الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي والبحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

N° Ref :……

Centre Universitaire de Mila

Institut des sciences et de la technologie

Département de Mathématiques et Informatique

Une approche d'optimisation coopérative à base d'agents optimiseurs appliquée au problème de transport à la demande (TAD)

Mémoire préparé En vue de l'obtention du diplôme de Master en Informatique

Préparé par : BOUSMINA Zaid.
GUELLIL Hamza

Encadré par : Mme. M. Zekiouk

Filière : Informatique

Spécialité : STIC

Année universitaire :2012/2013

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Centre universitaire de Mila
Institut des Sciences et de la technologie
Département de Mathématique et Informatique
Filière informatique

MÉMOIRE

Pour obtenir le Diplôme de
Master en Informatique
Spécialité : STIC

Thème

Une approche d'optimisation coopérative a base d'agents optimiseur appliqué au problème de transport à la demande

Préparé par :

- Bousmina zaid.
- Guellile Hamza.

Encadré par :

- Mme.M.Zkikouk .

Devant le jury :

- Mme.F.Affri.
- Mme.Z.Daffas .

Soutenu le : … / … / 2013

Année universitaire 2012 / 2013.

Remerciements

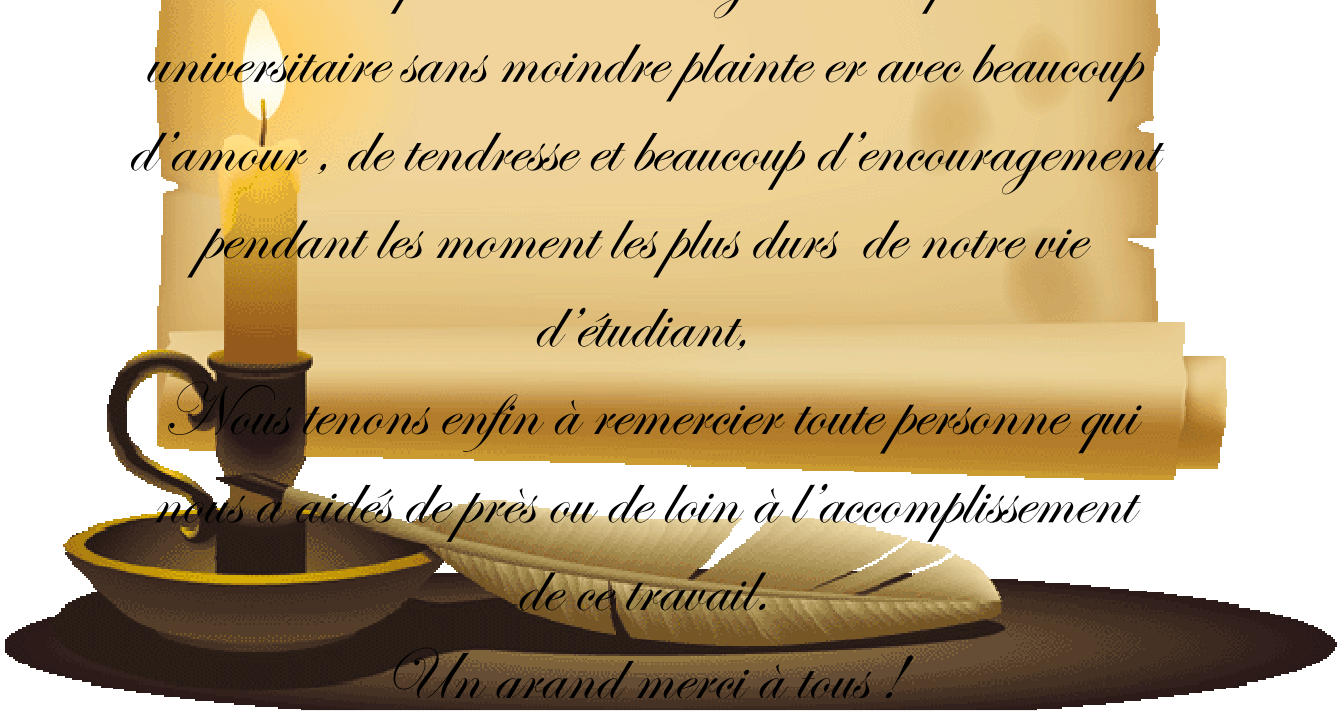
Nous tenons tous d'abord à remercier du fond notre cœur Allah le miséricordieux qui nous a donné la force, la volonté et la patience pour mener à terme notre mémoire et la très la respectueuse Mme Zekiouk pour nous avoir encadré et épaulé lors de la longue et fastidieuse élaboration de notre étude.

Nous remercions également les membres du jury pour avoir examiné et apprécier notre travail.

Sans oublier d'exprimer notre gratitude et nos plus sincères remerciements à nos familles qui nous ont soutenu ét épaulé tout au long de notre parcours universitaire sans moindre plainte er avec beaucoup d'amour , de tendresse et beaucoup d'encouragement pendant les moment les plus durs de notre vie d'étudiant,

Nous tenons enfin à remercier toute personne qui nous a aidés de près ou de loin à l'accomplissement de ce travail.

Un grand merci à tous !



Dédicace

Je dédie ce mémoire à mes très chers parents qui m'ont soutenu, encouragé avec amour et tendresse sans jamais faillir à leur devoir durant toutes mes années d'études et je les remercie de tout mon cœur pour ce que je suis aujourd'hui.

A mes frère **Amine, Kais et Islame que je chérie et respecte**

A ma petite sœur **Alia** que Dieu la préserve de tout mal et remplit sa vie de bonhoneur et de santéA mes tantes, mes oncles, mes cousins, mes cousines et au reste de ma famille et tous mes voisins du quartier.

A mon binôme **Hamza** qui m'a été d'une aide précieuse durant ce travail et le remercie pour sa collaboration sincère et eficace tout au long de nos recherches.

A tous mes amis : **Ibrahim, Raouf, Fares, Nouar, Yasser, Djaafar, Halim, Toufik, Nour dine,** et **Abde Nour** et les collègue de master et sans oublie les collègues filles de notre promotion avec lesquelles j'ai passé des moments inoubliables durant ces deux dernières années qui resteront gravés dans ma mémoire à jamais .

Zaid Bousmina

Dédicace

Je dédie ce mémoire à mes très chers parents qui m'ont accordé tant d'attention et de courage durant mes années d'études. Je les remercie de tout mon cœur pour ce que je suis aujourd'hui.

A mon cher frère Samir, qui est un exemple pour moi, qui m'a apprit ce qu'est la patience. Je lui souhaite une vie de couple épanouie et beaucoup de bonheur.

A mes sœurs qui m'ont apporté leur soutien et qui ont su me remonter le moral lors des moments difficiles.

A ma petite nièce **Racha** *que j'aime beaucoup et à qui je souhaite une longue vie pleine de bonheur et de réussite.*

A mes tantes, mes oncles, mes cousins, mes cousines et au reste de ma famille.

A mon binôme **Zaid** *qui m'a épaulé durant ce travail et qui m'a été d'une parfaite collaboration tout au long de nos recherches.*

A tous mes amies : **Toufik, Halim, Achraf, Nouri, Fekraoui, Moussa, Yazid, Hafedh, Dhiaa, Yasser, Djaafar** *; et collègues de la promotion, avec qui j'ai passé des moments inoubliables durant ces deux dernières années.*

A tous ceux qui m'aiment et que j'aime et qui font partie de ma vie.

A ma fiancée **Yamina** *avec qui j'espère vivre ma toute ma vie et qui donné du courage durant mes études.*

Hamza Guellile

Sommaire

Chapitre 2 : Les systèmes multi-agent et l'optimisation dans le domaine de TAD

Chapitre 3 : Une architecture multi-agent pour la résolution du problème De TAD

Chapitre 4 : Implémentation et Tests

Liste Des Figures

Liste des Algorithmes

Liste des Tableaux

Introduction général

La mobilité tient une place importante dans nos vies quotidiennes. Il s'agit de modes de transporte individuel (vélo, marche à pied,…), de modes de transport en communs (bus, train,…) ou de modes de transport flexibles qui s'adaptent à des zones moins densément peuplées. Dans le cadre de ce mémoire nous nous intéressons au transport à la demande (TAD). Ce mode de transport consiste à déterminer les tournées et horaires des véhicules qui effectuent le transport d'usagers à leurs demandes. Ce problème se pose, par exemple, dans le transport adapté de personnes handicapées, âgées et des patients. Vue l'importance de ce mode de transport dans l'amélioration de la vie des citoyens, les différents opérateurs de transport et les spécialistes de plusieurs domaines cherchent à combiner les efforts afin d'améliorer la qualité de services fournis par ce type de transport.

Dans la littérature le problème de TAD est définit comme étant un problème d'optimisation combinatoire NP-difficile. L'optimisation combinatoire est une discipline combinant diverses techniques des mathématiques discrètes et de l'informatique afin de trouver la meilleure solution dans l'ensemble des solutions réalisables.

Les efforts des chercheurs s'orientent aujourd'hui vers la combinaison des (méta) heuristiques avec d'autres approches de modélisation comme le paradigme multi-agent et les services web et ils essayent même de profiter des technologies de communication existantes. Généralement, la combinaison de ces domaines différents améliore beaucoup les types des solutions obtenues et même les types des problèmes traités.

Notre but dans ce mémoire est de concevoir et développer un système intelligent permettant de gérer les plans de circulation des véhicules de façon optimale pour assurer la satisfaction des clients et la non infraction des contraintes liées à un opérateur de transport (le nombre de place, l'autonomie de véhicule, la fenêtre de temps etc.). Notre deuxième objectif concerne la gestion des cas de panne en se basant sur un protocole de négociation efficace.

Pour la réalisation de nos objectifs nous proposons de modéliser notre système par une approche multi-agents afin de pouvoir utiliser un des protocoles de négociation fournis par ce paradigme. Concernant la génération des tournées nous proposons l'utilisation de la recherche tabou.

Ce mémoire est organisé en 4 chapitres comme suit :

- **Chapitre 1 : Le transport à la demande.**

  Dans ce chapitre nous avons présenté les définitions d'un problème de transport à la demande et les différentes typologies de ce dernier ainsi qu'une brève description des caractéristiques majeures liées a ce genre de problème.

- **Chapitre 2 : Les Systèmes Multi-Agents et l'Optimisation dans le Domaine de TAD.**

  Ce chapitre est consacré à un état de l'art général des systèmes multi agents et les méthodes d'optimisation utilisées dans la résolution des problèmes de TAD

- **Chapitre 3 : Une Architecture multi-agent pour la Résolution du problème de TAD.**

  Dans ce chapitre nous décrivons l'architecture générale proposée ainsi que le protocole de négociation et les algorithmes d'optimisation utilisés.

- **Chapitre 4: Implémentation et Tests.**

  Dans ce chapitre nous donnons une description de différentes plateformes utilisées dans la construction des systèmes multi-agent et nous présentons l'application développée et les résultats obtenus.

Chapitre I

Le Transport à la Demande

1. Introduction.

Le secteur de transport à la demande a une application très forte dans le contexte mondial actuel car les individus cherchent toujours des services de transport plus souple et plus proche aux besoins quotidiens.

Le problème de transport à la demande (TAD) fait partie de ce contexte, (il consiste à prendre en charge le transport des personnes à partir d'un lieu de départ vers un lieu d'arrivé. Il est caractérisé par un ensemble de demandes de transport et d'un nombre de véhicules disponible). Ce problème se pose, principalement, dans le transport adapté de personnes handicapées, âgées et des patients[1]. Actuellement, les recherches dans ce domaine sont en perpétuelle évolution vue l'importance de ce mode de transport dans l'amélioration de la vie des citoyens.

L'objectif de ce chapitre est de présenter une vue générale du problème de transport à la demande et ses caractéristiques ainsi que ses différentes typologies.

2. le transport à la demande.

2.1. Définitions.

Les définitions officielles de TAD varient d'un continent à un autre. Néanmoins, au-delà de ces définitions officielles s'imposent également les définitions usuelles du TAD, qui font écho à l'usage réservé au TAD en question et sont donc une question de points de vue différents, quand bien même le principe demeure relativement inchangé. Néanmoins, quelle que soit l'acception retenue, toutes s'accordent sur le point que le TAD est une forme de transport public. Dénommé « *Demand Responsive Transport* » (DRT) chez les Anglo-Saxons et dans la littérature scientifique associée. [1]

Le problème de transport à la demande (**TAD**) ou Demand Responsive Transport (**DRT**) est un problème générique qui englobe a priori tous les services de transports dont tout ou partie ne s'effectue qu'à la demande expresse de ceux qui les utilisent.

Ce problème générique peut se présenter de la manière suivante : dans un espace donné (un réseau), on cherche à acheminer des demandes de clients à l'aide de moyens de transport pouvant combiner différents types de véhicules. Chaque demande du client est définie par un lieu d'origine, un lieu de destination, un nombre de passagers et des contraintes temporelles.[1]

Le problème de TAD a été étudié pendant plus de 30 années, surgisse dans beaucoup de contextes tels que la logistique, les services ambulatoires. Nous pouvons aussi rencontrer ce problème dans notre vie quotidienne comme par exemple le ramassage scolaire, le transport de personnel, transport des patients…etc.[18]

Le schéma ci-dessus montre la vue générale du problème de transport à la demande.

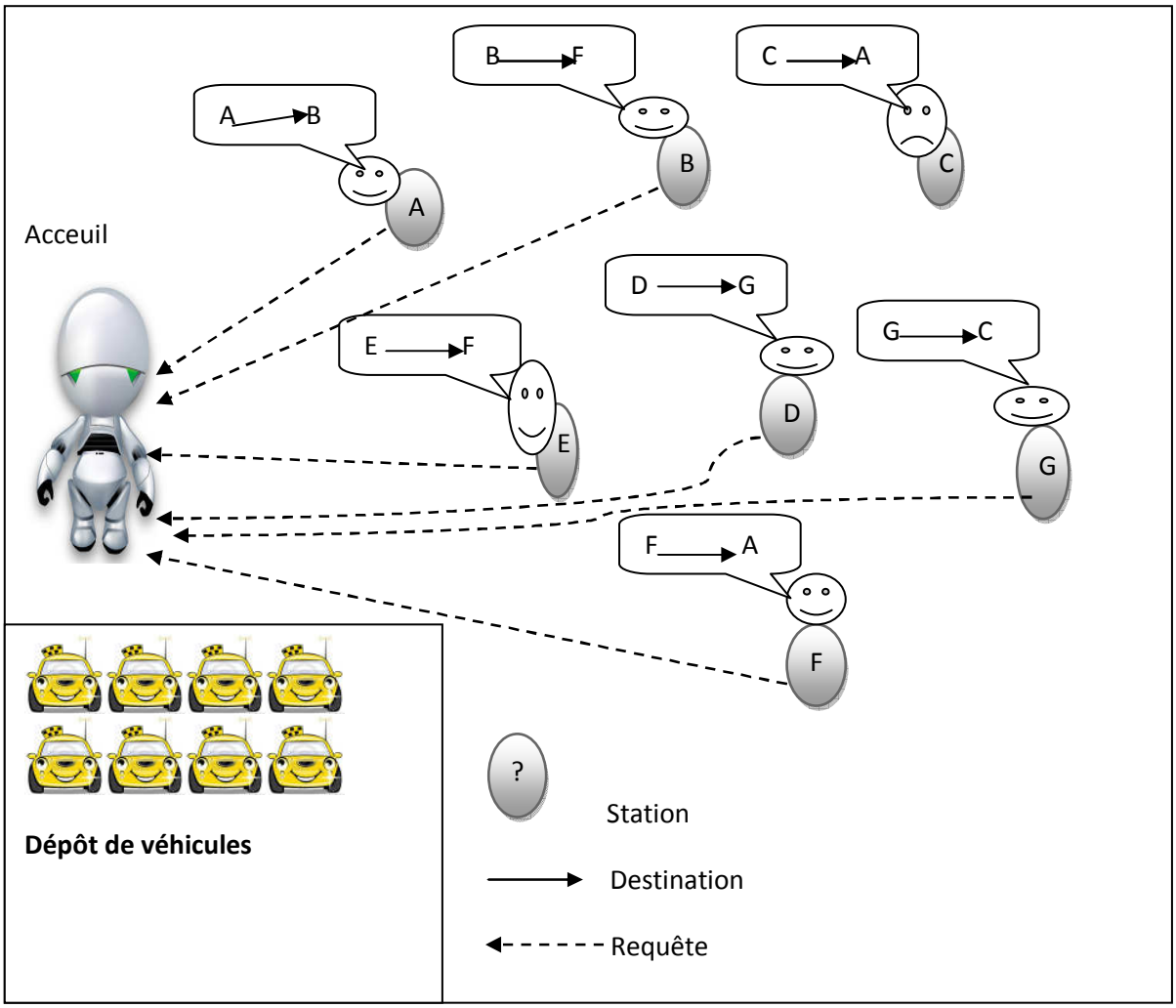


Figure 1.1 : Présentation schématique du problème du TAD.

2.2. Terminologie.

Dans ce paragraphe nous citons quelques termes utiles pour comprendre les principes de transport à la demande.[3]

- **Une requête** : est une demande de services pour une opération de transport, un ramassage et une livraison en général, formulée par des passagers.

- **Une tournée :** est le plan de route exact attribuable à un véhicule.une tournée contient l'ordre de la satisfaction des requêtes.

- **Une course :** est un fragment de tournée relatif à une requête, c'est-à-dire ce qui sépare le ramassage initial de la livraison finale.

- **Une station** : est un site géographique ou une adresse donnée par un client. la requête du client consiste à une station d'origine pour le chargement et une station de destination.

3. Le TAD et les problèmes de tournées de véhicules.

Dans ce qui suit, nous donnons une typologie des problèmes aboutissant au problème du TAD car ce dernier fait partie de la famille des problèmes de tournées de véhicules, qui ont des extensions du problème de voyageur de commerce [3]. Cette typologie nous permet de cerner le cadre de notre travail par rapport aux problèmes existants.

3.1. Problème du voyageur de commerce(TSP).

Le problème du voyageur de commerce (ou TSP: Traveling Salesman Problem) est le suivant : un représentant de commerce peut vendre sa marchandise dans un certain nombre de villes, il doit donc planifier sa tournée de manière à passer par toutes les villes en voyageant au total le moins possible. [3]

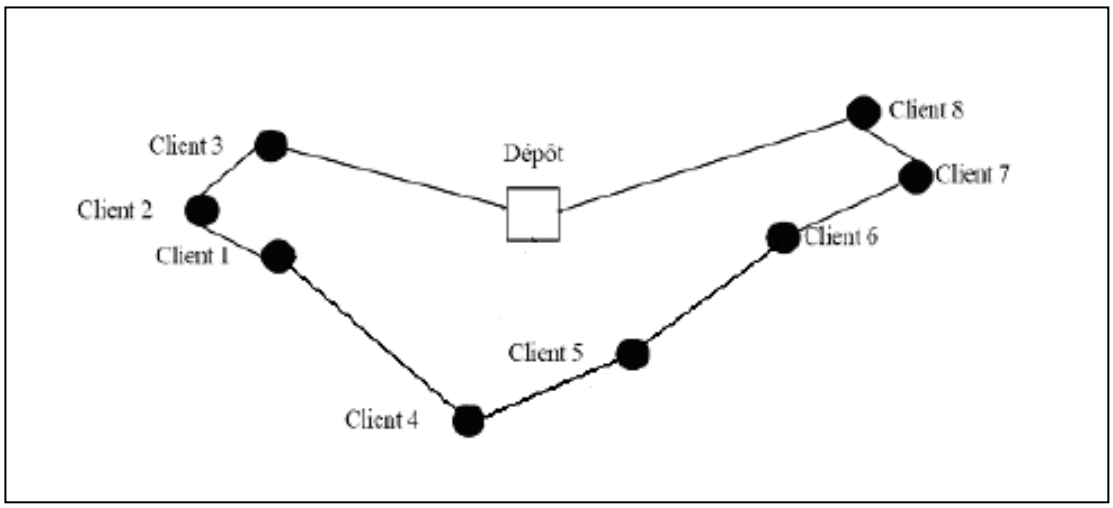


Figure 1.2 : Problème du TSP.

3.2. Problème des tournées de véhicules(VRP).

Le problème de tournées de véhicules (ou Vehicle Routing Problem = VRP) est une généralisation du TSP obtenue lorsque K véhicules de capacité Q sont disponibles au nœud-dépôt pour satisfaire les demandes des nœuds-clients. Chaque véhicule doit donc effectuer une tournée réalisable, c'est à dire quitter le dépôt, visité une fois des clients dont la somme des demandes ne dépasse pas la capacité Q, avant de retourner au dépôt. Chaque client doit être servi par un seul véhicule qui satisfait totalement sa demande.[3]

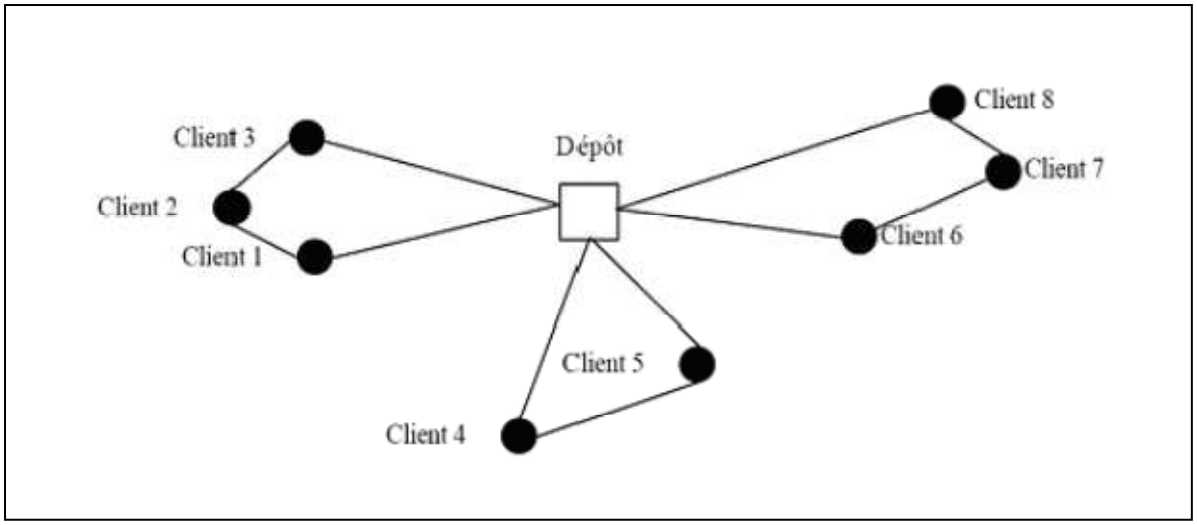


Figure1.3 : problème de VRP.

3.3. Problème de tournées de véhicules avec ramassage et livraison (PDP).

Ce problème consiste à ramasser les produits d'un ensemble d'expéditeur et ensuite de les transporter au fur et à mesure à un ensemble de clients[3].

3.4. Problème de tournées de véhicules avec fenêtres temporelles.

Un problème de tournées de véhicules avec fenêtres temporelles (VRPTW) est un VRP dans lequel un intervalle de temps est associé à chaque ville, et un temps de service pendant lequel le véhicule est obligé de stationner avant de pouvoir repartir. Un véhicule ne peut pas visiter un client en dehors de sa fenêtre temporelle, et ne peut le quitter avant d'y avoir stationné pendant le temps de service qui lui est associé.[18]

3.5. Le problème de tournée de véhicule dynamique (DVRP : Dynmic vehicle routing probleme).

Dans les problèmes de tournées de véhicule dynamiques, les données du problème sont reçues au fur et à mesure de l'exécution, et doivent être incorporées dans les solutions courantes. La dynamique observée concerne dans la majorité des systèmes de la littérature concerne les clients : ils apparaissent au fur et à mesure de la résolution. A tel point que le terme problème dynamique désigne essentiellement un problème où les clients ne sont pas tous connus avant le démarrage de la résolution. Les autres sources de dynamique comme le changement en cours de résolution des temps de parcours entre les nœuds du réseau sont désignés par d'autres termes[3].

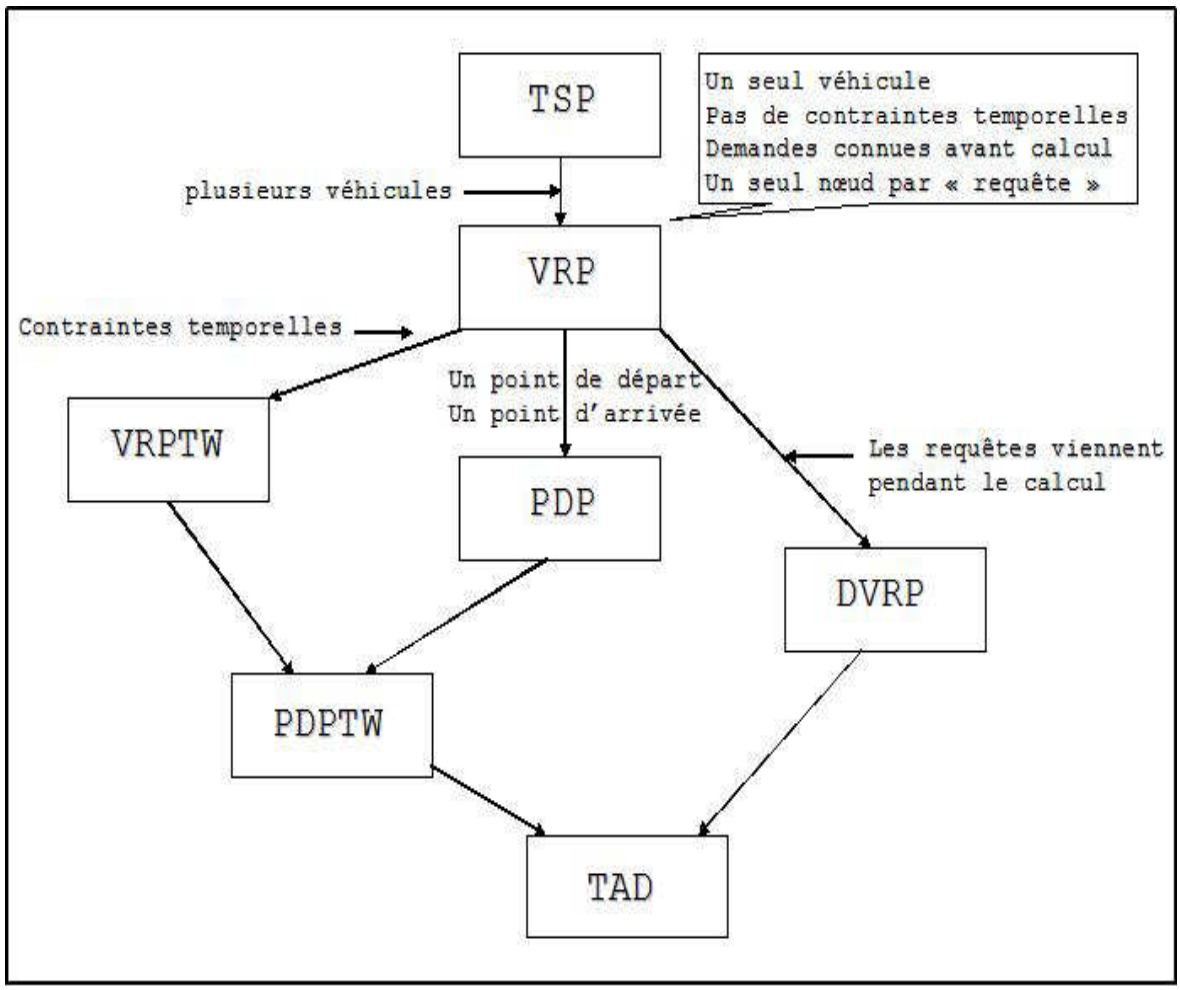


Figure 1.4 : Typologie des Problèmes de tournées de véhicules.[3]

4. Typologie des problèmes de TAD.

On peut distinguer divers types de TAD[19], en fonction :

- De leur caractère public (les taxis par exemple) ou privé (réservés aux membres d'une collectivité particulière : par exemple les chauffeurs d'une entreprise ou d'une administration ; ou bien encore certains modes de covoiturage).

- De leur caractère individuel (le taxi, la petite et la grande remise) ou collectif (les navettes qui desservent les aéroports et qui vont chercher leurs clients à leur domicile ou dans leurs hôtels).

- De la nature de la demande qui est expressément prise en compte. La demande peut porter sur l'horaire et les lieux : le taxi ou la navette viennent chercher le client là où il est et à l'heure à laquelle il souhaite se déplacer. le premier le dépose à l'endroit de son choix :

c'est ce que nous appellerons du TAD en porte-à-porte, la seconde le dépose à un point déterminé à l'avance : nous appellerons cela du TAD semi-polarisé. La prise en compte de la demande peut être encore plus partielle : le transporteur passe prendre le client à une heure négociée et à un lieu fixe (un arrêt prédéterminé). Mais font également partie du TAD les lignes de bus à tracé fixe dont les véhicules ne sont activés qu'en fonction des demandes effectives des clients : on appelle cela des "lignes de bus virtuelles".

- Une autre distinction peut être introduite en s'inspirant de la classification de Röpke et Pisinger[1] :
  - ✓ le *many-to-one* : les passagers sont collectés chez eux mais déposés au même endroit.
  - ✓ le *many-to-few* : passagers transportés en un nombre limité d'endroits.
  - ✓ le *many-to-many* : destinations multiples dans une même zone, autrement dit l'one-to-one ou le porte-à-porte.

- Savelsberg[2] a proposé une autre classification en se basant sur les deux critères suivants:
  - ✓ *Le nombre des véhicules.*
  - ✓ *La disponibilité de l'information sur les demandes.*

  Selon le premier critère il existe deux types :
  - ✓ *TAD à un seul véhicule* **:** dans ce genre de problèmes il existe un seul véhicule qui satisfait toute les demandes.
  - ✓ *TAD à plusieurs véhicules* **:** c'est le cas où les demandes sont partagées entre plusieurs véhicules, et chaque demande sera satisfaite par un seul véhicule.

  Selon la disponibilité de l'information sur les demandes, il existe aussi deux types :

- *TAD statique***:** c'est le cas où toutes les demandes sont connues avant que les tournées soient construites.
- *TAD dynamique***:** dans ce cas, certaines demandes sont connues avant que les tournées soient construites, et les autres demandes deviennent disponibles en temps réel pendant l'exécution de ces tournées.

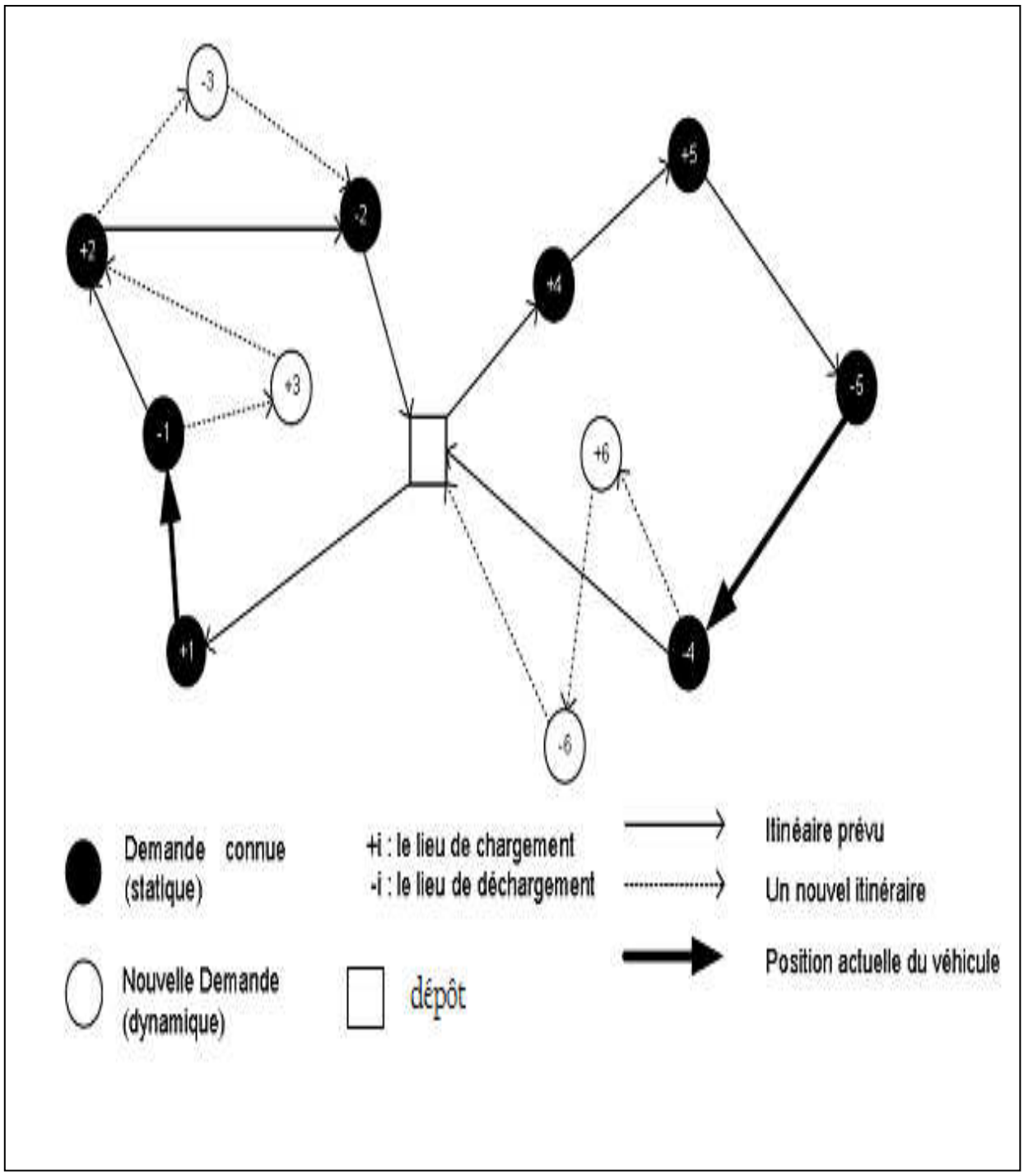


Figure.1.5 : Exemple d'un TAD dynamique.

5. Objectifs et enjeux de TAD.

Le TAD propose en complément des services de transport classique, les caractéristiques suivantes[1][3] :

- **La flexibilité** : peut s'exprime à différents niveaux qui peuvent être :

  - ✓ **La desserte spatial :** un client peut réserver une véhicule de n'importe qu'elle place sans avoir un problème, de l'autre coté le véhicule peut choisit le chemin vers le client. Ce chemin peut être choisir manuellement ou optimiser a travers un système informatique (résolution des tournées).

  - ✓ **Les horaires :** tout comme les trajets ne sont pas fixes, les horaires de prise en charge et de desserte sont, au mieux, optimisés selon les souhaits des usagers.

  - ✓ **La tarification :** celle-ci peut être fixés ou définie selon un mode de calcule (critère de distance).

- **Economie :** Une des raisons du succès des TAD auprès des collectivités réside dans son moindre cout de fonctionnement autrement dit, un TAD ne roule jamais à vide, contrairement à certains véhicules sur des lignes fixes. Les TAD peuvent être sous-traités auprès de compagnies privées ou des artisans taxis pour réduire les charges liées aux frais de fonctionnement et ne payer que les déplacements effectués.

6. Pratiques réelles des Systèmes de transport à la demande.

Quelques systèmes apparentés aux systèmes de TAD ont été proposés de par le monde. Dans ce qui suit nous présentons les principaux[3].

6.1. Navettes d'entreprises

Des entreprises ont mis en place des bus de ramassage pour leurs salariés. Plusieurs entreprises, situées sur la même zone d'activité et mal desservies par les transports en commun, se sont aussi regroupées pour proposer un tel service à leurs salariés respectifs.

Par exemple dans son usine de voitures de Regensburg, BMW a rencontré de problèmes qu'on retrouve à des niveaux territoriaux plus élevés : elle devait décider s'ils allaient construire de nouvelles aires de stationnement et élargir les voies d'accès pour résoudre les

problèmes de congestion, ou alors trouver des solutions alternatives. Ils ont opté pour l'introduction d'un système de bus de société pour leurs employés.

6.2. Le Car Sharing

Le Car Sharing propose la mise en commun d'une flotte de véhicules. Cette démarche est particulièrement développée en Suisse et en Allemagne, elle remplace efficacement la voiture individuelle, notamment la seconde voiture, souvent sous-utilisée par leurs propriétaires. Les adhérents peuvent faire leurs courses ou organiser une sortie en payant le véhicule à l'usage. L'adhérent évalue ainsi le cout de ses déplacements et peut choisir le mode de transport le mieux adapté à ses besoins. Cette nouvelle solution de mobilité en milieu urbain combine les avantages des transports publics et du véhicule privé.

6.3. Transport des patients.

Ce type de transport se trouve dans presque toutes les villes du monde. Il s'occupe de transport des patients vers les hôpitaux et les cliniques individuellement ou par groupe comme c'est le cas des patients en hémodialyse[17].

7. Modélisation Mathématique d'un problème de TAD.

La littérature définit le problème de transport à la demande comme étant un problème d'optimisation combinatoire NP-difficile.

L'optimisation combinatoire est une discipline combinant diverses techniques des mathématiques discrètes et de l'informatique afin de trouver la meilleure solution dans l'ensemble des solutions réalisables, noté par X. En général, cet ensemble est fini mais compte un très grand nombre d'éléments qui doivent tous satisfaire un ensemble C de contraintes. Si $\boldsymbol{f}$ est la fonction objectif qui permet d'évaluer chaque solution réalisable.

Alors, un problème d'optimisation combinatoire vise à déterminer une solution $x^* \in X$ qui minimise f. Donc, La résolution d'un problème de transport à la demande revient à la résolution d'un problème d'optimisation.

Généralement, la résolution d'un problème d'optimisation nécessite une description mathématique du problème étudié. Concernant le problème de TAD, plusieurs descriptions

formelles sont présentées pour la description de ce dernier. Parmi ces descriptions nous donnons celle proposée dans [20] comme suit :

N = L'ensemble des stations d'origine, destination et dépôt.

N'= L'ensemble des stations d'origine, destination.

N^+ = L'ensemble des stations d'origine.

N^- = L'ensemble des stations de destination.

K = Le nombre des véhicules.

d_{ij} = la distance euclidienne entre le nœud *i* et le nœud *j*, si $d_{ij} = \infty$ alors le chemin entre *i* et *j* n'existe pas (impasse, rue piétonne).

t_{ij}^k = le temps mis par le véhicule *k* pour aller du nœud *i* au nœud *j*.

q_i = la quantité traitée au nœud *i*, si q_i >0 alors le nœud *i* est un nœud d'origine,

si q_i <0 le nœud est un nœud de destination.

Q_k = la capacité du véhicule *k*.

i =0..*N* : indice des stations prédécesseurs.

j =0..*N* : indice des stations successeurs.

k =0..*K* : indice des véhicules.

Les variables de décision

$$x_{ij}^k = \begin{cases} 1 & \text{Si le véhicule k voyage du nœud i au nœud j} \\ 0 & \text{Sinon} \end{cases}$$

D_i : Le temps de départ du nœud *i*.

y_i^k : La quantité présente dans le véhicule *k* visitant le nœud *i*.

La fonction à optimiser

Les fonctions à optimiser diffèrent d'un problème à un autre, le critère le plus utilisé dans ce genre de problème est la minimisation du chemin parcourue par un ensemble minimum de véhicules.

Minimiser $\left(\alpha_1 K + \alpha_2 \sum_{k\in K} \sum_{i\in N} \sum_{j\in N} d_{ij} x_{ij}^k \right)$

Les contraintes

$$\sum_{i=1}^{n} \sum_{k=1}^{m} x_{ij}^k = 1 \quad j = 2,\ldots,n \qquad (1)$$

$$\sum_{j=1}^{n} \sum_{k=1}^{m} x_{ij}^k = 1 \quad i = 2,\ldots,n \qquad (2)$$

$$\sum_{i\in N} x_{i0}^k = 1 \qquad (3)$$

$$\sum_{j\in N} x_{0j}^k = 1 \quad \forall k \in K \qquad (4)$$

$$\sum_{i\in N} x_{iu}^k - \sum_{j\in N} x_{uj}^k = 0 \quad \forall k \in K; \forall u \in N \qquad (5)$$

$$x_{ij}^k = 1 \Rightarrow y_i^k = y_i^k + q_i \qquad (6)$$

$$y_0^k = 0 \quad \forall k \in K; \qquad (7)$$

$$Q_k \geq y_i^k \geq 0 \quad \forall i \in N; \forall k \in K \qquad (8)$$

$$x_{ij}^k = 1 \Rightarrow D_i + t_{ij}^k \leq D_j \quad \forall i, j \in N; \forall k \in K \qquad (9)$$

$$D_w \leq D_v \quad \forall i \in N; w = N_i^+, v = N_i^- \qquad (10)$$

$$D_0 = 0 \qquad (11)$$

1. Les équations (1) et (2) assurent que chaque sommet ne soit servi qu'une seule fois par un et un seul véhicule.
2. Les équations (3) et (4) assurent le non dépassement de la disponibilité d'un véhicule. Un véhicule ne sort de dépôt et n'y revient qu'une seule fois.

3. L'équation (5) assure la continuité d'une tournée par un véhicule : le sommet visité doit impérativement être quitté.

4. Les équations (6), (7), (8) et (9) assurent le non dépassement de la capacité de transport d'un véhicule.

5. Les équations (10) et (11) assurent le respect de précédences.

8. Exemple d'un problème d'optimisation.

La figure suivante représente un exemple simple d'optimisation. Les requêtes (flèches), les temps de trajet (pondération des flèches) estimés de l'origine (cercle surmonté de personnages) vers la destination. Les cercles accotés à un véhicule représentent les dépôts des véhicules.

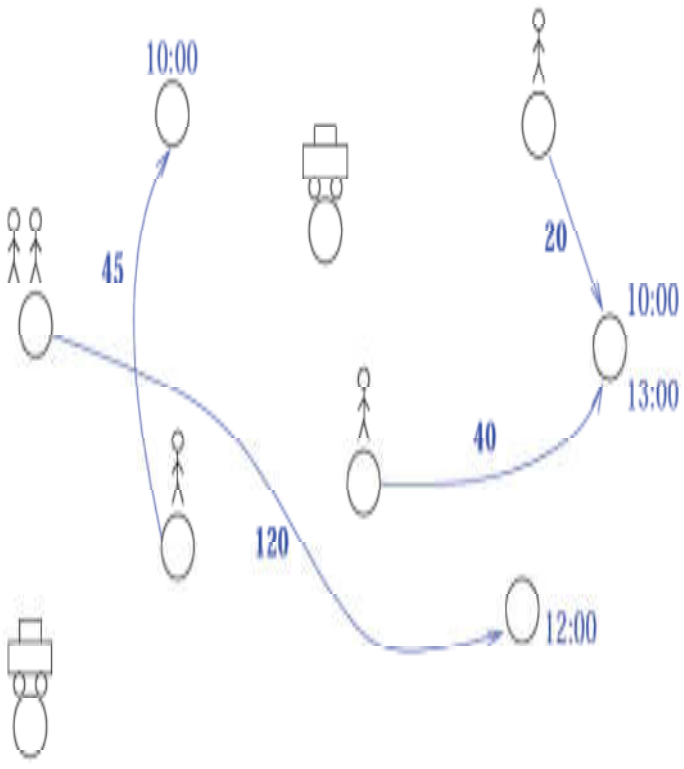


Figure 1.6. : Représentation des demandes.

La figure 1.2 suivante représente des tournées de véhicules répondant aux demandes précédentes. L'itinéraire de chaque véhicule est représenté en pointillés. L'heure de passage est indiquée pour chaque lieu (ramassage ou livraison).

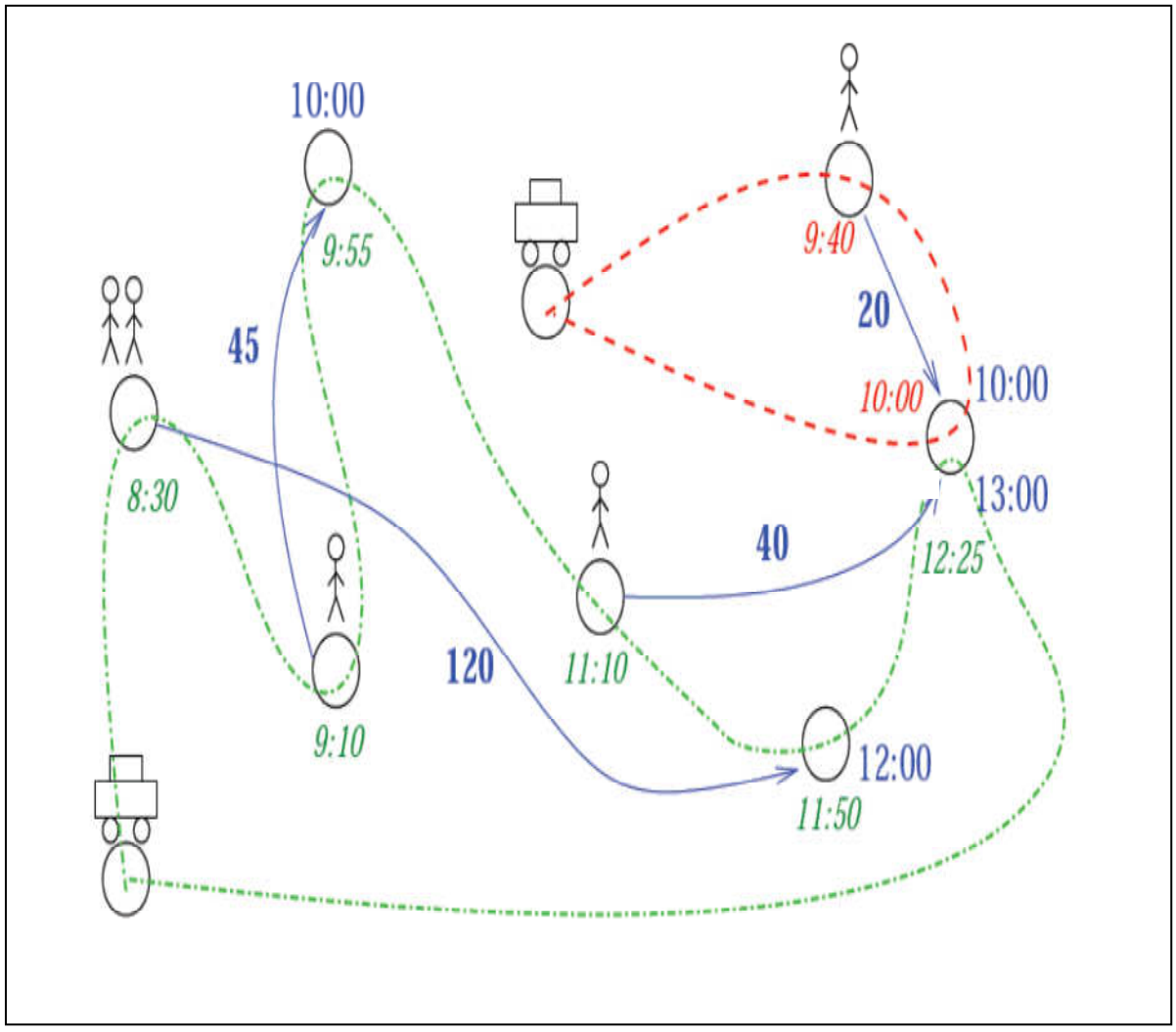


Figure. 1.7: Représentation des Tournées.

9. Choix des critères d'optimisation.

Le TAD est traité généralement de façon purement « multicritère » par une somme pondérée à minimiser (maximiser) qui inclut l'évaluation de chaque critère ou l'agrégation d'un même critère pour un ensemble de passagers.

Si nous considérons par exemple un TAD théorique pour lequel chaque client indique la date de chargement souhaitée, la date de déchargement souhaitée, le nombre de passagers transportés et une limitation de sa durée de transport. La société de transport dispose d'un nombre de véhicules disponibles au dépôt caractérisés par : une limitation de durée de service, un type et une capacité. Nous sélectionnons des critères que nous pensons représentatifs des problèmes posés[1][3] :

- Le premier concerne la personne. Les critères sélectionnés sont le temps de trajet, le temps d'attente pour commencer le service de chargement et déchargement, l'écart entre la date de chargement effective et la date de chargement déchargement souhaitée, et le temps pour répondre à la demande du client.

- Le second concerne les véhicules (les chauffeurs) ou la société de transport. Les critères sélectionnés sont la distance totale parcourue, le nombre de véhicules utilisés et le temps d'attente.

- Un troisième critère commun à l'utilisateur et à l'opérateur, concerne la sécurité, c'est-à-dire la capacité du système à pallier les erreurs, de communication, de rendez-vous, etc.

10. Principaux manques de TAD.

Il existe plusieurs freins majeurs à l'utilisation massive de TAD[2][3]:

- Le premier réside dans la notion d'optimisation. En effet, les routes empruntées par les véhicules pour prendre en charge les clients nécessitent des optimisations informatiques préalables. Plus la demande est forte et les temps de calcul des tournées sont élevés.

- Le deuxième frein réside dans les capacités de desserte. Les TAD sont souvent hélas pensés comme des substituts de lignes fixes qui suivent des tracés préétablis n'offrant pas l'ubiquité souhaitée par l'usager quand bien même celle-ci est à la base même d'un TAD. Notons que cette insuffisance venait aussi de la difficulté à optimiser la desserte quand le système subissait une forte montée en charge. Toutefois, les progrès réalisés permettent d'envisager une forte évolution des services, alliant qualité et efficacité économique.

- le principal point noir de la difficulté de développement du TAD reste le simple fait qu'ils sont déployés sur des territoires possédant généralement une offre de TC préalable.

11. Conclusion.

Dans ce chapitre, nous avons présenté le TAD comme variante du problème de tournée de véhicules. Après la définition du problème, nous avons présenté une classification de ce dernier. Ensuit, nous avons énuméré quelques avantages et points faibles qui frein la mise en place de ce mode de transport. Le chapitre est clôturé par une description formelle du problème.

Chapitre II

Les Systèmes Multi-Agents et l'Optimisation dans le Domaine de TAD

1. Introduction.

La simulation est utilisée dans la résolution de plusieurs problèmes dans le domaines de TAD surtout quand la réalisation de l'ensemble des expériences concrètes nécessaires à la validation ou l'infirmation d'une théorie prendrait un temps trop grand ou demanderait des moyens dont on ne disposerait pas. Néanmoins, la mise en œuvre d'une bonne simulation pour ce mode de transport se base généralement sur des méthodologies de modélisation et des approches d'optimisation efficaces. Parmi les méthodologies les plus utilisés dans le domaine de TAD, on trouve le paradigme des systèmes multi-agents (SMA).

Dans ce qui suit, nous présentons premièrement dans quel but nous avons opté pour une modélisation multi_agents, après nous citons les différents concepts concernant les SMA, ainsi que les techniques d'optimisation utilisées dans la résolution du problème de TAD.

2. Le paradigme multi-agent et le TAD.

Avant de présenter les principes des SMA, nous citons dans ce paragraphe les deux points essentiels qui justifient le choix d'une modélisation multi_agents.

- **Un système multi-agents fournit une *description naturelle* des systèmes de TAD :** Dans de nombreux cas, un système multi-agents est le plus naturel pour décrire et simuler un système composé d'entités « comportementales », les SMA permettent au modèle de paraître plus proche de la réalité. La modélisation est plus facilement interprétable par un observateur humain, car la description par un SMA est plus naturelle que par de simples processus.
- **Un système multi-agents est flexible :** La flexibilité des SMA peut être observée sur de multiples dimensions. Par exemple, il est simple d'ajouter plus d'agents à une simulation multi-agents. Les SMA fournissent également un cadre naturel pour adapter la complexité des agents : on peut jouer sur les règles de comportement, le degré de rationalité, la capacité à apprendre et à évoluer, ou bien encore les règles d'interaction.

3. Approche Agent.

3.1 Définitions.

Nous citons deux définitions pour le terme agent.

- **Définition1 :**

Un agent est une entité autonome, réelle ou abstraite, qui est capable d'agir sur elle-même et sur son environnement, celui-ci peut dans un univers multi-agents, communiquer avec d'autres agents, et dont le comportement est la conséquence de ses observations, de ses connaissances et des interactions avec les autres agents.[14]

- **Définition2 :**

On appelle agent une entité physique ou virtuelle (**Une entité physique** est quelque chose qui agit dans le monde réel: un robot, un avion ou une voiture sont des exemples d'entité physiques. En revanche, un composant logiciel, un module informatique sont des **entités virtuelles**, car elles n'existent pas physiquement)[15].

- ✓ qui est capable d'agir dans un environnement,

- ✓ qui peut communiquer directement avec d'autres agents,
- ✓ qui est mue par un ensemble de tendances (sous la forme d'objectifs individuels ou d'une fonction de satisfaction, voire de survie, qu'elle cherche à optimiser),
- ✓ qui possède des ressources propres,
- ✓ qui est capable de percevoir (mais de manière limitée) son environnement,
- ✓ qui ne dispose que d'une représentation partielle de cet environnement (et éventuellement aucune),
- ✓ qui possède des compétences et offre des services,
- ✓ qui peut éventuellement se reproduire,
- ✓ dont le comportement tend à satisfaire ses objectifs, en tenant compte des ressources et des compétences dont elle dispose, et en fonction de sa perception, de ses représentations et des communications qu'elle reçoit.[14]

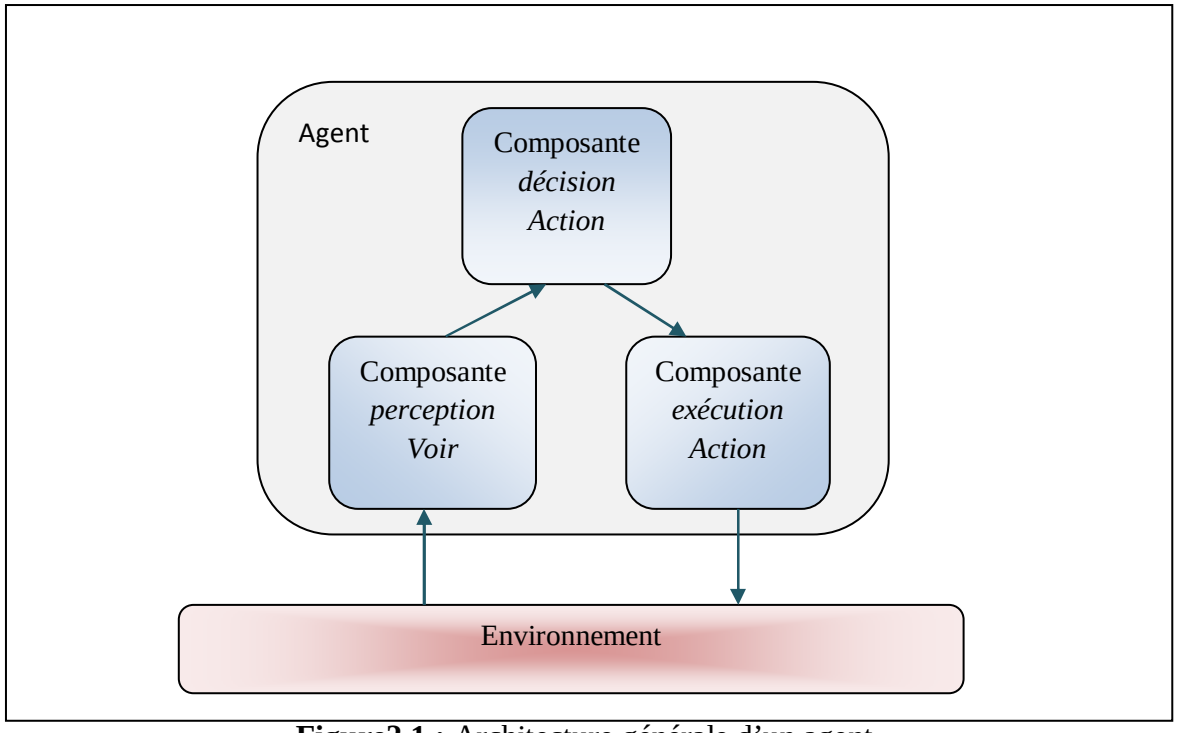


Figure2.1 : Architecture générale d'un agent.

3.2. Catégories des agents.

Il existe principalement trois types d'agent :

- **Agents réactifs :**

Ces agents sont basés sur l'idée que le monde est le meilleur modèle. Comme ils sont prévus pour manipuler des comportements simples, leur architecture est alors basée sur les schémas de routines simples en éliminant le raisonnement abstrait. La méthodologie est ascendante dans le sens ou les agents réagissent aux changements de l'environnement en utilisant des réponses simulées. Ceci est rendu possible grâce à l'exécution de simples routines qui correspondent à des stimulations spécifiques.

Contrairement aux agents cognitifs, les sociétés d'agents réactifs sont composées d'un nombre considérable d'agent de faible granularité. Un agent réactif ignore ses expériences passées parce qu'il ne possède pas de processus de raisonnement sophistiqués qui lui permettent de planifier d'apprendre. Ces agents sont très rapides dans la prise de décision puisqu'ils sont dotés d'un minimum de connaissances et de modèles de raisonnement **[21].**

Plusieurs chercheurs ont utilisé cette catégorie d'agents principalement dans le domaine de la robotique mobile, ou une réaction instantanée de l'agent est préférée à une réponse tardive de qualité. Cependant, peu de systèmes industriels se sont fondés sur cette catégorie d'agents parce qu'en fait, il n'existe aucune garantie sur la qualité des résultats des agents. En effet, les actions des agents sont régies par leur perception du monde et non par un processus de raisonnement élaboré comme c'est le cas des agents cognitifs

- **Agents cognitifs :**

Ces agent sont appelés aussi mentaux, rationnels ou délibératifs. Les agents cognitifs sont vus comme des systèmes intentionnels et utilisent les notions mentales appliquées aux être humains. En effet, l'expérience sur les aspects mentaux hérite de l'intelligence artificielle. L'agent possède un modèle explicite et symbolique du monde dans lequel il vit. Il se base sur des données particulières pour mettre à jour son modèle de l'environnement. C'est le composant chargé de la planification qui raisonne sur le modèle et décide quelles actions sont réalisées.

Généralement les systèmes à base d'agents cognitifs sont composés d'un petit nombre d'agents de grande granularité, i.e., ils possèdent un minimum de connaissance et des

comportements de haut niveau leur permettant de s'organiser, de se grouper, de coopérer, d'apprendre à coopérer en se servant de leurs expériences, et également de prévoir les résultats de leur comportements **[21].**

Une comparaison des deux catégories d'agents est synthétisée dans le tableau 1.1.

Systèmes d'agents cognitifs	**Systèmes d'agents réactifs**
Représentation explicite de l'environnement	Pas de représentation explicite
Possibilité de tenir compte de son passé	Pas de mémoire de son historique
Agents complexes	Fonctionnement stimulus/action
Petit nombre d'agent	Grand nombre d'agents

Tableau 2.1. Agents cognitifs vs agents réactifs **[21].**

- **Agents hybrides :**

Quelques travaux de recherche [14][15] ont essayé d'unifier les agents réactifs et ceux mentaux afin de surmonter leurs limites respectives, i.e., d'une part, la difficulté de mise en ouvre de l'approche cognitive dans les environnements complexes et à forte évolution, et d'autre part, le manque de modèles formels pour l'approche réactive. L'idée est la suivante : d'une façon simple, les agents hybrides réagissent à des stimulations simples en exécutant de simples routines. Cependant, le module cognitif contrôle celui qui est réactif quand ce dernier veut exécuter des actions à stimulation libre (comme le raisonnement) ou bien changer des buts à long terme.

Les fondateurs de cette catégorie d'agents augmentent son intérêt par les avantages quelle procure, notamment :

- un agent hybride possède une structure modulaire, ce qui est concrètement recommandé dans le développement de tout processus artificiel pour garantir l'évolution et la maintenance du système.
- Les capacités de traitement d'un agent peuvent être améliorées car ses différents composants peuvent fonctionner simultanément.
- Le comportement réactif de l'agent devient plus performant car l'organisation des connaissances d'un agent en partitions permet à chacun des composants réactifs ou cognitifs de manipuler partiellement ou totalement cette connaissance.

4. Systèmes multi agents.

Un Système Multi-agent (SMA) est un ensemble d'agents (agents logiciels ou humains) qui interagissent dans un environnement pour résoudre des problèmes qui dépassent les capacités ou les connaissances individuelles de chaque agent [22]. L'interaction peut aussi bien être une coopération qu'une compétition et l'environnement représente un espace commun d'interaction pour tous les agents. Cet environnement dépend de la nature de l'application considérée, il peut être par exemple le monde physique, les données d'un problème, une collection d'agents, etc [21]. Donc, un SMA comme est définit par J.Ferber consiste d'un *environnement*, d'un ensemble d'*objets passifs* et d'un ensemble d'*agents actifs*. Les objets sont reliés entre eux par un ensemble de *relations* définies et sont manipulés par les agents qui les perçoivent (création, modification et destruction).

Le paradigme des SMA permet en effet la compréhension, la conception et l'implantation de la façon la plus naturelle possible des différentes entités qui sont manipulées par des logiciels complexes, distribués et concurrents.

Les SMA font partie de l'IAD et sont actuellement utilisés dans une grande variété d'applications, telles que la production, le contrôle de trafic aérien, la surveillance du patient, le commerce électronique et les jeux[21].

L'IAD s'intéresse à des comportements intelligents qui sont le produit de l'activité coopérative de plusieurs agents. Ainsi, l'IAD et les SMA étudient la manière de répartir un problème sur un certain nombre d'entités coopératives. Elles s'intéressent à la manière de coordonner le comportement intelligent d'un ensemble d'entité selon des lois sociales. Ces entités ou agents sont autonomes et interagissent dans un environnement pour la résolution de problèmes [23].

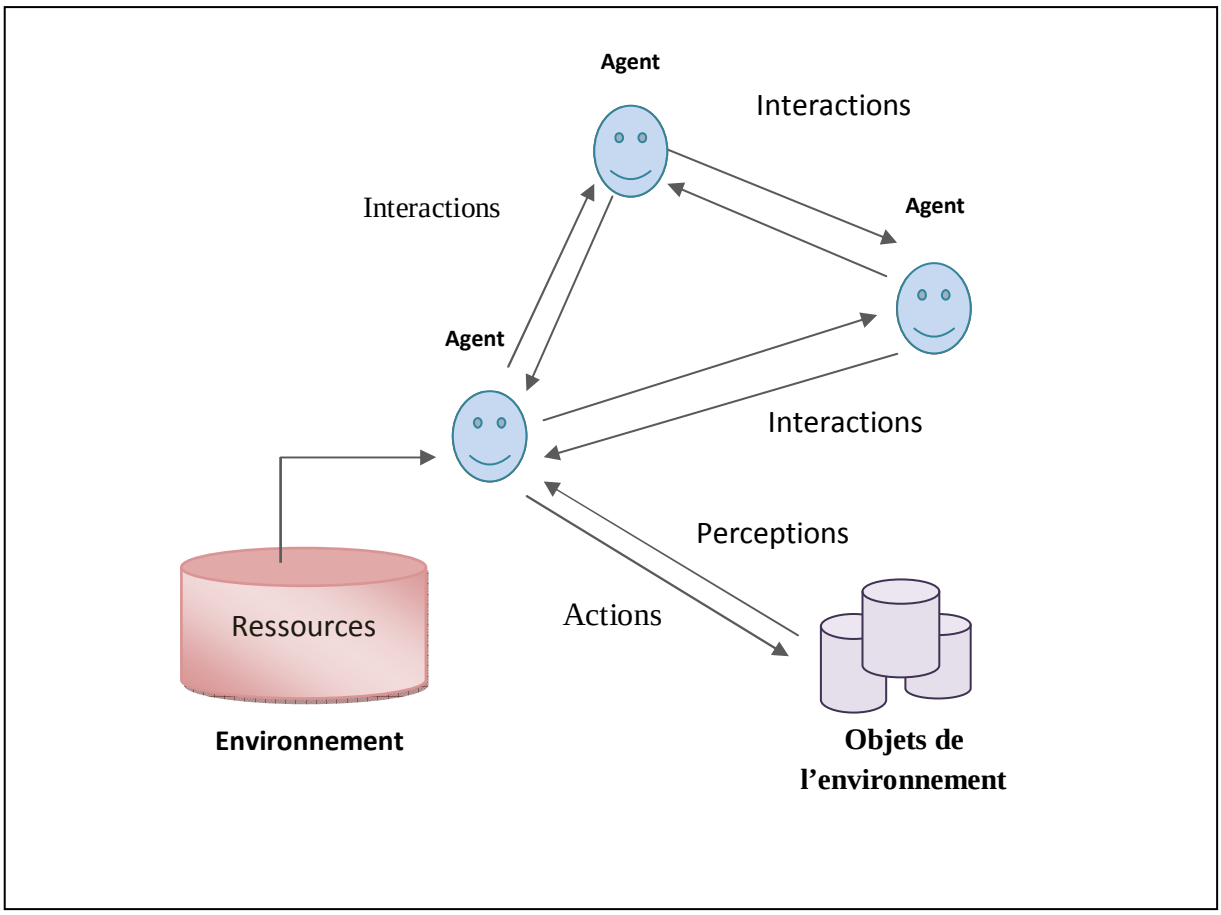


Figure 2.2 : Architecture d'un système multi-agent.

5. Caractéristiques des SMA.

Les systèmes multi agents possèdent plusieurs caractéristiques, parmi ces dernières nous citons les suivantes[21] :

- Autonomie des agents **:** un agent est capable de prendre des initiatives de manière autonome et d'exercer un contrôle sur ses actions.
- Distribution **:** différents types de distributions peuvent être prises en compte :
  - ✓ Distribution physique (traitements, données) : ce type prendre en compte la nature distribué des problèmes rencontrés. Ainsi bénéficier de l'efficacité de traitement distribué.
  - ✓ Distribution de la compétence des agents : faire mieux à plusieurs agents qu'un agent (la peut importance de perdre un agent dans un ensemble d'agent a un nombre élevé).
- il n'y a aucun contrôle global du système multi-agent.

- chaque agent a des informations ou des capacités de résolution de problèmes limitées, ainsi chaque agent a un point de vue partiel.

6. Principes d'interaction entre agents.

Le fonctionnement global d'un SMA est géré généralement par plusieurs types d'interactions. Parmi ces interactions nous citons les trois suivantes :

6.1. Coopération.

La coopération est une caractéristique très importante dans les SMA. Une résolution distribuée d'un problème est le résultat de l'interaction coopérative entre les différents agents. Un agent doit pouvoir mettre à jour le modèle du monde environnant, intégrer les informations émanant d'autre agents, interrompre pour aider d'autres agents et déléguer une tâche qu'il ne sait pas résoudre à un agent dont il connaît les compétences. Ces caractéristiques constituent les qualités essentielles d'un agent coopératif. [21].

6.2. Communication.

La communication est la base de la résolution coopérative des problèmes. Elle a pour objectif de synchroniser les actions des agents et résoudre les conflits de ressources et de buts par la négociation. Dans les SMA, les agents ne disposent d'aucune mémoire commune. La communication entre les agents repose explicitement sur les mécanismes d'envoie de messages, de réception et de synchronisation [24].

Les agents doivent disposer d'un langage de communication pour l'échange d'information afin de pouvoir coopérer pour la résolution d'un problème. Ce langage appelé mécanisme de communication inter-processus est un ensemble de primitives connues par chaque agent et dont l'ensemble constitue un protocole de communication. Dans la littérature, deux modes de communication sont connus :

- La communication par envoie de messages dans laquelle les agents envoient leurs messages directement et explicitement au destinataire. La seule contrainte est la connaissance de l'agent destinataire. Les systèmes fondés sur ce mode relèvent d'une distribution totale à la fois de la connaissance, des résultats et des méthodes utilisées pour la résolution du problème. [21]

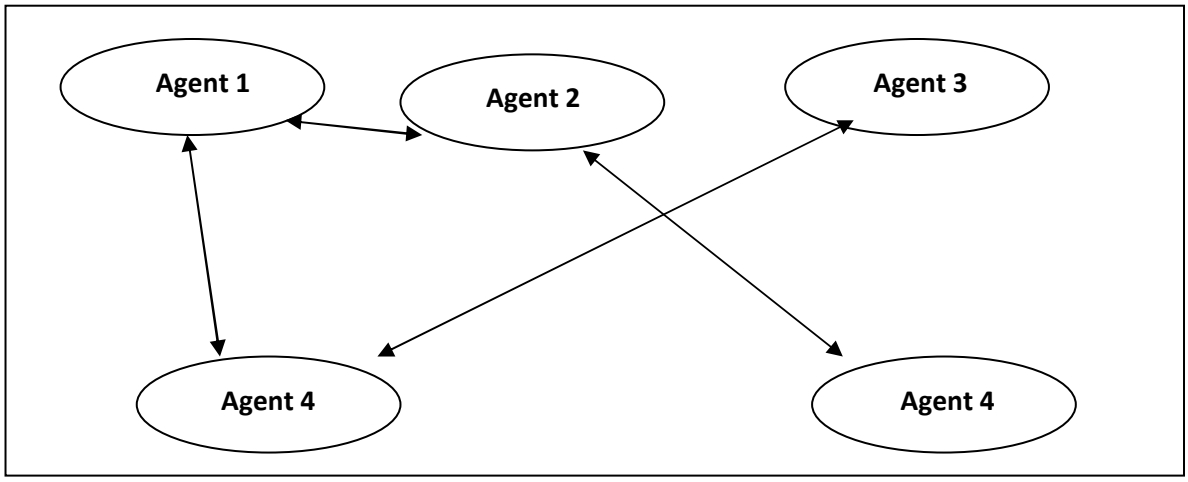


Figure2.3 : Communication par envoie de message.

- La communication par partage d'information où les agents ne sont pas en liaison directe mais communiquent via une structure de données partagée dans laquelle se trouvent les connaissances relatives à la résolution qui évolue durant le processus d'exécution. [21]

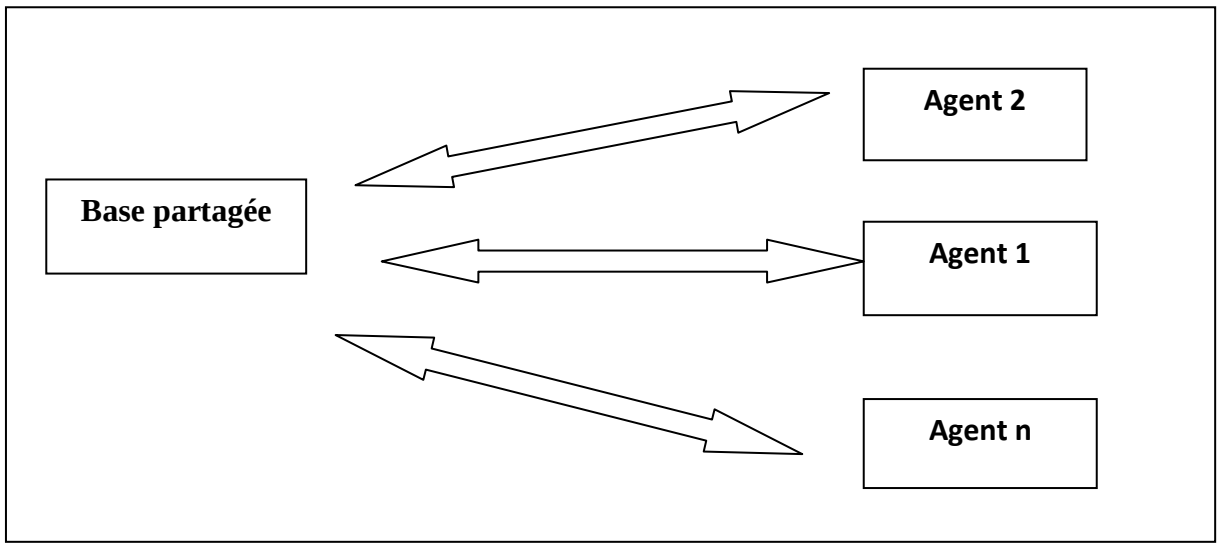


Figure 2.4 : Communication par partage d'information.

6.3. Négociation.

Les activités des agents dans un système distribué sont souvent interdépendantes et entrainent des conflits. Pour les résoudre, il faut considérer les points de vue des agents, les négocier et utiliser des mécanismes de décision concernant les buts ciblés. Le processus de négociation ne consiste pas forcément à trouver un compromis mais peut s'étendre à la

modification des croyances d'autres agents pour faire prévaloir un point de vue. Pour mener à bien ce type de processus, il est nécessaire de suivre un protocole qui facilite la convergence vers une solution. [24]

6.4. Coordination.

Les études sur la communication et la coordination ont généralement été effectuées de façon isolée et ont donc produit des résultats et des outils indépendants les uns des autres. Pourtant la communication et coordination sont complémentaires et les mécanismes les supportant doivent permettre leur intégration au sein d'un même environnement et au service des mêmes procédés [21].Le travail des agents doit être temporellement coordonné. Ceci permet aux agents de prendre en compte toutes les tâches en évitant la duplication du travail. Dans les systèmes IAD, on distingue deux schémas principaux de coordination [24] : une coordination au moyen d'un système capable de déterminer et de planifier les actions des différents agents, ou capable de donner une totale autonomie aux agents qui, à leur tour, identifient les conflits pour les résoudre localement..

7. SMA dans le domaine de transport

Dans le domaine de transport, plusieurs modèles se basent sur les SMA pour la résolution des problèmes. Parmi ces problèmes nous citons les suivants [9] :

- ***le contrôle du trafic aérien*** : En ce qui concerne le contrôle du trafic aérien, M. Ljungberg et A. Lucas propose en 1992 un SMA, nommé OASIS (*Optimal Aircraft Sequencing using Intelligent Scheduling*), pour le contrôle du trafic aérien de la région de Sydney en Australie. Il a pour but de réduire la congestion du trafic aérien en maximisant l'utilisation des pistes en ordonnançant les atterrissages.

- l***e transport de marchandises :*** Fischer et ses collèges présentent en 1995 une plate-forme, « MARS », de simulation multi-agent pour le transport des marchandises. Elle décrit un scénario lié à des compagnies de transport géographiquement distribuées, qui gèrent des commandes arrivant en temps réel. La coopération dans ce système se base sur une négociation à l'aide d'une version évoluée du « Contract Net Protocole ».

- ***systèmes d'aide au pilotage des avions militaires*** : ces systèmes ont été proposés par B. Chaib-draa en 1995[9]. L'objectif de ces systèmes est de simuler le fonctionnement global des avions dans le domaine militaires afin de minimiser les risques inattendu pendant le fonctionnement réel des avions.

- ***la gestion du trafic urbain*** : B. Chaib-draa aborde le problème de gestion du trafic urbain en 1996 par une approche agent. Les agents doivent constamment ajuster leurs actions pour éviter les bouchons et les accidents. Les agents dans cette approche sont dotés de lois sociales pour que leurs fonctions soient basées sur des compétences, ce qui implique des activités de coordination rapide et sans effort entre eux.

- ***l'optimisation, la recherche et la composition des itinéraires multimodaux :*** ce problème est développé par K. Zidi en 2006. Le transport multimodal est un mode de transport qui peut être assuré par plusieurs opérateurs de transport. Zidi et Kamoun proposent des approches multi-agents où chaque opérateur de transport est modélisé par un agent autonome. Ces agents coopèrent afin de répondre aux requêtes complexes qui nécessitent l'intervention de plusieurs opérateurs.

8. Les méthodes d'optimisation.

Comme nous l'avons précisé dans le chapitre précédant, le problème de transport à la demande est considéré comme un problème d'optimisation combinatoire multi objectif qui a fait l'objet de nombreux travaux de recherche. Il appartient à la catégorie des problèmes NP-difficile. Dans ce qui suit nous essayons de présenter une classification des méthodes d'optimisation utilisées dans la résolution de ce dernier.

Il existe, dans la littérature, une panoplie des méthodes de résolutions ou d'optimisations.ces méthodes peuvent être classées globalement, en deux grandes familles :

- Méthodes exactes où la solution obtenue est optimale.
- Méthodes approchées qui permettent d'obtenir de bonne solution sans tout fois pouvoir garantir leur optimalité.

Le schéma suivant résume la plupart de ces méthodes :

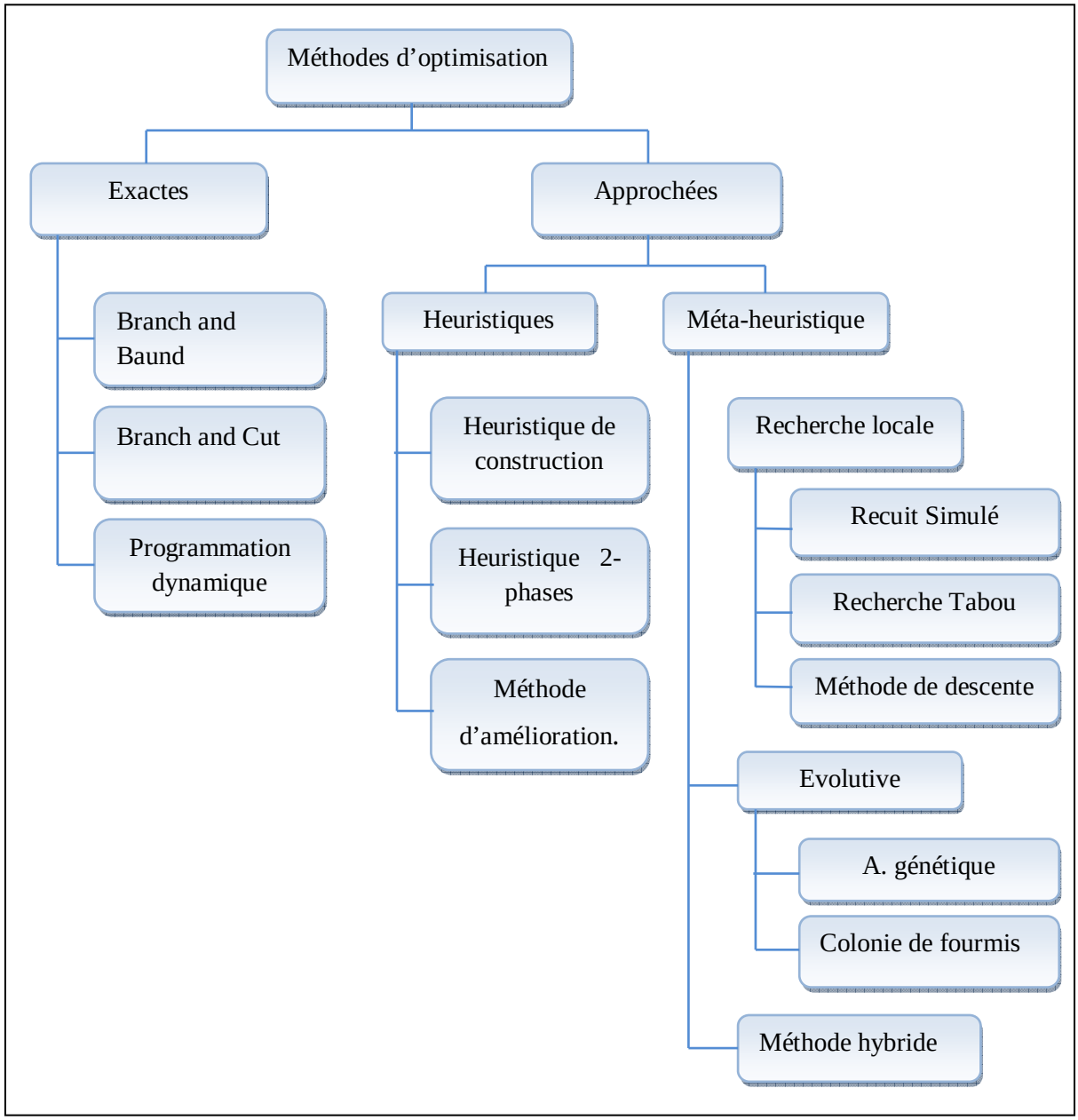


Figure2.5 : les principales méthodes d'optimisation.

8.1. Les méthodes exactes.

Une méthode de résolution est dite exacte si elle garantit l'obtention d'une solution optimale du problème. En effet, le temps de résolution de certains problèmes par les méthodes exactes connues augmente d'une manière exponentielle avec leurs tailles. Nous pouvons distinguer trois grands types de méthodes exactes [9]:

- Les méthodes de séparation et d'évaluation progressives (Branch and Bound).
- La méthode de programmation linéaire en nombre entière (Branch and Cut).

- La programmation dynamique.

8.2. Les méthodes approchées.

Les méthodes approchées cherchent des solutions proches de l'optimum en un temps raisonnable. L'optimalité n'est donc plus primordiale. D'une manière générale, ces méthodes ne garantissent pas la découverte de la meilleure solution, mais si le problème à optimiser possède plusieurs bonnes solutions proches de l'optimum, elles ont de grandes chances d'en trouver une . Cette classe de méthode regroupe d'une par, des heuristiques simples et d'autre part des méthode-aléatoire stochastiques. Ces méthodes sont connues sous le nom de méta-heuristiques[9].

8.2.1. Les heuristiques.

Les heuristiques constituent une alternative très intéressante pour traiter les problèmes d'optimisation de grande taille en un temps de calcul très petit, une heuristique représente un bon compromis entre le temps de recherche et la qualité de solution. Feignebaum et Feldman (1963) définissent une heuristique comme une règle d'estimation, une stratégie, une astuce, une simplification, ou tout autre sorte de système qui limite drastiquement la recherche des solutions dans l'espace des configurations possibles [9]. Parmi les heuristiques classiques utilisées dans la résolution du problème de TAD nous citons trois classes :

8.2.1.1. Les heuristiques de constructions.

L'idée de base d'une heuristique constructive est de réduire la taille du problème à chaque étape pour limiter progressivement l'ensemble des solutions réalisables (Figure 2.4). Les heuristiques constructives partent d'une solution vide et construisent étape par étape une solution finale *s* de *S*. les choix effectués à chaque étape *k* sont faits selon certaines règles [6]. Ces règles dépendent donc du problème considère. Bien évidente, à chaque étape k, l'ensemble des solutions réalisables S est donc réduit en un ensemble $s^k \subseteq S$. Ces heuristiques sont caractérisées par la facilité de mise en œuvre et la rapidité d'exécution. Par contre, la faible qualité des solutions trouvées est en général leur grand défaut [9]

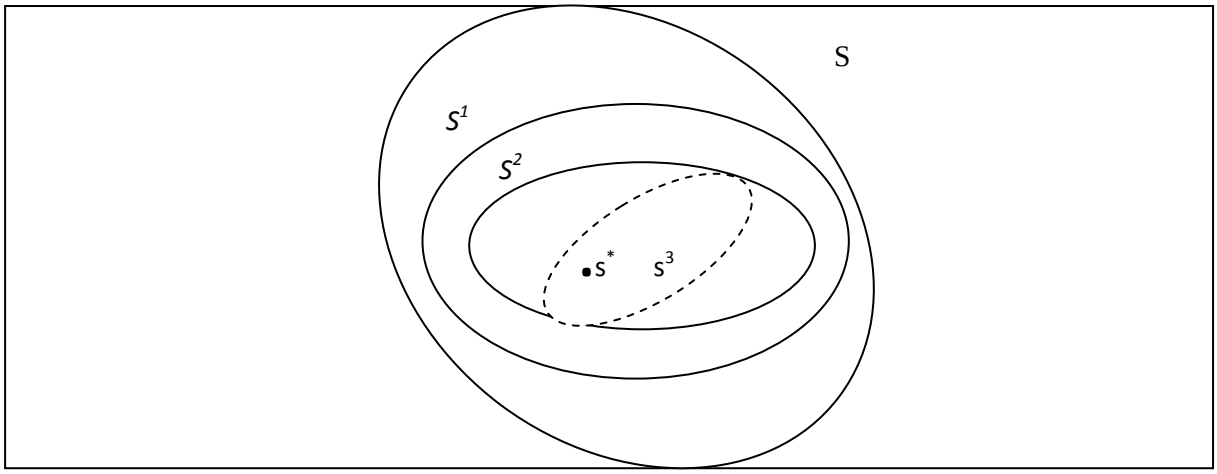


Figure 2.6 : Exploration de l'ensemble des solutions réalisables par une méthode constructive

8.2.1.2.Heuristique en 2-phases.

On distingue deux types d'heuristiques 2-phases[9]:

- Groupe en premier, route en second : (Route First - Cluster Second) Il s'agit de faire des sous ensembles de sommets voisins pour lesquels on construit ensuite une tournée et chaque sous ensembles est assigné à un véhicule.
- Route en premier, groupe en second : (Route First - Cluster Second) le principe de cette heuristique est de construire une tournée comportant un grand nombre de client et traite toutes les tâches en ignorant certaines contraintes, ensuite corrigée et découpée pour respecter les contraintes afin d'obtenir des solutions acceptables pour le problème.

8.2.1.3. Les méthodes d'amélioration:

Ces méthodes ont été initialement mise en œuvre par croes (1985) pour le TSP. Elles sont fondées sur le concept de k-échanges, cette approche est très utilisée dans d'autres problèmes tels que le problème de tournées de véhicules et de TAD [1].

Ces méthodes peuvent être utilisées dans la génération d'une solution initiale pour les méta-heuristiques, comme elles font appel à des méthodes de recherche locale, qui améliorent progressivement une solution initiale déjà obtenue. À chaque itération d'une recherche locale, il y aura une exploration d'un voisinage de la solution actuelle pour trouver une meilleure solution. Dans ce voisinage des solutions, nous cherchons à trouver en général la meilleure solution et pour chaque cas nous avons un critère d'arrêt spécial. Ces méthodes ont été développées pour aider la recherche à échapper aux minima locaux [9].

8.2.2. Les méta-heuristiques

Les méta-heuristiques forment une grande famille de méthodes d'optimisation qui visent à résoudre les problèmes d'optimisation difficile. Les problèmes d'optimisation concernés représentent aussi bien les problèmes d'optimisation combinatoire à variables discrètes que les problèmes d'optimisation globale à variables continues. Ces méthodes sont caractérisées par un haut niveau d'abstraction, ce qui permet de les adapter à une large gamme de problèmes différents, allant de la simple recherche locale aux algorithmes complexes de recherche globale. Deux grandes catégories de méta-heuristiques sont utilisées dans la plupart des problèmes d'optimisation[9][1]:

8.2.2.1. Méthodes de recherche locale.

Ces méthodes représentent une évolution des méthodes classiques d'amélioration itérative en acceptant des solutions voisines moins bonnes que la solution courante pour échapper aux optima (minima ou maxima) locaux. Les méthodes de recherche locale commencent à partir d'une solution réalisable $x0$, choisie arbitrairement de l'ensemble des solutions réalisables X. Les différentes versions de ces méthodes varient selon le choix de la solution voisine et le critère d'arrêt. Le passage d'une solution réalisable à une autre se fait selon un ensemble de modifications élémentaires qui dépend de la résolution adoptée. Les méthodes les plus connues sont[9][3] : Le Recuit Simulé, La méthode de descente, et la méthode Tabou.

a) <u>Méthode de recuit simulé</u> *:*

Le recuit simulé est souvent présenté comme la plus ancienne des méta-heuristiques proposée en 1983 par Kirkpatrick Kirkpatrick, Gelatt et Vecchi, en tout cas, la première à mettre spécifiquement en œuvre une stratégie d'évitement des minima locaux .Elle s'inspire d'une procédure utilisée depuis longtemps par les métallurgistes qui, pour obtenir un alliage sans défaut, chauffent d'abord à blanc leur morceau de métal, avant de laisser l'alliage se refroidir très lentement (technique du recuit) . Pour simuler l'évolution d'un système physique instable vers un état d'équilibre thermique à une température T fixée la méthode de recuit simulé exploite l'algorithme de *Métropolis*. Par analogie à l'énergie d'un matériau, la fonction objective du problème d'optimisation à résoudre est donc à minimiser [9]. Le recuit simulé a été appliqué pour résoudre plusieurs problèmes souvent de grandes tailles et a donné de très bons résultats. Mais ses limites se présentent dans les temps de calcul qui peuvent devenir très

importants, ce qui a conduit à des parallélisassions de la méthode. La méthode du Recuit Simulé est souple vis-à-vis des évolutions du problème et facile à implémenter, elle est décrite comme suit :

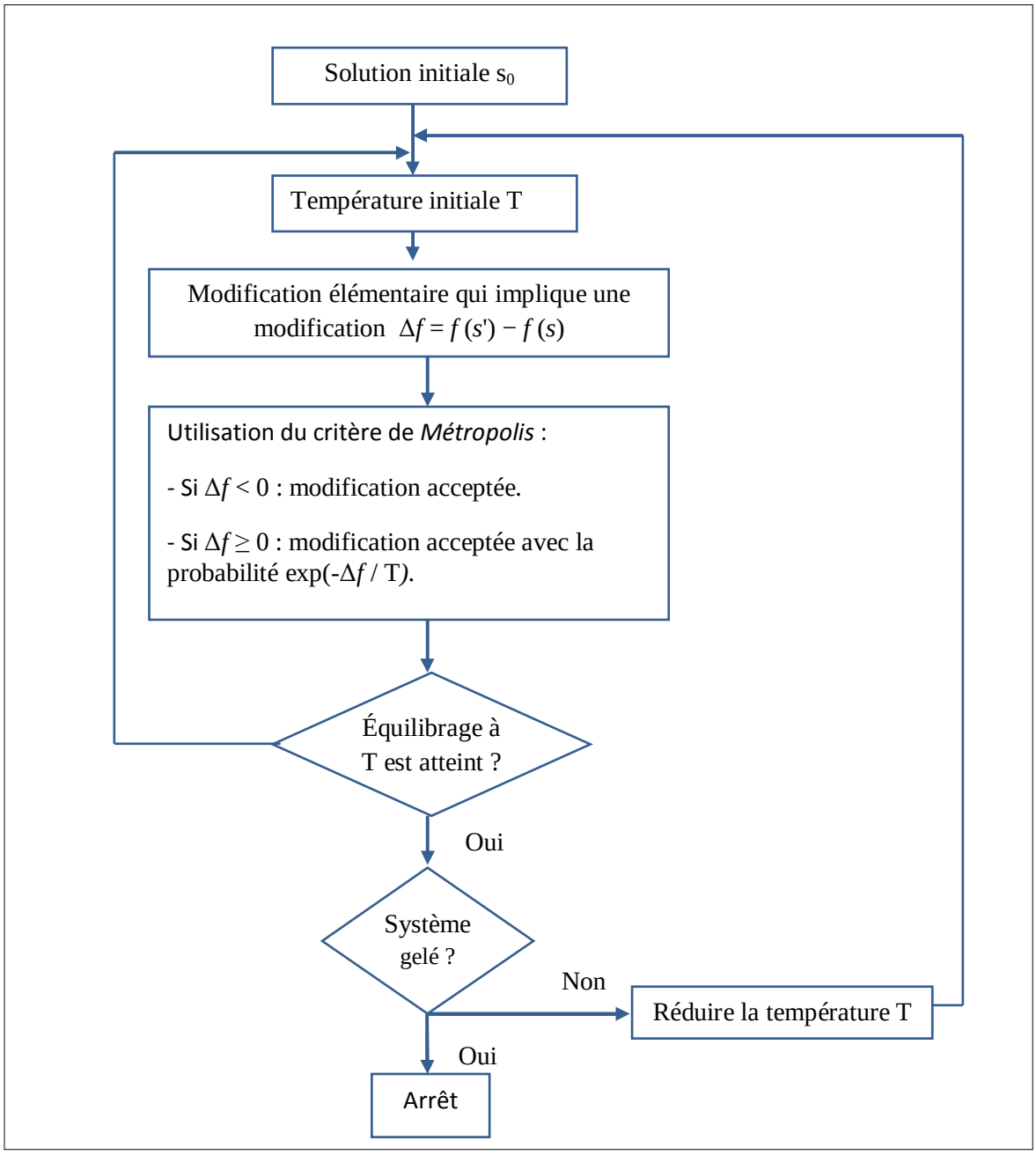


Figure 2.7 : Les étapes de la méthode de recuit simulé.

b) *Recherche tabou :*

La Recherche Taboue est une stratégie de recherche dont les principes ont été proposés pour la première fois par Fred Glover dans les années 80. Le principe de cette méthode est à chaque itération le voisinage de la solution courante est examiné. L'algorithme enregistre la meilleure solution parmi les voisins, même si elle est moins bonne que la solution courante. L'acceptation des solutions moins performantes que la solution courante permet d'éviter de tomber dans un optimum local. Pour échapper de tourner dans un cercle entre plusieurs solutions, l'algorithme interdit le passage par des solutions récemment visitées. En pratique la méthode stocke dans une liste taboue *T* les attributs des dernières solutions visitées. Dans l'itération suivante, la meilleure nouvelle solution voisine enlève la solution la plus ancienne dans la liste. Dans d'autres cas, la méthode mémorise les mouvements réalisés plutôt que les solutions. Ensuite, on interdit les mouvements inverses. Cette technique est rapide et consomme peu de mémoire. La Recherche Tabou peut être décrite comme suit :

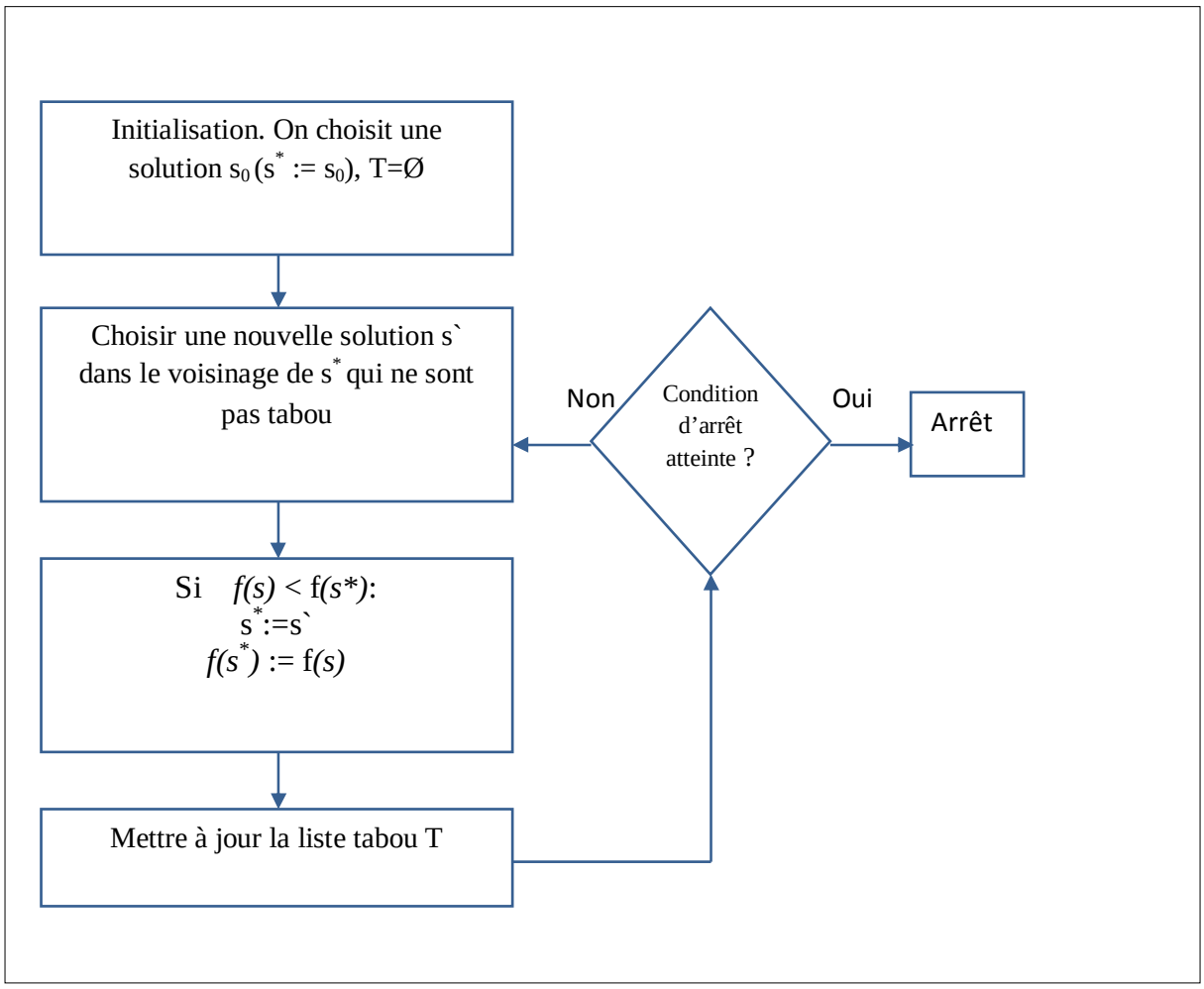


Figure 2.8 : Schéma de l'algorithme de la recherche tabou

c) ***<u>La méthode de descente :</u>*** La méthode de descente est très ancienne. Elle est caractérisée par la simplicité est la rapidité [12]. A chaque pas de la recherche, cette méthode progresse vers une solution voisine de meilleure qualité. La descente s'arrête quand tous les voisins candidats sont moins bons que la solution courante, c'est-à-dire lorsqu'un optimum local est atteint. On distingue différents types de descentes en fonction de la stratégie de génération de la solution de départ et du parcours du voisinage : la descente déterministe (en général on choisit la solution apportant la plus grande amélioration), la descente stochastique (choix aléatoire parmi les solutions améliorant la solution initiale) et la descente vers le premier meilleur.

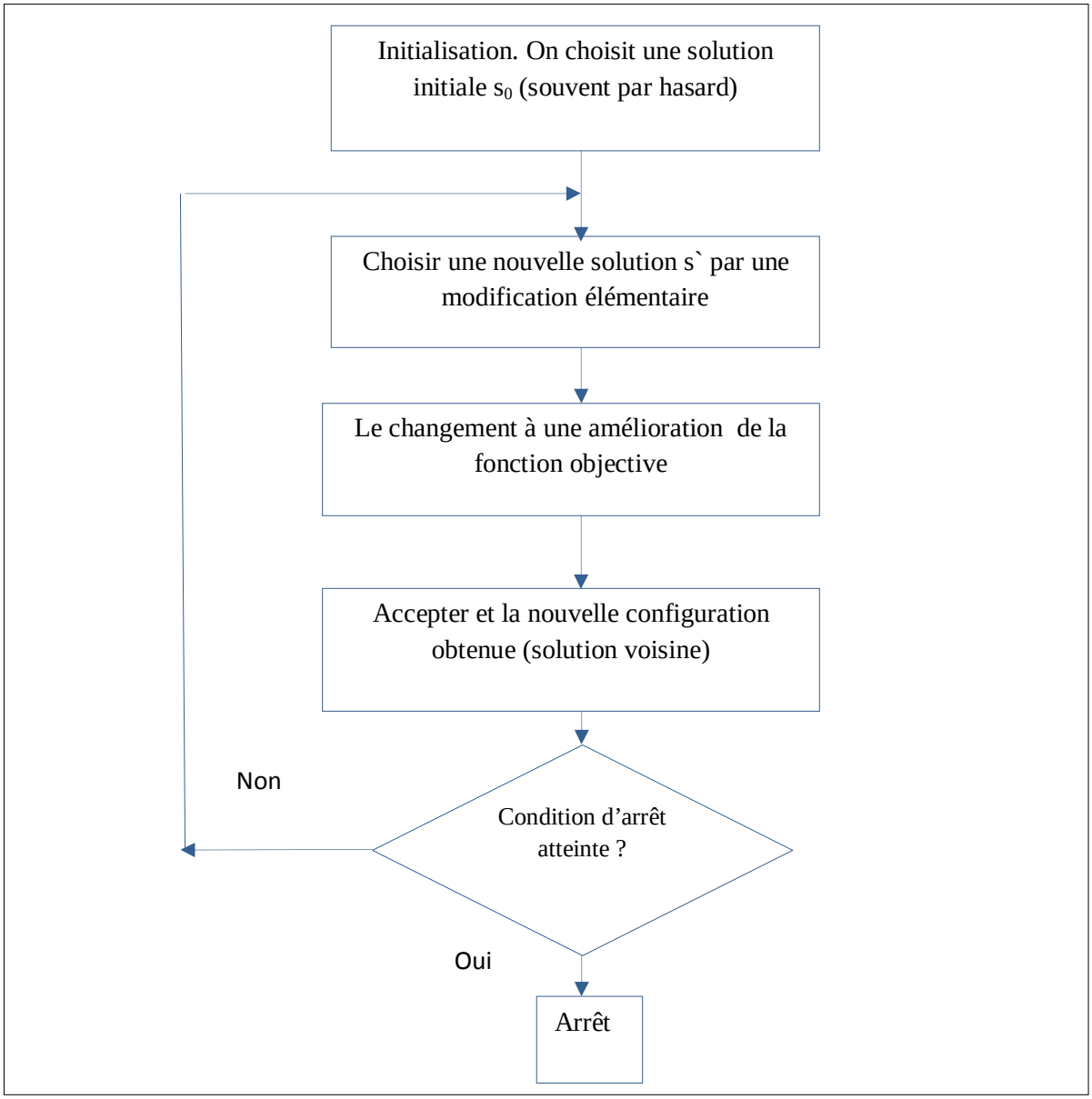


Figure 2.9 : Les différentes étapes de la méthode descente**.**

8.2.2.2. Méthodes évolutives.

La principale différence d'une approche évolutive par rapport aux autres approches citées précédemment, c'est qu'elle manipule un groupe de solutions réalisables, appelé *population*, elle se distingue des heuristiques présentées précédemment par le fait qu'elles gèrent un ensemble d'éléments. Ces élément sont appelées des individus et sont regroupés dans une population selon des règles bien précises. A chaque itération (que l'on va appeler génération dans le cadre des algorithmes évolutifs), on passe d'une population courante à une population voisine.

La fonction objectif permet d'associer à chaque individu de la population une valeur reflétant la qualité de la solution qu'il représente. En partant d'une population initiale, un algorithme évolutif essaie d'améliorer la qualité des individus en faisant évoluer cette population. Cette évolution se fait en générale en appliquant à chaque génération une phase de coopération collective (d'échange d'informations) entre tous les individus d'une même population et une phase d'adaptation individuelle. Les plus répandus de ces méthodes sont les suivantes[9][1] :

a) <u>Les algorithmes génétiques :</u>

Les algorithmes génétiques ont été développés par John Holland en 1975, c'est la variante la plus connue des algorithmes évolutionnaires, d'ailleurs, par abus de langage, beaucoup de spécialistes désignent et continuent à désigner les approches évolutionnaires par le terme « algorithmes génétiques ».

Comme leur nom l'indique, les algorithmes génétiques s'inspirent du patrimoine héréditaire d'un individu (génotype) représenté par ses chromosomes. L'interaction du génotype d'un individu avec son environnement détermine son phénotype qui peut être modifié par mutation. Le phénotype est évalué par décodage du génotype, qui est une chaine de symboles souvent binaires, dans le but de donner une valeur de performance exploitable par les opérateurs de sélection (couleur des yeux, des cheveux, traits du visage, etc.). Les opérateurs de variation (croisement et mutation) sont relatifs à la représentation binaire puisqu'ils agissent sur les chaines binaires des génotypes. Le croisement correspond à la phase de coopération entre les individus alors que la mutation correspond à la phase d'adaptation individuelle.

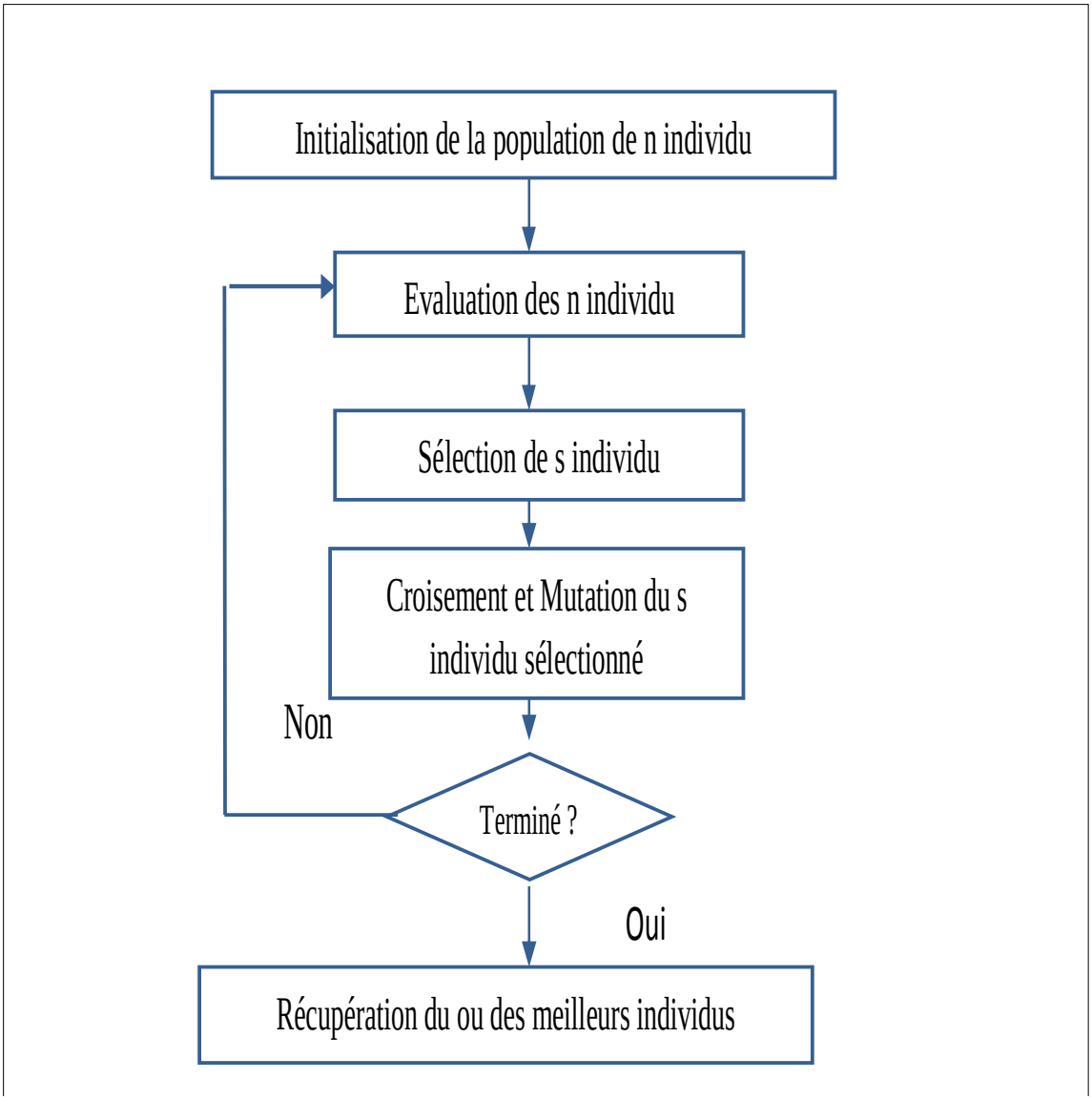


Figure 2.10 : Principe des algorithmes génétiques.

b) <u>Les colonies de fourmis :</u>

C'est une nouvelle méthode de résolution des problèmes combinatoires. Cette méta-heuristique a été proposée pour la première fois par Dorigo en 1992. Les colonies de fourmis sont basées sur le comportement réel et la communication chez les fourmis. Il est connu que les fourmis sont capables de déterminer le chemin le plus court entre leur nid et une source de nourriture grâce à la phéromone déposés sur les trajets parcourus. Cela peut paraitre surprenant au premier abord mais un chemin plus court reçoit plus de phéromones qu'un chemin plus long. Cet algorithme a été appliquée sur le problème du voyageur de commerce (TSP : Travelling Salesman problem).

c) La méthode de recherche distribuée :

C'est une variante peu connue des algorithmes évolutionnaires, cette méthode assure l'évolution des solutions à l'aide d'un opérateur de combinaison sans imposer de règles de codage. Enfin pour améliorer le résultat final, il est permis d'utiliser l'une des méthodes de la recherche locale.

8.2.2.3. Méthodes hybrides.

La tendance actuelle est d'avoir recours à des algorithmes dits hybrides. Il existe plusieurs types d'hybridations possibles[9][3] :

- On peut utiliser par exemple une Recherche Locale dans les méthodes évolutives. Des résultats intéressants ont été remarqués lors de la combinaison des algorithmes évolutifs avec des algorithmes de recherche locale. L'avantage est que la Recherche Locale réduit le danger de passer à côté d'une solution optimale sans la voir et les méthodes évolutives sont excellentes pour détecter de bonnes régions dans l'espace de recherche.

- On peut même exécuter en parallèle diverses méta-heuristiques, voire même plusieurs fois la même méta-heuristique mais avec divers paramètres.

- Une troisième forme d'hybridation consiste à combiner les méta-heuristiques avec des méthodes exactes. Une méta-heuristique peut par exemple fournir des bornes à une méthode de type branch-and-bound et une méthode exacte peut donner lieu à une technique efficace pour la détermination du meilleur voisin d'une solution (ce qui peut s'avérer plus judicieux que de choisir la meilleure solution parmi un échantillon de voisins)

9. Avantages et Inconvénients des méthodes :

Dans le tableau suivant nous citons les avantages et inconvénients de quelques méthodes :

Méthodes	Avantages	inconvénients
La méthode de descente	grande simplicité de mise en œuvre	l'efficacité des méthodes de recherche locale simples (descente) est très peu satisfaisante car la recherche s'arrête au premier minimum local rencontré.
La méthode recuit simulé	Elle offre des solutions de bonne qualité, tout en restant simple à programmer et à paramétrer. Il offre autant de souplesse d'emploi que l'algorithme de recherche local classique : on peut inclure facilement des contraintes dans le corps du programme.	une fois l'algorithme piégé à basse température dans un minimum local, il lui est impossible de s'en sortir tout seul.
La recherche tabou	L'efficacité de la méthode Tabou fait qu'elle est largement employée dans les problèmes d'optimisation combinatoire : elle a été testée avec succès sur les grands problèmes classiques (voyageur de commerce, ordonnancement d'ateliers) et elle est fréquemment appliquée sur les problèmes de constitution de planning, de routage, d'exploration géologique, etc.	la méthode Tabou exige une gestion de la mémoire de plus en plus lourde à mesure que l'on voudra raffiner le procédé en mettant en place des stratégies de mémorisation complexe.
Les algorithmes	parviennent à trouver de bonnes	les algorithmes génétiques sont

génétiques	solutions sur des problèmes très complexes,	coûteux en temps de calcul, puisqu'ils manipulent plusieurs solutions simultanément.
Les colonies de Fourmies	l'algorithme de colonies de fourmi offre finalement beaucoup de souplesse.	Demande une grande adaptation du problème aux principes des fourmis.

Tableau 2.2. Avantages et Inconvénients des méthodes.

10. Conclusion

Dans ce chapitre, nous avons présenté une étude bibliographique relative aux systèmes multi-agents (SMA) et les principes des méthodes d'optimisation. Ces deux domaines sont le fondement de plusieurs approches dédiées à la résolution du problème de transport à la demande. L'étude présentée nous incite à adopter ces domaines pour la construction de notre propre approche qui fera l'objet du prochain chapitre.

Chapitre III

Une Architecture multi-agent pour la Résolution du problème de TAD

1. Introduction.

Dans les premiers chapitres nous avons présenté le problème de transport à la demande et une description des différentes méthodes utilisées pour résoudre ce dernier, ainsi qu'un aperçu général des systèmes Multi-Agents et leur applications dans le domaine de transport.

Nous proposons dans ce chapitre une architecture multi-agents d'un système de Transport A la Demande (TAD) qui adapte l'offre à la demande afin de satisfaire les requêtes de transport des clients.

Dans ce qui suit, nous commençons premièrement par une description du problème à optimisé. Nous présentons les différents types d'agents utilisés dans la construction de notre architecture. Après, nous exposons le protocole de négociation adopté. Enfin, nous détaillons la description des algorithmes d'optimisations utilisés.

2. Description du problème.

- **Les données :**
  - **Opérateur de transport :** un opérateur possède un nombre connu de véhicules et une agence de réception des demandes clients.
  - **Les véhicules :** chaque véhicule possède une vitesse et une capacité (nombre de places).
  - **Les demandes :** un client (voyageur) envoi une demande vers l'opérateur, cette demande contient la station source et destination.
- **Problème :**

  Notre problème concerne la satisfaction d'un ensemble de demandes de transport basée sur le bon choix de chemin à traversé et en réduisant le cout total pour l'opérateur de transport. Alors, notre premier objectif est lié principalement à l'ordonnancement des demandes des clients dans des tournées. On appelle l'ensemble de ces tournées **« solution** » du problème pour une journée.

  Notre deuxième objectif concerne le cas de panne, car il faut trouver une nouvelle affectation des requêtes dans le cas de panne de un des véhicules. Dans cette situation l'adoption d'un mécanisme de négociation semble primordiale pour assurer la satisfaction des clients.

Mathématiquement nous définissons notre problème comme suite :

- Un ensemble de station *X*, où chaque station *i* est caractériser par une position (x_i, y_i) tel que la distance entre deux stations *i* et *j* est définit par la distance euclidienne :

$$dij = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} .$$

- Un ensemble de requêtes (demandes) *N*, où chaque requête *i* est caractérisée par :
  - **Une station source :** la station pour prendre le client.
  - **Une station destination :** la station pour déposer le client.

- Un dépôt de véhicules caractérisés par la capacité C et la vitesse moyenne V.

- **Contraintes :**

Une solution faisable doit satisfaire les contraintes suivantes :

- **Pairing :** un client doit se déplacer à partir d'une source S_i vers une destination S_j avec le même véhicule.
- **Précédence :** une station source doit être visitée avant une station destination.
- **Capacité de véhicule :** le nombre de requête satisfaite par un véhicule ne doit pas dépasser la capacité de ce dernier.
- **Durée maximal des tournées :** la durée d'une tournée ne doit pas dépasser T car le véhicule doit retourner au dépôt pour ravitailler en carburant.

Donc, l'objectif est de satisfaire les demandes des clients sur une journée, de façon à minimiser la durée totale des tournées en respectant les contraintes précédentes. Il s'agit donc d'un problème mono-objectif.

3. Principe de l'approche proposée.

3.1. Architecture générale.

Notre idée générale consiste à modéliser le système de TAD selon une approche multi-agents. Nous choisissons de doter les agents d'algorithmes d'optimisation et de leur laisser les moyens de communiquer directement via des protocoles de négociation. Notre architecture est basée sur deux types d'agents : un agent central qui s'occupe de la construction et l'optimisation des tournées et aussi l'affectation de ces dernières aux différents véhicules (agents) à partir des demandes envoyées, et nous avons aussi les agents véhicules qui s'occupent de la négociation dans les cas de panne.

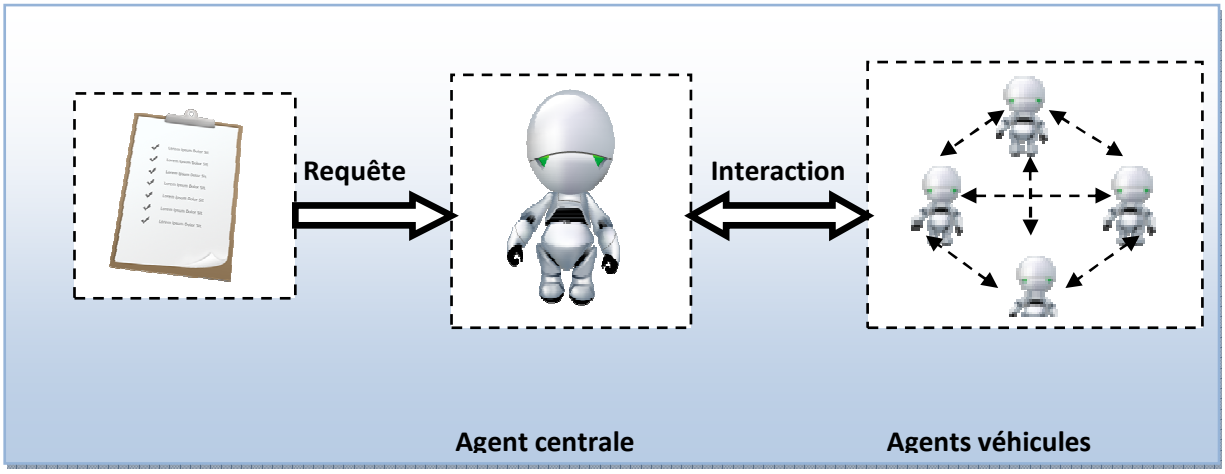


Figure 3.1 : Architecture générale du système

3.2. Comportement des agents dans le système.

Dans ce qui suit nous essayons de données une description détaillée des fonctionnalités assurées par les deux types d'agents (central et véhicule).

3.2.1. Agent central.

L'agent central dans notre architecture joue le rôle d'un bureau de réception dans une agence de transport à la demande. Il assure les fonctionnalités d'une interface entre les clients et le système de transport.

Cet agent est lancé dés le démarrage du système, il est responsable de la génération et l'optimisation des tournées et du lancement des agents véhicules avec les tournées correspondantes. Le comportement de l'agent central est résumé dans le diagramme d'activité suivant :

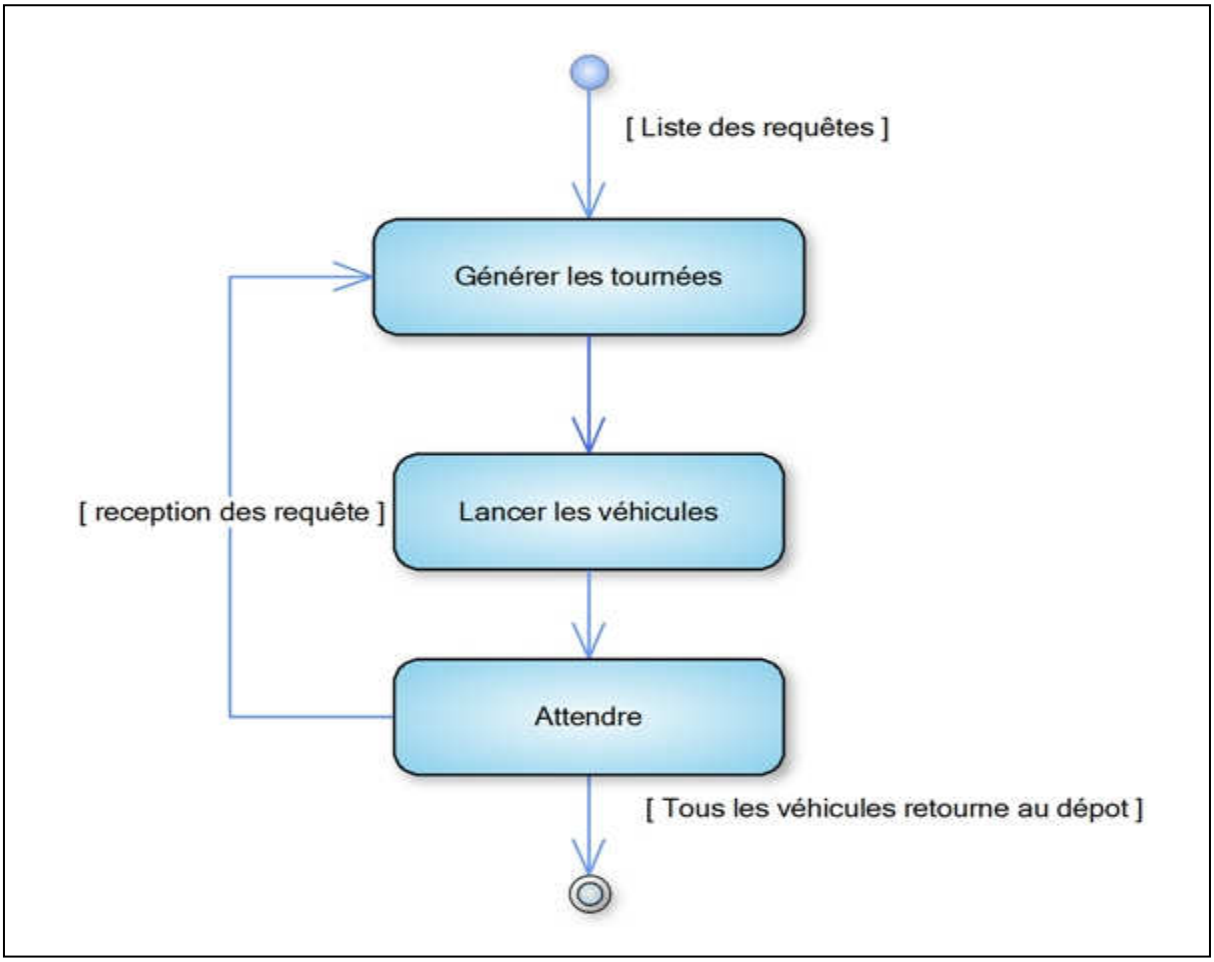


Figure 3.2 : comportements de l'agent central.

- **Générer les tournées :** l'agent central s'occupe de la génération des tournées par la métaheuristique « recherche tabou ».

- **Lancer les véhicules :** après la génération des tournées l'agent central crée un ensemble d'agent équivalent au nombre de tournées et il affecte à chaque agent une tournée comme plan de circulation.

- **Attendre :** après le lancement des agents véhicule, l'agent central passe à l'état d'attente pour être en mesure de recevoir les messages qui peuvent arriver de la part d'un véhicule en panne. Ces messages contiennent l'ensemble de demandes non satisfaites par les véhicules actifs même après la négociation.

3.2.2. Agent véhicule.

Les agents véhicules sont lancés par l'agent central, ce type d'agent simule le comportement d'un véhicule réel c'est-à-dire il se déplace suivant un plan de circulation (tournée). Le comportement d'un agent véhicule est résumé dans le diagramme d'activité suivant :

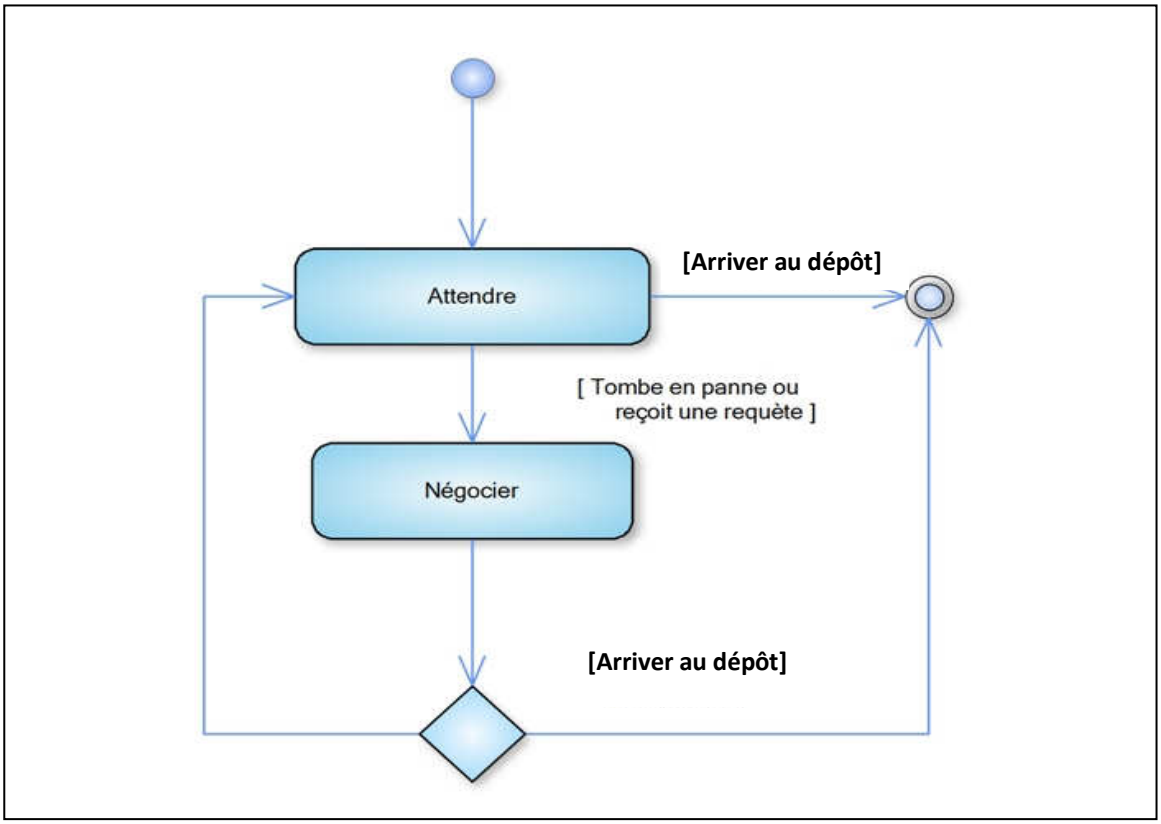


Figure 3.3 : comportements de l'agent véhicule.

- **Attendre** : Pendant que l'agent véhicule patrouille sa tournée, il peut tomber en panne dans un temps imprévu ou bien il peut recevoir une proposition d'insertion d'une

requête d'un autre véhicule en panne, donc il doit être toujours en attente d'un de ces deux événements.

- **Négocier :** Le processus de négociation est déclenché généralement par un agent en panne qui n'arrive pas à achever sa tournée. Alors, cet agent entre en interaction avec les autres agents afin d'assurer une insertion optimale des requêtes restantes dans sa tournées après la panne. Dans la section suivante nous présentons premièrement, le protocole de négociation que nous avons adopté (Contract-Net protocole), puis, nous spécifions comment ce dernier est utilisé dans la résolution de notre problème.

3.3. Principe de négociation entre agents (protocole Contract-Net).

Le protocole **réseau contractuel** ("Contract-Net" en anglais) a été une des premières approches utilisées dans les systèmes multi-agents pour résoudre le problème d'allocation des tâches [Davis et Smith, 1983]. Il consiste en un contrat élaboré entre deux participants : **Le contractant** et **le manager**. Le contractant est garant de l'exécution d'une sous-tâche et de la transmission de ses résultats au manager. Le manager est le responsable de la distribution des tâches et du traitement des sous-résultats. [18]

Ce protocole est qualifié de type 'sélection mutuelle' puisque pour signer un contrat l'agent choisi doit s'engager envers le manager pour l'exécution de la tâche et le manager de ça part ne sélectionne que l'agent ayant fourni la proposition la plus avantageuse [16].

Le protocole se déroule en trois phases distinctes. La figure suivante illustre les différentes étapes du protocole Contract-Net :

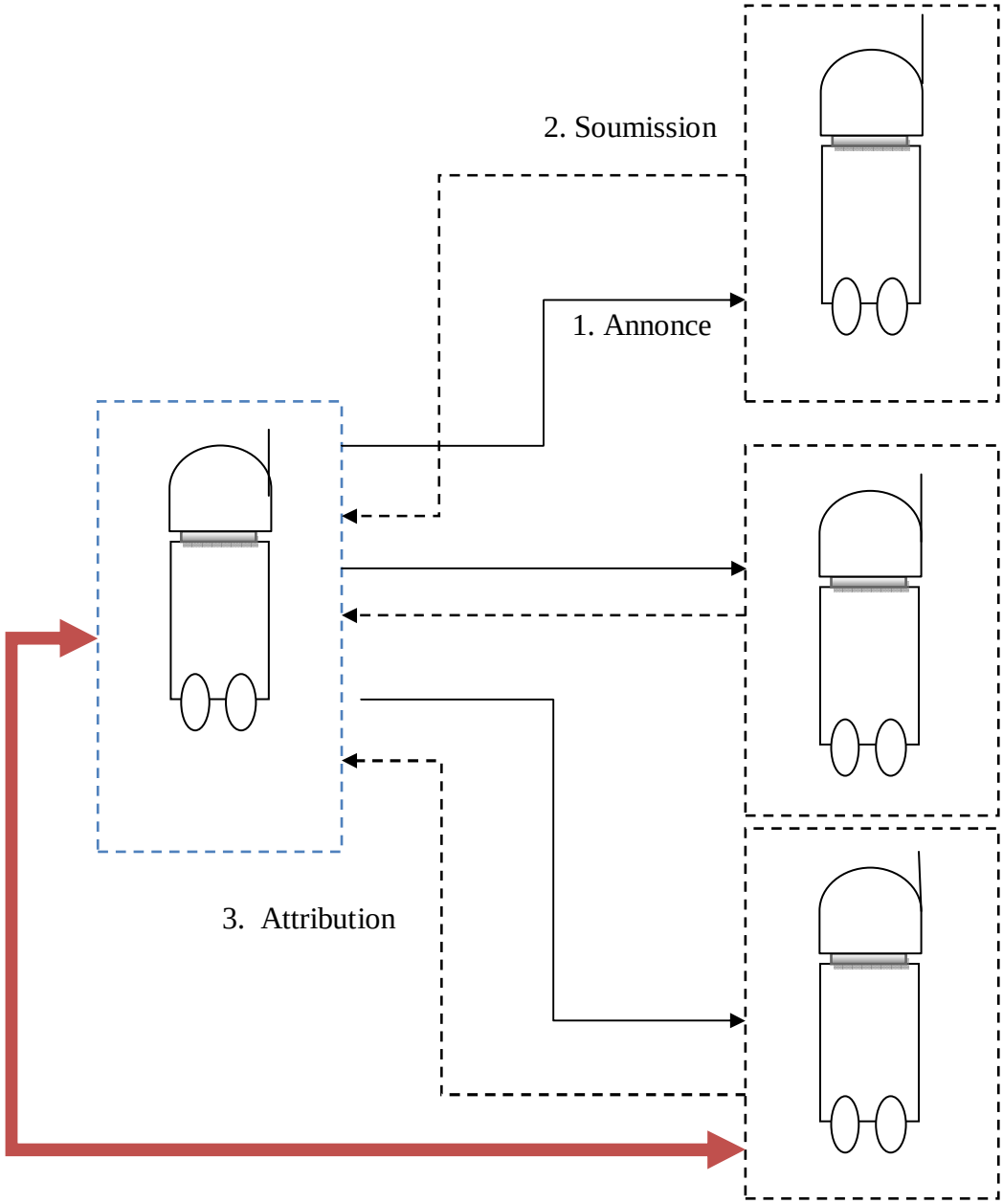


Figure 3.4 : Etapes du protocole réseau contractuel (Contract-Net).

- **Annonce des sous-tâches :** L'agent qui doit exécuter une tâche (le manager) commence par décomposer cette dernière en plusieurs sous-tâches. Le gestionnaire annonce chaque sous-tâche sur un réseau d'agents (les contractants).

- **Soumission :** Les agents qui reçoivent une annonce d'une sous-tâche à accomplir évaluent l'annonce. Les agents qui ont les ressources appropriées, l'expertise ou l'information requise pour accomplir la tâche, envoient au gestionnaire des soumissions ("bids" en anglais) qui indiquent leurs capacités à réaliser la tâche.

- **Attribution des tâches :** Le manager rassemble toutes les propositions qu'il a reçues et alloue la tâche à l'agent qui a fait la meilleure proposition. Ensuite, le gestionnaire et les contractants échangent les informations nécessaires durant l'accomplissement des tâches.

3.3.1. Application du protocole Contract-Net.

Le paragraphe suivant montre les différentes étapes concernant la négociation entre les agents véhicule en cas de panne :

1) L'agent en panne joue le rôle d'un manager. Alors, dans le cas de panne le manager (véhicule en panne) provoque les autres véhicules actifs avec un message qui contient le détail de sa tournée c'est-à-dire les requêtes non satisfaites au moment de la panne.

2) Lors de la réception des requêtes envoyées par le véhicule en panne, chaque agent actif calcule la meilleure insertion (utilisation de l'algorithme d'insertion) de ses requêtes dans sa tournée et il envoi le résultat vers le manager (véhicule en panne).

3) Après la réception de toutes les réponses envoyées par les véhicules actifs le manager choisit les meilleures insertions de ses requêtes dans les tournées des autres agents et les informent qu'il a accepté l'insertion chez eux.

4) Si le véhicule actif reçoit un message d'acceptation de la part de véhicule en panne il insère la requête dans sa tournée puis il répond avec une confirmation d'insertion.

5) s'il reste des requêtes que les agents véhicules n'arrivent pas à les insérer dans leurs tournées, l'agent en panne envoi ces dernières à l'agent central pour lancer d'autres véhicules afin de les satisfaire.

Le comportement des agents véhicule se résume par l'algorithme suivant :

Algorithme 3.1 : Protocole de négociation

```
Si Véhicule.etat = EnPanne alors (rôle de l'agent manager)
    Tant que Not-end-of(Liste_des_requêtes_restantes) faire
        Diffuser une requête et attendre la réception des réponses.
    Tant que la boite des messages < > vide faire
            Message ← premier message dans la boite/(supprimer le message de la boite).
            Si Message. Performative = ACCEPT ou Message. Performative = REFUSE alors
                Nbr_Reponses ← Nbr_Reponses+1
                Si Message. Performative = ACCEPT alors
                    Surcout ← Message. Contenu
                    Émetteur ← Message. Émetteur
                    Ajouter surcout et Émetteur à la liste des réponses.
                Fin si
                Si Nbr_Reponses = Nbr_véhicules alors
                    Classer les surcoûts pour cette requête
                    Envoyer un message ACCEPT PROPOSAL au véhicule qui a
                    le min surcoût
                    Supprimer la requête de la liste des requêtes restantes
                Fin si
            Fin si
        Fin tant que
    Fin tant que
            Si Liste_des requetes_restantes < > vide alors
            Envoyer la liste à l'agent central
Sinon (rôle des agents contractant en état d'attente)
    Message ←Consultation boite à lettre
        Si Message < > vide alors
            Si Message. Performative = propose_insertion alors
                Effectuer une insertion temporaire de la requête dans la tournée
```

```
        Si l'insertion est faisable alors
            Calculer le surcoût de l'insertion
            Répondre par un message ACCEPT contenant le surcoût
        Sinon
            Répondre par un message REFUSE
        Fin si
    Sinon
        Si Message. Performative = ACCEPT_PROPOSAL alors
            Modifier la tournée
            Confirmer l'insertion
        Fin si
    Fin si
Fin si
```

3.4. Algorithmes et méthodes d'optimisation utilisés.

Dans ce qui suit nous donnons une description des différents algorithmes moteurs des agents de notre architecture.

3.4.1. Algorithme d'insertion.

L'heuristique d'insertion est utilisée dans notre architecture par l'agent central pour générer une solution initiale. Le principe général de cet algorithme est le suivant : à partir des requêtes clients on doit établir des tournées via le principe d'insertion des requêtes l'une après l'autre. Avant l'insertion des requêtes il faut s'assurer que les contraintes annoncées précédemment ne sont pas violées par cette insertion.

Dans cet algorithme on parle du coût d'insertion qui est la distance ou bien la durée d'une tournée après l'insertion d'une requête. L'algorithme d'insertion est décrit comme suit :

Algorithme 3.3 : Heuristique d'insertion

```
LReq : Liste des requêtes
Solution : Liste des tournées
Tant que LReq < > vide faire
    Créer une tournée contenant seulement le dépôt
    Tirer une requête aléatoirement de LReq
    Insérer la requête dans la tournée
    Retirer la requête de LReq
    Tant que LReq < > vide faire
        Pour Chaque Requête de LReq faire
            Trouver la meilleure insertion faisable et calculer le coût d'insertion
        Fin pour
        Si Aucune insertion n'est faisable alors
            Sortir de la boucle
        Fin si
        Insérer la requête qui a le meilleur cout dans la tournée
        Retirer la requête de LReq
    Fin tant que
    Ajouter la tournée à solution
Fin tant que
```

3.4.2. Voisinage.

Cet algorithme est utilisé par l'algorithme de Recherche Tabou. Le voisinage d'une solution courante est construit avec l'opérateur de voisinage 2Opt, le principe générale de cet algorithme est le suivant : a partir d'un cycle cet algorithme retire 2 arêtes du cycle, et calcule un autre cycle en remplaçant ces arêtes par celles appropriées. L'algorithme suivant résume les différentes étapes de cet algorithme.

Algorithme 3.2 : 2_Opt

```
P = parcours ;
    Pour i=0 a NbrSommet-1 faire
        Pour j= i+1 a NbrSommet-1 faire
            K=i+1 ;
            L=j ;
            Tant que (k < l) faire
                [Inversion de l'ordre des sommets entre les indices i+1 et j]
                Tmp ← parcours[k] ;
                Parcours[k] ← parcours[l] ;
                Parcours[l] ← Tmp ;
            Fin Tant que
                Si (distance_parcours (parcours)> distance_parcours(p)) alors
                    Parcours ← p ;
                Sinon p=parcours ;
                Finsi
        Finpour

    Finpour
```

3.4.3. Recherche tabou.

Le but de la ***recherche Tabou*** est de trouver la meilleure solution du problème en partant d'une solution initiale comme solution optimale courante (cette solution est générée en utilisant ***l'heuristique d'insertion***). Puis, on fait appelle ***à l'algorithme de voisinage 2OPT*** pour choisir le meilleur voisin de la solution courante, cette étapes se répète à savoir le point d'arrêt (Nombre d'itération et la liste tabou). L'utilisateur est le responsable d'établir le point d'arrêt car le résultat se change avec le changement de ce paramètre. Le résultat final est une liste de tournées.

La structure générale de la méthode est décrite par l'algorithme suivant :

Algorithme 3.2 : Recherche tabou(adapté au problème de TAD)

TL : la liste tabou

It_Max : nombre Max d'itération permise sans amélioration

$S_{courante}$, $S_{optimale}$: Liste de tournées

$S_{courante}$ ⟵ Générer solution initiale

$S_{optimale}$ ⟵ $S_{courante}$

It ⟵ *0*

Tant que *It* ≤ *It_Max* **faire**

$S_{courante}$ ⟵ Meilleur voisin non tabou

Si *Coût*($S_{courante}$) < *Coût*($S_{optimale}$) **alors**

$S_{optimale}$ ⟵ $S_{courante}$

Si *Taille*(*TL*) = *TailleMax*(*TL*) **alors**

Retirer la tête de la liste *TL*

Fin si

Stocker $S_{optimale}$ dans *TL*

It ⟵ *0*

Sinon

It ⟵ *It* +*1*

Fin si

Fin tant que

4. Conclusion.

Dans ce dernier chapitre nous avons proposé une architecture multi-agent pour la résolution du problème de transport à la demande. Le fonctionnement global de l'architecture est assuré par deux types d'agents : un agent central et des agents véhicules. L'architecture est consolidée par un protocole de négociation pour gérer les cas de pannes. Concernant la notion d'optimisation nous avons opté pour trois algorithmes complémentaires : recherche Tabou, Heuristique d'insertion et l'algorithme 2OPT.

Dans le chapitre suivant, nous allons présenter l'environnement et les outils de développement ainsi que les interfaces graphiques de notre application.

Chapitre IV

Implémentation et Tests

1. Introduction.

Dans ce dernier chapitre nous allons présenter les différents outils de programmation utilisés dans le développement de notre application et nous donnons aussi une description des différents composants de cette dernière.

Le reste de chapitre est organisé en trois parties, la première partie concerne le langage de programmation, l'environnement de développement et les différentes installations nécessaires. Dans la deuxième partie, nous présentons les plateformes les plus utilisées dans le développement des systèmes multi-agent avec une justification de notre choix concernant la plateforme JADE. Enfin, nous donnons une description des interfaces graphiques de l'application.

2. Environnement de développement.

Pour le développement de notre application nous avons utilisé les outils suivants :

- **Machine :** Nous avons utilisé un ordinateur caractérisées par :
  - ✓ Processeur : Intel (R) Core(TM) i5-2450 CPU @ 2.50GHz 2.50GHz.
  - ✓ Mémoire installée (RAM) : 4,00 Go.
  - ✓ Système d'exploitation : Windows 7 Service Pack 01.
- **Logiciels :** Nous avons utilisé java comme langage de programmation et Eclipse Indigo (*Indigo Service Release 2*) comme environnement de développement intégré (IDE).Nous avons opté aussi pour *WindowBuilder Pro* **un plug-in** pour Eclipse, venant de Google, il permet d'éditer des interfaces graphiques directement dans l'IDE. Ce plug-in permet aux développeurs d'éditer des interfaces graphiques avec moins d'efforts et dans un temps réduit. Dans ce qui suit nous présentons les étapes d'installation de ce plug-in :

Etape 1 : Visitez ce lien : http://www.eclipse.org/windowbuilder/download.php

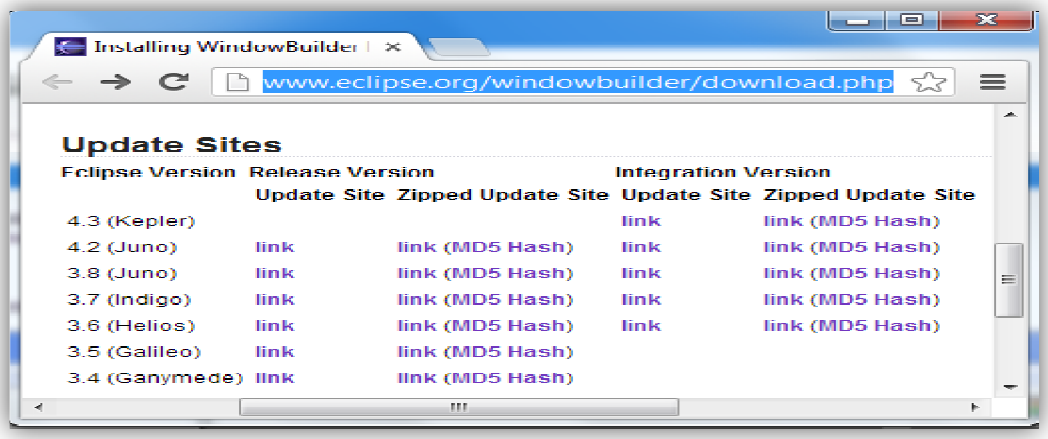


Figure 4.1: Installation WindowBuilder pro etap01.

Etape 2: Cliquez sur le lien de votre environnement de développement eclipse pour obtenir le lien d'installation. Ensuite, collez le lien sur votre eclipse dans le chemin suivant « *Help > Install New Software...> Work with* ».

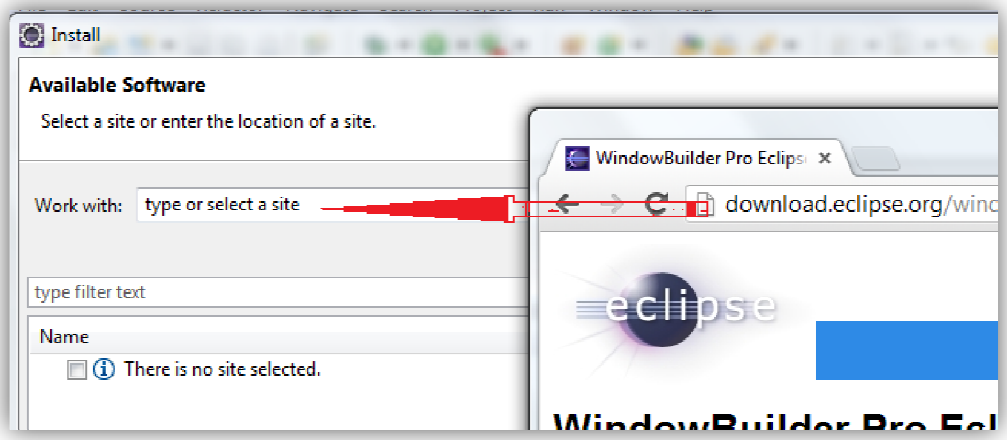


Figure 4.2: Installation WindowBuilder pro étape02.

Etape 3 : Dans la dernière étape choisissez les composants à installer puis cliquez sur « suivant » après vous suivez les autre étapes et vous attendez la terminaison de l'installation. Enfin, redémarrez votre eclipse. La figure suivante montre la nouvelle fenêtre d'eclipse avec le plug-in windowsbuilder pro.

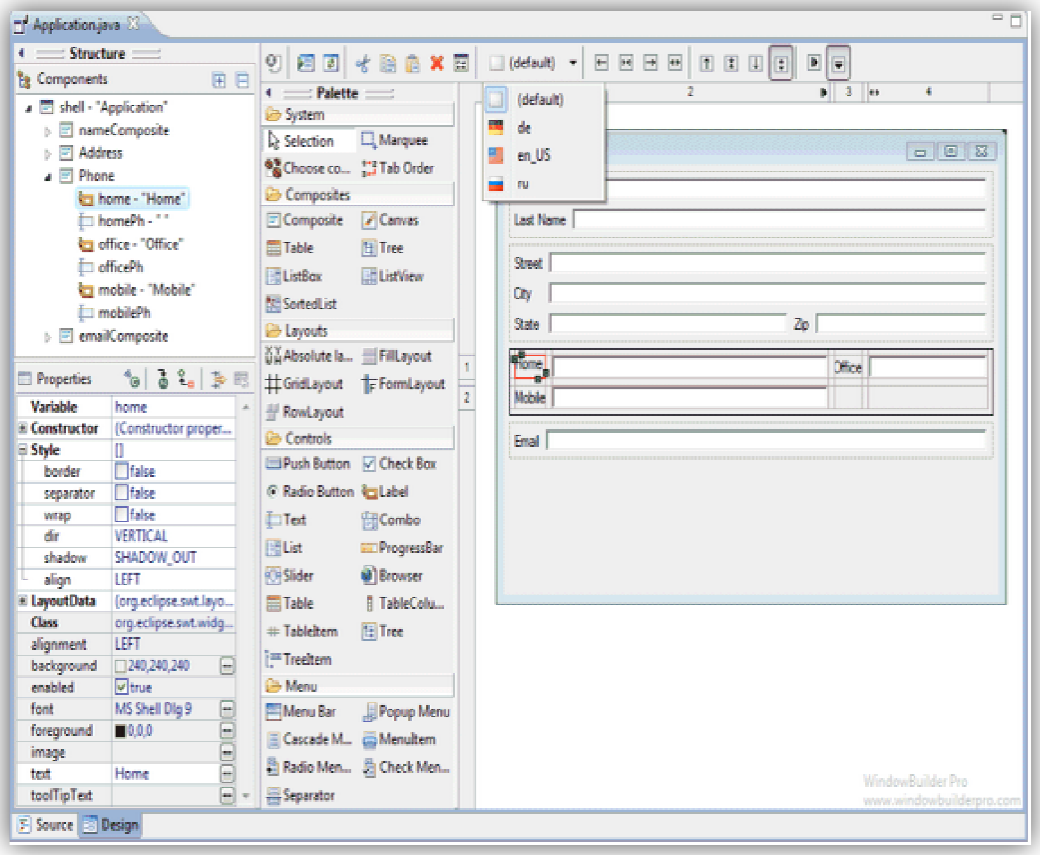


Figure 4.3: Le plug-in WindowBuilder pro.

3. Plates formes de développement des SMA.

Parmi les plates-formes fournies comme logiciels libres, il y a quelques plates-formes connues dans le développement de plusieurs applications : JADE, MADKIT, REPASTE, NETLOGO. Il faut noter que cette liste n'est pas unique, et qu'il y a aussi d'autres plates-formes (MAC, ZEUS, SWARM...) qui ont été utilisées aussi avec beaucoup de succès pour bâtir diverses applications.

3.1. JADE : (Java Agent Développement Framework).

JADE (Java Agent Développement Framework) est une plate-forme Java pour le développement des systèmes multi-agents. Elle respecte le standard FIPA. JADE a été développée par l'université de Parme et C-SELT centre de recherche télécom italien.

Le but de JADE est de simplifier le développement des systèmes multi-agents, il possède trois modules principaux nécessaires dans les normes FIPA. Ces modules sont lancés à chaque démarrage de la plate-forme :

- **DF** « Directory Facilitator » fournit un service de « pages jaunes » à la plate-forme.
- **ACC** « Agent Communication Channel » gère la communication entre les agents.
- **AMS** « Agent Management System » supervise l'enregistrement des agents, leur authentification, leur accès et l'utilisation du système.

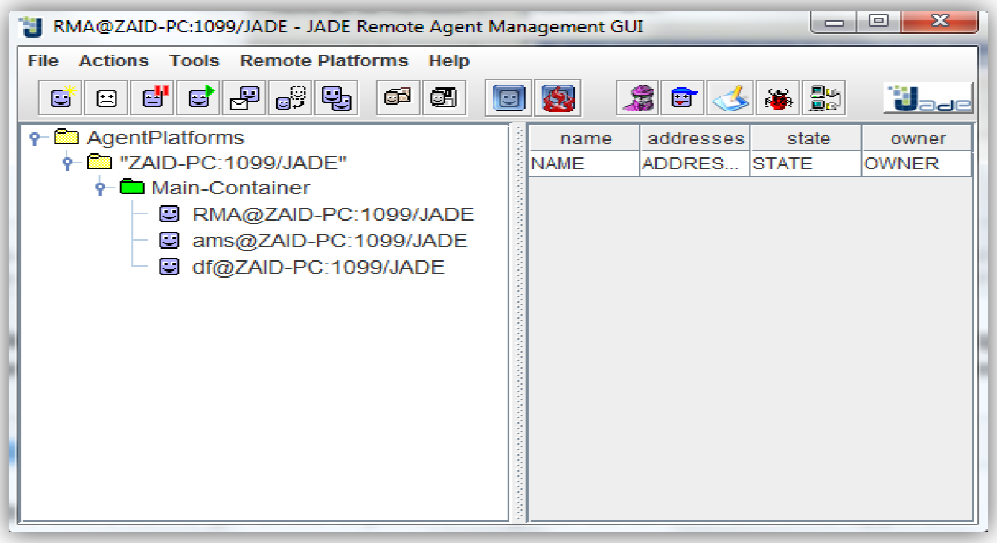


Figure 4.4 : Interface utilisateur de Jade (GUI).

3.2. MadKit : (Multi-Agents Développement Kit).

MadKit (Multi-Agents Développement Kit) est une plateforme multi-agents modulaire et scalable écrite en Java et conçue selon le modèle d'organisation AGR (Agent/Group/Role: des agents sont situés dans Des groupes et jouent des rôles). Elle est développée en 1996 par

Olivier GETKNECHT et Michel FERBER au Laboratoire d'Informatique de Robotique et de Microélectronique de Montpellier (LIRMM) de l'Université Montpellier II. Il s'agit d'une plate-forme libre. Un moteur d'exécution est utilisé dans MadKit où chaque agent est construit en partant d'un micronoyau. Chaque agent a un rôle et peut appartenir à un groupe.

MadKit est doté d'un environnement de développement graphique qui permet la construction facile des applications.

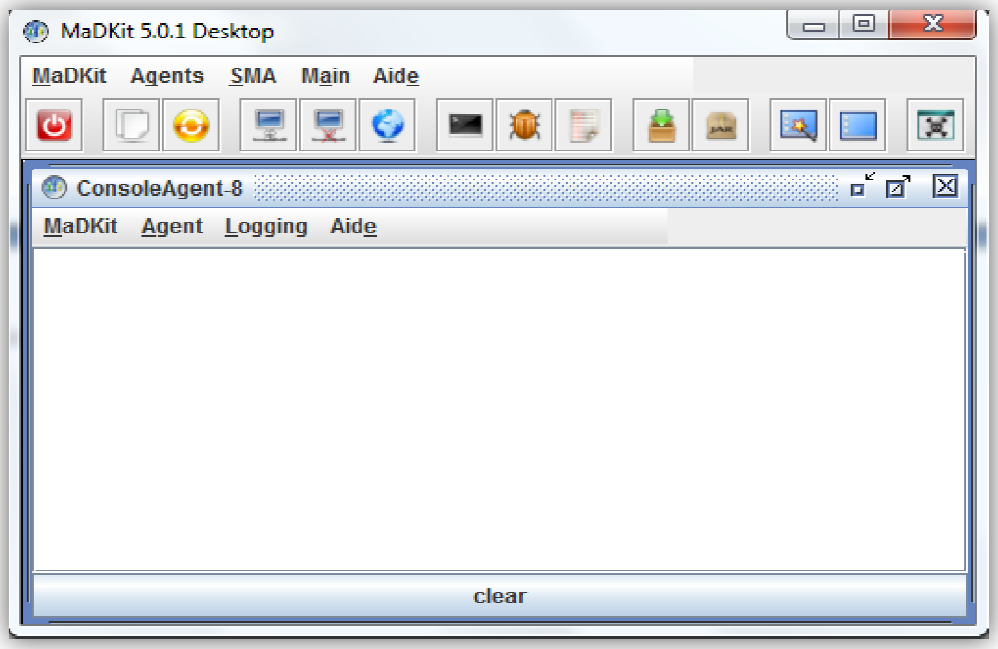


Figure 4.5 : Interface utilisateur MadKit.

3.3. NetLogo.

Historiquement NetLogo est une application autonome écrite en Java, elle peut donc fonctionner sur toutes les plateformes de calcul majeures. Avec 9 ans de développement (depuis 1999), NetLogo est un produit mature, stable et rapide. La bibliothèque des modèles de NetLogo a beaucoup de simulations que l'on peut explorer et modifier. Ces simulations abordent plusieurs domaines de contenu dans les sciences naturelles et sociales, y compris la biologie et la médicine, la physique et la chimie, l'économie et la psychologie sociale. NetLogo est un langage de programmation multi-agents et un environnement de la modélisation pour la simulation des phénomènes naturels et sociaux. Il est particulièrement bien adapté à la modélisation des systèmes complexes évoluant dans le temps. Les modeleurs peuvent attribuer des instructions à des centaines ou des milliers d'agents indépendants afin de faire simultanément des opérations. C'est pourquoi, on peut explorer les connections entre

les comportements de petit degré des individus et les modèles de grand degré qui émergent à partir de leurs interactions.

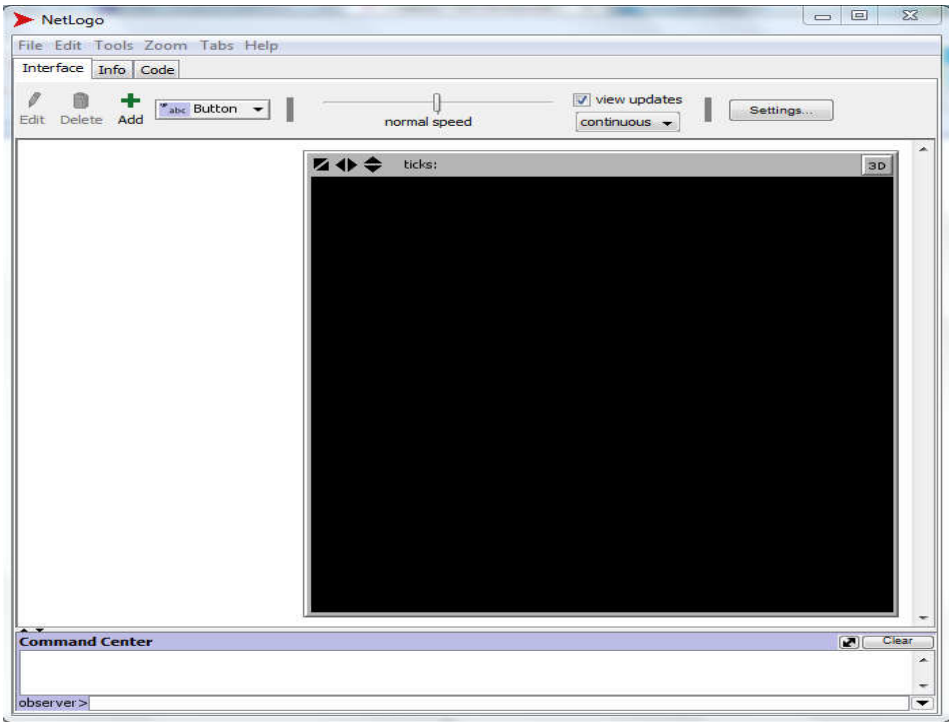


Figure 4.6 : Interface principale de NetLogo.

3.4. Repast.

Repast (Simulation Toolkit Agent poreux récursif) est un ensemble d'outils de simulation à base d'agents disponibles gratuitement spécifiquement conçu pour les applications en sciences sociales. Repast permet l'étude systématique des comportements de systèmes complexes à travers des expériences informatiques contrôlées et reproductibles. Développé à l'origine par David Sallach et d'autres chercheurs de l'Université de Chicago et l'Argonne National Laboratory,

Repast offre une collection de base de classes pour la création et l'exécution de simulations multi-agents et pour la collecte et l'affichage des données dans des tableaux, des diagrammes et des graphiques. Une caractéristique particulièrement intéressante de Repast est sa capacité à intégrer des données SIG (science de l'information géographique) directement dans les simulations.

4. Description détaillée de la plateforme JADE.

Dans le cadre de notre mémoire nous avons opté pour la plateforme JADE. Dans ce qui suit nous présentons les composants de cette plateforme suivie par une justification du choix de cette dernière pour le développement de notre application.

4.1. les principaux composants de Jade.

4.1.1. Agent RMA : (*Remote Management Agent).*

Le **RMA** permet de contrôler le cycle de vie de la plate-forme et. L'architecture répartie de JADE permet le contrôle à distance d'une autre plate-forme. Plusieurs RMA peuvent être lancés sur la même plate-forme du moment qu'ils ont des noms distincts.

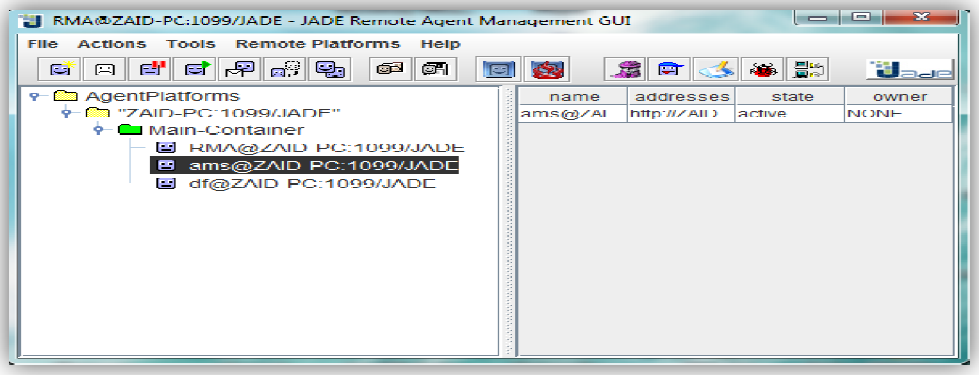


Figure 4.7 : l'interface de l'agent RMA.

4.1.2. Agent Dummy.

L'outil Dummy Agent permet aux utilisateurs d'interagir avec les agents JADE d'une façon particulière. L'interface permet la composition et l'envoi de messages ACL et maintient une liste de messages ACL envoyés et reçus. Cette liste peut être examinée par l'utilisateur et chaque message peut être vu en détail ou même édité. Plus encore, le message peut être sauvegardé sur le disque et renvoyé plus tard.

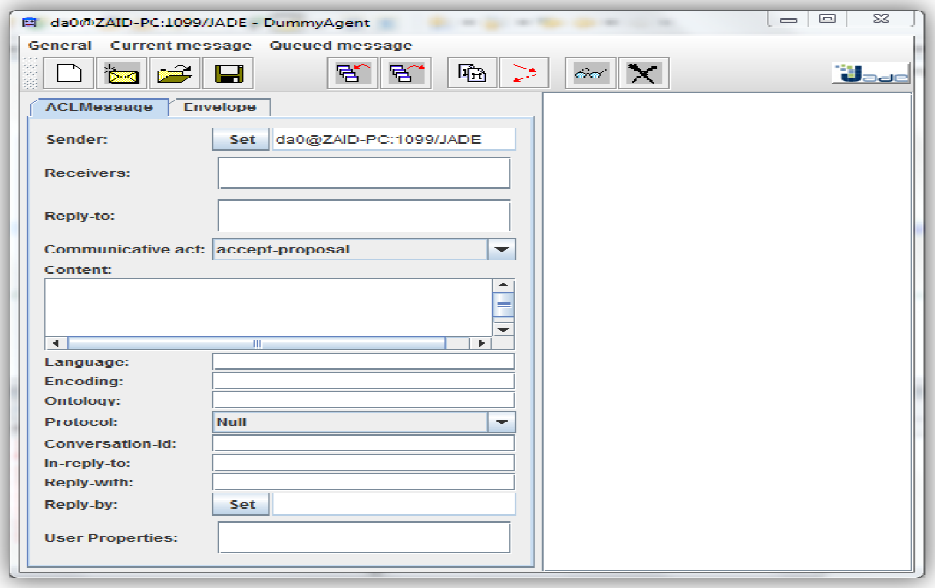


Figure 4.8 : l'interface Dummy Agent.

4.1.3. Agent Directory Facilitator.

L'interface du DF peut être lancée à partir du menu du RMA .Cette action est en fait implantée par l'envoi d'un message ACL au DF lui demandant de charger son interface graphique. L'interface peut être juste vue sur l'hôte où la plate-forme est exécutée. En utilisant cette interface, l'utilisateur peut interagir avec le DF.

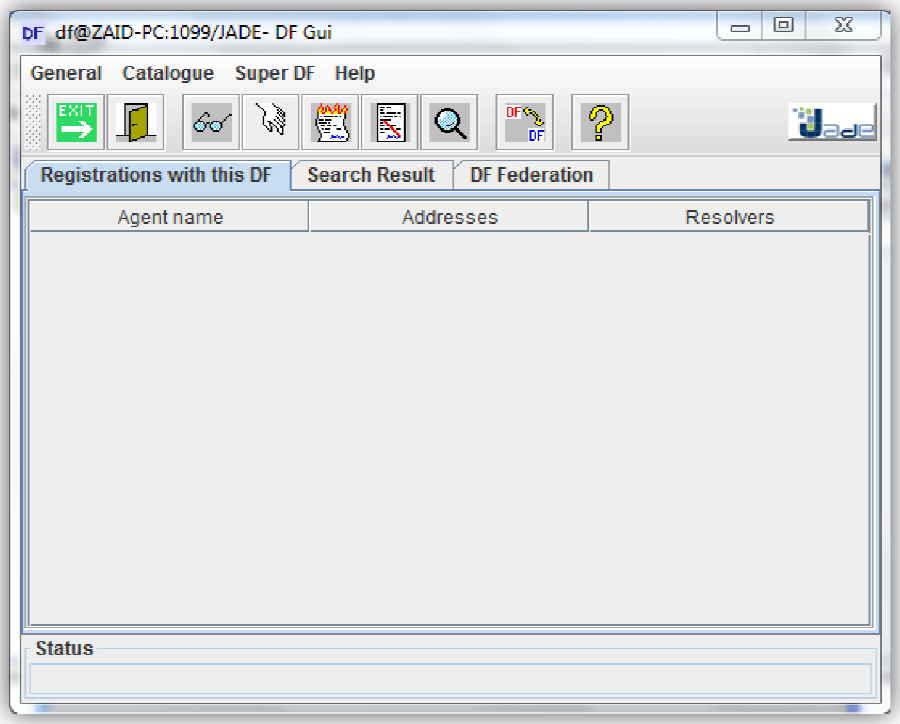


Figure 4.9 : l'interface d'Agent Directory Facilitator (DF).

4.1.4. Agent Sniffer.

Quand un utilisateur décide d'épier un agent ou un groupe d'agents, il utilise un agent sniffer. Chaque message partant ou allant vers ce groupe est capté et affiché sur l'interface du sniffer. L'utilisateur peut voir et enregistrer tous les messages, pour éventuellement les analyser plus tard. L'agent peut être lancé du menu du RMA ou de la ligne de commande suivante : *Java jade.Boot sniffer:jade.tools.sniffer.sniffer.*

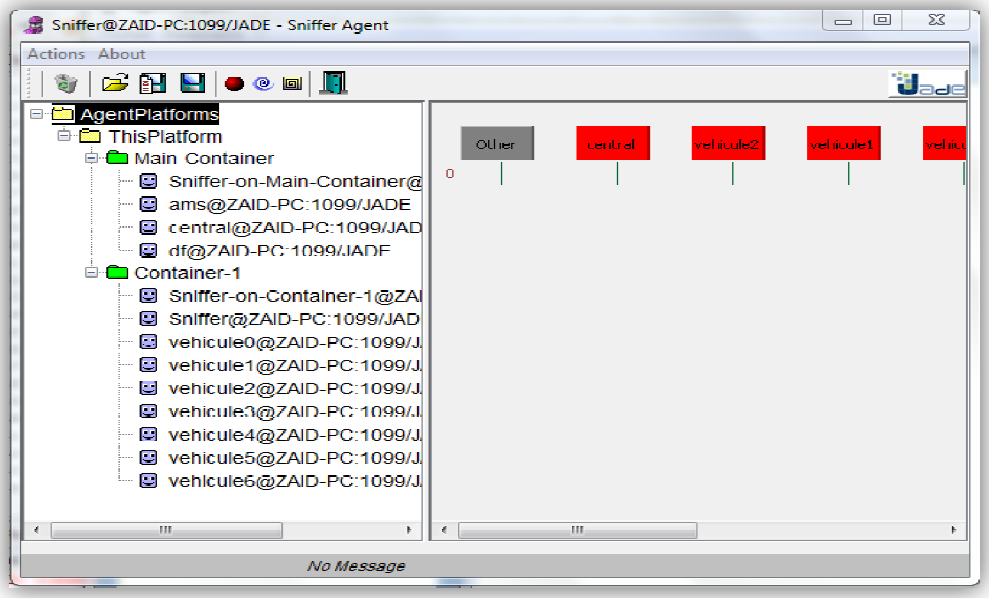


Figure 4.10 : L'interface de l'agent Sniffer.

4.1.5. Agent Inspector.

Cet agent permet de gérer et de contrôler le cycle de vie d'un agent s'exécutant et la file de ses messages envoyés et reçus.

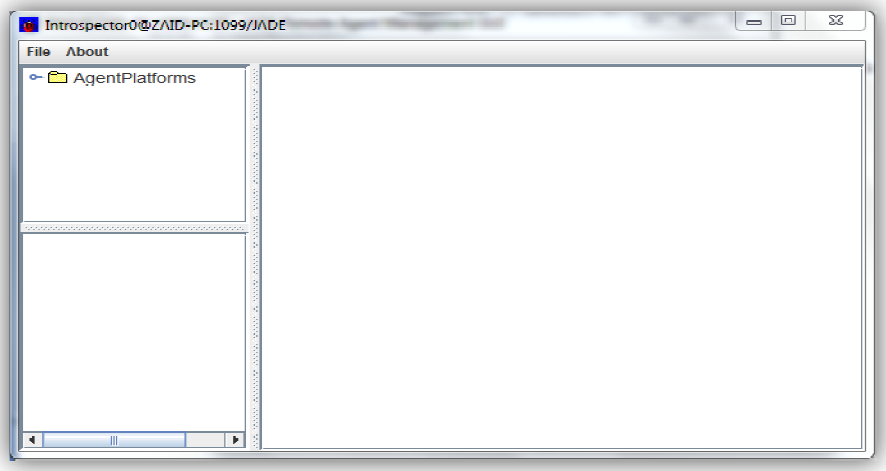


Figure 4.11 : L'interface de l'agent Inspector.

4.2. Pourquoi la plateforme JADE ?

Nous justifions notre choix de la plateforme JADE par les points suivants :

- ***JADE est une plateforme compatible FIPA***, contrairement à la majorité des plateformes qui ne proposent pas d'outils standards de développement.
- ***JADE est une plateforme qui peut être distribuée*** sur plusieurs machines, à condition qu'il n'y ait pas de pare-feu entre ces machines. Une seule application Java

(Machine Virtuelle Java) est exécutée sur chaque machine. Les agents sont implémentés comme des threads d'exécution Java et les événements Java sont utilisés pour la communication efficace et légère entre agents sur une même machine.

- ***JADE est une plateforme qui gère les différents principes d'interaction*** car elle permet d'implémenter de manière efficace les protocoles d'interaction entre les agents. Ceci est très important dans notre cas pour implémenter les différents protocoles de négociation modélisés par UML dans le chapitre 3 (diagramme de séquence). A titre d'exemple, les classes prédéfinies Propose *Initiator* et Propose *Responder* de *JADE* permettent d'instancier des objets behaviours qui implémentent le protocole d'interaction *FIPA-Propose*. *JADE* fournit d'autres classes de comportements prêts pour implémenter les protocoles d'interactions FIPA comme *FIPA Propose Protocol*, *FIPA Request Protocol*, *FIPA Contract Net Protocol*, *FIPA English Auction Protocol*. Ces classes se trouvent dans le package jade.proto.
- ***JADE est une plateforme qui gère l'identification des agents*** car elle contient un service d'attribution de noms compatible FIPA ; quand on lance la plate-forme, un agent obtient un identificateur unique (*Globally Unique Identifier - GUID*). Cette tâche facilite la création des agents de notre architecture d'une manière transparente car le nombre de ces derniers est variant selon les paramètres des simulations. [19]
- ***La notion de comportement nous permet de générer rapidement le squelette*** de code d'un agent *JADE* à partir du diagramme d'états-transitions qui modélise son comportement.

5. Jeu de données.

Pour tester le fonctionnement globale de notre architecture, noua avons optés pour des fichiers texte. Chaque fichier à la structure suivante :

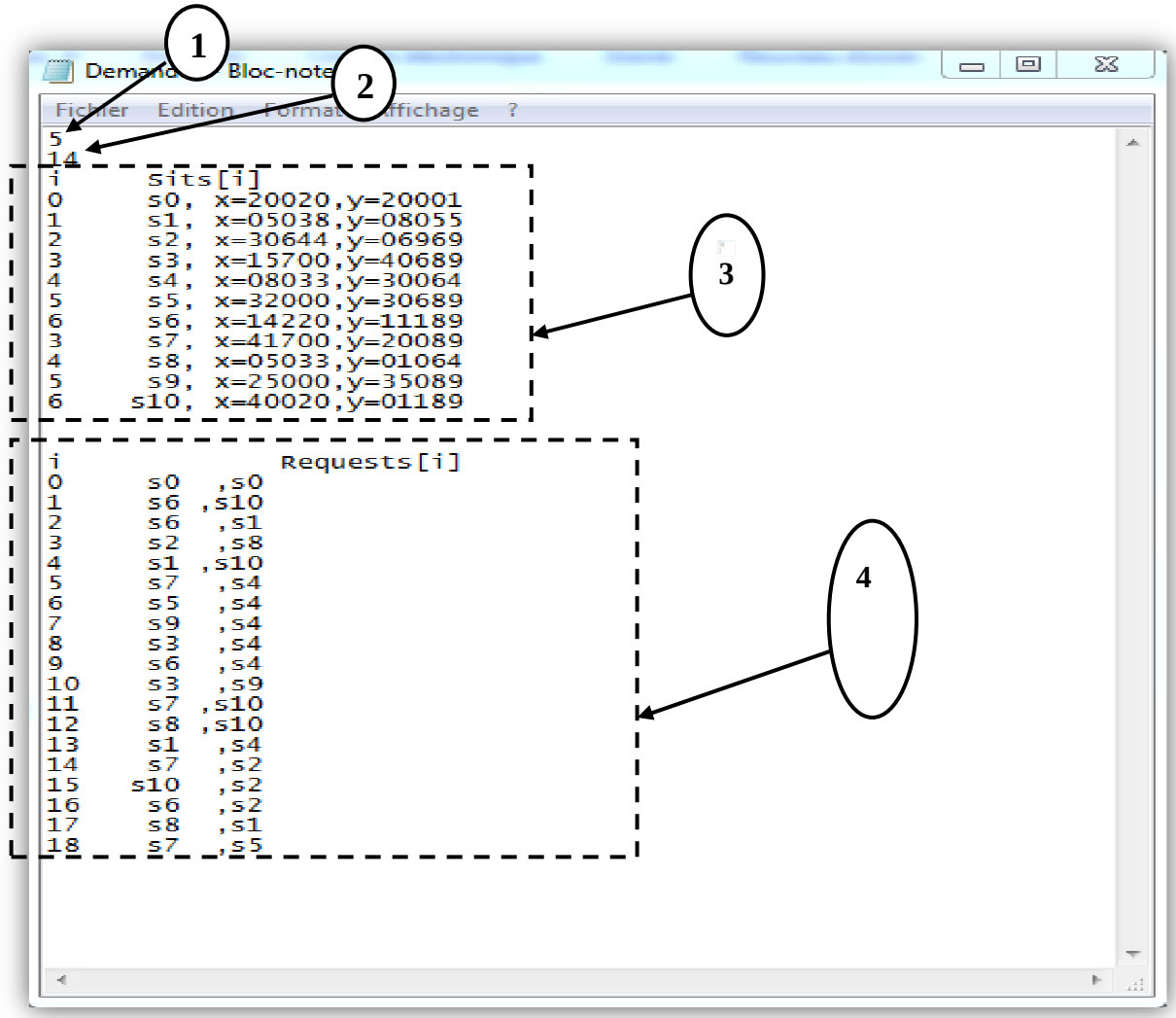


Figure 4.12: Structure d'une instance de demande.

1 : représente le nombre de site.

2 : représente le nombre des patients.

La zone 3 : représente les caractéristiques de chaque site, qui sont : l'identifiant du site, ses coordonnées (x, y).

La zone 4 : représente la suite des requêtes, chacune est caractérisée par : le numéro de requête, le site d'habitation, le site de dialyse.

6. Description des interfaces de l'application.

Dans cette partie nous allons présenter les différentes interfaces graphiques de notre application et nous commençons par l'interface principale suivante :

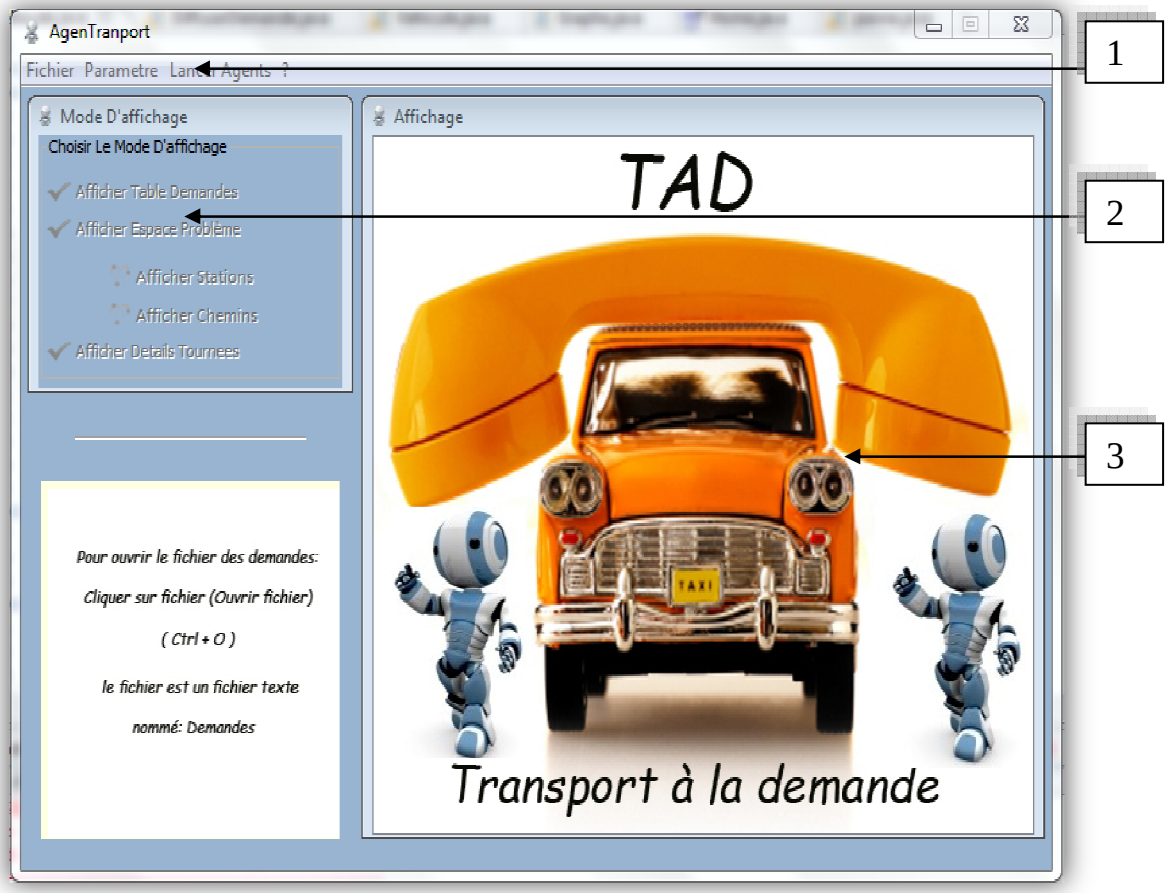


Figure 4.13 : L'interface de l'application**.**

1 **Barre de Menu :** qui contient trois menu comme suit :

1) **Fichier :** contient les opérations permettant de charger les données et de quitter L'application.
2) **Paramètre :** ce menu contient 2 opérations :

- **Paramètre Tabou :** configuration des paramètres concernant le véhicule et les paramètres de la recherche Tabou.

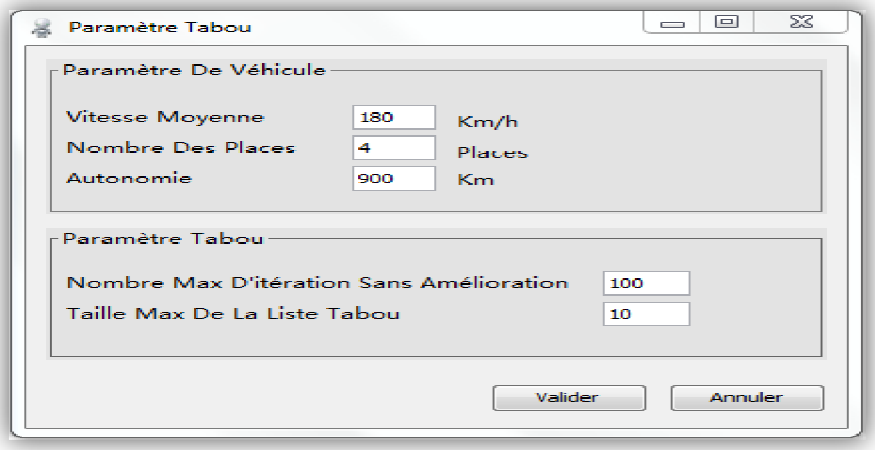


Figure 4.14 : Paramètre Tabou.

- **Etablir un cas de panne :**

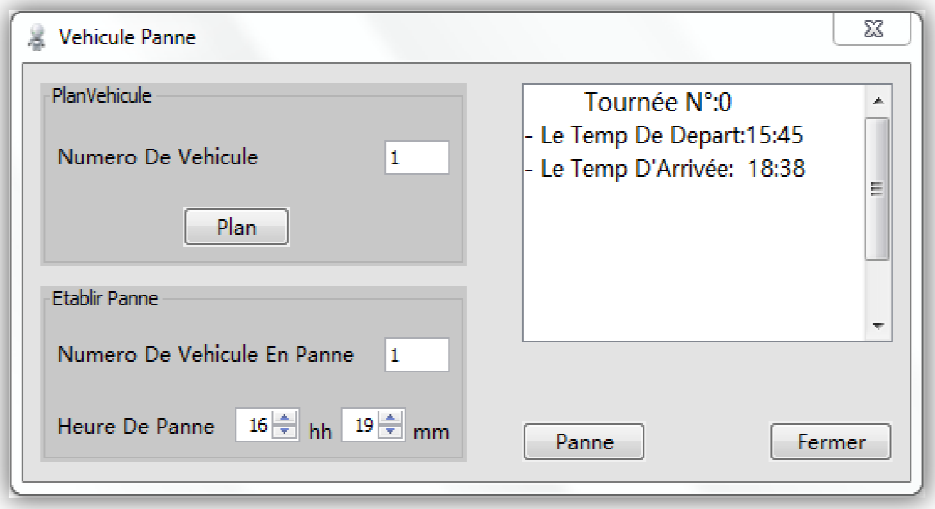


Figure 4.15 : Paramètre de panne.

3) **Lancer agents :** ce menu contient 2 opérations.

- **Lancer l'agent central** : pour exécuter a recherche tabou.

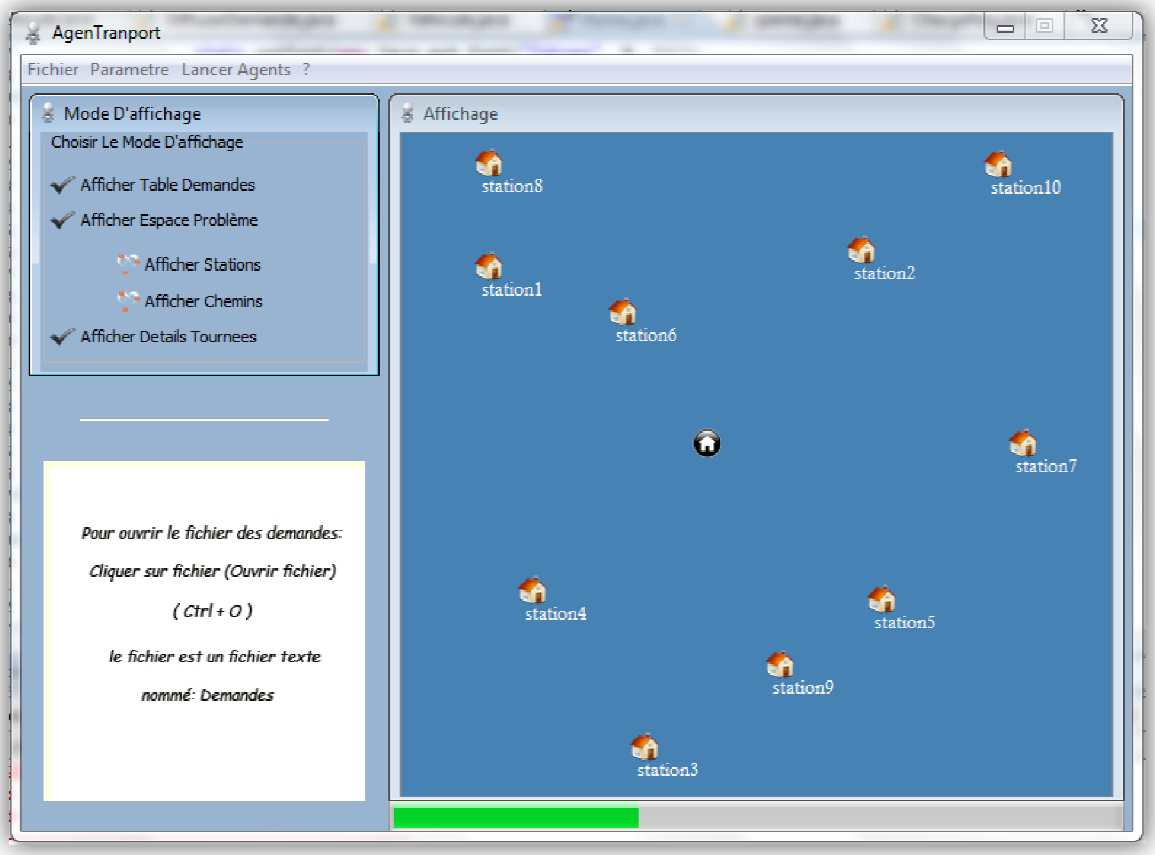


Figure 4.16 : L'Agent Central.

- **Lancer les agents véhicules : (Agent sniffer avant le cas de panne).**

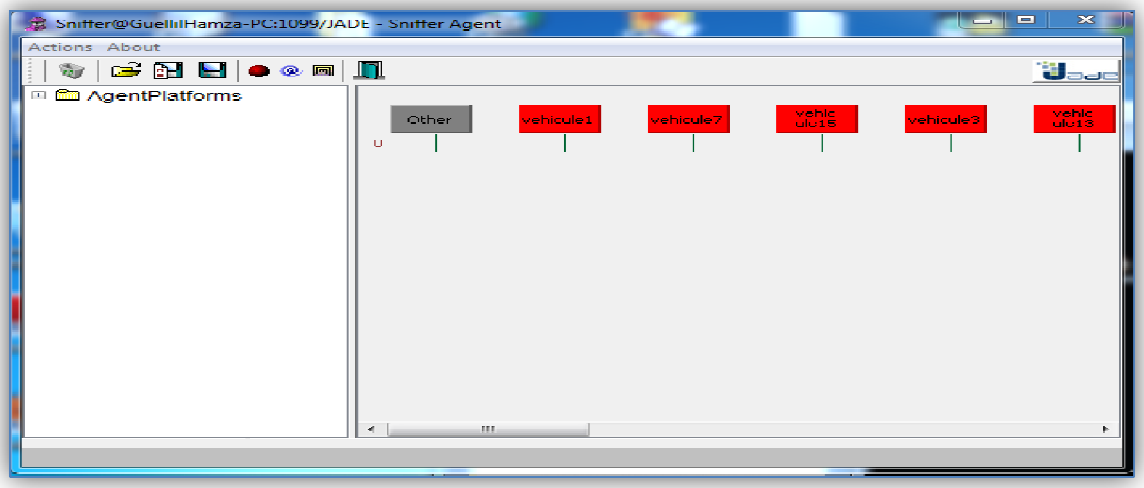


Figure 4.17 : L'Agent Sniffer avant la panne.

Chapitre IV

Implémentation et Tests

1. Introduction.

Dans ce dernier chapitre nous allons présenter les différents outils de programmation utilisés dans le développement de notre application et nous donnons aussi une description des différents composants de cette dernière.

Le reste de chapitre est organisé en trois parties, la première partie concerne le langage de programmation, l'environnement de développement et les différentes installations nécessaires. Dans la deuxième partie, nous présentons les plateformes les plus utilisées dans le développement des systèmes multi-agent avec une justification de notre choix concernant la plateforme JADE. Enfin, nous donnons une description des interfaces graphiques de l'application.

2. Environnement de développement.

Pour le développement de notre application nous avons utilisé les outils suivants :

- **Machine :** Nous avons utilisé un ordinateur caractérisées par :
  - ✓ Processeur : Intel (R) Core(TM) i5-2450 CPU @ 2.50GHz 2.50GHz.
  - ✓ Mémoire installée (RAM) : 4,00 Go.
  - ✓ Système d'exploitation : Windows 7 Service Pack 01.
- **Logiciels :** Nous avons utilisé java comme langage de programmation et Eclipse Indigo (*Indigo Service Release 2*) comme environnement de développement intégré (IDE).Nous avons opté aussi pour *WindowBuilder Pro* **un plug-in** pour Eclipse, venant de Google, il permet d'éditer des interfaces graphiques directement dans l'IDE. Ce plug-in permet aux développeurs d'éditer des interfaces graphiques avec moins d'efforts et dans un temps réduit. Dans ce qui suit nous présentons les étapes d'installation de ce plug-in :

Etape 1 : Visitez ce lien : http://www.eclipse.org/windowbuilder/download.php

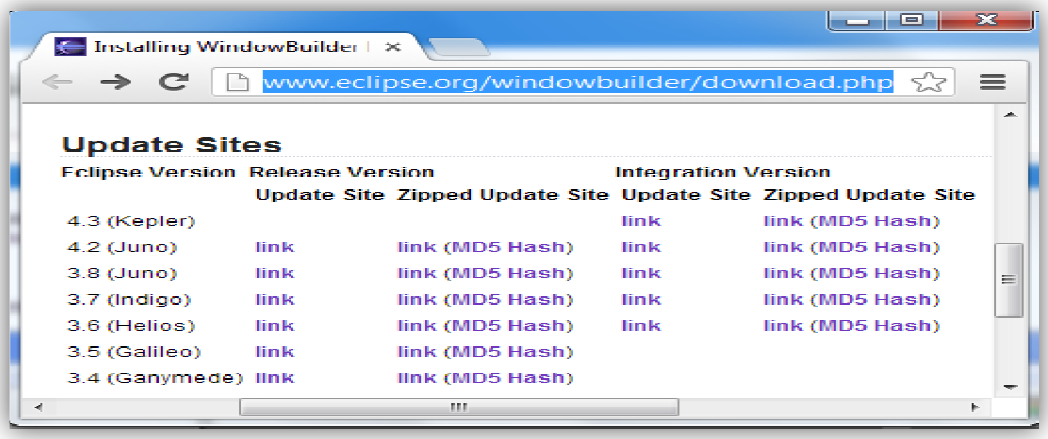


Figure 4.1: Installation WindowBuilder pro etap01.

Etape 2: Cliquez sur le lien de votre environnement de développement eclipse pour obtenir le lien d'installation. Ensuite, collez le lien sur votre eclipse dans le chemin suivant « *Help > Install New Software...> Work with* ».

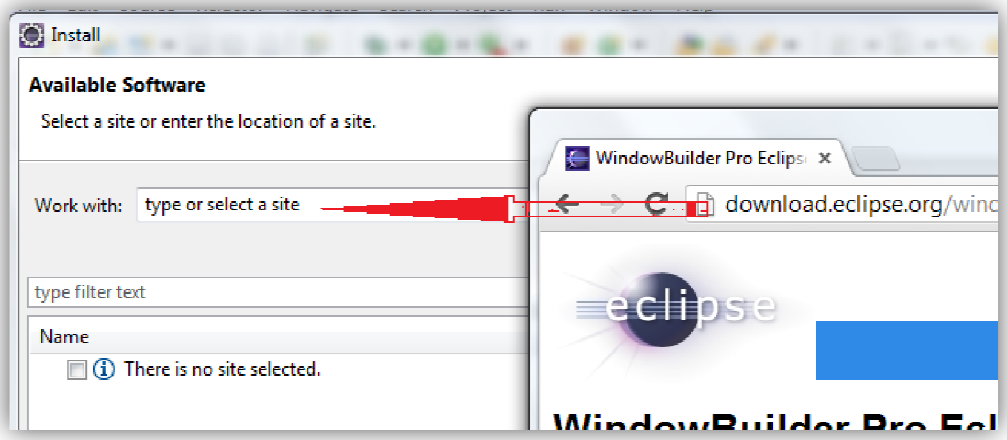


Figure 4.2: Installation WindowBuilder pro étape02.

Etape 3 : Dans la dernière étape choisissez les composants à installer puis cliquez sur « suivant » après vous suivez les autre étapes et vous attendez la terminaison de l'installation. Enfin, redémarrez votre eclipse. La figure suivante montre la nouvelle fenêtre d'eclipse avec le plug-in windowsbuilder pro.

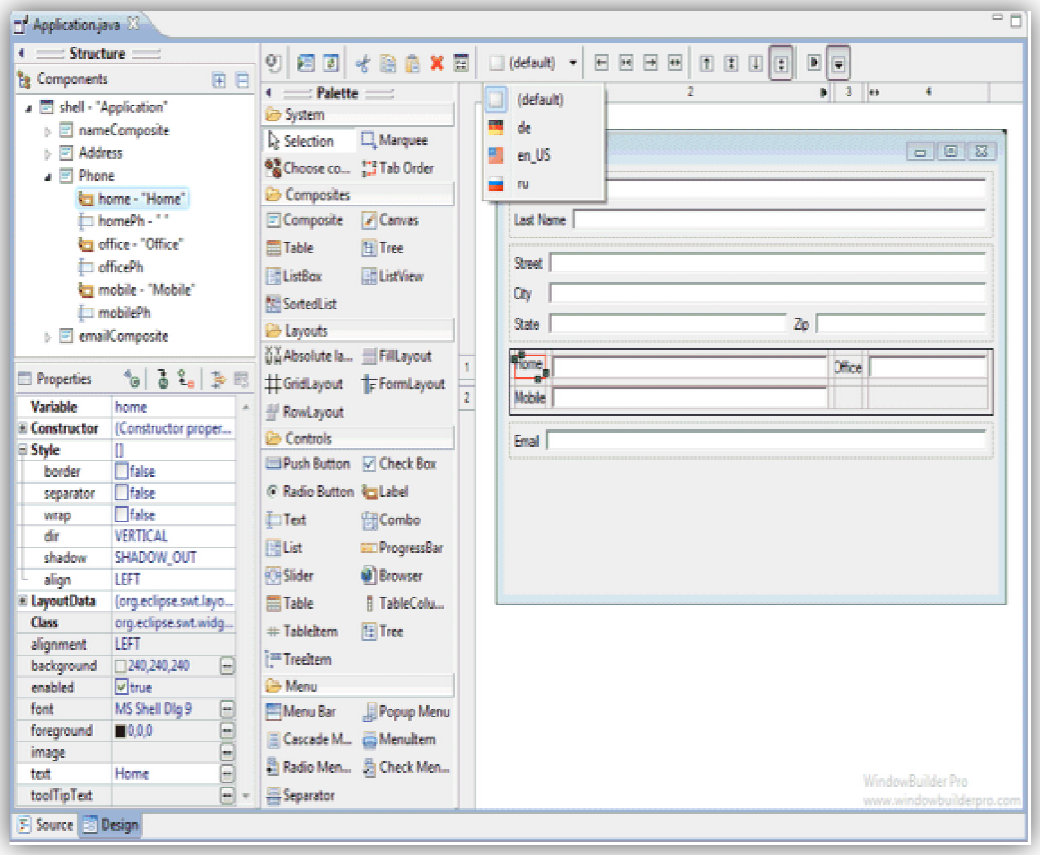


Figure 4.3: Le plug-in WindowBuilder pro.

3. Plates formes de développement des SMA.

Parmi les plates-formes fournies comme logiciels libres, il y a quelques plates-formes connues dans le développement de plusieurs applications : JADE, MADKIT, REPASTE, NETLOGO. Il faut noter que cette liste n'est pas unique, et qu'il y a aussi d'autres plates-formes (MAC, ZEUS, SWARM...) qui ont été utilisées aussi avec beaucoup de succès pour bâtir diverses applications.

3.1. JADE : (Java Agent Développement Framework).

JADE (Java Agent Développement Framework) est une plate-forme Java pour le développement des systèmes multi-agents. Elle respecte le standard FIPA. JADE a été développée par l'université de Parme et C-SELT centre de recherche télécom italien.

Le but de JADE est de simplifier le développement des systèmes multi-agents, il possède trois modules principaux nécessaires dans les normes FIPA. Ces modules sont lancés à chaque démarrage de la plate-forme :

- **DF** « Directory Facilitator » fournit un service de « pages jaunes » à la plate-forme.
- **ACC** « Agent Communication Channel » gère la communication entre les agents.
- **AMS** « Agent Management System » supervise l'enregistrement des agents, leur authentification, leur accès et l'utilisation du système.

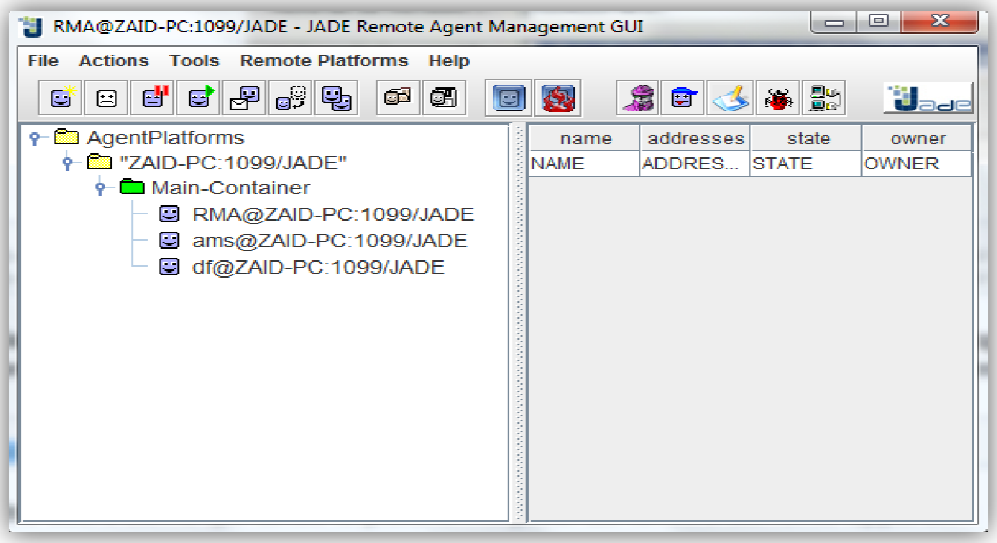


Figure 4.4 : Interface utilisateur de Jade (GUI).

3.2. MadKit : (Multi-Agents Développement Kit).

MadKit (Multi-Agents Développement Kit) est une plateforme multi-agents modulaire et scalable écrite en Java et conçue selon le modèle d'organisation AGR (Agent/Group/Role: des agents sont situés dans Des groupes et jouent des rôles). Elle est développée en 1996 par

Olivier GETKNECHT et Michel FERBER au Laboratoire d'Informatique de Robotique et de Microélectronique de Montpellier (LIRMM) de l'Université Montpellier II. Il s'agit d'une plate-forme libre. Un moteur d'exécution est utilisé dans MadKit où chaque agent est construit en partant d'un micronoyau. Chaque agent a un rôle et peut appartenir à un groupe.

MadKit est doté d'un environnement de développement graphique qui permet la construction facile des applications.

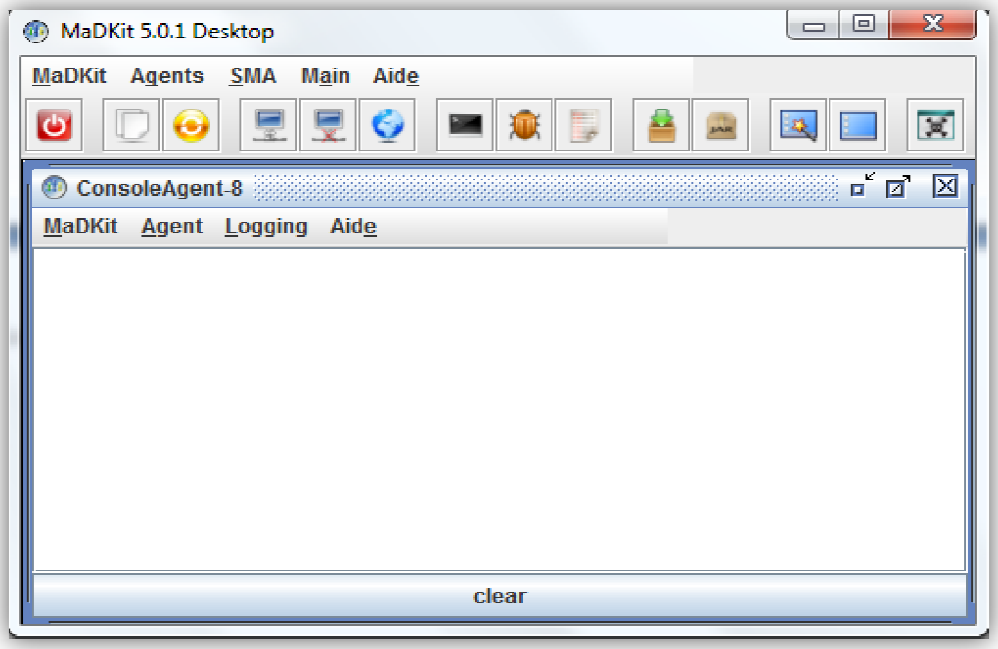


Figure 4.5 : Interface utilisateur MadKit.

3.3. NetLogo.

Historiquement NetLogo est une application autonome écrite en Java, elle peut donc fonctionner sur toutes les plateformes de calcul majeures. Avec 9 ans de développement (depuis 1999), NetLogo est un produit mature, stable et rapide. La bibliothèque des modèles de NetLogo a beaucoup de simulations que l'on peut explorer et modifier. Ces simulations abordent plusieurs domaines de contenu dans les sciences naturelles et sociales, y compris la biologie et la médicine, la physique et la chimie, l'économie et la psychologie sociale. NetLogo est un langage de programmation multi-agents et un environnement de la modélisation pour la simulation des phénomènes naturels et sociaux. Il est particulièrement bien adapté à la modélisation des systèmes complexes évoluant dans le temps. Les modeleurs peuvent attribuer des instructions à des centaines ou des milliers d'agents indépendants afin de faire simultanément des opérations. C'est pourquoi, on peut explorer les connections entre

les comportements de petit degré des individus et les modèles de grand degré qui émergent à partir de leurs interactions.

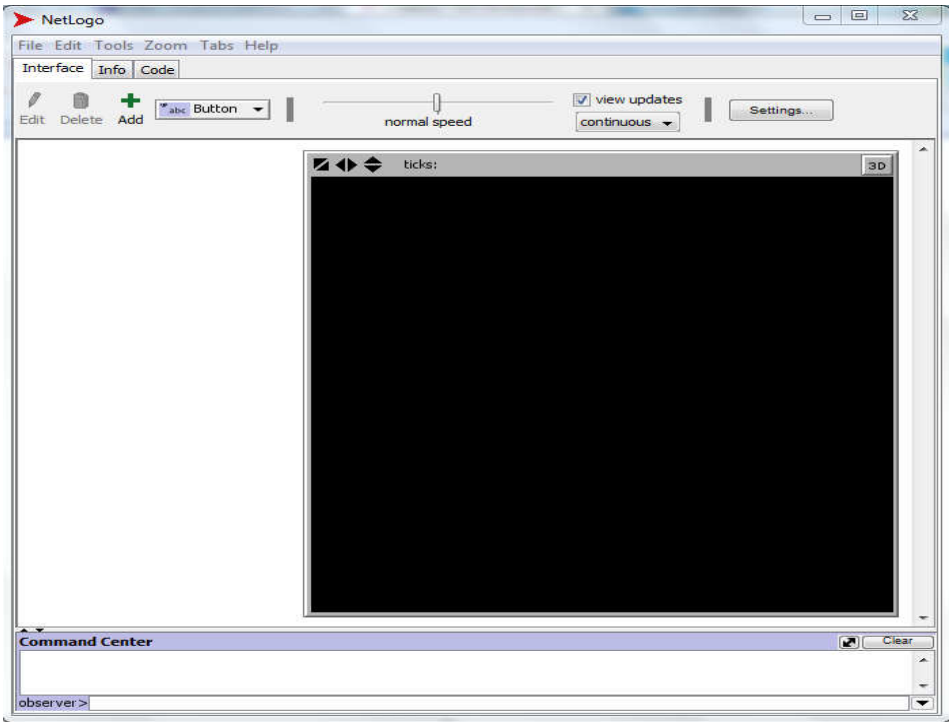


Figure 4.6 : Interface principale de NetLogo.

3.4. Repast.

Repast (Simulation Toolkit Agent poreux récursif) est un ensemble d'outils de simulation à base d'agents disponibles gratuitement spécifiquement conçu pour les applications en sciences sociales. Repast permet l'étude systématique des comportements de systèmes complexes à travers des expériences informatiques contrôlées et reproductibles. Développé à l'origine par David Sallach et d'autres chercheurs de l'Université de Chicago et l'Argonne National Laboratory,

Repast offre une collection de base de classes pour la création et l'exécution de simulations multi-agents et pour la collecte et l'affichage des données dans des tableaux, des diagrammes et des graphiques. Une caractéristique particulièrement intéressante de Repast est sa capacité à intégrer des données SIG (science de l'information géographique) directement dans les simulations.

4. Description détaillée de la plateforme JADE.

Dans le cadre de notre mémoire nous avons opté pour la plateforme JADE. Dans ce qui suit nous présentons les composants de cette plateforme suivie par une justification du choix de cette dernière pour le développement de notre application.

4.1. les principaux composants de Jade.

4.1.1. Agent RMA : (*Remote Management Agent).*

Le **RMA** permet de contrôler le cycle de vie de la plate-forme et. L'architecture répartie de JADE permet le contrôle à distance d'une autre plate-forme. Plusieurs RMA peuvent être lancés sur la même plate-forme du moment qu'ils ont des noms distincts.

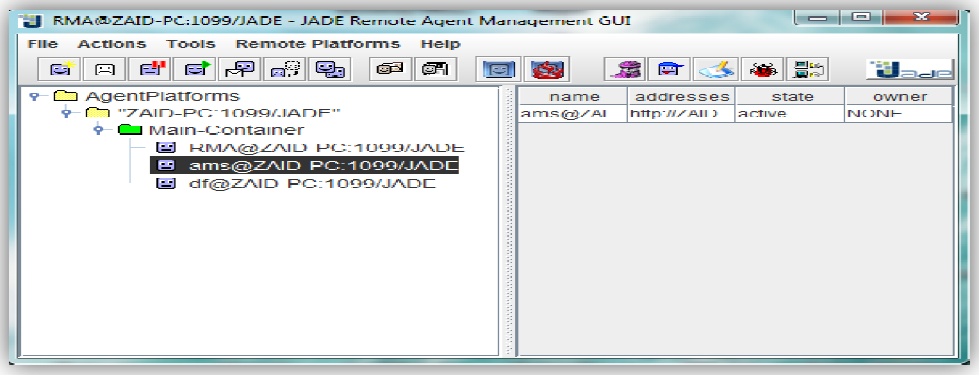


Figure 4.7 : l'interface de l'agent RMA.

4.1.2. Agent Dummy.

L'outil Dummy Agent permet aux utilisateurs d'interagir avec les agents JADE d'une façon particulière. L'interface permet la composition et l'envoi de messages ACL et maintient une liste de messages ACL envoyés et reçus. Cette liste peut être examinée par l'utilisateur et chaque message peut être vu en détail ou même édité. Plus encore, le message peut être sauvegardé sur le disque et renvoyé plus tard.

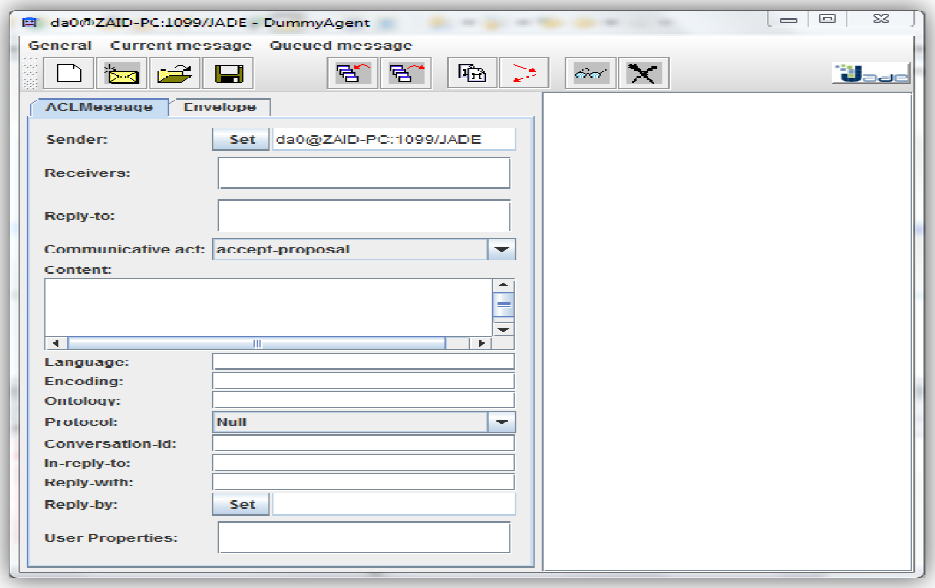


Figure 4.8 : l'interface Dummy Agent.

4.1.3. Agent Directory Facilitator.

L'interface du DF peut être lancée à partir du menu du RMA .Cette action est en fait implantée par l'envoi d'un message ACL au DF lui demandant de charger son interface graphique. L'interface peut être juste vue sur l'hôte où la plate-forme est exécutée. En utilisant cette interface, l'utilisateur peut interagir avec le DF.

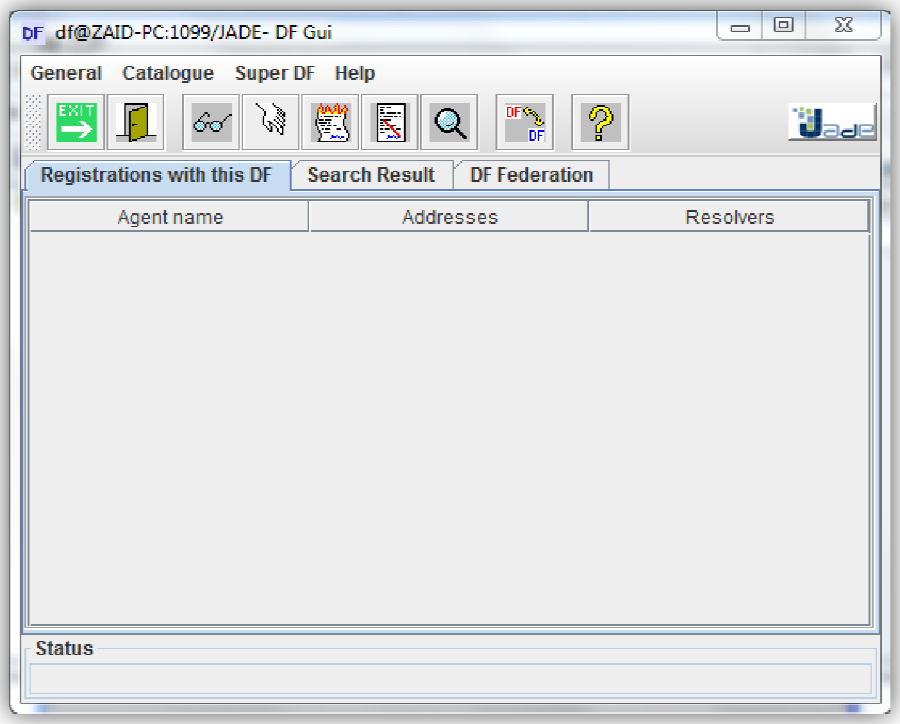


Figure 4.9 : l'interface d'Agent Directory Facilitator (DF).

4.1.4. Agent Sniffer.

Quand un utilisateur décide d'épier un agent ou un groupe d'agents, il utilise un agent sniffer. Chaque message partant ou allant vers ce groupe est capté et affiché sur l'interface du sniffer. L'utilisateur peut voir et enregistrer tous les messages, pour éventuellement les analyser plus tard. L'agent peut être lancé du menu du RMA ou de la ligne de commande suivante : *Java jade.Boot sniffer:jade.tools.sniffer.sniffer.*

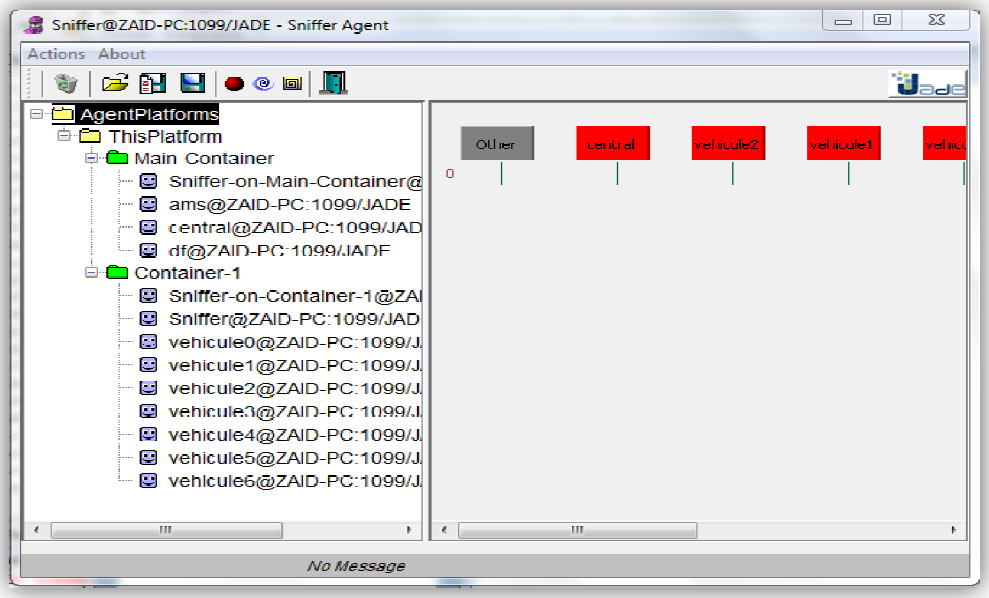


Figure 4.10 : L'interface de l'agent Sniffer.

4.1.5. Agent Inspector.

Cet agent permet de gérer et de contrôler le cycle de vie d'un agent s'exécutant et la file de ses messages envoyés et reçus.

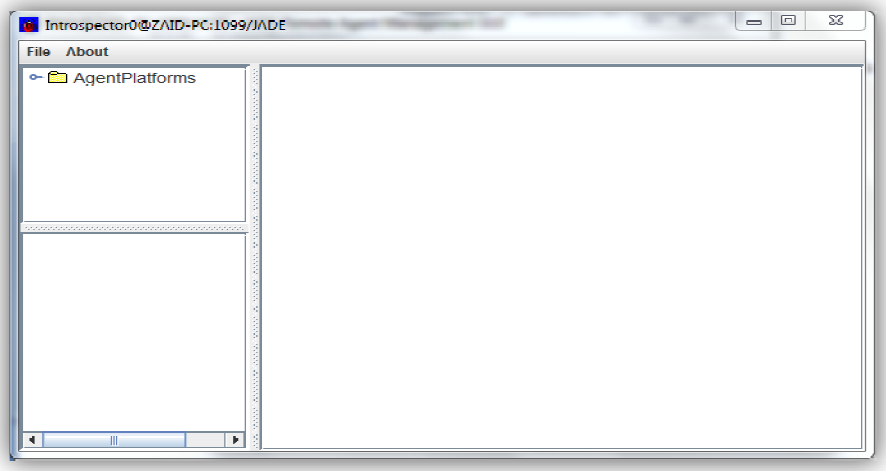


Figure 4.11 : L'interface de l'agent Inspector.

4.2. Pourquoi la plateforme JADE ?

Nous justifions notre choix de la plateforme JADE par les points suivants :

- ***JADE est une plateforme compatible FIPA***, contrairement à la majorité des plateformes qui ne proposent pas d'outils standards de développement.
- ***JADE est une plateforme qui peut être distribuée*** sur plusieurs machines, à condition qu'il n'y ait pas de pare-feu entre ces machines. Une seule application Java

(Machine Virtuelle Java) est exécutée sur chaque machine. Les agents sont implémentés comme des threads d'exécution Java et les événements Java sont utilisés pour la communication efficace et légère entre agents sur une même machine.

- ***JADE est une plateforme qui gère les différents principes d'interaction*** car elle permet d'implémenter de manière efficace les protocoles d'interaction entre les agents. Ceci est très important dans notre cas pour implémenter les différents protocoles de négociation modélisés par UML dans le chapitre 3 (diagramme de séquence). A titre d'exemple, les classes prédéfinies Propose *Initiator* et Propose *Responder* de *JADE* permettent d'instancier des objets behaviours qui implémentent le protocole d'interaction *FIPA-Propose*. *JADE* fournit d'autres classes de comportements prêts pour implémenter les protocoles d'interactions FIPA comme *FIPA Propose Protocol*, *FIPA Request Protocol*, *FIPA Contract Net Protocol*, *FIPA English Auction Protocol*. Ces classes se trouvent dans le package jade.proto.
- ***JADE est une plateforme qui gère l'identification des agents*** car elle contient un service d'attribution de noms compatible FIPA ; quand on lance la plate-forme, un agent obtient un identificateur unique (*Globally Unique Identifier - GUID*). Cette tâche facilite la création des agents de notre architecture d'une manière transparente car le nombre de ces derniers est variant selon les paramètres des simulations. [19]
- ***La notion de comportement nous permet de générer rapidement le squelette*** de code d'un agent *JADE* à partir du diagramme d'états-transitions qui modélise son comportement.

5. Jeu de données.

Pour tester le fonctionnement globale de notre architecture, noua avons optés pour des fichiers texte. Chaque fichier à la structure suivante :

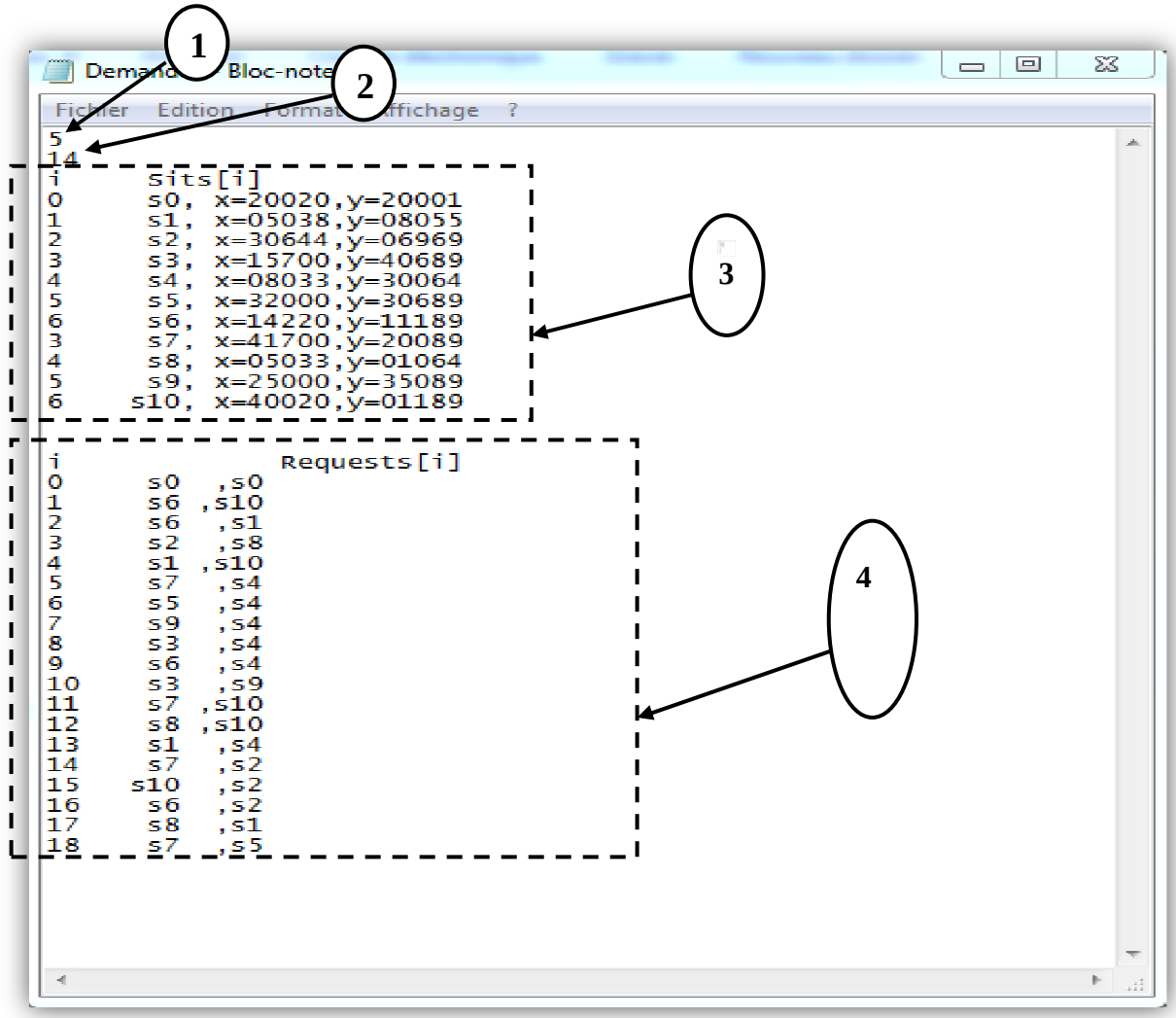


Figure 4.12: Structure d'une instance de demande.

1 : représente le nombre de site.

2 : représente le nombre des patients.

La zone 3 : représente les caractéristiques de chaque site, qui sont : l'identifiant du site, ses coordonnées (x, y).

La zone 4 : représente la suite des requêtes, chacune est caractérisée par : le numéro de requête, le site d'habitation, le site de dialyse.

6. Description des interfaces de l'application.

Dans cette partie nous allons présenter les différentes interfaces graphiques de notre application et nous commençons par l'interface principale suivante :

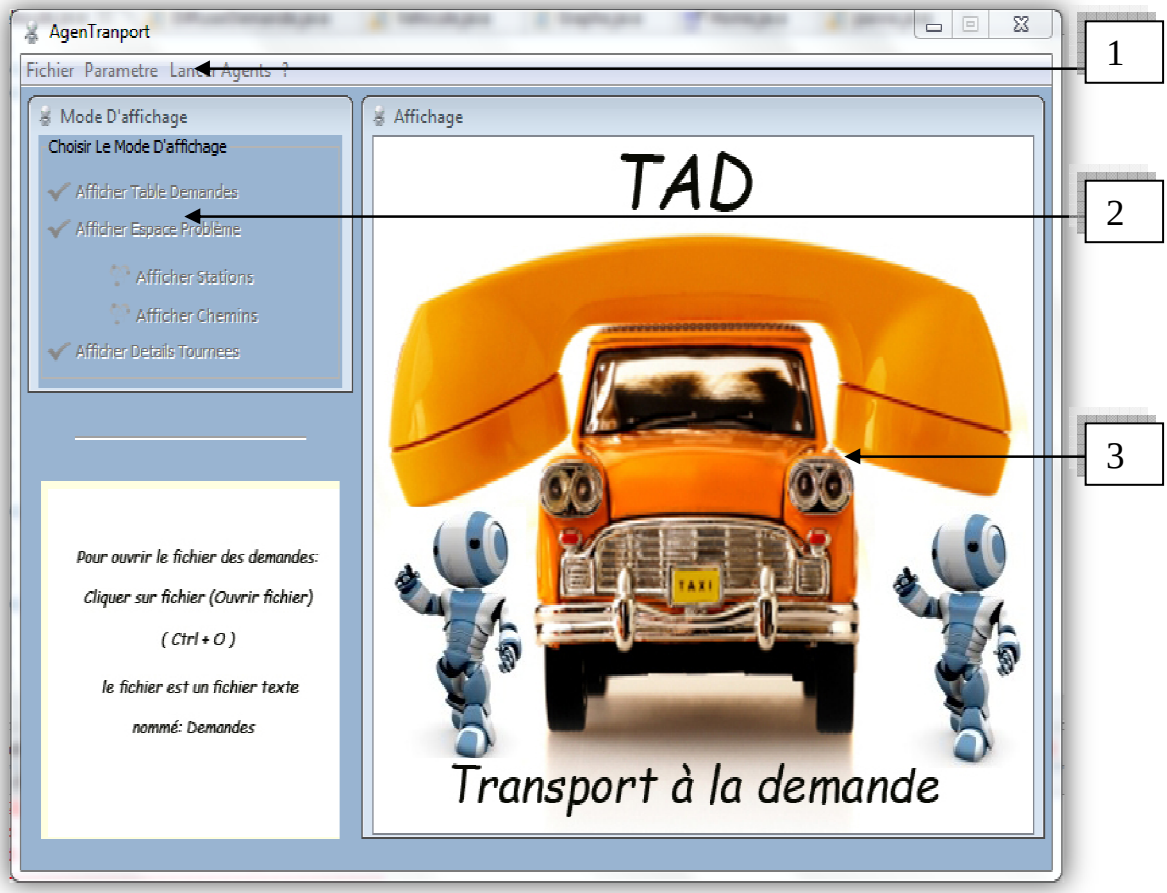


Figure 4.13 : L'interface de l'application**.**

1 **Barre de Menu :** qui contient trois menu comme suit :

1) **Fichier :** contient les opérations permettant de charger les données et de quitter L'application.
2) **Paramètre :** ce menu contient 2 opérations :

- **Paramètre Tabou :** configuration des paramètres concernant le véhicule et les paramètres de la recherche Tabou.

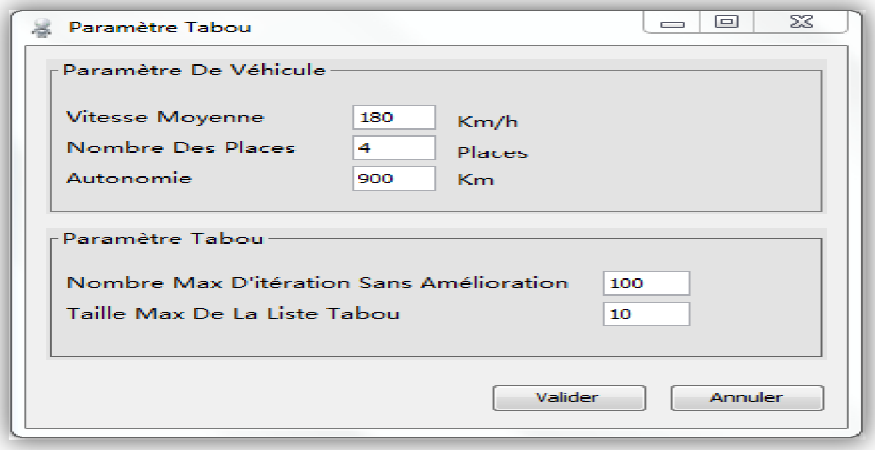


Figure 4.14 : Paramètre Tabou.

- **Etablir un cas de panne :**

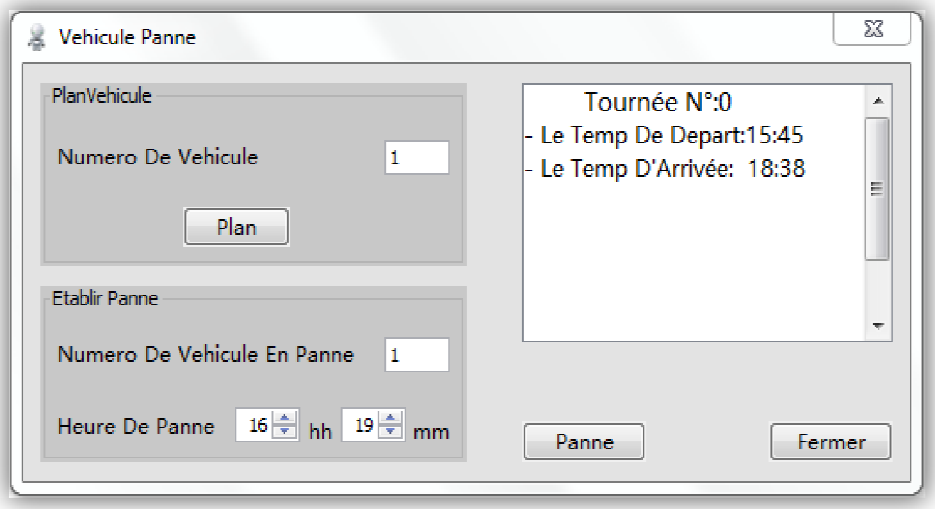


Figure 4.15 : Paramètre de panne.

3) **Lancer agents :** ce menu contient 2 opérations.

- **Lancer l'agent central** : pour exécuter a recherche tabou.

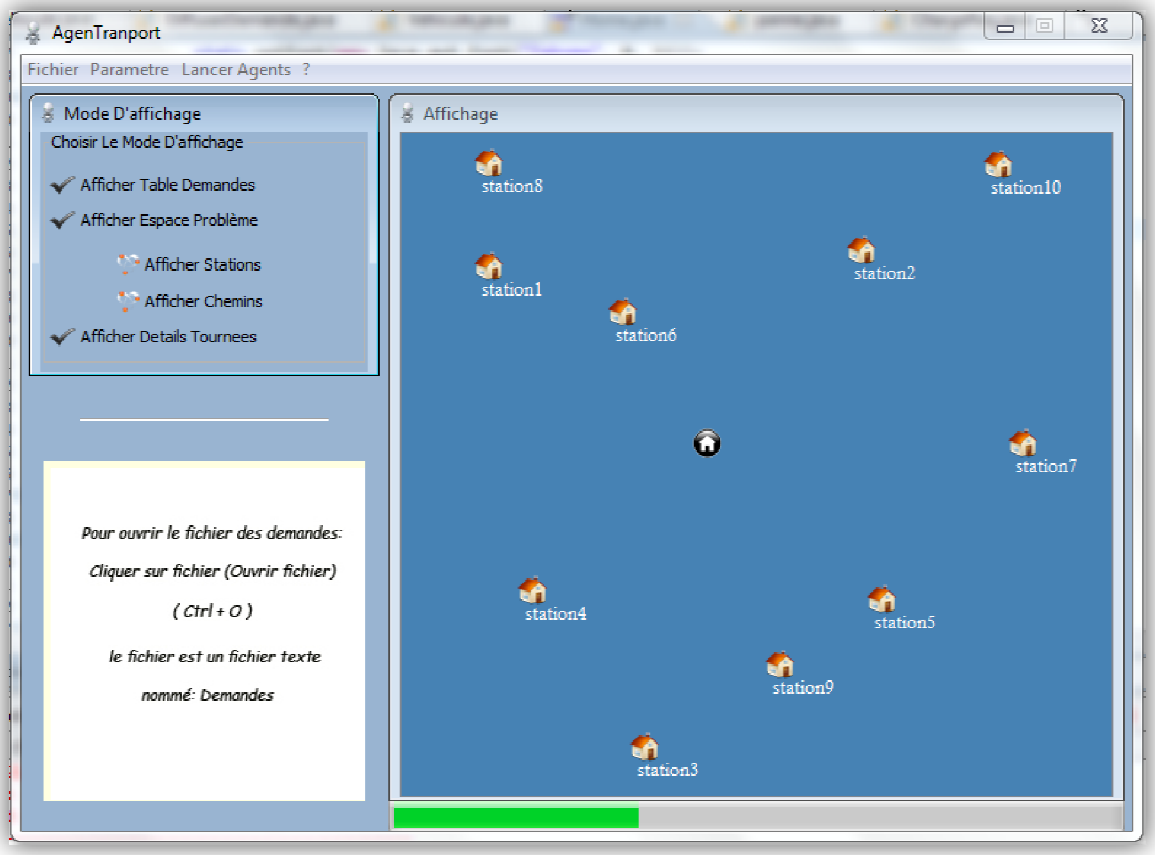


Figure 4.16 : L'Agent Central.

- **Lancer les agents véhicules : (Agent sniffer avant le cas de panne).**

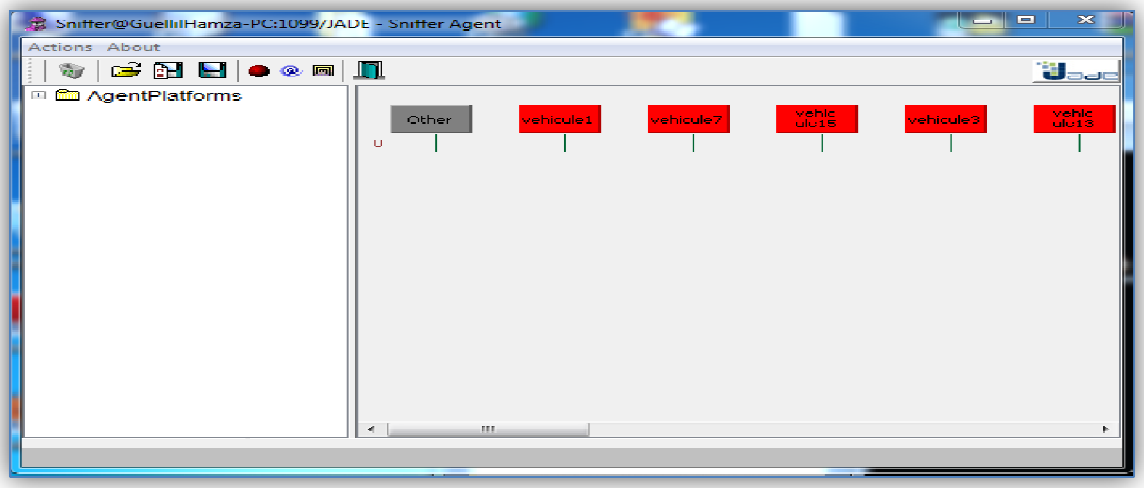
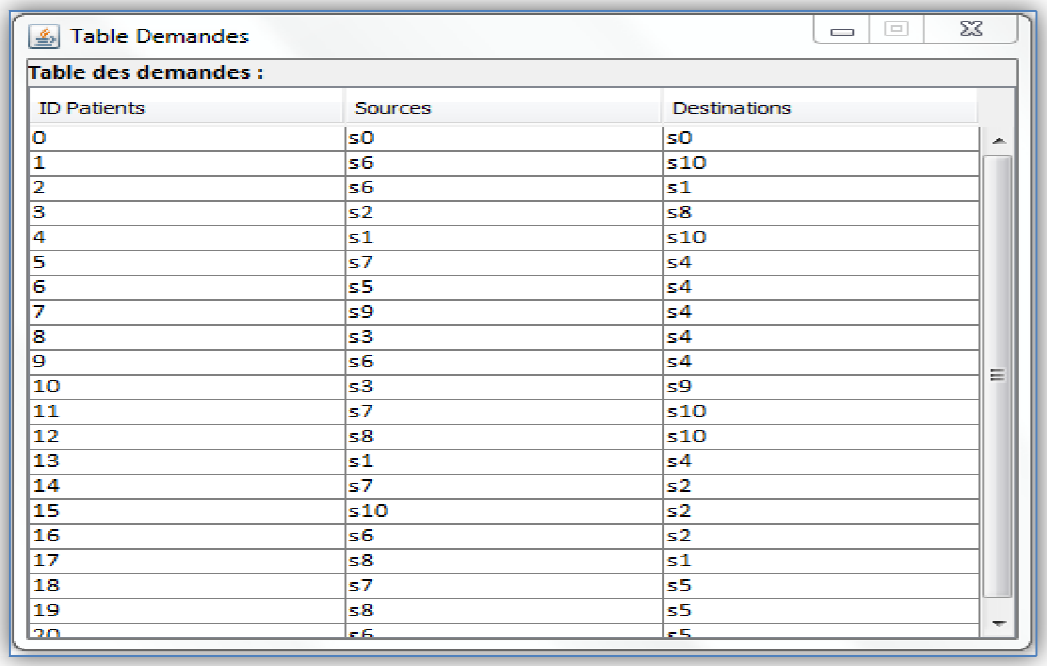


Figure 4.17 : L'Agent Sniffer avant la panne.

2 **Zone d'affichage :** contient les taches suivant :

- **Afficher Table Demande.**



ID Patients	Sources	Destinations
0	s0	s0
1	s6	s10
2	s6	s1
3	s2	s8
4	s1	s10
5	s7	s4
6	s5	s4
7	s9	s4
8	s3	s4
9	s6	s4
10	s3	s9
11	s7	s10
12	s8	s10
13	s1	s4
14	s7	s2
15	s10	s2
16	s6	s2
17	s8	s1
18	s7	s5
19	s8	s5
20	s6	s5

Figure 4.18 : Affiche la Table des demandes.

- **Afficher Espace Problème :** contient deux opérations
  - Afficher stations
  - Afficher chemins

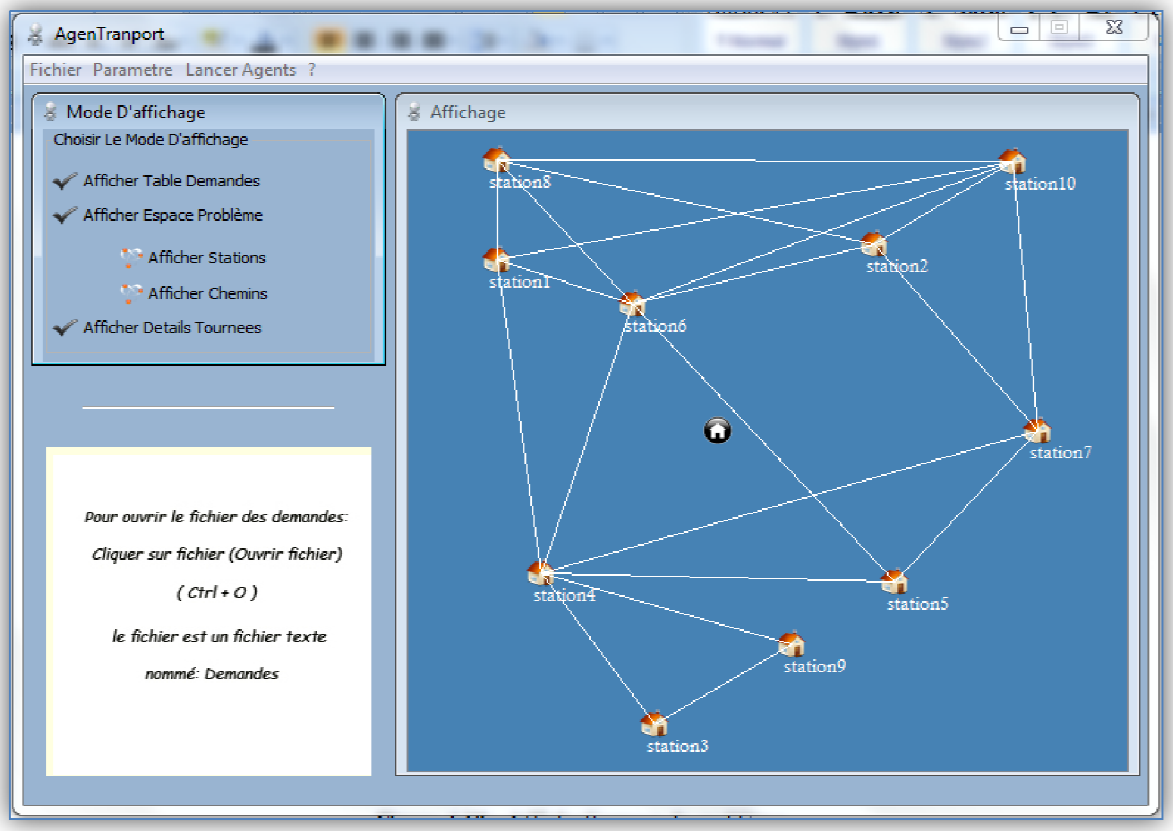


Figure 4.19 : Affiche l'espace de problème.

- **Afficher Détails Tournée.**

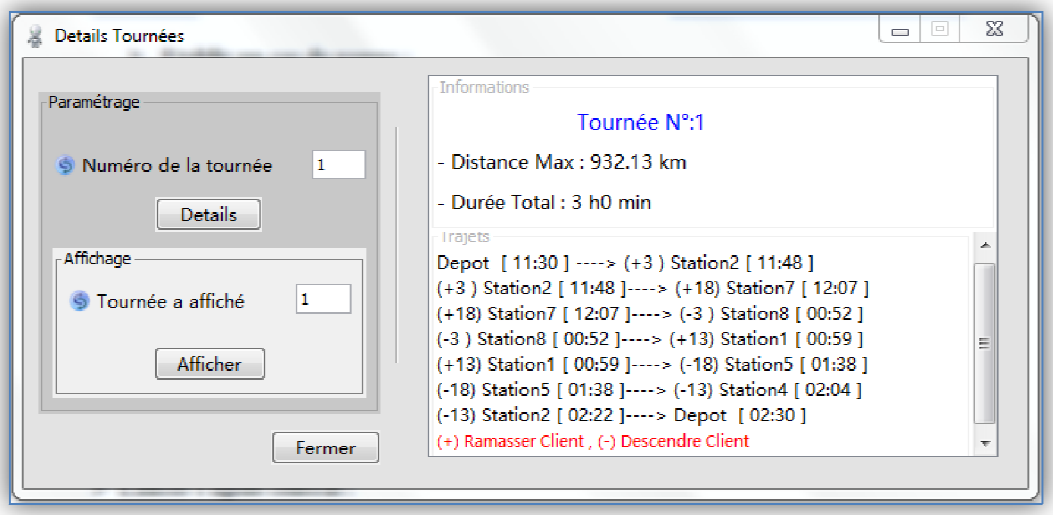


Figure 4.20 : Détails de Tournée (temps).

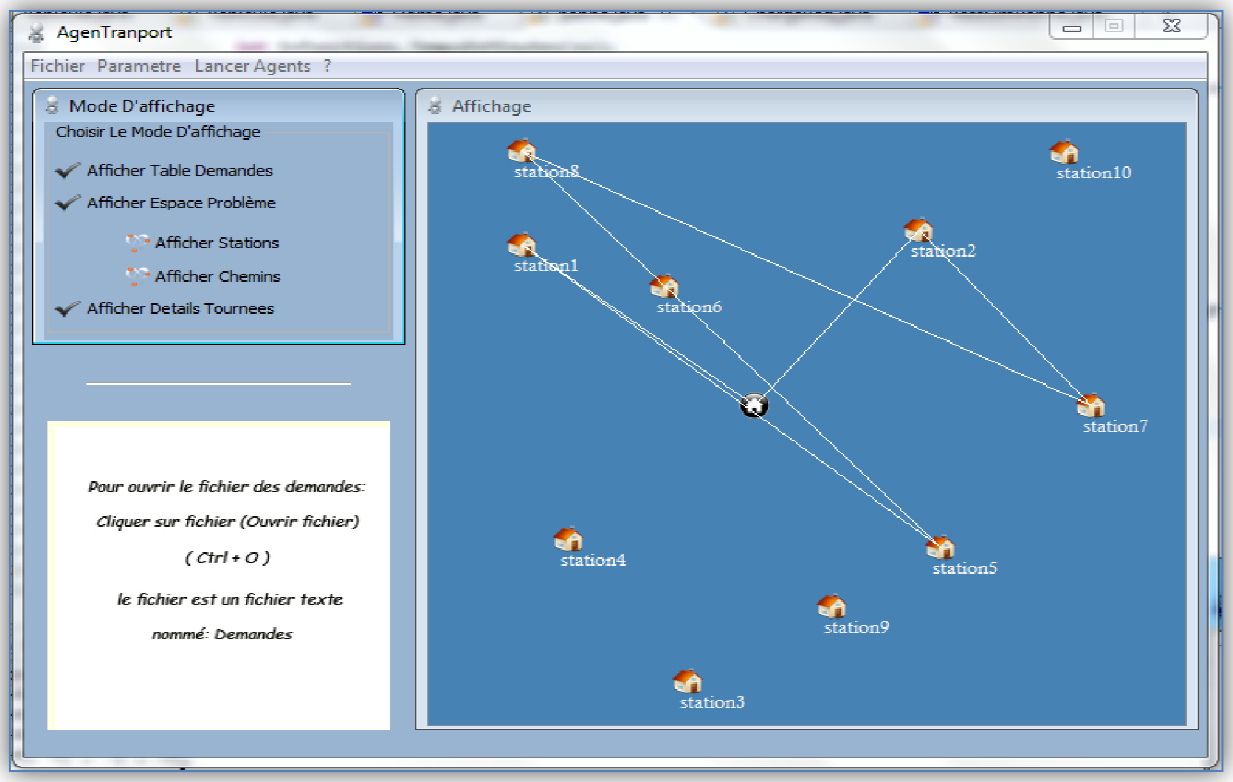


Figure 4.21 : Détails de Tournée (Graphique).

3 **Zone de travaille :** Un espace pour afficher les différents résultats.

- **Description de cas de panne :**

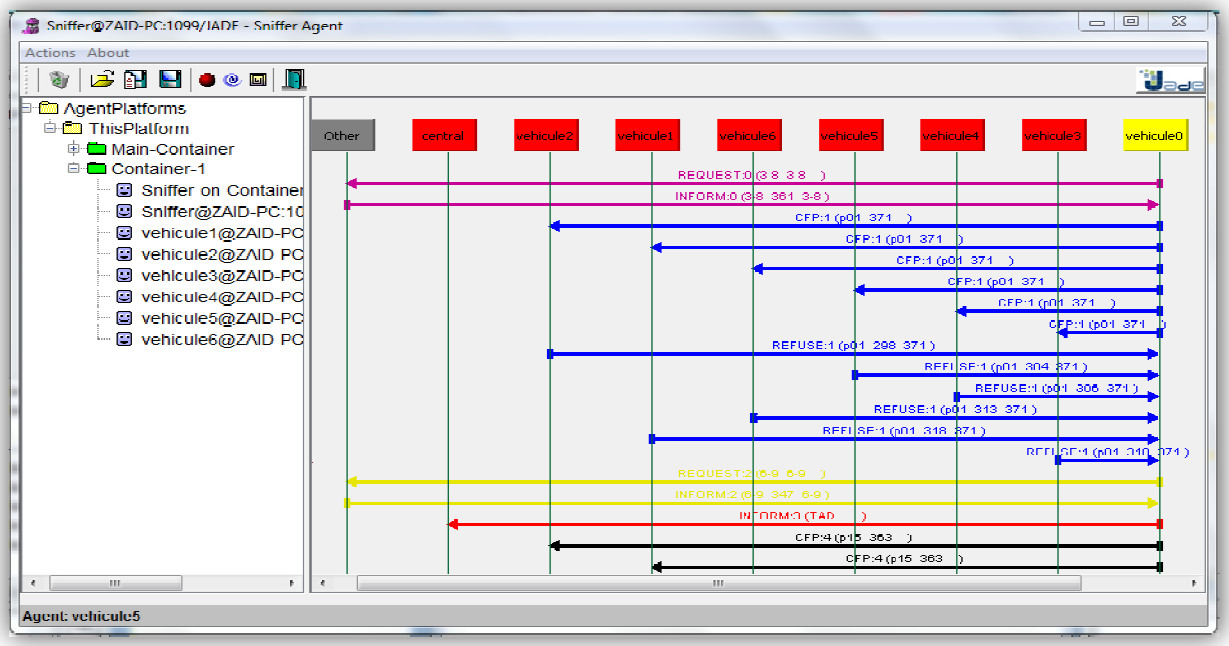


Figure 4.22 : L'Agent Sniffer après la panne.

➢ **Résultats de négociation :**

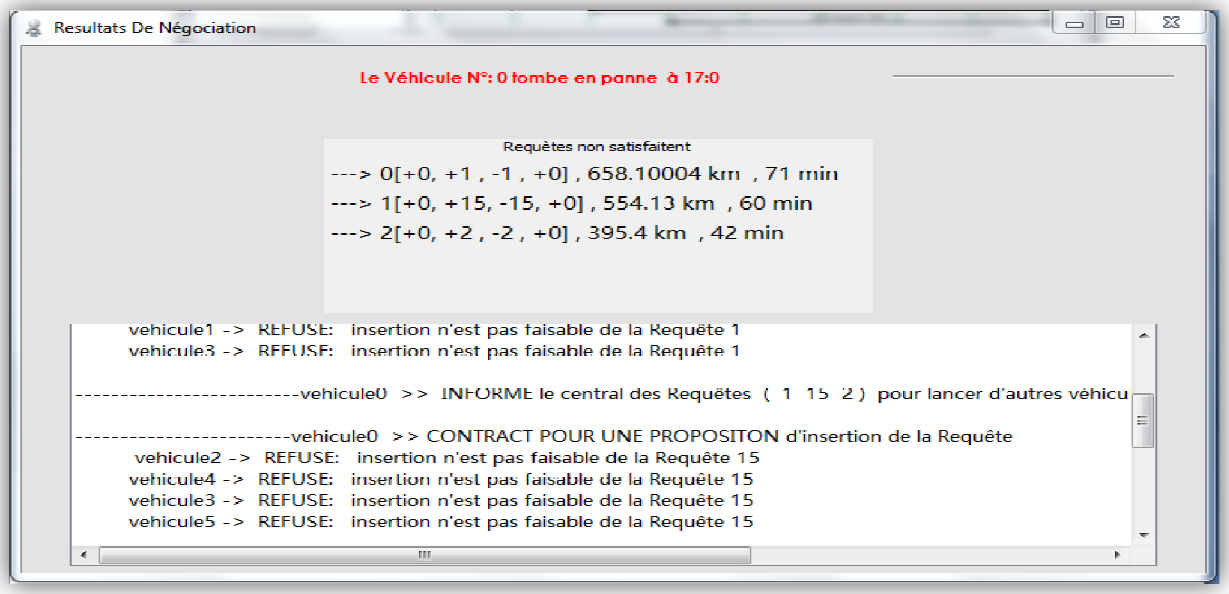


Figure 4.23 : Résultats de négociation.

7. Implémentation des agents du système.

Comme nous l'avons déjà mentionné dans le chapitre précèdent les fonctionnalités de notre architecture sont assurées par deux types d'agents : agent central et agents véhicule.

Avant d'entamer l'implémentation interne de chaque agent nous présentons quelques classes Jade et un ensemble de méthodes utilisées dans l'implémentation de ces deux derniers.

7.1. Classes et méthodes JADE utilisées.

✓ **La classe Agent :** Elle est définie dans le package jade.core et représente une super classe commune pour tous les agents définis par l'utilisateur. Du point de vue programmeur, la conséquence, est qu'un agent JADE est simplement une classe JAVA qui étend la clase de base " Agent ". Les méthodes de la classe Agent que nous avons utilisé sont :

- ***Setup () :*** sert à l'initialisation de l'agent.
- ***doDelete() :*** fait passer l'agent à l'état " Deleted ", ce qui aura pour effet de supprimer l'agent de l'AMS (et appeler la méthode tackDown () juste avant).
- ***Send (ACLMessage) :*** lorsque un agent souhaite envoyer un message, il doit créer un nouvel objet ACLMessage, compléter ses champs avec des valeurs appropriées et en fin appeler la méthode ***Send ().***

- ***Receive (MessageTemplate)*** : lorsque un agent souhaite recevoir un message, il doit employer la méthode ***Receive ().***
- ***addBehaviour(Behaviour) :*** permet d'ajouter un comportement (Behaviour) à la file de comportement d'un agent.

✓ **La classe Behaviour :** Elle est définie dans le package jade.core. JADE. Elle utilise l'abstraction comportement (Behaviour) pour modéliser les tâches qu'un agent peut exécuter. Tout objet de type Behaviour dispose d'une méthode action () et une méthode done ().

- ***Action () :*** constitue le traitement à effectuer par un objet de type Behaviour.
- ***done () :*** vérifier si le traitement d'un objet de type Behaviour est terminé.

✓ **La classe cyclicBehaviour :** Elle est définie dans le package jade.core. La classe cyclicBehaviour est une classe abstraite qui hérite la classe abstraite Behaviour. Modélise un comportement qui s'exécute continuellement dans le temps.

✓ **La classe oneShotBehaviour :** Elle est définie dans le package jade.core. La classe oneShotBehaviour est une classe abstraite qui hérite la classe abstraite Behaviour. Modélise un comportement qui s'exécute seulement une fois dans le temps.

✓ **La classe ACLMessage :** Définit dans le package jade.lang.acl, représente les messages qui peuvent être échangé par les agents.

✓ **La classe DFservice :** Elle est définie dans le package jade.domain. Nous avons utilisé trois méthodes implémentées par cette classe :

- **Register :** permet d'enregistrer un agent dans les pages jaunes du DF.
- **Deregister :** supprimer un agent des pages jeunes du DF.
- **Search :** permet à un agent d'obtenir une liste de tous les agents inscrits dans le DF qui peuvent réaliser la tache qu'il cherche à exécuter.

7.2. Implémentation de l'agent Central.

Après le lancement de l'interface principale de notre application et la validation d'un jeu de données l'agent central crée par la classe *jade.Core.Agent* entre en état d'activité. Il commence par l'exécution de ses comportements.

L'exécution des comportements est gérée par le déclanchement des méthodes suivantes :

- **Setup ( ) :** cette méthode gère l'exécution de l'agent Central. Chaque agent par défaut exécute la méthode setup ( ), elle réalise les opérations suivantes :

✓ *DFService.register (**this**, dfd) :* enregistrement de la description de l'agent dans DF (Directory Facilitator).

✓ addBehaviour ( ) : cette methode permet d'ajouter des comportements liés à l'agent central. Les comportements réalisés par l'agent central sont :

- **Génération des tournées :** appliquer l'heuristique Tabou.

```
Public class GenererTournees extends OneShotBehaviour {
Public void action () {

// partie declarations……..

            solutioninitial=Fonctions.SolutionInI();
            RechTabou.tabou(solutioninitial);
            solution=RechTabou.SolutionOptimal()}
                                                    }
```

Figure 4.24 : comportement de générations tournées.

- **Lancer l'agent véhicule :** l'agent central affecte les tournées obtenues aux agents véhicules comme plan de circulation et lance ces derniers.

```
public class LancerLesVehicules extends OneShotBehaviour{
public void action (){
        // partie déclarations.....
// Céer l'agent véhicule caractérisé par NomAgent, Plancirculation
Agent=LesVehicules.createNewAgent
("vehicule"+i,"P11.Vehicule",arg)
// Affecter chaque tournée comme un plan circulation
    plan=solution.get(i);
// Lancer l'agent véhicule
    Agent.start ();
 }// Cette action se répète à savoir le nombre de tournées
obtenues
```

Figure 4.25 : Principaux instruction du comportement lancer véhicule.

7.3. Implémentation de l'agent véhicule.

Dans notre application les agents véhicules sont lancés par l'agent central après la génération des tournées. Le comportement d'un agent véhicule se figure dans la gestion de cas de panne. Un cas de panne est résolu par l'ensemble des agents via le protocole Contract_Net offert par la plateforme jade (ce protocole est bien explique dans le chapitre précédent). Le principe de résolution est d'insérer les requêtes non satisfaites par un véhicule en panne dans un véhicule actif sans violer les contraint mentionner précédemment, si aucune véhicule ne propose une insertion alors le véhicule en panne contacte l'agent central pour établir une nouvelle tournée.

L'exécution des comportements d'un agent véhicule est gérée par le déclanchement des méthodes suivantes :

- ***Setup() :*** cette méthode est déclenché à chaque création d'un agent véhicule par l'agent central cette méthode réalise les opérations suivant :
  1. ***arg= getArguments() :*** Récupérer les arguments de chaque véhicule cet argument se présente comme un plan de circulation.
  2. ***DFService.register (this, dfd) :*** Enregistrement de la description de l'agent dans DF (Directory Facilitator).

```
public void setup()
      {
//Recuperation des arguents
            Object[] arg=getArguments();
            if (arg != null)
                  plan=(Tournee) arg[0] ;
//Enregistrement de l'agent de la page jaune
            DFAgentDescription dfd = new DFAgentDescription();
            dfd.setName(getAID());
            ServiceDescription sd = new ServiceDescription();
            sd.setType("TAD");
            sd.setName("JADE-TADSolving");
            dfd.addServices(sd);
            try {
              DFService.register(this, dfd);
            }
            catch (FIPAException fe) {
              fe.printStackTrace(){
            }
```

3. addBehaviour ( ) : cette méthode permet d'ajouter des comportements liée a l'agent Véhicule. Les comportements réalisés par l'agent Véhicule sont :

A. Tombe en panne :

private class TomberEnPanne **extends** OneShotBehaviour{}

Ce comportement est composé de 3 parties :

- Récupération de la liste des autres agents : l'agent véhicule récupère tout les agents actif après la date de panne.
- Récupération de la liste des requête non satisfaites : l'agent véhicule récupère toute les requêtes non satisfaite après le cas de panne
- Communication et négociation : la communication et la négociation se fait entre l'agent véhicule et les autres agents actif, elles sont présentées par la **classe DifuserDemande (). Public class** DiffuserDemande **extends** Behaviour {}.

Cette classe est composée des 4 parties suivantes :

i. Le contrat : l'agent en panne envoi une proposition d'insertion à chaque agent actif, cette proposition se présente comme un message le code suivant montre la structure générale du contrat.

```
ACLMessage msg = new ACLMessage(ACLMessage.CFP);//type de message.
           msg.setConversationId(id);
   msg.setContentObject((Serializable) dem); //contenue du message est la
                                  demande a inseré.
for(int i=0;i<ListVeh.length;i++)
    msg.addReceiver(ListVeh[i]); //definer la destination…
                   msg.setReplyWith ("cfp"+System.currentTimeMillis());
                   myAgent.send(msg);//envoi de message.
```

ii. Réception des réponses: l'agent en panne reçoit deux types de message.et choisis la meilleure proposition envoyée par les agents actifs.

- *Refuse* : l'agent actif refuse la proposition envoyée par l'agent en panne en cas de violation des contraint de son plan.
- *Accept* : l'agent actif accepte la proposition de l'agent en panne et envoi une autre comme un message contient la meilleure insertion dans son plan avec la durée de service.

```
// Recevoir tous les  proposals/refusals des autres agents vehicule
ACLMessage reply = myAgent.receive(mt);
    if (reply != null) {
      // Reponse reçu
       if (reply.getPerformative() == ACLMessage.PROPOSE) {
                        // l'insertion est faisable
              int surcout = Integer.parseInt(reply.getContent());
               if (BestVeh == null || surcout < BestSurcout) {
                 // c'est la meilleure insertion
                    BestSurcout = surcout;
                    BestVeh = reply.getSender();//véhicule choisit pour
l'insertion de la requete.
```

iii. Envoi l'acception : l'agent en panne envoi un message d'acceptation pour l'agent choisit précédemment.

```
//type de message a envoyé.
ACLMessage vainqueur = new ACLMessage(ACLMessage.ACCEPT_PROPOSAL);
            vainqueur.addReceiver(BestVeh);//véhicule choisit
            vainqueur.setConversationId(id);
             vainqueur.setReplyWith("Vainqueur"+System.currentTimeMillis());
                   myAgent.send(vainqueur);// l'envoi de message
```

iv. Confirmation : l'agent en panne recevoir un message de confirmation envoyé par le véhicule choisit précédemment pour l'insertion de la requête envoyé.

```
reply = myAgent.receive(mt);
                    if (reply != null) {
                      if (reply.getPerformative() == ACLMessage.CONFIRM)
RessSimPanne.Afficher("             "+reply.getSender().getLocalName()+"  ->
CONFIRME :   insertion avec succée de la Requête "+" ("
   + Graphe.GetNum((List<String>) dem)+")");
                     }

//Affichage d'un message de confirmation.
```

B. Traitement de nouvelle demande :

Public class TraitementNewDem **extends** CyclicBehaviour {}.

On peut composer ce comportement en 4 parties liées :

i. Récupérer la requête :

```
MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.CFP);
                    ACLMessage msg = myAgent.receive(mt);
//recuperation de la demande a inserer
     List<String> d = (List<String>) msg.getContentObject();
```

ii. Meilleur insertion : dans cette partie l'agent essaye de définir la meilleure insertion de la requête dans son plan de circulation.

```
//déterminer la meilleur insertion de cette demande dans la tournée (si
possible)
       List<String> BestIns=Tabou2Opt.two_opt(plan, d);
      float t=Fonctions.calculeDistance(BestIns)/2;
      int nbrPlace= ParametreTabou.getPlaces()*2+2;
      int taille=(BestIns).size();
      float distmax=ParametreTabou.getAutonomie();
```

iii. Propose : Si l'insertion de la requête est faisable l'agent actif envoi une proposition d'insertion cette proposition est envoyée comme la meilleur durée de service de cette requête.

```
  if(t<distmax && taille<nbrPlace){ // l'insertion est faisable. Repondre
par le cout(durée de service).
       reply.setPerformative(ACLMessage.PROPOSE);
      surcout=Graphe.calculeduree(BestIns);
      reply.setContent(String.valueOf(surcout));
      RessSimPanne.Afficher("         "+getAID().getLocalName()+
      " ->  PROPOSE:   insertion de la Requête "+Graphe.GetNum(d)
      +" avec le surcout "+surcout+" min" +"\n");
                                      }
```

iv. Refusals : Si l'insertion est infaisable l'agent actif envoi une refuse d'insertion.

```
  // l'insertion n'est pas faisable.
       reply.setPerformative(ACLMessage.REFUSE);
      reply.setContent("infaisable");
      RessSimPanne.Afficher("         "+getAID().getLocalName()+" ->
      REFUSE:   insertion n'est pas faisable de la Requête
      "+Graphe.GetNum(d)+"\n");
```

C. Insérer la nouvelle demande :

public class InsererNewDem **extends** CyclicBehaviour {}

Après la réception de l'acceptation envoyée par le véhicule en panne l'agent actif envoi un message de confirmation d'insertion pour le véhicule en panne et fait le mise a jour de son plan.

```
MessageTemplate mt =
MessageTemplate.MatchPerformative(ACLMessage.ACCEPT_PROPOSAL);
ACLMessage msg = myAgent.receive(mt);              //recevoir l'acceptation.
    ACLMessage reply = msg.createReply();
    reply.setPerformative(ACLMessage.CONFIRM);     //préparer la confirmation.
    myAgent.send(reply);                           //envoyer la confirmation
    Central.MAJVeh(getAID().getLocalName(), plan);//mise a jour du plan.
```

8. Conclusion.

Dans ce dernier chapitre nous avons présenté tout ce qui est lié au développement de notre application. Nous avons parlé des différents outils utilisés dans l'implémentation des différents composants. Nous avons présenté aussi une brève description des différentes plateformes utilisées dans le domaine de développement des SMA avec une justification de notre choix concernant la plateforme JADE. Le chapitre est clôturé par une description détaillée des interfaces graphiques de notre application

Conclusion générale

Dans ce projet, nous avons développé un système de gestion de transport à la demande (TAD). Ce problème préoccupe de plus en plus les compagnies de transport, car il a un impact direct sur la qualité des services offerts aux clients. Nous nous sommes intéressés à une variante de ce problème qui est le problème de transport à la demande à plusieurs véhicules de capacité limités. Ce problème, consiste à déterminer les tournées et les horaires pour les véhicules qui effectuent le transport des clients d'une origine vers une destination en minimisant le coût total des tournées. Notre problème concerne aussi la proposition des mécanismes pour la gestion des cas de panne des véhicules, afin d'assurer un service de transport efficace.

Avant d'aborder la résolution de notre problème, nous avons présenté une étude bibliographique relative au domaine du TAD. Nous avons présenté aussi les caractéristiques des systèmes multi-agents (SMA) et les principes des différentes méthodes d'optimisation existantes. Ces deux domaines sont le fondement de plusieurs approches dédiés à la résolution de plusieurs variantes du problème du TAD.

Ensuit, Après avoir décrit la description mathématique de notre problème nous avons présenté notre proposition sous la forme d'une architecture multi-agents. Le fonctionnement global de l'architecture proposée est assuré par une recherche tabou et une stratégie de négociation entre agents basé sur le protocole Contrat-Net pour la gestion des pannes.

En fin nous avons exposé les résultats des expérimentations effectuées sur un jeu de données conçu aléatoirement.

Bien que le domaine de transport à la demande est très vaste. Nous pensons que nous avons atteint les buts que nous avons fixés au début du fait que nous avons obtenu une bonne combinaison entre une approche distribué (SMA) et des méthodes d'optimisation. Pour cela, et dans une perspective d'amélioration de notre approche, nous proposons :

- D'intégrer d'autres contraintes comme l'état des clients pour traiter le cas de la flotte hétérogènes de véhicules.
- De traiter le problème de transport à la demande dans le cas dynamique

Bibliographies

[1] Xiagang ZHAO, **«Algorithmes pour les problèmes de tournées à la demande »**, 195p, thèse de doctorat en Informatique, Université Blaise pascal – Clermont-Ferrand II.

[2] M.W.P. Savelsbergh, **« Local Search for Routing Problems with Tirne Windows »**, Annals of Operations Research 4, 285-305p, 1995.

[3] Zeddini Besma, « **Modèles d'Auto-Organisation Multi-Agent pour le Transport à la Demande** », 182p, thèse de doctorat en Informatique et Mathématiques Appliquées, Université du Havre.

[4] Nejeoui. A, Elfazziki. A, Sadgal. M, **« Une Approche multi-agents pour la Modélisation et l'optimisation des Systèmes de gestion de transport maritime »**, Article de thése Dépt. Informatique, Faculté des Sciences Semlalia, Bd Pr. My Abdellah B.P 2390 Marrakech 40000 Maroc.

[5] F. Vernadat, **« Interoperable enterprise systems: Architectures, methods and metrics »**, Rapport technique, LGIPM, Université de Metz, France, 2007.

[6] LAGA SAIDA, NOUIOUA LEILA, **« Deux approches méta-heuristiques hybrides basées sur les colonies d'abeilles pour la résolution du problème de la T-coloration des graphes»**, Mémoire de fin d'études Pour l'obtention du diplôme d'ingénieur en Systèmes Informatiques, Ecole National Supérieure d'Informatique (ESI) Oued-Smar, Alger,2008/2009.

[7] Karima benatchba, **« Modèle d'execution pour l'aide à la résolution de problème MAX-SAT »**, 200p, Thése de doctorat en informatique, Institut national d'informatique, Alger, 30 Juin 2005.

[8] ALI KANSOU, **« Nouveaux algorithmes d'optimisation combinatoire pour les problèmes de tournées sur arcs »,** 160p, thése de doctorat en informatique, université de Havre, 18 Novembre 2009.

[9] Hayfa Zgaya, **« Conception et optimisation distribuée d'un système d'information d'aide à la mobilité urbaine : Une approche multi-agent pour la recherche et la composition des services liés au transport »**, 240p, thèse de doctorat en Automatique et Informatique Industrielle, Ecole centrale de tille, 6 juillet 2007.

[10] César Rego, Catherine Roucairol, **« Le probléme de tournées de véhicules : étude et résolution approchée»,** 116p, Rapport de recherche, N° 2197 février 1994.

[11] BEN MOHAMED AHMED, **« RÉSOLUTION APPROCHÉE DU PROBLÈME DE BIN-PACKING »**, 109p, Thése de doctorat en informatique, Université du Havre Laboratoire LMAH, 03 décembre 2009.

[12] C.H. Papadimitriou. **« The complexity of combinatorial optimization problems»,** PhD thesis, Princeton University, New Jersey, USA, 1976.

[13] S. Busse, R. Kutsche, U. Leser and H.Weber, **« federated information systems: Concepts, terminologies and architectures »**, Technical Report Nr.99-9, Berlin University, 1999.

[14] J.Ferber; **les systèmes Multi-agents: Vers une Intelligence Collective**. Techniques et science informatiques, vol. 16, n°8, 1997, pp.979-1012.

[15] M.Wooldridge, and J.R. Jennings; ‘’**Agent Theories, architectures, and languages: a Surevy**. In M. Wooldridge and J. R. Jennnings, editors, intelligent Agent, LNAI890, Springer Verlag, 1995, pp.1-39.

[16] Kamel ZIDI, **« Système Interactif d’Aide au Déplacement Multimodal (SIADM) »**, 138p, Thése de doctorat en Automatique et Informatique Industrielle, Ecole Centrale de Lille Université des Sciences et Technologies de Lille, 13 décembre 2006.

[17] J.-F Cordeau, G. Desaulniers, J. Desrosiers, M. Solomon, and F. Soumis. “**The VRP with Time Windows**”. « Technical Report, Département de Mathématiques et de Génie Industriel », Ecole polytechnique de Montréal, Canada, June 2000

[18] A. Kimouche « **Méta-heuristique pour la résolution des problèmes de transport : application pour le transport des patients ».** mémoire de Magistère, université de Batna,2012.

[19] B Zeddini, M.Temani, A.Yassine, et K.Ghedira. « **Du collectif pour le problème de transport à la demande : un modèle d’auto-organisation** ». In Proceedings of the 8th international conference NOuvelles TEchnologies de la REpartition, NOTERE, Lyon (France), 2008.

[20] M.W.P. Savelsbergh, « **Local Search for Routing Problems with Tirne Windows**”.

Annals of Operations Research 4, 285-305, 1995.

[21] N. Zarour ; **‘’contribution à la modélisation de la coopération des systèmes d’information distribués et hétérogènes : le système DAArCHE** ‘’. Thèse de doctorat de l’Université Mentouri de Constantine. Le 18 septembre 2004.

[22] L.Gasser; Social **Conceptions of Knowledge and action. Rapport technique** ACT-AI-355-90, MCC, Octobre 1990.

[23] J.P. Meutter; **‘’The Design of intelligent Agents: A Layered Approach’’**. LNAI1177, Springer Verlag, 1996.

[24] S. Bouzidi ; ‘’Un modèle **de Négociation basé sur la Formation de Coalition pour l’Allocation des Taches dans les Systèmes d’Information Coopératifs**‘’. Mémoire de magister.2005.

[23] http://**www.memoireonline.com**/07/08/1413/m_modelisation-systeme-multi-agents-hypermedia-educatif22.html.

[24] http://**perso.limsi.fr**/jps/enseignement/examsma/2004/GRAMOLI/html/index.html#ref10.

[25] http://**perso.limsi.fr**/jps/enseignement/examsma/2005/1.plateformes_1/jade.htm.

[26] http://**www2.econ.iastate.edu**/tesfatsi/repastsg.htm.

Résumé

Les Problèmes de transport à la demande (TAD) sont des problèmes d'optimisation combinatoire qui consistent à trouver les tournées des véhicules optimales satisfaisant des demandes de transport, généralement la résolution exacte de ces problèmes n'est pas possible dans un temps raisonnable, d'où le recours aux méthodes approchées ((méta-) heuristiques).

Dans ce travail, nous présentons un système de gestion de transport à la demande sur la méta-heuristique recherche tabou, ce système prend en considération le cas où un véhicule tombe en panne pour satisfaire les requêtes que ce véhicule ne sera pas en mesure de les satisfaire. L'aspect dynamique et distribué de ce problème nous a conduits à adopter une modélisation multi agents en raison des caractéristiques des agents et leurs capacités de communiquer et coopérer pour résoudre un problème.

Mots clés : Transport à la demande(TAD), Système multi agent(SMA), Négociation, Optimisation combinatoire, Méta-heuristiques, Heuristique d'insertion, Recherche tabou.

Abstract

The problems of transport on demand are combinatorial optimization problems which means finding the optimal vehicles routes to satisfy demands of transport, generally, the exact resolution of these problems is not possible in a reasonable time, hence the use of approximate methods (the heuristics).

In this work, we present a transportation management system based on the meta-heuristic tabu search. This system takes into account the case where a vehicle breaks down to satisfy the requests that this vehicle will not be able to satisfy. The dynamic and distributed aspects of the problem incite us to adopt a multi-agent approach due to the characteristics of agents and their capability to communicate and cooperate to solve a problem.

Keywords: Transport on demand, multi agent system, Negotiation, Combinatorial optimization, Meta-heuristic, Insertion heuristic, Tabu search.