations de numéros n1 et n2. Cette phase est réalisée par le pas 6.

Logique

Pas 1 : prec ⟵ nom de la station origine du routage
c'est-à-dire dont le numéro est n1.
k ⟵ 0
Aller au pas 2.

Pas 2 : i ⟵ numéro de la ligne dont prec est la station initiale.
Aller au pas 3.

Pas 3 : passe (k) ⟵ numéro de la station de nom prec
k ⟵ k + 1
prec ⟵ nom de la station finale de la ligne numéro i.
Aller au pas 4.

Pas 4 : Si prec est le nom de la station destination du routage, c'est-à-dire dont le numéro est n2, aller au pas 5. Sinon, aller au pas 2.

Pas 5 : passe (k) ⟵ n2.
Aller au pas 6.

Pas 6 : Affichage des stations participant au routage, en suivant "passe" et cela à raison de trois stations par ligne d'écran et de quatre lignes par écran.
Aller au pas 7.

Pas 7 : Fin de la procédure.

h) Fichier MEMSTAT.C : intro_stat()

Cette procédure réalise la saisie au terminal de la description des stations du réseau.

Logique

Pas 1 : Affichage d'un message d'introduction . nbs ⟵ o.
Aller au pas 2.

Pas 2 : Affichage à l'écran d'une grille de lecture vierge pour cinq stations.
Aller au pas 3.

Pas 3 : Saisie au terminal du nom de la station.
Aller au pas 4.

Pas 4 : Contrôle du nom introduit.
Si le nom vaut '*', aller au pas 8.
Si le nom est celui d'une station déjà saisie, aller au pas 3.
Si le nom est celui d'une station non déjà saisie, aller au pas 5.

Pas 5 : Saisie au terminal des autres informations sur la station . nbs ⟵ nbs + 1.
Aller au pas 6.

Pas 6 : Si nbs est strictement inférieur à SMAX-1, aller au pas 7. Sinon, aller au pas 8.

Pas 7 : Si nbs est un multiple de cinq, aller au pas 1.
Sinon, aller au pas 2.

Pas 8 : Fin de la procédure.

Cette logique peut être adaptée pour réaliser d'autres fonctions similaires quant à l'objectif. Les procédures suivantes reprennent la même démarche :

- fichier MEMNOEUD.C : intro_noeud()
- fichier MEMLIGNE.C : lec_uni()
  lec_bi()
- FICHIER MEMFICH.C : intro_fich()

i) Fichier MEMSTAT.C : mod_stat_intro()

Cette procédure réalise la visualisation des stations introduites et offre la possibilité de les modifier.

Logique

Pas 1 : Si aucune station n'a été introduite, affichage d'un message et aller au pas 12.
Sinon, aller au pas 2.

Pas 2 : Visualisation de maximum cinq stations.
Aller au pas 3.

Pas 3 : Offre de modification d'une des stations visualisées à l'écran.
Aller au pas 4.

Pas 4 : Saisie du choix. Si le choix est de continuer la visualisation, aller au pas 11. Si le choix est de quitter, aller au pas 12. Si le choix est de modifier une station, aller au pas 5.

Pas 5 : Affichage d'une grille de lecture de la description de la station à modifier.
Aller au pas 6.

Pas 6 : Saisie au terminal du numéro de la station à modifier. Si le numéro n'est pas celui d'une des stations affichées à l'écran, aller au pas 6.
Sinon, aller au pas 7.

Pas 7 : Saisie au terminal d'un nom pour la station.
Si ce nom est différent du nom courant de la station et est celui d'une station déjà mémorisée, aller au pas 7.
Sinon, aller au pas 8.

Pas 8 : Saisie au terminal des autres informations sur la station.
Aller au pas 9.

Pas 9 : Demande de confirmation de la modification à apporter.
Si la confirmation est donnée, aller au pas 10.
Sinon, aller au pas 3.

Pas 10 : Mise à jour de la table où sont mémorisées les stations, à partir des données introduites et mise à jour de l'affichage à l'écran de la station qui a été modifiée.
Aller au pas 3.

Pas 11 : Si toutes les stations ont été visualisées, aller au pas 12. Sinon, aller au pas 2.

Pas 12 : Fin de la procédure.

Cette logique peut être adaptée pour réaliser d'autres fonctions similaires quant à l'objectif. Les procédures suivantes reprennent la même démarche :

- fichier MEMNOEUD.C : mod_noeud_intro()
- fichier MEMLIGNE.C : mod_uni()
  mod_bi()
- fichier MEMFICHIER.C : mod_fich_intro()
- fichier MEMCAPAC.C : modl_capac()
  mod2_capac()
  mod3_capac()
  mod4_capac()
  mod2_const_capac()

- fichier MEMFIAB.C : mod_fiabilité()
- fichier MEMTRAF.C : mod_perm_traf()
  mod_jn_traf()

4.4. CONCLUSION

Dans ce chapitre quatre, nous venons de présenter quelques éléments permettant de comprendre la démarche suivie pour implémenter le logiciel. Ces éléments constituent un dossier minimal qui regroupe les points les plus délicats auxquels nous avons dû nous attacher dont principalement les problèmes posés par l'environnement matériel et logiciel.

C'est sur ces bases qu'a été implémenté notre logiciel. Cependant, au fil de son évolution, le logiciel est apparu limité à certains points de vue. Le chapitre cinq va tenter d'amorcer une réflexion sur des améliorations et extensions susceptibles d'être apportées au logiciel.

◊

CHAPITRE 5

ETUDE CRITIQUE DU LOGICIEL

5.1. ANALYSE CRITIQUE DU LOGICIEL

5.2. QUELQUES AMELIORATIONS POTENTIELLES

5.3. QUELQUES EXTENSIONS POTENTIELLES

5.4. CONCLUSION

Dans les chapitres précédents, nous avons présenté le développement d'un logiciel de conception assistée par ordinateur d'une configuration de réseau. Nous allons maintenant envisager une série d'améliorations et d'extensions qui pourraient être apportées à notre logiciel. Ces modifications ont pour objectif de répondre à un certain nombre de critiques qui peuvent être formulées quant à notre modèle présenté au chapitre 1 et à notre analyse fonctionnelle du chapitre 3.

5.1. ANALYSE CRITIQUE DU LOGICIEL

5.1.1. Les limites du logiciel

Au terme de notre travail, nous sommes parfaitement conscients que notre étude ne nous a permis que de développer une ébauche d'un logiciel de conception assistée par ordinateur. En effet, par manque de temps, un certain nombre d'aspects intéressants n'ont pu être envisagés.

Un premier aspect consiste en la mise au point d'un interface homme/machine très agréable et de fonctions facilitant au maximum le travail de l'utilisateur. Bien que disposant déjà d'un certain nombre de facilités, notre logiciel manque un peu de souplesse et, à ce niveau, un certain nombre d'aménagements sont envisageables, comme nous allons le détailler par la suite.

Un deuxième aspect qu'il serait bon de développer est lié à notre modèle théorique du chapitre 1. En effet, nous sommes partis d'un modèle certes complet mais un peu simpliste sur certains points. D'où une légère distorsion par rapport à la réalité. Pour pallier à ce défaut, quelques pistes de recherche intéressantes vont être présentées.

Enfin, une analyse sérieuse du logiciel ne peut manquer de mentionner son manque de portabilité, défaut dû aux contraintes imposées par l'environnement logiciel et matériel dans lequel nous avons dû travailler (4.2.). Ces mêmes contraintes impliquent une limite essentielle du logiciel : la taille des problèmes susceptibles d'être traités.

5.1.2. Applicabilité du logiciel

Lors de la conception initiale du logiciel, l'objectif a été de maintenir le plus de données en mémoire centrale pour favoriser la vitesse de consultation des informations. Au fil du temps, cette optique a conduit à garder en mémoire de très nombreux tableaux à plusieurs dimensions. Or, la capacité maximale permise pour les données est théoriquement de 64 KB. Il s'ensuit une limitation quant à l'applicabilité du logiciel. Dans ce qui suit, figure une étude pour cerner les éléments clé de cette limitation.

A la simple lecture des variables globales du logiciel, il apparaît que le facteur déterminant est le nombre de stations figurant dans la topologie (SMAX). Ci-dessous figurent quelques chiffres significatifs situant les limites réelles du logiciel.

CAS / BORNES	1	2	3	4	5
stations	13	12	18	17	10
noeuds	5	5	0	0	10
lignes	30	30	17	16	40
fichiers	5	10	5	10	10
capacités	20	20	20	20	20

Nous avons pris l'hypothèse que le jeu de valeurs de capacité disponibles est fourni par un organisme sous la forme d'un catalogue (ex RTT). Un jeu de cinq valeurs de capacité pour chacun des quatre TYPES (1.1.7.) est concevable. De plus, les lignes sont unidirectionnelles.

Dans les cas numéros 1, 2 et 5, les configurations classiques de topologie sont représentables. Dans les deux autres cas, une configuration en anneau unidirectionnel est envisagée, avec uniquement des stations. Même dans ce cadre restreint, le nombre maximal de stations est limité à dix-huit.

De plus, différentes expériences ont revélé que, lorsque le nombre maximal de stations est accru, l'espace mémoire total nécessaire au logiciel s'accroît plus vite.
Par exemple, dans le cas numéro 4, le passage de 16 à 17 stations a accru la place mémoire nécessaire de dix pourcents.

En résumé, le facteur limitant le plus l'applicabilité du logiciel est le nombre maximal de stations dans la topologie.

5.1.3. Conclusion

Il n'est évidemment pas possible de rester insensible à ces critiques parfaitement fondées. C'est la raison pour laquelle une réflexion a été entamée dans le but de proposer une série de modifications potentielles.

Dans les paragraphes suivants, nous avons classé les modifications qu'il serait envisageable de développer en deux catégories : les améliorations et les extensions.
Les améliorations consistent en des modifications qui peuvent être apportées au logiciel existant, sans pour cela changer de manière profonde le logiciel. A l'opposé, les extensions demandent à revoir en profondeur une large partie du logiciel,

voire même de le recommencer en ne reprenant que les parties utiles de l'existant.

5.2. QUELQUES AMELIORATIONS POTENTIELLES

5.2.1. Analyse théorique complémentaire

Dans le chapitre 1, nous avons exposé un modèle de description d'un réseau informatique. Par rapport à ce qui se passe dans la réalité, cette modélisation se révèle un peu simpliste par certains aspects. Pour cette raison, il s'avérerait intéressant de pousser plus loin l'analyse théorique de notre modèle.

A notre avis, les pistes de recherche les plus pertinentes sont :

- une nouvelle formule de calcul du délai moyen
- une nouvelle vision du trafic
- la définition de nouvelles fonctions de coût.

5.2.1.1. Analyse du délai

Dans notre modèle, nous utilisons pour le calcul du délai moyen de transmission, la formule développée par Kleinrock (KLE.70) :

$$T = \sum_{i=1}^{NBL} (\lambda_i/\gamma) * (1/(\mu_i * C_i - \lambda_i)) + \text{délai-réseau} * \delta_i$$

Dans cette formule interviennent uniquement les notions de valeur du flux d'information et de capacité des lignes de transmission, les concepts de temps de traitement aux noeuds ou de délai de propagation n'étant pas pris en compte.

Pour ce motif, il serait intéressant de développer une nouvelle formule de calcul du délai, où interviendrait le plus de paramètres réels possibles. Pour une première approche de ce problème, une courte synthèse des travaux existants est présentée dans le travail de Pichot et Detalle (PIC.84).

Comme nous le verrons par la suite (5.2.1.2.), si nous affinons notre vision du concept de trafic à travers le réseau, il devient envisageable de ne plus se contenter de la valeur moyenne du délai, mais d'extrapoler une "variance" de ce délai. Ceci peut constituer une autre piste de recherche.

Il est à signaler que dans notre modèle, nous considérons un délai global calculé sur toutes les classes de messages. Une approche plus réaliste peut être envisagée : considérer une contrainte de délai pour chaque type de messages. En particulier, le trafic de mise à jour pourrait ne pas subir de contrainte de délai.

5.2.1.2. Le trafic à travers le réseau

Dans le paragraphe 1.1.8., nous décomposons le trafic à travers le réseau en quatre composantes :

- le "query traffic"
- le "update traffic"
- le "query return traffic"
- le "update return traffic"

Pour chaque composante, nous supposons connaître sa valeur moyenne en terme de messages émis par seconde. Cependant, nous considérons dans ce cas que le trafic est constant et invariable. Or, dans la réalité, ce trafic peut varier considérablement au cours d'une journée et comporter à certains moments des creux et des pointes de débit très importants.

De ce fait, notre modèle se révèle insuffisant à ce niveau car dans notre logiciel, nous offrons seulement la possibilité de différencier le trafic de jour et de nuit.

Ceci permet déjà de dégrossir un peu le problème. Un cas typique résoluble par ce système est celui où le "update traffic" ne se produirait que la nuit.

Cependant, il va de soi que l'idéal serait de posséder la distribution du trafic sur une journée et, si nous sommes moins exigeant, sa variance.

Cette nouvelle approche de la notion de trafic complexifie évidemment notre problème, tout en le rendant plus réaliste. Il reste évidemment à examiner comment tirer parti de cette information supplémentaire. Il nous semble que c'est principalement au niveau du délai de transmission que l'intérêt porte en ce sens qu'il serait dès lors possible de mieux cerner le délai réel au cours d'une journée et en particulier ses variations.

5.2.1.3. Fonctions de coût

Si nous examinons de près les principes de tarification des valeurs de capacité, énoncés en 1.1.7., dans le cas des lignes non louées, nous remarquons que le coût des appels pour l'établissement des communications n'intervient pas. En effet, comme nous ne possédons que la valeur moyenne du trafic, il est quasi impossible de déterminer le nombre d'appels effectués.

Pour s'en rendre compte, il suffit d'observer que,lors des pointes de trafic, si plusieurs messages doivent être expédiés au même instant à une station, un seul appel sera effectué pour tous ces messages. A l'opposé, lors des périodes creuses, un appel sera effectué par message.

Seule, une connaissance de la distribution du trafic pourrait permettre d'extrapoler le nombre d'appels effectués sur une journée. Bien que peu évident de prime abord, ce problème nous semble mériter un peu d'attention, comme tout ce qui a trait au coût en général.

Il est à signaler qu'une approche plus simple, mais moins satisfaisante, pour faire intervenir les appels dans la fonction de coût, est de demander au concepteur de spécifier le nombre d'appels pour chaque liaison.

5.2.1.4. Conclusion

Les trois aspects que nous venons de présenter nous semblent très intéressants à étudier en ce sens qu'ils permettraient d'affiner notre modèle et de le rapprocher de la réalité. Le principal impact serait une meilleure estimation des coûts réels des solutions et de leur performance.

5.2.2. Amélioration de l'introduction des données

Après avoir implémenté les fonctions d'introduction d'une description de réseau, décrites au paragraphe 3.1., il est apparu que quelques facilités supplémentaires auraient pü être offertes à l'utilisateur.

Les facilités qui nous semblent les plus pertinentes sont :

- des fonctions d'ajout et de suppression d'entités lors de l'introduction
- une plus grande souplesse de modification de l'introduction
- la possibilité de mémoriser l'introduction sur plusieurs fichiers de données.

5.2.2.1. Fonction d'ajout et de suppression

Dans les spécifications fonctionnelles d'introduction des stations, des noeuds, des lignes, des fichiers et des capacités, nous avons précisé que, dans notre logiciel, une fois ces entités introduites, il n'est pas possible d'en ajouter d'autres ou d'en supprimer. Cette manipulation n'est réalisable que lors de la recherche d'une solution, au moyen de la fonction "Visualisation et modification du réseau courant".

Une amélioration intéressante serait de le permettre lors de la visualisation des entités introduites, en plus de la possibilité existante de modifier les valeurs des entités.

5.2.2.2. Souplesse de modification

Par le terme "souplesse de modification", nous entendons la possibilité, arrivé à un certain stade de l'introduction, de revenir en arrière dans ce qui a déjà été introduit et de le modifier.

Explicitons cela sur un exemple simple. Supposons que lors d'une précédente session de travail, l'utilisateur ait introduit les stations et les noeuds de son réseau. Avec le logiciel actuel, lors de la reprise du travail, la fonction d'introduction suivante, c'est-à-dire celle des lignes, est activée. L'utilisateur ne peut revoir ce qu'il a déjà introduit.

Une amélioration du logiciel est de permettre à l'utilisateur, avant de poursuivre l'introduction des données, de visualiser ce qu'il a déjà introduit et éventuellement de le modifier.

Ceci nous amène à envisager une fonction du type de celle présentée au paragraphe 3.2.4. "visualisation et modification du réseau courant".

Une telle approche apporte à l'utilisateur une plus grande souplesse car il peut très facilement corriger une erreur ou un oubli, sans devoir attendre d'avoir terminé toute son introduction.

De plus, un gain appréciable en temps peut en découler Envisageons un cas typique. Supposons que l'utilisateur ait décrit la topologie de son réseau. A ce moment, il désire ajouter une nouvelle station. Avec le logiciel actuel, ce n'est pas possible. L'utilisateur doit terminer son introduction et ensuite activer la fonction ad-hoc, ce qui entraîne une coûteuse mise à jour d'une grande partie des tables de description du réseau. Avec la solution proposée ici, il peut ajouter la station et la connecter au réseau avant d'introduire tous les autres paramètres comme si de rien n'était.

5.2.2.3. Stockage multiple

Dans nos spécifications fonctionnelles (3.1.11.), nous avons précisé que, lors du début de son travail, l'utilisateur doit fournir au logiciel le nom du fichier de données où seront stockées les données introduites. Sauf par des manipulations extérieures au logiciel, il ne peut en changer le nom ou en faire des copies.

Cependant, l'utilisateur peut souhaiter nommer différemment divers stades de son introduction. Envisageons un exemple. Supposons que l'utilisateur ait introduit la topologie d'un réseau et qu'il ait de ce fait créé un fichier de données nommé F1. Par la suite, il introduit tous les autres paramètres du résau (fiabilité, trafic,...). Une amélioration est de lui permettre de stocker son réseau entièrement décrit sur un fichier nommé F2.

Suite à cette opération, il existe deux fichiers liés au problème :

- F1 qui ne contient que la topologie du réseau
- F2 qui contient un réseau, issu de F1, complètement décrit.

Si par la suite, l'utilisateur désire modifier tous les paramètres de son réseau hormis la topologie, il peut

- soit activer lui-même les fonctions ad hoc du module "Visualisation et modification du réseau courant" (3.2.4.)
- soit fournir au logiciel le nom du fichier F1. Le logiciel active alors les fonctions d'introdiction nécessaires (3.1.).

En résumé, il est intéressant d'offrir les fonctions suivantes :

- reprise d'une session de travail en fournissant le nom d'un fichier de données
- sauver les données introduites sur un fichier de données auquel l'utilisateur lui donne le nom qu'il désire et cela après chacune des dix activations des fonctions d'introduction (3.1.1. —> 3.1.10.).

Note

Une extension de ce concept de gestion des fichiers de données est exposée au paragraphe 5.2.5.

5.2.3. Recherche d'une solution

A la suite de réflexions ultérieures à la définition des spécifications fonctionnelles de ce module (3.2.), nous pouvons proposer deux pistes de recherche pour améliorer certaines fonctions :

- une nouvelle gestion des capacités admissibles
- une étude approfondie des possibilités de fournir à l'utilisateur une aide à la conception plus poussée.

5.2.3.1. Gestion des capacités

Dans notre travail, lors de la détermination par le logiciel de l'allocation de capacités, l'ensemble des valeurs de capacité sont prises en compte. En d'autres termes, l'utilisateur ne sait pas déterminer à l'avance le type de capacité (louée ou non louée, avec commutation par paquets ou non) qui sera utilisé pour telle ligne.

Or, pour des raisons techniques, ou autres, l'utilisateur peut désirer que, pour telle ligne, le logiciel ne puisse utiliser que les valeurs de capacité de tel type.

Une amélioration de notre logiciel est de permettre à l'utilisateur de mentionner pour chaque ligne, le ou les type(s) de capacité admissible(s) parmi les quatre que nous proposons.

En allant même plus loin, l'utilisateur pourrait avoir la possibilité de fixer non seulement le type de la capacité, mais même la valeur à allouer à telle ligne. Bien entendu, le logiciel devrait alors vérifier que cette valeur de capacité est suffisante pour supporter le flux d'informations transitant sur cette ligne et établir, si ce n'est pas le cas, un dialogue avec l'utilisateur pour corriger cette insuffisance.

5.2.3.2. Aide à la conception

Dans les spécifications fonctionnelles énoncées au paragraphe 3.2.6., nous avons précisé la nature des messages d'aide à la conception que le logiciel peut fournir à l'utilisateur en cas de violation d'une contrainte de faisabilité.

Dans tous les cas, les messages ne contiennent que des informations passives, c'est-à-dire des informations que le logiciel peut extraire des tables où est mémorisée la description du réseau et cela sans devoir effectuer de

calculs. Il peut être pertinent d'étudier si, moyennant quelques calculs simples, le logiciel ne pourrait guider plus encore l'utilisateur, voir même proposer des solutions.

Envisageons comme exemple la vérification de la contrainte de stockage des stations. On peut imaginer qu'en cas de violation pour une station X, le logiciel propose de déplacer un des fichiers alloués en X vers une autre station Y proche de X, de telle sorte que la contrainte de stockage soit respectée.

Il va de soi que pour fournir une telle aide, le logiciel doit effectuer des calculs qui peuvent s'avérer assez lourds. Dans ce cas, il faut examiner si cette charge en temps-calcul est supportable, pour ce que cela peut rapporter à l'utilisateur. Il y a donc lieu de trouver un compromis car il est nécessaire de bien garder à l'esprit qu'un logiciel de conception assistée par ordinateur ne peut nécessiter de longs temps-calcul, ni de complexes algorithmes d'optimisation.

Une solution peut être de proposer une aide à la conception à deux niveaux. En cas de problème, le logiciel apporte à l'utilisateur des renseignements comme présentés en 3.2.6., c'est-à-dire une aide passive. Ensuite, à la demande de l'utilisateur, une aide plus poussée, active, est apportée par le logiciel. L'utilisateur est seul maître de savoir s'il accepte la charge de calculs découlant de cet apport de niveau supérieur.

Nous tenons à signaler que nous n'avons pas prolongé plus loin notre analyse quant à la teneur de cette aide de niveau supérieur ni même quant à son caractère réalisable. Nous nous contentons d'indiquer une piste de recherche qui peut s'avérer fructueuse.

5.2.4. Problème de terminal

Comme nous l'avons mentionné au paragraphe 4.2., pour implémenter notre logiciel, nous avons été confronté à un environnement logiciel et matériel qui nous a amené à restreindre la portabilité de notre travail.

Un problème important est l'absence de fonctions standard de gestion d'écran dans le langage C implémenté sur le système UNIX du PDP 11.

En l'absence d'une librairie standard, nous avons dû implémenter la fonction de positionnement du curseur mais en nous limitant à un type de terminal. Dans notre cas, le VT100 de DEC fut choisi.

De ce fait, notre logiciel n'est utilisable que sur un terminal de type VT100. Une amélioration serait d'élargir la gamme des terminaux utilisables.

Une solution serait de cacher la diversité des terminaux dans le module des fonctions techniques où figureraient les différentes fonctions de positionnement du curseur propres à chaque type de terminaux.

Lors du début de chaque session de travail, l'utilisateur serait amené à préciser le type de son terminal parmi la liste de ceux pouvant supporter le logiciel c'est-à-dire de ceux ayant une fonction ad hoc dans le module technique. Le logiciel se chargerait alors d'appeler la bonne fonction.

Ce raisonnement n'a d'intérêt que dans un environnement où cohabitent différents types de terminaux ou si nous désirons un logiciel le plus portable possible.

Note

Un raisonnement analogue peut être tenu quant à la fonction d'initialisation en mode asynchrone du terminal.

5.2.5. Gestion des fichiers de données par le logiciel

Dans notre analyse fonctionnelle, nous avons mis en évidence qu'il est intéressant de permettre à l'utilisateur de stocker les différentes étapes de son travail sur des fichiers de données (3.2.2.), ceci pour lui permettre de revenir en arrière dans sa démarche de résolution.

Dans notre logiciel, nous avons laissé à l'utilisateur le soin de gérer lui-même la liste de ses fichiers de données, le logiciel se limitant à effectuer des contrôles d'existence.

Dans le cadre d'une conception assistée par ordinateur, il serait intéressant que le logiciel aide l'utilisateur dans sa gestion.

5.2.5.1. La notion de "directory"

Une solution est que le logiciel gère un genre de "directory" des fichiers de données. En d'autres termes, le logiciel est amené à tenir à jour un mini-fichier dont le nom lui est donné par l'utilisateur, mini-fichier qui contient des fichiers de données relatives à un problème.

Pour l'utilisateur, à un problème à résoudre correspond un "directory". En plus de la liste des fichiers de données, le "directory" mentionne la date de la création ou de la dernière mise-à-jour de chacun des fichiers.

Exemple

directory ARPANET

liste des fichiers :	VERSION 1	3/07/86
	VERSION 2	6/07/86
	VERSION 3	8/07/86

Dans ce qui va suivre, nous allons présenter trois scénarios illustrant la logique de l'utilisation d'un "directory" de fichiers de données :

- définition d'un "directory"
- reprise d'un "directory"
- gestion d'un "directory"

5.2.5.2. Définition d'un "directory"

Lors de l'accès au logiciel, l'utilisateur doit préciser s'il désire définir un nouveau problème ou reprendre un problème antérieurement défini. Dans ce premier scénario, nous supposons qu'il veut définir un nouveau problème.

Dans ce cas, l'utilisateur doit fournir deux choses au logiciel :

- le nom identifiant le "directory"
- le nom identifiant le fichier de données où seront stockées les premières informations.

Le logiciel crée alors un mini-fichier ayant pour nom celui du "directory" et qui contient le nom du fichier de données auquel il ajoute la date courante. Cela étant fait, le logiciel peut commencer à activer les diverses fonctions d'introduction des données.

5.2.5.3. Reprise d'un "directory"

Plaçons-nous dans le cas de la reprise d'un problème antérieurement introduit. L'utilisateur doit alors fournir au logiciel le nom du "directory" relatif à son problème. En général, ce "directory" contient plusieurs noms de fichiers de données. Le logiciel ne sachant pas sur lequel travailler, il affiche à l'écran la liste des noms de fichiers. L'utilisateur doit alors sélectionner le fichier qu'il désire voir actif. Le logiciel peut ensuite suivre son cours normal.

5.2.5.4. Gestion d'un "directory"

Dans le logiciel tel qu'il est spécifié au chapitre 3, les seules manipulations des fichiers de données qui sont offertes à l'utilisateur sont :

- sauvetage de la description du réseau courant sur un fichier de données.
- reconstitution d'un réseau courant à partir d'un fichier de données.

Maintenant que nous avons défini la notion de "directory", nous pouvons envisager une gestion plus large des fichiers de données.

Les fonctions suivantes peuvent être envisagées :

a) Création d'un nouveau fichier de données à partir de la description du réseau courant en mémoire centrale.

b) Lecture d'un fichier de données pour reconstituer un nouveau réseau courant.

c) Suppression d'un fichier de données du "directory".

d) Changement du nom d'un fichier de données.

e) Copie d'un fichier de données sous un autre nom (raison de sécurité par exemple).

f) Insertion de commentaires sur les fichiers de données.

Nous pouvons imaginer qu'en plus de la date de la dernière mise à jour du fichier, le concepteur aimerait ajouter quelques commentaires pour lui permettre de mieux situer la version du réseau contenue dans un fichier de données. Ce commentaire peut être inséré lors de la création du fichier de données et modifié par la suite, sur demande de l'utilisateur.

g) Lecture du commentaire relatif à un fichier de données.

L'utilisateur a la possibilité de consulter le

commentaire relatif à n'importe lequel des fichiers dont le nom figure dans le "directory".

Par rapport au logiciel existant, cet ensemble de fonctions de gestion remplacerait les deux fonctions relatives aux fichiers de données, présentes dans le menu du module. "Recherche d'une solution" (3.2.2. et 3.2.3.).

5.2.6. Particularisation au réseau belge

Le logiciel que nous avons développé est applicable à tout réseau qui rentre dans le cadre de notre modèle théorique, en particulier au réseau belge.

Pour un utilisateur belge, il peut s'avérer fastidieux d'introduire au terminal une série de données qui sont invariantes pour tout réseau (liste des capacités, tarif,...) Pour cette raison, il peut être intéressant que le logiciel connaisse à l'avance ces paramètres. Dans ce cas, il suffit à l'utilisateur de préciser au logiciel s'il doit prendre en compte les valeurs qu'il possède en mémoire ou activer les fonctions d'introduction.

Bien évidemment, ces données permanentes du système peuvent changer au fil du temps, par exemple à cause de hausses de tarif. Il est donc nécessaire d'associer au logiciel des fonctions permettant de modifier ces données.

5.3. LES EXTENSIONS POTENTIELLES

5.3.1. Elargissement de l'applicabilité du logiciel

Dans ce paragraphe, nous allons présenter une brève analyse en rapport à une critique qui peut être formulée vis-à-vis de notre logiciel : la taille restreinte des exemples pouvant être traités.

Dans notre travail, nous prenons l'optique de garder toutes les informations en mémoire centrale avec stockage périodique sur un fichier de données. Cette décision est motivée par le souci de limiter au maximum les accès à des fichiers sur disque, accès qui nécessite un tempx CPU non négligeable. Hélas, les caractéristiques du système à notre disposition nous limitent quant à la taille des problèmes pouvant être envisagés.

Une extension triviale de notre travail est d'élargir l'applicabilité du logiciel à des exemples plus conséquents, c'est-à-dire à des problèmes de taille réelle.

Une première solution est évidemment de transférer le logiciel sur un système compatible mais dont les possibilités en mémoire centrale sont supérieures. Cependant, nous estimons que ceci ne fait que reporter le problème sans s'attaquer à son coeur.

Si nous voulons nous affranchir de cette contrainte liée à la place en mémoire centrale réservée aux données, il est nécessaire de faire appel à une mémoire auxiliaire.

De manière pratique, deux optiques s'offrent à nous : l'accès ponctuel à un fichier et l'accès sélectif.

L'accès ponctuel consiste à n'accéder à une information en mémoire auxiliaire que quand le logiciel en a besoin. Citons deux exemples. La procédure de vérification de l'existence d'une station effectue un accès à la mémoire auxiliaire pour obtenir l'information désirée. De même, la visualisation du nom d'une station nécessite un accès disque. Il est évident que, dans cette optique, la charge en temps CPU requis par ces accès est relativement importante et qu'une organisation particulière des structures de données est nécessaire pour tenter de réduire cette charge au maximum. L'avantage de cette technique est qu'il n'y a presque plus de limite à la taille des problèmes envisageables.

L'accès sélectif consiste à sélectionner en mémoire auxiliaire les informations strictement nécessaires à la réalisation d'une fonction. A titre d'exemple, la fonction "Introduction des fiabilités" ne fait appel qu'à la matrice des fiabilités et à la description des stations. Cependant, d'autres fonctions peuvent nécessiter beaucoup plus d'informations, comme par exemple la fonction "Introduction du trafic". Dans cette optique, nous pouvons espérer avoir la possibilité de traiter des problèmes de taille respectable, mais inférieure à ceux envisageables par l'accès ponctuel. Il est à signaler qu'il peut être également intéressant dans ce cadre, d'étudier une nouvelle structure de données plus économique en place mémoire.

Personnellement, nous estimons que l'optique de l'accès sélectif est la plus prometteuse car elle permet d'élargir l'applicabilité du logiciel sans faire exploser le temps CPU requis par les accès en mémoire auxiliaire. Cependant, les deux optiques méritent une analyse plus approfondie.

5.3.2. Connexion entre un logiciel C.A.O. et des algorithmes optimaux

D'un point de vue méthodologique, les options "Conception assistée par ordinateur", notée C.A.O., et "Algorithmes optimaux et heuristiques" sont fondamentalement opposées en ce sens que dans la C.A.O., il subsiste très peu de fonctions de calcul où interviennent les algorithmes d'optimisation et que dans ce cas, l'utilisateur du logiciel est fortement impliqué dans la démarche de résolution.

Dans le domaine qui nous occupe, la littérature stipule que sur des exemples un rien conséquents, les techniques algorithmiques sont inapplicables (PIC.84).
Cependant, à l'inverse, si un logiciel C.A.O. permet au concepteur d'apporter son expérience et sa connaissance du problème et de traiter des exemples plus vastes, il reste que le concepteur peut devoir chercher longtemps une solution

satisfaisante et s'estimer satisfait d'un résultat qui, économiquement parlant, peut encore être amélioré.

Sur base de ces observations, nous nous posons la question de savoir s'il n'est pas possible de concilier les deux approches pour tirer parti de leur avantage. Dans ce type de collaboration, deux techniques de base sont envisageables :

- une conception assistée par ordinateur suivie d'une recherche algorithmique
- une recherche algorithmique suivie d'une conception assistée par ordinateur.

Dans la première optique, le but de la C.A.O. est de fournir à un algorithme de recherche optimal ou heuristique, un point de départ qui soit déjà satisfaisant. Ce type de démarche est intéressant dans le cas où il existe un algorithme convergent dont la vitesse de convergence près de l'optimum est élevée. Cette technique peut être aussi envisagée sous un autre point de vue. Dans un premier temps, le concepteur détermine une solution avec le logiciel C.A.O. Quand il ne sait plus l'améliorer, il cède la main à une technique algorithmique pour tenter d'améliorer la solution, par exemple à un algorithme de type "add and drop" pour l'allocation de fichier (PIC.84). Il va de soi que dans ce cas, les algorithmes ne peuvent être trop lourds en temps CPU.

Dans la deuxième optique, le logiciel C.A.O. est utilisé pour affiner une solution obtenue par un algorithme optimal. Cette solution doit être perçue comme une première approximation calculée en un temps raisonnable. Il existe par exemple, dans la littérature, des techniques permettant de restreindre l'espace de recherche d'une solution (PIC.84).

En première analyse, nous ne pouvons indiquer laquelle de ces deux techniques est la plus prometteuse car cela dépend pour beaucoup de la qualité des algorithmes optimaux ou heuristiques mis en oeuvre.

Les quelques idées présentées ci-dessus constituent une piste de recherche pour laquelle rien ne nous permet d'affirmer sa validité. Cependant, il nous semble que ces idées méritent de s'y attarder un peu, même si ce n'est que pour se convaincre que cette optique est irréalisable ou sans intérêt (ce qui est déjà un résultat appréciable).

5.3.3. Un interface graphique

Comme nous l'avons déjà mentionné (3.1.), nous avons élaboré notre logiciel dans un environnement matériel ne disposant pas d'outils graphiques. Or, il est trivial qu'il est plus aisé de réfléchir avec un support graphique. Dans notre contexte, il est plus facile de visualiser la topologie d'un réseau si elle est affichée à l'écran via un schéma que par des tables.

En l'absence d'une telle technologie, l'utilisateur peut évidemment se satisfaire d'un schéma réalisé sur une feuille de papier. Cependant, ce système est peu apte à subir les fréquentes modifications qui peuvent être imposées à la topologie.

De cette observation est née l'idée de doter le logiciel d'outils graphiques lui permettant la visualisation du réseau. De plus, il est dès lors possible de tirer parti des outils graphiques pour offrir un interface homme/machine plus agréable.

Dans ce qui va suivre, nous allons exposer quelques idées qui permettront à une personne intéressée par cette extension, de démarrer une réflexion plus approfondie.

5.3.3.1. Visualisation de la topologie du réseau

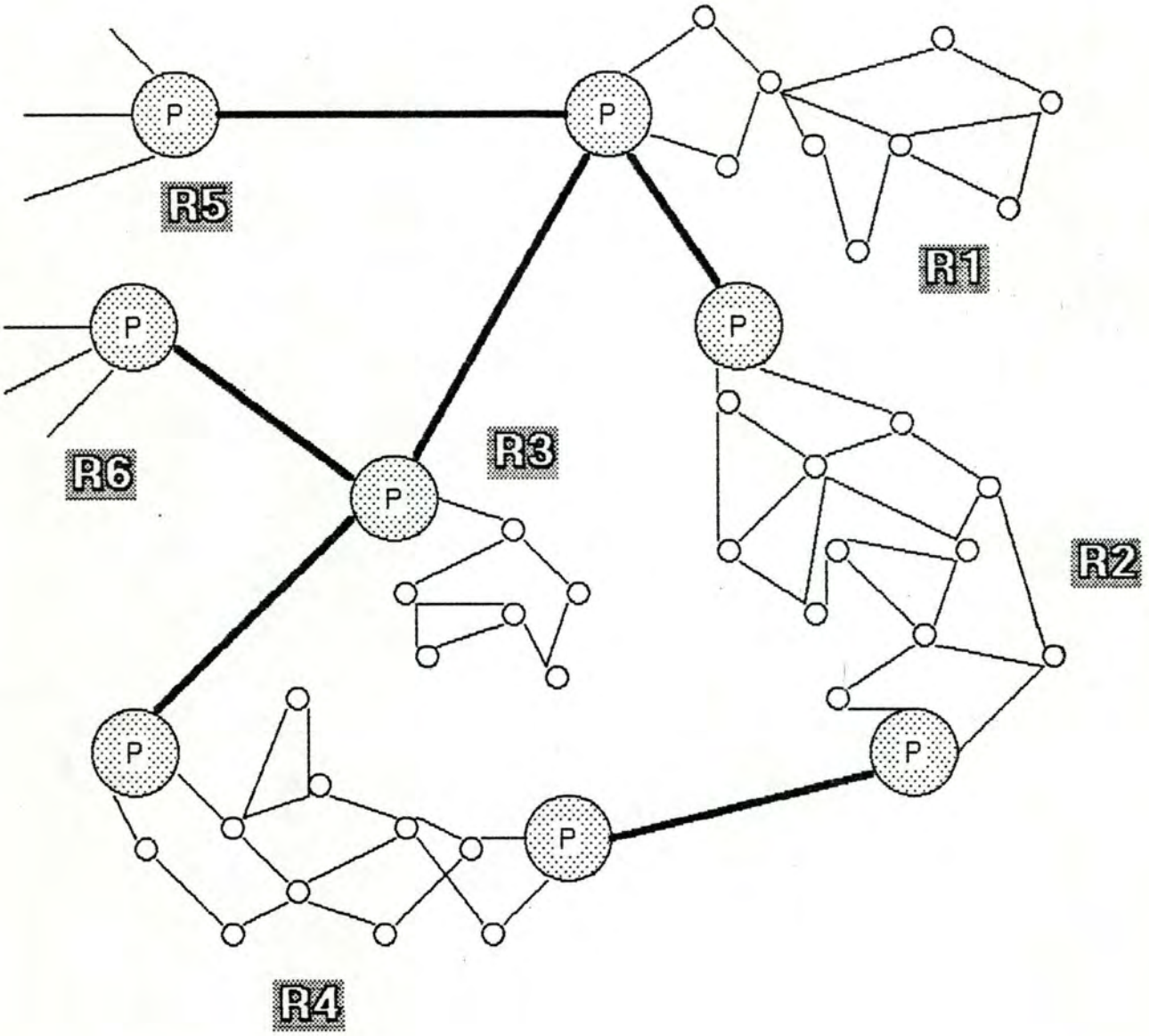
L'objectif d'une telle fonction consiste à permettre à l'utilisateur de réfléchir à son problème en ayant sous les yeux la topologie de son réseau. L'intérêt peut se marquer par exemple lors de la définition du routage

entre les stations, lors de la définition de l'allocation de fichiers, lors de la visualisation des flux d'informations et des capacités allouées aux lignes de transmission (cfr. 5.3.3.2.).

Pour que le concepteur puisse travailler le plus confortablement possible, il doit avoir la possibilité de "dessiner" lui-même la topologie de son réseau, et cela en parallèle avec l'introduction de la description des composantes du réseau.

Il subsiste cependant un problème : la taille du réseau à visualiser. En effet, sur des exemples de grandeur relativement modérée et avec un écran suffisamment grand, il est envisageable de visualiser toute la topologie en un écran. Mais si des problèmes de plus grande ampleur se présentent, il risque de devenir impossible de visualiser l'ensemble avec suffisamment de clarté. Pour pallier à cet inconvénient, une solution est proposée : le partitionnement de la topologie en régions.

Dans cette optique, le concepteur doit essayer de diviser son réseau en plusieurs parties, appelées régions, avec entre elles des connexions via des passerelles (fig. 5.1.).



ou P = passerelle

○ = station

/ = connection entre passerelles

/ = ligne du reseau

RJ = region n° j

fig 5.1 Partitionnement en regions

Le problème est évidemment de diviser le réseau complet en régions de telle sorte que les passerelles soient en nombre pas trop élevé, de même que les connexions entre passerelles. En fait, ce processus peut être assimilé à un problème d'interconnexions de mini-réseaux.

Le concepteur peut alors visualiser soit une région particulière, soit un schéma du réseau des régions (fig. 5.2.).

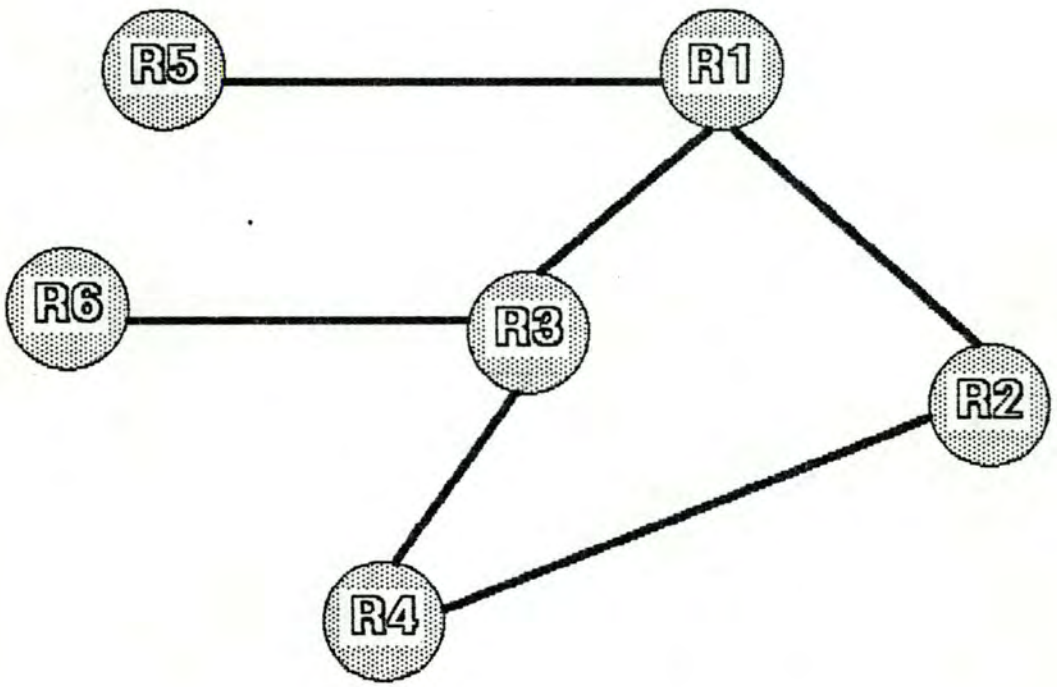


fig 5.2 Reseau des regions

En première analyse, ceci constitue la seule solution envisageable, même si son implémentation et son utilisation ne sont pas évidentes de prime abord.

5.3.3.2. Quelques facilités graphiques

Au niveau de l'interface homme/machine, l'utilisation des outils graphiques peut être vue de deux manières :

modification de l'interface actuel et exploitation de la visualisation du réseau courant.

Dans le logiciel existant, la majorité des interactions s'effectue via des menus à réponses alphanumériques. Grâce aux outils graphiques, et notamment à une "souris", des menus à "clicker" peuvent être proposés.

Par un mécanisme à double fenêtre, il est envisageable de visualiser différents types de données simultanément. Par exemple lors de la fonction "Introduction des lignes", il est possible de visualiser les lignes précédemment introduites ainsi que les stations et noeuds existants, ainsi que des messages d'aide à l'utilisation du logiciel.

Dans le paragraphe 5.3.3.1., la visualisation du réseau à l'écran a été présentée. Il est envisageable d'en tirer parti pour afficher sur le schéma du réseau diverses informations : le routage entre les stations, l'allocation d'un fichier, l'allocation des capacités.

En résumé, l'apport d'outils graphiques peut rendre le logiciel beaucoup plus agréable. Nous pouvons assimiler les présentes considérations à une extension de notre logiciel. En effet, le programmeur des fonctions sus-nommées peut reprendre les grandes idées de notre analyse fonctionnelle ainsi que toutes les fonctions de calcul, ceci limitant son étude à une réflexion sur l'interface, réflexion déjà complexe en soi.

5.4. CONCLUSION

Dans ce chapitre, nous venons d'exposer une réflexion sur une série, non hexaustive, d'améliorations et d'extensions qu'il est possible d'apporter à notre logiciel. Le but de cette analyse est de susciter des recherches en vue de réaliser un logiciel de conception assistée par ordinateur d'une configuration de réseau qui soit à la fois convivial, complet, efficace et apte à supporter des problèmes de taille réelle.

◊

C O N C L U S I O N

Au terme de cette étude, il convient de tirer quelques conclusions.

L'apport principal de notre travail est d'avoir mis en évidence et spécifié les diverses fonctionnalités d'un logiciel de conception assistée par ordinateur d'une configuration de réseau. A partir de ces spécifications, une proposition d'implémentation a été réalisée dans un environnement classique assez répandu, le système UNIX. Bien que partielle, cette implémentation a permis de mettre en lumière un certain nombre d'aspects qu'il serait intéressant d'améliorer ou d'ajouter au logiciel existant, et également de se convaincre qu'il pourrait être agréable de disposer d'outils graphiques pour améliorer l'interface homme/machine.

Notre seul regret est de n'avoir pu, pour des raisons strictement matérielles, terminer l'implémentation de toutes les fonctions décrites dans le chapitre 3, en particulier tout ce qui concerne la modification de la description du réseau, hormis la modification du routage à travers le réseau.
Dans son état actuel, les deux problèmes qui nous occupent peuvent être résolus avec des descriptions de réseau fixées une fois pour toutes, hormis le routage à travers le réseau et l'allocation de fichiers aux stations.

Cependant, sur base de cette réalisation partielle, notre sentiment est que la méthodologie proposée possède un certain avenir, surtout si un interface graphique est utilisé. Une seule crainte est à formuler : dans les cas de réseaux de taille très importante, le concepteur saura-t-il maîtriser toutes les données et garder une bonne vision d'ensemble de son problème ? A cela seule l'expérimentation pourra répondre.

A toute personne qui serait intéressée de poursuivre notre travail, nous pouvons suggérer deux démarches, selon l'environnement dont elle dispose. Si l'environnemenr reste classique, c'est-à-dire sans outil graphique, les étapes suivantes sont à suivre :

a) achever l'implémentation des fonctionnalités décrites dans le chapitre 3

b) améliorer la souplesse de modification des données lors de l'introduction de la description d'un réseau (5.2.2.)

c) introduire la gestion des fichiers de données par le logiciel (5.2.4.)

d) améliorer les messages d'aide à la conception (5.2.3.)

e) élargir l'applicabilité du logiciel (5.3.1.)

Si l'environnement dispose d'outils graphiques, la démarche est la suivante :

a) développer un interface graphique pour implémenter les fonctionnalités du chapitre 3

b) envisager les étapes b), c), d) et e) de la première démarche avec les outils graphiques

c) développer de nouvelles fonctions qui deviennent envisageables avec les outils graphiques. Par exemple, la visualisation simultanée de plusieurs types de données dans diverses fenêtres.

◊

◊ ◊

B I B L I O G R A P H I E

(GEU.85) : P. GEURTS
"Une étude préliminaire pour la réalisation d'un logiciel C.A.O. de configuration de réseau".
Institut d'Informatique FUNDP - Namur 1985.

(KER.78) : B.W. KERNIGHAN et D.M. RITCHIE
"The C programming language".
Prentice-Hall, Inc. Englewood Cliffs, New Jersey 1978.

(KLE.70) : L. KLEINROCK
"Analytic and simulation methods in computer network design".
S.J.C.C. 1970 - PP 569-579.

(PIC.84) : M. PICHOT et J.-C. DETALLE
"Contribution à la conception des systèmes informatiques répartis".
Mémoire de Licence et Maîtrise en Informatique.
Institut d'Informatique FUNDP - Namur 1984.

(TAN.81) : A.S. TANENBAUM
"Computer Networks".
Prentice-Hall, Inc. Englewood Cliffs, New Jersey 1981.

(UNI.79) : "UNIX Time-sharing system : Unix programmer's manual".
7the éd. vol. 1.
Bell laboratories, Inc. Murray Hill, New Jersey 1979.

(VB. 84) : P. VAN BASTELAER
"Les moyens de téléinformatique en Belgique. Leurs possibilités et leurs coûts.
Institut d'Informatique FUNDP - Namur 1984.

*

* *

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX , NAMUR

INSTITUT D'INFORMATIQUE

Année académique 1985-1986

CONCEPTION D'UN LOGICIEL C.A.O. DE CONFIGURATION D'UN SYSTEME INFORMATIQUE DISTRIBUE

*

ANNEXES

Promoteur de mémoire :

Monsieur le Professeur

Ph. VAN BASTELAER

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

A N N E X E S

ANNEXE 1 - LES SPECIFICATIONS DES PROCEDURES IMPLEMENTEES

ANNEXE 2 - MISE EN OEUVRE DU LOGICIEL

ANNEXE 3 - STRUCTURE D'UN FICHIER DE DONNEES

ANNEXE 1

LES SPECIFICATIONS DES PROCEDURES IMPLEMENTEES

L'annexe 1 reprend les spécifications de la majorité des procédures qui composent le logiciel réalisé. En effet, pour des raisons purement matérielles, il n'a pas été possible de rédiger les spécifications des procédures qui réalisent les dernières fonctions implémentées.

Nomenclature de l'annexe 1

Le texte source du logiciel se composant de plusieurs fichiers, cette découpe a été conservée pour la présentation des spécifications des procédures. Cette optique nous semble meilleure à une énumération par ordre alphabétique qui nécessite un va et vient continuel à travers le document pour examiner une fonctionnalité du logiciel.
De plus, de par la structure du logiciel, plusieurs procédures peuvent porter le même nom. En pratique, une procédure est identifiée par le nom du fichier qui la contient et son nom.

L'annexe 1 se décompose en trois parties :

- la liste complète des variables globales du logiciel, ainsi qu'un commentaire sur le rôle de chacune de ces variables.

- la liste des procédures implémentées, regroupées par fichier. Comme il est mentionné au chapitre 4, la majorité des procédures sont simples à comprendre et seuls quelques algorithmes sont explicités en 4.3.3.
  Dans la liste des procédures, chaque procédure dont la logique est détaillée ou apparentée à une logique détaillée, est suivie de la mention "algo".

- les spécifications des procédures, regroupées par fichier. Pour chaque procédure, sa spécification comprend les informations suivantes :

  . la fonction que doit remplir la procédure
  . les arguments en entrée
  . une précondition sur les arguments
  . les résultats en sortie
  . une postcondition sur les résultats.

*

* *

TEXTE SOURCE DU LOGICIEL

Un listing reprenant l'ensemble du texte source du logiciel implémenté, est disponible chez Monsieur le Professeur P. VAN BASTELAER de l'Institut d'Informatique des FUNDP à Namur.

```
/*****************************************************************************
*                                                                            *
*                              FICHIER MEMVAR.C                              *
*                              ================                              *
*                                                                            *
*         Ce fichier contient les variables globales du logiciel             *
*                                                                            *
*****************************************************************************/


#include "memtype.c"


/**********  definition des variables globales du reseau courant  **********/


struct cap alloc_cap [LMAX];  /* allocation des capacites aux lignes
                                                        du reseau */

int allocation_fichier [SMAX] [FMAX];  /* matrice d'enregistrement
                  de l'allocation des fichiers aux stations du reseau
                  allocation_fichier [i] [j] = 1 si le fichier j est
                                                    alloue a la station i
                                          = 0 sinon                    */

int bidir;  /* indique si les lignes du reseau sont unidirectionnelles
                    (  bidir = 0 ) ou bidirectionnelles ( bidir = 1 ) */

struct cap cap_tot [LMAX] [CMAX];  /* cette table contient, pour chaque
                                  ligne, la liste des capacites disponibles,
                                  triees par ordre de cout croissant */

float cout; /* cout de la solution courante */

int cout_min; /* cout_min = 1 si l'allocation de capacite calculee est
                                  de cout minimal  et  0 sinon  */

float delai_reseau;  /* delai moyen de transmission d'un message assure
                                   par le reseau lui-meme ( en secondes ) */

float duree_jour;  /* duree de la periode durant laquelle les matrices de
            trafic propres au jour sont d'application  ( en secondes )  */

float duree_nuit;  /* duree de la periode durant laquelle les matrices de
            trafic propres a la nuit sont d'application ( en secondes)  */

int emplacement [SMAX] [FMAX];  /* matrice des contraintes d'emplacement;
     emplacement [i][j] = 0 si pas de contraintes ,
                        = 1 si un exemplaire du fichier no j doit etre
                              alloue a la station no i ,
                        = 2 si aucun exemplaire du fichier  no j ne peut
                              etre alloue a la station no i .  */

int err [SMAX] [SMAX]; /* table de verification des connections dans
                          le reseau err[i][j]=1 s'il n'existe pas de
                          connection de la station i vers j */

struct sgttyb etat1,etat2; /* variables pour changer le mode du terminal */
```

```
int ex_route_oblig [SMAX] [SMAX]; /* ex_route_oblig[i][j]=1  <=>
                                     le concepteur a force le routage entre
                                     les stations i et j (cfr route) */

float fiabilite [SMAX] [SMAX];  /* fiabilite [i][j] = la fiabilite de la
                          communication entre les stations no i et no j  */

float gamma_jour;  /* valeur totale du flux dans le reseau durant la
                                          journee en regime jour-nuit */

float gamma_nuit;  /* valeur totale du flux dans le reseau durant la
                                            nuit en regime jour-nuit */

float gamma_perm;  /* valeur totale du flux dans le reseau en regime
                                                        permanent */

char input[12];  /*  cette variable contient le nom du fichier ou sont
                     stockees les informations relatives au reseau
                      en cours d'introduction.  */

float jour_lambda [LMAX];  /* jour_lambda [1] = le flux global sur la
                      ligne 1 durant la journee en regime jour-nuit */

int long_segment;  /* longueur d'un segment lors de l'utilisation de
                                             la commutation par paquets */

float mat_dist [SMAX+NMAX] [SMAX+NMAX]; /* matrice des distances entre deux
                                  stations ou noeuds si une ligne les relie */

int mat_ligne [SMAX+NMAX] [SMAX+NMAX]; /* matligne[i][j] : numero de la
                           ligne reliant les noeuds ou stations i et j */

int mod_topo; /* mod_topo = 1 si la topologie a ete modifiee et donc si
                                 le routage est a redefinir
                           = 0 sinon                                  */

float mu_jour [LMAX];  /* inverse de la longueur moyene d'un message
       passant sur chaque ligne, durant la journee en regime jour-nuit */

float mu_nuit [LMAX];  /* inverse de la longueur moyene d'un message
         passant sur chaque ligne, durant la nuit en regime jour-nuit */

float mu_perm [LMAX];  /* inverse de la longueur moyenne d'un message
                          passant sur chaque ligne, en regime permanent */

float mu_q_j;  /* inverse de la longueur moyenne d'un message pour
                     le 'query traffic' pour le jour en regime jour-nuit */

float mu_q_n;  /* inverse de la longueur moyenne d'un message pour
                     le 'query traffic' pour la nuit en regime jour-nuit */

float mu_q_p;  /* inverse de la longueur moyenne d'un message pour
                                     le 'query traffic' en regime permanent */

float mu_rq_j;  /* inverse de la longueur moyenne d'un message pour
            le 'query-return traffic' pour le jour en regime jour-nuit */

float mu_rq_n;  /* inverse de la longueur moyenne d'un message pour
            le 'query-return traffic' pour la nuit en regime jour-nuit */
```

```
float mu_rq_p;  /* inverse de la longueur moyenne d'un message pour
                        le 'query-return traffic' en regime permanent */

float mu_ru_j;  /* inverse de la longueur moyenne d'un message pour
          le 'update-return traffic' pour le jour en regime jour-nuit */

float mu_ru_n;  /* inverse de la longueur moyenne d'un message pour
          le 'update-return traffic' pour la nuit en regime jour-nuit */

float mu_ru_p;  /* inverse de la longueur moyenne d'un message pour
                        le 'return-update traffic' en regime permanent */

float mu_u_j;  /* inverse de la longueur moyenne d'un message pour
                     le 'update traffic' pour le jour en regime jour-nuit */

float mu_u_n;  /* inverse de la longueur moyenne d'un message pour
                     le 'update traffic' pour la nuit en regime jour-nuit */

float mu_u_p;  /* inverse de la longueur moyenne d'un message pour
                                le 'update traffic' en regime permanent */

int nbcap; /* nombre total de capacites existantes */

int nbcaploue; /* nombre de capacites louees existantes */

int nbcapnloue; /* nombre de capacites non louees existantes */

int nbdcsloue;  /* nombre de capacites DCS louees existantes */

int nbdcsnloue; /* nombre de capacites DCS non louees existantes */

int nbf;  /* nombre de fichiers du reseau courant */

int nbl;  /* nombre de lignes du reseau courant */

int nbn;  /* nombre de noeuds du reseau courant */

int nbs;  /* nombre de stations du reseau courant */

int niveau;  /* cette variable indique en permanence le stade auquel le
                concepteur est arrive dans sa session de travail.
                Exemples : niveau = 1 si les stations ont ete introduites
                et niveau = 10 si toutes les donnees ont ete introduites */

float nuit_lambda [LMAX];  /* nuit_lambda [l] = le flux global sur la
                           ligne l durant la nuit en regime jour-nuit */

float q_j_lambda [LMAX];  /* q_j_lambda [l] = flux sur la ligne l
                             provenant du 'query traffic'
                             durant la journee en regime jour-nuit */

float q_n_lambda [LMAX];  /* q_n_lambda [l] = flux sur la ligne l
                             provenant du 'query traffic'
                             durant la nuit en regime jour-nuit */

float q_p_lambda [LMAX];  /* q_p_lambda [l] = flux sur la ligne l
                        provenant du 'query traffic' en regime permanent */
```

```
float perm_lambda [LMAX];  /* perm_lambda [l] = le flux global sur la
                                        ligne l en regime permanent */

float prix_fx_ligne [LMAX] [CLMAX]; /* pour certaines lignes, le cout de la
                              ligne, si elle est louee, est une constante qui
                              depend de la capacite allouee et de la longueur
                              de la ligne ==> utilsation de cette table */

float prix_jour_segment;  /* unite de taxation pour le volume transmis
                                        pour le jour  ( en bits ) */

float prix_nuit_segment;  /* unite de taxation pour le volume transmis
                                        pour la nuit  ( en bits ) */

float prix_segment;  /* unite de taxation pour le volume transmis
                                        ( en bits ) */

int regime;  /* type de regime du trafic : permanent ou jour-nuit */

int route [LMAX] [SMAX] [SMAX]; /* table de routage forcee par le
                                        concepteur */

int routing [LMAX] [SMAX] [SMAX];  /* table de routage effective pour le
                    reseau ; routing[i][j][k]=1 <=> le chemin de j a k
                                        passe par la ligne i */

float rq_j_lambda [LMAX];  /* rq_j_lambda [l] = flux sur la ligne l
                              provenant du 'query-return traffic'
                              durant la journee en regime jour-nuit */

float rq_n_lambda [LMAX];  /* rq_n_lambda [l] = flux sur la ligne l
                              provenant du 'query-return traffic'
                              durant la nuit en regime jour-nuit */

float rq_p_lambda [LMAX];  /* rq_p_lambda [l] = flux sur la ligne l
            provenant du 'query-return traffic' en regime permanent */

float ru_j_lambda [LMAX];  /* ru_j_lambda [l] = flux sur la ligne l
                              provenant du 'update-return traffic'
                              durant la journee en regime jour-nuit */

float ru_n_lambda [LMAX];  /* ru_n_lambda [l] = flux sur la ligne l
                              provenant du 'update-return traffic'
                              durant la nuit en regime jour-nuit */

float ru_p_lambda [LMAX];  /* ru_p_lambda [l] = flux sur la ligne l
              provenant du 'update-return traffic' en regime permanent */

struct dcs_loue tabdcsl [DCSLMAX];  /* table des capacites louees
                           disponibles : cas de la commutation par paquets */

struct dcs_non_loue tabdcsnl [DCSNLMAX];  /* table des capacites non louees
                         disponibles  : cas de la commutation par paquets */

struct fichier tabfich [FMAX]; /* table des fichiers du reseau */

struct ligne tablig [LMAX];  /* table des lignes du reseau */

struct cap_loue tabloue [CLMAX];  /* table des capacites louees disponibles */

struct cap_non_loue tabnloue [CNLMAX];  /* table des capacites non louees
                                        disponibles */
```

```
struct statnoeud tabstat [NMAX+SMAX];  /* table des stations et des noeuds */

float tmax;  /* delai moyen de transmission d'un message : valeur
                                    maximale autorisee */

float tmoy;  /* delai moyen de transmission d'un message : valeur calculee */

float u_j_lambda [LMAX];  /* u_j_lambda [l] = flux sur la ligne l
                             provenant du 'update traffic'
                             durant la journee en regime jour-nuit */

float u_n_lambda [LMAX];  /* u_n_lambda [l] = flux sur la ligne l
                             provenant du 'update traffic'
                             durant la nuit en regime jour-nuit */

float u_p_lambda [LMAX];  /* u_p_lambda [l] = flux sur la ligne l
                        provenant du 'update traffic' en regime permanent */

float u_jour [SMAX] [FMAX];  /* matrice de 'query traffic' pour le jour
                                              en regime jour-nuit  */

float u_nuit [SMAX] [FMAX];  /* matrice de 'query traffic' pour la nuit
                                              en regime jour-nuit  */

float u_perm [SMAX] [FMAX];  /* matrice de 'query traffic' en regime
                                                permanent  */

float uprim_jour [SMAX] [FMAX];  /* matrice de 'query-return traffic'
                                    pour le jour en regime jour-nuit  */

float uprim_nuit [SMAX] [FMAX];  /* matrice de 'query-return traffic'
                                    pour la nuit en regime jour-nuit  */

float uprim_perm [SMAX] [FMAX];  /* matrice de 'query-return traffic'
                                              en regime permanent  */

int util_uniforme;  /* cette variable vaut 1 si les copies des fichiers
        sont utilisees uniformement lors d'un acces, ou 0 si la copie la
        plus proche est utilisee en priorite */

int valide;  /* valide = 1 si l'allocation de fichiers verifie les
                          contraintes d'emplacement et de coherence
                    = 0 sinon                                          */

float v_jour [SMAX] [FMAX];  /* matrice de 'update traffic' pour le jour
                                              en regime jour-nuit  */

float v_nuit [SMAX] [FMAX];  /* matrice de 'update traffic' pour la nuit
                                              en regime jour-nuit  */

float v_perm [SMAX] [FMAX];  /* matrice de 'update traffic' en regime
                                                permanent  */

float vprim_jour [SMAX] [FMAX];  /* matrice de 'update-return traffic'
                                    pour le jour en regime jour-nuit  */

float vprim_nuit [SMAX] [FMAX];  /* matrice de 'update-return traffic'
                                    pour la nuit en regime jour-nuit  */

float vprim_perm [SMAX] [FMAX];  /* matrice de 'update-return traffic'
                                              en regime permanent  */
```

LISTE DES PROCEDURES SPECIFIEES

```
/*****************************************************************************
*                                                                           *
*                          FICHIER MAITRE.C                                 *
*                          ================                                 *
*                                                                           *
*    Ce fichier contient le  programme coordinateur  d'un logiciel de       *
*    conception assitee par ordinateur d'une configuration de reseau.       *
*                                                                           *
*    Il comporte les procedures suivantes :  main()                         *
*                                            entree()                       *
*                                            lec1_input()                   *
*                                            lec2_input()                   *
*                                            introduction()      ALGO       *
*                                            val1()                         *
*                                            val2(i)                        *
*                                            val3()                         *
*                                            cont_intro()                   *
*                                            recherche()                    *
*                                            rech_appel(ch)                 *
*                                            fais_capac()                   *
*                                            modification()                 *
*                                            mod_appel(ch)                  *
*                                            station()                      *
*                                            noeud()                        *
*                                            ligne()                        *
*                                            route()                        *
*                                            fichier()                      *
*                                            capacite()                     *
*                                            parametre()                    *
*                                            trafic()                       *
*                                            fiab()                         *
*                                            emplac()                       *
*                                            alloc()                        *
*                                            reconstitution()               *
*                                            sauvetage()                    *
*                                            visualisation()                *
*                                            cont_stock()                   *
*                                            cont_dispo()                   *
*                                            cont_flux()                    *
*                                            cont_capac()                   *
*                                            cont_delai()                   *
*                                            calc_capac()                   *
*                                            mod_station()                  *
*                                            mod_noeud()                    *
*                                            mod_ligne()                    *
*                                            mod_route()                    *
*                                            mod_fichier()                  *
*                                            mod_capac()                    *
*                                            mod_para()                     *
*                                            mod_trafic()                   *
*                                            mod_fiabilite()                *
*                                            mod_emplacement()              *
*                                                                           *
*****************************************************************************/
```

```
/*****************************************************************************
*                                                                            *
*                          FICHIER  MEMALLOC.C                               *
*                         =====================                              *
*                                                                            *
*   Ce fichier contient les procedures necessaires a l'introduction de       *
*   l'allocation des fichiers aux stations du reseau.                        *
*                                                                            *
*  Il comporte les procedures suivantes : main(argc,argv)                    *
*                                         alloc()                            *
*                                         init_alloc()                       *
*                                         ecran1()                           *
*                                         all_stat()            ALGO         *
*                                         oblig_alloc()                      *
*                                         lec_fich(stat,line,col,nom)        *
*                                         lec2(line,col)                     *
*                                         all_valide()                       *
*                                         visu_pb(pr)                        *
*                                         resol_pb(pr)                       *
*                                         resol2_pb(fich,line,col,nom)       *
*                                         ecran2()                           *
*                                         all_fich()            ALGO         *
*                                         lec_stat(fich,line,col,nom)        *
*                                         visu_alloc()                       *
*                                         visu2_alloc()                      *
*                                         visu3_alloc()                      *
*                                         visu4_alloc()                      *
*                                         redef_alloc()                      *
*                                         clear_alloc()                      *
*                                         mod_alloc()                        *
*                                         m_stat_alloc()                     *
*                                         m2_stat_alloc(i)                   *
*                                         aj_fich(i)                         *
*                                         an_fich(i)                         *
*                                         mod_ecran(i)                       *
*                                         m_fich_alloc()                     *
*                                         m2_fich_alloc(i)                   *
*                                         aj_stat(i)                         *
*                                         an_stat(i)                         *
*                                         mod2_ecran(i)                      *
*                                         ok_an(i)                           *
*                                         quest_alloc()                      *
*                                                                            *
*****************************************************************************/
```

```
/*****************************************************************************
*                                                                            *
*                          FICHIER MEMCAPAC.C                                *
*                          ==================                                *
*                                                                            *
*  Ce fichier contient les procedures necessaires a l'introduction des       *
*  capacites disponibles dans le reseau et leur visualisation.               *
*                                                                            *
*  Il comporte les procedures suivantes :  main(argc,argv)                   *
*                                          ent_capac()                       *
*                                          capacite()                        *
*                                          intro_capac()                     *
*                                          titre_capac()                     *
*                                          titre1_lec()                      *
*                                          titre2_lec()                      *
*                                          titre3_lec()                      *
*                                          titre4_lec()                      *
*                                          lec1_capac()                      *
*                                          lec2_capac()                      *
*                                          lec3_capac()                      *
*                                          lec4_capac()                      *
*                                          mod1_capac()        ALGO          *
*                                          mod2_capac()        ALGO          *
*                                          mod3_capac()        ALGO          *
*                                          mod4_capac()        ALGO          *
*                                          const_capac()                     *
*                                          quest_capac()                     *
*                                          sauve-cap()                       *
*                                          mod_const_cap()                   *
*                                          mod2_const_cap()    ALGO          *
*                                                                            *
*****************************************************************************/


/*****************************************************************************
*                                                                            *
*                          FICHIER MEMCONTR.C                                *
*                          ==================                                *
*                                                                            *
*  Ce fichier contient les procedures necessaires a l'introduction des       *
*  contraintes d'emplacement sur les fichiers et leur visualisation.         *
*                                                                            *
*  Il comporte les procedures suivantes :  contr_emplacement()               *
*                                          init_contrainte()                 *
*                                          int_contrainte()                  *
*                                          l1_contrainte()     ALGO          *
*                                          l2_contrainte()     ALGO          *
*                                          l3_contrainte()                   *
*                                          l4_contrainte()                   *
*                                          mod-contrainte()                  *
*                                          m1_contrainte()                   *
*                                          m11_contrainte(i)                 *
*                                          m2_contrainte()                   *
*                                          m21_contrainte(i)                 *
*                                          m3_contrainte()                   *
*                                          ch_contr(i,j)                     *
*                                          que_contr()                       *
*                                                                            *
*****************************************************************************/
```

```
/*****************************************************************************
*                                                                            *
*                              FICHIER MEMEX.C                               *
*                              ===============                               *
*                                                                            *
*  Ce fichier contient des procedures d'utilite generale pour la majorite    *
*  des fonctions fondamentales du programme                                  *
*                                                                            *
*  Il contient les procedures suivantes :  ex_stat(nom,n)                    *
*                                          ex_noeud(nom,n)                   *
*                                          ex_lig(no1,no2,n)                 *
*                                          ex_fich(nom,n)                    *
*                                          ex_lcap(val)                      *
*                                          ex_nlcap(val)                     *
*                                          ex_ldcs(val)                      *
*                                          ex_nldcs(val)                     *
*                                          num_statnoeud(no)                 *
*                                          num_fich(no)                      *
*                                          num_ligne(no1,no2)                *
*                                                                            *
*****************************************************************************/


/*****************************************************************************
*                                                                            *
*                             FICHIER MEMFIAB.C                              *
*                             =================                              *
*                                                                            *
*   Ce fichier contient les procedures necessaires a l'introduction des      *
*   contraintes de fiabilite du reseau et leur visualisation.                *
*                                                                            *
*   Il comporte les procedures suivantes :  main(argc,argv)                  *
*                                           c_fiabilite()                    *
*                                           int_fiabilite()                  *
*                                           mod_fiabilite()      ALGO        *
*                                           que_fiabilite()                  *
*                                                                            *
*****************************************************************************/


/*****************************************************************************
*                                                                            *
*                            FICHIER MEMFICHIER.C                            *
*                            ====================                            *
*                                                                            *
*  Ce fichier contient les procedures necessaires a l'introduction des       *
*  fichiers presents dans le reseau et leur visualisation.                   *
*                                                                            *
*  Il comporte les procedures suivantes :  main(argc,argv)                   *
*                                          int_fich()                        *
*                                          fichier()                         *
*                                          intro_fich()        ALGO          *
*                                          ecran_fich()                      *
*                                          visu_fich()                       *
*                                          mod_fich_intro()    ALGO          *
*                                          q_modif()                         *
*                                                                            *
*****************************************************************************/
```

```
/*****************************************************************************
*                                                                            *
*                          FICHIER MEMFISTOCK.C                              *
*                          ====================                              *
*                                                                            *
*     Ce fichier contient les proscedures necessaires au sauvetage du        *
*     reseau courant sur un autre fichier de donnees.                        *
*                                                                            *
*     Il comporte les procedures suivantes : main(argc,argv)                 *
*                                            lire_output(output)             *
*                                                                            *
*****************************************************************************/


/*****************************************************************************
*                                                                            *
*                          FICHIER  MEMLIGNE.C                               *
*                          ===================                               *
*                                                                            *
*  Ce fichier contient les procedures necessaires a l'introduction des       *
*  lignes du reseau a etudier et leur visualisation.                         *
*                                                                            *
*  Il comporte les procedures suivantes : main(argc,argv)                    *
*                                         intro_ligne()                      *
*                                         ligne()                            *
*                                         uni_ou_bi()                        *
*                                         lec_noms_stat(j,l,no1,no2()        *
*                                         lec_uni()          ALGO            *
*                                         ecran_lig()                        *
*                                         visu_uni()                         *
*                                         mod_uni()          ALGO            *
*                                         qunimod()                          *
*                                         lec_bi()           ALGO            *
*                                         visu_bi()                          *
*                                         mod_bi()           ALGO            *
*                                         qbimod()                           *
*                                         table_lig()                        *
*                                                                            *
*****************************************************************************/


/*****************************************************************************
*                                                                            *
*                          FICHIER  MEMNOEUD.C                               *
*                          ===================                               *
*                                                                            *
*  Ce fichier contient les procedures necessaires a l'introduction des       *
*  noeuds du reseau a etudier et leur visualisation.                         *
*                                                                            *
*  Il comporte les procedures suivantes : main(argc,argv)                    *
*                                         noeud()                            *
*                                         intro_noeud()        ALGO          *
*                                         ecran_noeud()                      *
*                                         visu_noeud()                       *
*                                         mod_noeud_intro()    ALGO          *
*                                         ques_modif()                       *
*                                                                            *
*****************************************************************************/
```

```
/*****************************************************************************
*                                                                            *
*                              FICHIER MEMPARA.C                             *
*                              =================                             *
*                                                                            *
*  Ce fichier contient les procedures necessaires a l'introduction des       *
*  parametres du reseau et leur visualisation.                               *
*                                                                            *
*  Il comporte les procedures suivantes :  main(argc,argv)                   *
*                                          parametre()                       *
*                                          quest_para_modif()                *
*                                          intro_option()                    *
*                                          ecran_option()                    *
*                                          intro2_option(reg)                *
*                                          intro3_option()                   *
*                                          mod_option()                      *
*                                          mod_perm_option()                 *
*                                          mod_jn_option()                   *
*                                          mod_util()                        *
*                                          mod_delai()                       *
*                                          mod_long()                        *
*                                          mod1_tmax()                       *
*                                          mod1_prix()                       *
*                                          mod2_tmax()                       *
*                                          mod2_prix()                       *
*                                          mod3_prix()                       *
*                                          mod_jour()                        *
*                                          mod_nuit()                        *
*                                          mod1_regime(ch)      ALGO         *
*                                          mod2_regime(ch)      ALGO         *
*                                                                            *
*****************************************************************************/
```

```
/*****************************************************************************
*                                                                            *
*                          FICHIER MEMROUTE.C                                *
*                          ==================                                *
*                                                                            *
*   Ce fichier contient les procedures necessaires a la determination        *
*   du routage dans le reseau par calcul et/ou par introduction a l'ecran    *
*   par le concepteur.                                                       *
*                                                                            *
*   Il comporte les procedures suivantes : main(argc,argv)                   *
*                                          ent_route()              ALGO     *
*                                          ent2_route()                      *
*                                          visu_uni()                        *
*                                          visu_bi()                         *
*                                          ecran_ligne()                     *
*                                          visu_err()                        *
*                                          rech_hamilton()                   *
*                                          tout_ens(ens,n)                   *
*                                          enr_table_routage()      ALGO     *
*                                          lire_origine(origine,n)           *
*                                          lire_destination(dest,orig,n)     *
*                                          visu_route(r,pos,n1,n2)  ALGO     *
*                                          intro_route(orig,dest,ok)         *
*                                          lire_suiv(suiv,deja,rt,prc,pos)   *
*                                          ajout_ligne()                     *
*                                          ajout_uni()                       *
*                                          cont_aj(max)                      *
*                                          ajout_bi()                        *
*                                          lec_stat(no1,no2)                 *
*                                          table_ligne()                     *
*                                                                            *
*****************************************************************************/
```

```
/*****************************************************************************
*                                                                            *
*                          FICHIER MEMSAUVE.C                                *
*                          ==================                                *
*                                                                            *
*  Ce fichier contient les procedures necessaires a l'enregistrement et      *
*  la lecture sur fichier du reseau introduit a l'ecran .                    *
*                                                                            *
*  Il comporte les procedures suivantes :  ecrire_reseau(nom)                *
*                                          ecr_stat(fich)                    *
*                                          ecr_noeud(fich)                   *
*                                          ecr_ligne(fich)                   *
*                                          ecr_route(fich)                   *
*                                          ecr_fichier(fich)                 *
*                                          ecr_capac(fich)                   *
*                                          ecr_lcap(fich)                    *
*                                          ecr_nlcap(fich)                   *
*                                          ecr-ldcs(fich)                    *
*                                          ecr_nldsc(fich)                   *
*                                          ecr_cf(fich)                      *
*                                          ecr_para(fich)                    *
*                                          ecr_trafic(fich)                  *
*                                          ecr2_trafic(fich)                 *
*                                          ecr3-trafic(fich)                 *
*                                          ecr_fiabilite(fich)               *
*                                          ecr_emplacement(fich)             *
*                                          ecr_alloc(fich)                   *
*                                          lire_reseau(nom)                  *
*                                          lire_stat(fich)                   *
*                                          lire_noeud(fich)                  *
*                                          lire_ligne(fich)                  *
*                                          lire_route(fich)                  *
*                                          lire_fichier(fich)                *
*                                          lire_capac(fich)                  *
*                                          lire_lcap(fich)                   *
*                                          lire_nlcap(fich)                  *
*                                          lire_ldsc(fich)                   *
*                                          lir_nldcs(fich)                   *
*                                          lire_cf(fich)                     *
*                                          lire_para(fich)                   *
*                                          lire_trafic(fich)                 *
*                                          lire2_trafic(fich)                *
*                                          lire3_trafic(fich)                *
*                                          lir_fiabilite(fich)               *
*                                          lire_emplacement(fich)            *
*                                          lire_alloc(fich)                  *
*                                                                            *
*****************************************************************************/
```

```
/*****************************************************************************
*                                                                            *
*                          FICHIER  MEMSTAT.C                                *
*                          ==================                                *
*                                                                            *
*  Ce fichier contient les procedures necessaires a l'introduction des       *
*  stations du reseau a etudier et leur visualisation.                       *
*                                                                            *
*  Il comporte les procedures suivantes : main(argc,argv)                    *
*                                         ent_stat()                         *
*                                         station()              ALGO        *
*                                         intro_stat()                       *
*                                         ecran_stat()                       *
*                                         visu_stat()                        *
*                                         mod_stat_intro()       ALGO        *
*                                         quest_modif()                      *
*                                                                            *
*****************************************************************************/
```

```
/*****************************************************************************
*                                                                            *
*                          FICHIER  MEMSTOCK.C                               *
*                          ===================                               *
*                                                                            *
*   Ce fichier contient les procedures necessaires a la verification des     *
*   contraintes de stockage.                                                 *
*                                                                            *
*  Il comporte les procedures suivantes : main(argc,argv)                    *
*                                         stock()                            *
*                                         sfaisabilite(pb)                   *
*                                         visu_pb(pb)                        *
*                                         quest1_modif()                     *
*                                         quest2_modif()                     *
*                                         detail_pb(pb)                      *
*                                         mod_alloc()                        *
*                                         m_stat_alloc()                     *
*                                         m2_stat_alloc(i)                   *
*                                         aj_fich(i)                         *
*                                         an_fich(i)                         *
*                                         mod_ecran(i)                       *
*                                         m_fich_alloc()                     *
*                                         m2_fich_alloc(i)                   *
*                                         aj_stat(i)                         *
*                                         an_stat(i)                         *
*                                         mod2_ecran(i)                      *
*                                         ok_an(i)                           *
*                                         quest_alloc()                      *
*                                                                            *
*****************************************************************************/
```

```
/*****************************************************************************
*                                                                            *
*                              FICHIER MEMTRAF.C                             *
*                              =================                             *
*                                                                            *
*   Ce fichier contient les procedures necessaires a l'introduction du       *
*   trafic dans le reseau et leur visualisation.                             *
*                                                                            *
*   Il comporte les procedures suivantes :  main(argc,argv)                  *
*                                           trafic()                         *
*                                           intro_trafic()                   *
*                                           mod_trafic()                     *
*                                           ecran1_trafic()                  *
*                                           lec_p_long()                     *
*                                           lec_val(line,col)                *
*                                           l_jn_long()                      *
*                                           mod_p_long()                     *
*                                           m1_p_long()                      *
*                                           m2_p_long()                      *
*                                           m3_p_long()                      *
*                                           m4_p_long()                      *
*                                           m_jn_long()                      *
*                                           m1_jn_long()                     *
*                                           m2_jn_long()                     *
*                                           m3_jn_long()                     *
*                                           m4_jn_long()                     *
*                                           m5_jn_long()                     *
*                                           m6_jn_long()                     *
*                                           m7_jn_long()                     *
*                                           m8_jn_long()                     *
*                                           ecran2_trafic()                  *
*                                           lec_p_traf()                     *
*                                           ecran3_trafic()                  *
*                                           l_jn_traf()                      *
*                                           mod_p_traf()          ALGO       *
*                                           mod2_p_traf(i,j)                 *
*                                           m1_p_traf(i,j)                   *
*                                           m2_p_traf(i,j)                   *
*                                           m3_p_traf(i,j)                   *
*                                           m4_p_traf(i,j)                   *
*                                           m_jn_traf()           ALGO       *
*                                           m2_jn_traf()                     *
*                                           m21_jn_traf(i,j)                 *
*                                           m22_jn_traf(i,j)                 *
*                                           m23_jn_traf(i,j)                 *
*                                           m24_jn_traf(i,j)                 *
*                                           m25_jn_traf(i,j)                 *
*                                           m26_jn_traf(i,j)                 *
*                                           m27_jn_traf(i,j)                 *
*                                           m28_jn_traf(i,j)                 *
*                                           quest_traf_mod()                 *
*                                                                            *
*****************************************************************************/
```

```
/*****************************************************************************/
/*                                                                           */
/*                          FICHIER TECH.C                                   */
/*                          ==============                                   */
/*                                                                           */
/*     PROCEDURES DE GESTION D'ECRAN  :  set_pos(line,col)                   */
/*                                       clear(x,y,lim)                      */
/*                                       scr_blc()                           */
/*                                       getline(s,lim)                      */
/*                                       getfich(s,lim)                      */
/*                                       getdigit(s,lim)                     */
/*                                       getint(s,lim)                       */
/*                                       getproba(s,lim)                     */
/*                                       q_ou_c(p,n)                         */
/*                                       m_q_c(c)                            */
/*                                       init_vt100()                        */
/*                                       retab_vt100()                       */
/*                                       ret()                               */
/*                                                                           */
/*****************************************************************************/
```

FICHIER MAITRE . C

Ce fichier contient le programme coordinateur d'un logiciel de conception assistée par ordinateur d'une configuration de réseau.

main( )

Fonction = entrée au programme coordinateur.

entree( )

Fonction = coordinateur de la saisie au terminal du nom du fichier de données sur lequel le logiciel va travailler.

Résultat = retour de la fonction : entier.
Postcondition = le retour vaut 0 si aucun nom de fichier n'a été introduit et 1 sinon.

lecl_input( )

Fonction = saisie au terminal du nom d'un fichier de données contenant une description partielle ou totale d'un réseau.

Résultats = retour de la fonction : entier.
input : chaîne de caractères.
Postcondition = . input contient le nom d'un fichier existant ou le caractère '*' d'annulation.
. le retour égale 1 si le nom d'un fichier existant a été saisi et 0 sinon. Dans ce cas, input ne peut être pris en compte.

lec2_input( )

Fonction = saisie au terminal du nom d'un fichier inexistant ou qui existe mais peut être détruit.

Résultats = input : chaîne de caractères.
retour de la fonction : entier.

Postcondition = . le retour vaut 1 si la variable input contient le nom d'un fichier inexistant ou pouvant être considéré comme tel et 0 sinon.
. input contient le nom d'un fichier ou le caractère '*' d'annulation.

introduction( )

Fonction = coordinateur des diverses fonctions réalisant l'introduction des données décrivant un réseau, avec possibilité d'interruption entre les fonctions et de reprise ultérieure.

Argument = input : chaîne de caractères.

Précondition = input contient le nom d'un fichier de données existant ou non.

val1( )

Fonction = contrôle de l'existence d'un fichier.

Argument = input : chaîne de caractères.

Résultat = retour de la fonction : entier.

Postcondition = le retour vaut 1 si le fichier de nom input existe et 0 sinon.

val2(i)

Fonction = contrôle du niveau d'introduction renseigné par le fichier de données : égalité à une valeur i.

Arguments = input : chaîne de caractères.
i : entier.
Précondition = input contient le nom d'un fichier de données existant.

Résultat = retour de la fonction : entier.
Postcondition = le retour vaut 1 si le niveau renseigné par le fichier de données de nom input est égal à i et 0 sinon.

val3( )

Fonction = contrôle du niveau d'introduction renseigné par le fichier de données : supérieur à dix, c'est-à-dire que toute la description du réseau de données a été introduite.

Argument = input : chaîne de caractères.
Précondition = input contient le nom d'un fichier de données existant.

Résultat = retour de la fonction : entier.
Postcondition = le retour vaut 1 si le niveau renseigné par le fichier de données de nom input est supérieur ou égal à dix et 0 sinon.

cont_intro( )

Fonction = proposition d'interruption de la session de travail, lors de la succession des activations des fonctions d'introduction et saisie du choix.

Argument = input : chaîne de caractères.
Précondition = input contient le nom des fichiers de données courant.

Résultat = retour de la fonction : entier.
Postcondition = le retour vaut 1 si l'utilisateur désire continuer la session de travail et 0 sinon.

recherche( )

Fonction = coordinateur de type menu, des diverses fonctions relatives au module "Recherche d'une solution".

Résultat = retour de la fonction : entier.
Postcondition = le retour vaut 1 si, au sortir de recherche(), il faut activer la procédure introduction( ) pour le fichier courant et 0 pour sortir du logiciel.

rech_appel(ch)

Fonction = activation d'une des diverses fonctions du module "Recherche d'une solution".

Argument = ch : entier.
Précondition = ch contient le numéro de la fonction à activer (cfr listing).

Résultat = retour de la fonction : entier.
Postcondition = le retour vaut 1 s'il faut sortir de recherche() (au niveau supérieur) pour activer la procédure introduction() sur le fichier courant et 0 sinon.

fais_capac( )

Fonction = coordinateur des diverses fonctions réalisant les contrôles de faisabilité et la détermination de l'allocation de capacités, avec rupture en cas de problème dans les contrôles de faisabilité.

Argument = input : chaîne de caractères.
Précondition = input contient le nom d'un fichier de données.

modification( )

Fonction = coordination de type menu, des diverses fonctions de modification de la description du réseau courant.

mod_appel(ch)

Fonction = activation d'une des diverses fonctions de modification de la description du réseau courant.

Argument = ch : entier.
Précondition = ch contient le numéro de la fonction à activer (cfr listing).

station( )

Fonction = création d'un processus qui a pour mission de lancer l'exécution d'un programme réalisant la fonction d'introduction des stations du réseau.

Argument = input : chaîne de caractères.
Précondition = input contient le nom d'un fichier inexistant.

noeud( )

Fonction = création d'un processus qui a pour mission de lancer l'exécution d'un programme réalisant la fonction d'introduction des noeuds du réseau.

Argument = input : chaîne de caractères.
Précondition = input contient le nom d'un fichier de données de niveau égal à 1.

ligne( )

Fonction = création d'un processus qui a pour mission de lancer l'exécution d'un programme réalisant la fonction d'introduction des lignes du réseau.

Argument = input : chaîne de caractères.
Précondition = input contient le nom d'un fichier de données de niveau égal à 2.

routage( )

Fonction = création d'un processus qui a pour mission de lancer l'exécution d'un programme réalisant la fonction d'introduction du routage à travers le réseau.

Argument = input : chaîne de caractères.
Précondition = input contient le nom d'un fichier de données de niveau égal à 3.

fichier( )

Fonction = création d'un processus qui a pour mission de lancer l'exécution d'un programme réalisant la fonction d'introduction des fichiers à distribuer.

Argument = input : chaîne de caractères.
Précondition = input contient le nom d'un fichier de données de niveau égal à 4.

capacite( )

Fonction = création d'un processus qui a pour mission de lancer l'exécution d'un programme réalisant la fonction d'introduction des capacités disponibles.

Argument = input : chaîne de caractères.
Précondition = input contient le nom d'un fichier de données de niveau égal à 5.

paramètre( )

Fonction = création d'un processus qui a pour mission de lancer l'exécution d'un programme réalisant la fonction d'introduction des paramètres d'utilisation du réseau.

Argument = input : chaîne de caractères.
Précondition = input contient le nom d'un fichier de données de niveau égal à 6.

trafic( )

Fonction = création d'un processus qui a pour mission de lancer l'exécution d'un programme réalisant la fonction d'introduction du trafic à travers le réseau.

Argument = input : chaîne de caractères.
Précondition = input contient le nom d'un fichier de données de niveau égal à 7.

fiab( )

Fonction = création d'un processus qui a pour mission de lancer l'exécution d'un programme réalisant la fonction d'introduction de la fiabilité du réseau.

Argument = input : chaîne de caractères.
Précondition = input contient le nom d'un fichier de données de niveau égal à 8.

emplac( )

Fonction = création d'un processus qui a pour mission de lancer l'exécution d'un programme réalisant la fonction d'introduction des contraintes d'emplacement.

Argument = input : chaîne de caractères.
Précondition = input contient le nom d'un fichier de données de niveau égal à 9.

alloc( )

Fonction = création d'un processus qui a pour mission de lancer l'exécution d'un programme réalisant la fonction de définition de l'allocation de fichiers.

Argument = input : chaîne de caractères.
Précondition = input contient le nom d'un fichier de données de niveau supérieur ou égal à 10.

reconstitution( )

Fonction = saisie au terminal du nom d'un fichier de données pour qu'il devienne le fichier courant du logiciel.

Résultat = input : chaîne de caractères.
retour de la fonction : entier.

Postcondition = . input contient le nom d'un fichier de données.
. le retour vaut 1 si le nouveau fichier de données courant est tel que sa variable niveau est inférieure ou égale à 9.

sauvetage( )

Fonction = création d'un processus qui a pour mission de lancer l'exécution d'un programme réalisant la fonction de sauvetage sur fichier du réseau courant.

Argument = input : chaîne de caractères.

Précondition = input contient le nom d'un fichier de données de niveau supérieur ou égal à 10.

visualisation( )

Fonction = création d'un processus qui a pour mission de lancer l'exécution d'un programme réalisant la fonction de visualisation de la solution courante.

Argument = input : chaîne de caractères.

Précondition = input contient le nom d'un fichier de données dont le niveau est supérieur à 10.

cont_stock( )

Fonction = création d'un processus qui a pour mission de lancer l'exécution d'un programme réalisant la fonction de contrôle des contraintes de stockage.

Argument = input : chaîne de caractères.
Précondition = input contient le nom d'un fichier de données dont le niveau est égal à 11.

cont_dispo( )

Fonction = création d'un processus qui a pour mission de lancer l'exécution d'un programme réalisant la fonction de contrôle des contraintes de disponibilité.

Argument = input : chaîne de caractères.
Précondition = input contient le nom d'un fichier de données dont le niveau est égal à 12.

cont_flux( )

Fonction = création d'un processus qui a pour mission de lancer l'exécution d'un programme réalisant la fonction de contrôle des contraintes de flux.

Argument = input : chaîne de caractères.
Précondition = input contient le nom d'un fichier de données dont le niveau est égal à 13.

cont_capac( )

Fonction = création d'un processus qui a pour mission de lancer l'exécution d'un programme réalisant la fonction de contrôle des contraintes de capacité.

Argument = input : chaîne de caractères.
Précondition = input contient le nom d'un fichier de données dont le niveau est égal à 14.

cont_délai( )

Fonction = création d'un processus qui a pour mission de lancer l'exécution d'un programme réalisant la fonction de contrôle de la contrainte de délai.

Argument = input : chaîne de caractères.
Précondition = input contient le nom d'un fichier de données dont le niveau est égal à 14.

calc_capac( )

Fonction = création d'un processus qui a pour mission de lancer l'exécution d'un programme réalisant la fonction de calcul de l'allocation de capacités courante et des coûts et performances.

Argument = input : chaîne de caractères.
Précondition = input contient le nom d'un fichier de données dont le niveau est égal à 16.

mod_station( )

Fonction = création d'un processus qui a pour mission de lancer l'exécution d'un programme réalisant la fonction de modification des stations du réseau.

Argument = input : chaîne de caractères.
Précondition = input contient le nom d'un fichier de données dont le niveau est supérieur ou égal à 10.

mod_noeud( )

Fonction similaire à mod-station() mais pour les noeuds du réseau.

mod_ligne( )

Fonction similaire à mod-station() mais pour les lignes du réseau.

mod_route( )

Fonction similaire à mod-station() mais pour le routage à travers le réseau.

mod_fichier( )

Fonction similaire à mod-station() mais pour les fichiers à distribuer.

mod-capac( )

Fonction similaire à mod-station() mais pour les capacités disponibles.

mod_para( )

Fonction similaire à mod-station() mais pour les paramètres d'utilisation du réseau.

mod_trafic( )

Fonction similaire à mod-station() mais pour le trafic à travers le réseau.

mod_fiabilite( )

Fonction similaire à mod-station() mais pour la fiabilité du réseau.

mod_emplacement( )

Fonction similaire à mod-station() mais pour les contraintes d'emplacement.

FICHIER MEMALLOC . C

Ce fichier contient les procédures nécessaires à l'introduction de l'allocation des fichiers aux stations du réseau.

main(argc,argv)

Fonction = entrée au sous-programme réalisant l'introduction de l'allocation des fichiers.

Arguments = argc : entier.
argv : tableau de pointeurs.
Précondition = argv contient des paramètres : le niveau atteint (10) et le nom du fichier où il faut stocker les données introduites.

alloc( )

Fonction = coordinateur de la logique d'introduction

Argument = niveau : entier.
Précondition = 10 < = niveau.

Résultat = retour de la fonction : entier.
Postcondition = le retour égale 1 ou 2 si une allocation de fichiers a été définie et 0 sinon.

init_alloc( )

Fonction = proposition de deux techniques d'introduction de l'allocation des fichiers et activation du module ad hoc : par énumération des stations et par énumération des fichiers.

Résultat = retour de la fonction : entier.
Postcondition = le retour égale 1 ou 2 si une allocation de fichier a été définie et 0 sinon.

ecran1( )

Fonction = affichage d'un texte à l'écran (cfr listing).

all_stat( )

Fonction = pour chaque station, visualisation des allocations de fichiers déclarées obligatoires par des contraintes d'emplacement et saisie au terminal des autres fichiers à allouer à cette station.

Arguments = tabstat : tableau de structures statnoeuds.
tabfich : tableau de structure fichier.
emplacement : tableau entier.
nbs, nbf : entier.
Précondition = . 0 < = nbs, nbf.
. tabstat contient nbs stations distinctes 2 à 2.
. tabfich contient nbf fichiers distincts 2 à 2.
. pour tout i et j, emplacement (i)(j) = 0, 1 ou 2 selon que le fichier j est soumis, à la station i, à aucune contrainte, à une contrainte de présence obligatoire ou de présence interdite.

Résultat = allocation-fichier : tableau entier.
Postcondition = pour tout i et j, allocation-fichier (i)(j) = 1 si le fichier j est alloué à la station i et 0 sinon.

oblig_alloc( )

Fonction = initialisation de l'allocation de fichiers en allouant les fichiers soumis à une contrainte de présence obligatoire.

Arguments = emplacement : tableau entier.
nbs, nbf : entier.

Précondition = pour tout i et j, emplacement (i)(j) = 0, 1 ou 2 selon que le fichier j est soumis, à la station i, à aucune contrainte, à une contrainte de présence obligatoire ou de présence interdite.

Résultat = allocation-fichier : tableau entier.
Postcondition = pour tout i et j, allocation-fichier (i)(j) = 1 si emplacement (i)(j) = 1 et 0 sinon.

lec_fich(stat,line,col,nom)

Fonction = saisie au terminal du nom d'un fichier devant être alloué à la station n° stat avec contrôle de cohérence : existence du fichier, allocation autorisée et non déjà spécifiée.

Arguments = stat : entier.
line : entier.
col : entier.
emplacement : tableau entier.
allocation-fichier : tableau entier.
nbs, nbf : entier.

Précondition = . 1 < = stat < = nbs.
. 0 < = nbf
. 1 < = line < = 24.
. 1 < = col < = 80
. pour tout i et j, emplacement (i)(j) = 0, 1 ou 2 selon que le fichier j est soumis, à la station i, à aucune contrainte, à une contrainte de présence obligatoire ou de présence interdite.
. pour tout i et j, allocation-fichier (i)(j) = 1 si le fichier j est alloué à la station i et 0 sinon.
. la saisie se fait en position (line, col).

Résultats = nom : chaîne de caractères.
retour de la fonction : entier.

Postcondition = . le fichier de nom "nom" existe, n'est pas déjà alloué et peut l'être ou "nom" est à ignorer.
. le retour égale 1 si "nom" est à prendre en compte et 0 sinon.

lec2(line,col)

Fonction = affichage d'un message à l'écran avec positionnement du curseur en (line,col) (cfr listing).

Arguments = line : entier.
col : entier.
Précondition = . 1 < = line < = 24.
. 1 < = col < = 80.

all_valide( )

Fonction = contrôle de la contrainte de cohérence imposée à l'allocation de fichiers : tout fichier doit être alloué au moins une fois. En cas de non vérification, activation d'un module pour résoudre le problème.

Arguments = allocation-fichier : tableau entier.
nbs, nbf : entier.
Précondition = . 0 < = nbs, nbf.
. pour tout i et j, allocation-fichier (i)(j) = 1 si le fichier j est alloué à la station i et 0 sinon.

Résultat = valide : entier.
Postcondition = valide égale 1 si l'allocation de fichiers vérifie la contrainte de cohérence et 0 sinon.

visu_pb(pr)

Fonction = affichage à l'écran de la liste des fichiers ne vérifiant pas la contrainte de cohérence : tout fichier doit être alloué au moins une fois.

Arguments = pr : tableau entier.
tabfich : tableau de structures fichier.
nbf : entier.

Précondition = . 0 < = nbf.
. tabfich contient nbf fichiers distincts.
. pour tout i, pr(i) = 1 si le fichier n° i ne vérifie pas la contrainte de cohérence.

resol_pb(pr)

Fonction = pour chaque fichier ne vérifiant pas la contrainte de cohérence relative à l'allocation de fichiers, saisie au terminal des stations où un exemplaire de ce fichier doit être alloué.

Arguments = pr : tableau entier.
tabfich : tableau de structures fichier.
nbf : entier.

Précondition = . 0 < = nbf.
. tabfich contient nbf fichiers distincts.
. pour tout i, pr(i) = 1 si le fichier n° i ne vérifie pas la contrainte de cohérence.

Résultats = allocation-fichier : tableau entier.
valide : entier.

Postcondition = . pour tout i et j, allocation-fichier (i)(j) = 1 si le fichier j est alloué à la station i.
. valide égale 1 si la nouvelle allocation vérifie la contrainte de cohérence et 0 sinon.

résol2_pb(fich,line,col,nom)

Fonction = saisie au terminal du nom d'une station où doit être alloué un exemplaire du fichier n° fich, avec contrôle de cohérence : existence de la station, allocation autorisée et non déjà spécifiée.

Arguments = fich : entier.
line : entier.
col : entier.

emplacement : tableau entier.
allocation-fichier : tableau entier.
nbs, nbf : entier.

Précondition = . 1 < = fich < = nbf.
. 0 < = nbs.
. 1 < = line < = 24.
. 1 < = col < = 80.
. pour tout i et j, emplacement (i)(j) = 0, 1 ou 2 selon que le fichier j est soumis, à la station i, à aucune contrainte, à une contrainte de présence obligatoire ou de présence interdite.
. pour tout i et j, allocation-fichier (i)(j) = 1 si le fichier j est alloué à la station i et 0 sinon.
. la saisie a lieu en (line,col).

Résultats = nom : chaîne de caractères.
retour de la fonction : entier.

Postcondition = . la station de nom "nom" existe, peut recevoir un exemplaire du fichier n° fich et ne l'a pas encore reçu, ou "nom" est à ignorer.
. le retour égale 1 si "nom" est à prendre en compte et 0 sinon.

ecran2( )

Fonction = affichage d'un texte à l'écran (cfr listing).

all_fich( )

Fonction = pour chaque fichier, visualisation des allocations de ce fichier aux stations, déclarées obligatoires par des contraintes d'emplacement et saisie au terminal des autres stations où allouer le fichier.

Arguments = tabfich : tableau de structures fichier.
tabstat : tableau de structures statnoeud.
emplacement : tableau entier.
nbs, nbf : entier.

Précondition = . 0 < = nbs, nbf.
. tabstat contient nbs stations distinctes 2 à 2.
. tabfich contient nbf fichiers distincts 2 à 2.
. pour tout i et j, emplacement (i)(j) = 0, 1 ou 2 selon que le fichier j est soumis, à la station i, à aucune contrainte, à une contrainte de présence obligatoire ou de présence interdite.

Résultats = allocation-fichier : tableau entier.
valide : entier.

Postcondition = . pour tout i et j, allocation-fichier (i)(j) = 1 si le fichier j est alloué à la station i et 0 sinon.
. pour tout j, il existe i tel que
allocation-fichier (i)(j) = 1.
. valide égale 1.

lec_stat(fich,line,col,nom)

Fonction = saisie au terminal du nom d'une station où doit être alloué le fichier n° fich avec contrôle de cohérence : au moins une allocation du fichier n° fich, existence de la station, allocation autorisée et non déjà spécifiée.

Arguments = fich : entier.
line : entier.
col : entier.
emplacement : tableau entier.
allocation-fichier : tableau entier.
nbs, nbf : entier.

Précondition = . 1 < = fich < = nbf.
. 0 < = nbs.
. 1 < = line < = 24.
. 1 < = col < = 80.
. pour tout i et j, emplacement (i)(j) = 0, 1, 2 selon que le fichier j est soumis, à la station i, à aucune contrainte, à une contrainte de présence obligatoire ou de présence interdite.

. pour tout i et j, allocation-fichier (i)(j) = 1 si le fichier j est alloué à la station i et 0 sinon.
. la saisie a lieu en (line,col).

Résultats = nom : chaîne de caractères.
retour de la fonction : entier.
Postcondition = . la station de nom "nom" existe, peut recevoir un exemplaire du fichier n° fich et ne l'a pas encore reçu ou "nom" est à ignorer.
. le retour égale 1 si "nom" est à prendre en compte et 0 sinon.
. si (line,col) égale (6,6) alors le retour doit égaler 1 et donc "nom" est à prendre en compte.

corr_alloc( )

Fonction = coordinateur de la logique de modification de l'allocation de fichiers.

Argument = valide : entier.
Précondition = valide égale 1 si l'allocation de fichiers respecte la contrainte de cohérence : tout fichier doit être alloué au moins une fois, et 0 sinon.

op_redif( )

Fonction = proposition de visualisation de l'allocation de fichiers courante.

Résultat = le retour de la fonction : caractère.
Postcondition = le retour vaut 'y' ou 'n'.

visu_alloc( )

Fonction = proposition de deux techniques de visualisation de l'allocation de fichiers courante : par énumération des stations ou par énumération des fichiers. Activation des modules ad hoc.

Résultat = retour de la fonction : caractère.
Postcondition = le retour vaut provient de visu4-alloc() et vaut 'a', 'r' ou 'm'.

visu2_alloc( )

Fonction = pour chaque station, visualisation des fichiers alloués.

Arguments = tabfich : tableau de structures fichier.
tabstat : tableau de structures station.
allocation-fichier : tableau entier.
nbs, nbf : entier.

Précondition = . 0 < = nbs, nbf.
. tabstat contient nbs stations distinctes.
. tabfich contient nbf fichiers distincts.
. pour tout i et j, allocation-fichier (i)(j) = 1 si le fichier j est alloué à la station i et 0 sinon.

visu3_alloc( )

Fonction = pour chaque fichier, visualisation des stations où un exemplaire est alloué.

Arguments = tabfich : tableau de structures fichier.
tabstat : tableau de structures station.
allocation-fichier : tableau entier.
nbs, nbf : entier.

Précondition = . 0 < = nbs, nbf.
. tabfich contient nbf fichiers distincts.
. tabstat contient nbs stations distinctes.
. pour tout i et j, allocation-fichier (i)(j) = 1 si le fichier j est alloué à la station i et 0 sinon.

visu4_alloc( )

Fonction = proposition de deux techniques de modification

de l'allocation de fichiers : redéfinition de l'allocation ou modification de l'allocation courante.

Résultat = retour de la fonction : caractère.
Postcondition = le retour vaut 'r' pour redéfinir, 'm' pour modifier ou 'a' pour quitter.

redef_alloc( )

Fonction = proposition de deux techniques de redéfinition de l'allocation de fichiers : par énumération des stations ou des fichiers et activation du module ad hoc.

Résultat = retour de la fonction : entier.
Postcondition = le retour vaut 0 si aucune redéfinition n'est faite et 1 ou 2 sinon.

clear_alloc( )

Fonction = annulation de l'allocation de fichier courante.

Arguments = nbs, nbf : entier.
Précondition = 0 < = nbs, nbf.

Résultat = allocation-fichier : tableau entier.
Postcondition = pour i et j, allocation-fichier (i)(j) = 0.

mod_alloc( )

Fonction = proposition de deux techniques de modification de l'allocation de fichiers courante : par énumération des fichiers ou des stations et activation du module ad hoc.

m_stat_alloc( )

Fonction = pour chaque station, visualisation des fichiers

qui lui sont alloués avec possibilité de modification et contrôle de validité de la modification.

Arguments = tabfich : tableau de structures fichier.
tabstat : tableau de structures statnoeud.
allocation-fichier : tableau entier.
nbs, nbf : entier.

Précondition = . 0 < = nbs, nbf.
. tabfich contient nbf fichiers distincts.
. tabstat contient nbs stations distinctes.
. pour tout i et j, allocation-fichier (i)(j) = 1 si le fichier j est alloué à la station et 0 sinon.

m2_stat_alloc(i)

Fonction = pour la station numéro i, proposition de deux types de modification : insertion d'une allocation et annulation d'une allocation, et activation du module ad hoc.

Argument = i : entier.

Précondition = 1 < = i < = nbs.

Résultat = retour de la fonction : caractère.

Postcondition = le retour vaut 'q' pour quitter ou 'c' pour continuer.

aj_fich(i)

Fonction = saisie au terminal du nom d'un fichier devant être alloué à la station numéro i avec mise à jour de l'écran et contrôle de cohérence : existence du fichier, allocation autorisée et non déjà spécifiée.

Arguments = tabfich : tableau de structures fichier.
emplacement : tableau entier.
allocation-fichier : tableau entier.
nbf : entier.
i : entier.

Précondition = . 0 < = nbf.
. tabfich contient nbf fichiers distincts.

. pour tout k et j, emplacement (k)(j) = 0, 1 ou 2
selon que le fichier j est soumis, à la station k,
à aucune contrainte, à une contrainte de présence
obligatoire ou de présence interdite.
. pour tout k et j, allocation-fichier (k)(j) = 1
si le fichier j est alloué à la station k
et 0 sinon.
. 1 < = i < = nbs.

Résultat = allocation-fichier : tableau entier.
Postcondition = pour tout k et j, allocation-fichier (k)(j) = 1
si le fichier j est alloué à la station k
et 0 sinon.

an_fich(i)

Fonction = saisie au terminal du nom d'un fichier ne devant
plus être alloué à la station numéro i avec mise
à jour de l'écran et contrôle de cohérence : exis-
tence du fichier, allocation non obligatoire et
spécifiée.

Arguments = tabfich : tableau de structures fichier.
emplacement : tableau entier.
allocation-fichier : tableau entier.
nbf : entier.
i : entier.
Précondition = . 0 < = nbf.
. tabfich contient nbf fichiers distincts.
. pour tout k et j, emplacement (k)(j) = 0, 1 ou 2
selon que le fichier j est soumis, à la station k,
à aucune contrainte, à une contrainte de présence
obligatoire ou de présence interdite.
. pour tout k et j, allocation-fichier (k)(j) = 1
si le fichier j est alloué à la station k
et 0 sinon.
. 1 < = i < = nbs.
Résultat = allocation-fichier : tableau entier.
Postcondition = pour tout k et j, allocation-fichier (k)(j) = 1
si le fichier j est alloué à la station k
et 0 sinon.

mod_écran(i)

Fonction = visualisation à l'écran de l'allocation de fichiers courante, après qu'une modification ait été effectuée, à la station n°i.

Arguments = tabfich : tableau de structures fichier.
tabstat : tableau de structures statnoeud.
allocation-fichier : tableau entier.
nbs, nbf : entier.
i : entier.

Précondition = . 0 < = nbs, nbf.
. tabfich contient nbf fichiers distincts.
. tabstat contient nbs stations distinctes.
. pour tout k et j, allocation-fichier (k)(j) = 1 si le fichier j est alloué à la station k et 0 sinon.
. 1 < = i < = nbs.

m_fich_alloc( )

Fonction = pour chaque fichier, visualisation des stations où est alloué ce fichier avec possibilité de modification et contrôle de validité de la modification.

Arguments = tabfich : tableau de structures fichier.
tabstat : tableau de structures station.
allocation-fichier : tableau entier.
nbs, nbf : entier.

Précondition = . 0 < = nbs, nbf.
. tabfich contient nbf fichiers distincts.
. tabstat contient nbs stations distinctes.
. pour tout i et j, allocation-fichier (i)(j) = 1 si le fichier j est alloué à la station i et 0 sinon.

m2_fich_alloc(i)

Fonction = pour le fichier numéro i, proposition de deux types de modification : insertion d'une allocation et

annulation d'une allocation, et activation du module ad hoc.

Argument = i : entier.
Précondition = 1 < = i < = nbf.

Résultat = retour de la fonction : caractère.
Postcondition = le retour vaut 'q' pour quitter ou 'c' pour continuer.

aj_stat(i)

Fonction = saisie au terminal du nom d'une station où doit être alloué le fichier numéro i, avec mise à jour de l'écran et contrôle de cohérence : existence de la station, allocation autorisée et non déjà spécifiée.

Arguments = tabstat : tableau de structures station.
emplacement : tableau entier.
allocation-fichier : tableau entier.
nbs : entier.
i : entier.

Précondition =
. 0 < = nbs.
. 1 < = i < = nbf.
. tabstat contient nbs stations distinctes.
. pour tout k et j, emplacement (k)(j) = 0, 1 ou 2 selon que le fichier j est soumis, à la station k, à aucune contrainte, à une contrainte de présence obligatoire ou de présence interdite.
. pour tout k et j, allocation-fichier (k)(j) = 1 si le fichier j est alloué à la station k et 0 sinon.

Résultat = allocation-fichier : tableau entier.
Postcondition = pour tout k et j, allocation-fichier (k)(j) = 1 si le fichier j est alloué à la station k et 0 sinon.

an_stat(i)

Fonction = saisie au terminal du nom d'une station où ne doit plus être alloué le fichier numéro i, avec mise à jour de l'écran et contrôle de cohérence : existence de la station, allocation non obligatoire et spécifiée.

Arguments = tabstat : tableau de structures statnoeud.
emplacement : tableau entier.
allocation-fichier : tableau entier.
nbs : entier.
i : entier.

Précondition = . 0 < = nbs.
. 1 < = i < = nbf.
. pour tout k et j, emplacement (k)(j) = 0, 1 ou 2 selon que le fichier j est soumis, à la station k, à aucune contrainte, à une contrainte de présence obligatoire ou de présence interdite.
. pour tout k et j, allocation-fichier (k)(j) = 1 si le fichier j est alloué à la station k et 0 sinon.
. tabstat contient nbs stations distinctes.

Résultat = allocation-fichier : tableau entier.

Postcondition = pour tout k et j, allocation-fichier (k)(j) = 1 si le fichier j est alloué à la station k et 0 sinon.

mod2_ecran(i)

Fonction = visualisation à l'écran de l'allocation courante du fichier numéro i, après qu'une modification ait été apportée dans l'allocation.

Arguments = tabfich : tableau de structures fichier.
tabstat : tableau de structures statnoeud.
allocation-fichier : tableau entier.
i : entier.
nbs, nbf : entier.

Précondition = . 0 < = nbs, nbf.
. 1 < = i < = nbf.
. tabfich contient nbf fichiers distincts.
. tabstat contient nbs stations distinctes.
. pour tout k et j, allocation-fichier (k)(j) = 1 si le fichier j est alloué à la station k et 0 sinon.

ok_an(i)

Fonction = vérification d'une des contraintes de cohérence relative à l'allocation de fichier, pour le fichier i : au moins un exemplaire du fichier dans le réseau. Dans notre cas, on désire deux exemplaires, le contrôle étant réalisé avant d'annuler une allocation.

Arguments = allocation-fichier : tableau entier.
nbs : entier.
i : entier.
Précondition = . 0 < = nbs.
. 1 < = i < = nbf.
. pour tout k, allocation-fichier (k)(i) = 1 si le fichier i est alloué à la station k et 0 sinon.

Résultat = retour de la fonction : entier.
Postcondition = le retour vaut 0 si le nombre d'exemplaires du fichier n°i, alloués au réseau, est inférieur ou égal à 1 et 1 sinon.

quest_alloc( )

Fonction = proposition de visualisation de l'allocation de fichiers introduite.

Résultat = retour de la fonction : caractère.
Postcondition = le retour vaut 'y' ou 'n'.

FICHIER MEMCAPAC . C

Ce fichier contient les procédures nécessaires à l'introduction des capacités disponibles dans le réseau et leur visualisation.

main(argc,argv)

Fonction = entrée au sous-programme réalisant l'introduction des capacités avec stockage sur fichier.

Arguments = argc : entier.
argv : tableau de pointeurs.

Précondition = argv contient deux paramètres : le niveau atteint (5) et le nom du fichier où il faut stocker les données introduites.

ent_capac( )

Fonction = test du caractère cohérent de l'introduction (au moins une capacité).

Résultat = retour de la procédure : entier.

Postcondition = le retour égale 1 si au moins une capacité a été introduite et 0 sinon.

capacite( )

Fonction = coordinateur de la logique d'introduction.

intro_capac( )

Fonction = coordinateur de second niveau de l'introduction des quatre types de capacités.

titre_capac( )

Fonction = affichage d'un texte à l'écran (cfr listing).

titre1_capac( )

Fonction = affichage d'un texte à l'écran (cfr listing).

titre2_capac( )

Fonction = affichage d'un texte à l'écran (cfr listing).

titre3_capac( )

Fonction = affichage d'un texte à l'écran (cfr listing).

titre4_capac( )

Fonction = affichage d'un texte à l'écran (cfr listing).

lec1capac( )

Fonction = saisie au terminal de la description des capacités louées admissibles.

Argument = CLMAX : entier.
Précondition = 0 < CLMAX

Résultats = tabloue : tableau de structures cap-loue.
nbcaploue : entier.
Postcondition = . 0 < = nbcaploue < = CLMAX - 1.
. tabloue contient nbcaploue capacités louées 2 à 2 distinctes.

lec2_capac( )

Fonction = saisie au terminal de la description des capacités non louées admissibles.

Argument = CNLMAX : entier.
Précondition = 0 < CNLMAX.

Résultats = tabnloue : tableau de structures cap-non-loue
nbcapnloue : entier.
Postcondition = . 0 <= nbcapnloue < = CNLMAX - 1.
. tabnloue contient nbcapnloue capacités non louées 2 à 2 distinctes.

lec3_capac( )

Fonction = saisie au terminal de la description des capacités DCS louées admissibles.

Argument = DCSLMAX : entier.
Précondition = 0 < DCSLMAX.

Résultats = tabdcsl : tableau de structures dcs-loue.
nbdcsloue : entier.
Postcondition = . 0 < = nbdcsloue < = DCSLMAX - 1.
. tabdcsl contient nbdcsloue capacités DCS louées 2 à 2 distinctes.

lec4_capac( )

Fonction = saisie au terminal de la description des capacités DCS non louées admissibles.

Argument = DCSNLMAX : entier.
Précondition = 0 < DCSNLMAX.

Résultats = tabdcsnl : tableau de structures dcs-non-loue.
nbdcsnloue : entier.
Postcondition = . 0 < = nbdcsnloue < = DCSNLMAX - 1 .
. tabdcsnl contient nbdcsnloue capacités DCS non louées 2 à 2 distinctes.

mod1_capac( )

Fonction = visualisation par groupe de 5, des capacités louées mémorisées et possibilité de modification des descriptions affichées à l'écran.

Arguments = tabloue : tableau de structures cap-loue.
nbcaploue : entier.

Précondition = . 0 < = nbcaploue.
. tabloue contient nbcaploue capacités 2 à 2 distinctes.

Résultat = tabloue : tableau de structures cap-non-loue.

Postcondition = tabloue contient nbcaploue capacités 2 à 2 distinctes.

mod2_capac( )

Fonction = visualisation par groupe de 5, des capacités non louées mémorisées et possibilité de modification des descriptions affichées à l'écran.

Arguments = tabloue : tableau de structures cap-non-loue.
nbcapnloue : entier.

Précondition = . 0 < = nbcapnloue.
. tabnloue contient nbcapnloue capacités 2 à 2 distinctes.

Résultat = tabnloue : tableau de structures cap-non-loue.

Postcondition = tabnloue contient nbcapnloue capacités 2 à 2 distinctes.

mod3_capac( )

Fonction = visualisation par groupe de 5, des capacités DCS louées mémorisées et possibilité de modification des descriptions affichées à l'écran.

Arguments = tabdcsl : tableau de structures dcs-loue.
nbdcsloue : entier.
Précondition = . 0 < = nbdcsloue.
. tabdcsl contient nbdcsloue capacités 2 à 2 distinctes.

Résultat = tabdcsl : tableau de structures dcs-loue.
Postcondition = tabdcsl contient nbdcsloue capacités 2 à 2 distinctes.

mod4_capac( )

Fonction = visualisation par groupe de 5, des capacités DCS non louées mémorisées et possibilité de modification des descriptions affichées à l'écran.

Arguments = tabdcsnloue : tableau de structures dcs-non-loue.
nbdcsnloue : entier.
Précondition = . 0 < = nbdcsnloue.
. tabdcsnl contient nbdcsnloue capacités 2 à 2 distinctes.

Résultat = tabdcsnl : tableau de structures dcs-non-loue.
Postcondition = tabdcsnl contient nbdcsnloue capacités 2 à 2 distinctes.

const_capac( )

Fonction = introduction des prix des capacités louées, déclarés fixes pour certaines lignes.

Argument = tablig : tableau de structures ligne.
nbl : entier.
nbcaploue : entier.
bidir : entier.
Précondition = . 0 < = nbl.
. 0 < nbcaploue.
. bidir = 0 ou 1.
. tablig contient nbl lignes distinctes.

Résultat = prix-fx-ligne : tableau réel.
Postcondition = pour tout i et j, prix-fx-ligne (i)(j) > = 0.

quest_capac( )

Fonction = proposition de visualisation des capacités introduites.

Résultat = retour de la fonction : caractère.
Postcondition = le retour vaut 'y' ou 'n'.

sauve_cap( )

Fonction = mémorisation de toutes les valeurs de capacité introduites dans une table, avec calcul du coût de l'allocation des capacités louées aux lignes du réseau.

Arguments = prix-fx-ligne : tableau réel.
tablig : tableau de structures ligne.
tabloue : tableau de structures cap-loue.
tabnloue : tableau de structures cap-non-loue.
tabdcsl : tableau de structures dcs-loue.
tabdcsnl : tableau de structures dcs-non-loue.
nbl, nbcaploue, nbcapnloue, nbdcsloue, nbdcsnloue : entier.
Précondition = 0 < = nbl, nbcaploue, nbcapnloue, nbdcsloue, nbdcsnloue.
Résultat = cap-tot : tableau de structures cap.
Postcondition = . cap - tot contient toutes les valeurs de capacité introduites et présentes dans tabloue, tabnloue, tabdcsl et tabdcsnl.
. pour tout i et j, cap-tot(i)(j).coût = le coût de l'allocation à i si j est louée et 0 sinon.

mod_const_cap( )

Fonction = proposition de visualisation des prix des capacités louées, déclarés fixes pour certaines lignes.

mod2_const_cap( )

Fonction = visualisation des prix des capacités louées, déclarés fixes pour certaines lignes et possibilité de modification des prix affichés à l'écran.

Arguments = nbl, nbcaploue, bidir : entier.
tablig : tableau de structures ligne.

Précondition = . 0 < = nbl, nbcaploue.
. bidir = 0 ou 1.
. tablig contient nbl lignes distinctes.

Résultat = prix-fx-ligne : tableau réel.

Postcondition = pour tout i et j, prix-fx-ligne(i)(j) > = 0.

FICHIER MEMCONTR . C

Ce fichier contient les procédures nécessaires à l'introduction des contraintes d'emplacement sur les fichiers et leur visualisation.

main(argc,argv)

Fonction = entrée au sous-programme réalisant l'introduction des contraintes d'emplacement avec stockage sur fichier.

Arguments = argc : entier.
argv : tableau de pointeurs.
Précondition = argv contient deux paramètres : le niveau atteint (9) et le nom du fichier où il faut stocker les données introduites.

contr_emplacement( )

Fonction = coordinateur de la logique de l'introduction.

init_contrainte( )

Fonction = initialisation de la table qui indique le type de contrainte d'emplacement liant un fichier et une station; valeur prise : pas de contrainte.

Arguments = nbs, nbf : entier.
Précondition = 0 < = nbs, nbf.

Résultat = emplacement : tableau entier.
Postcondition = pour tout i et j, emplacement (i)(j) = 0

int_contrainte( )

Fonction = proposition de quatre types de technique d'introduction des contraintes d'emplacement : par énumération des stations, par énumération des fichiers, par introduction explicite et par énumération des couples (station, fichier) et activation du module ad hoc.

ll_contrainte( )

Fonction = pour chaque station du réseau, saisie au terminal du nom des fichiers soumis à une contrainte d'emplacement à cette station et du type des contraintes : pas de contrainte, présence obligatoire, présence interdite.

Arguments = tabstat : tableau de structures statnoeud.
nbs, nbf : entier.
emplacement : tableau entier.

Précondition = . 0 < = nbs, nbf.
. tabstat contient nbs stations distinctes 2 à 2.
. pour tout i et j, emplacement (i)(j) = 0.

Résultat = emplacement : tableau entier.

Postcondition = pour tout i et j, emplacement = 0, 1 ou 2 selon que le fichier j soit soumis, à la station i, à aucune contrainte, à une contrainte de présence obligatoire ou à une contrainte de présence interdite.

l2_contrainte( )

Fonction = pour chaque fichier du réseau, saisie au terminal du nom des stations où il subit une contrainte d'emplacement et du type des contraintes : pas de contrainte, présence obligatoire, présence interdite.

Arguments = tabfich : tableau de structures fichier.
nbs, nbf : entier.
emplacement : tableau entier.
Précondition = . 0 < = nbs, nbf.
. tabfich contient nbf fichiers distincts.
. pour tout i et j, emplacement(i)(j) = 0.
Résultat = emplacement : tableau entier.
Postcondition = pour tout i et j, emplacement (i)(j) = 0, 1 ou 2 selon que le fichier j est soumis, à la station i, à aucune contrainte, à une contrainte de présence obligatoire ou à une contrainte de présence interdite.

13_contrainte( )

Fonction = saisie au terminal d'une liste de triplets (station, fichier, contrainte subie par fichier à station) où la contrainte peut être de trois types : pas de contrainte, présence obligatoire, présence interdite.

Argument = emplacement : tableau entier.
Précondition = pour tout i et j, emplacement(i)(j) = 0.

Résultat = emplacement : tableau entier.
Postcondition = pour tout i et j, emplacement (i)(j) = 0, 1 ou 2, selon que le fichier j est soumis, à la station i, à aucune contrainte, à une contrainte de présence obligatoire, à une contrainte de présence interdite.

14_contrainte( )

Fonction = pour chaque couple (station, fichier), saisie au terminal du type de la contrainte subie par ce fichier à cette station : pas de contrainte, présence obligatoire, présence interdite.

Arguments = tabfich : tableau de structures fichier.
tabstat : tableau de structures station.
emplacement : tableau entier.
nbs, nbf : entier.

Précondition = . 0 < = nbs, nbf.
. tabstat contient nbs stations distinctes.
. tabfich contient nbf fichiers distincts.
. pour tout i et j, emplacement(i)(j) = 0.

Résultat = emplacement : tableau entier.

Postcondition = pour tout i et j, emplacement(i)(j) = 0, 1 ou 2 selon que le fichier j est soumis, à la station i, à aucune contrainte, à une contrainte de présence obligatoire ou à une contrainte de présence interdite.

mod_contrainte( )

Fonction = proposition de trois techniques de visualisation des contraintes d'emplacement avec possibilité de modification : par énumération des stations, par énumération des fichiers, par définition explicite.

ml_contrainte( )

Fonction = pour chaque station, visualisation des contraintes d'emplacement imposées aux fichiers avec possibilité de modification.

Arguments = tabstat : tableau de structures statnoeud.
nbs : entier.

Précondition = . tabstat contient nbs stations distinctes.
. 0 < = nbs.

mll_contrainte(i)

Fonction = visualisation des contraintes d'emplacement subies par la station numéro i avec possibilité de modification.

Arguments = tabfich : tableau de structures fichier.
emplacement : tableau entier.
nbf : entier.
i : entier.

Précondition = . tabfich contient nbf fichiers distincts.
. pour tout j, emplacement(i)(j) = 0, 1 ou 2 selon que le fichier j est soumis, à la station i, à aucune contrainte, à une contrainte de présence obligatoire ou de présence interdite.
. 1 < = i < = nbs.
. 0 < = nbf.

Résultat = emplacement : tableau entier.

Postcondition = pour tout j, emplacement(i)(j) = 0, 1 ou 2 selon que le fichier j est soumis, à la station i, à aucune contrainte, à une contrainte de présence obligatoire ou de présence interdite.

m2_contrainte( )

Fonction = pour chaque fichier, visualisation des contraintes d'emplacement imposées aux fichiers avec possibilité de modification.

Arguments = tabfich : tableau de structures fichier.
nbf : entier.

Précondition = . 0 < = nbf.
. tabfich contient nbf fichiers distincts.

m21_contrainte(i)

Fonction = visualisation des contraintes d'emplacement subies par le fichier numéro i avec possibilité de modification.

Arguments = tabstat : tableau de structures station.
emplacement : tableau entier.
nbs : entier.
i : entier.

Précondition = . tabstat contient nbs stations distinctes.
. pour tout j, emplacement(j)(i) = 0, 1 ou 2 selon que le fichier i est soumis, à la station j, à aucune contrainte, à une contrainte de présence obligatoire ou de présence interdite.
. 1 < = i < = nbf.
. 0 < = nbs.

Résultat = emplacement : tableau entier.
Postcondition = pour tout j, emplacement(j)(i) = 0, 1 ou 2 selon que le fichier i est soumis, à la station j, à aucune contrainte, à une contrainte de présence obligatoire ou de présence interdite.

m3_contrainte( )

Fonction = visualisation des contraintes d'emplacement pour une série de couples (station, fichier) introduits au terminal, avec possibilité de modification.

Argument = emplacement : tableau entier.
Précondition = pour tout i et j, emplacement(i)(j) = 0, 1 ou 2 selon que le fichier j est soumis, à la station i, à aucune contrainte, à une contrainte de présence obligatoire ou de présence interdite.

ch_cont(i,j)

Fonction = saisie au terminal d'une nouvelle valeur pour la contrainte d'emplacement imposée au fichier numéro j à la station numéro i.

Arguments = i, j : entier.
emplacement : tableau entier.
Précondition = . 1 < = i < = nbs.
. 1 < = j < = nbf
. emplacement(i)(j) = 0, 1 ou 2 selon que le fichier j est soumis, à la station i, à aucune contrainte, à une contrainte de présence obligatoire ou de présence interdite.

Résultat = emplacement : tableau entier.
Postcondition = emplacement (i)(j) = 0, 1 ou 2 selon que le fichier j est soumis, à la station i, à aucune contrainte, à une contrainte de présence obligatoire ou de présence interdite.

que_cont( )

Fonction = proposition de visualisation des contraintes d'emplacement introduites.

Résultat = retour de la fonction : caractère.
Postcondition = le retour égale 'y' ou 'n'.

FICHIER MEMEX . C

Ce fichier contient des procédures d'utilité générale pour le logiciel.

ex_stat(nom,n)

Fonction = contrôle de l'existence de la description de la station appelée nom, dans les n premières stations mémorisées.

Arguments = nom : chaîne de caractères.
n : entier.
tabstat : tableau de structures statnoeud.

Précondition = . 0 < = n < = nbs.
. tabstat contient nbs stations distinctes.

Résultat = retour de la fonction : entier.

Postcondition = le retour vaut 0 si la station appelée nom est déjà mémorisée dans tabstat et 1 sinon.

ex_noeud(nom,n)

Fonction = contrôle de l'existence de la description du noeud appelé nom, dans les n premiers noeuds mémorisés.

Arguments = nom : chaîne de caractères.
n : entier.
tabstat : tableau de structures statnoeud.
nbs : entier.

Précondition = . 0 < = nbs.
. 0 < = n < = nbn.
. tabstat contient nbn noeuds distincts mémorisés dans tabstat à partir de la position nbs + 1.

Résultat = retour de la fonction : entier.

Postcondition = le retour vaut 0 si le noeud appelé nom est déjà mémorisé dans tabstat et 1 sinon.

ex_lig(no1,no2,n)

Fonction = contrôle de l'existence de la description de la ligne identifiée par ses extrémités no1 et no2, dans les n premières lignes mémorisées.

Arguments = no1 : chaîne de caractères.
no2 : chaîne de caractères.
n : entier.
tablig : tableau de structures ligne.

Précondition = . 0 < = n < = nbl
. tablig contient nbl lignes distinctes.

Résultat = retour de la fonction : entier.

Postcondition = le retour vaut 1 si la ligne d'extrémité n°1 et n°2 est déjà mémorisée dans tablig et 0 sinon.

ex_fich(nom,n)

Fonction = contrôle de l'existence de la description du fichier appelé nom, dans les n premiers fichiers mémorisés.

Arguments = nom : chaîne de caractères.
n : entier.
tabfich : tableau de structures fichier.

Précondition = . 0 < = n < = nbf.
. tabfich contient nbf fichiers distincts.

Résultat = retour de la fonction : entier.

Postcondition = le retour vaut 0 si le fichier appelé nom est déjà mémorisé dans tabfich et 1 sinon.

ex_lcap(val)

Fonction = contrôle de l'existence de la valeur de capacité val dans les capacités louées.

Arguments = val : réel.
tabloue : tableau de structures cap-loué
nbcaploue : entier.

Précondition = . 0 < = nbcaploue.
. tabloue contient nbcaploue capacités louées distinctes.

Résultat = retour de la fonction : entier.
Postcondition = le retour vaut 1 si la valeur de capacité val est mémorisée dans tabloue et 0 sinon.

ex_nlcap(val)

Fonction = contrôle de l'existence de la valeur de capacité val dans les capacités non louées.

Arguments = val : réel.
tabnloue : tableau de structures cap-non-loue.
nbcapnloue : entier.
Précondition = . 0 < = nbcapnloue.
. tabloue contient nbcapnloue capacités non louées distinctes.

Résultat = retour de la fonction : entier.
Postcondition = le retour vaut 1 si la valeur de capacité val est mémorisée dans tabnloue et 0 sinon.

ex_ldcs(val)

Fonction = contrôle de l'existence de la valeur de capacité val dans les capacités DCS louées.

Arguments = val : réel.
tabdcsl : tableau de structures dcs-loue.
nbdcsloue : entier.
Précondition = . 0 < = nbdcsloue.
. tabdcsl contient nbdcsloue capacités dcs louées distinctes.

Résultat = retour de la fonction : entier.
Postcondition = le retour vaut 1 si la valeur de capacité val est mémorisée dans tabdcsl et 0 sinon.

num_statnoeud(no)

Fonction = détermination du numéro d'ordre de la station ou du noeud appelé no, dans les données introduites.

Arguments = no : chaîne de caractères.
tabstat : tableau de structures statnoeud.
nbs, nbn : entier.

Précondition = . 0 < = nbs, nbn.
. tabstat contient nbs stations positionnées de 1 à nbs et nbn noeuds positionnés de nbs + 1 à nbs + nbn.

Résultat = retour de la fonction : entier.

Postcondition = . le retour vaut num si num indique la position d'un noeud ou d'une station dont le nom est no, dans tabstat et 0 sinon.
. 0 < = retour < = nbs + nbn

num_fich(no)

Fonction = détermination du numéro d'ordre du fichier appelé no, dans la liste des fichiers mémorisés.

Arguments = no : chaîne de caractères.
tabfich : tableau de structures fichier.
nbf : entier.

Précondition = . 0 < = nbf.
. tabfich contient nbf fichiers distincts.

Résultat = retour de la fonction : entier.

Postcondition = . le retour vaut num si num indique la position d'un fichier dont le nom est no, dans tabfich.
. 0 < = retour < = nbf.

num_ligne(no1,no2)

Fonction = détermination du numéro d'ordre de la ligne dont les extrémités sont no1 et no2 dans les lignes mémorisées.

Arguments = no1, no2 : chaîne de caractères.
tablig : tableau de structures ligne.
nbl : entier.

Précondition = . 0 < = nbl
. tablig contient nbl lignes distinctes.

Résultat = retour de la fonction : entier.

Postcondition = . le retour vaut num si num indique la position d'une ligne identifiée par no1 et no2, dans tablig et 0 sinon.
. 0 < = retour < = nbl.

FICHIER MEMFIAB . C

Ce fichier contient les procédures nécessaires à l'introduction des contraintes de fiabilité du réseau et leur visualisation.

main(argc,argv)

Fonction = entrée au sous-programme réalisant l'introduction des contraintes de fiabilité avec stockage sur fichier.

Arguments = argc : entier.
argv : tableau de pointeurs.
Précondition = argv contient deux paramètres : le niveau atteint (8) et le nom du fichier où il faut stocker les données introduites.

c_fiabilité( )

Fonction = coordinateur de la logique de l'introduction.

int_fiabilité( )

Fonction = saisie au terminal de la valeur de la fiabilité de la liaison entre chaque couple de stations.

Arguments = tabstat : tableau de structures statnoeud.
nbs : entier.
Précondition = . tabstat contient nbs éléments 2 à 2 distincts.
. 0 < = nbs.

Résultat = fiabilité : tableau réel.
Postcondition = . pour tout i et j, 0 < = fiabilité(i)(j) < = 1.

mod_fiabilité( )

Fonction = visualisation des valeurs de fiabilité des liaisons entre chaque couple de stations avec possibilité de modification.

Arguments = tabstat : tableau de structure statnoeud.
fiabilité : tableau réel.
nbs : entier.

Précondition = . tabstat contient nbs éléments distincts 2 à 2.
. 0 < = nbs.
. pour tout i et j, 0 < = fiabilité(i)(j) < = 1.

Résultat = fiabilité : tableau réel.

Postcondition = ; pour tout i et j, 0 < = fiabilité(i)(j) < = 1.

que_fiabilité( )

Fonction = proposition de visualisation des valeurs de fiabilité introduites.

Résultat = retour de la fonction : caractère.

Postcondition = le retour égale 'y' ou 'n'.

FICHIER MEMFICHIER . C

Ce fichier contient les procédures nécessaires à l'introduction des fichiers présents dans le réseau et leur visualisation.

main(argc,argv)

Fonction = entrée au sous-programme réalisant l'introduction des fichiers avec stockage sur fichier de données.

Arguments = argc : entier.
argv : tableau de pointeurs.

Précondition = argv contient 2 paramètres : le niveau atteint (4) et le nom du fichier où il faut stocker les données introduites.

int_fich( )

Fonction = test du caractère cohérent de l'introduction, c'est-à-dire au moins un fichier.

Résultat = retour de la fonction : entier.

Postcondition = le retour égale 1 si au moins un fichier a été introduit.

fichier( )

Fonction = coordinateur de la logique d'introduction.

intro_fich( )

Fonction = saisie au terminal de la description des fichiers.

Argument = FMAX : entier.
Précondition = 0 < FMAX.

Résultats = tabfich : tableau de structures fichier.
nbf : entier.
Postcondition = . tabfich contient nbf fichiers 2 à 2 distincts.
. 0 < = nbf < = FMAX - 1.

ecran-fich( )

Fonction = affichage à l'écran d'une grille permettant d'introduire les données de type fichier.

visu-fich( )

Fonction = visualisation par groupe de 5, des fichiers mémorisés.

Arguments = tabfich : tableau de structures fichier.
nbf : entier.
Précondition = 0 < = nbf.

mod_fich_intro( )

Fonction = visualisation par groupe de 5, des stations mémorisées et possibilité de modification des descriptions des fichiers affichés à l'écran.

Arguments = tabfich : tableau de structures fichier.
nbf : entier.
Précondition = . 0 < = nbf.
. tabfich contient nbf fichiers 2 à 2 distincts.

Résultat = tabfich : tableau de structures fichier.
Postcondition = tabfich contient nbf fichiers 2 à 2 distincts.

q_modif( )

Fonction = proposition de visualisation des fichiers introduits.
Résultat = retour de la fonction : caractère.
Postcondition = le retour vaut 'y' ou 'n'.

FICHIER MEMFISTOCK . C

Ce fichier contient les procédures nécessaires au sauvetage du réseau courant sur un autre fichier de données.

main(argc,argv)

Fonction = entrée au sous-programme réalisant la copie du fichier de données contenant le réseau courant, sur un autre fichier de données.

Arguments = argc : entier.
argv : chaîne de pointeurs.

Précondition = argv contient un paramètre : le nom du fichier de données contenant la description du réseau courant.

lire_output(output)

Fonction = saisie au terminal du nom d'un fichier n'existant pas encore.

Résultats = output : chaîne de caractères.
retour de la fonction : entier.

Postcondition = . le retour vaut 1 si output est à prendre en considération et 0 si aucun nom de fichier n'a été donné.
. output contient le nom d'un fichier ou le caractère '*' de non-introduction.

FICHIER MEMLIGNE . C

Ce fichier contient les procédures nécessaires à l'introduction des stations du réseau à étudier et leur visualisation.

main( )

Fonction = entrée au sous-programme réalisant l'introduction des lignes.

Arguments = argc : entier.
argv : tableau de pointeurs.

Précondition = argv contient 2 paramètres : le niveau atteint (3) et le nom du fichier où il faut stocker les données introduites.

intro_ligne( )

Fonction = contrôle de la cohérence de l'introduction : une ligne au minimum doit être introduite.

Résultat = retour de la procédure : entier.

Postcondition = le retour égale 1 si au moins une ligne a été introduite et 0 sinon.

ligne( )

Fonction = coordinateur de la logique d'introduction.

uni_ou_bi( )

Fonction = introduction du type des lignes : unidirectionnelles ou bidirectionnelles.

Résultat = retour de la procédure : caractère.

Postcondition = . tablig contient nbl lignes 2 à 2 distinctes.
. nbl < = LMAX - 1.

ecran_lig( )

Fonction = affichage à l'écran d'une grille permettant d'introduire les données de type ligne.

visu_uni( )

Fonction = visualisation par groupe de 5, des lignes mémorisées : cas unidirectionnel.

Arguments = tablig : tableau de structures ligne.
nbl : entier.
Précondition = 0 < = nbl.

mod_uni( )

Fonction = visualisation par groupe de 5, des lignes mémorisées et possibilité de modification des descriptions des lignes affichées à l'écran : cas unidirectionnel.

Argument = tablig : tableau de structures ligne.
nbl : entier.
Précondition = . 0 < = nbl.
. les nbl éléments de tablig sont distincts 2 à 2.

Résultat = tablig : tableau de structures ligne.
Postcondition = les nbl éléments de tablig sont distincts 2 à 2.

qunimod( )

Fonction = proposition de visualisation des lignes introduites : cas unidirectionnel.

Résultat = retour de la fonction : caractère.
Postcondition = le retour vaut 'y' ou 'n'.

lec_bi( )

Fonction similaire à lec-uni() : cas bidirectionnel.

visu_bi( )

Fonction similaire à visu-uni() : cas bidirectionnel.

mode_bi( )

Fonction similaire à mode-uni(); cas bidirectionnel.

qbimod( )

Fonction similaire à qunimod() : cas bidirectionnel.

table_ligne( )

Fonction = construction de la table des distances entre les stations reliées par une ligne et de la table du numéro de la ligne entre chaque paire de stations.

Arguments = tablig : tableau de structures ligne.
nbl : entier.
nbs : entier.
nbn : entier.
Postcondition = 0 < = nbl, nbs, nbn.

Résultats = mat-ligne : tableau entier.
mat-dist : tableau réel.
Postcondition = soient num1 et num2 les numéros de 2 stations
. mat-ligne (num1)(num2) = i si la ligne i relie num1 à num2,
0 sinon.
. mat-dist (num1)(num2) = la longueur de la ligne les reliant si elle existe,
0 si num1 = num2,
999999 sinon.

FICHIER MEMNOEUD . C

Ce fichier contient les procédures nécessaires à l'introduction des noeuds du réseau à étudier et leur visualisation.

main(argc,argv)

Fonction = entrée au sous-programme réalisant l'introduction des noeuds avec stockage sur fichier.

Arguments = argc : entier.
argv : tableau de pointeurs.

Précondition = argv contient 2 paramètres : le niveau atteint (1) et le nom du fichier où sont stockées les informations.

noeud( )

Fonction = coordinateur de la logique d'introduction.

intro_noeud( )

Fonction = saisie au terminal de la description des noeuds du réseau

Arguments = NMAX : entier.
nbs : entier.
tabstat : tableau de structures statnoeud.

Précondition = . 0 < NMAX
. nbs = numéro de la dernière position occupée dans tabstat (> 2).

Résultats = tabstat : tableau de structures statnoeud.
nbn : entier.

Postcondition = . tabstat contient nbn noeuds 2 à 2 distincts en position nbs + 1 à nbs + nbn.
. nbn < = NMAX - 1.

ecran_noeud( )

Fonction = affichage à l'écran d'une grille permettant d'introduire les données de type noeud.

visu_noeud( )

Fonction = visualisation par groupe de 5, des noeuds mémorisés.

Arguments = tabstat : tableau de structures statnoeud.
nbs : entier.
nbn : entier.
Précondition = . 2 < = nbs.
. 0 < = nbn.

mod_noeud_intro( )

Fonction = visualisation par groupe de 5, des noeuds mémorisés et possibilité de modification des descriptions des noeuds affichés à l'écran.

Arguments = tabstat : tableau de structures statnoeud.
nbs : entier.
nbn : entier.
Précondition = . 0 < = nbn.
. 2 < nbs.
. deux éléments de tabstat ne peuvent avoir le même nom.

Résultat = tabstat : tableau de structures statnoeud.
Postcondition = tabstat contient nbs + nbn éléments 2 à 2 distincts dont les nbs premiers n'ont pas été modifiés.

ques_modif( )

Fonction = proposition de visualisation des stations introduites.
Résultat = retour de la fonction : caractère.
Postcondition = le retour vaut 'y' ou 'n'.

FICHIER MEMPARA . C

Ce fichier contient les procédures nécessaires à l'introduction des paramètres du réseau et leur visualisation.

main(argc,argv)

Fonction = entrée au sous-programme réalisant l'introduction des paramètres d'utilisation du réseau avec stockage sur fichier.

Arguments = argc : entier.
argv : tableau de pointeurs.

Précondition = argv contient 2 paramètres : le niveau atteint (6) et le nom du fichier où il faut stocker les données introduites.

paramètre( )

Fonction = coordinateur de la logique d'introduction.

quest_para_modif( )

Fonction = proposition de visualisation des paramètres introduits.

Résultat = retour de la fonction : caractère.

Postcondition = le retour vaut 'y' ou 'n'.

intro_option( )

Fonction = saisie au terminal de la valeur des paramètres d'utilisation du réseau (1e partie).

Résultats = régime : entier.
util-uniforme : entier.
délai-réseau : réel.
long-segment : entier.

Postcondition = . régime égale 0 ou 1 selon que le régime est permanent ou jour-nuit.
. util-uniforme égale 0 ou 1 selon que l'accès aux fichiers est uniforme ou limité au plus proche exemplaire.
. 0 < = délai-réseau.
. 0 < = long-segment.

intro2_option(reg)

Fonction = saisie au terminal de la valeur des paramètres d'utilisation du réseau (2e partie).

Arguments = reg : entier.

Précondition = reg égale 0 ou 1.

Résultats = tmax : réel.
prix-segment : réel.
prix-jour-segment : réel.
prix-nuit-segment : réel.

Postcondition = . 0 < = tmax.
. 0 < = prix-segment, prix-jour-segment, prix-nuit-segment.
. si reg = 0, seuls tmax et prix-segment sont à saisir; si reg = 1, seuls tmax, prix-jour-segment et prix-nuit-segment sont à saisir.

intro3_option( )

Fonction = saisie au terminal de la valeur des paramètres d'utilisation du réseau (3e partie).

Résultat = durée-jour : réel.
durée-nuit : réel.

Postcondition = . 0 < = durée-jour, durée-nuit.
. durée-jour + durée-nuit < = 86 400.

ecran_option( )

Fonction = affichage d'un texte à l'écran (cfr listing).

mod_option( )

Fonction = visualisation à l'écran des paramètres d'utilisation introduits.

Arguments = régime : entier.
util-uniforme : entier.
délai-réseau : réel.
long-segment : entier.
tmax : réel.
prix-segment : réel.
prix-jour-segment : réel.
prix-nuit-segment : réel.
durée-jour : réel.
durée-nuit : réel.

Précondition = . régime = 0 ou 1.
. util-uniforme = 0 ou 1.
. 0 < = délai-réseau, long-segment, tmax, prix-segment, prix-jour-segment, prix-nuit-segment, durée-jour, durée-nuit.
. durée-jour + durée-nuit < = 86 400.

mod_perm_option( )

Fonction = coordinateur de la modification des paramètres d'utilisation du réseau : cas où le régime est permanent.

mod_jn_option( )

Fonction = coordinateur de la modification des paramètres d'utilisation du réseau : cas où le régime est jour-nuit.

mod_util( )

Fonction = saisie au terminal d'une nouvelle valeur pour le type d'accès aux fichiers, avec mise à jour de l'écran créé par écran-option().

Résultat = util-uniforme : entier.
Postcondition = util-uniforme égale 0 ou 1 selon que l'accès aux fichiers est uniforme ou limité au plus proche exemplaire.

mod_delai( )

Fonction = saisie au terminal d'une nouvelle valeur pour le délai assuré par le réseau, avec mise à jour de l'écran créé par écran-option().

Résultat = délai-réseau : réel.
Postcondition = 0 < = délai-réseau.

mod-long( )

Fonction = saisie au terminal d'une nouvelle valeur pour la longueur d'un segment, avec mise à jour de l'écran créé par écran-option().
Résultat = long-segment : entier.
Postcondition = 0 < = long-segment.

modl_tmax( )

Fonction = saisie au terminal d'une nouvelle valeur pour le délai maximal autorisé, avec mise à jour de l'écran créé par écran-option() : cas où le régime est permanent.

Résultat = tmax : réel.
Postcondition = 0 < = tmax.

mod1_prix( )

Fonction = saisie au terminal d'une nouvelle valeur pour le prix d'un segment, avec mise à jour de l'écran créé par écran-option() : cas où le régime est permanent.
Résultat = prix-segment : réel.
Postcondition = 0 < = prix-segment.

mod2_tmax( )

Fonction similaire à mod1-tmax mais dans le cas où le régime est jour-nuit.

mod2_prix( )

Fonction = saisie au terminal d'une nouvelle valeur pour le prix d'un segment le jour, avec mise à jour de l'écran créé par écran-option().

Résultat = prix-jour-segment : réel.
Postcondition = 0 < = prix-jour-segment.

mod3_prix( )

Fonction = saisie au terminal d'une nouvelle valeur pour le prix d'un segment la nuit, avec mise à jour de l'écran créé par écran-option().

Résultat = prix-nuit-segment : réel.
Postcondition = 0 < = prix-nuit-segment.

mod_jour( )

Fonction = saisie au terminal d'une nouvelle valeur pour la durée du jour, avec mise à jour de l'écran créé par écran-option().

Arguments = durée-jour : réel.
durée-nuit : réel.
Précondition = 0 < = durée-jour + durée-nuit < = 86 400.

Résultat = durée-jour : réel.
Postcondition = . 0 < = durée-jour.
. durée-jour + durée-nuit < = 86 400.

mod_nuit( )

Fonction = saisie au terminal d'une nouvelle valeur pour la durée de la nuit, avec mise à jour de l'écran créé par écran-option().

Arguments = durée-jour : réel.
durée-nuit : réel.
Précondition = 0 < = durée-jour + durée-nuit < = 86 400.

Résultat = durée-nuit : réel.
Postcondition = . 0 < = durée-nuit.
. durée-jour + durée-nuit < = 86 400

modl_regime(ch)

Fonction = saisie au terminal d'une nouvelle valeur pour le type de régime, avec activation récursive du module permettant de modifier les paramètres dans le cas où le nouveau régime est jour-nuit (mod-jn-option()) : cas où le régime est permanent.

Résultat = régime : entier.
ch : pointeur vers un entier.
Postcondition = . régime = 0 ou 1.
. ch égale 0 si le régime a été modifié pour devenir jour-nuit.

mod2_regime(ch)

Fonction = saisie au terminal d'une nouvelle valeur pour le type de régime, avec activation récursive du module permettant de modifier les paramètres dans le cas où le nouveau régime est permanent (mod-perm-option()) : cas où le régime est jour-nuit.

Résultat = régime : entier.
ch : pointeur vers un entier.

Postcondition = . régime = 0 ou 1.
. ch égale 0 si le régime a été modifié pour devenir permanent.

FICHIER MEMROUTE . C

Ce fichier contient les procédures nécessaires à la détermination du routage dans le réseau par calcul et/ou par introduction au terminal par le concepteur.

main(argc,argv)

Fonction = entrée au sous-programme réalisant la détermination du routage avec stockage sur fichier.

Arguments = argc : entier.
argv : tableau de pointeurs.

Précondition = . argv contient 2 paramètres : le niveau atteint (3) et le nom du fichier où il faut stocker les données introduites.

ent_route( )

Fonction = coordinateur de la détermination du routage avec :
. calcul automatique d'un routage de longueur minimale,
. contrôle du caractère connexe du réseau avec possibilité de correction du réseau s'il ne l'est pas,
. offre de la possibilité d'introduire un routage.

Résultat = retour de la procédure : entier.

Postcondition = le retour égale 1 si le réseau est connexe et 0 sinon.

ent2_route( )

Fonction = coordinateur de la correction du réseau s'il n'est pas connexe.

Argument = bidir : entier.
Précondition = bidir = 0 ou 1.

visu_uni( )

Fonction = visualisation par groupe de 5, des lignes mémorisées : cas unidirectionnel.

Arguments = tablig : tableau de structures ligne.
nbl : entier.
Précondition = 0 < = nbl.

visu_bi( )

Fonction similaire à visu-uni() : cas bidirectionnel.

ecran_lig( )

Fonction = affichage à l'écran d'une grille pour visualiser les différents paramètres des lignes.

visu_err( )

Fonction = visualisation des inaccessibilités dans le réseau, détectées par la procédure "rech-hamilton()".

Arguments = tabstat : tableau de structures statnoeud.
err : tableau entier.
nbs : entier.
Précondition = . 0 < = nbs.
. err (i)(j) = 1 si on ne peut accéder à la station n° j à partir de la station n° i,
0 sinon.

rech_hamilton( )

Fonction = détermination automatique du routage entre les stations du réseau, avec contrôle du caractère connexe du réseau.

Arguments = mat-ligne : tableau entier.
mat-dist : tableau réel
(voir table-ligne()).
nbs, nbl, nbn : entier.
Précondition = 0 < = nbs, nbl, nbn.

Résultats = retour de la fonction.
err : tableau entier.
routing : tableau entier.
Postcondition = . le retour égale 1 si le réseau est connexe et 0 sinon.
. err (i)(j) = 1 si on ne peut accéder à la station n°j à partir de la station n°i, 0 sinon.
. routing (i)(j)(k) = 1 si le chemin entre les stations n°j et n°k passe par la ligne n°i.

tout_ens(ens,n)

Fonction = vérification de la valeur 1 pour les n éléments de ens.

Arguments = ens : tableau entier.
n : entier.
Précondition = . 0 < = n.
. pour tout 1 < = i < = n, ens(i) = 0 ou 1.

Résultat = retour de la fonction.
Postcondition = le retour vaut 1 si pour tout0 < = i < = n, ens(i) = 1 et 0 sinon.

enr_table_routage( )

Fonction = coordinateur de l'introduction au terminal d'un routage entre les stations, avec visualisation du routage mémorisé antérieurement.

Arguments = routing : tableau entier.
nbs : entier.
bidir : entier.
nbl : entier.
ex-route-oblig : tableau entier.
Précondition = . 0 < = nbs, nbl.
. bidir = 0 ou 1.
. ex-route-oblig = 0 ou 1.

Résultats = routing : tableau entier.
ex-route-oblig : tableau entier.
Postcondition = . routing contient le routage entre les stations (calculé ou forcé).
. ex-route-oblig (i)(j) = 1 si le routage entre les stations n°i et j a été forcé,
0 sinon.

lire_origine(origine,n)

Fonction = saisie au terminal du nom de la station origine du routage à modifier.

Argument = n : entier.
Précondition = 0 < = n.

Résultat = origine : chaîne de caractères.
Postcondition = . origine contient le nom d'une station existante ou le caractère '*'.

lire_destination(dest,org,n)

Fonction = saisie au terminal du nom de la station destination du routage à modifier.

Arguments = org : chaîne de caractères.
n : entier.
Précondition = . 0 < = n.
. org : nom de la station origine du routage à modifier.

Résultat = dest : chaîne de caractères.
Postcondition = dest contient le nom d'une station existante mais différent de org ou le caractère '*'.

visu_route(r,pos,n1,n2)

Fonction = visualisation du routage défini entre deux stations.

Arguments = r : tableau entier.
pos : entier.
n1 : entier.
n2 : entier.
tabstat : tableau de structures statnoeud.
tablig : tableau de structures ligne.
nbs : entier.
Précondition = . 0 < = nbs.
. 1 < = n1, n2 < = nbs.
. r contient la table du routage.

intro_route(orig,dest)

Fonction = introduction d'un nouveau chemin entre deux stations.

Arguments = orig : chaîne de caractères.
dest : chaîne de caractères.
nbs, nbl,nbn : entier.
route : tableau entier.
Précondition = . 0 < = nbs, nbl, nbn.
. orig est le nom d'une station existante.
. dest est le nom d'une station existante.
. route contient les chemins déjà forcés.
Résultats = route : tableau entier.
retour de la fonction : entier.
Postcondition = . le retour vaut 1 si le routage entre les stations orig et dest a été modifié dans route, ou 0 sinon.

lire_suiv(suiv,déjà,rt,prec,pos)

Fonction = saisie au terminal d'une station appartenant à un chemin, avec contrôle de cohérence de l'introduction.

Arguments = déjà : tableau entier.
prec : chaîne de caractères.
rt : tableau entier.
pos : entier.

Précondition = . 0 < = pos < = 10.
. prec contient le noeud d'une station ou d'un noeud existant.
. déjà (i) = 1 si la station n°i participe déjà au chemin en cours d'introduction

Résultat = rt : tableau entier.
déjà : tableau entier.
suiv : chaîne de caractères.

Postcondition = . suiv contient le nom d'une station existante, que l'on peut relier au chemin antérieurement introduit, qui n'y participe pas déjà ou le caractère '*'.
. soit i le numéro de la station suiv
rt en sortie = rt en entrée sauf rt(i) qui vaut 1 et déjà(i) = 1.

ajout_ligne( )

Fonction = coordinateur de l'ajout d'une ligne au réseau.

Arguments = nbl : entier.
bidir : entier.

Précondition = . 0 < = nbl.
. bidir = 0 ou 1.

ajout_uni( )

Fonction = ajout de la description d'une ligne : cas unidirectionnel (avec itération).

Argument = nbl : entier.
Précondition = 0 < = nbl.

Résultat = tablig : tableau de structures ligne.
nbl : entier.
Postcondition = tablig contient des éléments distincts 2 à 2.

cont_aj(max)

Fonction = proposition de continuer à introduire de nouvelles lignes si c'est possible.

Argument = max : entier.
nbl : entier.
Précondition = 0 < = max, nbl.
Résultat = retour de la fonction : caractère.
Postcondition = le retour vaut 'y' ou 'n'.

ajout_bi( )

Fonction similaire à ajout-uni() : cas bidirectionnel.

lec_stat(no1,no2)

Fonction = saisie au terminal des noms des stations origine et destination d'une ligne du réseau.

Arguments = nbn : entier.
nbs : entier.
Précondition = 0 < = nbs, nbn.

Résultats = no1 : chaîne de caractères.
no2 : chaîne de caractères.
retour de la fonction.
Postcondition = . no1 (no2) contient le nom de la station origine (destination) de la ligne.
. le retour égale 3 si no1 et no2 sont à prendre en compte et 1 sinon.

table_ligne( )

Fonction = construction de la table des distances entre les stations reliées par une ligne et de la table du numéro de la ligne entre chaque paire de stations.

Arguments = tablig : tableau de structures ligne.
nbl : entier.
nbs : entier.
nbn : entier.

Précondition = 0 < = nbs, nbl, nbn.

Résultats = mat-ligne : tableau entier.
mat-dist : tableau réel.

Postcondition = soient num1 et num2 les numéros de deux stations,
. mat-ligne(num1)(num2) = i si la ligne i relie num1 à num2, 0 sinon.
. mat-dist (num1)(num2) = la longueur de la ligne les reliant si elle existe,
0 si num1 = num2,
999999 sinon.

FICHIER MEMSAUVE . C

Ce fichier contient les procédures nécessaires à l'écriture et ou à la lecture d'un fichier de données.

écrire_réseau(nom)

Fonction = coordinateur de l'écriture des informations présentes dans un fichier de données.

Argument = nom : chaîne de caractères.
Précondition = /

écr_stat(fp)

Fonction = écriture du tableau des stations sur un fichier de données.

Arguments = fp : descripteur d'un fichier.
tabstat : tableau de structures station.
nbs : entier.
Précondition = . le descripteur "fp" symbolise un fichier ouvert.
. 0 < = nbs.

ecr_noeud(fp)

Fonction = écriture du tableau des noeuds sur un fichier de données.

Arguments = fp : descripteur d'un fichier.
tabstat : tableau de structures station.
nbs, nbn : entier.
Précondition = . le descripteur "fp" symbolise un fichier ouvert.
. 0 < = nbs.
. 0 < = nbn.

ecr_ligne(fich)

Fonction = écriture du tableau des lignes sur un fichier de données et d'informations annexes.

Arguments = fich : descripteur d'un fichier.
tablig : tableau de structures ligne.
mat-dist : tableau réel.
mat-ligne : tableau entier.
nbs, nbn, nbl, bidir : entier.

Précondition = . le descripteur "fich" symbolise un fichier ouvert.
. 0 < = nbs.
. 0 < = nbn.
. 0 < = nbl.
. bidir = 0 ou 1.

ecr_route(fich)

Fonction = écriture du routage à travers le réseau sur un fichier de données.

Arguments = fich : descripteur d'un fichier.
routing : tableau entier.
nbs, nbl : entier.

Précondition = . le descripteur "fich" symbolise un fichier ouvert.
. 0 < = nbs.
. 0 < = nbl.

ecr_fichier(fich)

Fonction = écriture du tableau des fichiers sur un fichier de données.

Arguments = fich : descripteur d'un fichier.
tabfich : tableau de structures fichier.
nbf : entier.

Précondition = . le descripteur "fich" symbolise un fichier ouvert.
. 0 < = nbf.

ecr_capac(fich)

Fonction = coordinateur de l'écriture des tableaux de capacités sur un fichier de données.

Argument = fich : descripteur d'un fichier.
Précondition = . le descripteur "fich" symbolise un fichier ouvert.

ecr_lcap(fich)

Fonction = écriture du tableau des capacités louées sur un fichier de données.

Arguments = fich : descripteur d'un fichier.
tabloue : tableau de structures cap-loue
nbcaploue : entier.
Précondition = . le descripteur "fich" symbolise un fichier ouvert.
. 0 nbcaploue.

ecr_nlcap(fich)

Fonction = écriture du tableau des capacités non louées sur un fichier de données.

Arguments = fich : descripteur d'un fichier.
tabnloue : tableau de structures cap-non-loue.
nbcapnloue : entier.
Précondition = . le descripteur "fich" symbolise un fichier ouvert.
. 0 < = nbnloue.

ecr_ldcs(fich)

Fonction = écriture du tableau des capacités DCS louées sur un fichier de données.

Arguments = fich : descripteur d'un fichier
tabdcsl : tableau de structures dcs-loue.
nbdcsl : entier.

Précondition = . le descripteur "fich" symbolise un fichier ouvert.
. 0 < = nbdcsl.

ecr_nldcs(fich)

Fonction = écriture du tableau des capacités DCS non louées sur un fichier de données.

Arguments = fich : descripteur d'un fichier.
tabdcsnl : tableau de structures dcs-non-loue.
nbdcsnl : entier.

Précondition = . le descripteur "fich" symbolise un fichier ouvert.
. 0 < = nbdcsnl.

ecr_cf(fich)

Fonction = écriture du tableau des coûts des capacités et des coûts fixes.

Arguments = fich : descripteur d'un fichier.
cap-tot : tableau de structures cap.
prix-fx-line : tableau réel.
nbcap, nbl, nbcaploue : entier.

Précondition = . le descripteur "fich" symbolise un fichier ouvert.
. 0 < = nbcap, nbl, nbcaploue.

ecr_para(fich)

Fonction = écriture des paramètres du réseau sur un fichier de données.

Arguments = fich : descripteur d'un fichier.
util-uniforme : entier.
délai-réseau : réel.
long-segment : entier.
tmax : réel.
régime : entier.
prix-segment : réel.
prix-nuit-segment : réel.
prix-jour-segment : réel.
durée-jour : réel.
durée-nuit : réel.

Précondition = . util-uniforme = 0 ou 1.
. 0 < délai-réseau.
. 0 < long-segment.
. 0 < tmax.
. régime = 0 ou 1.
. 0 < = prix-segment.
. 0 < = prix-nuit-segment.
. 0 < = prix-jour-segment.
. 0 < = durée-jour.
. 0 < = durée-nuit.
. le descripteur "fich" symbolise un fichier ouvert.

ecr_trafic(fich)

Fonction = écriture des paramètres du trafic sur un fichier de données.

Arguments = fich : descripteur d'un fichier.
mu_._. : réel.
régime : entier.
Précondition = . le descripteur "fich" symbolise un fichier ouvert.
. 0 < = mu_._.
. régime = 0 ou 1.

ecr2_trafic(fich)

Fonction = écriture des tableaux du trafic en régime permanent sur un fichier de données.

Arguments = fich : descripteur d'un fichier.
u-perm : tableau réel.
uprim-perm : tableau réel.
v-perm : tableau réel.
vprim-perm : tableau réel.
nbs, nbf : entier.
Précondition = . le descripteur "fich" symbolise un fichier ouvert.
. 0 < = u-perm, uprim-perm, v-perm, vprim-perm.
. 0 < = nbs, nbf.

ecr3_trafic(fich)

Fonction = écriture des tableaux du trafic en régime jour-nuit sur un fichier de données.

Arguments = fich : descripteur d'un fichier.
u-jour : tableau réel.
u-nuit : tableau réel.
v-jour : tableau réel.
v-nuit : tableau réel.
uprim-jour : tableau réel.
uprim-nuit : tableau réel.
vprim-jour : tableau réel.
vprim-nuit : tableau réel.
nbs, nbf : entier.

Précondition = . le descripteur "fich" symbolise un fichier ouvert.
. 0 < = u-jour, u-nuit, v-jour, v-nuit, uprim-jour, uprim-nuit, vprim-jour, vprim-nuit.
. 0 < = nbs, nbf.

ecr_fiabilité(fich)

Fonction = écriture du tableau des fiabilités sur un fichier de données.

Arguments = fich : descripteur d'un fichier.
fiabilité : tableau réel.
nbs : entier.

Précondition = . le descripteur "fich" symbolise un fichier ouvert.
. 0 < = fiabilité.
. 0 < = nbs.

ecr_emplacement(fich)

Fonction = écriture du tableau des contraintes d'emplacement sur un fichier de données.

Arguments = fich : descripteur d'un fichier.
emplacement : tableau entier.
nbs, nbf : entier.

Précondition = . le descripteur "fich" symbolise un fichier ouvert.
. emplacement = 0 ou 1 ou 2.
. 0 < = nbs, nbf.

ecr_alloc(fich)

Fonction = écriture de l'allocation du fichiers sur un fichier de données.

Arguments = fich : descripteur d'un fichier.
allocation-fichier : tableau entier.
valide : entier.
nbs, nbf : entier.

Précondition = . le descripteur "fich" symbolise un fichier ouvert.
. allocation-fichier = 0 ou 1.
valide = 0 ou 1.
. 0 < = nbs, nbf.

Correspondant à ces fonctions d'écriture, il existe des fonctions de lecture. A chaque fonction portant le nom "ecr_.", est associée une fonction portant le nom "lire_.".

Dans un souci de concision, nous ne spécifierons pas les fonctions de lecture dans le présent ouvrage. Pour obtenir leur spécification, il suffit de suivre la démarche suivante :

1) dans la rubrique "Fonction", remplacer le mot "écriture" par "lecture";

2) dans la rubrique "Argument", ne conserver que la variable "fich";

3) dans la rubrique "Précondition", ne conserver que la condition sur "fich";

4) transférer toutes les variables présentes dans la rubrique "Argument", sauf "fich", dans une rubrique "Résultat".

Note : pour la fonction "lire-réseau", n'appliquer que le point 1).

FICHIER MEMSTAT . C

Ce fichier contient les procédures nécessaires à l'introduction des stations du réseau à étudier et leur visualisation.

main(argc,argv)

Fonction = entrée au sous-programme réalisant l'introduction des stations avec stockage sur fichier.

Arguments = argc : entier.
argv : tableau de pointeurs.

Précondition = argv contient 2 paramètres : le niveau atteint (0) et le nom du fichier où il faut stocker les données introduites.

ent_stat( )

Fonction = procédure d'accueil à la fonction d'introduction et test de caractère cohérent du réseau introduit (au moins deux stations).

Résultat = retour de la procédure : entier.

Postcondition = le retour égale 1 si au moins deux stations ont été introduites et 0 sinon.

station( )

Fonction = coordinateur de la logique d'introduction (cfr 3.1.1).

intro_stat( )

Fonction = saisie au terminal de la description des stations du réseau.

Argument = SMAX : entier.
Précondition = 0 < SMAX.

Résultats = tabstat : tableau de structures statnoeud.
nbs : entier.
Postcondition = . tabstat contient nbs stations 2 à 2 distinctes.
. 0 < = nbs < = SMAX - 1.

ecran_stat( )

Fonction = affichage à l'écran d'une grille permettant d'introduire les données de type station.

visu_stat( )

Fonction = visualisation par groupe de 5, des stations mémorisées.

Arguments = tabstat : tableau de structures statnoeud.
nbs : entier.
Postcondition = 0 < = nbs.

mod_stat_intro( )

Fonction = visualisation par groupe de 5, des stations mémorisées et possibilité de modification des descriptions des stations affichées à l'écran.

Arguments = tabstat : tableau de structures statnoeud.
nbs : entier.
Précondition = . 0 < = nbs.
. deux éléments de tabstat ne peuvent avoir le même nom.

Résultat = tabstat : tableau de structures statnoeud.
Postcondition = tabstat contient nbs stations 2 à 2 distinctes.

quest_modif( )

Fonction = proposition de visualisation des stations introduites.

Résultat = retour de la fonction : caractère.
Postcondition = le retour vaut 'y' ou 'n'.

FICHIER MEMSTOCK . C

Ce fichier contient les procédures nécessaires à la vérification des contraintes de stockage.

main(argc,argv)

Fonction = entrée au sous-programme réalisant la vérification des contraintes de stockage.

Arguments = argc : entier.
argv : tableau de pointeurs.

Précondition = argv contient deux paramètres : le niveau atteint (11) et le nom du fichier où est stockée la description du réseau.

stock( )

Fonction = coordinateur de la vérification des contraintes de stockage, de la visualisation des problèmes éventuels et de leur résolution.

Résultat = retour de la fonction : entier.
niveau : entier.

Postcondition = . le retour vaut 1 si les contraintes de stockage sont vérifiées et 0 sinon.
. niveau vaut 12 si les contraintes de stockage sont vérifiées et 11 sinon.

sfaisabilité( )

Fonction = algorithme de vérification des contraintes de stockage (cfr(PIC.84)).

Arguments = allocation-fichier : tableau entier.
tabfich : tableau de structures fichier.
tabstat : tableau de structures statnoeud.
nbs, nbf : entier.

Précondition = . 0 < = nbs, nbf.
. pour tout i et j, allocation-fichier(i)(j) = 1 si le fichier j est alloué à la station i, et 0 sinon.
. tabstat contient nbs stations distinctes.
. tabfich contient nbf fichiers distincts.

Résultat = : retour de la fonction : entier.

Postcondition = le retour vaut 1 si les contraintes de stockage sont vérifiées et 0 sinon.

visu_pb(pb)

Fonction = visualisation des stations ne vérifiant pas les contraintes de stockage.

Arguments = pb : tableau entier.
tabstat : tableau de structures statnoeud.
nbs : entier.

Précondition = . 0 < = nbs.
. tabstat contient nbs stations distinctes.
. pour tout i, pb(i) = 1 si la station numéro i ne vérifie pas la contrainte de stockage et 0 sinon.

quest_modif( )

Fonction = proposition de divers traitements lorsque les contraintes de stockage ne sont pas vérifiées : visualiser en détail les problèmes, modifier l'allocation de fichiers courante, quitter.

Résultat = retour de la fonction : caractère.

Postcondition = selon le traitement choisi, le retour vaut 'd' pour détailler, 'm' pour modifier et 'q' pour quitter.

ques2_modif( )

Fonction = proposition de modification de l'allocation de fichiers courante.

Résultat = retour de la fonction : caractère.
Postcondition = le retour vaut 'm' pour modifier ou 'q' pour quitter.

detail_pb(pb)

Fonction = pour chaque station ne vérifiant pas sa contrainte de stockage, visualisation de divers paramètres : nom de la station, sa capacité de stockage, le dépassement de la capacité, la liste des fichiers alloués à cette station avec leur longueur et leur contrainte d'emplacement.

Arguments = pb : tableau entier.
allocation-fichier : tableau entier.
emplacement : tableau entier.
tabfich : tableau de structures fichier.
tabstat : tableau de structures station.
nbs, nbf : entier.

Précondition = . 0 < = nbs, nbf.
. tabstat contient nbs stations distinctes.
. tabfich contient nbf fichiers distincts.
. pour tout i et j, emplacement(i)(j) = 0, 1 ou 2 selon que le fichier j est soumis à aucune contrainte, à une contrainte de présence obligatoire ou de présence interdite.

Pour les procédures mentionnées ci-dessous, consulter les spécifications de la rubrique "FICHIER MEMALLOC. C"

mod_alloc( )
m_sta_alloc( )
m2_stat_alloc( )
aj_fich(i)
an_fich(i)
mod_ecran(i)
m_fich_alloc( )
m2_fich_alloc(i)
aj_stat(i)
an_stat(i)
mod2_ecran(i)
ok_an(i)
ques_alloc( )

FICHIER MEMTRAF . C

Ce fichier contient les procédures nécessaires à l'introduction du trafic dans le réseau et sa visualisation.

main(argc,argv)

Fonction = entrée au sous-programme réalisant l'introduction du trafic avec stockage sur fichier.

Arguments = argc : entier.
argv : tableau de pointeurs.

Précondition = argv contient deux paramètres : le niveau atteint (7) et le nom du fichier où il faut stocker les données introduites.

trafic( )

Fonction = coordinateur de la logique du module.

intro_trafic( )

Fonction = coordinateur de l'introduction du trafic en fonction du caractère permanent ou non du trafic.

Argument = régime : entier.

Précondition = régime égale 0 ou 1 selon que le régime est permanent ou jour-nuit.

mod_trafic( )

Fonction = coordinateur de la modification du trafic introduit en fonction du caractère permanent ou non du trafic.

Argument = régime : entier.
Précondition = régime égale 0 ou 1 selon que le régime est permanent ou jour-nuit.

ecran1_trafic( )

Fonction = affichage d'un texte à l'écran (cfr listing).

lec_p_long( )

Fonction = saisie au terminal des longueurs des messages de chacun des quatre types de trafic : cas du régime permanent.

Résultats = mu-q-p : réel.
mu-rq-p : réel.
mu-u-p : réel.
mu-ru-p : réel.
Postcondition = . 0 < = mu-q-p, mu-rq-p,mu-u-p, mu-ru-p.

lec_val(line,col)

Fonction = saisie au terminal d'un nombre réel de six positions, différent de 0.0 .

Arguments = line : entier.
col : entier.
Précondition = . 1 < = line < = 24.
. 1 < = col < = 74.

Résultat = retour de la fonction : réel.
Postcondition = le retour est différent de 0 et a six positions.

l_jn_long( )

Fonction = saisie au terminal des longueurs des messages de chacun des quatre types de trafic : cas du régime jour-nuit.

Résultats = mu-q-j : réel.
mu-q-n : réel.
mu-rq-j : réel.
mu-rq-n : réel.
mu-u-j : réel.
mu-u-n : réel.
mu-ru-j : réel.
mu-ru-n : réel.
Postcondition = 0 < mu-q-j, mu-q-n, mu-rq-j, mu-rq-n,mu-u-j, mu-u-n, mu-ru-j, mu-ru-n.

mod_p_long( )

Fonction = visualisation des longueurs de messages introduites avec possibilité de modification : cas du régime permanent.

Arguments = mu-q-p : réel.
mu-rq-p : réel.
mu-u-p : réel.
mu-ru-p : réel.
Précondition = 0 < mu-q-p, mu-rq-p,mu-u-p, mu-ru-p.

Résultats = mu-q-p : réel.
mu-rq-p : réel.
mu-u-p : réel.
mu-ru-p : réel.
Postcondition = 0 < mu-q-p, mu-rq-p, mu-u-p, mu-ru-p.

ml_p_long( )

Fonction = saisie au terminal d'une nouvelle valeur pour la longueur des messages du "query traffic" avec mise à jour de l'écran : cas du régime permanent.

Résultat = mu-q-p : réel.
Postcondition = 0 < mu-q-p.

m2_p_long( )

Fonction similaire à m1-p-long() mais pour le "query return traffic".

m3_p_long( )

Fonction similaire à m1-p-long() mais pour le "update traffic".

m4_p_long( )

Fonction similaire à m1-p-long() mais pour le "update return traffic".

m_jn_long( )

Fonction = visualisation des longueurs de messages introduites avec possibilité de modification : cas du régime jour-nuit.

Arguments = mu-q-j : réel.
mu-q-n : réel.
mu-rq-j : réel.
mu-rq-n : réel.
mu-u-j : réel.
mu-u-n : réel.
mu-ru-j : réel.
mu-ru-n : réel.

Précondition 0 < mu-q-j, mu-q-n, mu-rq-j, mu-rq-n, mu-u-j, mu-u-n, mu-ru-j, mu-ru-n.

Résultats = mu-q-j : réel.
mu-q-n : réel.
mu-rq-j : réel.
mu-rq-n : réel.
mu-u-j : réel.
mu-u-n : réel.
mu-ru-j : réel.
mu-ru-n : réel.

Postcondition = 0 < mu-q-j, mu-q-n, mu-rq-j, mu-rq-n, mu-u-j, mu-u-n, mu-ru-j, mu-ru-n.

m1_jn_long( )

Fonction = saisie au terminal d'une nouvelle valeur pour la longueur des messages du "query traffic" de jour avec mise à jour de l'écran : cas du régime jour-nuit.

Résultat = mu-q-j : réel.
Postcondition = 0 < mu-q-j.

m2_jn_long( )

Fonction similaire à m1-jn-long() mais pour le "query traffic" de nuit.

m3_jn_long( )

Fonction similaire à m1-jn-long() mais pour le "query return traffic" de jour.

m4_jn_long( )

Fonction similaire à m1-jn-long() mais pour le "query return traffic" de nuit.

m5_jn_long( )

Fonction similaire à m1-jn-long() mais pour le "update traffic" de jour.

m6_jn_long( )

Fonction similaire à m1-jn-long() mais pour le "update traffic" de nuit.

m7_jn_long( )

Fonction similaire à m1-jn-long( ) mais pour le "update return traffic" de jour.

m8_jn_long( )

Fonction similaire à m1-jn-long() mais pour le "update return traffic" de nuit.

ecran2-trafic( )

Fonction = affichage d'un texte à l'écran (cfr listing).

lec_p_traf( )

Fonction = saisie au terminal des volumes du trafic à travers le réseau : cas où le régime est permanent.

Arguments = tabstat : tableau de structures statnoeud.
tabfich : tableau de structures fichier.
nbs, nbf : entier.

Précondition = . 0 < nbs, nbf.
. tabstat contient nbs stations distinctes.
. tabfich contient nbf fichiers distincts.

Résultats = u-perm : tableau réel.
v-perm : tableau réel.
uprim-perm : tableau réel.
vprim-perm : tableau réel.

Postcondition = pour tout i et j, u-perm(i)(j), vperm(i)(j), uprim-perm(i)(j), vprim-perm(i)(j) > = 0.

ecran3_trafic( )

Fonction = affichage d'un texte à l'écran (cfr listing).

1_jn_traf( )

Fonction = saisie au terminal des valeurs du trafic à travers le réseau : cas où le régime est jour-nuit.

Arguments = tabstat : tableau de structures statnoeud.
tabfich : tableau de structures fichier.
nbs, nbf : entier.

Précondition = . 0 < = nbs, nbf.
. tabstat contient nbs stations distinctes.
. tabfich contient nbf fichiers distincts.

Résultats = u-jour : tableau réel.
u-nuit : tableau réel.
v-jour : tableau réel.
v-nuit : tableau réel.
uprim-jour : tableau réel.
uprim-nuit : tableau réel.
vprim-jour : tableau réel.
vprim-nuit : tableau réel.

Postcondition = pour tout i et j, u-jour(i)(j), u-nuit(i)(j), v-jour(i)(j), v-nuit(i)(j), uprim-jour(i)(j), uprim-nuit(i)(j), vprim-jour(i)(j), vprim-nuit(i)(j) > = 0.

mod_p_traf( )

Fonction = visualisation des valeurs des quatre types de trafic pour chaque paire (station, fichier) avec possibilité de modification : cas du régime permanent.

Arguments = tabstat : tableau de structures statnoeud.
tabfich : tableau de structures fichier.
u-perm : tableau réel.
v-perm : tableau réel.
uprim-perm : tableau réel.
vprim-perm : tableau réel.
nbs, nbf : entier.

Précondition = . 0 < = nbs, nbn.
. pour tout i et j, u-perm(i)(j), v-perm(i)(j), uprim-perm(i)(j), vprim-perm(i)(j) > = 0.

mod2_p_traf(i,j)

Fonction = sélection du type de trafic à modifier entre une station et un fichier et activation du module ad hoc de modification : cas du régime permanent.

Arguments = i : entier.
j : entier.
Précondition = . 1 < = i < = nbs.
. 1 < = j < = nbf.

m1_p_traf(i,j)

Fonction = saisie au terminal d'une nouvelle valeur pour le "query traffic" entre la station numéro i et le fichier numéro j avec mise à jour de l'écran : cas du régime permanent.

Arguments = i,j : entier.
u-perm : tableau réel.
Précondition = . 1 < = i < = nbs.
. 1 < = j < = nbf.
. pour tout k et l, u-perm(k)(l) > = 0.

Résultat = u-perm : tableau réel.
Postcondition = . u-perm en résultat égale u-perm en argument
. sauf éventuellement u-perm(i)(j).
. u-perm(i)(j) > = 0.

m2_p_traf(i,j)

Fonction similaire à m1-p-traf mais pour le "query return traffic".

m3_p_traf(i,j)

Fonction similaire à m1-p-traf mais pour le "update traffic".

m4_p_traf(i,j)

Fonction similaire à m2-p-traf mais pour le "update return traffic".

m_jn_traf( )

Fonction = visualisation des valeurs des quatre types de trafic pour chaque paire (station, fichier) avec possibilité de modification : cas du régime jour-nuit.

Arguments = tabstat : tableau de structures statnoeud.
tabfuch : tableau de structures fichier.
u-jour : tableau réel.
u-nuit : tableau réel.
v-jour : tableau réel.
v-nuit : tableau réel.
uprim-jour : tableau réel.
uprim-nuit : tableau réel.
vprim-jour : tableau réel.
uprim-nuit : tableau réel.
nbs, nbf : entier.

Précondition = . 0 < = nbs, nbf.
. tabstat contient nbs stations distinctes.
. tabfich contient nbf fichiers distincts.

m2_jn_traf(i,j)

Fonction = sélection du type de trafic à modifier entre une station et un fichier et activation du module ad hoc de modification : cas du régime jour-nuit.

Arguments = i : entier.
j : entier.

Précondition = . 1 < = i < = nbs.
. 1 < = j < = nbf.

m21_jn_traf(i,j)

Fonction = saisie au terminal d'une nouvelle valeur pour le "query traffic" de jour entre la station numéro i et le fichier numéro j avec mise à jour de l'écran : cas du régime jour-nuit.

Arguments = i,j : entier.
u-jour : tableau réel.

Précondition = . 1 < = i < = nbs.
. 1 < = j < = nbf.
. pour tout k et l, u-jour(k)(l) > = 0.

Résultat = u-jour : tableau réel.
Postcondition = . u-jour en résultat égale u-jour en argument sauf éventuellement u-jour(i)(j).
. u-jour(i)(j) > = 0.

m22_jn_traf(i,j)

Fonction similaire à m21-jn-traf mais pour le "query traffic" de nuit.

m23_jn_traf(i,j)

Fonction similaire à m21-jn-traf mais pour le "query return traffic" de jour.

m24_jn_traf(i,j)

Fonction similaire à m21-jn-traf mais pour le "query return traffic" de nuit.

m25_jn_traf(i,j)

Fonction similaire à m21-jn-traf mais pour le "update traffic" de jour.

m26_jn_traf(i,j)

Fonction similaire à m21-jn-traf mais pour le "update traffic" de nuit.

m27_jn_traf(i,j)

Fonction similaire à m21-jn-traf mais pour le "update return traffic" de jour.

m28_jn_traf(i,j)

Fonction similaire à m21-jn-traf mais pour le "update return traffic" de nuit.

quest_traf_mod( )

Fonction = proposition de visualisation du trafic introduit.

Résultat = retour de la fonction : caractère.
Postcondition = le retour vaut 'y' ou 'n'.

FICHIER TECH . C

Ce fichier contient une série de fonctions d'utilité générale et permettant une gestion d'écran d'un terminal VT 100.

set_pos_(line,col)

Fonction = positionnement du curseur à l'écran, à la ligne numéro "line" et à la colonne numéro "col".

Arguments = line : entier.
col : entier.

Précondition = . 1 < = line < = 24.
. 1 < = col < = 80.

clear(x,y,lim)

Fonction = apparition de "lim" lignes vierges à l'écran, à partir de la position (x,y).

Arguments = x : entier.
y : entier.
lim : entier.

Précondition = . 1 < = x < = 24.
. 1 < = y < = 80.
. 1 < = lim < = 24 - x.

scr_blc( )

Fonction = apparition d'un écran vierge.

getline(s,lim)

Fonction = saisie au terminal d'une chaîne de "lim" - 1 caractères au maximum.

Argument = lim : entier.
Précondition = 2 < = lim.

Résultat = s : chaîne de caractères.
Postcondition = s est composée d'au plus "lim" - 1 caractères, du caractère "RETURN" et du caractère "/0".

getfich(s,lim)

Fonction = saisie au terminal d'une chaîne de "lim" - 1 caractères au maximum, en négligeant le "RETURN".

Argument = lim : entier.
Précondition = 2 < = lim.

Résultat = s : chaîne de caractères.
Postcondition = s est composé d'au plus "lim" - 1 caractères et du caractère "/0".

getdigit(s,lim)

Fonction = saisie au terminal d'une chaîne de "lim" - 1 caractères numériques au maximum représentant un réel sous la forme xx.yy

Argument = lim : entier.
Précondition = 2 < = lim.

Résultat = s : chaîne de caractères.
Postcondition = s est composé d'au plus "lim" - 1 caractères numériques (y compris éventuellement le ".") et du caractère "/0".

getin(s,lim)

Fonction = saisie au terminal d'une chaîne de "lim" - 1 caractères numériques au maximum représentant un entier.

Argument = lim : entier.
Précondition = 2 < = lim.

Résultat = s est composé d'au plus "lim" - 1 caractères numériques et du caractère "/0".

getproba(s,lim)

Fonction = saisie au terminal d'une chaîne de "lim" - 1 caractères numériques au maximum représentant un réel entre 0 et 1 non compris.

Argument = lim : entier.
Précondition = 2 < = lim.

Résultat = s : chaîne de caractères.
Postcondition = s est composé des caractères "0", ".", d'au plus "lim" - 3 caractères numériques et du caractère "/0".

q_ou_c(p,n)

Fonction = saisie au terminal d'un caractère en réponse à une question qui est :
<continuer ou quitter> si p < = n,
<tapez quitter> si p > n.

Argument = p : entier.
n : entier.
Précondition = /

Résultat = retour de la fonction : caractère.
Postcondition = si p < = n, le retour vaut 'c' ou 'q' selon l'introduction au terminal,
si p > n, le retour vaut 'q'

m_q_c(p,n)

Fonction = saisie au terminal d'un caractère en réponse à

une question qui est :
<Modifier une ligne ou Continuer ou Quitter>
s p < = n,
<Modifier une ligne ou Quitter> si p > n.

Argument = p : entier.
n : entier.
Précondition = /

Résultat = retour de la fonction : caractère.
Postcondition = si p < = n, le retour vaut 'c' ou 'm' ou 'q' selon l'introduction.

init_vt100( )

Fonction = initialisation d'un terminal de type VT 100 à un mode tel que le terminal devienne asynchrone et mémorisation du mode courant du terminal.

Résultat = état 1 : structure SGTTYB.
Postcondition = état 1 contient les caractéristiques courantes du terminal.

retab-vt100( )

Fonction = rétablissement d'un terminal de type VT 100 à son mode classique.

Argument = état 1 : structure SGTTYB.
Précondition = état 1 contient le mode classique du terminal et provient de la procédure "init-vt100()".

ret( )

Fonction = saisie au terminal du caractère "RETURN".

ANNEXE 2

MISE EN OEUVRE DU LOGICIEL

Comme il a déjà été mentionné, le logiciel se compose d'un ensemble de fichiers. L'annexe 2 met en évidence la démarche à suivre pour rendre le logiciel exécutable.
Les instructions citées ci-après se limitent aux fonctions qui sont réellement implémentées.

Phase 1

Compilation séparée des fichiers suivants par la commande cc

maître.c
memalloc.c
memcapac.c
memcapsol.c
memcfais.c
memchroute.c
memcontr.c
memdispo.c
memex.c
memfiab.c
memfichier.c
memfistock.c
memflux.c
memligne.c
memnoeud.c
mempara.c
memroute.c
memsauve.c
memstat.c
memstock.c
memtfais.c
memtraf.c

memvar.c
memvisu.c
tech.c

Cela produit des fichiers postfixés par ".o" au lieu de ".c".

Phase 2

Compilation en vue de créer les différents programmes :

. cc memvar.o tech.o maître.o - io maître
. cc memvar.o tech.o memsauve.o memex.o memalloc.o
  - io allocation
. cc memvar.o tech.o memsauve.o memex.o memcapac.o
  - io capacité
. cc memvar.o memsauve.o tech.o memcapsol.o
  - io calc-capac
. cc memvar.o memsauve.o tech.o memex.o memcfais.o
  - io cont-capac
. cc memvar.o memsauve.o tech.o memex.o memchroute.o
  - io mod-route
. cc memvar.o memsauve.o tech.o memex.o memcontr.o
  - io emplacement
. cc memvar.o memsauve.o tech.o memex.o memdispo.o
  - io cont-dispo
. cc memvar.o memsauve.o tech.o memex.o memfiab.o
  - io fiabilité
. cc memvar.o memsauve.o tech.o memex.o memfichier.o
  - io fichier
. cc memvar.o memsauve.o tech.o memfistock.o
  - io sauvetage
. cc memvar.o memsauve.o tech.o memex.o memflux.o
  - io cont-flux
. cc memvar.o memsauve.o tech.o memex.o memligne.o
  - io ligne
. cc memvar.o memsauve.o tech.o memex.o memnoeud.o
  - io noeud

. cc memvar.o memsauve.o tech.o memex.o mempara.o
  - io parametre
. cc memvar.o memsauve.o tech.o memex.o memroute.o
  - io route
. cc memvar.o memsauve.o tech.o memex.o memstat.o
  - io station
. cc memvar.o memsauve.o tech.o memex.o memstock.o
  - io cont-stock
. cc memvar.o memsauve.o tech.o memex.o memtfais.o
  - io cont-delai
. cc memvar.o memsauve.o tech.o memex.o memtraf.o
  - io trafic
. cc memvar.o memsauve.o tech.o memex.o memvisu.o
  - io visualisation
. cc memvar.o memsauve.o tech.o memex.o memchroute.o
  - io mod-route

Phase 3

Le logiciel démarre en introduisant au terminal maître.

Note

En tête de chaque fichier doit figurer la ligne suivante :
include "memtype.c"

*

* *

ANNEXE 3

STRUCTURE D'UN FICHIER DE DONNEES

Un fichier de données a une structure séquentielle. Les diverses informations se présentent dans l'ordre suivant :

- niveau d'introduction
- nombre de stations
- liste des descriptions des stations
- nombre de noeuds
- liste des descriptions des noeuds
- nombre de lignes
- caractère bidirectionnel ou non
- liste des descriptions des lignes
- matrice des distances entre deux stations reliées par une ligne
- matrice des numéros de ligne entre chaque paire de stations
- matrice de routage
- matrice indiquant les liaisons dont le chemin a été imposé
- nombre de fichiers
- liste des descriptions des fichiers
- nombre de capacités louées
- liste des descriptions des capacités louées
- nombre de capacités non louées
- liste des descriptions des capacités non louées
- nombre de capacités DCS louées
- liste des descriptions des capacités DCS louées
- nombre de capacités DCS non louées
- liste des descriptions des capacités DCS non louées
- matrice des coûts des allocations de capacités aux lignes
- matrice des coûts d'allocation fixes pour les capacités louées
- liste des paramètres d'utilisation du réseau, en fonction du régime

- longueur des messages, en fonction du régime
- matrices de trafic, en fonction du régime
- matrice des fiabilités des liaisons
- matrice des contraintes d'emplacement
- matrice de l'allocation de fichiers
- matrices du flux sur les lignes
- matrice de l'allocation de capacités
- variable indiquant si la solution est de coût minimal ou non
- coût de la solution courante
- délai moyen assuré par le réseau.

Le contenu d'un fichier de données dépend de la valeur du niveau d'introduction (4.3.2.).

*

* *