MASTER EN SCIENCES INFORMATIQUES

Contribution à l'élaboration d'une plate-forme de simulation de groupe d'ascenseurs

Misukami, Mia-Kanda

Award date:
1993

Awarding institution:
Universite de Namur

Facultés Universitaires
N.D. de la Paix

NAMUR

INSTITUT D'INFORMATIQUE

Contribution à l'élaboration d'une plate-forme de simulation de groupe d'ascenseurs

MISUKAMI Mia-Kanda

Promoteurs :

Monsieur le Professeur J.P. LECLERCQ

Monsieur P. HENNEAU

Mémoire présenté en vue de l'obtention
du titre de licencié et maître
en Informatique

Année Académique
1992-1993

Remerciements

Avant tout, j'aimerais remercier mes promoteurs , Jean-Paul Leclercq et Philippe Henneau qui m'ont entouré de leurs conseils tout au long de l'élaboration de ce travail.

Je tiens aussi à remercier de tout coeur pour leur accueil et leur gentillesse les membres du personnel de la société SCHINDLER de Sint-Pieters-Leeuw où s'est déroulé mon stage. Je pense notamment à Aslanadis I., Delhauwe S., Hermans C., Leone S., Matton K. et Nottebart D.

Enfin, pour leurs soutiens matériel et moral, pour rien au monde, je n'oublierai de dire merci à Monsieur l'Abbé Greindl R. et à Fabienne.

Nous abordons dans ce travail différentes techniques et méthodes de l'informatique pour développer une plate-forme de simulation de fonctionnement de groupe d'ascenseurs. Cela va de la simulation de systèmes à l'émulation des processeurs, de la logique floue aux systèmes multitâche et temps réel.

Nous nous attacherons avant tout à dégager toutes les informations nécessaires aux différentes composantes du groupe et développerons un algorithme de groupe permettant d'allouer une seule cabine d'ascenseur à une demande venant d'un palier d'un bâtiment en vue de minimiser au mieux le temps d'attente de chacun des clients du système.

In the present work, we tackle the different computer science technologies and methods in order to develop a simulation platform which will run a lift group. This includes system simulation, processor emulation, fuzzy logic, multitask and real-time systems.

First of all, we will try to draw all the necessary information from different parts of the group, and then, we will develop a group algorithm to allocate only one lift to one landing request in order to minimize, at best, the waiting time of each customer of the system.

Table des matières

Introduction

Dans ce travail, nous avons essayé d'apporter notre pierre à la conception et au développement d'une plate-forme de simulation de fonctionnement de groupe d'ascenseurs pour une entreprise de la périphérie bruxelloise, SCHINDLER S.A., qui fabrique et entretient des ascenseurs en Belgique et au Grand-duché du Luxembourg. SCHINDLER-Belgique, qui emploie quelques 800 personnes, est une filiale du groupe SCHINDLER dont le siège est installé à Ebikon en Suisse.

Ce travail s'insère dans le cadre général du projet de modernisation du parc des ascenseurs installés chez ses clients. La grande majorité des installations existantes étant des systèmes à relais, son remplacement par des systèmes nouveaux gérés par microprocesseurs se rapprochant de plus en plus des réseaux d'ordinateurs, il a semblé nécessaire aux responsables du Laboratoire de développement des logiciels d'initier ce projet de développement de plate-forme de simulation, en vue de disposer d'un outil d'évaluation, d'aide et de formation susceptible de fournir une approche intuitive et pratique du fonctionnement des ascenseurs tant pour les professionnels que pour les simples usagers et visiteurs.

Notre travail a été divisé en 5 grands chapitres.

Le premier chapitre donnera un aperçu général du vocabulaire et des formules de mesures des performances des ascenseurs du point de vue des concepteurs de bâtiments (architectes, ingénieurs en construction).

Le second chapitre sera consacré à la modélisation de groupe d'ascenseurs (un ascenseur, étant considéré comme un groupe avec un seul ascenseur). Nous l'avons découpé en différentes composantes ayant des fonctionnalités bien précises.

Le troisième chapitre illustrera quelques méthodes et outils qui seront utilisés dans la suite pour le développement de notre application. Nous y évoquerons quelques notions des systèmes multitâche et temps réel, de simulation et de logique floue; ensuite nous décrirons l'environnement de développement des programmes dans lequel nous avons évolué.

Dans le quatrième chapitre, nous expliquerons en détail la conception et/ou l'implémentation de l'algorithme de groupe, du détecteur de type de trafic et un embryon d'analyseur de trafic et de générateur d'appels.

Enfin, dans le dernier chapitre, nous essayerons de montrer les problèmes qui restent posés et d'indiquer quelques pistes pour leur résolution en guise de conclusion générale.

1. Introduction à l'ascenseur

1.1. SYSTEME A UNE CABINE OU SIMPLEX

Un ascenseur est un appareil utilisé dans les édifices à plusieurs étages, dans les mines, pour monter ou descendre d'un niveau à un autre des marchandises ou des personnes. C'est en l'an 26 avant J.-C. qu'apparaissent chez les romains des ascenseurs rudimentaires qui emploient la traction humaine, animale, ou la force de l'eau. L'ascenseur du type hydraulique apparaît en Europe au début du XIX^{e} siècle. La cabine est placée au sommet d'un piston qui se déplace verticalement dans un cylindre, sous la pression de l'eau. Ces types d'ascenseurs fonctionnent encore aujourd'hui mais ne sont utilisés que pour de petits parcours, et seulement si les vitesses requises ne sont pas importantes. C'est en 1889 aux Etats-Unis qu'apparaissent les premiers ascenseurs électriques.

Les différentes parties d'un ascenseur électrique sont:

- le moteur servant à la traction,
- le treuil, entraîné par le moteur, sert à tirer la cabine,
- la cabine destinée à accueillir les personnes,
- le contrepoids,
- les câbles de traction portant à une extrémité la cabine et à l'autre le contrepoids et, enfin,
- le circuit électrique de commande.

Le moteur.

Il est alimenté normalement par un courant alternatif ou, pour les grandes vitesses, par un courant continu.

Le treuil

Il comprend, en général, un réducteur de vitesse à engrenages et la poulie de traction qui porte des cannelures où s'enroulent les câbles de traction.

Les caractéristiques d'un ascenseur sont:

- **la portée,**
- **le parcours,**
- **le nombre des arrêts,**
- **la vitesse,**
- **le système de manoeuvre.**

La portée.

On entend par **portée ou charge nominale** le nombre maximum de personnes qui peuvent en toute sécurité trouver place dans la cabine. Le poids individuel est fixé à 80 kg.

Le **parcours.**

Le parcours est la distance existant entre le premier et le dernier arrêt.

La vitesse.

La vitesse est celle de la cabine animée d'un mouvement uniforme.On classe les ascenseurs en plusieurs catégories, "lents" dont la vitesse est inférieure à 0.4 m/s, "rapides" dont la vitesse dépasse 1.2 m/s, normaux et semi-rapides pour les vitesses intermédiaires. Il existe aussi des ascenseurs à grande vitesse, pouvant atteindre 7 m/s, qui sont de plus en plus utilisés dans les très hauts bâtiments. Des limitations de vitesse sont imposées non pour des raisons techniques inhérentes à la construction et au fonctionnement des divers appareils, mais pour éviter les mauvais effets physiologiques que peuvent provoquer sur l'organisme humain les fortes accélérations et les rapides variations de pression atmosphériques.

Pour manoeuvrer les ascenseurs, on utilise actuellement deux systèmes de commandes:

- **la manoeuvre automatique simple ou universelle** et
- **la manoeuvre enregistrée,** dite **collective et sélective.**

Avec la manoeuvre simple, l'ascenseur répond à une seule demande à la fois et plus précisément à la première de celles qui lui ont été faites au moment de la mise en marche.

Avec la manoeuvre sélective-collective, toutes les demandes sont enregistrées, en provenance de tous les étages. Il s'opèrent ensuite une sélection de ces demandes qui s'exécutent non pas dans l'ordre où elles ont été données, mais dans l'ordre de succession des étages dans le sens de la marche de la cabine.

Tous les ascenseurs sont dotés des dispositifs de sécurité qui en interdisent l'usage dans le cas d'une erreur de manoeuvre ou qui actionnent des freins d'urgence dans le cas de rupture des câbles de soutien ou quand la vitesse dépasse les limites.

1.2. SYSTEME A PLUSIEURS CABINES OU MULTIPLEX

Pour assurer un transport vertical efficace dans les bâtiments modernes, on est amené à installer plus d'une cabine ayant un système commun de sélection de la cabine pour desservir n'importe quelle demande, dénommée **appel palier**, des passagers. Le nombre de cabine à installer, leur portée et leur vitesse dépendent du nombre de niveaux à desservir et de la population totale du bâtiment. Ce dernier peut être de faible ou de moyenne hauteur, à usage commercial, d'habitation ou hospitalier. Un système à deux ascenseurs est appelé un duplex, à trois un triplex, à quatre un quadruplex,... D'où l'appellation générique de multiplex ou batterie ou groupe d'ascenseurs pour un système à plusieurs ascenseurs ayant une manoeuvre commune.

Le nombre de niveaux occupés comprend tous les niveaux au-dessus du niveau d'entrée. Dans certains bâtiments, on trouve un ou plusieurs niveaux d'entrée. Dans ce cas, le nombre de niveaux considérés est pris à partir du niveau d'entrée situé le plus bas. La population à transporter est estimée à partir des éléments suivants:

- pour un immeuble d'habitation, le nombre de résidents peut être estimé en fonction du nombre de chambre à coucher et de leur occupation moyenne.(de 1.5 à 1.9 personnes/chambre)

- pour un immeuble de bureaux, le nombre de personnes occupées est fonction de la surface nette de plancher; on considère qu'une personne occupe 7 à 12 mètres carrés de surface nette.

- pour un hôtel, le nombre de lits et l'occupation maximum estimée. (3 personnes/lit)

1.3. CRITERES DE PERFORMANCE ET DEFINITIONS UTILES

La performance d'une batterie d'ascenseurs est définie en considérant les critères quantitatifs et qualitatifs suivants:

- la capacité de transport, qui doit être suffisante pour éviter la formation de longue file d'attente, même durant les périodes de trafic intense.

- l'intervalle, qui affecte les temps d'attente des passagers et donc la qualité du service.

- la durée de parcours, qui est aussi un facteur de qualité de service. Une vitesse plus grande peut compenser une course plus longue.

La capacité de transport *d'une batterie est définie comme étant le rapport, exprimé en pourcentage, de population transportée en cinq minutes à la population totale du bâtiment.* Cela veut dire qu'une installation ayant 14% de capacité de transport devrait être capable de transporter, par 5 minutes, 14% de la population totale du bâtiment. Il existe des valeurs typiques de capacité de transport selon la situation et le type du bâtiment considéré. Par exemple, un bâtiment à usage de bureaux occupé par un seul service situé au centre-ville devrait avoir une capacité de transport minimum de 16%, d'après l'Institut Belge de Normalisation.

La capacité utile d'une cabine est le remplissage maximum estimé à 80% de la charge nominale.

La durée de parcours est définie comme étant le temps théorique de parcours entre les niveaux extrêmes. Elle est obtenue en divisant le parcours par la vitesse nominale de l'ascenseur.

L'occupation d'une cabine, c'est le pourcentage moyen de la charge nominale de la cabine transportée à chaque voyage.

L'occupation d'un bâtiment est le nombre total de personnes occupant les différents étages au-dessus du niveau supérieur de chargement et pouvant être desservis par la batterie.

Le temps d'un cycle ou temps d'attente maximal ou temps de rotation *est le temps moyen entre deux départs consécutifs d'une même cabine depuis le niveau d'entrée, incluant les temps de chargement et de déchargement, de fermeture et d'ouverture de porte et de parcours aller et retour.*

L'intervalle *est un temps calculé en divisant le temps d'un cycle par le nombre d'ascenseurs de la batterie.* Dans une installation correctement définie, le temps d'attente moyen doit varier autour de cette valeur. Par exemple, dans un hôpital, la valeur recommandée par l'Institut Belge de Normalisation pour l'intervalle varie entre 30 et 50 secondes.

Le temps d'attente, *c'est le temps écoulé entre un appel palier et l'arrivée d'une cabine.*

1.4. PRINCIPES DE FONCTIONNEMENT ET TYPES DE TRAFIC

Les principaux types de trafic d'un groupe d'ascenseurs sont:

- le trafic en pointe montée (Up-peak),
- le trafic en pointe descente (Down-peak),
- le trafic normal (Interfloor),
- le trafic de maintenance ou de priorité absolue et
- le trafic incendie.

Le trafic en pointe montée.

Cela arrive souvent dans un bâtiment à usage de bureaux par exemple, le matin aux alentours de l'heure du début de service. Dans ce cas, le trafic dominant se fait dans la direction montée à partir du niveau principal d'entrée et le taux d'occupation des cabines est élevé et proche de la pleine charge au démarrage pendant un certain temps.

Le trafic en pointe descente.

Cela arrive en fin de journée dans un bâtiment à usage de bureaux. Dans ce cas, un grand nombre d'appels en descente est enregistré et très peu en montée pendant un certain temps.

Trafic normal.

Cette situation s'installe lorsqu'il n'y a aucune prédominance du trafic en montée ou en descente. Les appels sont équitablement distribués entre les différents niveaux.

Trafic de maintenance ou de priorité absolue.

Dans ce cas, quelqu'un a introduit une clé spéciale à l'intérieur de la cabine et celle-ci est déconnectée du groupe d'ascenseurs. Elle n'obéira plus qu'à un seul appel cabine.

Trafic incendie.

Ce type de trafic est mis en route, comme dans le trafic de maintenance, par introduction d'une clé spéciale à l'extérieur de la cabine, au niveau du bouton d'appel palier. Dans ce cas, l'usage des ascenseurs est interdit. Le système de gestion de ce type de trafic devrait prendre les mesures appropriées pour libérer les passagers qui se trouvent encore dans les ascenseurs et pour en interdire l'usage en laissant ouvertes leurs portes. Seuls les pompiers peuvent utiliser les ascenseurs par l'introduction d'une clé.

1.5. METHODES CONVENTIONNELLES DE CALCULS DES PERFORMANCES DES SYSTEMES D'ASCENSEURS

Lors de la conception d'un système d'ascenseurs ou d'un groupe d'ascenseurs, la procédure habituelle consiste à déterminer la capacité de transport (handling capacity) pour un trafic en pointe montée (Up-peak). Cette dernière situation représentant le cas où le risque de formation de longue file d'attente est la plus grande. Si l'on détermine donc la capacité de transport dans le pire des cas, le système pourra donc tenir dans toutes les autres situations de trafic sans provoquer ce que l'on redoute le plus:

- la détérioration de la qualité de service,
- l'insatisfaction des passagers ou l'augmentation de ce que les japonais appellent le "degré d'irritation".

A défaut d'améliorer les conditions objectives d'insatisfaction des passagers, il y a moyen d'y pallier par des mesures psychologiques. Par exemple, on n'affiche pas la position des cabines qui circulent. Ainsi, si une cabine dépassait un étage où il y avait un passager en attente, celui-ci risquerait d'être agacé alors que la cabine qui passe était déjà pleine et qu'une autre avait été choisie par le système qui gère l'allocation des cabines pour le desservir. Généralement, les gens croient, à tort, que c'est toujours la cabine d'ascenseurs la plus proche qui doit venir répondre à leur demande.

1.5.1. Quelques définitions : capacité de transport, temps de cycle, intervalle

Les systèmes d'ascenseurs sont conçus pour pouvoir transporter verticalement la population fréquentant un bâtiment pendant la période de pointe montée sans occasionner un temps d'attente excessif, ou ce qui revient au même sans provoquer de files d'attente interminables. La capacité de transport doit donc être calculé de façon à pouvoir satisfaire la demande des passagers qui arrivent pendant cette période.

La capacité de transport peut être définie autrement de telle manière qu'on puisse la déterminer aisément.

La capacité de transport d'un groupe d'ascenseur **(handling capacity, HC)** est le nombre de passagers que le groupe peut transporter dans les 5 minutes durant une période de trafic en pointe montée avec pour chaque cabine d'ascenseurs un taux moyen de remplissage spécifique.

1.5.2. Comment calcule-t-on la capacité de transport d'un système d'ascenseurs ?

Considérons un système à une cabine arrivant au niveau d'entrée principal d'un bâtiment. Elle va prendre à son bord les passagers qui le souhaitent pour les amener à leurs étages de destination. Après avoir déchargé le dernier passager, la cabine d'ascenseur vide va immédiatement rejoindre le niveau d'entrée principal.

De là va émerger la notion de cycle et de temps de cycle.

Le temps de cycle (Round Trip Time, RTT) est le temps, exprimé en secondes, pour une cabine d'ascenseur de partir du niveau d'entrée portes fermées jusqu'au moment où il va rouvrir ses portes au même niveau d'entrée après avoir déposé tous ses passagers à destination.

On admet en règle générale que le temps de cycle ne devrait pas dépasser deux ou trois minutes, sauf éventuellement dans les très hauts bâtiments.

Connaissant le nombre de voyage d'une cabine pendant les 5 minutes de pointe montée, on peut en déduire son temps de cycle.

$$\textbf{nombre de voyage} = \frac{5 * 60}{RTT} \qquad (1)$$

De même, la capacité de transport d'une cabine durant les 5 minutes de pointe peut être calculée par la formule suivante:

$$\textbf{HC} = \textbf{Nombre de passagers par voyage} * \frac{5 * 60}{RTT} \qquad (2)$$

A la place du nombre de passagers par voyage dans la formule (2), on pourrait utiliser la capacité de transport nominale (contract capacity, CC). On sait par expérience que le nombre de passagers pouvant être transportés par voyage en moyenne est égal à 80 % de la capacité nominale.

D'où, l'équation (2) peut s'écrire:

$$HC = 80 * CC * \frac{300}{100 * RTT} = \frac{240 * CC}{RTT} \qquad (3)$$

La formule (3) est valable pour une seule cabine d'ascenseur.

Dans un groupe d'ascenseurs, le membre de droite de la formule (3) doit être multiplié par, L, le nombre de cabines d'ascenseurs. La formule (3) devient :

$$\mathbf{HC} = \frac{\mathbf{240 * L * CC}}{\mathbf{RTT}} \qquad (4)$$

La capacité de transport d'un groupe d'ascenseurs est un critère quantitatif de service. Les passagers sont plus concernés par les critères qualitatif dont le plus important est la qualité de service. Ce critère est évalué par le temps d'attente. Ceci va nous conduire à définir la notion d'intervalle qui donne une indication sur le temps d'attente moyen des passagers.

L'intervalle (INT) est le temps moyen entre deux arrivées successives de cabines au niveau d'entrée principal, les cabines pouvant être chargées ou déchargées à un niveau quelconque.

$$\mathbf{INT} = \frac{\mathbf{RTT}}{\mathbf{L}} \qquad (5)$$

L'intervalle en pointe montée (up-peak interval, UPPINT) est définie pour une cabine remplie à 80 % de sa charge nominale lors d'un trafic en pointe montée.

L'équation (4) peut-être réarrangée de la manière suivante:

$$\mathbf{HC} = \frac{\mathbf{240 * CC}}{\mathbf{UPPINT}} \qquad (6)$$

1.5.3. Estimation du temps de cycle (RTT)

Considérons une cabine d'ascenseur effectuant une course à partir du niveau principal d'entrée dans le cas d'un trafic en pointe montée. D'abord, la cabine va ouvrir ses portes et les passagers vont accéder à l'intérieur et indiqueront leurs destinations. Les portes vont se fermer. Puis, la cabine entamera sa course et va s'arrêter au premier niveau d'ordre cabine après être passé par les étapes suivantes: elle va commencer par accélérer de manière uniforme, atteindra sa vitesse nominale, décéléra et s'arrêtera. Notons que si la distance n'est pas très grande entre le niveau de départ et l'arrêt suivant, il peut arriver que la cabine n'atteigne pas sa vitesse nominale. La cabine va satisfaire tous les ordres cabines qu'elle a reçu et rejoindra le niveau principal d'entrée après avoir débarqué le dernier passager.

Cette séquence a mis en évidence les éléments suivants qu'on doit prendre en compte pour estimer le temps de cycle d'une cabine d'ascenseur:

- le temps d'entrée des passagers, t_l
- le temps de sortie des passagers, t_u
- le temps d'ouverture et de fermeture des portes, t_o et t_c
- le temps de parcours entre le départ d'un étage et le premier arrêt, $t_f(1)$. On suppose que la vitesse nominale est chaque fois atteinte.
- le temps de parcours à vitesse nominale entre deux étages quelconques non-adjacents
- le temps de parcours entre le dernier niveau servi et le niveau principal d'entrée

Comme on peut le remarquer, le temps de parcours dépend fortement du nombre d'arrêts effectués au dessus du niveau principal d'entrée (S), du nombre de passagers transportés (P) et du plus haut niveau atteint en moyenne lors de chaque course (H).

Nous pouvons écrire la formule suivante pour exprimer le temps de parcours :

$$RTT = P.t_{ft} + P.t_{fs} + (S+1)(t_c + t_o) + (S+1)t_r(1) + (H-S)t_v + (H-1)t_v \quad (7)$$

où t_v est le temps de parcours entre deux étages quelconques à la vitesse nominale et (S+1) est employé puisque le niveau principal d'entrée est un point d'arrêt.

$P.t_{ft} + P.t_{fs}$: est considéré comme le temps de transfert des passagers

$(S+1)(t_c + t_o)$: temps lié aux opérations sur les portes

$(S+1)t_r(1)$: temps d'accélération et de décélération

$(H-S)t_v$: temps pour atteindre les étages restants, t_a

$(H-1)t_v$: temps pour atteindre le niveau principal d'entrée après le dernier arrêt, t_e

En combinant et en simplifiant tous les éléments de la formule, on a:

$$RTT = 2Ht + (S+1)(t + t + t(1) - t) + P(t + t) \quad (8)$$

Prenons, t_p comme étant le temps moyen pour un passager particulier d'entrer et de sortir d'une cabine, et t_s, étant le temps associé à chaque arrêt,

$t_s = t_c + t_o + t_r(1) - t_v$.

Alors la formule (8) devient:

$$RTT = 2Ht_v + (S+1)t_s + 2Pt_p \qquad (9)$$

Donc pour calculer le temps de cycle, il faudra d'abord déterminer H, P, S.

1.5.4. Détermination du nombre d'arrêt espéré (S)

Considérons un building comportant N étages au-dessus du niveau principal d'entrée. Supposons que la population occupant le building est également distribuée dans tous les étages et que la probabilité d'arrêt à chaque niveau est la même. La probabilité qu'un passager quitte l'ascenseur à un niveau particulier vaut donc 1/N. De la même façon, la probabilité pour un passager de ne pas quitter l'ascenseur est de $1-\frac{1}{N}=\frac{N-1}{N}$. On suppose que chaque passager est indépendant de tous les autres. La probabilité qu'aucun passager parmi les P passagers ne quitte la cabine est égal en utilisant la loi du produit des probabilités à $(\frac{N-1}{N})^P$. Alors, la probabilité que la cabine s'arrête à un niveau quelconque est le complément de la formule ci-dessus: $1-(\frac{N-1}{N})^P$.

Pour N étages, le nombre moyen d'arrêts ou le nombre d'arrêts espérés vaut :

$$S=N[1-(\frac{N-1}{N})^{P}] \qquad (10)$$

1.5.5. Détermination du plus haut niveau atteint lors d'une course (H)

On suppose parfois que H vaut N, ou pour les bâtiments très élevés qu'il vaut N-1. Il est cependant possible de déduire H à partir de la théorie des probabilités. On fait les mêmes hypothèses que pour le calcul de S. Rappelons que la probabilité qu'aucun des P passagers ne quitte l'ascenseur dans un immeuble de N étages est égal à $(\frac{N-1}{N})^{P}$. Nous pouvons donc dire que la probabilité qu'aucune cabine d'ascenseur n'atteigne un niveau supérieur au i-ème étage est égal à la probabilité qu'aucun passager ne quitte la cabine au N-ième, (N-1)-ème, ... et (i+1)-ème étages.

Ce qui donne: $(\frac{i}{N})^{P}$.

Il est alors possible d'écrire que la probabilité que i soit le plus haut niveau atteint est égal à la probabilité que la cabine ne traverse pas les étages supérieurs au i-ème moins la probabilité que la cabine ne traverse pas les étages supérieur au (i-1)-ème étage.

P[i soit le plus haut niveau atteint] $= (\frac{i}{N})^{P} - (\frac{i-1}{N})^{P}$

Le plus haut niveau atteint en moyenne, H, est:

$$H = \sum_{i=1}^{N} i[(\frac{i}{N})^P - (\frac{i-1}{N})^P]$$

En développant et en simplifiant cette formule, on a:

$$H = N - \sum_{i=1}^{N-1} (\frac{i}{N})^P \qquad (11)$$

Le nombre de passagers transportés, P.

Il est d'usage de prendre P égal à 80 % de la capacité nominale de la cabine considérée. Puisque les passagers arrivent de manière aléatoire, attendre que la cabine soit chargée à 100 % allongerait inutilement le temps d'attente.

1.5.6. Conclusion

Toutes les formules que nous avons écrites plus haut donnent chaque fois des estimations pessimistes car souvent les destinations des clients sont groupées. Le nombre d'arrêt espéré, par exemple, est beaucoup plus grand qu'en réalité. De même, pour le haut niveau atteint, ainsi que pour le temps de cycle (RTT). Nous pouvons les prendre comme étant des valeurs indicatives et prudentes.

2. Modélisation de groupe d'ascenseurs

2.1. INTRODUCTION

Après avoir introduit les concepts fondamentaux utilisés dans le monde des concepteurs des ascenseurs, nous allons modéliser de manière générale un groupe d'ascenseurs et définir d'une manière aussi complète que possible les fonctionnalités des diverses composantes que nous mettrons en évidence peu à peu. Cette partie tiendra lieu de spécification fonctionnelle de notre projet de simulation de groupe d'ascenseurs. Avant tout, nous allons fournir les buts et objectifs du projet.

2.2. BUTS ET OBJECTIFS DU PROJET

Ce projet, baptisé 'Lift Group Simulator Platform', LSP en abrégé, a pour objectif le développement d'une plate-forme de simulation de groupe d'ascenseurs qui soit à la fois didactique et professionnelle.

Didactique, pour permettre par son aspect graphique et assez ludique de donner un aperçu intuitif et visuel du fonctionnement d'un groupe d'ascenseurs dans un bâtiment d'une hauteur raisonnable à des personnes qui, d'habitude, prennent les ascenseurs sans trop bien savoir ce qui se passe derrière.

Professionnel, pour permettre aux concepteurs et développeurs d'algorithmes de groupe d'avoir un cadre de simulation assez réaliste et pas trop coûteux qui leur offre tous les paramètres qu'il faut pour mettre au point leurs algorithmes. Professionnel aussi, car la plate-forme de simulation leur fournira les outils nécessaires pour évaluer les performances de leurs algorithmes.

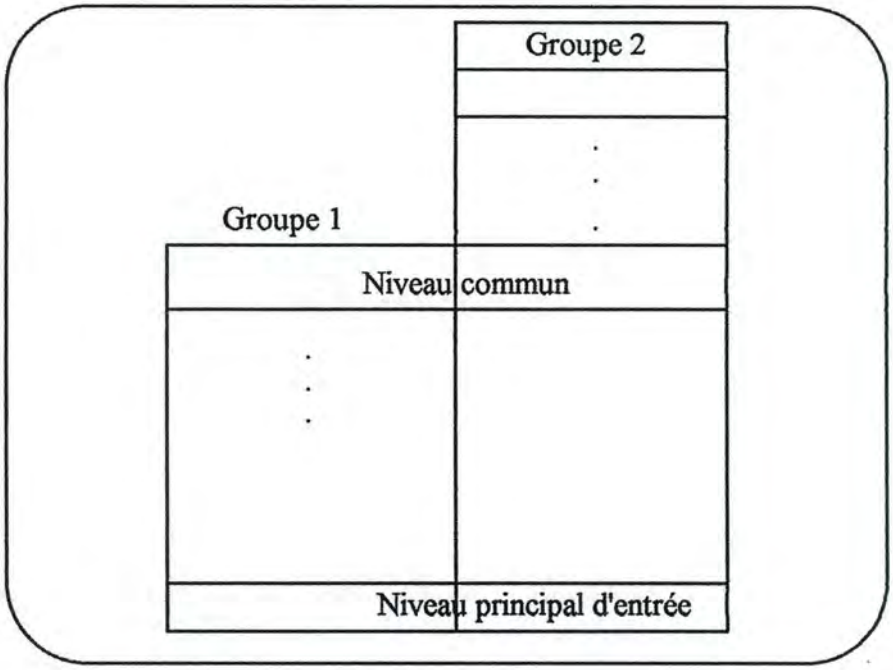
Avant de pouvoir atteindre tous ces objectifs, le projet devra mettre en évidence tous les paramètres sur lesquels il faudra 'jouer' pour faire un choix adéquat entre deux ou plusieurs cabines d'ascenseurs susceptibles de servir la demande d'un utilisateur.

2.3. CARACTERISTIQUES PRINCIPALES D'UN GROUPE D'ASCENSEURS

Un bâtiment à plusieurs étages peut contenir un ou plusieurs ascenseurs. On parle de groupe d'ascenseurs, quand on a au moins deux ascenseurs dans le bâtiment qui peuvent répondre de manière optimum à une demande d'un passager. Ceci veut dire qu'à un appel palier, une seule cabine d'ascenseur sera choisie par le système commun de commande, appelé parfois superviseur, parmi toutes les cabines d'ascenseurs disponibles pour le desservir. Ce choix se fera selon des critères définis à l'avance. Il est bon de rappeler que le nombre de cabines d'ascenseurs, leur capacité, leur vitesse de déplacement sont déterminés pour coïncider avec une valeur de l'intervalle déterminé à l'avance selon la configuration du bâtiment, sa fonction et sa population. Un algorithme de groupe, autrement dit un ensemble de règles permettant la sélection d'un ascenseur, sera mis en oeuvre pour assigner un appel palier à une cabine d'ascenseur donnée pour minimiser le temps de réponse du groupe, ou selon le point de vue des passagers pour minimiser le temps d'attente.

Remarquons que dans certains très hauts bâtiments, on peut trouver plus d'un groupe d'ascenseurs; chaque groupe servant une zone bien précise.

Immeuble avec deux batteries d'ascenseurs



Dans notre introduction aux ascenseurs, nous avions décrit un certain nombre d'éléments assurant des fonctionnalités bien distinctes. Nous les énumérons ci-dessous:

- Un bâtiment a un certain nombre d'étages. Chaque étage peut avoir une certaine hauteur. Il se pourrait que cette hauteur ne soit pas la même pour tous les étages. Dans certains bâtiments, selon le niveau où l'on est, les passagers peuvent débarquer ou embarquer d'un coté ou de l'autre d'une cabine d'ascenseurs.

- une cabine d'ascenseur a une certaine capacité nominale, et se meut à une certaine vitesse nominale. Ici aussi, il n'est pas nécessaire que toutes les cabines se déplacent à la même vitesse nominale et chargent le même nombre de passagers au plus.

- Dans la cabine, on trouve un tableau ou clavier qui permet au passager d'indiquer leur destination. Ce clavier dispose en plus d'un certain nombre de touches ayant des fonctions diverses:
  - une touche permettant d'arrêter une cabine à n'importe quel niveau, la touche 'STOP';
  - une touche d'alarme;
  - une touche d'attente;
  - une serrure permettant l'introduction d'une clé pour réserver la cabine pour une course spéciale, dans ce cas la cabine est détachée du groupe.

  On trouve aussi, dans la cabine, au-dessus de la porte d'entrée, un tableau avec la liste des étages desservis par l'ascenseur qui s'illumine sur le niveau auquel on se situe lors des déplacements.

- A chaque étage, on a aussi des boutons qui permettent de faire les demandes de cabines. En dehors des extrémités, nous avons deux types de boutons :
  - l'un pour effectuer les appels en montée et
  - l'autre pour les appels en descente.

  Au niveau principal d'entrée, on a aussi une serrure qui permet aux pompiers d'immobiliser les cabines au niveau principal d'entrée en cas d'incendie.

2.4. MODELE DE GROUPE D'ASCENSEURS

Nous regroupons tous ces éléments selon les fonctionnalités qu'ils assurent de la manière suivante (nous donnons entre parenthèses les abréviations que nous utiliserons dans la suite. Nous les préfixons par LSP, -Lift Group Simulation Platform - qui est l'acronyme du projet en anglais):

2.4.1. un configurateur de bâtiment (LSP_CFG) :

Nous rassemblons sous cette dénomination toutes les informations concernant le bâtiment, à savoir :

- le nombre d'étages,
- la hauteur de chaque étage,
- le temps d'ouverture et de fermeture des portes,
- la vitesse nominale et
- la capacité nominale de chaque cabine d'ascenseurs.

2.4.2. un simulateur d'ascenseur (LSP_LS) :

Le simulateur de cabines aura pour but de simuler de manière réaliste le déplacement de chaque cabine particulière selon

- sa vitesse nominale,
- les appels cabines qu'elle contient,
- les allocations qu'elle a reçues de l'algorithme de groupe,
- sa direction,
- le temps d'ouverture et de fermeture des portes.

Ce simulateur assure le contrôle complet de la cabine en respectant toutes les lois de la physique :

démarrage,

accélération,

décélération,

arrêt à un niveau donnée.

2.4.3. un générateur d'appels (LSP_CA) :

Le générateur d'appels sera chargé de générer suivant des distributions statistiques appropriées l'arrivée des passagers, l'étage de départ des appels paliers, l'étage de destination des passagers qui rentrent suivant le type de trafic en cours.

2.4.4. un répartiteur d'appels (LSP_GA) :

Le répartiteur d'appel est ce que l'on appelle, l'algorithme de groupe. Suivant les informations de fonctionnement des cabines disponibles, le répartiteur d'appel choisit une cabine pour servir une demande faite d'un palier à un certain étage. Tant que la cabine n'est pas arrivée au niveau de l'appel palier, le répartiteur d'appel peut trouver une cabine plus intéressante que celle précédemment choisie et annuler l'allocation initiale. C'est ce que l'on appelle la désallocation.

2.4.5. un détecteur de type de trafic (LSP_TD) :

Le détecteur de trafic servira à détecter le type de trafic en cours et à lancer l'exécution de l'algorithme correspondant de manière souple et dynamique. Les algorithmes de groupe sont conçus et dédiés à chaque type de trafic particulier. Selon le profil des appels reçus et le nombre de passagers dans le système, le détecteur de trafic servira à passer du trafic en pointe montée au trafic en pointe descente et vice versa, du trafic en pointe montée au trafic en 'interfloor' et vice versa, du trafic en pointe descente au trafic en 'interfloor' et vice versa.

2.4.6. un analyseur de trafic (LSP_TA) :

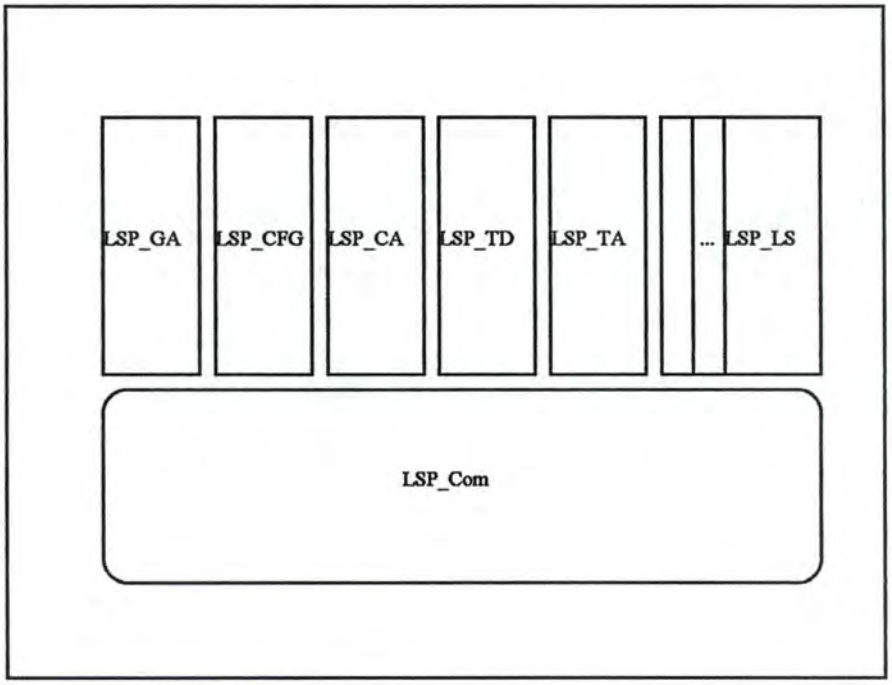
L'analyseur de trafic servira à établir les statistiques du trafic :

- le temps d'attente pour chaque appel palier reçu,
- le temps d'attente moyen,
- le temps de parcours de chacun des passagers
- le temps de parcours moyen,...

2.4.7. un système de communication (LSP_Com) :

C'est le système de communication qui assure le transfert d'informations entre les différents éléments du système. C'est grâce à ce système que le LSP_CA va communiquer avec le LSP_GA, que le LSP_GA communiquera avec le LSP_LS et réciproquement, ...

Nous pouvons donc représenter notre système au complet de la manière suivante :



2.5. ANALYSE DETAILLEE DE QUELQUES COMPOSANTES DU SYSTEME

2.5.1. Le LSP_LS

Le 'Lift Simulator' va simuler de manière déterministe le fonctionnement des cabines d'ascenseurs. Pour mettre en évidence le fonctionnement d'un ascenseur, examinons le comportement d'une cabine d'ascenseur particulière dans différentes situations et identifions les données qu'elle fournit et qu'elle reçoit.

Une cabine peut être soit à l'arrêt, soit en mouvement dans une direction précise (montée ou descente) pour aller desservir un ordre cabine d'un passager déjà chargé ou en mouvement suite à un ordre du superviseur pour répondre à une demande de cabine faite par un utilisateur.

2.5.1.1. Une cabine à l'arrêt sans passagers

Dans ces conditions, la cabine a ses portes fermées.(Si elle était au niveau principal d'entrée ses portes seraient ouvertes). La direction n'est pas indiquée. Sa position physique correspond à sa position logique. La position physique correspond à sa position réelle, tandis que sa position logique est celle indiquée quand la cabine se déplace. Elle indique le premier étage auquel l'ascenseur peut s'arrêter compte tenu de sa direction, de sa vitesse et du temps de décélération. Sa charge, en nombre de passagers, est nulle.

2.5.1.2. Une cabine à l'arrêt avec passagers

Dans ces conditions, ses portes peuvent être dans l'une des situations suivantes : fermées,

en train de se fermer,

en train de s'ouvrir parce que le passager a appuyé sur la touche 'STOP' ou sur la touche d'attente.

Sa direction peut déjà être indiquée si le passager a déjà donné un ordre cabine pour une certaine destination.

Sa charge, en nombre de passagers, indique de manière approchée le nombre de passagers présents dans la cabine. Il est difficile dans la réalité de déterminer le nombre exact de passagers présents dans une cabine d'ascenseur, puisque les passagers peuvent porter des sacs ou des colis, et le poids des passagers ne correspond pas toujours au poids moyen pris comme référence pour déterminer le nombre de passagers.

Comme on est en simulation et que les passagers fréquentant le système sont générés de manière stochastique, on sait exactement le nombre de passagers qu'il y a dans le système.

2.5.1.3. Une cabine en mouvement

Une cabine doit passer par les étapes suivantes quand elle veut quitter sa position de repos. Elle va fermer ses portes, si elles étaient ouvertes, et démarrer en pleine puissance pour s'arracher à la pesanteur. Ce qui implique une grande dépense d'énergie; son moteur est conçu pour lui permettre de partir dans toutes les conditions de confort pour les passagers qui sont à bord. Son accélération est toujours uniforme.

Supposons que la cabine atteint, lors de chaque course sa vitesse nominale, elle va commencer à accélérer, atteindre sa vitesse nominale et à partir d'un endroit déterminer à l'avance selon sa destination, cet endroit est appelé sélecteur, elle va commencer à décélérer, puis s'arrêter et ouvrir ses portes.

Selon les constructeurs, le moment d'ouverture des portes est différents. Dans certaines installations, les portes commencent à s'ouvrir quelques secondes avant l'arrêt total. Dans d'autres, les portes s'ouvrent à l'arrêt complet.

Une cabine en déplacement a une direction unique, soit en montée, soit en descente. Tant qu'elle est mouvement sa position physique est toujours différente de sa position logique. La différence dépend de la vitesse nominale. Elle peut être d'un demi-étage, d'un étage ou plus. Quand elle est en montée, sa position logique est toujours supérieure à sa position physique. C'est le contraire qui se passe quand elle est en descente. Dès qu'elle commence à décélérer pour s'arrêter les deux positions se rejoignent.

Une cabine en mouvement sert en priorité les ordres cabines de ses passagers. Elle pourra s'arrêter dans la mesure du possible pour répondre à une allocation faite par l'algorithme de groupe. Dans la mesure du possible, cela veut dire que la cabine peut s'arrêter, mais sans décélérer de manière intempestive et sans changer de direction si des passagers sont à bord.

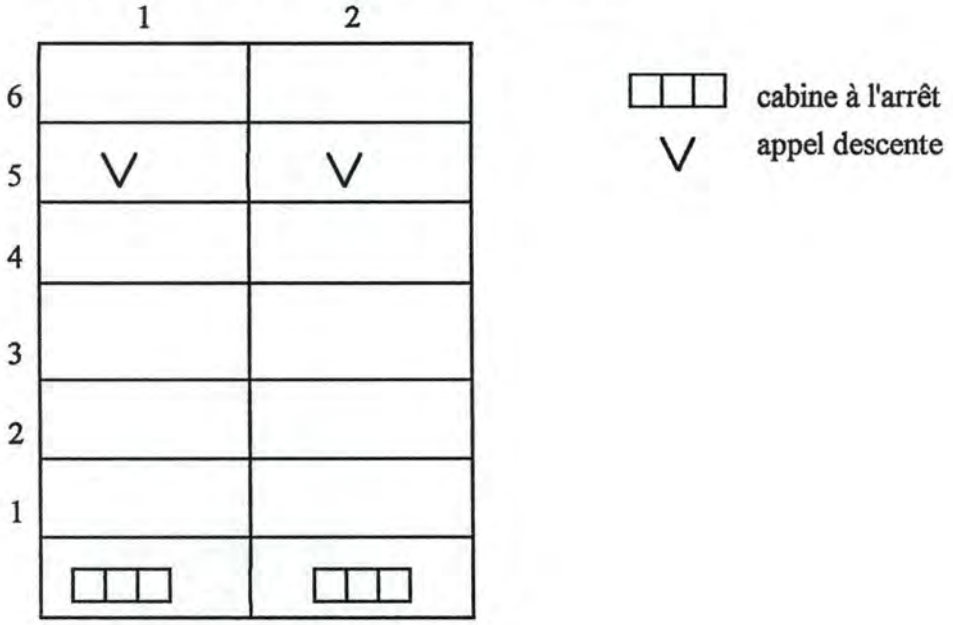
2.5.2. Le LSP_GA

2.5.2.1. Pourquoi un algorithme de groupe ?

Avant de répondre à cette question, regardons ce qui se passe quand on a deux ascenseurs indépendants commandés chacun par une manoeuvre enregistrée, dite 'collective/sélective'.

Si nous sommes, par exemple, dans un bâtiment avec 6 étages au-dessus du niveau principal d'entrée et qu'il y ait un passager au 5ème étage, qui fait une demande pour descendre, les deux cabines étant au niveau d'entrée, les gens, le plus souvent, ne se contentent pas de faire une seule demande, ils appuient sur les boutons d'appels des deux cabines.

Système avec deux cabines indépendantes



Résultats: les deux cabines se déplaceront pour servir cet appel. Pendant ce temps, les passagers qui arriveront au niveau principal d'entrée devront attendre plus longtemps pour rien car l'une des deux cabines fait une course inutile.

D'où l'idée d'avoir un système commun de gestion de demande de cabines. Dans la même situation que ci-dessus, le système commun de gestion, que nous appelons l'algorithme de groupe, choisira une seule cabine pour aller desservir la demande.

Un autre problème qui se pose est celui des passagers qui appuient à la fois sur les boutons d'appel pour monter (UP) et pour descendre (DOWN). Ce comportement occasionne aussi une détérioration de l'utilisation du système. Face à cela, on est démuni sauf si on assure une formation à l'utilisation du système aux usagers.

Définition

Un **algorithme de groupe** est un ensemble de règles définissant le comportement des ascenseurs dans les différents types de trafic.

Cette définition implique donc les différences de comportement selon le type de trafic en cours. Les règles applicables quand on est en pointe montée par exemple ne sont plus les mêmes dans le cas d'un trafic normal (interfloor').

Le 'Group Algorithm' est chargé d'assurer la répartition des demandes de cabines d'ascenseurs entre les différentes cabines disponibles de manière équilibrée et selon une politique compatible avec l'objectif principal assigné au système, c'est-à-dire minimiser le temps de réponse du système. Ce temps de réponse est aussi le temps d'attente des utilisateurs du système. Si cet objectif est atteint, on évitera la formation de longue file d'attente et on augmentera la satisfaction des utilisateurs du système.

Minimiser le temps de réponse du système peut conduire à un allongement du temps de parcours de passagers et à une augmentation du nombre d'arrêt d'une cabine. Cette augmentation du nombre d'arrêt peut occasionner une détérioration de la qualité de service du système du point de vue des usagers. Il faudra donc trouver un compromis pour satisfaire l'objectif principal sans trop augmenter le temps de parcours des usagers.

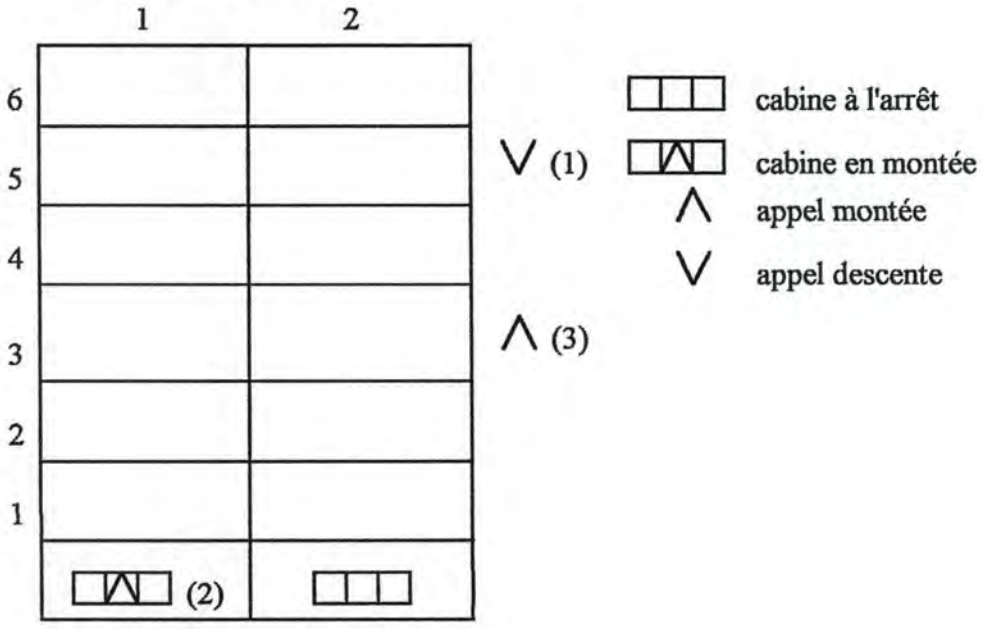
Les algorithmes de groupe diffèrent fort suivant la technologie employée pour recueillir les informations qui permet de s'assurer du nombre plus ou moins grand de paramètres pour une meilleure prise de décision. Pour résoudre le problème d'allocation d'appels dans un groupe d'ascenseurs, un certain nombre d'algorithmes aux performances et objectifs variés ont été mis au point. Il serait bon de noter qu'il y a des algorithmes qui font des allocations une fois pour toutes et il y en a d'autres qui peuvent allouer un appel précédemment assigné à une cabine à une autre cabine d'ascenseur. Cela est désigné sous le vocable de 'désallocation'. Allocation/désallocation, voilà donc les deux faces d'une même réalité.

2.5.2.2. L'intérêt d'un algorithme d'allocation/désallocation

Imaginons la situation suivante : nous sommes toujours dans le même immeuble que ci-dessus. Nous avons un premier appel palier de descente au cinquième étage (1) (voir la figure ci-dessous). La cabine 1 a été choisie (2). Quelques secondes après un second appel apparaît au troisième étage pour monter (3).

La cabine 1 est de nouveau choisie puisqu'elle va dans la direction de l'appel et qu'elle est la plus proche. Au troisième, plusieurs personnes rentrent avec comme destination le sixième étage. Le nombre de personnes qui sont rentrées dans la cabine est tel que sa charge est proche de sa capacité nominale, la rendant ainsi indisponible pour servir un autre appel. Puisque les ordres cabine sont prioritaires par rapport aux allocations, la cabine 1 ira d'abord au sixième étage et l'appel du 5ème risquerait d'attendre encore si au 6ème, un groupe de personnes occupaient de nouveau la cabine avec une capacité proche de sa capacité nominale pour descendre au 3ème et au 2ème étage.

Système d'allocation/désallocation



D'où l'intérêt d'un système de désallocation d'appel. Dans notre cas, il serait judicieux de choisir la cabine 2 qui est libre et au repos pour aller desservir l'appel du 5ème étage et décharger la cabine 1 de ce service, sinon l'appel du 5ème risquerait d'attendre longtemps.

Notons, enfin que, la minimisation de la consommation de l'énergie électrique des moteurs des ascenseurs n'est pas un critère à prendre en compte dans les différents algorithmes.

2.5.3. Le LSP_CA (call generator)

Le LSP_CA va générer l'arrivée des passagers, leurs appels paliers, leurs ordres cabines d'une manière compatible avec le fonctionnement du système. Par exemple, supposons qu'il génère l'arrivée d'un passager au 3ème étage d'un immeuble. Il pourra ensuite générer un appel pour monter. A l'arrivée de la cabine, on attendra le temps qu'il faut pour que la cabine ouvre ses portes et que le passager puisse embarquer et générer un ordre cabine pour le 6ème étage, par exemple.

Le LSP_CA tiendra aussi compte du type de trafic. Si on était en pointe descente, par exemple, la plupart de ses ordres cabines seront orientés vers le niveau principal d'entrée.

2.5.4. Le LSP_TD (traffic detector)

Le 'Traffic Detector' servira à expérimenter un nouveau moyen de passage d'un type d'algorithme à un autre. Rappelons que dans une batterie d'ascenseurs, on peut passer par trois types principaux de trafic: la pointe montée, la pointe descente et l'interfloor. On écrit trois types d'algorithme pour faire face à ces trois types de trafic; l'un des moyens utilisé pour passer d'un type d'algorithme à un autre est de programmer à l'avance de manière statique les heures de passage.

Imaginons que le trafic en pointe montée soit prévu de 8h30 à 9h45 dans un bâtiment de bureaux et qu'un jour pour une raison ou une autre, il y ait un léger décalage dans les heures d'arrivée des usagers. Notre système de gestion ne répondra pas comme il faut. Aussi, nous proposons un système dynamique de détection du type de trafic en cours dont nous établirons les règles dans la suite de cette section, qui pourra faire appel à l'algorithme adéquat au bon moment. Ce système utilisera la logique floue qui sera expliquée dans le chapitre suivant.

2.5.5. Le LSP_TA (traffic analyzer)

2.5.5.1. De la difficulté d'évaluer le temps d'attente d'un passager

Il est difficile, en pratique, d'évaluer les temps d'attente de chacun des usagers du système. Comment peut-on déterminer l'heure d'arrivée d'un client ? Sinon, par l'intermédiaire d'un appel palier. Or, quand quelqu'un fait une demande de cabine, celle-ci est valable pour toutes les personnes se trouvant en ce moment sur le palier.

De même pour les ordres cabines : les destinations peuvent être groupées. Huit personnes dans une cabine peuvent avoir une ou deux destinations seulement. Rien ne l'interdit. Les formules de calcul vu au chapitre II donnent chaque fois une évaluation trop pessimiste.

Comme nous ferons de la simulation, les passagers et leurs comportements seront générés par le LSP_CA, on pourra aisément faire leurs statistiques.

2.5.6. Le LSP_Com (Communication)

En ce qui concerne la communication entre les différents composants dans notre plate-forme de simulation, nous utiliserons un logiciel de communication appelé VCOM, Virtual Communication. C'est un logiciel-maison développé par la société SCHINDLER dans le cadre de la modernisation de ses installations. Une interface des fonctions en langage C est fournie aux programmeurs qui souhaitent transmettre les informations entre différentes applications. Nous allons expliquer brièvement le principe de fonctionnement du logiciel et nous fournirons en annexe la liste de toutes les fonctions qu'on peut utiliser.

2.5.6.1. Protocole de communication en VCOM

VCOM dégage le programmeur de la responsabilité d'assurer la communication des informations entre les tâches. Il n'a même pas à connaître l'origine des informations qu'il souhaite obtenir. Tout ce qu'il a à faire est de d'obéir au protocole ci-après.

Nous avons deux types de tâches, celles qui demandent les informations et celles qui les fournissent. La tâche qui demande les informations doit tout simplement ouvrir une file d'attente[1] en indiquant son numéro et dire que c'est dans cette file qu'elle souhaite recevoir les informations qu'elle spécifie dans une série d'appel de fonctions de requête. Lorsque l'information deviendra disponible, elle n'aura qu'à aller la lire. VCOM s'occupe de tout.

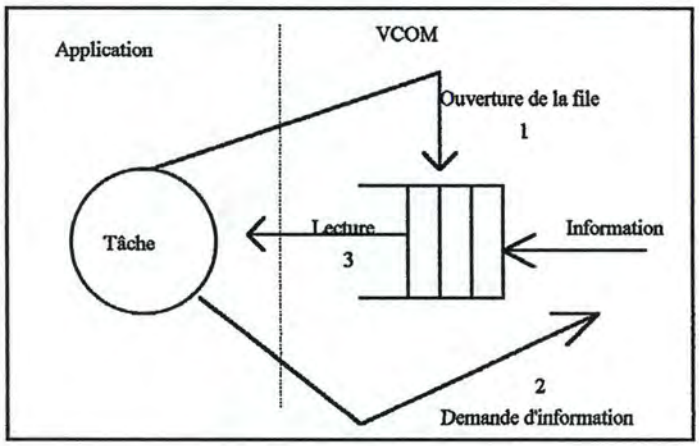
La tâche qui fournit les informations doit aussi indiquer les informations qu'elle peut envoyer. Lorsque celles-ci seront disponibles, elle utilisera la fonction appropriée de VCOM qui se chargera d'envoyer les informations aux tâches qui les avaient demandées.

[1]C'est en fait une zone de mémoire tampon

La tâche qui reçoit les informations doit faire les trois opérations suivantes :

1. Ouvrir une file d'attente
2. Faire des requêtes des informations qu'on désire voir mises dans la file d'attente
3. Lire le contenu de la file d'attente.

Protocole de requête des informations



La tâche qui fournit les informations doit faire les opérations suivantes :

1. Faire la liste des informations qu'elle peut fournir;
2. Laisser un petit délai pour que VCOM ait le temps d'ouvrir le canal virtuel entre la tâche qui envoie et la tâche qui reçoit les informations;
3. Envoyer les informations aussi longtemps qu'il y en a.

Notons que VCOM envoie les informations dans toutes les files qui ont demandé cette information.

En VCOM, les informations sont envoyées dans des structures dites "télégrammes". Appelons une structure de ce genre, VCOM_TELEGRAM. Elle est composée des champs suivants :

- un champ d'information, désigné comme VCOM_INFO
- un champ contenant le nombre des informations
- un champ contenant un tableau des données dont la longueur est spécifiée au point précédent.

VCOM_INFO est un tableau de structure ayant les cinq champs suivants:

- 'telegram_group_id
- 'lift_id'
- 'telegram_id'
- 'attribute_1'
- 'attribute_2'

Telegram_group_id est une variable qui peut contenir les valeurs de 1 à 255 ou 0 qui est considéré comme une valeur passe-partout. C'est-à-dire quand on rentre 0 comme numéro d'identification de groupe, la correspondance se fait avec tous les groupes existants.

Lift_id, cette variable peut être 0 ou toute valeur entre 1 et 11. Elle identifie, l'ascenseur qui envoie les informations. Quand sa valeur est égale à 11, cela désigne l'ascenseur courant dans le système. Ceci est utile quand on veut avoir les informations sur l'ascenseur en cours sans connaître de manière absolue son numéro.

Telegram_id est une variable entière qui prend les valeurs de 0 à 4095.

Attribute_1 est une variable entière. Elle prend les valeurs de 0 à 511.

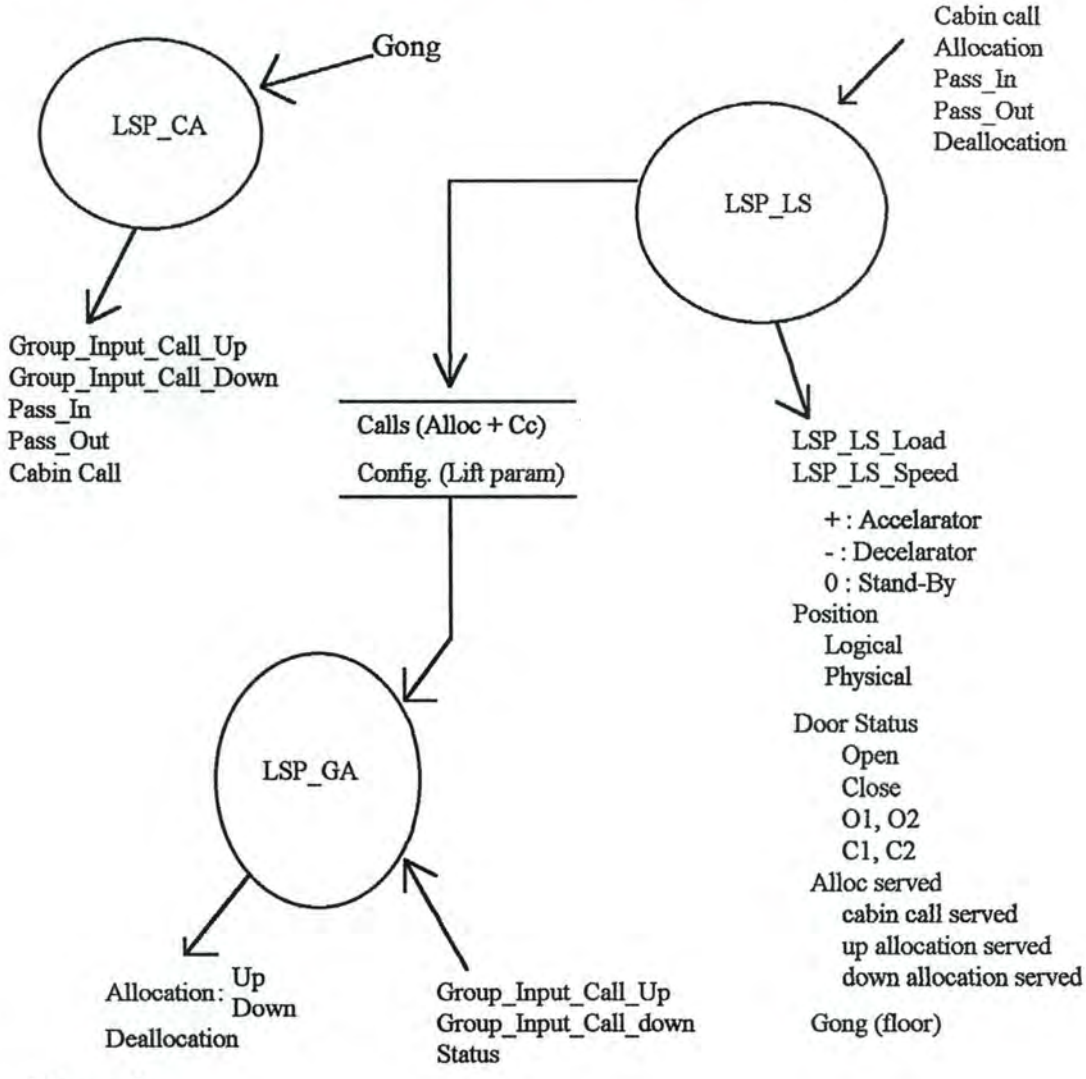
Attribute_2 est une variable entière qui prend les valeurs de 0 à 15.

Attribute_3 est une variable entière. Elle prend les valeurs de 0 à 7.

La sémantique à donner aux valeurs de ces variables est laissée à la discrétion du programmeur.

2.6. DIAGRAMME DES FLUX DES DONNEES

Nous allons mettre en évidence dans cette partie les principales informations échangées entre les différentes composantes de notre système.



Légende :

: ceci représente une base de données (une variable globale, ou un fichier commun)

: ceci représente un module assurant une fonctionnalité précisée par le libellé situé à l'inté

: ceci indique les messages reçus ou envoyés.

Le 'LSP_LS' reçoit les messages suivants :

les appels cabines	(Cabin call)
les allocations	(Allocation)
les désallocations	(deallocation)
le nombre de passagers entrant	(Pass_In)
le nombre de passagers sortant	(Pass_Out)

Il envoie les messages suivants (toutes ces informations seront désignées comme étant le statut d'une cabine) :

la charge de la cabine (en nombre de passagers)	(LSP_LS_Load)
la vitesse de la cabine. Elle enverra plutôt les phases d'accélération de la cabine : accélération, décélération, attente.	(LSP_LS_Speed : accelerator, decelarator, stand-by).
la position physique	(physical position)
la position logique	(logical position)

le statut des portes :	[Doors status :
fermée,	close (C1)
ouverte,	open (O1)
en train de se fermer,	closing (C2)
en train de s'ouvrir	opening (O2)]
Allocation servie :	(Allocation served :
ordre cabine servi,	cabin call served,
allocation montée servie,	up allocation served,
allocation descente servie.	down allocation served).
Gong :	(Gong :
l'étage où l'on est arrivé.	floor).

Le 'LSP_GA' reçoit les messages suivants :

les appels paliers montée	Group_Input_Call_Up
les appels paliers descente	Group_Input_Call_Down
le statut des cabines	status

Le 'LSP_GA' envoie les messages suivants:

les allocations montée ou descente	Allocation Up or Down
les désallocations montée ou descente	Deallocation Up or Down

Le 'LSP_CA' reçoit simplement le GONG, c'est-à-dire le niveau où se trouve chaque cabine du groupe.

Il envoie comme messages :

les appels paliers montée	Group_Input_Call_Up
les appels paliers descente	Group_Input_Call_Down
le nombre de passagers entrant	(Pass_In)
le nombre de passagers sortant	(Pass_Out)
les appels cabines	(Cabin call)

3. Méthodes et Outils utilisés dans la mise en oeuvre

3.1. SYSTEME MULTITACHE ET TEMPS REEL

3.1.1. Introduction

Cette section est justifiée par le fait notre application est typiquement temps réel. Nous avons divisé notre projet de simulation de groupe d'ascenseurs en plusieurs modules indépendants les uns des autres et s'exécutant en parallèle et communicant par messages. Rappelons que nous avons un module qui simule le fonctionnement des ascenseurs (lift simulator), un module pour la génération des arrivées des passagers, des demandes de cabines et de destination (call generator), un module pour l'algorithme de groupe (group algorithm), un module de configuration du bâtiment (building configurator).

Tous ces modules seront implémentés sous formes de tâches tournant en compétition (ce qui implique aussi la préemptivité) et en coopération dans un système monoprocesseur. D'où la nécessité de travailler dans un environnement multitâche et temps réel capable de garantir à chaque tâche une réaction appropriée dans le temps face aux événements qui arrivent. Il ne serait pas convenable qu'une requête de cabine soit prise en compte après un temps supérieur à l'intervalle, par exemple.

Nous allons donc expliquer dans un premier temps ces deux notions et dans un deuxième temps examiner l'exécutif temps réel de VRTX32 que nous utiliserons. Tous ces termes vont être expliquer brièvement dans les quelques pages qui suivent car il est vrai que diverses interprétations, parfois erronées, circulent même dans les milieux où on fait usage du temps réel.

3.1.2. Définitions

3.1.2.1. déterminisme, préemptivité, système temps multitâche et réel

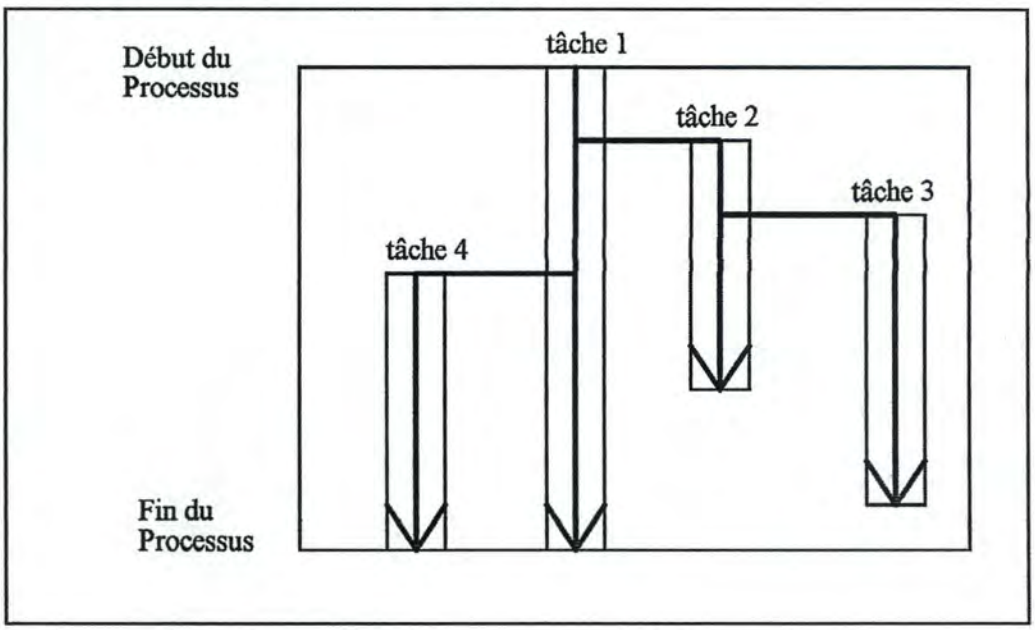
Un système temps réel est un système qui est soumis à de contraintes de temps d'origines matérielles et non humaines. Ce système doit répondre à des événements venant du monde extérieur en un temps fini et spécifié. Cette dernière caractéristique définissant le déterminisme est primordiale car il serait inadmissible qu'une tâche ayant encore besoin du processeur ne l'ait pas après un certain temps d'utilisation, l'empêchant ainsi de faire le traitement voulu dans les délais requis.

Dans notre cas, il serait inconvenant qu'une cabine d'ascenseur soit renseigné comme étant arrêté au deuxième étage alors qu'elle a déjà atteint le quatrième étage d'un immeuble. Il y a donc une certaine urgence à ce que la tâche qui simule un ascenseur donné en déplacement puisse avoir le processeur pour effectuer son traitement. C'est ce qu'implique la **préemptivité**. C'est la réquisition du processeur pour l'exécution d'une tâche et d'une seule pendant un temps déterminé.

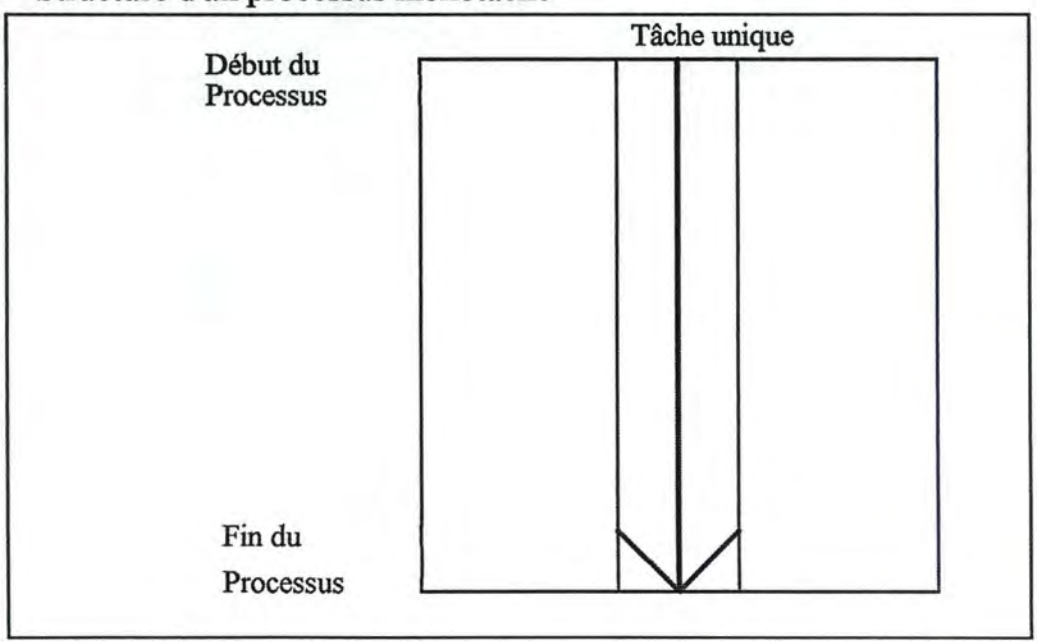
Le but d'un système multitâche est d'optimiser l'utilisation du processeur tandis qu'un système temps réel nécessite le déterminisme et l'urgence dans le traitement des événements extérieurs. On peut dire tout simplement que dans un système multitâche lorsqu'une interruption survient, elle doit être prise en compte mais son traitement peut ne pas être immédiat. Il est différé. Alors que dans un système temps réel, on s'arrange pour prendre en compte et effectuer le traitement associé à une interruption sans délai.

Pour les applications complexes, il est plus facile au niveau de la conception et de la programmation de travailler dans un environnement multitâche. Mais, les applications simples ou à très fortes contraintes de temps sont généralement programmées en monotâche. Ci-dessus, nous montrons les structures des processus en multitâche et en monotâche.

Structure d'un processus multitâche.



Structure d'un processus monotâche



3.1.3. Les contraintes de temps

Pour un système temps réel, la contrainte la plus important à respecter est le temps de réponse de la machine suivant l'environnement en présence. Les applications peuvent être regroupées dans en catégories suivant leurs contraintes de temps.

3.1.3.1. Les applications à contraintes de temps faibles

Ce sont des applications qui permettent une programmation en mode bouclé, ce qui s'apparente très fort à une attente active. Dans ce mode de programmation, on scrute en permanence les périphériques d'entrées/sorties. La lecture des entrées permet de détecter si un événement s'est produit ou non. Dans le cas où l'événement s'est produit, on fait le traitement approprié, et on envoie les résultats à la sortie. Puis, on recommence la lecture des entrées, ... La somme des temps de lecture, de traitement et d'écriture doit être inférieure au temps d'échéance fixé à l'avance. Une application typique de cette catégorie est celle concernant la lecture des demandes de cabines d'ascenseurs.

Ce mode de programmation est simple à implémenter, mais difficile à maintenir car il trop dépendant des performances du matériel dont on dispose. En cas de modification ou d'adjonction de fonctions, si les contraintes de temps n'étaient plus respectées, la réécriture du programme pourrait être éviter en utilisant un matériel plus performant.

3.1.3.2. Les applications à contraintes de temps faibles avec quelques événements contraignants

Dans ce genre d'applications, il faut répondre tout de suite à l'événement qui survient sinon il pourrait devenir sans intérêt. Mais ces événements ne sont pas assez nombreux et arrivent de manière aléatoire. A chaque survenance de l'événement, le programme en cours sera suspendu, et le traitement associé sera lancé grâce au mécanisme d'interruption.

3.1.3.3. Les applications à fortes contraintes de temps

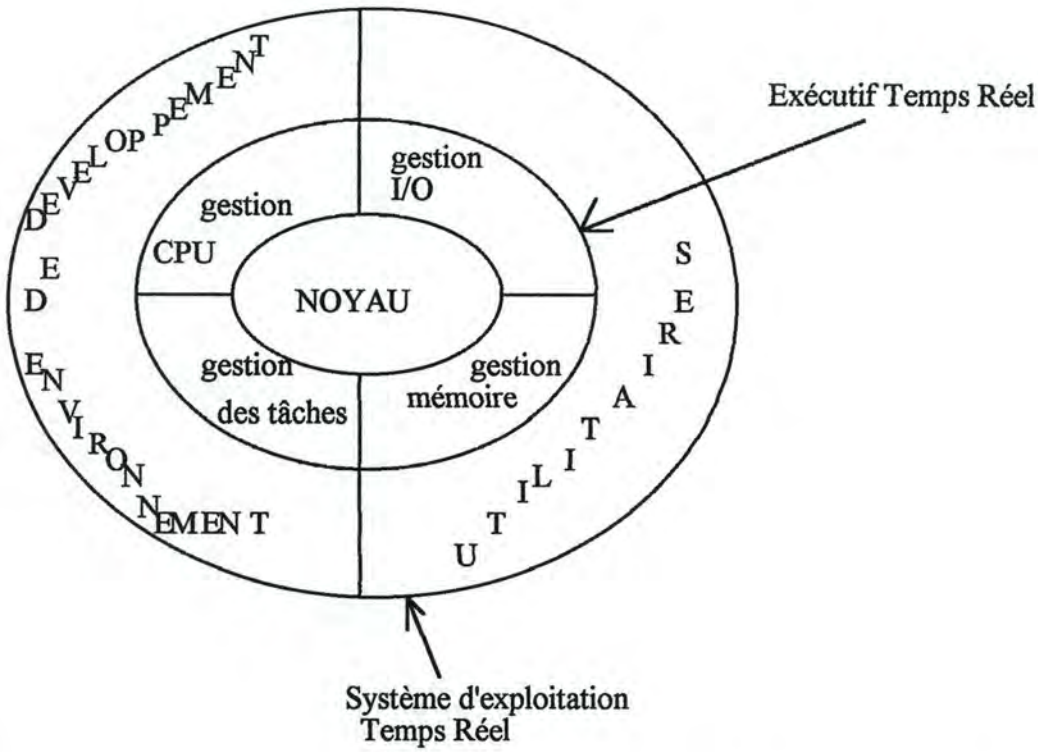
Un système monoprocesseur ne peut traiter qu'une tâche à la fois à chaque instant. Si on en a plusieurs, elles entreront en possession du processeur les unes après les autres. Si on était face à des applications à fortes contraintes de temps, un seul processeur ne pourrait plus suffire. On pourra faire appel à un système multiprocesseur, avec même certains processeurs spécialisés.

3.1.3.4. Ordres de grandeur de temps de réponse

Pour clore cette section consacrée aux contraintes de temps liées au concept de système temps réel, nous allons donner quelques ordres de grandeur de temps de réponse dans différents domaines d'application:

- la milliseconde pour les systèmes radar,
- la seconde pour des systèmes de visualisation humaine
- la demi-minute pour les systèmes de contrôle de stockage
- quelques minutes pour les systèmes de contrôle de fabrication
- quelques heures pour les systèmes de contrôle de certaines réactions chimiques.

3.1.4. Etat de l'art



Pour mettre en oeuvre une application temps réel, on doit soit travailler avec un système d'exploitation temps réel, soit employer un exécutif temps réel. Un système temps réel multitâche fournit un environnement de développement convivial, des utilitaires dans le but d'exploiter les ressources de la machine et de favoriser le dialogue machine-homme tandis qu'un exécutif temps réel vise surtout à maximaliser les performances dans le but de faciliter le dialogue machine-capteur. Les exécutifs temps réel sont souvent utilisés dans les systèmes embarqués, sans systèmes d'exploitation ou en complément de celui-ci.

Dans les systèmes d'exploitation, les tâches sont des programmes complets et sont appelés processus lorsqu'ils sont en exécution, alors que dans un exécutif temps réel les tâches sont des procédures d'un même programme.

Signalons qu'à un niveau plus bas que l'exécutif, on trouve le noyau temps réel. Mais, il est parfois difficile de tracer la frontière entre les primitives du noyau et les primitives de l'exécutif.

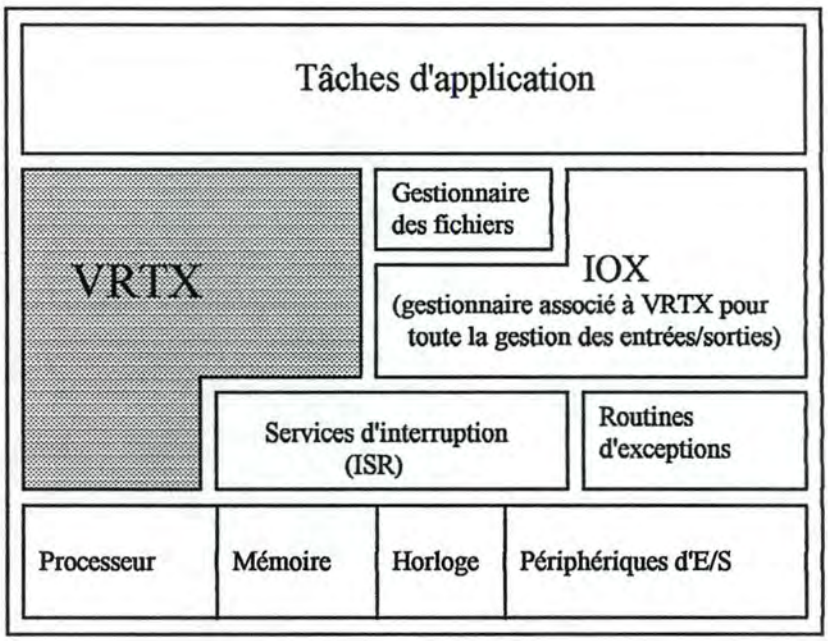
3.1.4.1. Description d'un exécutif temps réel : le VRTX32

3.1.4.1.1. Présentation

VRTX32, abréviation de Versatile Real-Time eXecutive, de la société Ready Systems, est un exécutif temps réel offrant toutes les caractéristiques de la définition d'un exécutif : gestion de tâche, communication et synchronisation par message, allocation dynamique de la mémoire, ordonnancement par priorité, traitement des interruptions (avec s'il le faut appel explicite à l'ordonnanceur, dans le cas où il y a modifications de la liste des tâches actives), une horloge de commande temps réel.

C'est l'exécutif le plus répandu dans le milieu des systèmes embarqués. VRTX32 est disponible pour tous les types de microprocesseurs 16 et 32 bits. L'utilisation de sa librairie des fonctions en langage C et l'usage de ce dernier langage rendent le développement d'applications multitâches indépendant d'une architecture particulière. VRTX32 fournit en plus un grand nombre de services supplémentaires, dont la gestion de la mémoire ou celle des entrées/sorties. Etant un exécutif; VRTX offre un environnement de développement convivial.

Nous présentons ci-dessous la structure générale de VRTX, avec ses modules d'extension :



3.2. SIMULATION

3.2.1. Introduction

Jusqu'à présent le seul moyen, chez SCHINDLER, de tester le fonctionnement des ascenseurs et des programmes de commande associés étaient de construire un prototype. Ce procédé coûteux, parfois dangereux, exige de longues périodes d'observations pour obtenir des renseignements significatifs et parfois ne permet pas de tester toutes les hypothèses. Avec les ordinateurs, il est possible d'étudier le fonctionnement d'un système réel, dans notre cas d'un ascenseur, sans le construire physiquement. Ainsi il sera possible de tester différentes hypothèses sans danger et en plus il sera possible d'obtenir tous les renseignements nécessaires et parfois même d'avoir des renseignements qu'il serait impossible d'obtenir par des observations.

On fera donc de la simulation pour imiter la réalité. Pour cela, on commencera par faire un modèle du système à simuler. Il faudra donc décrire le fonctionnement du système à simuler de manière logique et tenir compte de tous les facteurs qui entrent en jeu. Par exemple, le déplacement de l'ascenseur d'un étage à l'autre, l'ouverture des portes, etc.

Pour étudier le trafic des passagers, il ne serait pas raisonnable de construire un bâtiment rien que pour faire ce genre d'études. On pourrait les faire dans une installation existante, mais il n'est pas sûr qu'on puise obtenir des résultats exploitables. Les modèles mathématiques et les distributions statistiques sont donc les bienvenus pour nous aider à imiter le réel. Les passagers arrivant dans ce système de manière stochastique, nous utiliserons un modèle pseudo-aléatoire pour générer leurs arrivées et leurs destinations dans les différents étages d'un bâtiment fictif mais vraisemblable. Ce qui nous conduit à la définition suivante de la simulation.

3.2.2. Définition

La simulation est une technique visant à développer et à utiliser des modèles en vue d'aider à évaluer des hypothèses et à étudier des systèmes dynamiques.

Cette technique est particulièrement appropriée lorsque le processus à simuler est trop dangereux, coûteux à l'usage ou non disponible. La simulation permet de faire des tests préalables à l'installation d'un ascenseur ou d'un groupe d'ascenseurs dans un bâtiment à construire. Par exemple, on pourrait étudier l'effet d'installer un restaurant au deuxième étage ou au dernier étage d'un immeuble de 18 étages sur la capacité de transport et sur le temps d'attente des futurs usagers.

3.2.3. Différents types de simulation

L'introduction de cette section nous a permis de mettre en lumière deux catégories de simulation selon la nature du problème et le modèle choisi. On peut regrouper toutes les techniques de simulations en quatre catégories. Nous les représentons dans le tableau ci-dessous :

PROBLEME

MODELE		**déterministe**		**stochastique**
	analytique	Mathématiques Programmation linéaire		Probabilité Statistique
	analogique	Physique Mécanique		
	aléatoire	Monte-Carlo		Simulation

Pour l'ascenseur qui est un problème déterministe, on prendra donc un modèle analytique. Et pour l'étude du trafic des passagers, on fera de la simulation, proprement dite.

3.2.4. La prise en compte du temps en simulation

Le type de technique de simulation choisie pour étudier un ascenseur est dit discret, par opposition à la simulation continue qui peut se faire à l'aide des équations algébriques ou différentielles.
Une simulation discrète peut être basée soit sur le **temps**, soit sur les **événements**.

Dans une simulation basée sur les événements, dite aussi simulation à incrément de temps variable, la mise à jour du système se fait chaque fois qu'un événement survient; on pourrait avoir une liste des événements avec leur calendrier et on va pouvoir passer d'un événement à l'autre à la vitesse de traitement de l'ordinateur.

Tandis que dans une simulation basée sur le temps, il y a une horloge qui tourne avec un incrément de temps fixe.

La simulation des ascenseurs conduit à traiter beaucoup d'informations de nature différente. Certaines demandent à être traiter immédiatement, comme la réception d'une allocation du répartiteur d'appel; d'autres peuvent attendre un certain temps sans dommage, par exemple, le retour au niveau principal d'entrée quand il n'y a aucun passager dans le système.

Pour la simplicité de l'implémentation, nous avons choisi la simulation basée sur le temps. Le seul problème est de trouver un incrément de temps capable de servir tous les événements dans un temps raisonnable.

3.3. LA LOGIQUE FLOUE

Nous allons proposer dans le prochain chapitre une méthode de détection de type de trafic en cours qui va utiliser la logique floue dont nous exposons les fondements dans cette section.

La dénomination de cette technique est malheureuse car elle fait penser qu'on fait appel à une méthode qui n'est pas très rigoureuse. Or, la logique floue a des bases mathématiques solides. Elle tire toute sa validité de la théorie des sous-ensembles flous.

La logique floue, qui est à la fois souple, correcte et rigoureuse, s'applique aux problèmes mal définis ou difficilement représentables par des équations mathématiques. Elle est aussi adaptée dans les domaines où l'expertise humaine s'énonce plus facilement en termes linguistiques, en langage naturel, qu'en chiffres ou mesures bien précises.

Elle a été lancée en 1965, par L.A. Zadeh, professeur à l'Université de Californie à Berkeley, pour résoudre les problèmes posés par des connaissances imprécises et incertaines ("très probable"). Sa pratique dans les problèmes de commande a montré qu'elle était aussi bien adaptée aux problèmes posés par des connaissances imprécises ("une maison située aux environs de Namur"), vagues ("un trafic dense") mais certaines.

3.3.1. La théorie des sous-ensembles flous (fuzzy set)

Dans la théorie des ensembles classiques, un ensemble est défini comme étant une collection finie ou infinie, dénombrable ou non-dénombrable d'objets. Un objet ou élément peut appartenir ou ne pas appartenir à un ensemble donné.

Un ensemble classique peut être défini par énumération ou analytiquement. Définir un ensemble par énumération revient à faire la liste précise de tous ses éléments, tandis que le définir analytiquement se fera en donnant une propriété générale résumant les caractéristiques de tous ses éléments. Nous pourrons définir, par exemple, l'ensemble A, comme étant l'ensemble de tous les nombres entiers inférieurs ou égaux à 3. Ceci est une définition analytique. Par énumération, on dira que A est l'ensemble dont les éléments sont 0,1,2,3. Notation : A={0,1,2,3}.

Prenons un nombre entier quelconque x. Ce nombre, en théorie classique des ensembles, peut être défini aussi par la fonction caractéristique μ :

$x \in A$ alors $\mu_A(x)=1$

$x \notin A$ alors $\mu_A(x)=0$.

Il n'y a donc que deux possibilités pour caractériser l'appartenance d'un élément à un ensemble dans la théorie classique des ensembles.

Dans la théorie des sous-ensembles flous, la notion d'appartenance d'un élément à un sous-ensemble donné n'est pas aussi dichotomique. On peut indiquer le degré d'appartenance d'un élément à ce sous-ensemble flou.

3.3.1.1. Définition d'un sous-ensemble flou

Soit X, un ensemble donné, x, les éléments de cet ensemble. $\tilde{A}$, étant un sous-ensemble flou de X ($\tilde{A} \subset X$), peut être défini par un ensemble de couples ordonnés défini comme suit :

$\tilde{A} = \{(x, \mu_{\tilde{A}}(x)) | x \in X\}$.

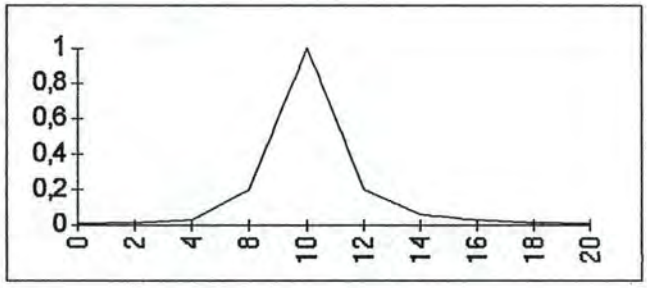
$\mu_{\tilde{A}}(x)$ est appelée la **fonction caractéristique** ou degré d'appartenance (aussi degré de vérité ou degré de compatibilité) de x dans le sous-ensemble $\tilde{A}$ qui associe au domaine de départ X le domaine d'arrivée M. (Si M contient seulement deux points, alors $\tilde{A}$ n'est pas flou et $\mu_{\tilde{A}}(x) = \mu_A(x)$). Le domaine de la fonction caractéristique comprend tous les nombres réels non négatifs compris entre O et 1. $\mu_{\tilde{A}}(x) \in [0,1]$.

Exemple:

$\tilde{A}$ = "l'ensemble des nombres proche de 10".

$$\tilde{A} = \{(x, \mu_{\tilde{A}}(x)) \mid \mu_{\tilde{A}}(x) = \frac{1}{1 + (x-10)^2}\}$$

Ce qui peut-être représenté par :



3.3.1.2. Caractéristiques d'un sous-ensemble flou

Ces caractéristiques sont utilisées pour bien décrire un sous-ensemble flou $\tilde{A}$ de X. Elles permettent de le distinguer d'un sous-ensemble ordinaire ou classique de X.

La première de ces caractéristiques est appelée le **support** de $\tilde{A}$, c'est-à-dire l'ensemble des éléments de X qui appartiennent, au moins un peu, à $\tilde{A}$. Il est noté supp($\tilde{A}$). C'est la partie sur laquelle la fonction d'appartenance de X n'est pas nulle :

supp(A) = $\{\mu_{\tilde{A}}(x) > 0 \mid x \in X\}$

La deuxième caractéristique est ce qu'on appelle son **α-coupe** (α-level cut). Elle désigne le sous-ensemble ordinaire A_α de X qui regroupe tous les éléments appartenant au sous-ensemble flou $\tilde{A}$ avec un degré d'appartenance >= α :

$A_\alpha = \{\mu_{\tilde{A}}(x) >= \alpha \mid x \in X\}$.

La troisième caractéristique de $\tilde{A}$ est sa **hauteur**, notée $h(\tilde{A})$, c'est-à-dire le plus fort degré avec lequel un élément de X appartient à $\tilde{A}$:

$\tilde{A}) = \sup\{ \mu_{\tilde{A}}(x) : x \in X \}$

On dit que le sous-ensemble flou est **normalisé** si sa hauteur vaut 1.

L'ensemble de tous les éléments appartenant de manière absolue à $\tilde{A}$ est appelé le **noyau** de $\tilde{A}$ et noté $noy(\tilde{A})$:

$noy(\tilde{A}) = \{ x \in X \mid \mu_{\tilde{A}}(x) = 1\}$.

Si A est un sous-ensemble ordinaire de X, il est normalisé, identique à son support et à son noyau.

Une dernière caractéristique du sous-ensemble flou $\tilde{A}$ de X (lorsque X est fini) est sa **cardinalité,** évaluant le degré global avec lequel les éléments de X appartiennent à $\tilde{A}$. Elle est définie par :

$|\tilde{A}| = \sum_{x \in X} \mu_{\tilde{A}}(x)$.

3.3.1.3. Opérations sur les sous-ensembles flous

Comme la notion de sous-ensemble flou est une généralisation de la notion de l'ensemble classique de X, il est utile d'introduire les opérations sur les sous-ensembles flous qui sont équivalentes aux opérations classiques de la théorie des ensembles.

Soit $\tilde{A}$ et $\tilde{B}$, deux sous-ensembles de X. Nous définissons d'abord les notions d'inclusion et d'égalité de la façon suivante :

Inclusion : $\tilde{A} \subseteq \tilde{B}$ $\quad \forall x \in X \quad \mu_{\tilde{A}}(x) <= \mu_{\tilde{B}}(x)$.

Egalité : $\tilde{A} = \tilde{B}$ $\quad \forall x \in X \quad \mu_{\tilde{A}}(x) = \mu_{\tilde{B}}(x)$.

Les opérations d'intersection, d'union, de complémentation ont été définies de la façon suivante (nous donnons entre parenthèses leur correspondant logique) :

Intersection (ET) :

$\tilde{A} \cap \tilde{B}$: $\quad \forall x \in X \quad \mu_{\tilde{A} \cap \tilde{B}}(x) = \min(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x))$

Union (OU) :

$\tilde{A} \cup \tilde{B}$: $\quad \forall x \in X \quad \mu_{\tilde{A} \cup \tilde{B}}(x) = \max(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x))$

Complémentation (NON) :

$\forall x \in X \quad \mu_{\tilde{A}^c}(x) = 1 - \mu_{\tilde{A}}(x)$

3.3.1.4. Les variables linguistiques

Cette notion sert à modéliser les connaissances imprécises ou vagues sur une variable dont la valeur précise est inconnue. Soulignons que la représentation, puis le traitement, de connaissances exprimées symboliquement et non numériquement, passe par l'utilisation des variables linguistiques.

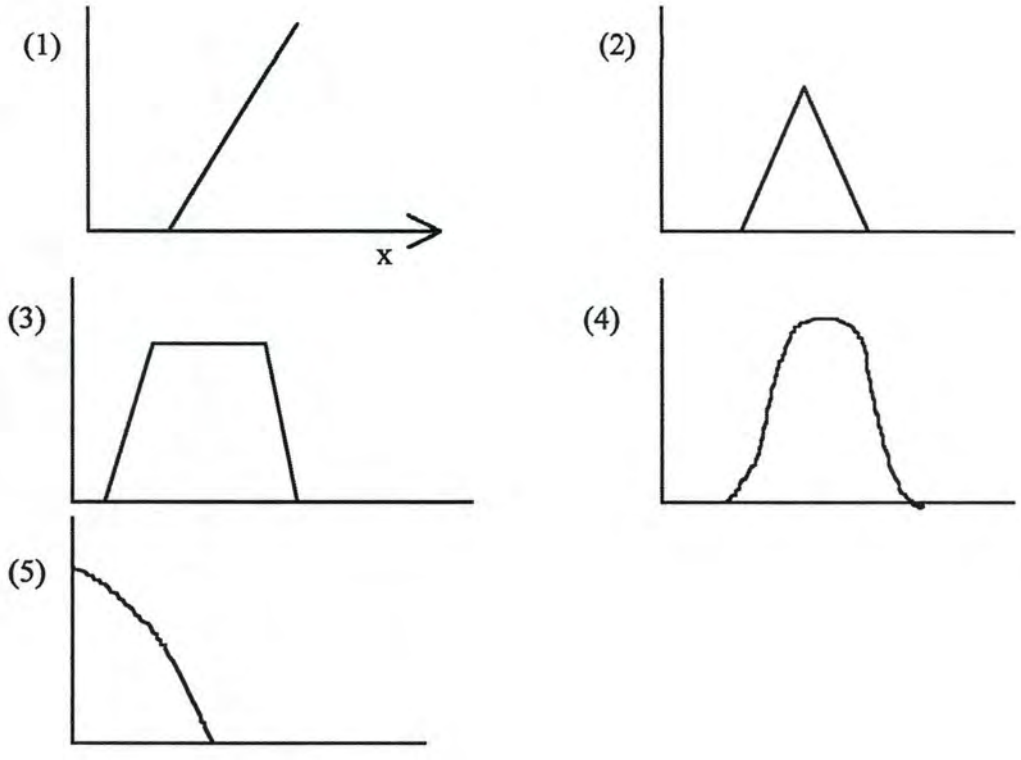
Définition.

Une variable linguistique est représentée par un triplet (V, X, T_V), dans lequel V est une variable définie sur un ensemble de référence X (qui peut-être un ensemble des nombres entiers ou des nombres réels ...), sa valeur pouvant être n'importe quel élément de X; $T_V = \{ A_1, A_2, ... \}$ est un ensemble, fini ou infini, de sous-ensembles flous de X, qui sont utilisés pour caractériser V, définissant des restrictions des valeurs que prend V dans X. Par exemple, si l'on utilise "chaud", on restreint l'ensemble des valeurs possibles pour la température aux nombres positifs.

3.3.2. Application de la logique floue

Nous allons indiquer les principales étapes nécessaires pour concevoir une application utilisant la logique floue.

Premièrement, nous devons identifier les principales variables de commande et déterminer les sous-ensembles qui les définissent complètement. Par exemple, une variable linguistique, comme "taille" définie par l'ensemble des sous-ensembles flous {petit, moyen, grand} pour une certaine application pourrait requérir une subdivision plus fine, comme {très petit, petit, moyen, grand, très grand} dans un autre type d'application. Nous devons en plus choisir le type de fonction caractéristique apte à représenter correctement notre variable. Il peut s'agir d'une fonction monotone (1), triangulaire (2), trapézoïdale (3), en cloche (4) ou sinusoïdale (5).



Le nombre de fonctions d'appartenance de sous-ensembles flous choisies et leur forme dépendent de la précision, du temps de réponse et de la stabilité, de la facilité d'implémentation et de la maintenance qu'on souhaite accorder au système.

Dans la plupart des applications, on travaille avec les fonctions caractéristiques en forme de triangle (baptisée "nombre flou")et de trapèze (baptisé "intervalle flou") car elles permettent une interprétation aisée.

Deuxièmement, nous devons développer une base de connaissance et de règles d'évaluation à partir des variables linguistiques de commande. Cette base de connaissance et des règles peut-être constituée en interrogeant les experts du domaine en question ou les opérateurs qualifiés dans la conduite du processus existant. L'ensemble de ces règles sont définies sous forme de **If ... Then** (Condition ==> Action). Par exemple, si on voulait installer un système automatique de commande de chauffage dans un immeuble, on pourrait employer les règles suivantes :

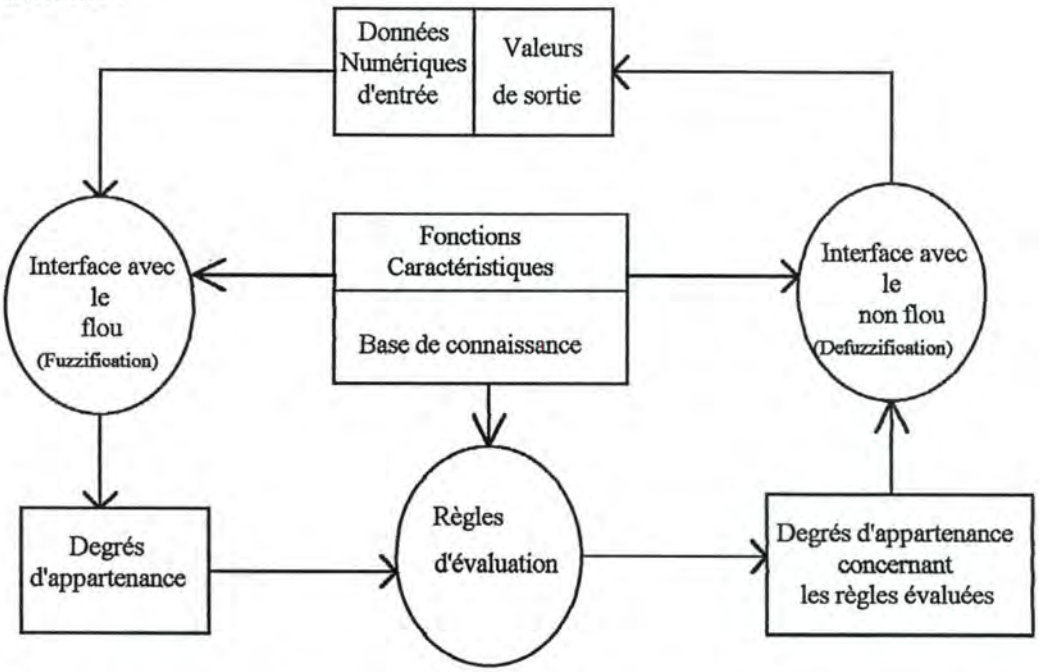
Condition : la température baisse lentement.

Action : augmenter un peu la chaleur fournie par le brûleur.

Condition : la température est trop basse.

Action : mettre le brûleur au maximum.

La configuration générale d'un contrôleur flou peut se présenter de la manière suivante :

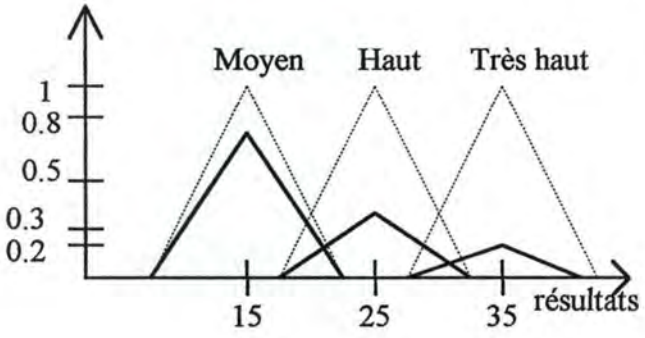


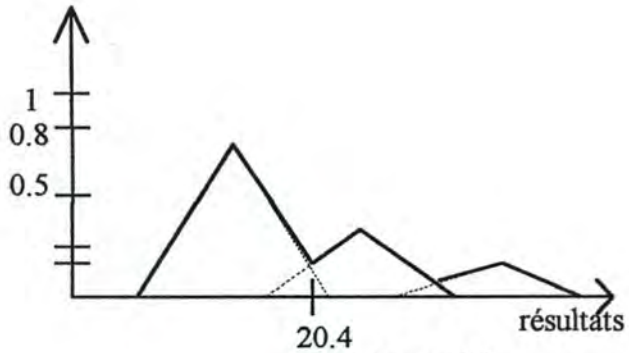
Il serait intéressant d'expliquer comment se déroule l'opération de passage du flou au concret ("Defuzzification"). Considérons la situation ci-dessous : après évaluation des règles de connaissance, nous avons les résultats suivants :
(0.8, moyen), (0.3, haut), (0.2, très haut). Pour pouvoir calculer la valeur de sortie de ces résultats, il faudra trouver une manière de les agréger. Deux méthodes principales ont été proposées dans la littérature concernant la logique floue :

- **la méthode de centre de gravité min-max** et
- **la méthode de centre de gravité produit-somme**.

La méthode de centre de gravité min-max ne donne pas toujours de bons résultats. Elle consiste à prendre les minima de chaque règle considérée et ensuite de prendre le maximum de ces minima.

La méthode de centre de gravité produit-somme est illustrée ci-dessous :





Explication : Pour chaque ensemble flou nous redessinons le triangle qui indique le degré d'appartenance de l'ensemble. Nous multiplions la hauteur du triangle par la valeur du degré d'appartenance de la règle évaluée (cfr. première figure). Le résultat est donc une série de 3 triangles plus petits (un triangle par règle de connaissance). On additionne ensuite les surfaces de ces trois petits triangles (cfr. deuxième figure) et on détermine le centre de gravité de la nouvelle figure. La valeur du centre de gravité sur l'axe des abscisses est la valeur recherchée.

Tous ces calculs sont résumés par la formule suivante :

$$\text{centre de gravite} = \frac{\sum_{x=1}^{q} (a(x).\text{valeur moyenne}.\text{aire})}{\sum_{x=1}^{q} (a(x).\text{aire})}$$

où

$\alpha(x)$ est le résultat de la règle de connaissance x,

q est le nombre de règles de connaissance dont la valeur se rapporte à la valeur de commande à calculer

la valeur moyenne est la valeur qui se trouve directement en-dessous du sommet du triangle auquel se rapporte la règle de connaissance x et

aire est en fait la surface du triangle concerné.

3.3.3. Conclusion

La logique floue, avec la théorie des sous-ensembles flous dont elle découle, est un sujet vaste dont nous n'avons fait qu'effleurer ici la substance. Le lecteur intéressé trouvera dans la bibliographie quelques références assez conséquentes sur le sujet.

Les applications pratiques de ses résultats, surtout au Japon, qui donnent des solutions aussi bonnes et parfois meilleures que celles des applications utilisant les techniques classiques, ne se comptent plus. On peut relever ses applications dans le domaine de contrôle de machines-outils, de bras de robots, d'usines (cimenteries, usines chimiques,...), de véhicules sans pilotes, d'appareils photographiques, d'appareils électroménagers, d'installations domestiques (douches, climatisation), d'ordinateurs à écriture manuelle sans clavier, ...

3.4. ENVIRONNEMENT DE DEVELOPPEMENT

Nous allons décrire dans cette section les points suivants :

le choix du langage de programmation et

l'environnement de programmation dans lequel nous avons évolué.

3.4.1. langage de programmation

Nous avons employé le langage C. Le problème de choix ne s'est pas posé car c'était le langage de développement déjà utilisé au Laboratoire où l'on développe non seulement des logiciels mais aussi le matériel (hardware) associé, c'est-à-dire des cartes électroniques de commande d'ascenseurs.

Le langage C est un langage très prisé dans le milieu industriel depuis un certain temps, car il permet de travailler à un niveau proche du matériel et de s'affranchir ainsi de l'assembleur qui dominait auparavant dans le développement des programmes.

Comme certaines librairies de fonctions utilisées, par exemple le programme de communication entre tâches, VCOM, avaient été conçues en Suisse dans un environnement VMS (Virtual Memory System, un système d'exploitation du VAX) et Motorola 68000 en C et en vue d'assurer la compatibilité des codes sources, le choix du compilateur s'était porté sur le MCC68K de Microtec (la version de l'exécutif temps réel utilisée, VRTX32, est une version du 68000 aussi). La spécificité de ce compilateur est qu'il permet de générer sur PC (personal computer), c'est-à-dire avec un microprocesseur de la famille INTEL 80x86, un code objet pour le microprocesseur 68x00 de Motorola.

3.4.2. Emulation du 68000 sur le 80x86

Ce code objet peut-être directement exécuté sur place grâce à l'émulateur HP64700. Un émulateur est un appareil qui permet de remplacer dans un autre microprocesseur (ici, le 80386), le microprocesseur cible (dans ce cas, le 68000) en vue d'aider à tester les logiciels et les matériels pour celui-ci. Le but de l'émulateur est d'opérer exactement comme le processeur qu'il remplace, en donnant notamment des informations sur l'état des signaux électriques qui parcourent le bus. Il permet en plus de voir le contenu des registres, de la mémoire du système cible et les ressources utilisées pour les entrées/sorties.

4. Mise en oeuvre

4.1. INTRODUCTION

Nous expliquerons dans ce chapitre les choix que nous avons dus faire pour le développement et l'implémentation du projet.

Après toute l'analyse du problème faite dans les chapitres précédents, la découpe en différents modules est facilitée par la méthode d'analyse qui avait été adoptée : les différentes composantes du groupe d'ascenseurs que nous avons mises en évidence deviennent des modules. Ceux-ci formeront les tâches de la plate-forme de simulation que nous développons.

Ces modules sont :

- le LSP_CA, le générateur d'appels,
- le LSP_LS, le simulateur d'ascenseurs,
- le LSP_TA, l'analyseur de trafic,
- le LSP_TD, le détecteur de trafic,
- le LSP_GA, le répartiteur d'appel ou l'algorithme de groupe.

Notons que le LSP_CFG, le configurateur du bâtiment sera implémenté sous forme de fichier en-tête (header file) et les différents paramètres sur le bâtiment et les ascenseurs seront des constantes qu'on pourra aller modifier directement dans ce fichier sans perturber les programmes.

J'ai été chargé de développer le 'Group Algorithm' et de proposer un détecteur dynamique de type de trafic.

4.2. ALGORITHMES DE GROUPE

4.2.1. Inventaire de quelques algorithmes de groupe

Divers algorithmes ont déjà été mis au point. L'objectif de la plupart de ces algorithmes est d'éviter un temps d'attente excessif aux clients. Mais, avant tout, posons-nous la question suivante : Comment estime-t-on qu'un temps d'attente n'est pas trop excessif ? Pour y répondre, on doit prendre en compte des données sur la psychologie humaine, du type et de la hauteur de bâtiments considérés (bâtiment commercial, résidentiel ou bureau),... Quelque soit le type de building, on estime de manière générale, qu'un temps d'attente de 30 secondes peut être considéré comme étant une moyenne raisonnable et plus de 180 secondes peut être considéré comme étant excessif.

On demande aussi à l'algorithme de groupe d'éviter le phénomène de 'bunching', qui est caractérisé par la circulation simultanée dans la même direction de toutes les cabines d'ascenseurs du groupe.

Signalons pour terminer que tous les algorithmes que nous verrons dans cette section sont conçus pour un trafic normal ("interfloor"). En effet, pour faire face, aux trafic de pointe montée et de pointe descente, d'autres règles sont mises en oeuvre. Ce sont des règles beaucoup plus simples que dans le cas du trafic normal. Notons cependant que, même en situation de pointe montée ou de pointe descente, on pourrait garder un certain nombre de cabines en fonctionnement normal. Dans ce cas la situation devient plus complexe et de même que les algorithmes à développer.

Nous ne tiendrons pas compte de toutes ces particularités ici et maintenant.

4.2.1.1. Algorithme de zones.

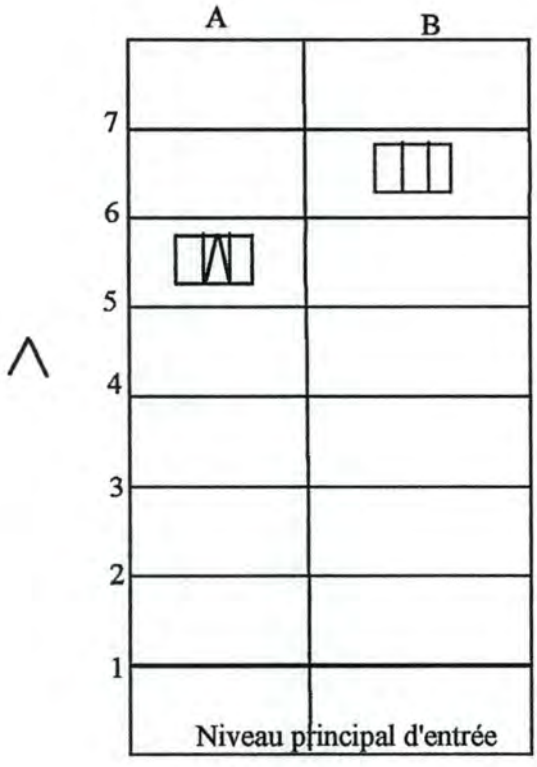
4.2.1.1.1. PRINCIPE

C'est un algorithme assez simple car il ne tient compte que deux paramètres : la distance et la direction. De plus, cet algorithme assigne les cabines d'ascenseurs aux appels des utilisateurs une fois pour toutes. Il ne fait que des allocations.

Il a été baptisé algorithme de zone car à chaque appel, on partage le bâtiment en deux zones selon la position et la direction de la cabine et de l'appel : une zone susceptible de prendre l'appel et une zone de non-disponibilité.

4.2.1.1.2. METHODE D'ASSIGNATION ET APPLICATION

Supposons que nous ayons un appel palier "montée" au 4ème étage. Les 2 cabines sont en déplacement. L'une (A) est arrivée au 5ème étage et monte vers le 7ème. L'autre (B) est au repos eu 6ème. Nous avons donc 4 tableaux de zones de service : 2 pour la cabine A (un pour les appels "montées" et l'autre pour les appels descentes) et 2 pour la cabine B.



: cabine immobile et libre

: cabine en montée

: appel monté

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

Tableau de zone montée pour A

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

Tableau de zone descente pour A

1	2	3	4	5	6	7
1	1	1	1	1	1	1

Tableau de zone montée pour B

1	2	3	4	5	6	7
1	1	1	1	1	1	1

Tableau de zone descente pour B

0 : signifie que la cabine X n'est pas disponible pour ce niveau

1 : signifie que la cabine X est disponible pour ce niveau

Comme l'appel apparaît en montée au quatrième étage, nous avons aussi un tableau des appels "montées" et un autre tableau des appels descentes.

Dans le tableau des appels "montées", on met 1 dans la case correspondant au 4ème étage et on fait un ET LOGIQUE entre ce tableau et les tableaux "montées" de A et de B.

Nous aurons donc un 1 avec le tableau de B. Ce sera donc l'ascenseur B qui servira cet appel.

Il y a en plus d'autres règles. Comme par exemple, si les deux cabines sont disponibles pour servir l'appel, cela sera la cabine la plus proche qui desservira l'appel ou si elles étaient dans la même position, cela serait la première suivant l'ordre alphabétique qui desservirait l'appel.

Si au moment des calculs, il n'y a aucun ascenseur disponible, on attendra qu'il y en ait un. Il est à noter que la détermination des zones se fait tout le temps.

4.2.1.2. Algorithme de calcul de coûts.

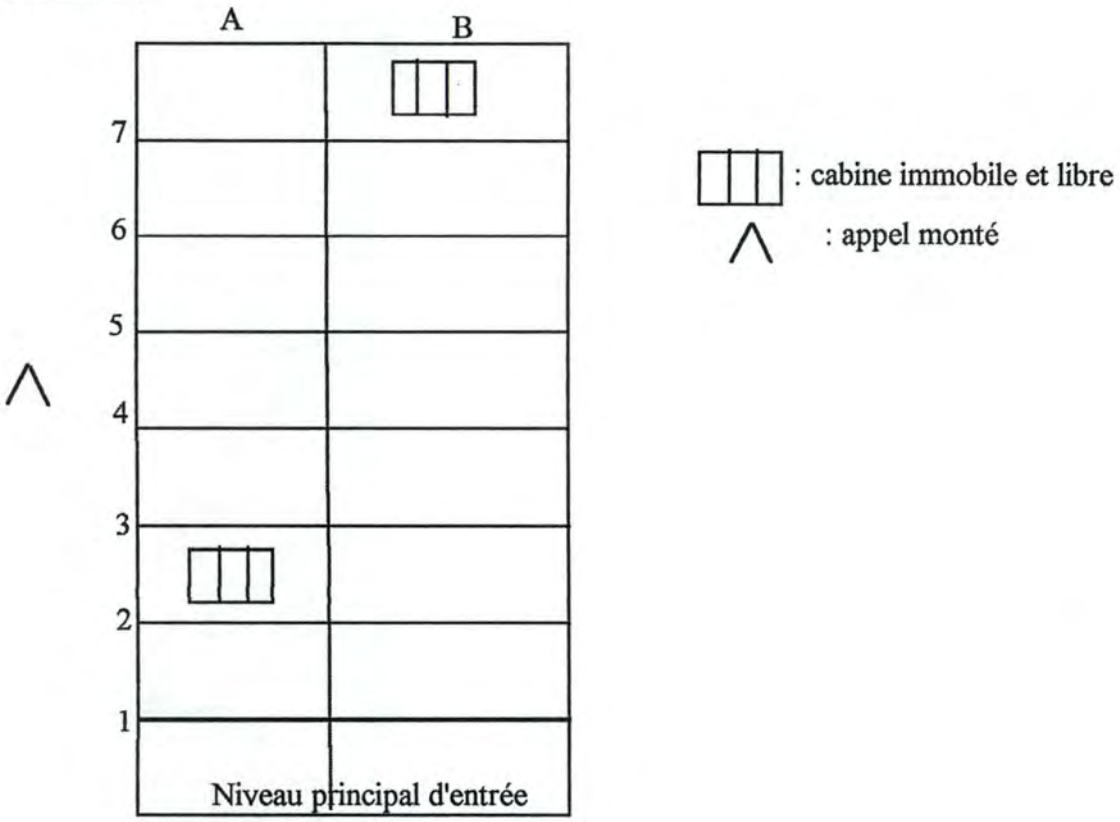
4.2.1.2.1. PRINCIPE

C'est un algorithme d'allocation/désallocation. Cela veut dire qu'un appel n'est pas attribué à une cabine de manière définitive tant qu'il n'est pas encore servi, ou du point de vue de la cabine, tant qu'elle n'a pas encore déclenché sa séquence de ralentissement pour servir cet appel. A tout moment, on peut l'allouer à une autre cabine plus avantageuse en terme de coût.

4.2.1.2.2. METHODE DE CALCUL ET APPLICATION

Le calcul de coût se fait de la manière suivante :

Scénario 1.



Nous avons deux cabines A et B. A est arrêtée au 2ème étage et B au 6ème. Leurs portes sont fermées. Un appel "montée" survient au 4ème. Le total des coûts est déterminé uniquement par la durée de parcours.

Supposons qu'il faut 5 secondes pour le démarrage et le freinage et 1 seconde pour parcourir un étage.

La cabine A qui est à 2 étages de l'appel aura les coûts suivants :

coût extérieur [1] : 5 + 2 = 7

coût intérieur [2] : 0

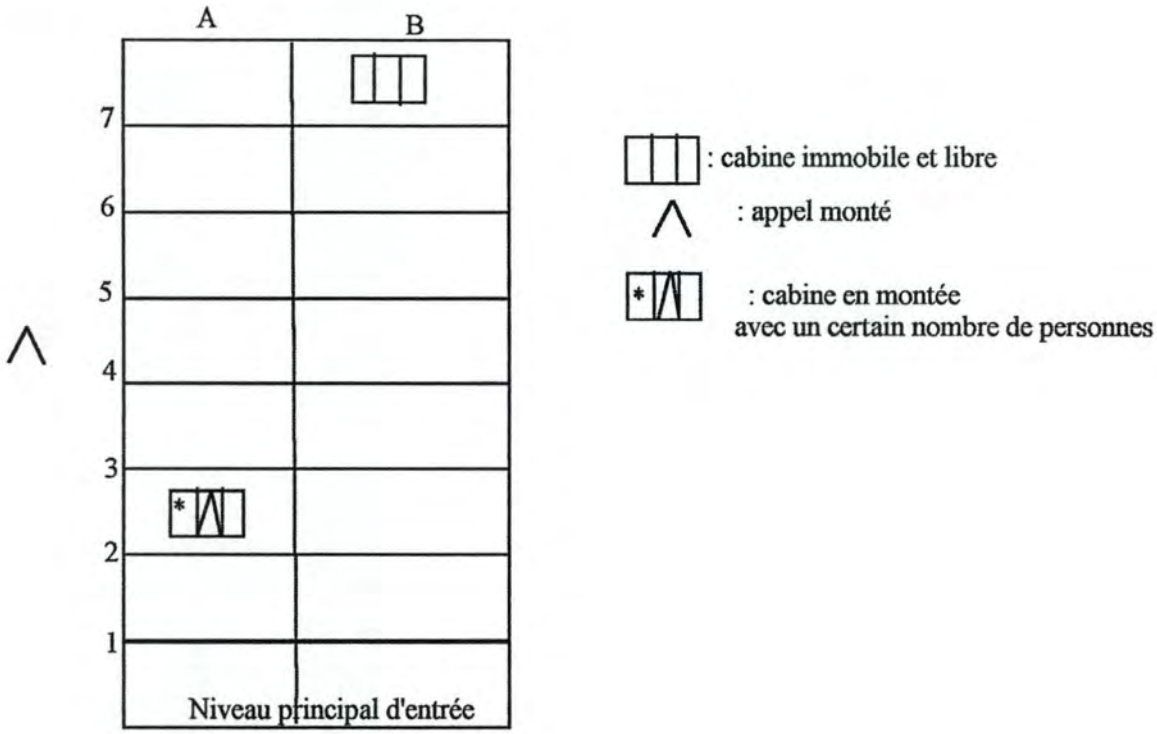
coût total = 7 + 0 = 7

La cabine B qui est à 3 étages de l'appel aura quant à elle, un coût total égal à 8.

A ayant le coût le plus bas, elle est favorite et prend donc l'appel.

[1]coût pour aller desservir l'appel
[2]coût dû aux appels cabines et aux allocations

Scénario 2.



A démarre avec 3 personnes à bord du 2ème étage.

B est au repos au 7ème.

Un appel "montée" apparaît au 4ème étage.

Calcul de coût pour la cabine A :

coût extérieur : 7

coût intérieur : 3 x 10 = 30 (10, est le coût par passager)

coût total == 30 + 6 = 36.

Calcul de coût pour la cabine B :

coût extérieur : 8

coût intérieur : 0

coût total == 8.

C'est la cabine B qui ira desservir l'appel (puisqu'elle le coût le plus bas).

Scénario 3.

Même situation qu'au scénario 2 mais la cabine A a un appel cabine pour le niveau 4.

Calcul de coût pour A :

coût extérieur : 7

coût intérieur : 0 (coïncidence d'un appel palier et d'un ordre cabine)

coût total == 7.

Le coût total pour la cabine B reste inchangé.

L'appel sera alloué à la cabine A.

4.2.1.3. Algorithme d'estimation du temps d'arrivée.

4.2.1.3.1. PRINCIPE

C'est aussi un algorithme d'allocation/désallocation, basé sur le principe suivant. Dès qu'un appel apparaît, on essaie d'estimer le temps qu'il faut à chacune des cabines d'ascenseur du groupe pour le desservir. La cabine pouvant arriver la première pour desservir l'appel est favorite. Ce calcul est répété pour tous les appels tant qu'ils n'ont pas encore été servis et si au cours d'un calcul ultérieur une cabine plus avantageuse était trouvée, on pourrait lui assigner cet appel en la désallouant à la cabine précédemment choisie pour autant que le bénéfice soit vraiment significatif.

4.2.1.3.2. METHODE DE CALCUL ET APPLICATION

voir l'algorithme.

4.2.1.3.3. ALGORITHME DETAILLE

Algorithme d'allocation d'une cabine à un instant donné.

0. Réception d'un appel palier au niveau NV.

1. Tester la disponibilité des cabines, c'est à dire pour une cabine
   voir si elle effectue une course prioritaire(maintenance, réservation, incendie) ou non,
   si elle est en pleine charge ou non.

2. Si on ne trouve pas de cabine disponible, reprendre au point 1 sinon continuer au point suivant.

3. **Calcul des temps d'arrivée.**

Pour chaque cabine disponible :

3.1. demander le niveau de son prochain arrêt possible (FIRSTSTOP).

3.2. demander son statut.

Si elle monte alors :

Si la position de NV >= FIRSTSTOP alors
aller au point 4
sinon
prendre la cabine suivante disponible et aller au point 2.

Si elle descend alors :

Si la position de NV <= FIRSTSTOP alors

aller au point 4

sinon

prendre la cabine suivante disponible et aller au point 2.

Si la cabine est à l'arrêt, aller au point 4.

4. **Calcul du temps d'arrivée pour la cabine en cours.**

4.1. Rechercher le ou les temps de parcours (TPARCOURS) dans la table des temps de parcours(TABTP) des cabines d'un niveau à l'autre.

4.2. Lire le statut de la porte de la cabine, pour déterminer sa contribution au coût total. Il est entendu que le temps d'ouverture (TPOUV) et le temps de fermeture d'une porte (TPFERM) pour une cabine en mouvement sont nuls.

4.3. Trois cas possibles :

4.3.1. la demande va dans la direction de déplacement de la cabine.

Si PCAB + 1 < PAC < NV (c'est à dire qu'il y a des ordres intermédiaires) alors

$$TT = \sum_{i=1}^{nc-1} TC_{n[i],n[i+1]} + TP * nc + TE_{n[nc],m} \qquad (1)$$

sous la condition: n[nc]<m

sinon

$$TT = TE_{k,m} \qquad (2)$$

où PCAB et k indique la position courante de la cabine,

PAC : la position du prochain appel cabine, c'est à dire premier arrêt certain,

TT = le temps total,

nc = nombre d'ordre cabine situé entre la position de la cabine et la position de l'appel extérieur,

$TC_{i,j}$ = temps pour desservir un ordre cabine de l'étage i à l'étage j.

Lire ce temps dans la table des temps de parcours (TABTP).,

TP = TPOUV + TPFERM, temps total d'ouverture et de fermeture d'une porte,

$TE_{i,j}$ = temps pour desservir un appel extérieur de l'étage i à l'étage j,

m = NV, position de l'appel extérieur,

n[ ] = table contenant les ordres pour la cabine.

Attention : Si un appel extérieur a été alloué à la cabine considérée, il peut être considéré comme un ordre cabine un peu spécial.

4.3.2. la demande va dans la direction opposée de la cabine alors tous les temps intermédiaires des ordres cabines sont à additionner.

D'où : on utilise les mêmes formules qu'au point 4.3.1. sans tenir compte de la condition.

4.3.3. Si la cabine est au repos : appliquer la formule **(2)**

5. Comparaison des temps d'arrivée (coûts)

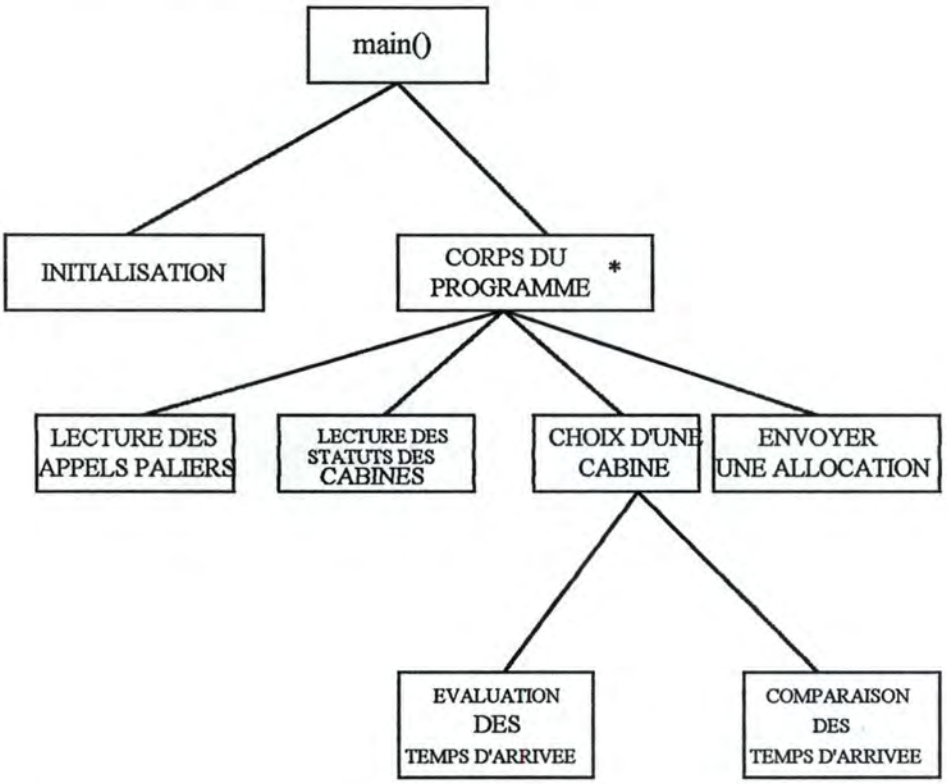
L'algorithme fournit un temps TT pour un appel palier pour chaque cabine disponible. Tous les résultats sont mémorisés dans une table des coûts. (TABCOUT[ ]).

A la fin, on compare les coûts et on fait une première allocation à la cabine offrant le coût le plus bas ou on désalloue un appel à une cabine au profit d'une autre.

La désallocation/ré-allocation est effectuée pour un appel alloué précédemment à une cabine et non encore servi au profit d'une autre offrant le coût le plus bas.

Note : Essayer d'éviter l'effet de "yo-yo" en allouant et désallouant successivement un même appel à des cabines différentes pour cela prévoir une différence de temps seuil au-dessous de laquelle aucune désallocation ne pourra être effectuée. On essaiera de favoriser la cabine ayant déjà obtenue l'allocation.

4.2.1.3.4. STRUCTURE DU PROGRAMME



(*) : bouclage infini du corps du programme, tant qu'il y a un appel palier à servir.

4.2.1.3.5. Structure des données

Les appels sont rangés dans une liste doublement chaînée implémentée sous la forme d'un tableau avec les particularités suivantes :

- la borne supérieure du tableau vaut 2*MAX_FLOOR+1;
- les appels pour monter, sont indiqués par les indices du tableau, qui correspondent au numéro des étages (c'est à dire un chiffre entre 1 et MAX_FLOOR);

- les appels descentes sont indiqués par les indices supérieurs à MAX_FLOOR.

Par exemple, un appel descente du 3ème étage sera indiqué dans le tableau à l'indice numéro MAX_FLOOR+ 3.

Prenons MAX_FLOOR = 4.

	APPEL	SUIVANT	PRECEDENT
1	appel 1	7	0
2			
3			
4			
5			
6			
7	appel 2	0	1
8			
9			

Nous avons donc la structure suivante en C pour la gestion des appels paliers :

```
typedef struct{

        COST cost; /* le coût associé à cet appel  (le temps qu'il faut pour que
                la cabine choisie arrive au niveau de l'appel )*/

        LIFT_NUMBER lift_favorite; /* le numéro de la cabine qui dessert
                                l'appel */

        LSP_GA_DIRECTION dir; /* la direction de l'appel */

        LSP_GA_CALL_STATUS status; /* le statut de l'appel : est-il servi ?
                                        oui ou non; est-il alloué ? */

        LSP_GA_HALL_TIME        time1, /* l'heure de l'appel */

                                time2; /*  l'heure d'arrivée de la cabine */

        FL_NUMBER predcall, /* appel précédent */

                        nextcall; /* appel suivant */

} LSP_GA_ELEMENT_HALL_CALL;
```

Cette structure-ci correspond à APPEL, SUIVANT, PRECEDENT.

La liste chaînée complète est représenté par la structure :

```
typedef struct {
FL_NUMBER firstcall, /* indique le premier appel */
          lastcall; /* indique le dernier appel */
LSP_GA_ELEMENT_HALL_CALL  hc_list[DOUBLE_MAX_FLOOR + 1];
                /* la liste des appels paliers */
}LSP_GA_HALL_CALL;
```

A cette structure des données sont associées les fonctions suivantes :

- une fonction d'initialisation des appels :

  void **LSP_GA_Initialisation_Call**(LSP_GA_HALL_CALL *hc);

- une fonction d'insertion d'appel :

  void **LSP_GA_Add_Call**(LSP_GA_HALL_CALL *hc,
  FL_NUMBER i, LSP_GA_DIRECTION dir);

- une fonction de suppression d'appel :

  void **LSP_GA_Remove_Call**(LSP_GA_HALL_CALL *hc,
  FL_NUMBER i, LSP_GA_DIRECTION dir);

où **hc** est la liste doublement chaînée, **i** le niveau de l'appel et **dir** la direction de l'appel.

Toutes ces fonctions ont été testées rigoureusement et sont sensées être correctes.

Nous avons une variable globale, LSP_call, partagée entre le LSP_LS et le LSP_GA. C'est un tableau indiquant pour une cabine tous ses appels cabines.

Voici sa déclaration : unsigned char **LSP_call** [MAX_LIFT*MAX_FLOOR]

Les autres variables globales importantes sont :

COST **cost**[MAX_LIFT + 1] : contient les coûts pour chaque ascenseur.

LSP_GA_DISPONIBILITY **disp**[MAX_LIFT + 1]: indique si un ascenseur est disponible ou non.

4.2.1.3.5. SPECIFICITES DE NOTRE IMPLEMENTATION

1. Nous n'avons pas implémenté le point 4.3.2. de l'algorithme puisqu'il entrait en contradiction avec la pratique de la firme.

2. Dans la fonction de choix de cabine : nous utilisons une petite astuce pour ne pas commencer chaque fois par la même. Au départ, on faisait appel à la fonction d'évaluation en débutant toujours nos calculs par la première cabine. Si on avait 3 cabines, par exemple, les calculs se faisaient chaque fois dans la séquence : 1, 2 et 3. Ce qui pouvait avoir comme conséquence d'user davantage le premier ascenseur, puisque nous essayons de privilégier l'ascenseur ayant déjà reçu une allocation.

   Au lieu de cela, nous avons implémenté un mécanisme de permutation circulaire. Ainsi, la première fois, nous aurons la séquence d'évaluation : 1, 2 et 3. La deuxième fois : 2, 3, et 1.
   La troisième fois : 3, 2 et 1.
   Et ainsi de suite,...

Nous donnons ici son code :

```
#define lift_number 3
#define lift_number_less_1 (lift_number-1)
LIFT_NUMBER LSP_GA_Find_Cabin(LSP_LS_Hall_CALL
                *hall_call){
char n, j;
static start_value=0;
for (j=0; j<lift_number; j++){
        n = j + start_value;
        if (n > lift_number_less_1)
                n -= lift_number;
/*   ON MET ICI, LA PARTIE QUI UTILISE n   */
/* n vaut successivement : 0, 1, 2, 1, 2, 0, 2, 0, 1 */
}
if (start_value < lift_number_less_1)
        start_value++;
else
        start_value = 0;
}
```

4.2.2. Analyse des résultats

Ne disposant pas d'outils efficaces pour analyser les résultats, nous n'avons pas pu le faire de manière chiffrée. Toutefois, comme vous aurez l'occasion de le constater lors de la démonstration, après la présentation orale de ce travail, vous verrez que tous les appels enregistrés obtiennent leurs allocations après un temps fini et qu'il y a de temps en temps quelques désallocations d'appel, ce qui était prévisible. La fréquence de ces désallocations dépendent du seuil de tolérance introduite dans le programme par essai et erreur et aussi du mouvement et de la position des cabines d'ascenseurs par rapport aux appels. Ce seuil a pour valeur la différence entre le temps qui reste à la cabine favorite pour arriver au niveau de l'appel et le temps qu'il faut à la candidate favorite.

Nous avons introduit dans l'algorithme de groupe un embryon d'outils d'analyse, il nous permet de connaître le temps qu'il a fallu pour qu'une cabine arrive pour desservir un appel donné.

On pourrait grâce à ces données calculer le temps d'attente moyen des clients après un temps déterminé.

De plus, pour faire une bonne analyse, il aurait fallu plus d'un algorithme pour pouvoir faire des comparaisons des performances. Ce dont nous ne disposions pas, malheureusement, puisque nous ne sommes qu'encore au niveau des tests d'une partie de la plate-forme de simulation et que nous n'avons pas encore eu l'occasion d'implémenter d'autres algorithmes de groupe.

4.3. DETECTEUR DE TYPE DE TRAFIC

Nous allons proposer dans cette partie un détecteur de type de trafic. Rappelons que dans un groupe d'ascenseur, on peut distinguer 3 types principaux de trafic :

- le trafic en pointe montée (up-peak),
- le trafic en pointe descente (down-peak) et
- le trafic normal (interfloor).

Et pour chacun de ce type de trafic, on met en oeuvre un algorithme bien spécifique. Dans notre travail, nous avons développé un algorithme pour le trafic normal.

4.3.1. Règles de détection

Imaginons qu'on a installé un quadriplex dans un bâtiment de 15 étages au-dessus du niveau principal d'entrée. Chaque cabine d'ascenseur roule à la vitesse nominale de 3 m/s et possède une capacité nominale de 20 personnes. La hauteur de chaque étage est de 3 mètres. L'intervalle en pointe montée étant de 10 secondes, le nombre de passagers pouvant être transportés dans les 5 minutes est de 480 personnes; pendant le même temps, il pourrait aussi y avoir au plus 30 arrivées successives de cabines d'ascenseurs au niveau principal d'entrée.

Nous proposons la procédure suivante pour tenter de prédire le type de trafic en cours. Nous allons créer une tâche qui se réveillera toutes les deux minutes. Elle aura besoin des informations suivantes en entrée :

- le nombre d'arrivées des cabines au niveau principal d'entrée (NAC)
- le nombre de départ des cabines du niveau principal d'entrée (NDC) et
- la charge moyenne des cabines en nombre de passagers (CM).

Dans notre cas par exemple, après deux minutes, un trafic en pointe montée pourra être induit si le nombre de départ des cabines du niveau principal d'entrée est de 12 et si la charge moyenne est égale à 16.

De même, on pourra passer en pointe descente, si le nombre d'arrivée de cabines du niveau principal d'entrée et si la charge moyenne est légèrement inférieure à 16.

Par défaut, on est toujours en trafic normal.

Nous pouvons définir les sous-ensembles flous suivant pour le nombre d'arrivées et de départ de cabines du niveau principal d'entrée.

Nous exprimons le degré d'appartenance de différentes valeurs que nous allons avoir en entrée pour le NAC et le NDC (cfr. la figure ci-dessous) et pouvons les représenter par les quatre points du trapèze.

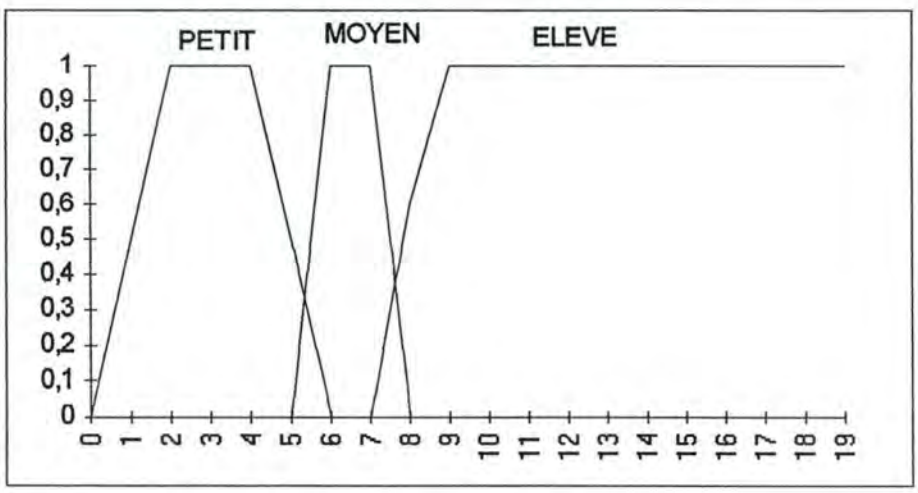
NAC et NDC:

PETIT : 0 3 4 6

MOYEN : 5 6 7 8

ELEVE : 7 9 20 21

L'usage est de recouvrir les sous-ensembles adjacents d'environ 25 %.

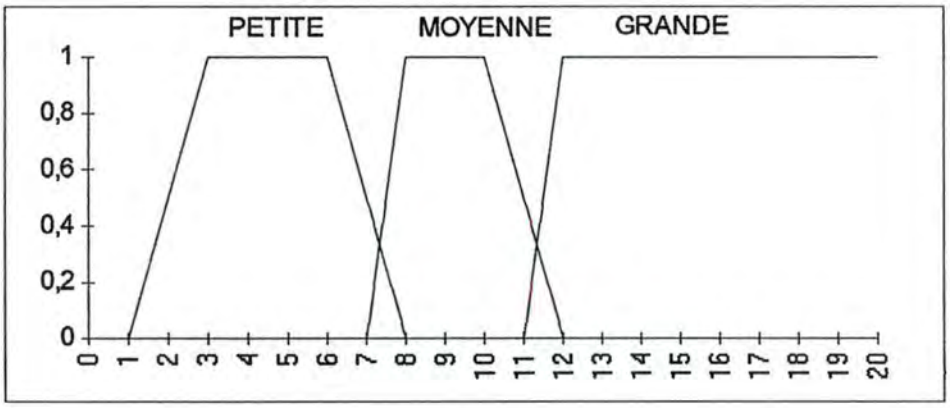


Pour la charge, nous pouvons dire qu'elle est :

PETITE : 0 5 7 10

MOYENNE : 7 8 10 12

GRANDE : 11 12 20 21



Comme ensemble de sortie, nous aurons, soit UP-PEAK, soit DOWN-PEAK.

Il restera à établir toutes les règles d'évaluation et à déterminer le type de trafic approprié suivant les données en entrée. L'implémentation de ces règles avec la méthode du centre de gravité min-max pourrait se faire sans difficulté.

Conclusions générales

Au terme de ce travail, nous pouvons tirer les enseignements suivants :
il n'existe pas d'algorithmes universels pour résoudre le problème d'allocation d'appel enregistré de cabines d'ascenseurs quand on ne connait que l'origine du client. Pour pouvoir optimiser le temps d'attente et aussi le temps de parcours, il faudrait un système qui puisse permettre de connaître et l'origine et la destination du client qui fait l'appel.

L'algorithme de groupe n'étant pas universel, il serait peut-être souhaitable d'adopter d'autres techniques de résolution de ce problème que les techniques de calculs classiques. Nous pensons notamment aux techniques d'intelligence artificielle, de systèmes-experts qui pourraient permettre au système d'apprendre de manière automatique et continue le profil du trafic du site où il est installé, car chaque bâtiment à un profil type de trafic selon sa fonction (résidence, commerce, bureau), et de pouvoir y répondre de façon appropriée.

Nous n'avons pas pu résoudre et implémenter tous les modules de notre plate-forme de simulation puisque nous avons dû nous plonger dans un autre univers avec d'autres références. Nous pensons notamment à la conception et au développement des systèmes multitâches et temps réels. Et nous avons exploré le domaine promoteur de la logique floue pour essayer de mettre en oeuvre d'autres types de solutions.

Ce travail de "débroussaillage" pourrait peut-être servir dans l'avenir aux personnes qui se lanceraient dans la conception et le développement de logiciels de groupe d'ascenseurs.

Bibliographie

Aho, Hopcroft, Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Massachussets, 1974.

Barney G.C. et dos Santos S.M., Elevator Traffic Analysis, Design And Control, Revised Second Edition, Peter Peregrinus Ltd., London, 1985.

Bertz R., Mathematical Methods for Elevator Traffic Simulation (Part One), in Elevator World, July 92, p.87-89.

C. von Altrock, B. Krause and H.J. Zimmermann, Advanced fuzzy logic control of a model car in extreme situations, in Fuzzy Sets and Systems, n° 48, 1992,
p. 41-51.

Dorseuil A. et Pillot P., Le Temps Réel En Milieu Industriel, Concepts, Environnements Multitâches, Dunod, Paris, 1991.

Dubois D. et Prade H., Théorie des Possibilités, Applications à la représentation des connaissances en informatique, Masson, 2ème Ed., 1988

Dubois D., Prade H., Les logiques du flou et du très possible, dans la Recherche n° 237, Novembre 1991, volume 22, p. 1308-1315.

H.-J. Zimmermann , Fuzzy Set Theory-And Its Applications, Kluwer Academic Publisher, Boston/Dordrecht/London, 1991.

Hitoshi Aoki and Kenji Sasaki, Group Supervisory Control System Assisted By Artficial Intelligence, in Elevator World, Feb. 90, p. 70-80.

Kaufmann A., Introduction à la théorie des sous-ensembles flous, vol. 1, Masson, 1973.

Kosko Bart, Neural Networks And Fuzzy Systems, A Dynamical Systems Approach To Machine Intelligence, Prentice-Hall International Editions, 1992.

Reza Langari and Hamid R. Berenji, Fuzzy Logic In Control Engineering. *Handbook Of Intelligent Control, Neural, Fuzzy, And Adaptative Approaches*, edited by David A.White and Donald A. Sofge, VNR Computer Library, New York, 1992.

Schiper A., Programmation Concurrente, Presses Polytechniques Romandes, 1986.

Thomas H. Naylor, Joseph L. Balintfy, Donald S. Burdick, Kong Chu, Computer Simulation Techniques, John Wiley & Sons, Inc., New York, 1966.

Tschirhart D., Commande En Temps Réel, Conception Et Mise En Oeuvre d'un Exécutif Temps Réel Et Multitâche, Dunod, Paris, 1990.

Vio G, Fuzzy Logic in C, in Dr. Dobb's Journal, February 1993, p.40-49

William H. Press, Brian P. Flannery, Saul A. Teukolsky, William T. Vetterling, Numerical Recipes in C, The Art Of Computing, Cambridge University Press, 1988.