Optimisation distribuée pour la recherche des itinéraires multi-opérateurs dans un réseau de transport co-modal

Mohamed Firas Feki

N° d'ordre : 145

ECOLE CENTRALE DE LILLE

THESE

présentée en vue
d'obtenir le grade de

DOCTEUR

en

Spécialité : Automatique et Informatique Industrielle

par

Mohamed Firas FEKI

DOCTORAT DELIVRE PAR L'ECOLE CENTRALE DE LILLE

Titre de la thèse :

Optimisation distribuée pour la recherche des itinéraires multi-opérateurs dans un réseau de transport co-modal

Soutenue publiquement le 9 Décembre 2010 devant le jury d'examen :

Président	Etienne, CRAYE, Professeur, Ecole Centrale de Lille
Rapporteur	Mariagrazia, DOTOLI, Professeur, Poltecnico di Bari
Rapporteur	Mekki, KSOURI, Professeur, ENIT
Examinateur	Farouk, KAMOUN, Professeur, ENSI
Examinateur	Christophe, DI POMPEO, Professeur, Université Lille 2
Examinateur	Moncef, ZOUARI, pdg
Directeur de thèse	Slim, HAMMADI, Professeur, Ecole Centrale de Lille

Thèse préparée dans le Laboratoire LAGIS

Ecole Doctorale SPI 072
PRES Université Lille Nord-de-France

La théorie,

c'est quand on sait tout et que rien ne fonctionne.

La pratique,

c'est quand tout fonctionne et que personne ne sait pourquoi.

Ici, sont réunies théorie et pratique :

rien ne fonctionne... et personne ne sait pourquoi

Albert Einstein

REMERCIEMENTS

Je tiens d'abord à exprimer ma profonde reconnaissance au Professeur Slim HAMMADI, Professeur à l'École Centrale de Lille et directeur de cette thèse, pour son encadrement consciencieux, la grande autonomie qu'il a su m'accorder ainsi que sa présence et son soutien scientifique et moral tout au long de ces trois années au sein du LAGIS. Merci pour la confiance dont tu as fait preuve à mon égard. Merci pour tes orientations, tes précieux conseils et ta patience.

Mes sincères remerciements vont aussi au Professeur Étienne CRAYE, Professeur à L'École Centrale de Lille et directeur de cet honorable établissement, pour l'honneur qu'il m'a fait en acceptant de présider ce jury.

J'adresse aussi mes vifs remerciements au Professeur Mariagrazia DOTOLI, Professeur de la Poltecnico di Bari et au Professeur Mekki KSOURI, Professeur à l'ENIT qui m'ont fait le grand honneur d'accepter de rapporter cette thèse. Je les remercie infiniment pour le temps consacré à cet effet en dépit de toutes les responsabilités qu'ils ont.

Il m'est aussi agréable d'exprimer toute ma reconnaissance au Professeur Farouk KAMOUN, Professeur de l'ENSI et au Professeur Christophe DI POMPEO, Professeur de l'Université Lille2 pour avoir accepté d'être les examinateurs de cette thèse et de faire partie de mon jury.

Je souhaite adresser mes remerciements au docteur Mohamed Amine KAMOUN qui a su m'initier à la compréhension des problèmes et me mettre sur les rails dans ma recherche de solutions.

Ces remerciements ne seraient pas complets sans l'expression de toute ma gratitude au membre de l'équipe OSL : Sawsan SAAD, Jerbi KARAMA, Manel SGHAIER et Haifa ZGAYA pour leurs disponibilités, leurs aides et le temps qu'ils m'ont consacré.

J'adresse une pensée toute particulière à mes amis : Karim SELLAOUTI, Aymen HOSNI, Nasr MAKNI, Zied JAMOUSSI, Anouar CHAHED, Gassen MZOUGHI, Riadh TRAD, Johann GOULLEY, Frédéric FAUQUETTE et Fabrice RENAUDINEAU sans lesquels cette thèse ne serait pas de cette qualité. Je vous remercie d'avoir été là quand j'avais besoin d'aide et pour m'avoir soutenu. Ma reconnaissance dépasse ces quelques mots et ne peut être exprimée que par le cœur.

Je remercie également mes collègues de travail et tous ceux qui m’ont apporté leur soutien et leur aide d’une façon ou d’une autre à la réalisation de ce travail.

Enfin, je souhaite remercier ma famille qui m’a permis d’en arriver là et qui a toujours cru en moi. Je remercie du fond du cœur mes parents, Ali et Afifa FEKI, ma sœur Haifa et mon frère Salim pour leur soutien permanent, leur disponibilité et leurs sacrifices. Je remercie ma femme Ines pour sa patience et sa compréhension et mon fils Youssef pour le soutien qu’il m’a apporté avec son sourire.

Je leur dédie avec plaisir ce travail ainsi qu’à la mémoire de ma merveilleuse grand-mère Baya. Qu’ils voient en ces quelques lignes le témoignage de ma profonde reconnaissance.

Sommaire

Table des figures

Glossaire

SMA : Système Multi-Agent

SIAD : Système d’Information d’aide au déplacement.

ACI : Algorithme de Calcul d’Itinéraires

Covoiturage : partage d’un véhicule personnel et du coût du trajet entre conducteur et autres usagers

AutoPartage : voiture en libre service en ville

EcoPartage : Utilisation collective de la voiture particulière, elle englobe le **Covoiturage** et l’**AutoPartage.**

Introduction générale

De nos jours, l'intérêt porté à la préservation de l'environnement à travers la réduction des émissions de gaz à effet de serre prend de plus en plus d'ampleur. Cet enjeu conditionne la politique des transports dans le monde, et spécialement en Europe, qui recherche désormais un développement rentable et durable. En effet, dans un premier temps, les autorités européennes ont essayé de développer et d'encourager l'utilisation de différents types de transports en commun. Cette politique de transport multimodale a évolué depuis 2006 vers une politique comodale [**41**] qui n'oppose plus la voiture au transport public, mais encourage une combinaison de tous les modes de transport[**72**].
En passant de la concurrence à la complémentarité, le transport en général et le transport des personnes en particulier ont évolué vers plus d'efficience et plus de pertinence tant sur le plan environnemental (combinaison la moins polluante), que sur le plan économique (moindre coût). La « comodalité », qui a pour objectif premier d'optimiser les combinaisons des modes de transport (maritime , ferroviaire, routier et aérien), a ainsi remplacé la multimodalité [**56**].

Nous focaliserons notre étude sur un service en particulier : le transport de personnes, qui s'inscrit au cœur des politiques comodales, car elle combine tous les modes de transport en commun (train , métro, tram, bus ..) et promeut de nouveaux modes d'utilisation de la voiture particulière comme le covoiturage (partage d'un véhicule personnel et du coût du trajet entre conducteur et autres usagers) , ou l'AutoPartage (voiture en libre service en ville).

Un tel service ne peut être exploité pleinement que si l'information voyageur (information de l'itinéraire à prendre) est disponible facilement, optimale et flexible (un carnet de voyage au lieu d'un itinéraire unique), ce qui améliore significativement la qualité de service.

Toutefois, un problème de taille se pose : pour générer un itinéraire exploitant les services de plusieurs opérateurs de transport couplés à des services d'EcoPartage (covoiturage et d'AutoPartage), il faut, encore aujourd'hui consulter plusieurs sites internet (relatifs aux systèmes d'information des opérateurs et de covoiturage …). Selon le déplacement à réaliser, cette tâche de planification complexe peut s'avérer très difficile à réaliser et ne garantit pas toujours l'optimalité de l'itinéraire sélectionné. De plus, à l'issue de cette opération, un seul itinéraire est sélectionné ; or, pour une meilleure flexibilité et une meilleure qualité de service, il serait préférable de bénéficier d'un carnet de voyage proposant une liste de départs à des heures différentes.

D'autre part, les perturbations (grèves, problèmes techniques …) dans les réseaux des transports sont assez fréquentes et provoquent généralement une grande gêne chez l'usager. Dans le contexte d'un trajet comodal, le problème est encore plus complexe. En effet, le changement de mode de transport est conditionné par des heures de départ et d'arrivée fixes : dès lors, un décalage dans un tronçon dû à

une perturbation se répercute nécessairement et directement sur le reste du trajet. Pour une qualité optimale du service de transport, il est donc important d'informer les usagers en cas de perturbation et d'avoir la possibilité de leur proposer des solutions.

Notre analyse a pour but d'étudier les technologies permettant d'améliorer la qualité de service des transports, en temps normal comme en cas d'une perturbation. Nous nous sommes donc intéressés à la conception d'un système d'aide au déplacement capable de fournir une information voyageur (comodale) mettant en relation plusieurs opérateurs de transport (en commun et individuel). Le système en question doit être capable d'assister l'utilisateur dans la phase de planification par la constitution d'un carnet de voyage proposant plusieurs itinéraires possibles, chaque itinéraire pouvant inclure un ou plusieurs opérateurs de transport en commun ou individuel. De plus, il assiste l'utilisateur au cours du voyage en l'informant dans le cas d'une perturbation et en lui proposant des itinéraires de secours si nécessaire.

Ce travail s'appuie sur différentes avancées technologiques relatives au système afin de faciliter l'optimisation de l'information dans un environnement distribué (Multi-agent – SOA) ou à l'utilisateur afin de rendre l'information accessible à travers un grand nombre de médias (téléphone, mobile, PDA …).

Le présent document décrivant le travail effectué est organisé de la manière suivante :

Dans le premier chapitre, nous commençons par étudier le transport de personnes et les systèmes d'informations existants liés à cette problématique et nous nous intéressons à leurs architectures ainsi qu'à leurs apports.

Dans le deuxième chapitre, nous détaillons la conception et la modélisation des différentes composantes de notre système.

Dans le troisième chapitre, nous nous intéressons aux algorithmes de recherche d'itinéraires en exposant un bref état de l'art des algorithmes existants selon leurs contextes. Nous proposons, ensuite, un algorithme qui répond au besoin de notre système.

Dans le dernier chapitre, nous nous consacrons aux aspects relatifs au développement, à l'intégration et au déploiement de notre système et nous consolidons tout ceci par des simulations

I. Du Transport aux Architectures Systèmes

I.1. Introduction

Nous commençons par étudier l'évolution de l'utilisation des transports en commun avant d'analyser les causes et les conséquences des différents types de perturbations qui peuvent les impacter. Nous détaillons les problématiques liées à différents types « d'information voyageur » (intermodal, multimodal et comodal) et aux systèmes d'informations dédiés au transport que ce soit du côté client (pour simplifier l'utilisation du réseau) ou du côté exploitant (pour aider l'exploitant à mieux gérer son réseau).

Dans un second temps, nous analysons les architectures logicielles existantes afin de trouver l'infrastructure la plus adaptée. Nous exposons les contraintes auxquelles nous sommes soumis et nous expliquons notre choix.

I.2. Transport et Système d'Information

I.2.1. Evolution de l'utilisation des transports en commun

I.2.1.1. Evolution du nombre de voyageurs

Depuis les années quatre-vingt, le déplacement en interne des voyageurs, en France, mais aussi dans les différents pays de l'Europe, augmente progressivement. Les voyageurs parcourent quotidiennement une distance plus longue et dans les déplacements quotidiens (le plus souvent : domicile-travail) et dans les trajets touristiques.

La [Figure I-1] représente une superposition des diagrammes de l'évolution des voyageurs en interne pour la France et en Europe. L'allongement des déplacements est marqué par une certaine stabilité pour la France entre 2004 et 2008.

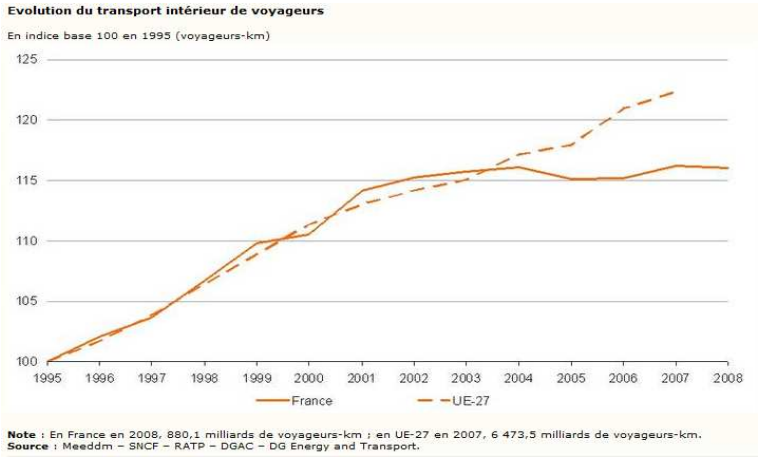

Figure I-1 - Evolution du transport de voyageurs

Pour repérer l'évolution des voyageurs, l'INSEE propose d'utiliser l'indicateur voyageur-kilomètre qui est une unité de mesure équivaut au transport d'un voyageur sur une distance d'un kilomètre.

I.2.1.2. Impact des transports sur l'environnement

La [Figure I-2] représente l'évolution à la hausse des émissions des gaz à effet de serre entre 1990 et 2006.

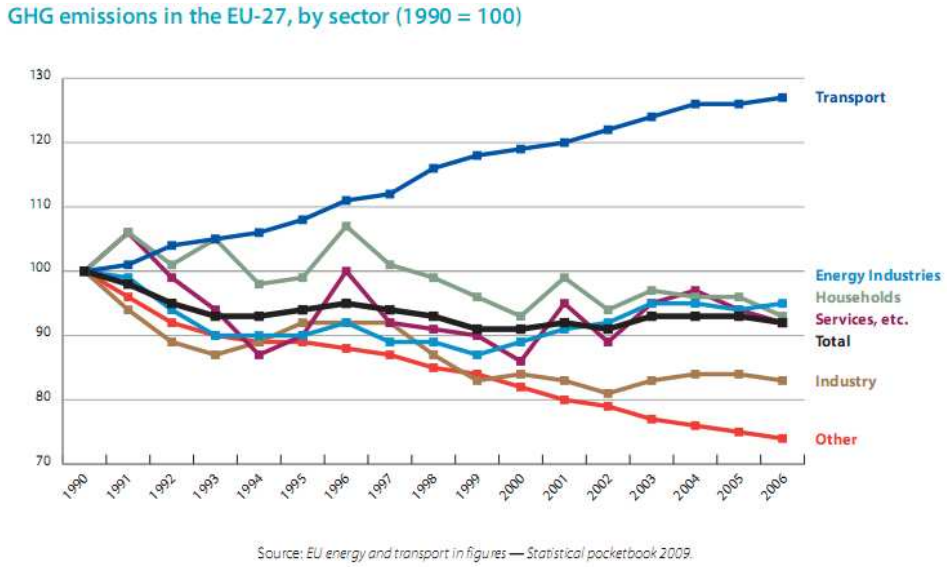

Figure I-2 - Emission de Gaz à effet de serre en Europe

Cette évolution suscite des inquiétudes : l'impact sur l'environnement va à l'encontre d'un processus de développement durable [**40**].

En effet, selon les données de l'Agence pour l'Environnement de l'Union européenne, les transports représentent presque le quart (23,8%) des émissions totales de GES et un peu plus d'un quart (27,9%) du total des émissions de CO2 dans l'UE-27 en 2006. Le plus inquiétant est que ces

valeurs ont tendance à croitre. En outre, bien que le total des émissions soit en légère baisse, les émissions des transports continuent à augmenter.

De plus, le secteur de transport s'appuie à 97% sur les énergies fossiles [**40**] et représente à ce titre 50% de la consommation de produits pétroliers pour l'année 2002 [Figure I-3].

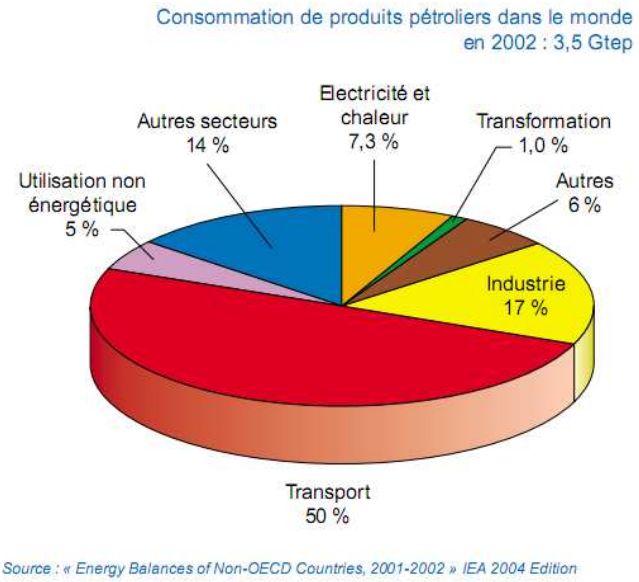

Figure I-3 - Consommation de produits pétroliers dans le monde

I.2.1.3. Répartition du transport intérieur

Les études récentes réalisées autour de la mobilité des Français [**77**] montrent que le transport automobile (voiture particulière) reste le mode de transport le plus utilisé [Figure I-4], suivi par les transports ferroviaires , puis les autobus et autocars.

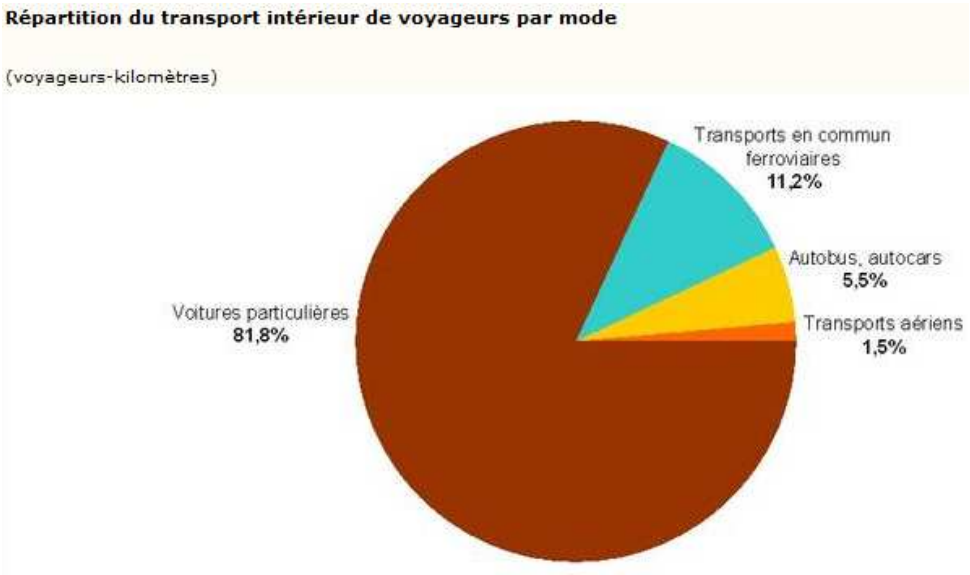

Figure I-4 - Répartition du transport intérieur de voyageurs par mode

La prédominance de la voiture particulière ne représente pas un phénomène émergent dans la mesure où, Dupuy[**38**], en 1999, évoquait déjà une dépendance aux transports automobiles.

Malgré cette nette différence entre l'utilisation de la voiture personnelle et des transports en commun, ce dernier mode présente cinq avantages majeurs[1] :

Consommation d'énergie et pollution : En regroupant plusieurs personnes dans un même véhicule (Bus, Train, Tram …), les transports en commun permettent de faire une réduction d'échelle et donc d'énergie.

Fluidité du trafic : En diminuant le nombre de voitures sur les routes, les transports en commun fluidifient la circulation. L'exemple le plus simple : transport de 60 personnes par bus n'occupe que la surface prise par deux voitures. Or une voiture ne transporte généralement que 1.5 voyageur.

Rapidité : les transports en commun sont plus rapides lorsque les réseaux sont équipés de voies réservées aux transports publics.

Sécurité : Le fait d'avoir des voies réservées au transport en commun, séparées de la route commune et des véhicules, conduits et contrôlés par des professionnels du transport, diminue le nombre d'accidents.

Coût : Le fait d'utiliser les transports en commun permet de réaliser des économies en termes de dépense sur le voyage surtout en comparaison avec un itinéraire d'une voiture personnelle transportant une seule personne.

[1] http://fr.wikipedia.org/wiki/Transport_en_commun#Avantages

La prise de conscience de ces avantages est croissante en France et en Europe surtout vis-à-vis de l'impact sur l'environnement. Cela est visible en analysant l'évolution de leur utilisation depuis les années quatre-vingt-dix.

I.2.1.4. Evolution de la répartition des modes dans le transport intérieure

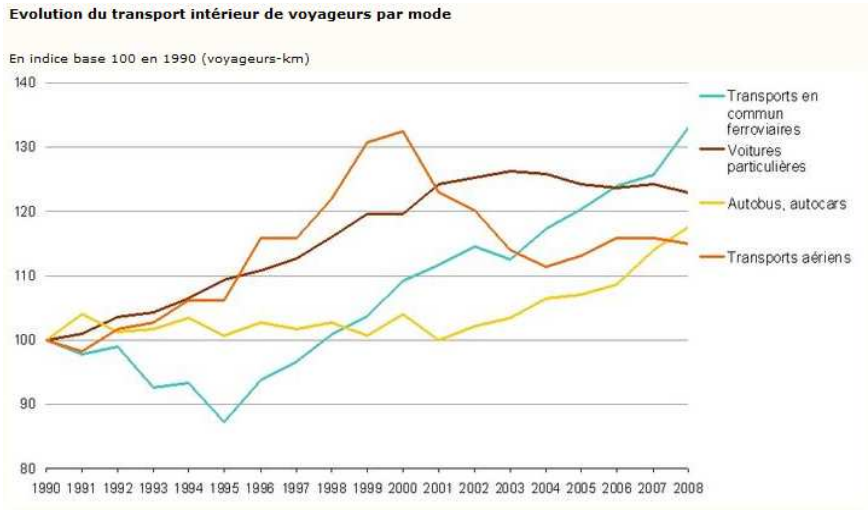

Figure I-5 - Evolution du transport intérieur de voyageur par mode

Sur la dernière vingtaine d'années, on remarque une nette hausse dans l'utilisation des transports en commun contre une légère baisse de l'utilisation des véhicules personnels qui représente une première depuis les années soixante dix. Cette baisse commence à partir des années 2005 et 2006 et qui continue jusqu'à présent [Figure I-5].

I.2.2. Perturbation, irrégularité et impact sur les voyageurs

I.2.2.1. Définition d'une perturbation

Tout incident pouvant affecter un réseau de transport d'une manière aléatoire et qui a comme effet un écart entre le TMT[1] et le tableau de marche réel[**127**] peut être défini comme une perturbation.

Une perturbation se traduit généralement par des irrégularités de circulation modifiant les intervalles entre les véhicules et prolongeant ainsi le temps d'attente aux arrêts. Ce temps d'attente supplémentaire affecte la répartition des charges entre les véhicules et détériore la qualité du service de transport.

[1] Tableau de Marche Théorique

I.2.2.2. Perturbation et irrégularité dans les transports

L' « information voyageur » devient cruciale en cas de perturbation des transports. Plusieurs facteurs peuvent être à l'origine de ces perturbations, Nguyen Duc [**80**] propose de classifier ces perturbations en fonction des facteurs externes et internes [Figure I-6].

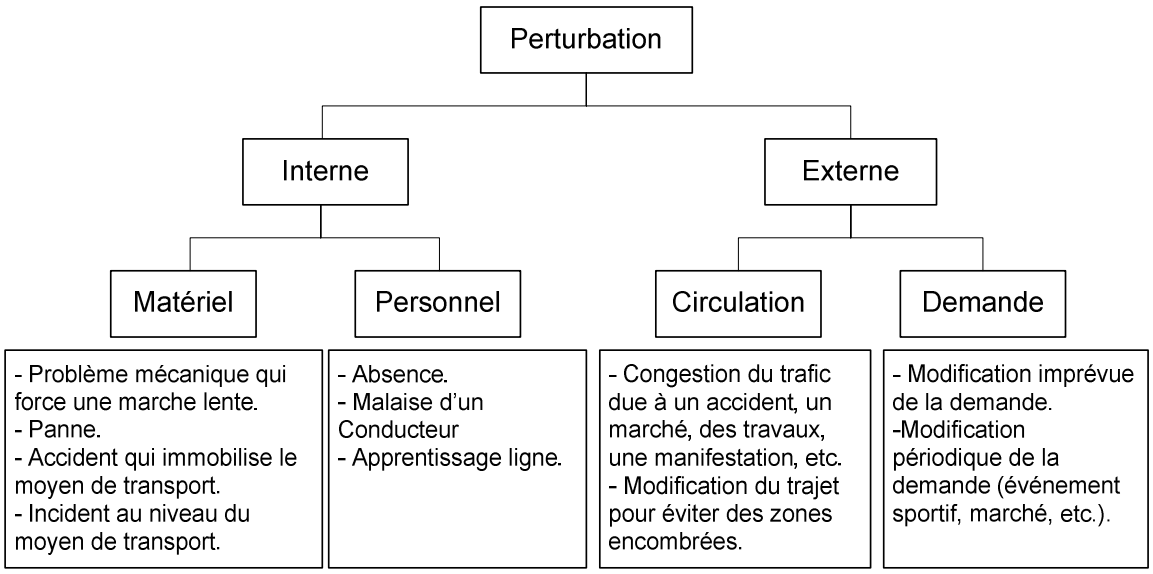

Figure I-6 - Classification des perturbations selon les facteurs [**80**]

Dans sa thèse, COQUIO[**24**] propose une autre classification, plus classique, distinguant les aléas naturels, technologiques et humains. Nous résumons sa classification dans la [Figure I-7].

Pour le voyageur, ces perturbations se traduisent par un temps d'attente plus ou moins long et dans certains cas, par l'annulation du voyage, notamment lorsque la destination n'est plus desservie ou quand le temps d'attente est très long ou inconnu.

Les exploitants de transport utilisent plusieurs indicateurs pour mesurer les perturbations (on parle d'indice d'irrégularité).

Tous les facteurs énumérés dans les deux tableaux de classification précédents sont à l'origine de l'une des perturbations suivantes:

- Interruption d'une ou plusieurs lignes.
- Impact sur les heures d'arrivée.
- Une station n'est plus desservie par une ligne.

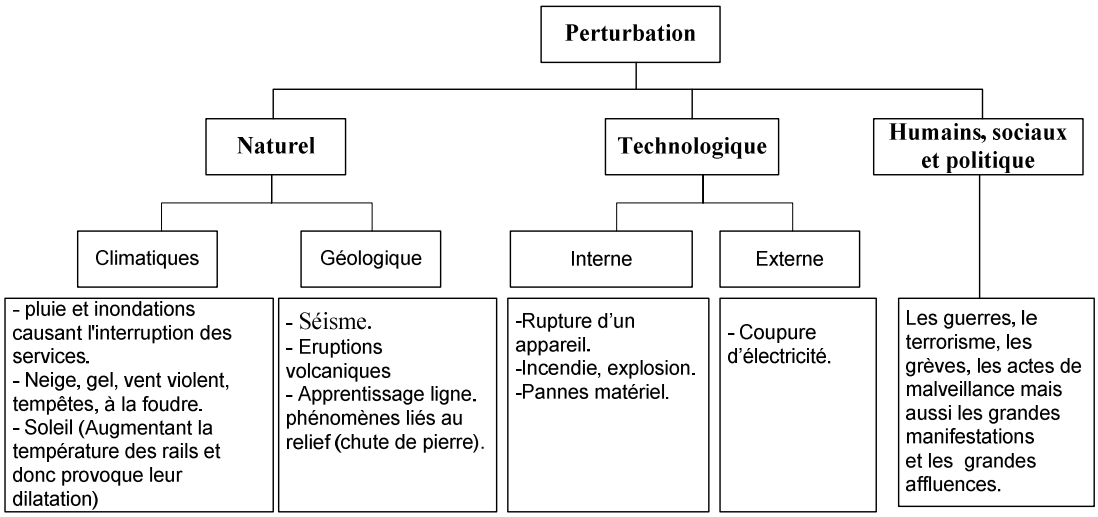

Figure I-7 - Classification des perturbations selon les aléas naturels, technologiques et humaines

Dans la [Figure I-8], nous présentons l'évolution de l'indice d'irrégularité pour la SNCF et la RATP durant la dernière décennie. L'indice d'irrégularité est calculé selon le pourcentage de trains, bus, trams, métros arrivant avec cinq minutes de retard à destination.

On constate que, pour la SNCF et la RATP, cet indice a tendance à croitre et ceci impacte directement sur la qualité du service.

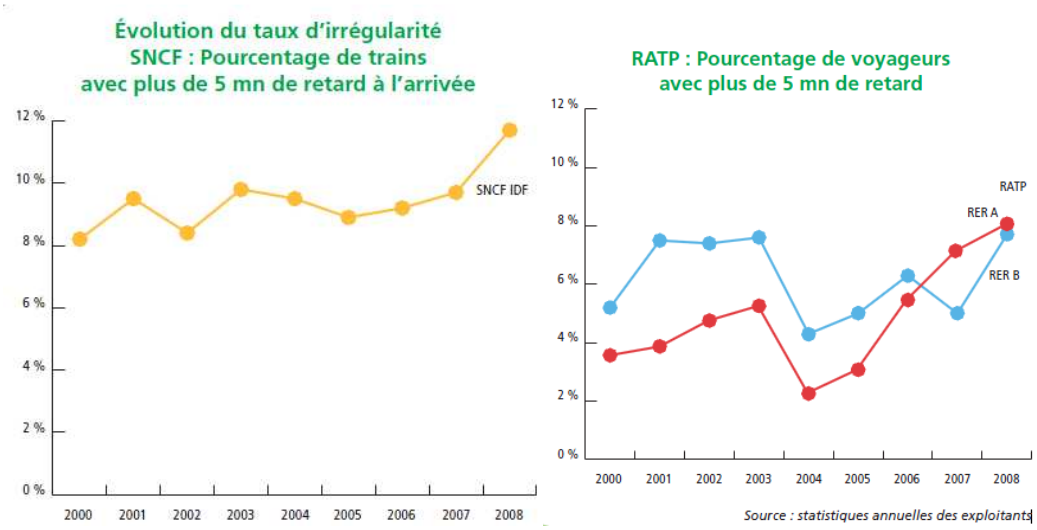

Figure I-8 - Evolution de l'indice d'irrégularité

I.2.2.3. Impact des perturbations sur les utilisateurs

En 2010, le cabinet Technologia[1] a mené une enquête pour évaluer le niveau de stress des salariés du groupe France Telecom. Les conclusions de cette enquête indiquent que les transports en commun

[1] Cabinet spécialisé dans les risques au travail. http://www.technologia.fr/

représentent une source d'inconfort et d'incertitude impactant non seulement la santé mentale des salariés, mais aussi leur santé physique.

Le problème principal vient de l'allongement du temps mobilisé pour le travail dû à des perturbations du transport (trajet domicile-travail ou travail-domicile). En effet, suite à une perturbation, le salarié passe plus de temps pour aller au travail ou pour rentrer. Ce temps décompté des heures de repos du salarié est totalement invisible vis-à-vis de l'entreprise. Le salarié se trouve donc à faire des efforts supplémentaires sans avoir de contrepartie, dégradant significativement l'équilibre entre efforts fournis par le salarié et reconnaissance de l'entreprise. Ce déséquilibre représente un facteur central des risques psychosociaux et donc de stress.

I.2.3. Comodalité et problématique du transport

I.2.3.1. De l'intermodalité à la comodalité

La combinaison de plusieurs modes de transport pour des déplacements donne plus de flexibilité aux voyageurs et rend le réseau de transport plus rentable.

Cette combinaison de modes de transport a donné naissance à plusieurs nouvelles notions dans le domaine du transport. On peut citer l'***intermodalité***, la ***multimodalité*** et la ***comodalité***. Pour bien définir ces trois notions, nous commencerons par définir ce qu'est ***un mode de transport*** et ce qu'est la ***monomodalité.*** Nous définirons aussi l'***opérateur de transport*** et les itinéraires intra-***operateur*** et ***multi-operateur***.

- ***Mode de transport*** : Désigne le moyen utilisé pour déplacer un objet d'un lieu A à un lieu B [**34**]. C'est une forme particulière de transport qui se distingue principalement par le véhicule utilisé[1].
- ***Monomodal*** : Déplacement réalisé avec un seul mode de transport tout au long du trajet.
- ***L'intermodalité*** [**128**][**126**]: qui à l'opposé de la monomodalité. Consiste à utiliser deux ou plusieurs modes de transport pour réaliser un déplacement d'un point d'origine à un point de destination. D'après l'ARENE [**3**], l'intermodalité vise à réduire l'usage de la voiture particulière au profit de transport moins polluant (transports collectifs, vélos …).
- ***La multimodalité :*** Offre de plusieurs moyens de transport pour un déplacement entre une origine et une destination. La Multimodalité est le sujet de plusieurs travaux récents[**128**][**126**][**62**]. En effet, l'offre de transport est un axe important d'amélioration de la qualité de service du transport, en adéquation avec les enjeux environnementaux, économiques et par la suite politiques.

[1] http://fr.wikipedia.org/wiki/Mode_de_transport

- ***La comodalité*** : La notion de comodalité a été introduite en 2006 par la commission européenne dans le domaine de la politique des transports. Elle est définie comme « la combinaison optimale des différents modes sur la chaine de transport » [**41**] représentant ainsi « le recours efficace à différents modes de transport isolément ou en combinaison ». Extrait de l'avis de la commission : « C'est uniquement grâce à une interopérabilité réelle entre les différents modes de transport dans des conditions de marché équitables que peut être obtenue une optimisation naturelle des transports … On ne peut améliorer la situation des transports en Europe qu'en instaurant des conditions équitables pour tous les «modes» de transport, sans en privilégier aucun »[**42**].

  La grande différence entre les notions de « intermodalité et multimodalité » d'un côté et de la comodalité d'un autre, est que la première approche s'inscrit dans le cadre d'une politique de transport opposant les transports routiers (voiture particulière …) aux autres types de transports (transport en commun …) en suscitant la concurrence et en essayant de favoriser principalement les transports collectifs[**41**]. Par contre, la comodalité est plutôt une approche destinée à atteindre un même objectif : l'optimisation des transports. Cette union est au dessus de toute discrimination : il n'y a plus de favorisation des transports en commun par rapport au transport routier, mais plutôt une complémentarité entre tous les modes. On passe ainsi de la concurrence à la complémentarité [**56**]. Le choix s'effectue alors selon la pertinence, le contexte et la situation. D'après Giannopoulos [**55**], l'approche comodale, de la même manière que l'approche multimodale, consiste à développer des infrastructures et prendre des mesures et actions qui assureront une combinaison ***optimale*** des différents modes de transport.

  L'***optimalité*** dans ce contexte concerne différents niveaux d'efficacité : économique, environnementale, commerciale, financière …

- ***Opérateur de transport*** : Organisme chargé de l'exploitation d'un réseau de transport sur une zone géographique.

- ***Intra-operateur*** : (mono-operateur) Un itinéraire se déployant d'un point d'origine à un point de destination est mono-exploitant s'il est géré par un seul opérateur de transport. Un itinéraire peut-être multimodal et mono-opérateur, si plusieurs modes de transport sont nécessaires pour réaliser le trajet et que tous les modes utilisés appartiennent au même opérateur de transport.

- ***Multi-operateur*** : Un itinéraire peut être composé de plusieurs tronçons ; si au moins deux de ces tronçons sont gérés par deux opérateurs différents, on dit que l'itinéraire est multi-opérateur.

I.2.3.2. Information, service et problématique dans le transport

Nous définissons ci-dessous les notions d'« information multimodale » et d' « information voyageur » et les problèmes qui y sont liés.

Information multimodale : Information qui décrit des itinéraires intermodaux et multimodaux [**62**].

Information voyageur : On parle d'une « information voyageur » lorsque l'on évoque le besoin d'information du voyageur. L' « information voyageur » fournit à l'usager (personne ou organisme) les indications (conditions, horaires, etc) utiles et nécessaires au déplacement. Ces informations concernent ce qui se passe avant ou durant le déplacement [**128**][**126**].

Cette information rejoint la définition de l'information multimodale proposée par Uster [**107**] : « C'est l'information qui propose un ou des itinéraires à l'usager des transports, en tenant compte de l'ensemble de l'offre disponible pour une destination donnée et d'offrir une information pertinente et immédiatement opérationnelle. Réduisant son incertitude pendant le voyage ».

I.2.3.2.1. Les services d'information voyageur

Ces services offrent des informations concernant les horaires de parcours et l'état du trafic[1]. Ils peuvent également offrir des services de calcul d'itinéraire, des informations sur les perturbations, les retards ou les évènements indirectement liés au transport lui-même [**126**] comme : les évènements culturels, les informations touristiques…

I.2.3.2.2. Problématique de l'information voyageuse

Le développement de l'information voyageur est confronté à plusieurs obstacles. Des études réalisées par l'INRETS et ITS [**109**] ont relevé les trois principales raisons qui dissuadent les exploitants à utiliser, à investir ou à faciliter (par l'autorisation d'accès aux données) le développement de système qui fournit l'information voyageur [**129**]. Ces trois raisons sont les suivantes :

- La mise en place de tels systèmes coûte trop cher.
- C'est une opération innovante. Elle constitue donc un grand risque notamment en termes de faisabilité.
- Le système ne répond pas à sa stratégie institutionnelle ou commerciale.

Malgré ces obstacles, les industriels et les exploitants sont bien conscients de l'importance de cette information surtout en termes de qualité de service [**79**] qui est en mesure de rendre leur réseau plus attractif [**92**] et donc d'augmenter, par la suite, sa fréquentation[**108**][**107**].

[1] Définition des ITS Journée technique du 10 Décembre 2009

I.2.3.2.3.Le carnet de voyage

Le carnet de voyage est un ensemble d'offres d'itinéraires à partir des horaires de transport. Chaque itinéraire peut être intra-opérateur, multi-opérateur, multimodal, inter-modal ou comodal. Un itinéraire multi-opérateur est, dans la plupart des cas, multimodal. Le carnet de voyage vise en premier lieu à assister l'utilisateur dans la planification de ses déplacements en mettant à sa disposition une panoplie d'itinéraires et en lui facilitant le choix selon ses préférences. Une telle stratégie vise à améliorer l'attractivité des réseaux de transports en facilitant l'accès aux données et en diversifiant les services disponibles.

I.2.4. Système d'information pour le transport des voyageurs

I.2.4.1. Généralités et définitions

Un Système d'Information (SI) est un ensemble organisé d'éléments qui permet de regrouper, classifier et diffuser de l'information sur un phénomène donné [**25**]. Dans le système ISO, un système d'information est un système de traitement de l'information accompagné des ressources organisationnelles associées, telles que les ressources humaines, techniques et financières, et qui fournit et répartit des informations pour les éventuels utilisateurs[**1**].

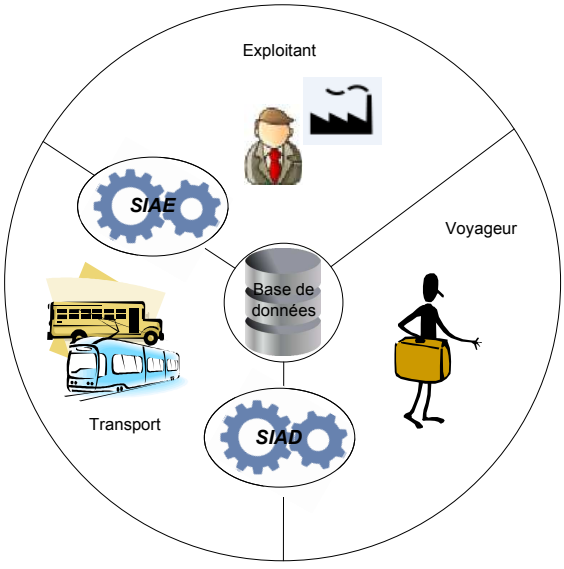

Figure I-9 - Intervenant et SI de transport

Dans le domaine du transport, on met en relation principalement trois intervenants différents [Figure I-9]. D'un coté, les transports : l'infrastructure (réseau de transport) et équipement (Bus, tram train …), de l'autre : les exploitants qui investissent et gèrent les réseaux de transport. Enfin, il faut prendre en compte les voyageurs qui utilisent le réseau de transport dans leurs déplacements. Au cœur de cet environnement, les données du réseau (horaires planification …) sont indispensables.

Sachant que l'information dans le transport est un élément clé, il faut faire un choix entre les multiples systèmes d'information destinés à exploiter au mieux le réseau.

I.2.4.2. SI pour l'Exploitant : SAE/SAD

Un premier type de SI traite de façon différenciée exploitants et transports. Ce sont les systèmes d'informations côté exploitant. En effet, l'exploitation d'un réseau de transport passe principalement par deux phases-clé : une phase de planification et une phase de régulation. On trouve donc des Systèmes d'aide à l'Exploitation (SAE) et des systèmes d'Aide à la Décision (SAD) utilisés pour la réalisation de ces deux phases.

I.2.4.2.1.Phase de planification

La première phase est la phase de planification réalisée en amont de la mise en service du réseau. Elle consiste à concevoir, planifier et ordonnancer les ressources du transport. C'est une phase importante pour les exploitants ; elle est valable pour les véhicules et pour la planification de la main-d'œuvre (conducteurs) [**52**].

Cette problématique est directement liée aux algorithmes d'ordonnancement et particulièrement aux algorithmes de tournées de véhicules (VSP) [**27**]. Actuellement plusieurs entreprises de transport reposent toutes leurs phases de planification sur des outils, logiciels et systèmes de planification pour générer les tableaux horaires (TMT) et les fiches de services des conducteurs [**117**] [**96**].

Parmi les outils informatiques les plus utilisés au sein des compagnies de transport et spécifiquement les transports en commun [**119**][**27**] nous pouvons citer HASTUS développé et distribué par GIRO[1]. Une description complète de l'algorithme utilisé par cet outil peut être trouvée dans [**12**]. Nous trouvons l'outil BUSMAN [**118**] intégrant l'outil IMPACS. Une description des ces algorithmes est disponible dans [**120**]. Un autre logiciel utilisé pour la planification des transports et du personnel est le logiciel « HOT system» [**26**]. C'est l'outil le plus répandu et le plus utilisé dans cette phase du processus.

I.2.4.2.2.Phase de régulation

La deuxième phase survient suite à la mise en place du service de transport. C'est une phase de régulation qui consiste à affiner les horaires, le nombre de véhicules aux heures de pointe pour améliorer la qualité de service et optimiser l'utilisation du réseau.

[1] ttp://www.giro.ca/fr/

I.2.4.3. SI pour Client : Aide au déplacement SIAD

Le deuxième type de SI lie le client (le voyageur) et les transports. Ce sont les systèmes d'information d'aide au déplacement (SIAD). L'objectif principal de ce type de SI est d'aider le voyageur dans la phase de planification en proposant le chemin le plus court ou le moins couteux.

Actuellement, les systèmes d'informations destinés aux clients sont généralement **monomodaux** et dans le cas où ces systèmes sont **multimodaux**, ils ne concernent qu'un seul opérateur et donc sont mono-operateur. Ces systèmes se présentent sous forme de site web. Ils offrent en plus des informations usuelles (d'horaires, de disposition des stations dans une carte) un calculateur d'itinéraire interne. Il s'agit d'un moteur de recherche d'itinéraire accédant aux données locales d'un seul exploitant il fournit donc des itinéraires **monomodaux**. Parmi les exploitants proposant de tels sites on peut citer Transpole[1] pour la métropole Lilloise, RATP[2] pour l'ile de France, TCL[3] pour la ville de Lyon ou encore la TAN[4] qui utilise le calculateur d'itinéraire DESTINEO[5].

Tous ces systèmes demeurent mono-opérateur. De plus en plus de projets sont mis en place dans le but d'intégrer les données issues de plusieurs opérateurs. Les travaux réalisés ont suivi principalement deux stratégies, la première vise à centraliser les données de tous les opérateurs dans un énorme gisement de données et l'exploiter par la suite, la deuxième tend à exploiter les données distantes en créant un système médiateur entre les systèmes existants. Le paragraphe suivant détaille ces deux approches.

I.2.4.3.1. Disposition des Données : centralisées et distribuées

Plusieurs systèmes ont été développés en Europe pour l'intégration et l'exploitation de l'information multimodale. Suivant l'architecture adaptée, deux stratégies d'intégration se dégagent. En effet, on peut procéder soit par l'intégration des applications mises en place par les différents opérateurs, soit par l'intégration des données de ces opérateurs :

– Une architecture distribuée qui vise à intégrer les services existants pour la production de l'information multimodale.

– Une architecture centralisée qui vise à intégrer les données des différents opérateurs.

Dans les deux cas, chaque opérateur dispose d'un Système d'Information d'Aide au Déplacement. Généralement, ces systèmes sont composés de deux parties indépendantes dans leur mise en place, mais fortement liées de par leur exploitation.

[1] www.transpole.fr
[2] www.ratp.fr
[3] www.tcl.fr/
[4] www.tan.fr
[5] www.destineo.fr

En effet, chaque SIAD dispose d'une base de données (BD) sur laquelle il exécute un Algorithme de Calcul d'Itinéraires (ACI) permettant de déterminer le meilleur chemin.

La base de données décrit, via un ensemble d'informations statiques, la globalité du réseau en termes de stations, d'horaires de passages, de tarifs, de lignes et de ressources disponibles. L'ACI exploite cette base de données locale pour trouver le ou les itinéraires qui conviennent au mieux à la requête de l'utilisateur.

Pour produire l'information multimodale, il est nécessaire de procéder à une intégration de l'information locale à chaque opérateur. Deux stratégies d'intégration sont alors à considérer.

La première consiste à intégrer les différentes bases de données dans une même base globale et centralisée et la deuxième consiste à intégrer les différents SIAD existants d'une manière automatique.

- **Première approche :**

La première approche tend à uniformiser toutes les données suivant un seul et unique modèle relationnel. Une telle approche peut être avantageuse si l'on n'exécute qu'un seul ACI global sur une seule base pour la composition d'itinéraires. Cette stratégie implique une uniformisation des données hétérogènes selon un standard donné et entraîne des contraintes relatives à la synchronisation de la base. En effet, il faudra constamment prendre en considération les changements effectués par les différents opérateurs sur leurs bases respectives et mettre à jour la base de données centrale, ce qui dégrade nettement les performances, d'autant plus que cette dégradation est proportionnelle au nombre de bases impliquées et par conséquent au nombre d'opérateurs. Par ailleurs, la mise en place et l'exploitation de cette base peuvent poser un certain nombre de problèmes. En effet, il faudra prévoir assez d'espace pour l'ensemble des données. D'autre part, l'algorithme de composition d'itinéraires aura à interroger une base globale alors qu'au départ il pouvait interroger un certain nombre de bases locales, ce qui augmenterait considérablement le temps de réponse de la base, d'où l'idée de l'approche distribuée.

- **Deuxième approche :**

L'idée de cette approche est d'aller chercher l'information ciblée sur le SIAD correspondant. Une telle stratégie entraînerait des communications supplémentaires et donc, à première vue, une dégradation des performances.

Cependant, les ACI s'exécutent sur des bases de données moins importantes que dans la stratégie centralisée et par conséquent, le temps de réponse de la base est amélioré. Cette approche est mieux adaptée à la parallélisation étant donné que les requêtes s'exécutent sur des bases de données différentes. On pourra ainsi gagner en performance tout en évitant la mobilisation de toute la base. En

effet, si l'on considère plusieurs requêtes d'itinéraires, il est possible d' exécuter ces requêtes de façon séquentielle, car on mobilise la base pour chaque requête. Dans l'approche distribuée, les différentes requêtes peuvent concerner des SIAD différents et par conséquent on pourra traiter plusieurs requêtes en parallèle.

De même, l'approche distribuée ne nécessite pas de synchronisation des données ; elle est donc moins coûteuse.

Dans le paragraphe suivant, nous présenterons quelques projets européens et internationaux concernant cette problématique.

I.2.4.3.2. Projets européens et internationaux

I.2.4.3.2.1. Projet ayant adopté une approche distribuée

Les deux exemples les plus cités dans la littérature sont l'exemple Allemand avec leur système 'DELFI' et l'exemple du Royaume Uni avec le système 'Journeyweb'. Dans les deux cas, il s'agit d'une approche répartie visant l'intégration des systèmes d'information d'aide au déplacement existants.

DELFI[1] : Initié en 1996 en Allemagne, le projet 'DELFI' « Durchgängige ELektronische FahrplanInformation » vise à intégrer les différents calculateurs d'itinéraires locaux des différents opérateurs relatifs aux divers états ('landers') de la République Fédérale allemande.

Cette intégration se base sur des interfaces et des API utilisant le middleware CORBA. Pour déterminer un itinéraire, DELFI commence par déterminer les deux calculateurs locaux relatifs à l'origine et à la destination. Les pôles d'échanges entre les opérateurs sont calculés et fournis par le serveur de l'opérateur National. Un calculateur principal va concaténer les offres recueillies pour composer les différentes alternatives de la requête globale.

Le projet se basant sur des API et des interfaces standardisées, une évolution actuelle vise l'intégration du standard TRIDENT comme norme d'échange des données. Le projet « Netzwerk Direct » est une alternative au projet DELFI en Allemagne qui reprend la même architecture, mais qui se base sur l'infrastructure XML/SOAP pour l'intégration des applications.

JourneyWeb[2] : Le projet 'JourneyWeb' a été initié en 2000 par le ministère de transport public Anglais pour la réalisation d'un système national d'information multimodale.

[1] www.delfi.de
[2] www.journeyweb.org.uk

JourneyWeb utilise un protocole de communication spécialement mis en place pour le projet. Il vise à faciliter l'échange des données entre les systèmes d'information des différents opérateurs. Ainsi, chaque opérateur serait capable de répondre à des requêtes globales relatives à plusieurs opérateurs.

Les données échangées entre les différents SIAD sont en format XML et conformes à un standard local Transexchange.

JourneyWeb exploite deux bases de données:

- NPTG (National Public Transport Gazetteer) : La NPTG est une base de données nationale regroupant l'ensemble des villes et des villages pour les différentes régions. Par ailleurs, elle inclut les pôles d'échange nationaux qui permettent de basculer d'un réseau régional vers le réseau national.
- NaPTAN (National Public Transport Access Node) : C'est aussi une base de données nationale, répertoriant l'ensemble des points d'accès (arrêts et stations) au réseau de transport d'un opérateur local, et ce, pour les différentes régions.

Afin de répondre à une requête d'itinéraire globale, chaque calculateur est capable de rechercher un itinéraire d'un point de départ appartenant à la zone qu'il dessert vers un pôle d'échange national permettant l'accès au réseau incluant la destination. Ceci suppose que chaque calculateur local dispose d'une carte des opérateurs nationaux, information fournie par le NTPG et le NaPTAN. Cette information est ensuite concaténée à l'itinéraire de l'opérateur d'arrivée constituant ainsi une réponse à la requête globale.

I.2.4.3.2.2. Projet ayant adopté une approche centralisée

L'approche centralisée vise à intégrer les données de chaque opérateur dans le système d'information.

Plusieurs projets ont été menés suivant cette approche:

GOFAS : GOFAS est un projet suisse. Il est composé de deux sous-systèmes :

- Infopool qui permet l'importation des données horaires et géographiques à partir des bases de données des différents opérateurs et de les stocker dans une même base de données centralisée couvrant tout le territoire national.
- Internet-Gis qui permet de calculer les itinéraires à partir de la base de données centralisée.

Projet **9292**[1] : Le projet 9292 est un portail national hollandais intégrant les données de plusieurs opérateurs. L'avantage de ce système c'est qu'il renseigne sur d'éventuelles perturbations du service.

[1] www.9292ov.nl

I.2.5. Discussion

Bien que l'utilisation de la voiture soit en régression, elle demeure le mode de transport dominant en Europe. L'impact sur l'environnement est tangible, car elle provoque, en amont, la surexploitation des ressources pétrolières et augmente, en aval, le taux des émissions de gaz carbonique. Une première politique européenne allait dans le sens d'une mise en concurrence des transports en commun vis-à-vis de la voiture particulière en essayant ainsi de diminuer l'utilisation de cette dernière. Mais les limites de cette politique furent vite atteintes. En effet, l'utilisation de la voiture occupe toujours la plus grande part du transport urbain et ceci malgré les différents avantages que proposent les transports en commun. Il est donc plus logique d'aller dans le sens actuel du mouvement qui tend, au lieu de concurrencer les véhicules particuliers, à profiter de leur maniabilité, leur confort tout en préservant l'environnement. Un objectif réalisable si l'on exploite efficacement la voiture particulière, en encourageant, par exemple, une utilisation collective de cette dernière. La notion de multimodalité est dès lors remplacée par celle de comodalité, mise en avant par la nouvelle politique du transport urbain en Europe. En effet, il n'est plus question de mettre en concurrence les transports en commun et la voiture particulière, il s'agit plutôt de trouver une complémentarité entre les différents modes. Cette complémentarité peut se traduire par une composition d'itinéraire utilisant des tronçons en voiture et des tronçons en transport en commun. De plus, afin de minimiser la pollution émise par la voiture, il est nécessaire d'augmenter le nombre de personnes transportées durant le voyage et par la suite réduire le nombre de voitures en circulation.

Une étude des systèmes d'information liée au transport nous a montré qu'il existe deux types de SI, ceux dédiés aux exploitants et ceux utilisés pour orienter les clients. Les SI coté exploitant appelé aussi ***SAE*** n'ont pas particulièrement une influence sur les choix des itinéraires des clients, mais sont plutôt utilisés pour une meilleure gestion du réseau. C'est du côté client qu'il faut chercher. Nous avons recensé plusieurs projets concernant les systèmes d'information et d'aide au déplacement, ces systèmes sont généralement monomodaux ou mono-opérateur. Rares sont les projets multimodaux et multi-opérateurs mais leur nombre est croissant. La fonctionnalité fondamentale de ces systèmes est l'optimisation des chemins au sein du réseau (ou des réseaux concernés). Ils suivent deux stratégies différentes, la première tend à centraliser les données et la deuxième à exploiter les données distantes. D'un autre côté, nous trouvons des systèmes d'information qui encouragent l'utilisation collective et efficace de la voiture comme les sites de covoiturage et d'AutoPartage. Nous avons constaté qu'il n'existe pas de système qui couple l'utilisation de tous les modes de transport (voiture particulière comprise).

Pour aborder ce problème, une étude de la complexité s'impose. Nous commencerons par le choix d'une infrastructure adaptée. En effet, la complexité du problème impose plusieurs contraintes. Il est donc important d'identifier l'architecture qui convient le mieux. Afin de pouvoir porter un choix sur la

mise en place de l'architecture la mieux appropriée, il convient de comparer les architectures systèmes les plus utilisées dans le domaine du génie logiciel.

I.3. Etat de l'art des Architectures Système Complexes

La spécification de notre problème nous impose plusieurs contraintes. Nous essayons donc dans ce chapitre de choisir la technologie la plus adéquate. Dans un premier temps, nous présentons les technologies les plus utilisées dans le domaine du génie logiciel afin de pouvoir les comparer par la suite. Ensuite, nous détaillons les contraintes auxquelles notre système est confronté et nous ferons notre choix d'architecture technologique.

I.3.1. Technologies du Génie Logiciel

La demande croissante en systèmes d'information et en logiciels de plus en plus complexes a créé un nouveau besoin en « architecture système ». L'apparition de nouveaux concepts structurels et architecturaux en génie logiciel a répondu à cette demande. Ainsi, nous disposons actuellement d'une multitude d'outils et de technologies permettant la création d'architectures complexes. Parmi ces concepts, on peut citer les plus répandus: les composants logiciels, les systèmes multi-agents et les services web. Dans ce qui suit, nous analysons les architectures relatives à ces concepts et qui sont aussi les architectures les plus répandues, à savoir : les architectures à base de composants, d'agents et de services.

I.3.1.1. Architecture à base de Composants [4]

Cette technologie est issue des approches orientées objets [**13**]. C'est une réponse à la demande de la réutilisation du code dans le développement des applications informatiques. Depuis les années quatre-vingt-dix, cette nouvelle vision des logiciels informatiques ne cesse d'évoluer [**75**]. Elle consiste à considérer le logiciel informatique comme un ensemble de plusieurs composants (De même qu'un circuit électrique est constitué de composants électroniques assemblés).

Un Composant logiciel est défini selon Szypersky [**105**] comme une entité autonome de déploiement, comportant des interfaces d'entrée et de sortie qui lui permettent d'interagir avec d'autres composants [Figure I-10].

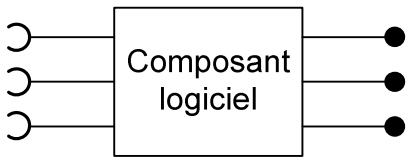

Figure I-10- Composant logiciel

Comme c'est aussi le cas pour l'orienté objet, la programmation à base de composants trouve son efficacité dans la réutilisation[**6**]. La notion de composant vient combler les limites de l'approche objet en terme de granularité de réutilisation en proposant une définition plus précise des entrées et sorties de chaque composant. Assurant ainsi une meilleure sécurité de la réutilisabilité des composants [**105**][**76**]. En effet, contrairement à un objet qui risque de faire des réutilisations non sécurisées (suite à des appels à des services externes sans spécification explicite), un composant ne peut utiliser que des services bien spécifiés compte tenu de sa conception et des précisions sur ses interfaces.

I.3.1.2. Architecture à base d'Agents

Les systèmes à base d'agents tiennent leurs origines dans le développement de l'intelligence artificielle. En effet, le concept d'agent est une évolution de l'intelligence artificielle distribuée née de la fusion entre les systèmes distribués et l'intelligence artificielle [**22**].

Selon Leriche [**71**], un agent est « une entité autonome capable de communiquer, disposant de connaissances et d'un comportement privé ainsi que d'une capacité d'exécution propre. Un agent agit pour le compte d'un tiers (un autre agent, un utilisateur) qu'il représente sans être obligatoirement connecté à lui ».

Pour résoudre les problèmes les plus complexes pouvant représenter une limite des capacités de résolution d'une entité (agent) unique, la mise en commun des connaissances de plusieurs agents et leur coopération permet de franchir cette limite et de donner plus d'élan dans les capacités de résolution. Le concept de coopération dépasse ici la simple addition de connaissance et représente une réelle fusion d'un savoir global qui offre une vue plus synthétique sur les problèmes.

I.3.1.3. Architecture orientée services

L'architecture orientée services (ou Service Oriented Architecture, SOA) est une nouvelle organisation applicative permettant une interaction entre des composants distants de l'application à travers des services. La notion du service peut être représentée ici par un objet produit par un fournisseur et consommé par un client. La SOA propose un nouveau concept facilitant l'échange de messages, réutilisable et avec une bonne sécurité (utilisation de protocoles standardisés).

L'architecture orientée service est d'après Rouillard [**95**] « une manière d'organiser les applications logicielles isolées à travers une mise en place d'une infrastructure accueillant un ensemble de services interconnectés. Chaque service étant accessible au travers de standards et de protocoles d'échange de message » Les architectures orientées services représentent une vision basée sur l'offre et la demande de service. Le service n'étant pas exclusivement un service Web, mais peut-être tout type de service respectant un/des protocole(s) et une description précise mise à la disposition du client. Par exemple les WSDL (pour Web Services Description Language).

I.3.2. Comparaison des Architectures

Pour analyser au mieux les différentes architectures, nous nous réfèrerons aux recherches effectuées par BRIOT [**15**] dans le cadre de son analyse comparative des architectures à base de composants logiciels et les systèmes multi-agents.

Dans son analyse, Briot propose de comparer les différents concepts en se projetant sur un référentiel commun à trois dimensions [**54**], où chaque dimension traite un axe important de la programmation et du logiciel. Ces axes sont les suivants [Figure I-11] : la sélection de l'action, le niveau d'abstraction et la flexibilité du couplage.

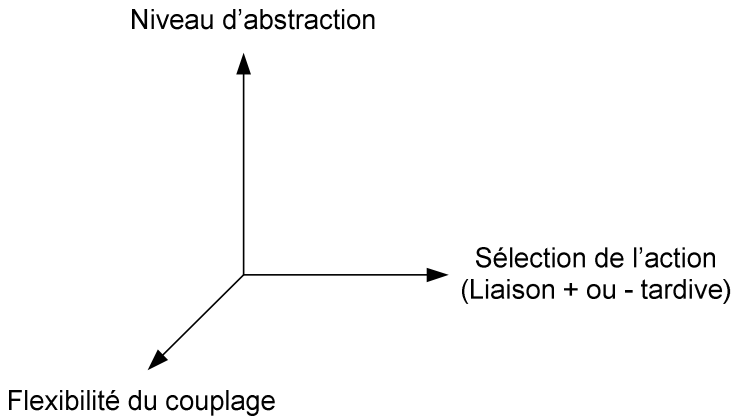

Figure I-11 - Référentiel de comparaison des langages de programmation [**15**]

I.3.2.1. Sélection de l'action

Elle identifie quand et comment l'entité logicielle active l'action du code correspondant.

Globalement, on remarque que « la sélection de l'action » a été repoussée à plus tard dans les compilateurs avec l'évolution des langages de programmation.

En effet, dans les premiers langages (Ex. Fortran), les différentes instructions sont identifiées par leur numéro de ligne. La sélection de l'action est par conséquent exprimée globalement et statiquement. Ensuite sont apparus les langages de programmation structurée ou modulaire, (Ex. Pascal), qui innovent par une encapsulation du code dans des procédures. Cette modularité implique un appel par un nom symbolique et non pas par numéro de ligne. L'invocation n'est donc plus globale et devient externe. Cependant, l'identification de la sélection de l'action demeure statique.

La grande innovation des langages orientés objet dans ce domaine est l'apparition de la liaison tardive (late binding[1]) qui consiste à ce que la méthode à appeler soit identifiée selon son appartenance à un objet. Ainsi la sélection de l'action est repoussée jusqu'à l'exécution et n'est plus identifiée à la compilation d'une manière statique, la sélection est donc externe dynamique.

D'après Briot les composants logiciels n'innovent pas particulièrement en termes de sélection de l'action. La seule différence par rapport à la programmation-objet réside dans le fait qu'un composant n'a pas besoin d'information externe pour son exécution (tout le code et la documentation d'un composant se trouvent dans le composant lui-même [**88**]). Par opposition, pour un objet, une partie de son code se trouve dans sa classe mère. La sélection tend donc à devenir interne, cependant elle est toujours externe et dynamique.

Les agents apportent une grande évolution à la sélection de l'action. En effet, contrairement aux procédures et méthodes qui sont invoquées à un lieu particulier du code, la sélection de l'action dans le concept agent (comportement) dépend aussi du contexte, de l'état et des connaissances de ce dernier. Cette nouvelle autonomie à la sélection de l'action rend l'agent proactif [**85**] en plus de son aspect réactif. Ainsi la sélection de l'action devient interne et dynamique.

I.3.2.2. Flexibilité du couplage

Elle est définie par la capacité à mettre en relation plusieurs entités logicielles. On distingue deux types de couplage : le couplage structurel et le couplage de communication, nous nous concentrons dans le paragraphe suivant sur le couplage structurel.

Le couplage structurel est mis en place dans les langages de programmation à travers les références. En effet, un objet X référence un deuxième Y en mettant simplement la référence de Y dans une variable locale à X qu'on notera Vx. Donc pour changer la Référence de X vers un autre artéfact Z par exemple, il suffit de changer la valeur de la variable Vx en lui affectant Z.

Un problème majeur intervient si on souhaite que X référence en même temps plusieurs artéfacts ($Z_{1...}Z_n$). Dans ce cas, on est obligé de passer par des collections informatiques tel que les listes, les files, les piles... En plus, il faudra changer les instructions communiquant avec la référence vu que ce n'est plus un objet, mais une collection d'objets qui est traitée. Il y aura donc un ajout d'itérateur de collection et ceci nous oblige à ré-implémenter le code de l'objet X.

La nouveauté des portes/interfaces de sorties [Figure I-12] dans l'architecture des composants permet de palier au problème décrit précédemment. Un composant ayant un ensemble de bornes d'entrée et un ensemble de bornes de sorties facilite la manipulation du couplage en l'externalisant [Figure I-12].

[1] Liaison des symboles effectuée au moment du chargement d'un programme, ou au moment de l'utilisation du symbole.

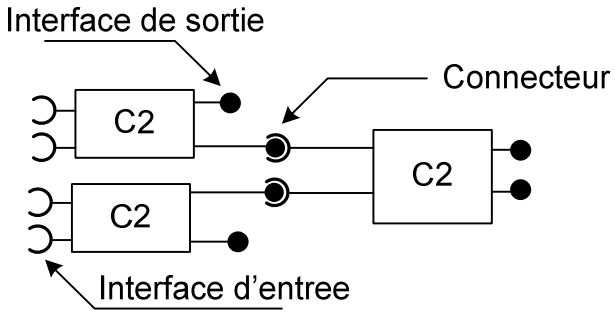

Figure I-12 - Connexion de plusieurs composants

L'architecture SOA et en particulier les web services rendent les couplages des différentes entités applicatives dynamiques à travers la consultation de l'annuaire et les choix du service.

Les systèmes multi-agents intègrent aussi cet aspect dynamique de sélection du service à exécuter vu qu'ils implémentent entre autres des agents annuaires. Ce dernier type d'agent existant dans l'architecture FIPA[1] et permet aux autres agents d'un système de retrouver la liste d'agent fournisseurs d'un service en particulier. C'est un annuaire sous forme d'agent qui facilite la mise en relation avec un agent proposant un service recherché.

De plus, les systèmes multi-agents proposent un couplage sémantique basé sur une organisation sociale et des connaissances.

Ceci fait évoluer les langages de programmation et les logiciels vers un niveau plus élevé en connaissance avec l'utilisation du concept d'organisation.

En outre, la séparation de la description de l'architecture du système d'une part et de l'implémentation d'autre part, représente un grand pas dans l'évolution de la programmation. Ceci est confirmé dans les architectures à base de composant. Les composants étant des entités indépendantes avec une description précise, l'architecture du système assimilé à une carte électronique constituée de composants interconnectés est totalement indépendante de son implémentation. Cette avancée technologique est encore plus développée dans les architectures service grâce à la découverte, le choix automatique et l'invocation des services. D'autant plus chez les architectures multi-agents avec leur fameuse collaboration, organisation, négociation et coordination.

I.3.2.3. Niveau d'abstraction

Il est défini par le niveau d'expression des concepts offerts aux concepteurs et aux programmeurs. On remarque aisément la grande corrélation entre les langages de programmation et le niveau

[1] FIPA (Foundation for Intelligent Physical Agents) Organisation IEEE des standards des technologies à base d'agent et leurs interopérabilités avec d'autres standards. http://www.fipa.org/ .

d'abstraction. En effet, dans les premiers langages de programmation (très proche de la machine) on se basait sur les instructions et les numéros de lignes. Ensuite apparaissent les langages modulaires (procédure, fonctions) avec un degré plus haut d'abstraction, qui continue à augmenter avec l'apparition des structures de données et surtout la programmation-objet qui nous fait évoluer des données vers des concepts via le principe des objets informatiques représentant des objets physiques [**93**].

Les composants n'innovent pas sur cet axe contrairement aux agents qui permettent d'étendre cette évolution vers une vision à base de connaissance. En effet, les agents cognitifs par exemple, apportent la notion d'état mental qui est une modélisation symbolique de concepts cognitifs qui permettent une coordination et une organisation entre les agents.

I.3.2.4. Synthèse

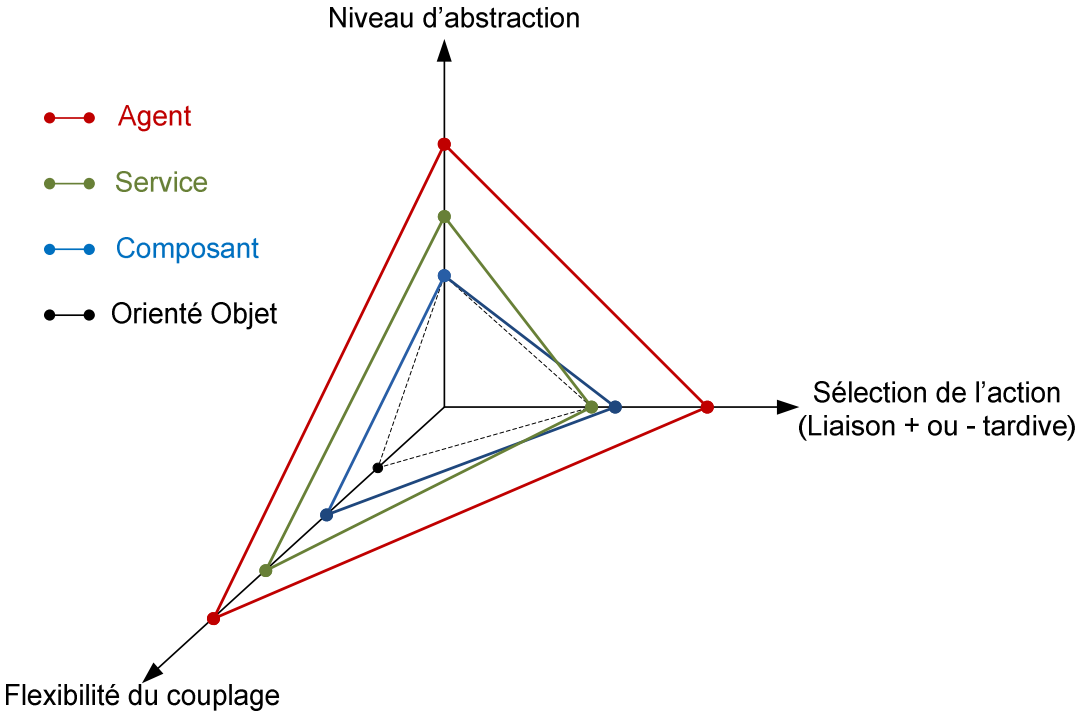

Figure I-13 - Comparaison des technologies

L'architecture Composant est très proche de l'architecture orientée objet, elle présente une légère évolution au niveau de la « sélection de l'action » et « Flexibilité du couplage ». Cependant, son « niveau d'abstraction demeure limité ». L'architecture Orientée Service présente un niveau plus élevé d'abstraction et une meilleure flexibilité de couplage, mais pas d'innovation dans la « Sélection de l'action ». Finalement, l'architecture multi-agent est la plus évoluée sur les trois axes [Figure I-13]. Sa faiblesse existe peut-être ailleurs, au niveau de la mise en place ou des performances.

I.3.3. Contraintes

Afin de générer l'offre globale de transport, notre système doit pouvoir accéder en temps réel aux systèmes d'information existants et s'adapter automatiquement à leur offre de service. Ensuite il faut pouvoir appliquer des algorithmes de recherche d'itinéraire et de régulation complexes tout en gérant l'intégration à chaud de nouveaux SI.

Les contraintes qu'on peut donc dégager sont les suivantes :

- Communication inter-SI : L'architecture de notre système doit pourvoir faciliter l'échange de messages entre le système lui-même et sont environnement. En effet, la base de son fonctionnement réside dans la récupération automatique de données à partir de SI de transport existant et à les transmettre à une entité de calcul. Cette transmission nécessite une forme particulière d'échange de messages entre les entités du système et doit respecter certains standards. D'autre part, la récupération de données à partir des SIAD est aussi une forme d'échange de messages entre les SIAD et le système et ceci doit présenter une certaine forme d'intelligence Cela nous ramène à la deuxième contrainte à savoir l'adaptation aux offres de service des SI existants.

- Adaptation intelligente aux offres de services : L'offre de service proposée par les SIAD diffère d'un SI à un autre. Chaque SI étant décrit dans un fichier particulier, le système doit pouvoir s'adapter aux offres pour en extraire l'information pertinente qui sera utile à la génération de la réponse globale.

- Intégration à chaud des nouveaux SIs : Au cours du calcul, des SIAD peuvent s'arrêter ou redémarrer suite à des problèmes d'ordre informatique ou dû aux perturbations au niveau des lignes de transport physique. Dans tous les cas, notre système ne doit pas être affecté dans sa globalité. En effet, il doit permettre une inscription et une désinscription automatique des SIAD sans avoir à modifier le code source ou même à redémarrer le système. Ceci est fait dans l'optique d'avoir une meilleure fiabilité en cas de perturbation.

- Calcul complexe : La résolution des problématiques de recherche d'itinéraire et de régulation suite à des perturbations nécessite l'application d'algorithmes complexes et une adaptation au calcul et à la résolution de problèmes.

I.3.4. Choix stratégique

Sans surprise, nous remarquons que les trois technologies répondent à la première contrainte à savoir la communication entre SI. Mais leur forme d'échange de messages diffère.

Comme précisés précédemment, nous sommes confrontés à deux types de transfert de message :

Un transfert inta-système : qui consiste dans les transferts de messages entre les différentes entités du système.

Un transfert inter-systèmes : qui consiste dans les transferts de messages entre notre système et les autres SIAD.

Sachant que l'architecture agent est la plus développée en matière de transfert de message puisqu'elle n'a pas besoin de connaitre l'interface de son interlocuteur contrairement à l'architecture composant et service, nous estimons que cette technologie est la plus adaptée à notre problématique. Nous sommes donc tentés de l'utiliser dans les deux formes d'échange de messages de notre système. Cependant, l'inconvénient est que les SIAD n'utilisent pas tous les architectures agent. Ce sont plutôt les offres de services et spécialement les Web Services qui sont les plus utilisés. De plus, pour les SIAD qui n'utilisent aucune des architectures citées, il est beaucoup plus simple de mettre en place une offre de service (WS) permettant l'accès à leur calculateur que de concevoir une architecture multi-agent offrant le même service.

En conclusion, nous préconisons donc l'utilisation de l'architecture multi-agent en interne du système pour l'échange des messages intra-système, et des WS pour l'échange de messages avec les SIAD (inter-systèmes).

Concernant la deuxième contrainte, l'architecture multi-agent est encore la plus adaptée à ce type de problème. En effet, grâce à la souplesse et l'intelligence de ses agents, l'adaptation au autres SI est simplifiée. L'architecture à base de services, de son coté, apporte un dynamisme qui se traduit par la découverte et l'invocation des services qui peut être très utile pour une adaptation aux SI, mais ceci reste très rigide (pas d'adaptation automatique) et nécessite un effort de développement supplémentaire en plus de la difficulté d'intégration à chaud. Ce qui nous ramène à la troisième contrainte.

Notre système doit permettre l'intégration de nouveau SI ou de les désinscrire sans arrêter le fonctionnement global du système. Dans le cadre de l'architecture à base de composants, les nouveaux composants doivent être conçus en amont et ensuite implémentés avec le système ce qui rend l'intégration de composants à chaud difficile. Les agents quant à eux peuvent être implémentés indépendamment les uns des autres et il est, par conséquent, très simple de démarrer ou d'arrêter un agent lorsque le système tourne sans aucun impact sur le fonctionnement global. Ceci est d'autant plus facilité si le système repose sur une architecture FIPA[1] et dispose par la suite d'un agent annuaire.

Enfin, notre problématique nous impose l'exécution d'un algorithme dans un environnement distribué. Dans ce genre de contexte, différentes entités doivent exécuter des instructions indépendamment des autres et communiquer entre elles par la suite pour regrouper les différents résultats dans une réponse globale. Les trois architectures permettent cette approche avec un léger désavantage pour l'approche

[1] Organisation IEEE des standards des technologies à base d'agent et leurs interopérabilités avec d'autres standards. http://www.fipa.org/

service qui par concept n'est pas destinée au calcul algorithmique et d'une manière générale n'est pas l'hébergeur idéal pour une masse de calcul. Les deux autres technologies, quant à elles, sont très utilisées dans la recherche algorithmique distribuée.

Suite à cette analyse, nous avons été conduits à envisager la possibilité d'une fusion de deux des technologies étudiées précédemment : l'architecture multi-agent couplée au Web Services pour une communication externe avec les SIAD.

Un premier croquis de notre système est présenté dans la [Figure I-14]. Le système est formé de plusieurs agents communiquant entre eux avec un langage particulier et le système en lui-même communique avec son environnement (autres SIAD et utilisateur) via des web services.

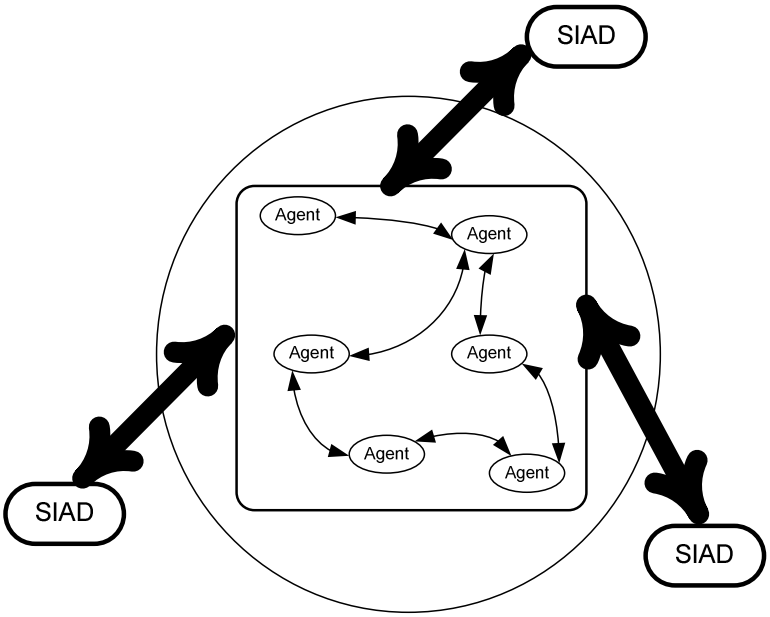

Figure I-14 - Mariage entre une architecture multi-agent et une architecture à base de WS

Comme notre choix s'est porté sur un système multi-agent, nous détaillons dans le paragraphe suivant les propriétés d'un tel système. Nous exposons par la suite le processus de développement et les méthodologies utilisées pour mettre en place un SMA.

I.4. Les Systèmes Multi Agents (SMA)

I.4.1. Généralités et définitions

En 2001 Wilber [**115**] propose une conception permettant l'intégration d'une plus grande partie de connaissance. Ses travaux aboutissent au modèle AQAL (all Quadrants All Levels) qui permet de faire une cartographie des connaissances.

La théorie de la vision intégrale propose de décomposer toute analyse suivant deux axes :
- Un axe de vision mettant en jeu « La perspective individuelle » en opposition à « La perspective collective ».

- Un axe visant « La perspective interne (subjectivité, états mentaux, représentations, ...) » en opposition à « La perspective externe (objectivité, objet, comportement manifesté, ...) ».

La mise en commun de ces deux axes d'analyse constitue une carte en quatre quadrants [Figure I-15] où chaque quadrant correspond à un point de vue spécifique suivant lequel un individu, une situation ou un système social peuvent être analysés.

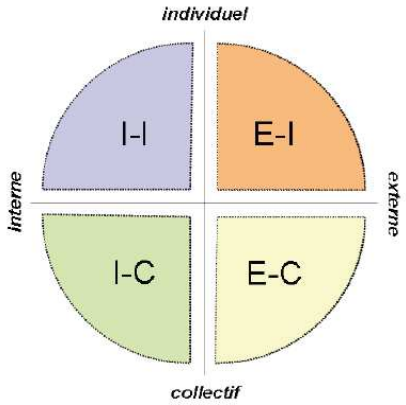

Figure I-15 - Les 4 quadrants [**115**]

Les quatre quadrants sont :

Le quadrant Interne – Individuel (I–I) : Quadrant de la subjectivité. On met en avant le ressenti et le vécu d'un individu qui se traduit par ses états internes.

Le quadrant Externe – Individuel (E–I) : Le quadrant E–I est celui de l'objectivité. On considère l'objet individuellement et indépendamment de ses interactions avec son environnement.

Le quadrant Externe – Collectif (E–C) : Le quadrant E–C est celui de l'inter-objectivité. On considère les relations complexes et interdépendantes qui existent entre les parties d'un système.

Le quadrant Interne – Collectif (I–C) : Le quadrant I–C est celui de l'intersubjectivité. On considère les éléments subjectifs qui sont propres non pas à un individu, mais à un groupe d'individus ou à une société. L'ensemble des éléments subjectifs partagés par les individus d'un groupe constitue la culture de ce groupe.

D'après Wilber, toute chose est constituée au minimum de ces quatre dimensions, il est donc nécessaire pour chaque analyse de se projeter sur ces quatre perspectives pour aboutir à une compréhension intégrale.

En 2008, Tranier applique cette vision intégrale aux SMA et à leur modélisation [**106**]. Il précise que le terme intégral est à prendre dans le sens d'intégrer, de rassembler, de mettre en relation avec l'objectif de trouver une unité dans la diversité et non pas d'uniformiser.

Nous utiliserons ponctuellement l'analyse de Tranier pour bien expliquer certains concepts de SMA et leurs modélisations.

La notion d'agent et de systèmes multi-agents a été citée à plusieurs reprises dans des travaux de recherche dans le domaine du transport. Nous présentons ces notions dans le paragraphe suivant.

I.4.1.1. Définition d'un agent

Plusieurs travaux dans le domaine du transport et du transport distribué ont recensé les définitions d'un agent [**62**][**128**][**126**], la définition la plus récurrente est celle de Ferber[**47**] qui définit un agent comme une entité virtuelle ou physique autonome :

- capable d'agir dans un environnement
- pouvant communiquer directement avec d'autres agents
- mue par un ensemble de tendances
- possédant des ressources propres
- capable de percevoir une partie limitée de son environnement
- possédant une représentation partielle, voir nulle de cet environnement
- possédant des compétences et offrant des services
- pouvant éventuellement se reproduire
- se comportant pour atteindre ses objectifs en fonction des perceptions, représentations et communications qu'elle reçoit et grâce aux compétences et ressources qu'elle possède.

En effet, Feber définit un agent selon son interaction avec son environnement «un agent est une entité capable de percevoir son environnement et de réagir en fonction de ces perceptions ». Ceci permet de ressortir le cycle de vie d'un agent : Perception de l'environnement en utilisant les connaissances et les comportements, prise de décision de l'action suivante (aussi appelée délibération) en exploitant les

connaissances et l'intelligence (cette prise de décision représente l'autonomie de l'agent) et enfin, l'exécution de l'action choisie [Figure I-16].

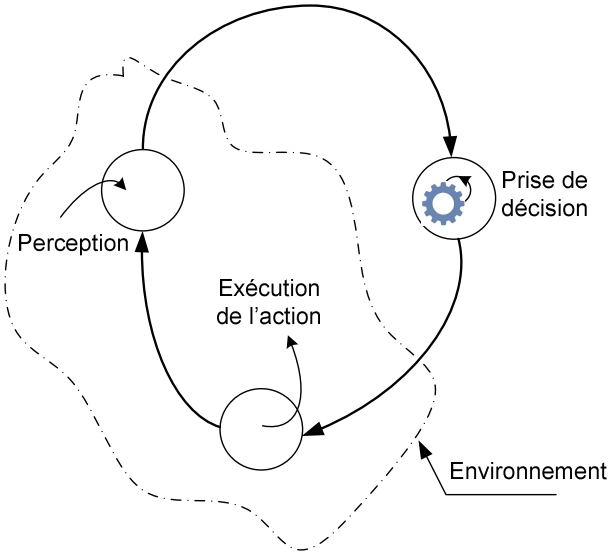

Figure I-16 - Cycle de vie d'un agent

Plusieurs autres définitions existent, mais rejoignent, toutes, des concepts qu'on pourra synthétiser de la manière suivante « Un agent est un système informatique doué de raisonnement et capable de communiquer avec ses semblables et qui agit sur son environnement d'une façon autonome et en fonction de sa perception pour atteindre les objectifs pour lesquels il a été conçu. »

D'après Tranier la structure interne d'un agent contient tous ses états internes et elle ne doit être accessible que par l'agent lui-même. Ceci positionne la structure interne d'un agent dans le quadrant I–I [Figure I-15].

I.4.1.2. Propriétés des agents

L'autonomie : l'agent opère sans l'intervention d'une autre entité et ne subit aucune intervention externe sur les actions propres à lui [**98**].

La réactivité : l'agent perçoit son environnement et agit en fonction des changements qui interviennent.

La pro-activité : l'agent doit avoir des buts internes à réaliser et il doit prendre des initiatives.

La sociabilité : les agents interagissent entre eux et échangent les informations grâce à des langages de communication.

I.4.1.3. Définition d'un SMA

Un système multi-agent est « un système composé d'un ensemble d'entités autonomes et intelligentes qui coordonnent leurs connaissances pour atteindre un objectif ou résoudre un problème » [**114**]. Le concept de système multi-agent a été détaillé dans le paragraphe précédent**Erreur ! Source du renvoi introuvable.**. Un SMA n'est pas une simple addition d'agents, mais une mise en commun de plusieurs intelligences aboutissant à une intelligence beaucoup plus importante que la somme des intelligences mises en commun. Ceci est visible principalement dans la conception où une définition indépendante de chaque agent ne suffit pas pour définir le système complet. L'aspect d'organisation, de communication, d'interaction et d'environnement est d'autant plus important.

Tranier propose de schématiser un SMA dans le diagramme à quatre quadrants en se basant sur les définitions des concepts d'Agent, d'Interaction, d'Environnement, d'Organisation, de Norme et d'Institution [Figure I-17].

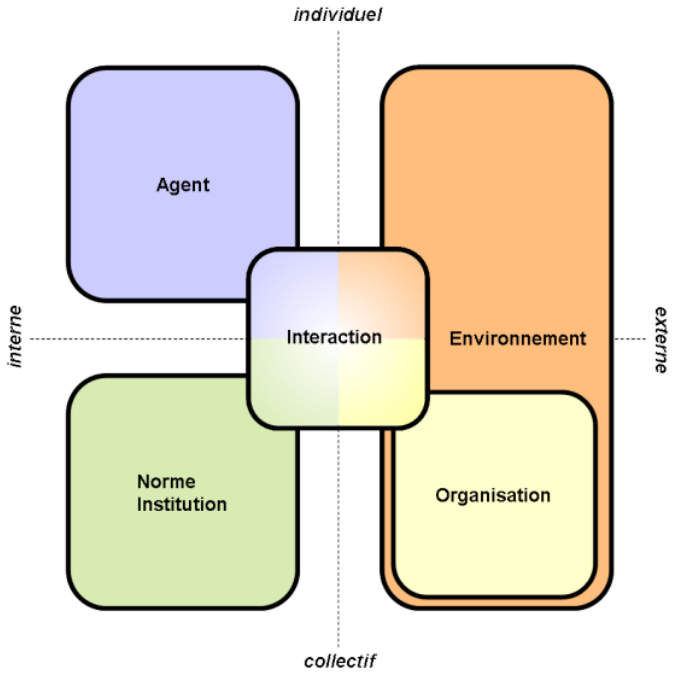

Figure I-17 - Concept des SMA sur les 4 quadrants

I.4.1.4. Organisation et auto-réorganisation d'un SMA

Les systèmes multi-agents sont des systèmes dans lesquels plusieurs entités cohabitent dans un environnement en commun. Cette cohabitation implique l'utilisation d'une organisation particulière qui facilite et canalise la communication entre les entités. Dans cette organisation, chaque agent se verra attribuer un rôle qui définit son degré de collaboration avec les autres agents.

Ainsi, chaque agent joue un rôle et, par conséquent, fournit un ensemble de services au groupe. Une telle organisation permet de décomposer les tâches globales en tâches élémentaires exécutées par l'ensemble des agents.

La puissance d'un SMA réside dans l'intelligence globale du système. Dans un contexte particulier, le changement de l'organisation pourrait faciliter ou accélérer la résolution d'un problème. On dit qu'un SMA est auto-organisé lorsqu'il s'adapte à son environnement en changeant d'organisation.

Ce changement d'organisation obéit à des règles particulières [**74**] et constitue un aspect de la prise de décision du système entier et non pas d'un agent en particulier.

D'après Tranier, les organisations des SMA permettent une structuration du système au travers des notions de groupe et de rôle en éliminant les états mentaux de cette structuration. Les organisations se situent donc au niveau du quadrant E–C. Les aspects normatifs, quant à eux, se situent dans le cadrant I-C (interne – collectif) vu que les normes exercent une pression sociale en interne sur le groupe tout en préservant l'autonomie des membres [Figure I-17].

I.4.1.5. Environnement

L'environnement d'une entité se définit comme tout ce qui lui est extérieur et qui interagit avec elle. Tout d'abord, il faut préciser que l'environnement d'un SMA est inclus dans l'environnement de l'agent lui-même. En effet, l'environnement d'un agent est constitué de l'environnement du SMA en rajoutant les autres agents.

L'environnement d'un SMA n'est pas constitué uniquement de l'environnement de déploiement. En effet, durant plusieurs années, on considérait à tort que l'environnement d'un SMA est constitué uniquement des serveurs, de la plateforme de déploiement, du réseau de communication et de tout le matériel utilisé pour exploiter les agents. Or tout ceci ne constitue qu'une partie de l'environnement réel.

Viroli [**110**] explique que l'environnement du SMA représente une dimension de conception à part entière qui peut intégrer une grande partie de la complexité du système principalement en terme de service. De ce fait, l'environnement du SMA contient aussi tout type de système, objet, service en interaction avec le SMA lui-même.

Tranier explique que l'environnement conserve une réalité commune aux agents du SMA et représente le support de l'activité des agents et par la suite la structure de mise en relation des agents. L'environnement correspond donc aux deux quadrants externes (E–I et E–C) [Figure I-17].

I.4.1.6. Interaction et communication

Les interactions sont très importantes dans les systèmes multi-agents. En effet, pour pouvoir modéliser ses comportements sociaux, il faut un minimum d'interactions entre les différents membres.

La communication est l'un des aspects d'interaction qui ne se limite pas à l'échange de messages, mais aussi à l'échange de données et d'objets selon des règles, des protocoles et des ontologies.

Ferber [**47**] définit les interactions comme la mise en relation dynamique de deux ou plusieurs agents par le biais d'un ensemble d'actions réciproques.

Tranier, quant à lui, précise que l'interaction concerne l'ensemble des quatre quadrants vu qu'une interaction démarre toujours d'une manière individuelle (I-I). De plus, l'interaction dépend des environnements (E–I et E–C), de la structure et donc de l'organisation (E–C). Enfin, les interactions peuvent refléter une certaine culture ou norme (I-C).

I.4.2. Processus de Développement d'un SMA

Une fois l'architecture du système choisie, nous nous intéressons au processus de développement de notre système.

Différentes étapes composent le processus de développement d'un SMA. On trouve, aussi, une similitude dans le processus de développement des systèmes orientés objet et celui des SMA. En effet, dans les deux cas, ce processus est composé de quatre phases [**81**]. Les trois premières sont pratiquement équivalentes aux systèmes usuels: il s'agit des phases d'analyse, de conception et de développement. La différence principale réside dans la quatrième phase [Figure I-18]. En effet, celle-ci correspond -dans un processus orienté objet- à la phase de maintenance et donc à une phase de correction d'anomalies et d'intégration des évolutions souhaitées par l'utilisateur. Dans les SMA, nous remplaçons cette dernière phase de maintenance par la phase de déploiement. Comme décrit par Demazeau [**33**][**94**] et repris par la suite par Laichour [**70**] cette phase de déploiement est nécessaire dans la conception des SMA pour contenir l'évolution des agents durant l'exécution. Plus précisément, il s'agit d'ajouter, modifier ou supprimer des agents durant le fonctionnement de l'application.

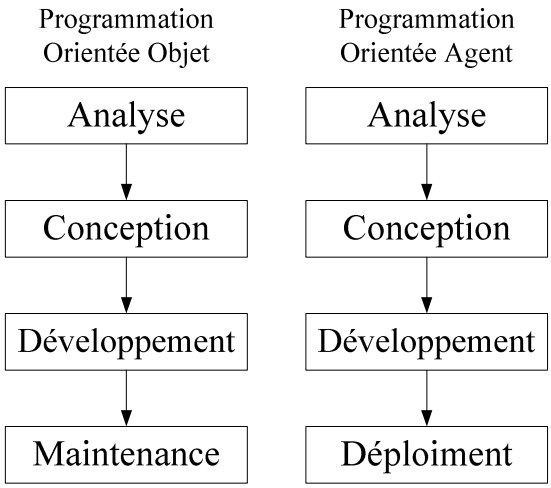

Figure I-18 - Processus de développement Orienté Objet et SMA

I.4.3. Méthodologies SMA

La modélisation des SMA est encore un sujet de recherche très ouvert, plusieurs méthodologies et approches ont été proposées, mais on n'a toujours pas fixé une méthode standard de développement malgré la multitude de travaux qui tentent de standardiser les méthodologies [**99**][**100**][**101**]. Néanmoins, les différentes méthodologies ont été classifiées et comparées, facilitant ainsi au concepteur de choisir la méthodologie qui correspond au mieux à ses besoins.

Parmi ces travaux de classification, nous pouvons citer ceux de Lahlouhi [**69**] et de Laichour [**70**] et principalement les travaux de Bauer et Mueller [**8**] qui proposent une table de classification selon les bases, les phases, les appuis et la zone d'application.

On distingue principalement trois groupes [Figure I-19] :

- Méthodologies basées sur l'orienté objet.
- Les méthodologies à base de connaissances.
- Les méthodologies fondées sur l'Orientation-Agent.

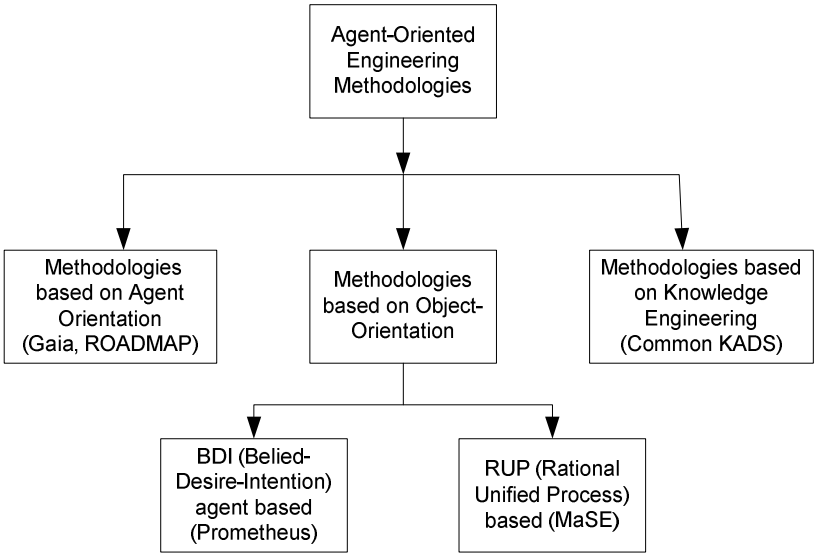

Figure I-19 - Classification des méthodologies Multi-agent

Nous présenterons ci-dessous une brève description des différentes méthodologies et leur classification. Nous expliquerons ensuite notre choix de méthodologie.

I.4.3.1. Les méthodologies fondées sur l'orienté Agent

Ce type de méthodologie est caractérisé par une orientation sur la capture de l'abstraction du niveau social [**113**].

I.4.3.1.1.Gaia

Fondée principalement sur une abstraction organisationnelle, GAIA [**124**][**116**] considère un SMA comme étant composé d'un ensemble d'entités autonomes où chaque entité (agent) joue un rôle défini par quatre attributs (responsabilités, permissions, activités et protocoles). Deux phases composent ce processus : phase d'analyse qui produit deux modèles : le modèle de rôles et le modèle d'interactions et la phase de conception qui transforme les modèles abstraits de la phase d'analyse en modèles moins abstraits modèle d'agent, modèle de service et modèle de relation [**124**].

Gaia repose principalement sur cinq modèles :

Modèle d'organisation : donne une vision hiérarchique du système en décomposant chaque module en des composants plus petits. Cette décomposition est basée sur les objectifs des composants et la fréquence de leur interaction.

Modèle d'environnement : modélise l'environnement selon ses ressources.

Modèle préliminaire de rôle : ce modèle propose une définition préliminaire des rôles et des protocoles de l'organisation. Un rôle étant défini par les permissions (relation de l'agent avec son environnement) et la responsabilité incluant les états d'un agent et la sécurité d'un agent.

Modèle préliminaire d'interaction : représente les relations entre les différents rôles.

Modèle de règle organisationnelle : définit comme étant la responsabilité de l'organisation.

I.4.3.2. Les méthodologies à base de connaissances.

La puissance de ce type de méthodologie réside dans la modélisation de l'état mental des agents. Seulement elle ne propose pas une modélisation du comportement social des agents. La Méthodologie CommonKADS[**102**] est la référence dans ce domaine en Europe, nous présentons dans la suite la méthodologie MAS-CommonKADS qui est une extension à cette dernière.

I.4.3.2.1.MAS-CommonKADS

La méthodologie MAS-CommonKADS (Multiagent System – Knowledge Analysis and Development System) [**59**] est une extension à la méthodologie d'ingénierie des connaissances CommonKADS [**102**] qui combine des techniques de l'orientée objet (comme la Object Oriented Software Engineering (OOSE) [**60**]) et les méthodologies d'ingénierie de protocole (comme Specification and Description Language (SDL) [**17**]) [**16**].

La méthodologie consiste en l'élaboration de sept modèles. Chaque modèle étant composé des entités à modéliser et de leurs relations. Les modèles construits sont les suivants :

Modèle Agent, il décrit les caractéristiques de chaque agent, capacités de raisonnement, compétences, hiérarchie…

Le modèle de travail décrit les tâches que les agents effectuent et les méthodes de résolution de problèmes.

Modèle d'Expertise décrit les connaissances requises par les agents pour atteindre leurs objectifs.

Modèle d'Organisation : décrit les relations structurelles entre agents.

Modèle de coordination décrit les conversations entre agents: leurs interactions, les protocoles ;

Modèle de communication interactions humain-agent logiciel.

Modèle de conception, affine les modèles précédents et détermine l'architecture la plus appropriée pour chaque agent.

I.4.3.3. Méthodologies basées sur l'orienté objet

Ce type de méthodologie est caractérisé par une tendance à étendre les techniques orientées objet. On peut distinguer deux orientations dans ce type de méthodologie :

I.4.3.3.1.BDI (Beliefs, Desires, Intentions)

Comme son nom l'indique repose sur[1] :

Les ***croyances*** (Beliefs) qui reflètent les connaissances que peut avoir un agent sur l'univers auquel il appartient.

Les ***désirs*** (Desires), ou options, qui représentent l'ensemble des opportunités offertes à l'agent et sont générés à partir des croyances de l'agent à un moment donné et des objectifs à plus long terme que l'agent a pu se fixer.

Les ***engagements*** (Intentions), qui sont les options retenues par l'agent. Elles mènent à une action.

I.4.3.3.1.1. MMTS (A Methodology and Modeling Technique for Systems of BDI agents) [**67**]

Elle propose deux niveaux pour modéliser les agents

- INTERNE : Basé sur trois modèles (modèle de croyance, modèle d'objectif et modèle de plan) ce niveau permet d'identifier les classes d'agent BDI.
- EXTERNE : Basé sur deux modèles (modèle d'agent et modèle d'interaction) permet de décomposer le système en agent et de définir leurs interactions

Une autre méthodologie qui appartient au type BDI est la méthodologie Prometheus [**89**].

I.4.3.3.2.RUP [**68**]

RUP (Processus unifié) est une méthode incrémentale et itérative de développement des logiciels orientés objet. Parmi les méthodologies qui s'inscrivent dans ce type, on peut citer :

I.4.3.3.2.1. MASE (Multiagent Systems Engineering) [**31**]

Cette méthodologie couvre l'étape de conception et le début de l'étape d'implémentation à travers deux langages AgML[2] (Agent Modeling Language) et AgDL (Agent Definition Language). Elle utilise les techniques d'OMT[3] (Object Modeling Technique) et UML[4] (Unified Modeling Language) en les adaptant aux spécifications des agents. D'après SCOTT [**29**], La méthodologie MaSE était à la base conçue pour développer des systèmes multi-agents en général. Mais par la suite, cette méthodologie a été utilisée dans tous les domaines (systèmes robotiques, coopératives et anti-virus…). Quatre étapes principales composent cette méthodologie : Conception du domaine, conception d'agents, conception de composantes et conception du système. La méthodologie MaSE n'est pas en mesure de gérer le

[1] http://fr.wikipedia.org/wiki/Mod%C3%A8le_cognitif#Les_mod.C3.A8les_BDI
[2] http://en.wikipedia.org/wiki/AGML
[3] http://fr.wikipedia.org/wiki/OMT
[4] http://fr.wikipedia.org/wiki/Unified_Modeling_Language

dynamisme des systèmes multi-agents. En effet, elle ne propose pas une modélisation de l'entrée/sortie d'un agent dans un SMA dynamiquement (au cours de l'exécution du système)[**29**].

I.4.3.3.2.2. O-MASE

O-MaSE [**30**] (Organization – based Multiagent System Engineering) est une extension à la méthodologie MasE qui la complète par la dimension d'organisation. O-MaSE considère un système multi-agent comme une organisation sociale. Chaque agent étant un membre de cette organisation et joue un rôle spécifique en fonction de sa capacité.

Elle est composée principalement des modèles (de But, d'Organisation, de Rôles, d'Ontologie, d'Agent, de Protocole et d'Etat d'agent)

I.4.4. Choix d'une méthodologie

Les méthodes les plus utilisées sont celles fondées sur l'orienté objet, comme MASE et Promoetheus. En fait, ces méthodologies couvrent les phases d'analyse, de conception et, pour certaines d'entre elles, la phase d'implémentation au moyen d'outils spécifiques.

Plusieurs systèmes multi-agents conçus pour le transport public et la recherche d'itinéraire ont été développés en utilisant la méthodologie MaSE. Balachandran [**5**] par exemple, expliquent leurs choix par la simplicité de la méthodologie.

Nous sommes également tentés d'utiliser cette méthodologie, car, outre sa simplicité, elle propose un cycle de vie qui couvre une grande partie du processus de développement.

Toutefois, cette méthodologie a de limites et risque de nous bloquer durant le processus de développement. En effet, la méthodologie MaSE n'est pas en mesure de gérer la mobilité et le dynamisme d'un système [**32**]. En d'autres termes, il n'est pas possible de modéliser l'apparition et la disparition des agents dynamiquement dans le système. Or dans notre système, une perturbation peut apparaître et disparaître dynamiquement. L'agent qui gère cette perturbation doit, donc, faire de même. De plus, des SIAD peuvent être intégrés au cours de l'exécution. Cette intégration se traduit par la création d'un agent.

La méthodologie O-Mase qui est une extension de la Méthodologie MaSE vient combler cette lacune en rajoutant une autre dimension permettant la gestion des systèmes dynamiques.

Nous avons donc choisi la méthodologie (O-Mase) pour développer notre système. Comme décrit précédemment, cette méthodologie permet de concevoir un système multi-agent selon une vision organisationnelle en se basant sur des méta-modèles.

I.4.5. Les phases de la méthodologie O-MaSE

L'un des plus grands avantages de cette méthodologie réside dans sa large couverture du processus de développement des systèmes multi-agents. En effet, elle permet de suivre l'évolution du MAS dès la définition des objectifs jusqu'à la conception et l'implémentation finale de l'application [Figure I-20] et donc d'augmenter et de faciliter l'industrialisation des systèmes multi-agents.

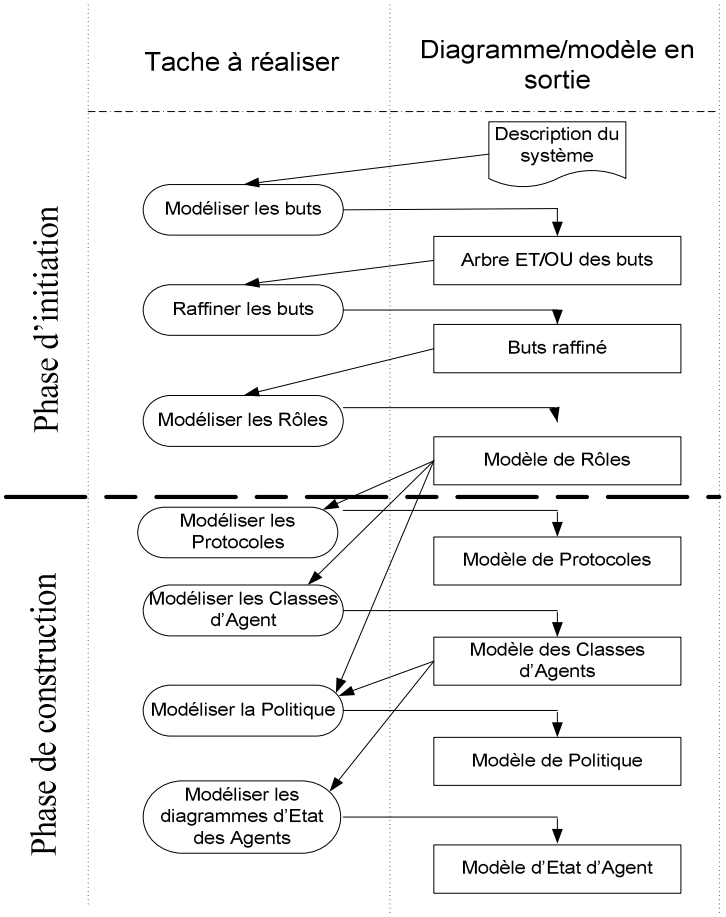

Figure I-20 - Processus de modélisation O-MASE

Cette méthode se décompose en deux phases principales : La phase d'Initiation et la phase de Construction.

I.4.5.1. Phase d'initiation :

La phase d'Initiation permet de fixer les buts/objectifs du système et de définir les rôles qui permettent de les atteindre.

La phase d'Initiation est composée à son tour de deux étapes : l'étape de collecte et de spécification des besoins (Besoin) et l'étape d'analyse (Analyse).

La première étape de cette phase est l'étape de définition des besoins :

I.4.5.1.1.Définition des Besoins

Cette étape est composée de deux sous-étapes :

La Capture de buts :

C'est une étape de collecte d'information et de définition des besoins, elle prend en entrée les spécifications et les exigences du système et donne en sortie un diagramme structuré de but. Cette étape est réalisée en créant des scénarios à partir des exigences qui serviront à l'identification des objectifs. On définit donc l'objectif principal et tous les sous-objectifs du système. Une fois les objectifs définis, une étape de structuration sera réalisée. Elle permet de mettre les buts déjà définis dans un arbre de buts, qui se traduit par un diagramme de hiérarchie représentant les buts et les relations sous-buts.

Application du Cas d'Utilisation :

On traduit les buts définis en diagramme de cas d'utilisation bien détaillés. Les cas d'utilisation décrivent une séquence d'évènement représentant le comportement désiré du système. On crée au moins un seul diagramme de séquence. Si plusieurs scénarios sont envisageables, on doit en créer plusieurs.

I.4.5.1.2.Analyse

Pour chaque objectif identifié dans les étapes précédentes, on associe un rôle. Chaque rôle sera composé de tâches conçues pour atteindre un but et ses sous-buts (s'il en a). Ceci est réalisé via des diagrammes d'état – transition. Qui permettront de décrire les rôles et les relations entre eux (diagramme de Rôle). Ensuite, via des notations UML[1] standard, on génère un diagramme de domaine qui permet de décrire les relations entre les différentes entités, les attributs et l'organisation générale.

I.4.5.2. Phase de construction :

C'est l'étape de conception, elle vise à mettre en place un système pouvant concrétiser les rôles et les buts définis dans les étapes précédentes. En effet, elle permet de modéliser les agents (en leurs affectant les rôles) et permet de définir les types de communication et protocoles liant les différents intervenants du système. Sept sous-étapes constituent cette phase :

Construction du modèle d'Agents

Le diagramme d'agent permet de décrire les agents, l'attribution des rôles, les capacités de chaque agent et enfin les services fournis.

[1] http://fr.wikipedia.org/wiki/UML

Le modèle d'agents est un diagramme de classe d'agents similaires au diagramme de classe de l'orienté objet. Les attributs d'un agent sont les rôles que peut jouer ce dernier et les relations entre les agents représentent des conversations.

Construction du Modèle de protocole

Un protocole liant deux agents est décrit par un diagramme de séquence qui représente les différentes interactions entre les deux entités en précisant l'ordre des messages échangés.

Construction du modèle de plan

Un modèle de plan d'agent décrit l'évolution d'un agent au sein du système. Le diagramme de plan est un diagramme d'état-transition basé sur la notation UML. Pour chaque état, on définit les conditions d'entrée et de sortie, l'état initial, les états finaux et les comportements.

Construction du modèle de politique

Le modèle de politique définit un ensemble de règles formelles qui précisent la manière selon laquelle l'organisation doit évoluer dans une situation particulière. Ce modèle permet d'identifier les règles et contraintes du système et les traduit par des autorisations ou interdictions de jouer des rôles/combinaison de rôles dans des situations particulières.

Construction du modèle de Capacité

Le modèle de capacité est représenté par un diagramme de classes UML Figure I-21. Il est défini par des capacités atomiques et des capacités d'action. Par ailleurs, le diagramme de capacité peut comporter trois types de catégories: Action, Protocole, et plan.

Une capacité d'action est une capacité atomique qu'un Agent peut posséder. Une capacité de protocole est un ensemble d'étapes et de conditions qui sont exécutées sous certaines actions.

Un plan capacité est un ensemble de capacités d'actions et des protocoles qui sont définis pour atteindre un ou plusieurs buts.

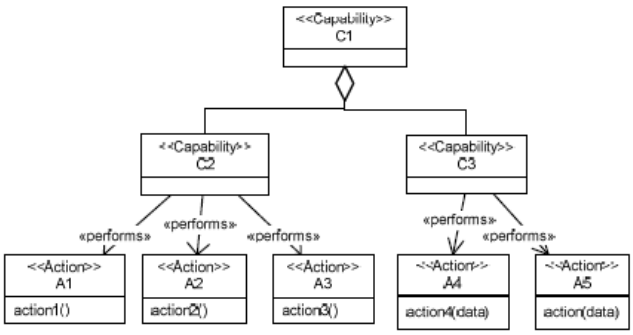

Figure I-21 - Exemple de modèle de capacité

Construction du modèle d'action

Le modèle d’action est constitué d’un diagramme de classes UML et un document décrivant les conditions. Le diagramme de classes UML est utilisé pour décrire les différentes actions qu'un agent peut effectuer. Le document, quant à lui, précise les pré et post-conditions pour chaque action.

Construction du modèle de service

Le modèle de service est un diagramme UML schématisant les grandes lignes des caractéristiques et comportements des rôles exécutés par les agents.

I.5. Conclusion

Nous avons exposé ci-dessus la problématique actuelle du transport des personnes et de l’«information voyageur ». Nous avons ensuite analysé les architectures systèmes disponibles pour sélectionner celle qui répond au mieux aux contraintes de notre problématique. Avons-nous ainsi choisi d’utiliser une architecture SMA et des web services comme interface de communication avec l’environnement extérieur. Dans le chapitre suivant, nous détaillons la conception de notre système et nous expliquons les fonctionnalités des différentes entités qui le constituent.

II. Conception d'un système d'aide au déplacement et de gestion des perturbations

II.1. Introduction

Comme décrit dans le paragraphe (I.3.4), notre système consiste en une fusion entre une architecture multi-agent et une architecture orientée service. La première gère les algorithmes de recherche et d'optimisation d'itinéraires et la seconde prend en charge la transmission de messages entre le système de calcul d'itinéraire et son environnement extérieur. Par ailleurs, un troisième niveau sera nécessaire pour rendre l'information exploitable par un utilisateur final (cf. Figure II-1). Ce dernier niveau se charge d'adapter les requêtes et les réponses au format de l'interface de l'utilisateur (Ordinateur, PDA, téléphone …) et d'effectuer les réglages nécessaires. Dans les paragraphes suivants, nous détaillons la conception de chaque niveau et leurs fonctionnements.

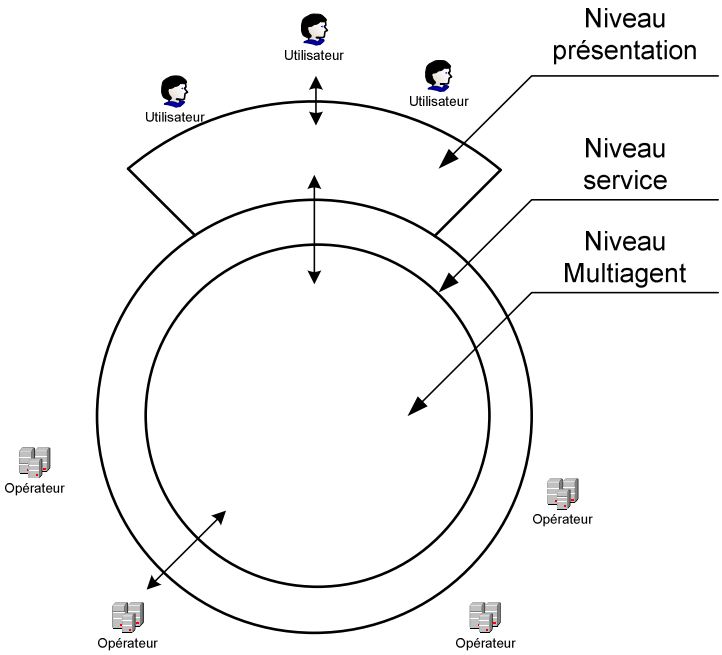

Figure II-1 - Architecture à trois niveaux

II.2. Conception du SMA[43]

Pour concevoir notre système multi-agent, nous nous référons à la méthodologie O-MaSE (comme décrit dans le paragraphe [I.4.4]). Nous commençons donc par exposer notre problématique : l'objectif principal est de guider au mieux un voyageur pour son déplacement en trafic normal, mais aussi lors d'une perturbation dans les transports. Une des difficultés réside dans le fait que les bases de

données des différents opérateurs de transport sont totalement éparses et isolées. Il faut prendre en compte cette contrainte dès l'étape de définition des besoins dans la phase d'initiation.

II.2.1. Diagramme de buts

Notre objectif est de construire le diagramme de buts : pour cela on utilise la méthode de « décomposition ET/OU » qui consiste à identifier les sous-buts liés à un but parent et d'identifier s'ils sont nécessaires ou s'ils représentent des solutions alternatives pour un même objectif.

Dans notre cas, le but global du système est de guider le voyageur dans ses déplacements, et ce, même en cas de perturbation dans les transports. Cet objectif représente le but ultime, noté (but0). Le but0 dépend de la réalisation de deux buts fils qui sont : la recherche des itinéraires selon les besoins du voyageur pour mieux l'orienter (but1) et la gestion des perturbations éventuelles (but2).

Le but1 dépend à son tour de la réalisation de plusieurs buts fils. En effet, afin de construire un itinéraire, on doit commencer par consulter les données de transports (but1.1). Ensuite, on calcule le chemin global en utilisant des algorithmes d'optimisation d'itinéraire (but1.2). Enfin, on structure la réponse pour qu'elle soit exploitable par le voyageur (sur un PC ou un PDA...) (but1.3).

A ce niveau, il faut prendre en compte le caractère distribué des données. En effet, comme décrit précédemment, les différents opérateurs de transport n'ont pas la possibilité de fusionner leurs bases de données. Il faut donc limiter les bases de données consultées à celles qui sont impliquées dans la recherche du chemin global. Autrement dit, il ne faudra interroger que les opérateurs qui peuvent proposer un tronçon du chemin final, ce qui se traduit au niveau du diagramme de buts par la décomposition du but1.1 en but1.1.1 et but1.1.2 où but1.1.1 est l'identification des opérateurs de transport impliqués dans la recherche du chemin global et le but1.1.2 est l'extraction des informations (horaires de transport) à partir des opérateurs concernés.

Le but2 du diagramme représentant « la gestion des perturbations » est décomposé en deux sous-buts. En effet, la gestion de la perturbation de transport se traduit par la détection de la perturbation (but2.1) qui va enclencher le deuxième but à savoir « l'orientation du voyageur en conséquence » (but 2.2). L'orientation du voyageur concerne l'utilisateur des transports avant ou pendant le voyage. Son orientation consiste principalement en deux actions : relier l'information de la perturbation au voyageur et lui proposer un itinéraire bis. En outre, l'utilisateur s'intéressant exclusivement aux informations de perturbation qui le concernent, une action de filtrage apparaît obligatoire. Ce qui nous ramène à la décomposition du but2.2 en trois sous-buts : calcul de l'impact de la perturbation détectée sur l'itinéraire du voyageur en prenant en compte la position du voyageur dans son voyage (but2.2.1). En cas d'impact, informer le voyageur des éventuels retards (but2.2.2) et proposer un itinéraire bis (but2.2.3) si la perturbation nécessite une meilleure offre d'itinéraire. Le diagramme de but ainsi construit est présenté dans la [Figure II-2].

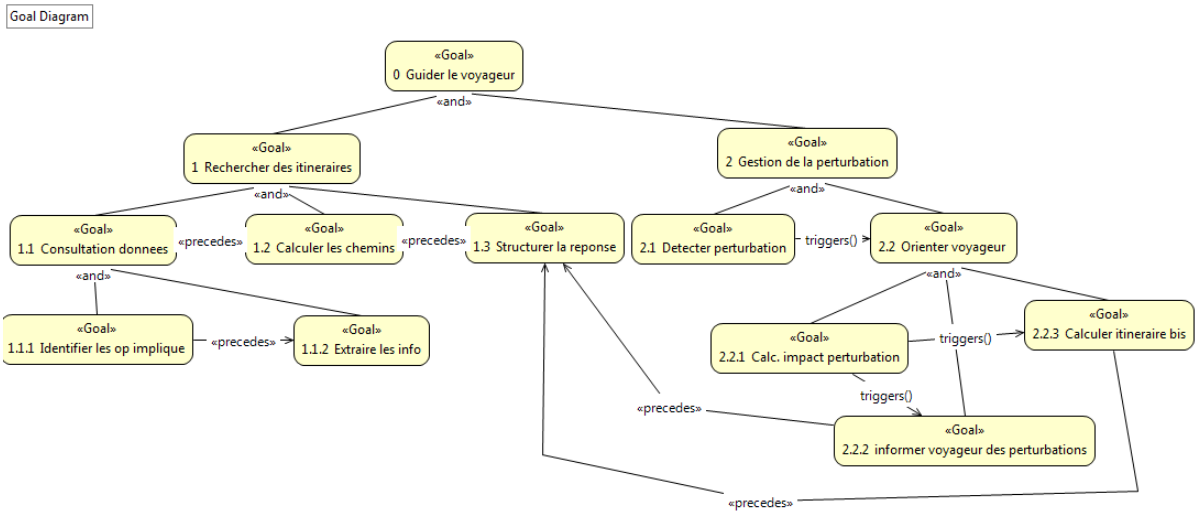

Figure II-2 - Diagramme de but

II.2.2. Diagramme de rôles et document de description des rôles

Notre objectif est de créer le diagramme de rôle qui décrit les différents rôles pouvant être joués par les agents du système.

Pour chaque but/sous-but identifié précédemment, nous devons créer un rôle pouvant le réaliser. Un rôle peut en l'occurrence réaliser deux buts en même temps. Afin de réaliser un but, un rôle doit avoir à sa disposition une (ou plusieurs) « capacité » qui se traduit généralement par des plans d'exécution (diagramme d'état-transition décrivant la manière selon laquelle un agent doit se comporter).

Comme décrit précédemment, huit but-feuilles composent notre diagramme de buts. Pour les réaliser, nous avons identifié sept rôles comme suit (voir Figure II-3) :

Pour le but1.1.1 « identifier les opérateurs impliqués » nous créons le rôle « limitation de la zone de recherche ». Ce rôle doit pouvoir identifier les opérateurs ayant un impact sur un itinéraire recherché. Ainsi, la première capacité requise pour ce rôle est le plan de recherche d'opérateurs impliqué. De plus, pour pouvoir rechercher ces opérateurs, ce rôle doit avoir accès aux opérateurs existants. Une deuxième capacité est donc requise, prise en charge par le plan d'enregistrement des opérateurs.

Pour réaliser le but1.1.2 «Extraire les info » nous avons identifié le rôle « extraire les informations » qui requiert la capacité plan d'extraction des informations.

Les buts « Calculer les chemins » but1.2 et « calculer chemin bis » but2.2.3 sont réalisés par un même rôle à savoir le rôle « composeur d'itinéraire », ce rôle dépend du plan de composition d'itinéraire.

Le but « structurer la réponse » (but1.3) est réalisé par le rôle « formater la réponse » qui nécessite le plan d'exécution « plan de formatage de la réponse ».

Le rôle « détection perturbation » réalise le but2.1 « détecter perturbation » et il nécessite la capacité « plan détection perturbation »

Le but2.2.1 « calcul impact perturbation » est réalisé par le rôle « calculateur impact perturbation » et il dépend du plan « plan de calcul impact perturbation ».

Enfin, nous avons identifié le rôle « informer voyageur perturbation » qui permet de réaliser le but2.2.2 « informer voyageur perturbation » et qui nécessite le plan « plan informer perturbation »

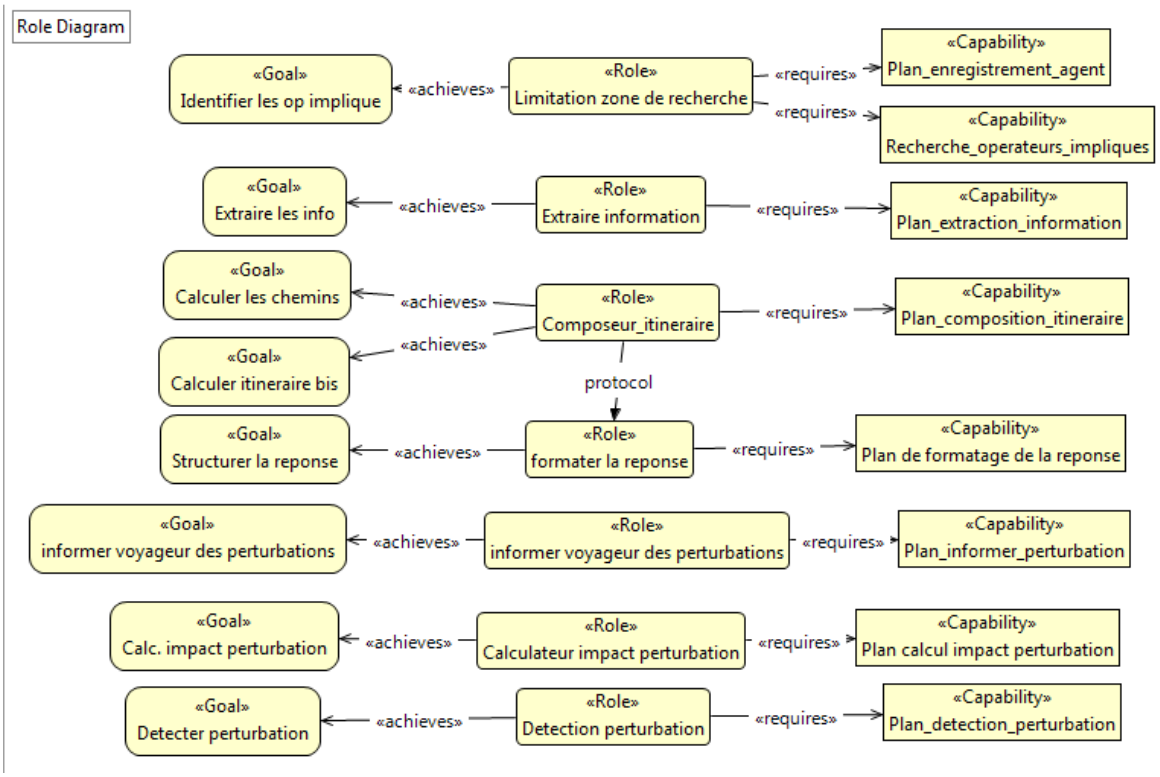

Figure II-3 - Diagramme de rôle

II.2.3. Diagramme d'Agent

Ce diagramme est créé afin qu'il y ait au moins une catégorie d'agent possédant toutes les capacités nécessaires pour jouer chaque rôle. Chaque classe représente un modèle pour un type d'agent qui pourra être instancié plusieurs fois selon les besoins du système.

La philosophie de la modélisation O-MaSE fournit une grande flexibilité de conception. De plus, le faite de séparer le rôle de l'agent lui-même permet de réduire le nombre d'entités mises en place dans le système. En effet, il y aura un nombre minimal d'agents qui assurent le bon fonctionnement du

système et changeront les rôles joués selon la situation. Ainsi, on assure une optimisation de l'utilisation des ressources mémoires.

Par ailleurs, il est plus intéressant de séparer les tâches de calcul dans des rôles distincts. Il est possible ainsi de paralléliser les calculs sur des agents différents. Il s'agit donc de trouver le meilleur compromis entre un nombre minimal d'agent et une bonne parallélisation.

Pour l'exécution de notre système, trois agents au moins doivent fonctionner simultanément (voir Figure II-4) :

Un premier agent qu'on notera « DPM_1 » jouera les rôles : (Composeur itinéraire, Calculateur impact perturbation, Informer voyageur des perturbations, Formater la réponse). Ainsi l'agent DPM_1 doit posséder toutes les capacités requises par ces rôles (Plan_composition_itineraire, Plan de formatage de la réponse, Plan_informer_perturbation, Plan calcul impact perturbation).

Un deuxième agent qu'on notera « DPM_2» jouera le rôle : (limitation de zone de recherche) il aura donc besoin des deux capacités (Plan_enregistrement_agent et Plan_Recherche_operateurs_impliques)

Enfin, un troisième agent DPM_3 jouera les rôles (Extracteur d'information et Détecteur perturbation) il aura donc besoin des capacités (Plan_extraction_information, Plan_detection_perturbation)

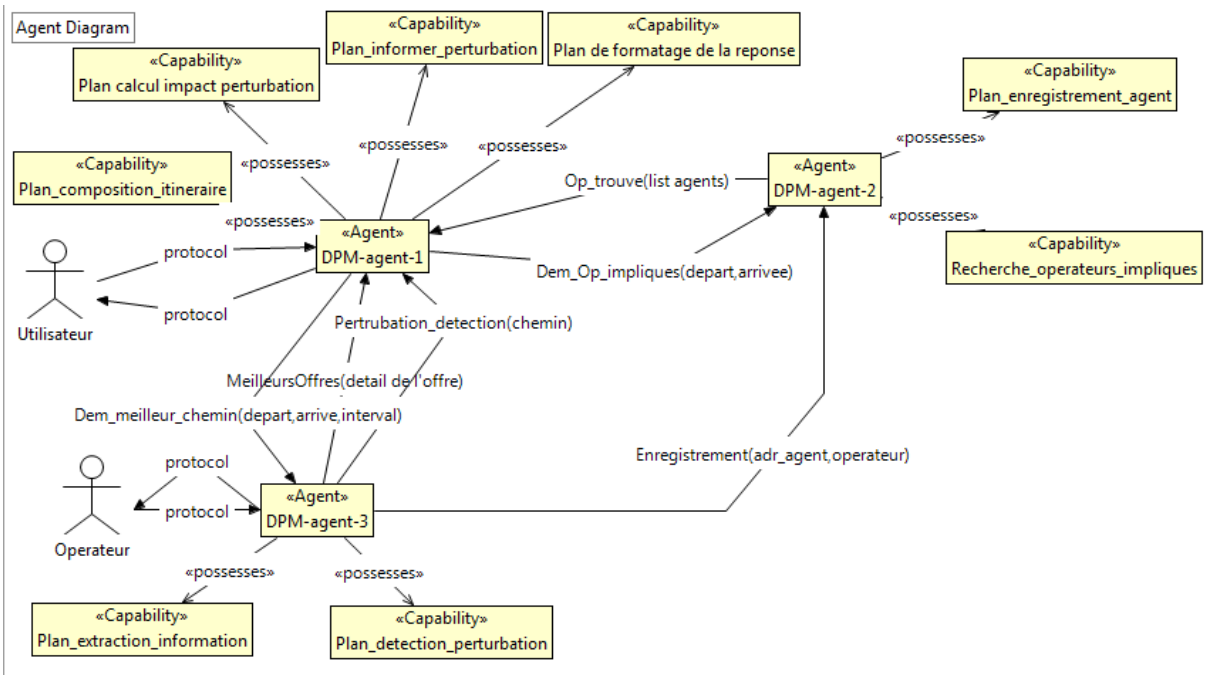

Figure II-4 - Diagramme d'agent

A partir de ce diagramme (Figure II-4) nous commençons à pouvoir visualiser la structure globale du SMA et ses interactions avec l'environnement extérieur.

Le point d'entrée est une requête venant d'un utilisateur extérieur. On précise ici que l'« utilisateur » ne désigne pas nécessairement une personne : en effet, l'origine de la requête peut provenir également d'un autre système qui fait appel au service de recherche d'itinéraire du SMA. Dans notre cas, c'est le niveau service qui effectue cet appel. C'est l'agent DPM_1 qui reçoit la requête (il joue le rôle « Formater la réponse » donc il peut communiquer avec l'utilisateur). Il demande ensuite à un agent DPM_2 jouant le rôle « limitation de zone de recherche » de lui fournir une liste d'opérateurs (L_{op}) et limitant ainsi la zone de recherche. L'agent DPM_1 récupère cette L_{op} et vérifie si pour chaque opérateur de cette liste, il existe un agent capable d'en extraire les informations sinon il demande sa création. Ensuite, il constitue la liste des itinéraires à retourner à l'utilisateur en consultant les agents ($DPM_3, \dots DPM_n$) jouant les rôles « Extracteur d'information ».

Du point de vue de l'utilisateur, une perturbation peut survenir à trois périodes différentes. Ce qui nous ramène à trois scénarios possibles pour un utilisateur. Le premier est une perturbation qui se produit avant que l'utilisateur ne commence à planifier son itinéraire. Le deuxième est une perturbation qui survient juste après que l'utilisateur a planifié son itinéraire, mais avant qu'il ne commence son voyage. Le troisième est une perturbation qui survient durant le voyage.

Notre système fonctionne en temps réel et il récupère toutes ses données à partir des Systèmes d'Information des Opérateurs de Transport (TOIS). Donc pour détecter une perturbation, le TOIS doit impérativement mettre à jour sa base de données. En effet, chaque opérateur est responsable de ses données et de leurs mises à jour. On peut même supposer qu'un opérateur subissant une perturbation prévient tout utilisateur de son SI en mettant un drapeau dans les réponses de ses web services ou exposer un web service indiquant l'état du SIAD (en ligne, perturbée, maintenance …) qui précise en plus les détails de la perturbation (ligne, type de perturbation, impact…). En général, un SIAD ne fait que la mise à jour de ses données interne. Donc en cas de ligne coupée, il ne proposera plus d'itinéraire passant par cette ligne et, en cas de retard, il donnera le chemin le plus rapide, tout en prenant en compte le retard estimé.

Pour résumer, le comportement de notre système vis-à-vis d'une perturbation se produisant dans un opérateur dépend des efforts réalisés par ce dernier : si le SIAD concerné prévient les utilisateurs par un service indépendant, notre agent jouant le rôle « Detection_perturbation » consultera ce service périodiquement pour identifier les perturbations. Sinon, si le SIAD met tout simplement ses données à jour après chaque perturbation, l'agent jouant le rôle « Detection_perturbation » associé à ce SIAD va détecter la différence dans les données et préviendra donc l'utilisateur. La détection de cette différence s'effectue en lançant périodiquement la même requête et en comparant les réponses du SIAD.

Nous nous intéressons au premier comportement vu que le deuxième alourdit fortement les traitements et nous oblige à lancer un nombre important d'appels de services.

Dans tous les cas, c'est l'agent jouant le rôle « Détection_perturbation » ($DPM_3, \dots DPM_n$) qui reçoit l'information de perturbation et le transfère au DPM_1 qui joue le rôle « Calculateur impact perturbation ». Le DPM_1 prend en compte l'information de perturbation, les itinéraires fournis à l'utilisateur et l'heure actuelle. Pour déterminer la situation du voyageur (avant le voyage ou en cours de voyage) et si la perturbation impacte le voyage ou pas. Selon le résultat de ces comparaisons, le DPM_1 jouera un autre rôle soit tout simplement « Informer voyageur des perturbations » ou « Composeur itinéraire » pour pouvoir proposer un itinéraire bis.

II.2.4. Diagrammes de Plan

La modélisation des plans d'exécution permet de décrire les algorithmes utilisés par les agents afin d'atteindre un objectif ou un ensemble d'objectifs. Nous détaillons dans ce paragraphe les diagrammes de plan des capacités identifiées précédemment.

Plan « extraction_information »

A chaque opérateur de transport existant dans la liste L_{op}, nous associons un agent (DPM_k) jouant le rôle « Extracteur d'information ». L'agent se charge alors d'extraire les données pertinentes du TOIS auquel il est associé. Cette extraction de données s'effectue sous forme d'appel de service distant. L'agent doit pouvoir répondre à une requête simple du type « Quel est le meilleur chemin que ton opérateur offre entre les stations X et Y si on part de X à une heure t_X » pour simplifier l'écriture, nous notons cette question de la manière suivante : $Req(X, Y, t_X)$. Donc, une fois cette requête reçue, l'agent consulte le TOIS associé via un web service et extrait la réponse relative à cette requête. Le TOIS comporte un calculateur interne et un algorithme de recherche d'itinéraire capable de répondre à cette requête simple. Il transmet alors la réponse via le web service [Figure II-5].

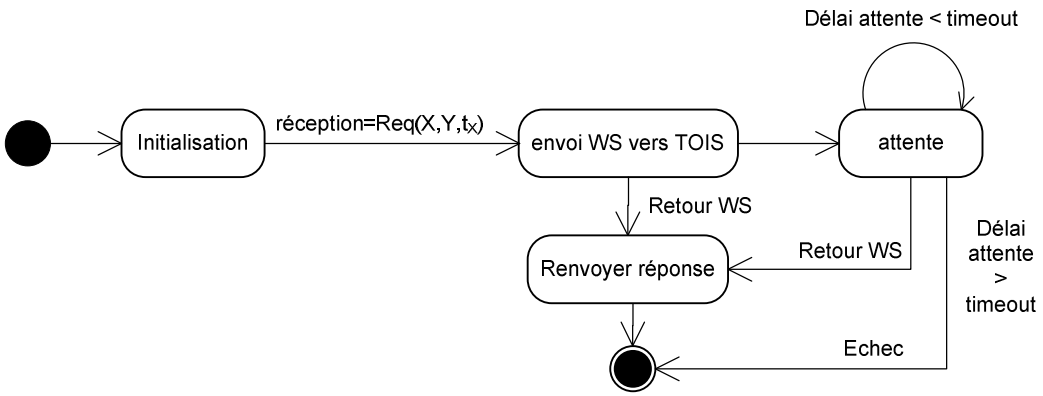

Figure II-5 - Plan extraction information

Plan «Detection_perturbation»

Comme précisé précédemment, le comportement des agents vis-à-vis de la détection des perturbations dépend des services proposés par le SIAD. Dans le cas où le SIAD ne fait que mettre à jour ses données, le plan de détection de perturbation se résume à une exécution périodique du « **plan extraction information** » et une comparaison des réponses. Si deux réponses successives sont différentes, il y a modification des données et donc un impact potentiel sur l'itinéraire proposé. Dans ce cas, l'agent jouant le rôle «**Detection_perturbation**» remonte l'information.

Intéressons-nous ici à l'autre possibilité : celle où le SIAD est plus évolué et propose un ou plusieurs services pour connaitre l'état de l'opérateur.

La (Figure II-6) présente le diagramme de plan qui détaille le comportement de l'agent jouant le rôle «***Detection_perturbation***» vis-à-vis d'un tel SIAD. Après son initialisation, l'agent vérifie l'état du SIAD en envoyant un service web. Si la réponse indique que l'opérateur associé au SIAD n'est pas perturbé, l'agent s'endort pendant une période T (paramétrable). A son réveil, il relance le même web service.

Sinon, (si la réponse indique que l'opérateur associé au SIAD n'est pas perturbé), l'agent tente un appel de service pour récupérer les détails de cette perturbation. Selon le degré d'évolution du SIAD, un tel service peut, ou peut ne pas exister. Si l'appel n'aboutit pas (échec), l'agent remonte l'information de perturbation à l'agent qui joue le rôle «***Calcul_impact_perturbation »***. Ce dernier se chargera de vérifier si l'opérateur perturbé est impliqué dans l'itinéraire proposé et vérifier l'impact de cette perturbation sur le chemin global en lançant une requête de type $Req(X, Y, t_X)$ (voir ***«Plan calcul_impact_perturbation»***).

Si l'appel de récupération des détails de perturbation aboutit, toutes les informations sont remontées à l'agent jouant le rôle «***Calcul_impact_perturbation»*** qui vérifiera directement l'impact de cette perturbation sur le chemin global proposé et lui évite des requêtes supplémentaires de type $Req(X, Y, t_X)$.

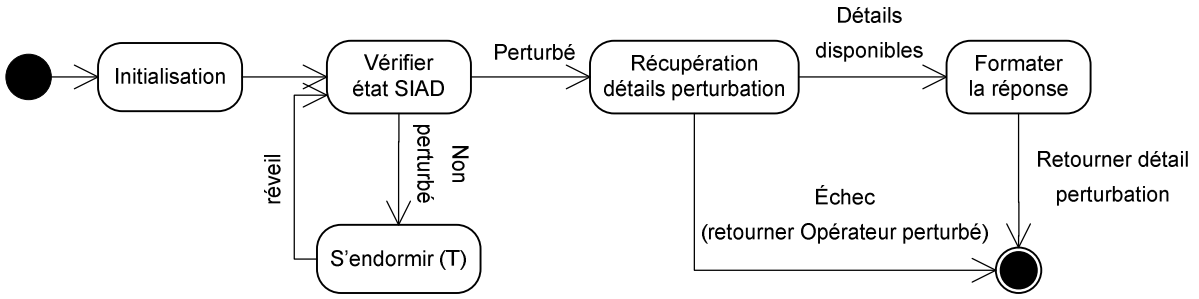

Figure II-6 - Plan détection_perturbation

Plan « calcul_impact_perturbation»

Le calcul de l'impact de la perturbation détectée consiste, dans un premier temps, à vérifier si l'itinéraire proposé à l'utilisateur comporte un tronçon touché par la perturbation. Ce calcul est réalisé de deux manières selon l'information de perturbation reçue en entrée. En effet, on commence par vérifier si le détail de la perturbation est disponible (information reçue de l'agent jouant le rôle « détection perturbation »). Si le détail de la perturbation est disponible, on compare la ligne touchée et l'itinéraire proposé à l'utilisateur. Si l'itinéraire proposé comporte un tronçon touché par la perturbation on passe à l'étape 2.

Si le détail de la perturbation n'est pas disponible, on a donc uniquement le nom de l'opérateur. On commence alors par vérifier si cet opérateur est impliqué dans l'itinéraire proposé à l'utilisateur. Dans l'affirmative, on isole le tronçon appartenant à cet opérateur (intersection entre cet opérateur et l'itinéraire global qu'on a proposé). Supposons que ce tronçon va d'une station A vers une station B avec un départ de A à t_A et une arrivée à B à t_B. On demande alors à l'agent jouant le rôle « extraire information » associé à cet opérateur de nous fournir la réponse via la requête $Req(A, B, t_A)$. Avec $Req(A, B, t_A)$ =*« Quel est le chemin le plus rapide entre une station X et une station Y si on part de X à* $t = t_X$ *».* Dans la réponse de cette requête, on récupère t'_B l'heure d'arrivée à la station B. On compare alors t'_B et t_B. Si $t_B \neq t'_B$ cela veut dire que la perturbation impacte l'itinéraire proposé on passe alors à l'étape 2.

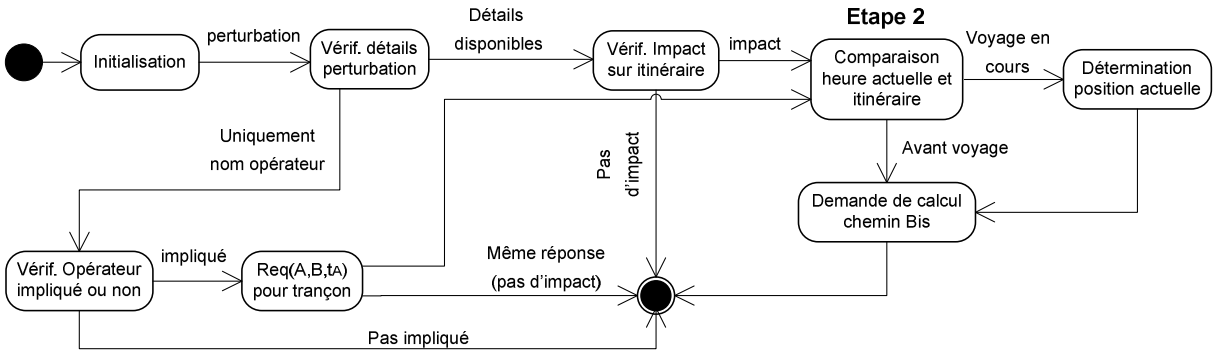

Figure II-7 - Plan calcul_impact_perturbation

Etape 2 : A ce niveau, on est sûr que la perturbation détectée impact l'itinéraire qu'on a proposé à l'utilisateur. On essaye d'identifier la situation de l'utilisateur (Est-il parti en voyage ou pas encore ?). Si le départ de l'itinéraire proposé n'est pas encore atteint, on fait appel au rôle «Composeur itinéraire » et la capacité « calcul itinéraire bis» pour proposer un itinéraire bis. Si le départ est déjà passé, on estime la position de l'utilisateur par rapport au trajet. Cette dernière tâche peut se réaliser soit par un détecteur GPS, soit en estimant mathématiquement la position (sachant l'heure de départ et

d'arrivée et l'heure actuelle, on calcule une moyenne pour déterminer le pourcentage du chemin parcouru). Selon la position déterminée, on lance le rôle de recherche d'itinéraire bis.

Ce traitement est schématisé dans la Figure II-7.

Plan « Plan_Recherche_operateurs_impliques»

Suite à la réception d'une requête du type $Req(X,Y)$, donc une requête qui demande de rechercher les compositions d'opérateurs liant la station X à la station Y. Autrement dit, en créant un graphe simple où les opérateurs constituent des nœuds, quelles sont les successions d'opérateurs possibles pour lier la station X à la station Y (Cette problématique et l'algorithme utilisé seront détaillés dans le chapitre III.5)

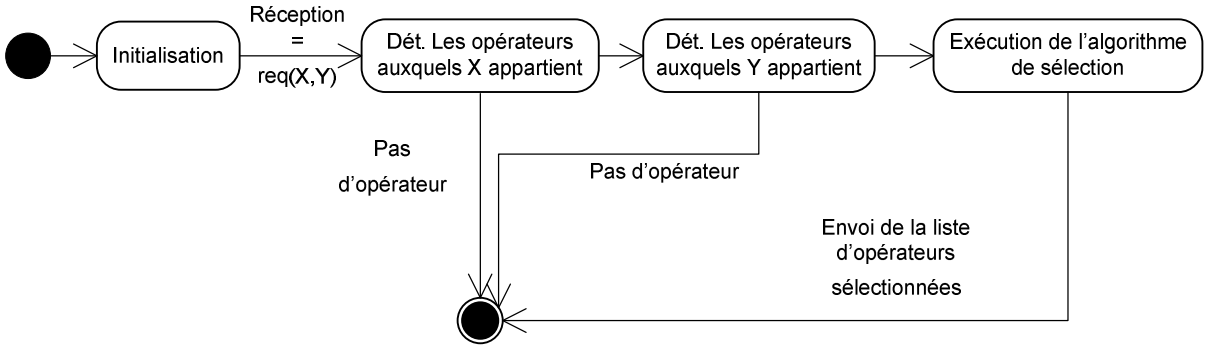

Figure II-8 - Plan_Recherche_operateurs_impliques

L'agent qui joue le rôle « limitation de zone de recherche » se charge d'utiliser un algorithme de sélection pour déterminer les groupements d'opérateurs capables de répondre à la requête. Ces groupements d'opérateurs seront stockés dans la liste (L_{op}) et renvoyés en réponse (voir Figure II-8).

Plan « Plan_composition_itineraire»

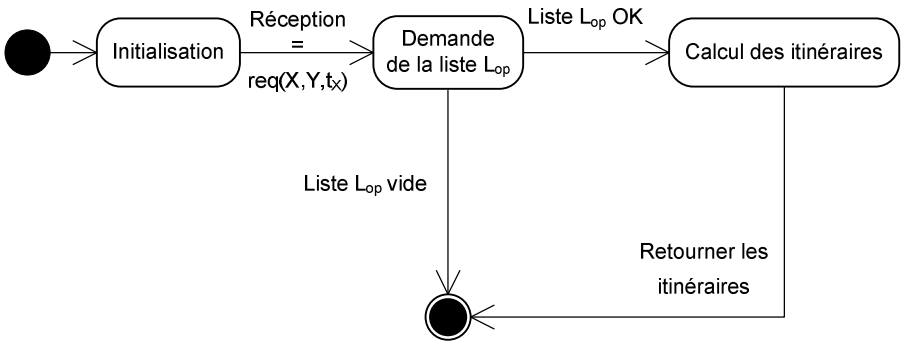

Figure II-9 - Plan_composition_itineraire

Suite à la réception de la requête du type $Req(X,Y,t_X)$, l'agent jouant le rôle « Composeur d'itinéraire » la transforme en $Req(X,Y)$ et communique avec un autre agent pour récupérer la liste contenant les groupements d'agents L_{op} (voir Figure I-9). Ensuite, il utilise un algorithme de recherche

d'itinéraire dans un environnement dynamique et distribué (cet algorithme sera détaillé dans le chapitre III) pour trouver les chemins les plus rapides partant de X à t_X pour arriver à Y au plutôt. Il s'agit dans cet algorithme de composer un itinéraire multi-opérateur à partir d'itinéraires locaux.

II.3. Le niveau service

Comme précisé précédemment, pour le niveau service, nous utilisons les web services. Ces derniers facilitent la communication, vu qu'ils représentent un standard de communication sur le Web.

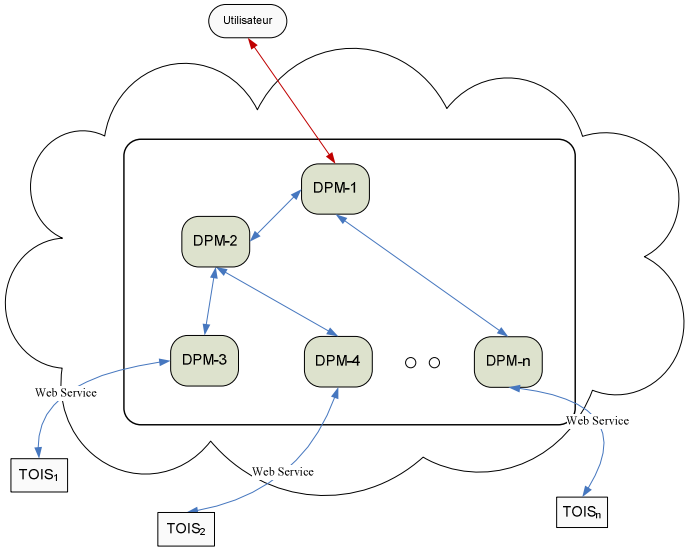

Figure II-10 - DPM et communication web services

Le niveau service se charge de la communication entre le système multi-agent et son environnement [Figure II-10]. D'après les diagrammes décrits précédemment, seules trois capacités nécessitent un échange entre le SMA et l'environnement extérieur. Ces trois capacités sont :

- Le Plan_extraction_information
- Le Plan_detection_perturbation
- Le Plan_informer_perturbation

Pour l'extraction des informations de transport à partir des SIADs, un web service sert de lien de communication entre un agent et un SIAD. Le service exposé par un SIAD doit pouvoir répondre à une requête classique du type $Req(X, Y, t_X)$:

« Quel est le chemin le plus rapide entre une station X et une station Y si on part de X à $t = t_X$ »

X et Y appartenant au réseau géré par le SIAD concerné, il lui suffit alors d'utiliser son calculateur interne pour répondre à cette requête.

Un diagramme de séquence décrivant la consommation de ce web service est présenté dans la Figure II-11. Un agent lance un appel de service de type $Req(X, Y, t_X)$. Le SIAD traite cette requête de son

côté et interroge le calculateur interne pour répondre. Une réponse KO est générée en cas d'erreur technique ou de timeout (délai d'attente dépassé). Si la génération de la réponse s'est déroulée sans problème, le chemin le plus rapide est retourné.

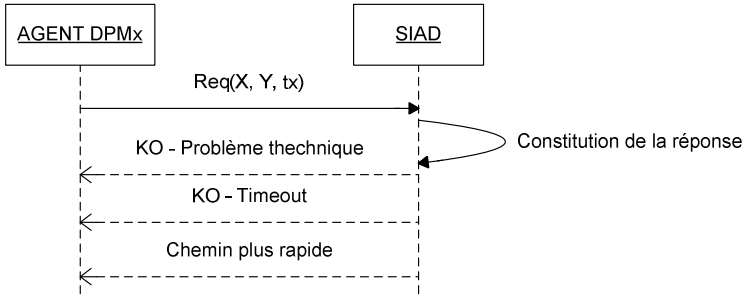

Figure II-11 - Diagramme de séquence du WS extraction d'information

Concernant la détection de perturbation [Figure II-12], un agent commence par interroger un SIAD via un premier web service pour identifier l'état de son opérateur associé. La réponse à ce web service est un « état » indiquant la situation dans laquelle se trouve l'opérateur. L'agent, identifie deux états : « perturbé » ou « non perturbé » = («en ligne » …). Dans le cas d'un timeout ou d'une erreur technique, le système considère que l'opérateur est perturbé et ne propose donc pas d'itinéraire exploitant cet opérateur, assurant ainsi une fiabilité des itinéraires proposés.

A l'issue du premier appel, et si la réponse indique que l'opérateur est perturbé, l'agent tente un deuxième appel pour essayer, cette fois-ci, de récupérer les détails de la perturbation (ligne perturbée, type de perturbation …). Selon l'évolution du SIAD, ce dernier peut proposer un tel service. Dans ce cas, une réponse d'indisponibilité où un simple 'timeout' indiquera cette information. Sinon le SIAD renverra les détails.

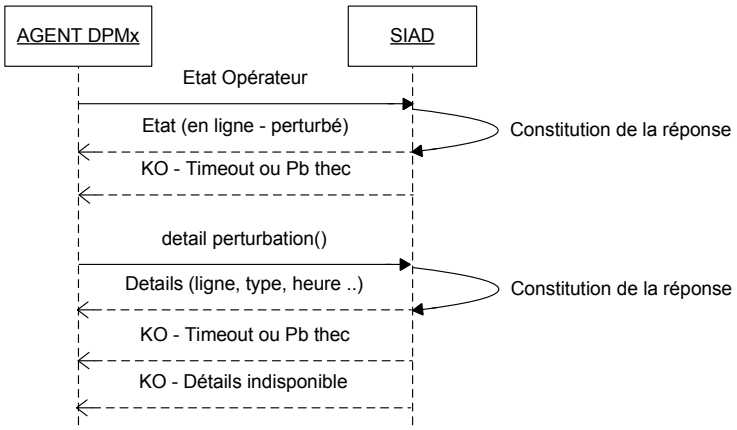

Figure II-12 - Diagramme de séquence du WS détection de perturbation

Enfin, pour informer l'utilisateur des éventuelles perturbations, il n'est pas possible de considérer que le voyageur expose un web service pour être informé. Il n'est pas non plus pertinent de considérer que le client dispose de système en fonctionnement (sur PC ou autre) tout au long de son voyage même après la consultation des itinéraires.

Nous proposons alors une autre approche pour mieux prévenir le voyageur. A la connexion au système, le voyageur indique un e-mail ou un numéro de téléphone portable. Ainsi selon que l'utilisateur possède un (PDA/Smartphone) ou un simple téléphone portable, il sera informé par un e-mail ou un SMS.

II.4. Le niveau présentation

L'environnement utilisé pour mettre en place le niveau présentation, est un environnement J2EE il permet d'accéder aux services offerts par le système à travers des pages JSP et des servlets écrites en langage JAVA.

Les applications J2EE sont des applications multicouches [Figure II-13] : la couche interface (présentation) s'occupe de la livraison des données au client, la couche métier quant à elle, effectue les transformations sur les données. Enfin, la couche data communique avec la base de données.

Les applications J2EE sont gérées par des conteneurs au sein des serveurs d'applications. Ils s'occupent de diriger les transactions, gérer la persistance et le dialogue avec l'environnement externe ainsi que la gestion des threads.

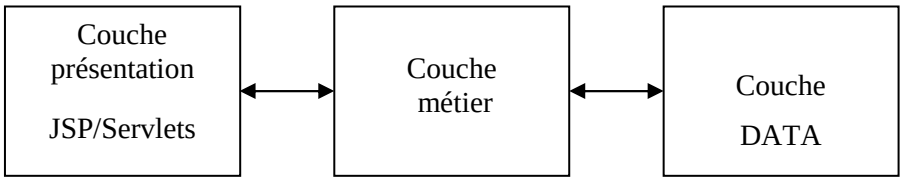

Figure II-13 - Architecture Muli-couche J2EE

Dans notre système d'information, il n'y a pas de base de données proprement dite. Les informations manipulées se trouvent dans les messages échangés entre les agents. Pour cela, la troisième couche de communication avec la base de données sera remplacée par une couche de communication avec le SMA ou communication avec le niveau Service.

Pour concevoir le niveau présentation du DPM nous avons opté pour une architecture à trois couches formées par les couches interface, traitement et connexion au Service/SMA. Les trois couches sont indépendantes. Ceci garantit une certaine stabilité et facilite la maintenance. En effet, si l'une de ces couches subit des modifications, les deux autres sont généralement épargnées. Les deux premières couches sont analogues à celles de l'architecture J2EE. La dernière remplace la couche de connexion à

la base de données, elle sert de lien entre le niveau présentation et les deux autres niveaux [Figure II-14].

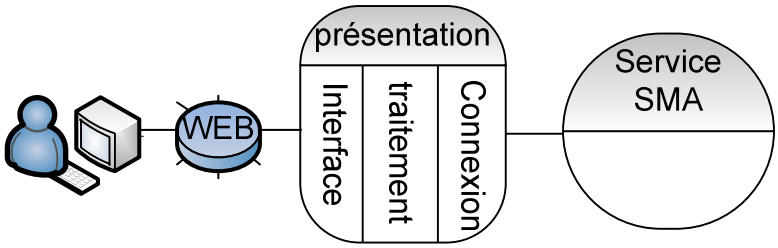

Figure II-14 - Les trois couches du niveau présentation

II.5. Type d'opérateur : Transport en Commun et transport individuel

Jusqu'à présent, nous avons parlé de SIAD comme des systèmes d'information d'aide au déplacement liés à des opérateurs de transport, sans toutefois préciser ou spécifier le type d'opérateur. Dans un premier temps, on peut considérer ces opérateurs comme des opérateurs de transport public, ainsi les itinéraires proposés seront des itinéraires multimodaux totalement assurés par les transports en commun.

Or en cas de perturbation, il serait judicieux d'augmenter les offres de transport disponibles, en s'orientant vers les offres d'AutoPartage, de covoiturage et de vélos-en-libre-service. De cette manière, l'offre globale proposée serait une offre comodale associant des tronçons en transport en commun et des tronçons en transport individuel (voiture, vélo ...).

Les systèmes d'AutoPartage et de covoiturage sont aussi gérés par des systèmes d'informations spécifiques. Pour l'AutoPartage, ces systèmes précisent les points de retrait des véhicules, le nombre d'unités disponibles. Dans le cas du covoiturage, ils indiquent les offres disponibles en précisant les stations de départ, la station d'arrivée, l'heure de départ et la durée du trajet.

Le problème qui peut se présenter est le suivant : certains de ces SI n'intègrent pas de calculateurs internes, mais proposent uniquement les stations de départ et d'arrivée. Or, pour pouvoir exploiter ces opérateurs dans notre système, ces derniers doivent être capables de répondre à une requête de type « Aller de A vers B avec une heure de départ t_A» comme pour les transports en commun.

La solution pour gérer le cas où l'opérateur ne dispose pas de calculateur est de passer par un site proposant le calcul d'itinéraire en voiture ou en vélo entre deux points. Plusieurs sites proposent cette fonctionnalité, les plus connus étant «Google Maps[1]» [Figure II-15] et « Mappy[2]» mais on en trouve

[1] http://maps.google.com/
[2] http://fr.mappy.com/itinerary_homepage

d'autres spécialisés dans les itinéraires pédestre et cyclable comme « Running Map[1]» ou «Calcul Itinéraire[2]».

Pour résumer, dans le cas d'une perturbation, notre système élargit sa zone de recherche en exploitant des opérateurs de transport individuels (covoiturage, AutoPartage, vélo-en-libre-service …) et si les SIADs de ces derniers ne proposent pas un service de calcul d'itinéraire, on consulte un SI indépendant qui fournit, pour un emplacement de départ et un emplacement d'arrivée, l'itinéraire optimal et la durée du trajet global.

Notre système se comportera alors vis-à-vis d'un SI de covoiturage ou d'AutoPartage de la même manière qu'un SIAD de transport en commun. On assimile ainsi tout SI lié au transport (que ce soit transport en commun ou individuel) à un SIAD capable de répondre par la meilleure offre d'itinéraire entre deux points de son réseau.

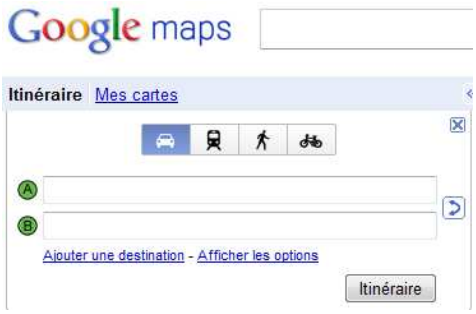

Figure II-15 - Interface de Google Maps de recherche d'itinéraire

Pour différentier les deux types de SIAD au niveau de notre système, nous utilisons un indicateur d'identification du type (en-commun / individuel) du système d'information. Cet indicateur sera utilisé dans la recherche d'itinéraire en temps normal ou perturbé. En effet, les opérateurs individuels seront désactivés en temps normal et activé si une perturbation se produit.

II.6. Fonctionnement global du système et des stratégies de gestion des perturbations

Rappelons tout d'abord que notre vision du problème est une vision « côté client ». Notre objectif premier est l'optimisation de la qualité du service offert aux voyageurs. Mais contrairement aux travaux réalisés « côté opérateur », nous n'avons ni la possibilité d'agir sur le nombre de bus/wagons en transit ni d'influer sur la fréquence de passage des moyens de transport. Plusieurs travaux traitent de cette problématique côté « opérateur » et proposent des algorithmes et des stratégies pour fluidifier

[1] http://www.runningmap.com/
[2] http://www.calculitineraires.fr

le trafic suite à une perturbation, survenant dans une seule zone ou dans plusieurs zones simultanément. Parmi les travaux qui étudient la problématique de régulation, on peut citer ceux de ZIDI [**127**], de OULD SIDI[**87**] et de Borne et al [**14**].

Dans notre cas, il s'agit d'essayer d'aider au mieux le voyageur en temps normal et en cas de perturbation sans pour autant agir sur la disposition du réseau et les fréquences. Nous procèderons alors d'une manière informative. Nous proposons des itinéraires optimisés en temps normal et en cas de perturbation, nous informons le voyageur et lui proposons un itinéraire de secours. Ce qui suit décrit le fonctionnement du système et les stratégies appliquées en cas de perturbation.

Au lancement du système, pour chaque SIAD disponible, un agent sera créé. Chacun de ces agents, jouera les rôles « Extraire Information » et « Détection perturbation » et sera associé à un SIAD unique. L'agent aura toutes les informations nécessaires pour consulter le SIAD. Ensuite, tout au long de l'exécution du système, si un nouveau SIAD se rend disponible, un agent sera créé et associé à ce SIAD. L'agent meurt si le SIAD auquel il est associé est indisponible.

Pour référencer ces agents dans ce paragraphe, on notera $Agent_{Sx}$ l'agent associé au $SIAD_x$ ($Agent_{S1}$ associé au $SIAD_1 \ldots$).

Un autre agent sera aussi créé au démarrage du système, ce dernier jouera le rôle (limitation de zone de recherche). Il se charge de maintenir la liste des agents vivants dans le système, leurs adresses, et les services qu'ils proposent. Comme précisé précédemment, il constitue une sorte d'Annuaire. On le notera $Agent_A$. Cet agent meurt à l'arrêt total du système. Chaque agent qui se connecte au système doit s'enregistrer dans l' $Agent_A$ en fournissant ses coordonnées. Avant de mourir, chaque agent se désenregistre, ce qui permet à l' $Agent_A$ de maintenir une liste à jour.

En se connectant au système, l'utilisateur communique avec le niveau présentation, il formule sa requête (en choisissant les paramètres : station de départ, station d'arrivée, heure/intervalle de départ) et indique le moyen avec lequel il souhaite être informé en cas de perturbation (SMS/e-mail). Ces informations sont transmises grâce au niveau service vers le niveau SMA, où sera créé un nouvel agent. Ce dernier jouera les rôles (Composeur itinéraire, Calculateur impact perturbation, Informer voyageur des perturbations, Formater la réponse) et sera, en quelque sorte, associé à cet utilisateur (l'utilisateur qui vient de se connecter). On notera cet agent $Agent_{Ux}$ pour $l'Utilisateur_x$.

L'$Agent_{Ux}$ gardera en mémoire le moyen de contacter $l'Utilisateur_x$(tél/e-mail) et le dernier itinéraire sélectionné par l'utilisateur. Il restera en vie tant que l'heure actuelle n'est pas supérieure à l'heure d'arrivée à la dernière station de l'itinéraire en mémoire. Autrement dit, si l'itinéraire choisit par l'$Utilisateur_x$ va de A vers B et la réponse envoyée à $l'Utilisateur_x$ indique un itinéraire arrivant à B à une heure t_B. L'$Agent_{Ux}$ continue à vivre dans le système jusqu'à ce que l'heure actuelle soit

supérieure à t_B.Une fois cette heure (t_B) atteinte, l'agent meurt libérant ainsi la mémoire. De cette manière, en cas de perturbation, l'$Agent_{Ux}$ est capable de prévenir $l'Utilisateur_x$ puisqu'il a les coordonnées nécessaires pour le contacter. Si l'$Agent_{Ux}$ n'a pas d'itinéraire en mémoire, cela veut dire que le système DPM n'a pas encore calculé de solution à l'$Utilisateur_x$ dans ce cas, l'heure de mort de l'$Agent_{Ux}$ est fixé à l'infini (à l'arrêt totale du système).

Dés la réception de la requête, l'$Agent_{Ux}$ interroge $l'Agent_A$ pour identifier la zone de recherche. En effet, ce dernier a la liste des opérateurs disponibles et utilise un algorithme de délimitation de la zone de recherche (algorithme décrit dans le chapitre suivant) pour déterminer les SIAD (et donc les $Agent_{Sx}$) concernés. En réponse, l'$Agent_{Ux}$ recevra une liste d'agents $Agent_{Sx}$.

L'$Agent_{Ux}$ se charge par la suite de composer le chemin global. Il interroge les différents agents $Agent_{Sx}$ pour récupérer les différentes offres locales. Ceci est réalisable étant donnée que chaque $Agent_{Sx}$ est capable de répondre à une requête d'itinéraire, du type aller de A vers B, avec A et B stations gérées par le $SIAD_x$.

Pour composer le chemin global, l'$Agent_{Ux}$ utilise un algorithme dynamique de recherche de plus court chemin distribué avec un intervalle de départ (algorithme décrit dans le chapitre suivant). A l'issue de cet algorithme, une liste d'itinéraires optimaux est générée. Cette liste transite via le niveau service vers le niveau présentation en utilisant des services web. Dans le niveau présentation, une couche de traitement se charge d'adapter le format de la réponse au dispositif utilisé par l'utilisateur pour se connecter au système. Si le service de recherche retourne une liste d'itinéraires, l'utilisateur doit sélectionner la solution qui lui convient au mieux. Cette solution sera gardée en mémoire de l'$Agent_{Ux}$ et sera exploitée en cas de perturbation pour mieux l'assister.

Si un $Agent_{Sx}$ détecte une perturbation sur le $SIAD_x$, il commence par interroger l' $Agent_A$ sur les adresses de tous les $Agent_{Uy}$ vivants (jouant le rôle « informer voyageur des perturbations »). Ensuite, il leur transmet l'information de perturbation. Chacun des agents $Agent_{Uy}$ se charge d'estimer l'impact de la perturbation sur les itinéraires proposés antérieurement aux utilisateurs (Voir le diagramme du Plan « calcul_impact_perturbation»).

Quatre scénarios sont possibles :

1. La perturbation n'a pas d'impact sur l'itinéraire proposé. L'agent ignore alors l'information.

2. La perturbation a un impact sur l'itinéraire, mais le tronçon concerné a déjà été parcouru par le voyageur. L'agent ignore donc l'information de perturbation.

3. La perturbation impacte le tronçon en cours ou un tronçon ultérieur et l'utilisateur est en cours de voyage. Comme dans l'exemple [Figure II-16] où un voyageur se déplace de la station A vers la station B et alors qu'il se retrouve sur le tronçon $[X, Y]$ une perturbation survient sur le tronçon $[Y, Z]$. L' $Agent_{Uy}$ utilise dans ce cas un algorithme dynamique de recherche d'itinéraire distribué en activant les opérateurs de transport individuels. Il recherche un itinéraire partant de la prochaine station du mode de transport emprunté à l'instant où la perturbation se produit ($t_{perturbation}$) (dans notre exemple la station Y) et en prenant comme heure de départ l'heure d'arrivée à cette station (donc t_Y) pour atteindre toujours la même destination (la station B).

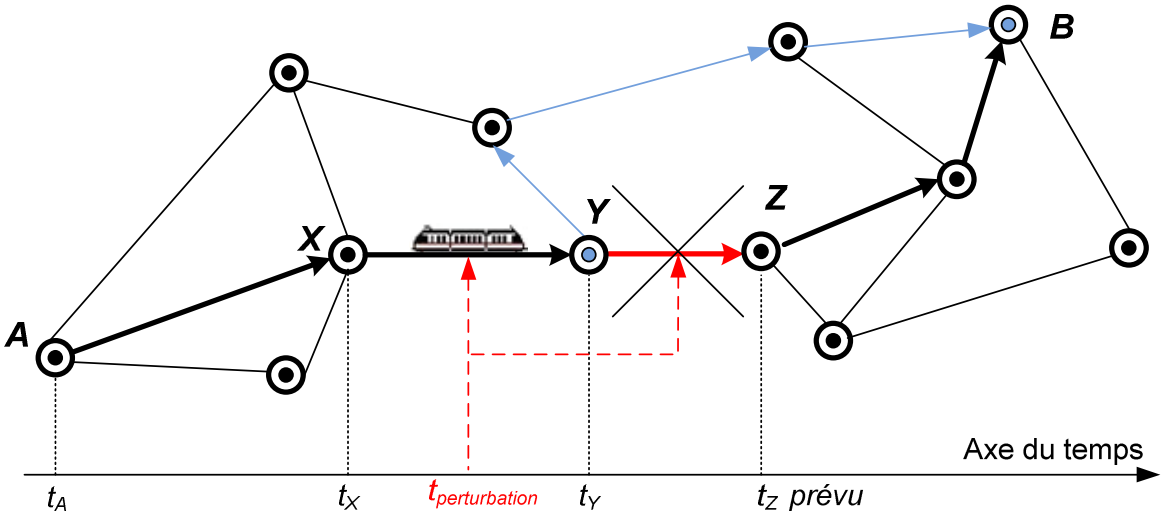

Figure II-16 - Perturbation sur le tronçon suivant

L' $Agent_{Uy}$ retrouve alors un chemin bis (menant de Y vers B avec une heure de départ t_Y) et la transmet au client.

4. La perturbation touche un tronçon dans l'itinéraire, mais le voyageur n'est pas encore parti. L' $Agent_{Uy}$ ré-exécute le même algorithme dynamique de recherche d'itinéraire distribué avec intervalle de départ, mais en activant les opérateurs de transport individuels. La perturbation sera prise en compte vu que les SIAD mettent à jour leurs données.

II.7. Conclusion

Dans ce chapitre, nous avons présenté la conception globale de notre système en détaillant le rôle, le fonctionnement et les choix retenus pour chaque niveau. Pour définir le niveau SMA, nous avons utilisé la méthodologie O-MaSE. Cette méthodologie organisationnelle, qui sépare les agents de leurs rôles, nous permet une grande flexibilité de conception.

Nous avons proposé, ainsi, un système capable de gérer une perturbation en informant le voyageur concerné et en lui proposant des itinéraires de secours. Les rôles étant indépendants des agents, chaque agent peut changer de rôle selon la situation dans laquelle il se trouve. De plus, cette séparation rôle/agent facilite la modélisation de la propriété de vie et de mort d'un agent.

Pour un bon fonctionnement de notre système, trois agents doivent être simultanément en vie assurant ainsi un nombre minimal de rôles «Extracteur d'information » à partir des opérateurs existants, «Limitation de zone de recherche » pour définir un domaine de recherche réduisant la zone sur laquelle va être appliqué l'algorithme de recherche et « Composeur itinéraire » qui compose justement le chemin global en utilisant un algorithme de recherche de chemin plus rapide. A ces rôles, et selon la situation, peuvent se rajouter d'autres rôles comme « Détection perturbation » ou « Calcul impact perturbation » pour identifier une perturbation et estimer son impact sur un itinéraire proposé.

Ce dernier rôle, définit des stratégies de gestion des perturbations. Selon la situation du voyageur, il isole le tronçon impacté par la perturbation et relance le même algorithme de recherche d'itinéraire utilisé dans la planification du voyage, mais avec un paramétrage différent. Grâce à ces stratégies, on ramène notre problématique de gestion de perturbation à une problématique de recherche d'itinéraire.

Il nous reste donc à définir l'algorithme de recherche d'itinéraire paramétrable qui sera utilisé à la fois pour la planification initiale du voyage et pour la recherche d'un itinéraire bis en cas de perturbation.

III. Graphes et algorithmes d'optimisation

III.1. Introduction

Dans le chapitre précédent, nous avons présenté des stratégies de gestion des perturbations et les comportements des agents vis-à-vis des imprévus. Les actions réalisées par ces agents se résument généralement par une isolation du tronçon touché par la perturbation et une réexécution de l'algorithme de recherche d'itinéraire pour trouver un chemin de secours.

On centralise donc les algorithmes utilisés par notre système dans un algorithme de recherche d'itinéraire paramétrable. Les paramètres étant, un point de départ, un point d'arrivé et une heure/intervalle d'heures de départ. Ces paramètres sont calculés par l'agent jouant le rôle «Calcul impact perturbation » selon la situation du voyageur au moment où la perturbation est détectée et en concordance avec les stratégies de gestion des perturbations.

Dans ce chapitre, nous focalisons notre étude sur les algorithmes de recherche d'itinéraire qui seront utilisés dans un premier temps pour la planification de l'itinéraire initial et dans un second lieu pour la recherche d'un itinéraire bis en cas de perturbation ou de retard.

Nous exposons les résultats de recherche tirés de la théorie des graphes et de la recherche opérationnelle.

III.2. Définitions et notations

Un graphe est défini par le couple $G = (S, A)$ avec :

- ✓ S un ensemble des éléments appelés sommets ou nœuds.
- ✓ A un ensemble des éléments appelés « arrêtes ». Une arrête est un couple $(i, j) \in A$ tel que $i, j \in S$.

On distingue deux types de graphes : les graphes non-orientés et les graphes orientés [Figure III-1]. Dans un graphe non-orienté, l'arrête (i, j) est équivalente à l'arrête(j, i). Dans un graphe orienté, l'ensemble A est un ensemble de couples ordonnés de sommets. On parle plutôt d'arc (i, j) où i représente le sommet source et j le sommet destination. L'arc (i, j) est donc différent de l'arc (j, i).

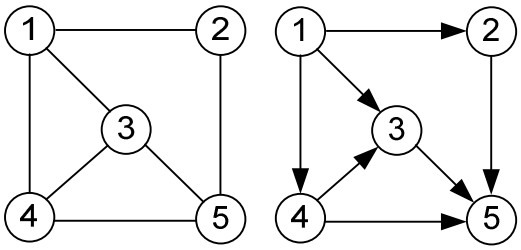

Figure III-1 - Graphe non orienté (à gauche) et graphe orienté (à droite)

Dans la suite, on s'intéresse aux graphes orientés. Dans un graphe orienté, la notion de couples ordonnés nous permet de définir deux nouveaux ensembles pour chaque nœud $x \in S$. L'ensemble des successeurs de x est noté $N^+(x)$ et l'ensemble des prédécesseurs de x est noté $N^-(x)$.

Formellement, ces deux ensembles sont définis comme suit :

$$N^+(x) = \{v \in S / (x,v) \in A\} \text{ et } N^-(x) = \{v \in S / (s,v) \in A\}$$

Un chemin $Ch(i,j)$ entre deux nœuds $(i,j \in S)$ d'un graphe orienté est une succession d'arcs liant i à j. On note ce chemin en listant successivement les nœuds qui le composent $Ch(i,j) = (s_0, s_1, .., s_n)$. Chaque deux nœuds successifs représentent un arc du graphe.

$$Ch(i,j) = (s_0, s_1, .., s_n) \text{ avec } \begin{cases} s_k \in \mathrm{S}, \forall k \in [0,n] \\ s_0 = i, s_n = j \\ \forall s_k, s_{k+1} \in Ch(i,j), \text{tel que } (s_k, s_{k+1}) \in A \end{cases}$$

L'ensemble des chemins liant le nœud i au nœud j est noté $E_{ch}(i,j) = \{ch(i,j)\}$. On notera dans ce qui suit l'arc $(i,j) = a_{i,j}$ avec $i,j \in S$.

Un graphe G est dit « valué » si on lui associe une fonction de poids qui permet d'attribuer une valeur (un poids) à chaque arc du graphe. On notera $w_{i,j}$ le poids de l'arc $a_{i,j}$. Il faut remarquer que, dans un contexte du transport urbain, les poids des arcs ne peuvent pas être négatifs. Ainsi dans ce cas $\forall i,j \in S,\ w_{i,j} \geq 0$.

Dans un graphe valué, on définit la fonction "$co\hat{u}t$" qui permet de calculer le coût d'un chemin de la manière suivante : $\forall i,j \in S$, coût⟨ch(i,j)⟩ = coût⟨$(a_0, a_1, .., a_n)$⟩ = $\sum_{k=0}^{k=n-1} w_{k,k+1}$

III.3. Graphe statique

III.3.1. Définition d'un graphe statique

Un graphe pondéré $G = (S,A)$ est dit statique si $\forall i,j \in S, w_{i,j}$ est une constante. On l'appelle alors graphe à pondération statique ou tout simplement graphe statique [Figure II-2].

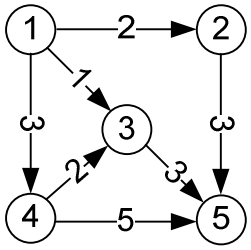

Figure III-2 - Graphe statique

III.3.2. Problème du plus court chemin dans un graphe statique

Cette problématique est aussi connue sous le nom de recherche du plus court chemin classique. Elle consiste à rechercher le chemin ayant le coût minimum liant un nœud i à un nœud j. On notera $Ch^*(i,j)$ le plus court chemin entre i et j. $Ch^*(i,j) = min_{ch(i,j)\in \mathrm{E_{ch}(i,j)}}(\mathrm{coût}\langle \mathrm{ch(i,j)}\rangle)$.

Dans la littérature de recherche opérationnelle et d'optimisation combinatoire, cette problématique a fait l'objet de nombreuses recherches, durant plusieurs décennies. Dans ce cadre, plusieurs algorithmes ont été mis en place pour traiter les spécificités des différents types de graphes (graphes cycliques, graphes avec valeurs négatives …). On classe généralement ces algorithmes dans deux catégories décrites dans les paragraphes suivants.

III.3.2.1. Algorithmes de correction d'étiquettes (Label Correction Algorithmes)

La méthode globale utilisée dans ce type d'algorithmes est d'associer chaque chemin à un coût initial et de corriger la valeur de ce coût au fur et à mesure du parcours des nouveaux nœuds. En effet, ces algorithmes reposent sur un marquage non définitif, un marquage qui évolue à chaque itération. Les algorithmes les plus connus qui s'inscrivent dans ce cadre sont l'algorithme de Bellman-Ford [**11**] [**49**] et l'algorithme de Moore [**78**]. Nous présentons dans ce qui suit un aperçu du fonctionnement de l'algorithme de Bellman-Ford.

La spécificité de cet algorithme est qu'il peut gérer des poids négatifs sur les arcs. De plus, il est capable d'identifier des cycles absorbants[1] dans le graphe.

Soit $G = (S, A)$ un graphe et $X \in S$ un nœud de départ (nœud source).

[1] Un circuit absorbant dans un graphe pondéré est un circuit dont la somme des poids des arcs qui le composent est négative.

On note Ω_k le coût du chemin menant du nœud source X au nœud k. $\Omega_k = \text{coût}\langle \text{ch}(\text{X}, \text{k})\rangle$.

Et $pred(k)$ le nœud prédécesseur du nœud k.

1. Initialisation :
   $\Omega_X = coût\langle ch(X,X)\rangle = 0$
   $\Omega_k = \infty, \forall k \in S - \{X\}$
   $pred(X) = X$

2. Pour i de 1 à n-1 faire
   $\forall a_{u,v} \in A$ faire
   si ($\Omega_u + w_{u,v} < \Omega_v$) alors
   $\Omega_v = \Omega_u + w_{u,v}$
   $pred(v) = u$

3. $\forall a_{u,v} \in A$ faire
   si ($\Omega_u + w_{u,v} < \Omega_v$) alors
   retourner « faux »

4. retourner « vrai »

Algorithme de Bellman-Ford

Le principe de son fonctionnement est de faire une (ré)estimation des poids des arcs au début de chaque itération et de calculer, par la suite, le chemin le plus court de la source vers tous les nœuds du graphe en se basant sur ces valeurs estimées. Ceci permet après plusieurs itérations d'atteindre la valeur minimale du plus court chemin. Si l'algorithme détecte un circuit de longueur négative, il retourne la valeur « Faux ».

On pose $card\langle S\rangle = n$ et $card\langle A\rangle = m$, la complexité de l'algorithme de Bellman-Ford dans le pire des cas est donc $O(nm)$. Ce qui correspond à $O(n^3)$ dans un graphe simple dense.

III.3.2.2. Algorithmes de création d'étiquettes (Label Setting Algorithms)

Contrairement aux algorithmes de correction d'étiquettes, le principe de ce type d'algorithmes est un marquage définitif de chaque nœud parcouru. L'algorithme le plus célèbre dans ce contexte est l'algorithme de Dijkstra[**35**]. Publié par l'informaticien néerlandais Edsger Dijkstra en 1959. C'est l'algorithme le plus cité dans la littérature de la recherche d'itinéraire. Il est de type glouton[1] et ne s'applique qu'aux graphes valorisés avec des poids positifs. Son principe est de valoriser le coût des chemins liant le nœud départ vers chaque nœud et de classer les nœuds selon cette valorisation.

[1] Un algorithme glouton est un algorithme qui suit le principe de faire, étape par étape, un choix optimum local, dans l'espoir d'obtenir un résultat optimum global. Dans le cas de l'algorithme de Dijkstra, à chaque itération on traite un nouveau nœud.

On note

Ω_k le coût du chemin menant du nœud source X au nœud k. $\Omega_k = coût\langle ch(X,k)\rangle$.
$pred(k)$ le nœud prédécesseur du nœud k.
$E_{courant}$ Ensemble de nœuds courants.

1. Initialisation :
   $\Omega_X = coût\langle ch(X,X)\rangle = 0$
   $\Omega_k = \infty, \forall k \in S - \{X\}$
   $pred(X) = X$ et $E_{courant} = \{X\}$

2. On sélectionne un nœud v tel que $v = min_{k \in E_{courant}}(\Omega_k)$
   $E_{courant} = E_{courant} - \{v\}$

3. alors
   $\Omega_k = \Omega_v + w_{v,k}$
   $pred(k) = v$

   Si $k \notin E_{courant}$ alors $E_{courant} = \{k\} \cup E_{courant}$

4. Si $E_{courant} = \emptyset$ alors fin de l'algorithme sinon aller à l'étape 2.

Algorithme de Dijkstra

L'implémentation la plus simple de l'algorithme de Dijkstra stocke la liste des sommets dans une liste liée. La sélection du nœud v (étape 2) dans ce cas est un simple parcours linéaire de la liste. La complexité est donc dans ce cas égale à $O(n^2)$.

III.3.3. Problème des K plus courts chemins dans un graphe statique

Également appelée KSP pour « K Shortest Path », cette problématique est une extension de la recherche d'un chemin optimal unique. Elle consiste à rechercher les *K* premiers chemins optimaux, *K* étant un entier positif. Autrement dit, on essaye de déterminer le premier, le second, …, le *K*-ième plus court chemin liant un nœud départ à un nœud d'arrivée. Bien que cette problématique soit moins étudiée dans la littérature que la problématique de recherche d'un unique chemin optimal, on trouve dans la littérature plusieurs centaines d'articles à ce sujet [**73**].

Comme dans la problématique de recherche d'un plus court chemin, les algorithmes sont divisés en deux grandes familles : les algorithmes à correction d'étiquettes et les algorithmes à création d'étiquettes. Nous nous intéressons dans la suite aux algorithmes de création d'étiquettes, car ils sont plus efficaces [**121**].

Nous cherchons plusieurs chemins dans un seul graphe, nous sommes amenés à maintenir pour chaque nœud *K* plusieurs étiquettes. Nous définissons une fonction, notée généralement *h*, qui permet de retrouver pour une étiquette, le nœud auquel elle appartient. Chaque étiquette correspondant à un

chemin et contient des informations concernant le nœud prédécesseur de K et le coût total du chemin menant du nœud source au nœud K associé à l'étiquette.

Dans la problématique des K plus courts chemins, on différencie la recherche des chemins pouvant contenir des boucles et les chemins ne pouvant pas contenir de boucles. Le premier problème à été résolu par Fox [**50**] avec un algorithme d'une complexité $O(m * K * log(n))$ au pire des cas. A la suite, Eppstein [**39**] a proposé un algorithme plus performant d'une complexité $O(m + n * log(n) + K)$. On s'intéresse plutôt au chemin ne pouvant pas contenir de boucles dans le contexte du transport de personne. Dans ce domaine, l'algorithme le plus utilisé est celui de Yen [**123**] avec une complexité $O(K * n * (m + n * \log(n)))$.

Nous présentons brièvement dans ce qui suit l'algorithme de K-plus courts chemins de Yen.

1. Initialisation
   $P^* =$ Dijkstra (X, Y) // Calcule du plus court chemin de X vers Y avec Dijkstra (voir §III.3.2.2)
   $ECh_{candidat} = \{P^*\}$ //Chemins candidat
   $ECh_{courant} = \{\}$ //Chemins en cours
2. $Pour\ i\ allant\ de$ 1 à $K\ faire$
   $P = Le\ plus\ court\ chemin\ dans\ ECh_{candidat}$
   $v = Le\ noeud\ de\ déviation\ entre\ P\ et\ tous\ les\ chemins\ de\ ECh_{courant}$
   $ECh_{courant} = ECh_{courant} \cup \{P\}$
   $Tant\ que\ v \neq Y\ faire$
   $Supprimer\ tous\ les\ noeuds\ de\ P\ de\ X$ à $v\ et\ leurs\ liens\ associés$
   $\forall s \in S, supprimer\ les\ arcs\ a_{X,s}\ si \exists\ ch\ \in ECh_{courant}\ tel\ que\ a_{X,s} \in ECh_{courant}$
   $P' =$ Dijkstra (v, Y)
   $P' = concat(P, P')$
   $ECh_{candidat} = ECh_{candidat} \cup \{P'\}$
   $Restaurer\ le\ graphe$
   $v = successeur(v, P)$

Algorithme de Yen

Son principe se fonde sur une notion de déviation de chemin, qui consiste à trouver le nœud à partir duquel le chemin le plus court diffère de tous les autres chemins stockés dans la liste $ECh_{candidat}$.

Le nœud de déviation d'un chemin ch par rapport à un ensemble de chemins $ECh = \{Ch_1, Ch_2, .., Ch_n\}$ est le premier nœud du chemin ch à partir duquel il y a une différence avec les autres chemins de ECh. Dans la [Figure III-3] le nœud de déviation du chemin ch par rapport à l'ensemble des chemins $ECh = \{Ch_1, Ch_2, \ldots, Ch_n\}$ est le nœud 0.

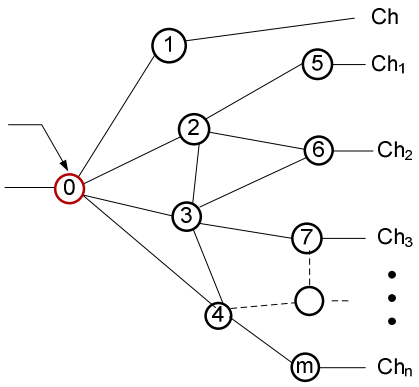

Figure III-3 - Déviation d'un chemin par rapport à un ensemble de chemins

Pour initialiser l'algorithme on calcule le plus court chemin entre X et Y en utilisant l'algorithme de Dijikstra. Ensuite, à chaque itération, on calcule le nœud de déviation entre le plus court chemin et les chemins candidats ($ECh_{condidat}$). On supprime ensuite les nœuds et les liens allant du nœud source jusqu'au nœud de déviation et on calcule le plus court chemin *P'* entre le nœud de déviation et le nœud de destination. On concatène, par la suite, le chemin *P'* au sous-chemin retenu entre le nœud source et le nœud de déviation.

III.4. Graphe dynamique

III.4.1. Définition d'un graphe dynamique

Le graphe $G = (S, A)$ est dit à pondération dynamique si le poids d'un arc est une valeur qui dépend du temps. On associe alors à chaque arc $a_{i,j} \epsilon A$ du graphe G une fonction $w(i, j, t)$ (on utilise aussi la notation $w_{i,j}(t)$) où t représente la date de départ i. Un chemin entre i et j dépend donc du temps, on étend alors la définition d'un chemin qu'on notera désormais $Ch(i, j, t)\ tel\ que\ i, j \in S\ et\ t$ représentant l'heure de départ.

Ce graphe est appelé 'graphe à pondération dynamique' ou tout simplement graphe dynamique. Ce type de graphe est généralement utilisé dans le domaine de transport où des réseaux informatiques.

Le temps peut être considéré comme « continu » ou « discret » selon le contexte et le domaine d'application du graphe. Cependant, dans la littérature, nous remarquons que la majorité des travaux ont considéré le temps comme « discret ». Cette hypothèse sera appliquée dans notre travail pour des raisons de simplification.

Une première représentation du temps peut être celle proposée par Hizem[**58**] où le temps est représenté par une fonction constante par intervalle et par rapport au temps $w_{i,j}(t)$. Le temps est donc divisé en périodes égales de longueur T. La fonction du poids est égale à une constante durant chaque période T. Autrement dit : si $t, t' \in [kT, (\mathrm{k}+1)\mathrm{T}]$ alors $w_{i,j}(t) = w_{i,j}(t')$.

Nous optons plutôt pour la représentation proposée par Chabini [**19**] où il associe à chaque arc $a_{i,j} \in A$ une liste de départs possibles notés $D = \{d_{i,j}(t)/a_{i,j} \in A\}$ et leurs coûts correspondants $C = \{c_{i,j}(t)/a_{i,j} \in A \; et \; c_{i,j}(t) = \mathrm{coût}\langle \mathrm{ch(i,j,t)} \rangle\}$. Dans cette représentation nous ne fixons pas d'intervalle T, ce qui correspond à la réalité de la problématique traitée. En effet, dans un réseau de bus, par exemple, les horaires de départ des différents bus ne sont pas forcément périodiques, selon une période unique.

III.4.2. Problème du plus court chemin dans un graphe dynamique

Bien que la recherche du plus court chemin fût l'un des problèmes fondamentaux du XXème Siècle, on constate que les recherches s'orientent massivement vers le réseau dynamique dans les deux dernières décennies. La problématique de recherche du plus court chemin dans un graphe dépendant du temps aussi appelée « recherche du plus court chemin dynamique » a été formulée, pour la première fois, par Cooke et Halsey [**23**]. Avec cette contrainte temporelle, on essaye de chercher le chemin le plus rapide qui nous mène à destination. On se place sur un nœud de départ X avec une heure d'arrivée au nœud de départ t_0 et on calcule le chemin qui va nous permettre d'atteindre le nœud destination Y le plus tôt possible. On parle alors du problème du chemin le plus rapide. Mais, par abus de langage, on utilise également le terme « problème du plus court chemin », sachant que, dans ce contexte, le coût sera calculé en termes de temps.

Dans ce type de problème, notre objectif est de trouver le temps d'arrivée « au plus tôt » à chaque nœud. On modifie alors la notation afin d'avoir un renseignement sur le coût du chemin allant de la source à la destination si on part de la source à une heure fixe t_0. On note alors $\Omega_k(t_0)$ l'heure d'arrivée au plus tôt au nœud k si on part du nœud source X à t_0 ($\Omega_k(t_0)$ représente le coût dans un graphe dynamique). On remarque que, dans la notation $\Omega_k(t_0)$, on n'indique pas le nœud source étant donné que ce dernier est fixe et connu dés le début.

Dans ces conditions, on a : $\Omega_k(t_0) = min_{i \in N^-(j)} \left(min_{t \geq \Omega_i(t_0)} \left(t + w_{i,j}(t) \right) \right) si \; j \neq X \; et \; t_0 \; sinon$

III.4.2.1. Graphe FIFO et graphe non-FIFO (First In First Out)

Un graphe est dit FIFO si tous ses arcs sont FIFO. Un graphe contenant au moins un arc non-FIFO est dit non-FIFO.

Un arc $a_{i,j} \in A$ est dit FIFO si la condition suivante est réalisée

$\forall t, t' \geq 0, t \leq t' \Rightarrow t + w_{i,j}(t) \leq t' + w_{i,j}(t')$

La propriété FIFO est aussi appelée « propriété de non-dépassement » (non-overtaking property) car elle stipule que si un trajet P_1 part du nœud i à l'instant t et un autre trajet P_2 part de i à $t \leq t'$. P_2 ne peut pas arriver à j avant P_1 en utilisant l'arc $a_{i,j}$. P_1 ne peut donc pas dépasser P_2.

III.4.2.2. Algorithmes dans les Graphes FIFO

Orda et Rom [**86**] ont prouvé que la recherche du plus court chemin dynamique avec une heure de départ est difficile. Mais le problème demeure polynomial dans le cas d'un graphe FIFO. Dans la littérature de recherche du plus court chemin dans un graphe dynamique, on peut trouver plusieurs variantes qui vont de la recherche « de l'heure d'arrivée ***au plus tôt*** à partir de n'importe quel nœud du graphe vers tous les nœuds si on part à l'instant t » à la « recherche de l'heure d'arrivée ***au plus tard*** à un nœud j si on part d'un nœud i à l'instant t ». Dans ses travaux, Dean [**28**] a énuméré seize variantes de ce problème et il a montré que ces variantes sont des cas particuliers de deux problématiques fondamentales. Le premier problème étant le calcul du plus court chemin d'un nœud source vers tous les autres nœuds du graphe pour une date de départ du nœud source fixé. Ce problème est aussi appelé Un-à-Plusieurs plus court chemin (One-To-All Shortest Path OTASP)[**20**]. Le deuxième est le calcul du plus court chemin d'un nœud source vers tous les autres nœuds du graphe pour toutes les dates de départ possibles du nœud source. Ce problème est aussi appelé Plusieurs-à-Un plus court chemin (All-To-One Shortest path ATOSP).

III.4.2.3. OTASP

Pour résoudre le problème de calcul du plus court chemin d'un nœud source vers tous les autres nœuds du graphe dynamique pour une date de départ, plusieurs algorithmes existent et sont classés dans les deux catégories « Algorithmes à correction d'étiquettes » et « Algorithmes à fixation d'étiquettes ». Seulement, dans ce cadre, le seul avantage des algorithmes à correction d'étiquettes est la simplicité. En effet, les algorithmes de fixation d'étiquettes sont beaucoup plus performants. On s'intéressera dans la suite à ce type d'algorithme. Le résultat le plus célèbre dans ce contexte est que quand la propriété FIFO est valide, une généralisation (adaptation) de l'algorithme de Dijkstra résout le problème avec la même complexité que dans le problème d'acheminement classique. Ce résultat a été prouvé une première fois par Ahn et Shin [**2**] ensuite par Kaufman et Smith [**66**] et une preuve plus simple a été proposée ensuite par Chabini [**20**].

Nous présentons dans ce qui suit la version dynamique de l'algorithme de Dijkstra. A partir d'un graphe dynamique $G = (S, A)$, on cherche le chemin le plus rapide entre X et Y en partant de X à une heure $t = t_0$. Vu qu'on est dans un graphe FIFO, on a $min_{t \geq \Omega_i(t_0)} \left(t + w_{i,j}(t)\right) = \Omega_i(t_0) + w_{i,j}\left(\Omega_i(t_0)\right)$ ainsi $\Omega_k(t_0) = min_{i \in N^-(j)} \left(\Omega_i(t_0) + w_{i,j}\left(\Omega_i(t_0)\right)\right) si\ j \neq X\ et\ t_0\ sinon.$

Dans cet algorithme, on notera $\mathrm{pred}_{t_0}(k)$ le nœud prédécesseur au nœud k si on part du nœud de départ X à t_0 et $E_{courant}$ l'ensemble de nœuds courants.

L'algorithme de Dijkstra dans un graphe dynamique commence par initialiser les temps d'arrivée au plus tôt $\Omega_k(t_0)$ de tous les nœuds k à la valeur ∞. Il est de même pour leurs nœuds prédécesseurs $pred_{t_0}(k)$. A chaque itération, l'algorithme sélectionne un nœud à explorer dans l'ensemble $E_{courant}$ qui contient seulement le nœud source au début de l'algorithme.

1. Initialisation :
   $\Omega_X(t_0) = t_0$
   $\Omega_k(t_0) = \infty \; et \; pred_{t_0}(k) = \infty, \; \forall k \in S - \{X\}$
   $pred_{t_0}(X) = X$ et $E_{courant} = \{X\}$

2. On sélectionne un nœud v tel que $v = min_{k \in E_{courant}}(\Omega_k(t_0))$
   $E_{courant} = E_{courant} - \{v\}$

   $\forall \, k \in N^+(v) \; faire$
   $si \; (\Omega_v(t_0) + w_{v,k}(t_0) < \Omega_k(t_0)) \;$ alors
   $\Omega_k(t_0) = \Omega_v(t_0) + w_{v,k}(t_0)$
   $pred_{t_0}(k) = v$

   Si $k \notin E_{courant}$ alors $E_{courant} = \{k\} \cup E_{courant}$

3. Si $E_{courant} = \emptyset$ alors fin de l'algorithme sinon aller à l'étape 2.

Extension de l'algorithme de Dijkstra pour un graphe dynamique

La sélection du nœud v se fait selon la meilleure offre d'heure d'arrivée au plus tôt $\Omega_v(t_0)$. Le nœud est retiré de l'ensemble des nœuds courants. On explore ensuite tous les nœuds accessibles à partir de v et on met à jour leurs étiquettes $\Omega_k(t_0)$ et $pred_{t_0}(k)$. Une fois l'ensemble $E_{courant}$ vide, cela indique qu'on a obtenu le chemin optimal de la source X à la destination Y et le chemin est retrouvé par chainage arrière de Y vers X. Ceci est réalisé en regardant la valeur de l'étiquette $pred_{t_0}$ en partant du nœud Y.

III.4.2.4. ATOSP

En 1997, Chabini [**18**] propose un algorithme qui permet de calculer le chemin le plus court dans un graphe dynamique de plusieurs nœuds sources à un nœud destination pour plusieurs heures de départs. Le concept de l'algorithme de Chabini appelé DOT (Decreasing Order of Time) est l'affectation des étiquettes aux nœuds dans un ordre décroissant par rapport au temps. Il réduit ainsi l'intervalle de temps concerné jusqu'à l'obtention d'un seul point temporel. L'algorithme permet de convertir l'intervalle de temps initial $[t_d, t_a]$ en k points temporels et de construire un graphe statique pour chaque point temporel. Donc, en faisant k copies du graphe initial.

Dans l'algorithme DOT, on suppose que la fonction $d_{i,j}(t)$ prend une valeur statique après un nombre fini « M» d'intervalles de temps. Ainsi l'ensemble $D = \{d_{i,j}(t)/a_{i,j} \in A\}$ contenant les temps de départs possibles contient M éléments donc $\text{card}\langle D\rangle = M$. D'où $(si\ t > M-1\ alors\ \Omega_i(t) = \Omega_i(M-1)\ \forall i \in S)$. On recherche les plus courts chemins de tous les nœuds vers un nœud destination 'ar' pour toutes les heures de départs. Dans l'initialisation de l'algorithme DOT, on a recours à un algorithme de recherche de Plus Court Chemin Statique ($PCCS$). Cet algorithme pourrait être tout simplement l'algorithme de Dijkstra dans un graphe statique.

1. Initialisation :
   $\Omega_k = \infty, \forall k \in S - \{ar\}$ et $\Omega_{ar} = 0, \forall t < M-1$
   $\Omega_k(M-1) = PCCS\big(w_{i,j}(M-1), q\big), \forall i$

2. Pour $t = M-2$ à 0 faire
   Pour $i, j \in S$ faire
   $$\Omega_i(t) = \min\Big(\Omega_i(t), w_{i,j}(t) + \Omega_j\Big(t + w_{i,j}(t)\Big)\Big)$$

Algorithme DOT de Chabini

La complexité de l'algorithme DOT est $O(PCCS + nM + mM)$. Donc si on utilise l'algorithme de Dijkstra comme PCCS on obtient une complexité au pire des cas $O(n^2 + nM + mM)$.

III.4.2.5. Algorithmes dans les Graphes non-FIFO

Pour résoudre le problème de calcul du plus court chemin dans un graphe dynamique non-Fifo, Pallottino et Scutella [**90**] proposent un nouveau paradigme algorithmique appelé Chrono-SPT basé sur une visite chronologique des nœuds du graphe. Ils définissent une liste appelée Seau-Liste (bucket-list noté B) qui servira de support de structure de donnée dans l'algorithme. Le Seau-Liste B contiendra n éléments $B = \{B_1, B_2 .. B_n\}$ chaque élément constitue l'ensemble des nœuds visités à un instant t.

Sélectionner i de B_k

$B_k = B_k - \{i\}$

$\forall j \in N^+(i)$ *faire*

$t_h = t_k + w_{i,j}(t_k)$

Si $\Omega_i(t_k) + w_{i,j}(t_k) < \Omega_j(t_h)$ *alors*

$Pred_j(t_h) = i_k$

Si $j \notin B_h$ *Alors*

$B_h = B_h \cup \{j\}$

Fin Si

Fin Si

Algorithme de Pallottino et Scutella pour les graphes Non-Fifo

L'algorithme traite chronologiquement les différents sous-ensembles de B. En effet, dans l'itération « k » de l'algorithme, on traite le sous-ensemble B_k. Le traitement consiste à parcourir tous les nœuds contenus dans $B_k$et à mettre à jour les étiquettes de leurs nœuds successeurs.

Chabini [**21**] propose une autre solution au problème en enrichissant l'algorithme de Djikstra adapté aux graphes dynamiques par une liste qui sauvegarde les arrivées possibles à un nœud et les nœuds prédécesseurs pour chaque heure d'arrivée possible.

III.4.3. Problème des K-plus courts chemins dans un graphe dynamique

Dans la pratique, et surtout dans le domaine du transport, trouver le meilleur chemin n'est pas toujours suffisant. Il est tout aussi intéressant de trouver la liste des meilleurs chemins ou des meilleures routes. Ceci permet d'avoir le choix dans une liste de solutions ou d'avoir une alternative à la meilleure solution.

Bien que la littérature de la recherche opérationnelle regorge de documents et d'algorithmes permettant la recherche des K-plus courts chemins dans un réseau statique. Il nous a été difficile de trouver les algorithmes traitant le cas d'un réseau dynamique.

La notation dans cette problématique, à savoir la TD-KSP Time Dependent K-Shortest Path, est la même que dans le problème de recherche du plus court chemin dans un graphe dynamique. C'est seulement le résultat de recherche qui change. En effet, dans ce contexte, on recherche le $1^{er}, 2^{ème}, \dots, K^{ième}$ (K une constante fixée) plus court chemin dans un graphe dynamique $G = (S, A)$ pour aller d'un nœud de départ d à un nœud d'arrivée a. Le départ du nœud a étant à une heure précise $t = t_0$.

Dans ses travaux, Subramanian [**104**] étudie deux algorithmes pour trouver les K-plus courts chemins dans un graphe dynamique. Le premier est un algorithme à correction d'étiquette créé par une hybridation des algorithmes de K-SP et de TDSP proposés dans le cadre du développement du logiciel DYNASMART à l'Université de Texas[**130**].

Le deuxième algorithme qu'on notera TD-KSP est dérivé de l'algorithme de Dreyfus[**37**]. Il est avantageux par rapport au premier si l'une des hypothèses suivantes est réalisée:

- L'ensemble optimal des chemins recherchés ne concerne qu'un petit nombre d'heures de départ.

- Le nombre d'heures de départs possibles (cardinal de D) est très grand par rapport à la valeur de la constante K.

Dans le cas d'un réseau de transport, la deuxième hypothèse est généralement réalisée, pour cela on s'intéresse particulièrement à cet algorithme.

On définit $\Omega_j^k(t_a)$ l'étiquette du $k^{ième}$ plus court chemin dynamique allant du nœud origine a à un nœud j avec $j \in S$ et partant du nœud a à l'instant $t = t_a$. L'étiquette $\Omega_j^k(t_a)$ contient une liste d'heures d'arrivée au nœud auquel elle est associée.

On définit une liste d'éligibilité qu'on notera EL qui contiendra les nœuds éligibles pour les traitements ultérieurs. Enfin, on notera ${E_p}^j(t)$ la valeur de l'étiquette provisoire du nœud j pour une heure de démarrage t.

1. Initialisation :
   $\Omega_a^k(t_a) = \{0, \infty, \dots, \infty\}, \forall\, t_a \; et \; k = 1,2, \dots, K$
   $\Omega_i^k(t_a) = \{\infty, \infty, \dots, \infty\}, \forall i \neq a, \forall\, t_a \; et \; k = 1,2, \dots, K$
   $EL \leftarrow d$
   $i \leftarrow a$

2. Pour chaque $j \in N^+(i), \forall t_s, k = 1,2, \dots, K$ faire
   $${E_p}^j(t) = d_{i,j}\left(\Omega_i^k(t_a)\right) + \Omega_i^k(t_a)$$

Conserver les K plus petites valeurs entre ${E_p}^j(t_a)$ et $\Omega_i^k(t_a)$.

Si l'étiquette d'un nœud $i \in EL$ n'a pas été examinée $alors$ supprimer $i \; de \; EL$.
Si un élément quelconque du nœud j a changé $alors$ insérer $j \; dans \; EL$.
$Si \; EL$ est vide $alors$ arrêter l'algorithme.

Algorithme TD-KSP

Wu et Hartley [**122**] ont proposé un algorithme de recherche des K-Plus courts chemins dynamiques qui permet de trouver les chemins optimaux selon les préférences des utilisateurs. L'algorithme

commence par classer d'une manière croissante les chemins optimaux (minimisant le temps de voyage) ; ensuite, ces chemins sont comparés en fonction des préférences et des besoins des usagers. Enfin, le chemin qui coïncide le plus aux besoins des utilisateurs est sélectionné comme « chemin optimal ». L'algorithme prend en compte deux types de moyens de transport : déplacement en bus ou déplacement à pied. Il gère, d'autre part, la particularité des deux types de déplacements.

D'autres travaux se sont plutôt intéressés à la résolution du problème des K-plus courts hyper-chemins dynamiques [**83**]. Un hyper-chemin est une extension de la notion de chemin dans un hypergraphe. L'hypergraphe, quant à lui, est une généralisation d'un graphe. Son avantage réside dans le fait qu'un nœud peut se connecter non seulement à un autre nœud, mais aussi à un ensemble de nœuds.

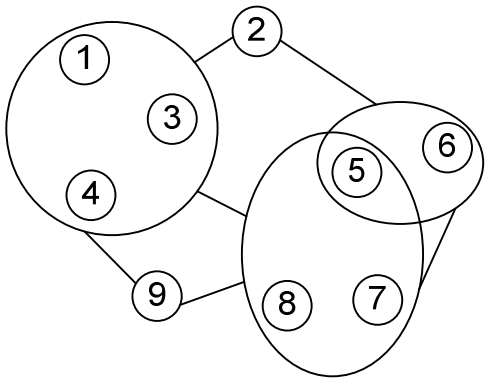

Figure III-4 - Exemple d'hypergraphe

Formellement, un hypergraphe H est le couple $H = (S, E)$ où S est un ensemble de nœuds (ou sommet) et E est un ensemble de sous-ensembles non vides appelés hyperarêtes.

Dans l'exemple d'hypergraphe [Figure III-4], $S = \{1,2,3,4,5,6,7,8,9,\}$ et $E = \{\{1,3,4\}, 2, \{5,6\}, \{5,7,8\}, 9\}$.

Les hypergraphes sont utilisés dans plusieurs domaines, dont celui du transport urbain [**82**]. En effet, dans un réseau de transport en commun composé d'un ensemble de lignes de bus interconnectées, le passage des bus est indiqué uniquement par une fréquence. Il est donc impossible d'appliquer les algorithmes de recherche des plus courts chemins déterministes. Pour modéliser ces graphes, on utilise des méthodes stochastiques qui associent à chaque arc et à chaque heure de départ une probabilité. Ainsi, le successeur d'un nœud dans un graphe n'est plus un simple nœud, mais un ensemble de nœuds où chacun d'entre eux est associé à une valeur de probabilité. Plus de détails sur les notions d'hypergraphe et d'« hyperpath » sont disponibles dans [**53**].

III.5. Graphe distribué

III.5.1. Définition d'un système distribué

Un système distribué est composé de plusieurs sous-systèmes autonomes et d'un serveur de calcul central. Un sous-système gère sa propre base de données et il peut répondre aux demandes du serveur de calcul [Figure III-5]. Ce dernier n'a pas une vue globale des bases de données. Il ne peut qu'envoyer des requêtes aux différents sous-systèmes pour récupérer des informations. Les sous-systèmes peuvent présenter des chevauchements à certains endroits, on parle alors de points d'intersection entre les différents sous-systèmes.

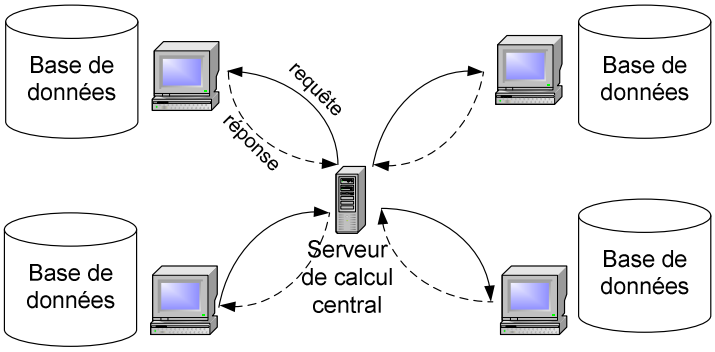

Figure III-5 - Exemple de système distribué

Ce type de système distribué est généralement associé aux systèmes d'information pour le transport distribué tel que c'est le cas pour le système Delfi décrit dans le (paragraphe I.2.4.3.2) [**111**].

Il a été mis en place dans le cadre du développement d'un système d'information pour les voyageurs qui interagit avec des bases de données de diverses entreprises.

Suite à cette définition, on peut voir que notre sujet rejoint une problématique de système distribué. En effet, nous essayons de faire communiquer des systèmes d'information distincts exploitant des bases de données indépendantes. On est donc dans le cadre d'un système distribué où notre système représente le serveur de calcul central et les systèmes d'information existants représentent les sous-systèmes.

III.5.2. Définition d'un graphe distribué

Un graphe distribué est une représentation d'un système distribué. On associe à chaque sous-système i une classe qu'on notera C_i. Une classe C_i est décrite par un graphe statique G_i avec $G_i = (S_i, A_i)$. S_i et A_i représentent respectivement l'ensemble des sommets et l'ensemble des arcs de la classe C_i.

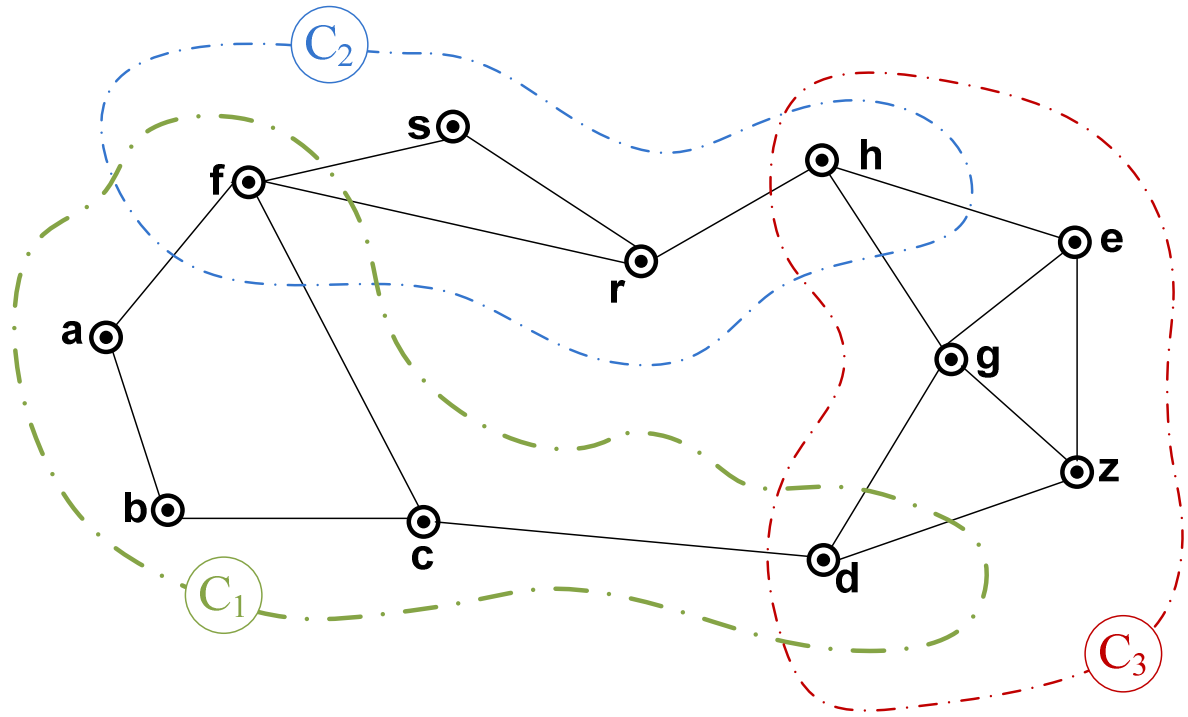

Figure III-6 - Graphe distribué

Pour $x, y \in S_i$, on note $a^i_{x,y}$ l'arc menant de x à y s'il existe un chemin entre x et y inclus dans la classe C_i.

Le graphe global $G = (S, A)$ est un graphe virtuel formé par l'association des graphes des n classes. On a alors $S = \bigcup_{i=1}^{n} S_i$ et $A = \bigcup_{i=1}^{n} A_i$. On parle de graphe virtuel car il n'est pas enregistré ni dans le serveur central de calcul ni dans aucun des sous-systèmes. C'est une schématisation théorique de la combinaison des différents graphes des classes.

Si un nœud appartient à deux classes différentes, nous parlons alors de « nœud d'intersection » (pôle d'intersection ou nœud de correspondance). Formellement, si $s \in S_i$ et $s \in S_j$ alors s est un nœud d'intersection des classes C_i et C_j.

On étend la notion de poids (valorisation des arcs) aux graphes distribués. On note alors $w_{x,y}$ le poids de l'arc $a_{x,y}$. Or dans un graphe distribué, nous pouvons avoir entre deux nœuds x,y plusieurs arcs. Chacun d'entre eux appartenant à une classe différente. Donc la valorisation de cet arc dans une classe C_i sera différente de sa valorisation dans la classe C_j. On distingue alors la valorisation d'un arc $a^i_{x,y}$ dans la classe C_i par rapport aux autres, en notant $w^i_{x,y}$ le poids de l'arc $a^i_{x,y}$ dans la classe C_i. Par exemple dans la [Figure III-7] l'arc $a_{X,Y}$ appartient en même temps à la classe C_i et à la classe C_j, il aura donc deux valorisations possibles $w^i_{X,Y}$ pour l'arc $a^i_{X,Y}$ et $w^j_{X,Y}$ pour l'arc $a^j_{X,Y}$.

De manière similaire à un graphe statique, nous définissons un chemin entre deux nœuds noté $Ch(i,j)$ avec $i,j \in S$ comme une succession d'arcs liants i à j. Une spécificité du graphe distribué est qu'entre deux nœuds, on peut avoir plusieurs chemins qui ne sont pas totalement inclus dans une seule classe. On distingue alors un chemin local (intra-classe) et un chemin global (inter-classe). Un chemin est dit local à une classe C_i si tous les arcs qui le composent sont inclus dans la classe C_i. Par exemple le chemin (A,B,C,D) est un chemin local à la classe C_i vu que les arcs $a_{A,B}, a_{B,C}, a_{C,D} \in A_i$. Le chemin est dit global si les arcs qui le composent n'appartiennent pas tous à une même classe. Exemple le chemin (A,X,L,M) est un chemin inter-classe vu qu'il est composé de $a_{A,X} \in A_i \; et \; a_{X,L}, a_{L,M} \in A_j$.

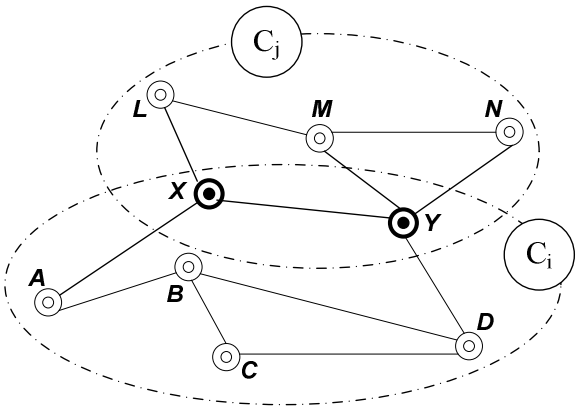

Figure III-7 - Valorisation d'un arc dans deux classes

Le coût d'un chemin est la somme des coûts des arcs qui le composent. Chaque arc étant valorisé par rapport à la classe à laquelle il appartient. $\forall x,y \in S$, coût⟨ch(x,y)⟩ = coût$\langle(a_0^{i_0}, a_1^{i_1}, .., a_m^{i_m})\rangle = \sum_{k=0}^{k=m-1} w_{k,k+1}^{i_k}$. Avec m le nombre d'arcs qui composent le chemin, $i_k \in [0..n]$ et n est le nombre de classe dans le système.

III.5.3. Problème du plus court chemin dans un graphe distribué

Le problème consiste à rechercher le chemin ayant le coût minimum liant un nœud x à un nœud y tel que $x,y \in S$. Les nœuds $x \; et \; y$ peuvent appartenir à n'importe quelle classe. Ainsi le chemin ch(x,y) peut être un chemin inter-classe.

On suppose que $E_{ch}(x,y)$ est l'ensemble des chemins qui lient $x \; et \; y$. On notera $Ch^*(x,y)$ le plus court chemin, donc formellement nous écrivons : $Ch^*(x,y) = min_{ch(x,y)\in E_{ch}(x,y)}$(coût⟨ch(x,y)⟩).

L'objectif est donc de déterminer ce chemin global $Ch^*(x,y)$.

Les recherches dans ce domaine sont très rares, nous pouvons citer quelques approches heuristiques qui permettent de trouver le meilleur chemin dans certaines conditions, mais ne garantissent pas l'optimalité du résultat. Parmi ces approches nous pouvons citer [**51**] [**65**] [**125**].

Wang et Kaempke [**111**] ont proposé le premier algorithme polynomial qui permet de résoudre le problème du plus court chemin dans un système distribué et qui garantit l'optimalité de la solution trouvée.

Leur travail se réfère à la définition des deux valeurs suivantes :

$INTRA(\mathrm{x},\mathrm{y})_{i,k}$ désigne le poids du $k^{\text{ème}}$chemin intra-classe de la classe C_i entre les sommets $x\ et\ y$. Supposons qu'il existe m chemins intra-classe de C_i entre $x\ et\ y$, le poids du chemin le plus court intra-classe dans C_i est noté $Ch^*(\mathrm{x},\mathrm{y})_{\mathrm{i}} = min_{k=1,2,..,m}\{INTRA(\mathrm{x},\mathrm{y})_{i,k}\}$.

$INTER(\mathrm{x},\mathrm{y})_i$ désigne le poids du $i^{\text{ème}}$chemin inter-class entre les sommets $x\ et\ y$ dans le graphe global virtuel $G(S,A)$. Supposons qu'il existe M chemins inter-classe de C_i entre $x\ et\ y$, le poids du plus court chemin inter-class entre $x\ et\ y$ est calculé de la manière suivante $Ch^*(\mathrm{x},\mathrm{y}) = min_{k=1,2,..,M}\{INTER(\mathrm{x},\mathrm{y})_i\}$.

L'approche de Wang et Kaempke repose sur la construction de deux nouveaux graphes : graphe complet d'intersection et graphe d'intersection étendu.-

- L'algorithme qu'ils proposent est le suivant :

1. Construire le graphe d'intersection complet G_{cint} associé à $G = (S,A)$
2. Construire le graphe d'intersection étendu G_{eint} associé à $G = (S,A)$
3. Calculer le plus court chemin $Ch^*(\mathrm{x},\mathrm{y}) = (x = s_1, s_2, .., s_p = y)$ dans G_{eint} en utilisant l'algorithme de Dijkstra
4. Pour chaque paire de nœuds (s_k, s_{k+1}) du chemin $Ch^*(\mathrm{x},\mathrm{y})$ faire
   - Déterminer la classe C_i de l'arc $a_{k,k+1}$
   - Déterminer le détail de $Ch^*(\mathrm{k},\mathrm{k}+1)_{\mathrm{i}}$

Algorithme DSRA

Dans ce qui suit, nous considèrerons le graphe distribué de la [Figure III-6] pour illustrer l'utilisation de l'algorithme DSRA. On suppose qu'on recherche le chemin le plus court entre le nœud « a » et le nœud « z ». On déroule alors l'algorithme, on commence par construire le graphe complet d'intersection ensuite le graphe d'intersection étendu.

III.5.3.1. Graphe complet d'intersection

Le graphe d'intersection complet $G_{cint} = (S_{cint}, A_{cint})$ est créé à partir des informations concernant l'intersection des différents sous-graphes G_i et les meilleures offres locales proposées par les classes aux nœuds S_{cint}. Le graphe d'intersection complet contient autant d'informations que nécessaire pour une optimisation inter-classe et il est complètement construit à partir de données fournies par les sous-systèmes.

- Les nœuds du graphe d'intersection complet S_{cint} sont les nœuds d'intersection du graphe global virtuel $G(S, A)$. En d'autres termes : $S_{cint} = \cup_{i=1,j=1}^{n}(S_i \cap S_j)$.

  $x \in S_{cint}$ **si et seulement si** x est un nœud d'intersection dans $G(S, A)$.

- Les arcs du graphe d'intersection sont définis comme suit : Si, dans le graphe global, il existe un chemin liant deux nœuds d'intersection (nœuds appartenant à S_{cint}), alors il y a un arc dans le graphe d'intersection qui lie ces deux mêmes nœuds. Le poids de cet arc est égal au coût du plus court chemin entre ces nœuds. En d'autres termes**,** il y a un arc $a_{x,y} \in A_{cint}$ **si et seulement si** x et y sont des nœuds d'intersection ($x, y \in S_{cint}$), ils sont liés dans une classe C_i et l'arc $a_{x,y}$ représente le plus court chemin entre x et y dans les différentes classes ($\exists i \in [0..n]\ tel\ que\ x, y \in S_i$ et $a_{x,y} = Ch^*(\mathrm{x,y})$).

Ainsi si x et y sont deux nœuds du graphe d'intersection, et $a_{x,y}$ est un arc du graphe d'intersection alors le poids de l'arc $a_{x,y}$ est égal au minimum des coûts des chemins possibles entre x et y dans les différentes classes. L'arc $a_{x,y}$ représente donc la meilleure offre locale possible entre $x\ et\ y$ en comparant les offres proposées par les différentes classes.

On suppose qu'il existe plusieurs classes offrant un chemin entre x et y. Donc le couple $\{x, y\}$ appartient à plusieurs classes. On note $I(x, y)$ l'ensemble des index de ces classes. On a donc $I(x, y) = \{i\ /\ \{x, y\} \in C_i\ \}$.

Wang et Kaempke proposent d'affecter à chaque arc $a_{x,y}$ deux étiquettes :

- ❖ Une première étiquette indiquant le coût de l'arc dans G_{cint}

$$w_{x,y} = min_{i \in \mathrm{I}(x,y)} co\hat{u}t(ch_i^*(x, y))$$

- ❖ Une deuxième étiquette rec indiquant la classe à laquelle appartient le chemin retenu pour représenter l'arc $a_{x,y}$.

$$rec(x, y) = k \textbf{ si } w_{x,y} = co\hat{u}t(ch_k^*(x, y)) \text{ avec } k \in I(x, y)$$

Pour illustrer le calcul du graphe complet d'intersection, nous présentons ici le graphe d'intersection issu du graphe distribué de la [Figure III-6].

Pour simplifier, nous considèrerons que $w^{i}_{X,Y} = w^{i}_{Y,X}$. Autrement dit, le sens du parcours d'un arc n'impacte pas son poids. Nous utiliserons les poids suivants :

C_1	$w^1_{a,b}$=5 $w^1_{b,c}$=3 $w^1_{c,d}$=15 $w^1_{c,f}$=5 $w^1_{a,f}$=2
C_2	$w^2_{f,s}$=2 $w^2_{s,r}$=2 $w^2_{f,r}$=5 $w^2_{r,h}$=2
C_3	$w^3_{h,z}$=7 $w^3_{h,g}$=2 $w^3_{e,g}$=1 $w^3_{e,z}$=2 $w^3_{d,e}$=4 $w^3_{d,g}$=2 $w^3_{g,z}$=4

On abouti alors au graphe d'intersection de la [Figure III-8] en se basant sur les calculs suivants :

- $I(f,h) = \{2\},\ I(f,d) = \{1\}$ et $I(d,h) = \{3\}$.
- $w_{f,d} = min_{i \in I(f,d)} coût(ch_i^*(f,d)) = coût(ch_1^*(f,d)) = w^1_{f,c} + w^1_{c,d} = 5 + 15 = 20$
- $w_{f,h} = min_{i \in I(f,h)} coût(ch_i^*(f,h)) = coût(ch_2^*(f,h)) = w^2_{f,s} + w^2_{s,r} + w^2_{r,h} = 2 + 2 + 2 = 6$
- $w_{d,h} = min_{i \in I(d,h)} coût(ch_i^*(d,h)) = coût(ch_3^*(d,h)) = w^3_{h,g} + w^3_{g,d} = 2 + 2 = 4$
- Donc $rec(f,d) = 1, rec(f,h) = 2$ et $rec(d,h) = 3$.

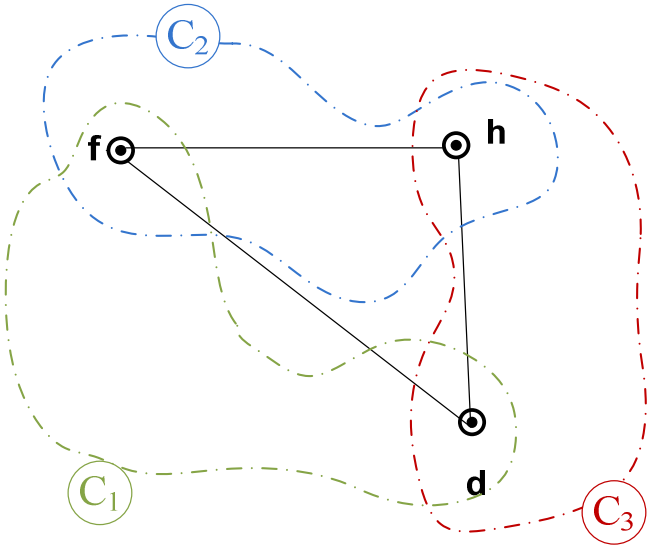

Figure III-8 - Exemple de graphe d'intersection complet

On passe donc à l'étape suivante de construction du graphe d'intersection étendu.

III.5.3.2. Graphe d'intersection étendu

Le graphe complet d'intersection ne contient pas nécessairement les nœuds de départ et d'arrivée.

Il est donc nécessaire de définir un nouveau graphe appelé graphe d'intersection étendu $G_{eint} = (S_{eint}, A_{eint})$ qui contiendra en plus du graphe G_{cint}, les nœuds de départ et d'arrivée et leurs liaisons avec le G_{cint}.

Le graphe G_{eint} est donc composés des nœuds du graphe G_{cint} et des nœuds de départ et d'arrivée, des arcs qui existent dans G_{cint} et des arcs liant les deux nœuds de départ et d'arrivée aux autres nœuds d'intersection.

Comme dans le G_{cint} les arcs qu'on ajoute dans G_{eint} représentent les plus courts chemins locaux liant les nœuds de départ et d'arrivée aux nœuds de S_{cint}.

Nous supposons que nous cherchons le meilleur chemin entre les nœuds $X\ et\ Y$ dans un graphe global $G = (S, A)$. On note $G_{cint} = (S_{cint}, A_{cint})$ le graphe d'intersection associé à G.

Si $X \notin S_{cint}$ alors $\exists!\ i_X \in [1..n]$ tel que $X \in S_{i_X}$. En effet, si X n'est pas un nœud d'intersection, alors X appartient nécessairement à une seule classe C_{i_X} (i_X étant l'indice de cette classe).

On note les nœuds d'intersection contenus dans la classe C_{i_X} par $S_{i_X}^{int}$. Donc $S_{i_X}^{int}$=$S_{i_X} \cap S_{cint}$ de même pour $S_{i_Y}^{int}$=$S_{i_Y} \cap S_{cint}$. On cherche alors les plus courts chemins possible entre X et les nœuds contenus dans $S_{i_x}^{int}$. Vu que X et les nœuds contenus dans $S_{i_x}^{int}$ appartiennent tous à la même classe il est possible de calculer ce type de plus court chemin en utilisant l'algorithme de Dijkstra par exemple.

Nous définissons l'ensemble $E_{cint}^{X} = \{a_{X,v}\ tel\ que\ v \in S_{i_X}^{int}\}$ avec $w_{X,v} = ch_{i_X}^{*}(X, v)$. De même nous définissons l'ensemble $E_{Y}^{cint} = \{a_{v,Y}\ tel\ que\ v \in S_{i_Y}^{int}\}$ avec $w_{v,Y} = ch_{i_Y}^{*}(v, Y)$.

Nous définissons alors $G_{eint} = (S_{eint}, A_{eint})$ comme suit :

- $S_{eint} = S_{cint} \cup \{X, Y\}$
- $E_{eint} = E_{cint} \cup E_{cint}^{X} \cup E_{Y}^{cint}$

Reprenons notre **exemple** de la [Figure III-8], comme précisé précédemment on recherche le plus court chemin entre le nœud « a » et le nœud « z ». Le graphe d'intersection étendu sera donc composé du graphe complet d'intersection complété par les deux nœuds « a » et « z ». On obtient le graphe suivant : [Figure III-9].

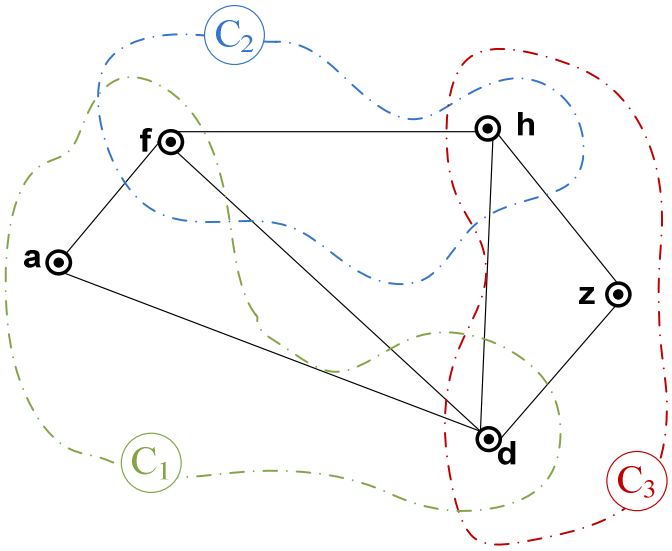

Figure III-9 - Exemple de graphe d'intersection étendu

Les valeurs des poids des arcs ajoutés par rapport au graphe complet d'intersection sont calculées comme suit :

- $\mathrm{I}(a,\mathrm{f}) = \{1\},\ \mathrm{I}(a,\mathrm{d}) = \{1\},\ \mathrm{I}(h,\mathrm{z}) = \{3\}$ et $\mathrm{I}(h,\mathrm{d}) = \{3\}$.
- $w_{a,f} = min_{i\in \mathrm{I(a,f)}} co\hat{u}t\left(ch_i^*(a,f)\right) = co\hat{u}t\left(ch_1^*(a,f)\right) = \mathrm{w}_{\mathrm{a,f}}^1 = 2$
- $w_{a,d} = min_{i\in \mathrm{I(a,d)}} co\hat{u}t\left(ch_i^*(a,d)\right) = co\hat{u}t\left(ch_1^*(a,d)\right) = \mathrm{w}_{\mathrm{a,f}}^1 + \mathrm{w}_{\mathrm{f,c}}^1 + \mathrm{w}_{\mathrm{c,d}}^1 = 2 + 5 + 15 = 22$
- $w_{h,z} = min_{i\in \mathrm{I}(h,z)} co\hat{u}t\left(ch_i^*(h,z)\right) = co\hat{u}t\left(ch_3^*(h,z)\right) = \mathrm{w}_{\mathrm{h,g}}^3 + \mathrm{w}_{\mathrm{g,e}}^3 + \mathrm{w}_{\mathrm{e,z}}^3 = 2 + 1 + 2 = 5$
- $w_{d,z} = min_{i\in \mathrm{I}(d,z)} co\hat{u}t\left(ch_i^*(d,z)\right) = co\hat{u}t\left(ch_3^*(d,z)\right) = \mathrm{w}_{\mathrm{d,g}}^3 + \mathrm{w}_{\mathrm{g,e}}^3 + \mathrm{w}_{\mathrm{e,z}}^3 = 2 + 1 + 2 = 5$
- Donc $rec(a,f) = 1, rec(a,d) = 1,\ rec(b,z) = 3$ et $rec(d,z) = 3$.

Suite à la construction du graphe d'intersection étendu, on peut utiliser l'algorithme de Dijkstra pour trouver le plus court chemin. On trouve alors le chemin $Ch^*(\mathrm{a,z}) = (a,f,h,z)$. Pour chaque paire de nœuds de ce chemin, on détermine la classe d'appartenance et le détail correspondant [Figure III-10].

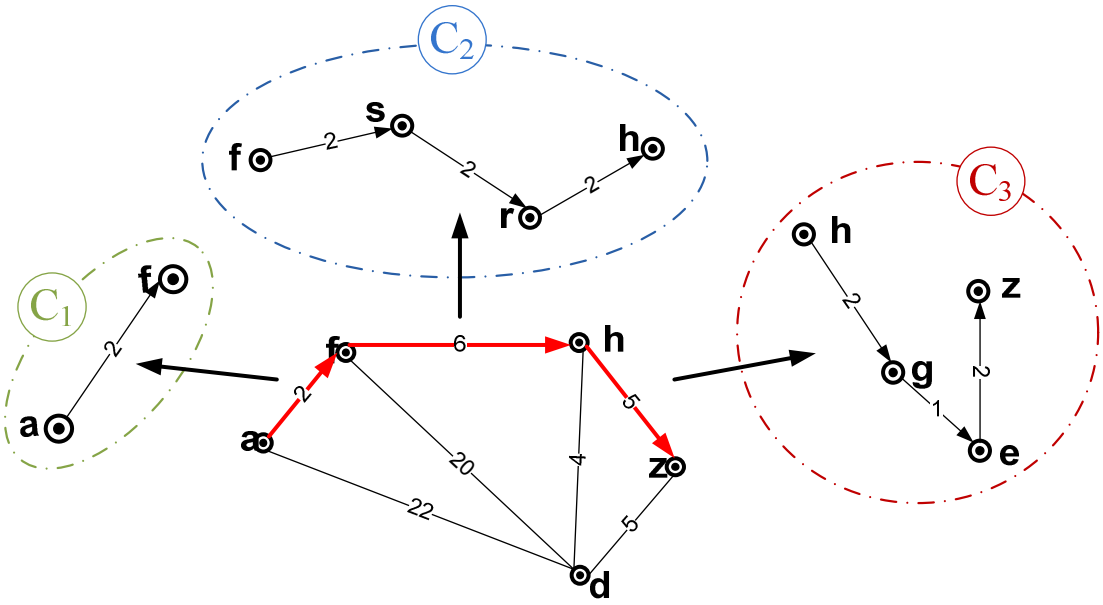

Figure III-10 - Plus court chemin dans le graphe distribué

III.5.3.3. Optimalité de l'algorithme

Dans leurs travaux, Wang et Kaempke démontrent que l'algorithme ***DSRA*** calcule le plus court chemin global sur le graphe virtuel $G = (S, A)$ en démontrant les deux lemmes suivants :

1. Un chemin de longueur fini du graphe G_{eint} correspond à un chemin dans le graphe global virtuel Gavec la même longueur.

2. S'il existe un plus court chemin Ch^*_G(X,Y) sur $G = (S, A)$ alors il existe un chemin $Ch^*_{G_{ecint}(X,Y)}$(X,Y) qui contiennent les mêmes nœuds d'intersection contenus dans Ch^*_G(X,Y). De plus : $co\hat{u}t\left(Ch^*_G(\mathrm{X,Y})\right) = co\hat{u}t\left(Ch^*_{G_{ecint}(X,Y)}(\mathrm{X,Y})\right)$

III.5.4. Problème des K plus courts chemins dans un graphe distribué

A notre connaissance, il n'y a pas encore eu de travaux pour la recherche des *K* plus courts chemins dans un graphe distribué. Les seules références dans ce domaine concernent des algorithmes distribués pour la recherche des *K* plus courts chemins, mais dans des graphes statiques. Parmi ces travaux on peut citer Ruppert[**97**] et Guerriero et al[**57**].

Il faut bien souligner la différence entre un algorithme de recherche du plus court chemin **distribué** dans un graphe **statique** et un algorithme de recherche du plus court chemin dans un graphe **distribué**. La différence réside dans le type du graphe traité.

La signification du mot **distribué** dans le premier type d'algorithme indique la manière dans laquelle ce dernier est exécuté : il s'agit d'une **exécution en parallèle** sur plusieurs processeurs avec une

synchronisation en fin de traitement dans un graphe statique. Dans le deuxième type d'algorithme, le mot **distribué** est associé au type du graphe, il s'agit d'une recherche d'itinéraire dans un **graphe distribué (cf. III.5.2)**.

III.6. Graphe distribué dynamique

Comme son nom l'indique, c'est une hybridation entre un graphe dynamique et un graphe distribué. Nous l'utilisons pour gérer des systèmes distribués (comme décrit dans le paragraphe III.5.1) c'est à dire avec un serveur central et des sous-systèmes mais en plus, nous manipulons des données horaires, donc des graphes à pondération dynamique. Comme décrit dans le paragraphe III.4, le coût des arcs dans ces graphes dépend du temps donc le coût d'un arc $a_{x,y}$ qui lie le nœud x au nœud y dépend de l'heure de départ du nœud x. Ces graphes distribués dynamiques sont généralement utilisés pour représenter des réseaux de transport gérés par différents opérateurs. En effet, on associe à chaque sous-système un opérateur de transport et un système d'information représente le serveur de calcul qui se charge des échanges entre les différents sous-systèmes pour générer une réponse globale.

Un graphe distribué dynamique est donc une représentation d'un système distribué où les sous-systèmes sont des opérateurs de transport qui gèrent des fiches horaires (représentant les heures de départ et les heures d'arrivée des lignes de transport). On associe alors à chaque sous-système i (et donc à chaque opérateur) une classe qu'on notera C_i. Une classe C_i est décrite par un graphe **dynamique** G_i avec $G_i = (S_i, A_i)$. S_i et A_i représentent respectivement l'ensemble des sommets et l'ensemble des arcs de la classe C_i. Le graphe global virtuel est noté $G = (S, A)\ avec\ S = \bigcup_{i=1}^{n} S_i$, $A = \bigcup_{i=1}^{n} A_i$ et n est le nombre de classe dans le système.

Un arc $a_{x,y}^{i} \in A_i$ ***si et seulement si*** $x, y \in S_i$ et le transport entre x et y est réalisé dans la classe C_i. On ajoute le i dans la notation $a_{x,y}^{i}$ pour différencier l'association d'un arc par rapport à une classe. En effet, l'arc $a_{x,y}$ peut être réalisé par la classe C_i ou une autre classe C_j [Figure III-7].

Etant donné que G_i est un graphe dynamique, le coût des arcs dans chaque sous-graphe est dépendant du temps. On associe alors à chaque arc un ensemble d'heures de départ qu'on notera $HD(a_{x,y}^{i})$. Et pour identifier le poids d'un arc $a_{x,y}^{i}$ on utilisera la notation $w_{x,y}^{i}(t)$ (qui est donc dépendante du temps) et indique le poids d'un arc liant x à y dans la classe C_i.

Le système qu'on propose (DPM) gère des systèmes d'information pour le transport. Chaque SI de transport gère un graphe dynamique. Une représentation globale du réseau est donc un graphe distribué dynamique. Pour améliorer la qualité de service proposé au voyageur, on propose de trouver le meilleur chemin en prenant en considération les différents graphes dynamiques des différents SI existants. Il s'agit donc de trouver le ou les plus court(s) chemin(s) dans un graphe distribué dynamique.

III.6.1. Problème du plus court chemin dans un graphe distribué dynamique

Dans le contexte d'un graphe dynamique, on parle d'un problème de chemin le plus rapide au lieu du chemin le plus court. En effet, avec la contrainte temporelle, notre objectif est d'atteindre le nœud destination le plus tôt possible.

Comme dans un graphe dynamique non-distribué, les données d'entrée sont un nœud de départ x, un nœud d'arrivée y et une heure d'arrivée au nœud de départ t_{dep} (aussi appelé heure de départ). Le problème est de rechercher le chemin qui nous permet d'atteindre le nœud y au plutôt si on part de x à t_{dep}. Les nœuds $x\ et\ y$ n'appartiennent pas nécessairement à la même classe, le chemin recherché $\mathrm{ch}(\mathrm{x},\mathrm{y})$ peut donc être un chemin inter-class (globale).

Mise à part les quelques études heuristiques que nous cité dans le cas des graphes distribués [**51**] [**65**] [**125**], les travaux de recherche dans ce type de graphe sont très limités.

Dans la littérature, on trouve deux approches assez similaires qui permettent de résoudre ce problème dans un temps polynomial et qui garantissent l'optimalité du résultat. La première est l'approche de Wang et Kaempke [**112**] qui est une adaptation de l'approche décrite dans le paragraphe (III.5) au cas dynamique. En effet, Wang et Kaempke ajoutent la contrainte temporelle dans leur algorithme et utilisent un algorithme de Dijkstra étendu qui permet de rechercher le meilleur chemin à l'intérieur des classes (dans les sous-systèmes). Autrement dit, ils proposent de faire un modèle espace-temps du graphe d'intersection qui va chercher toutes les informations relatives aux départs possibles d'un nœud d'intersection, ainsi qu'aux temps de trajet.

La deuxième approche est celle de Kamoun[**62**], elle s'inspire aussi de l'approche Wang et Kaempke pour résoudre le problème du plus court chemin distribué [**111**]. Cette dernière se distingue par une étape antérieure de sélection du domaine de recherche. Elle réduit ainsi le nombre de sous-systèmes mis en cause pour la recherche du chemin le plus rapide. De plus, cette approche évite la construction d'un Graphe d'intersection espace-temps qui intègre toutes les informations, elle propose à la place de construire un graphe à la volé liée à une requête et de récupérer les données nécessaires durant l'exécution de l'algorithme.

On définit alors pour une requête $Req(A, B, t_A)$ de recherche du chemin le plus rapide de A vers B avec une heure de départ de A à t_A un nouveau type de graphe : Graphe Virtuel Partiel $G_{vint(A,B)}$.

III.6.1.1. Graphe partiel virtuel

Le graphe $G_{vint(A,B)} = (S_{vint(A,B)}, A_{vint(A,B)})$ est définit pour une requête $Req(A, B, \boldsymbol{t_A})$. Il comprend l'ensemble des nœuds d'intersection du graphe global virtuel $G = (S, A)$, le nœud de départ A et le nœud d'arrivée B.

On note $I(x) = \{i / x \in C_i\}$ l'ensemble des index des classes auxquelles appartient un nœud x avec $x \in S_{vint(A,B)}$. Les arcs $A_{vint(A,B)}$ de $G_{vint(A,B)}$ lient chaque nœud x aux nœuds y accessibles dans une même classe.

Kamoun[**62**] démontre que tout arc du graphe $G_{vint(A,B)}$ vérifie la propriété FIFO par la suite le graphe $G_{vint(A,B)}$ est un graphe FIFO.

De ce fait, une fois le graphe $G_{vint(A,B)}$ créé, il suffit d'executer un algorithme du plus court chemin distribué FIFO. Par exemple l'algorithme (PCCD-FIFO) suivant :

On utilisera $\Omega_{a,b}(t_a)$ pour identifier l'heure d'arrivée au plus tôt au nœud b si on part du nœud a à $t = t_a$.

1. Initialisation

   $\Omega_X(t_0) = t_0$
   $\Omega_k(t_0) = \infty,\ pred_{t_0}(k) = \infty\ et\ r(i) = \infty,\ \forall k \in S - \{X\}$
   $pred_{t_0}(X) = X$ et $E_{courant} = \{X\}$

2. $\forall k \in N^+(v)\ faire$
   $si\ (\Omega_{v,k}(\Omega_{X,v}(t_0)) < \Omega_{X,k}(t_0))$ alors
   $\Omega_k(t_0) = \Omega_{v,k}(\Omega_{X,v}(t_0))$
   $pred_{t_0}(k) = v$
   r(k) = index de la classe
   Si $k \notin E_{courant}$ alors $E_{courant} = \{k\} \cup E_{courant}$

PCCD-FIFO

L'algorithme global qui demande la création du $G_{vint(A,B)}$ et fait appel au PCCD-FIFO est l'algorithme PCCDH (Plus Court Chemin Distribué et Horaire).

III.6.1.2. Algorithme et fonctionnement

1. *Construire le* $G_{vint(A,B)}$ *associé au G(N,E).*

   Déterminer $P^*_{Gvint(A,B)} = \{s_1 - \ldots - s_m\}$ *en exécutant* $PCCD - FIFO$.

2. *Pour* $k = 1$ *à* $k = m$-1

   Pour (s_k, s_{k+1}) *déterminer la classe* C_i *relative à* $r(s_{k+1})$

   Pour (s_k, s_{k+1}) *déterminer le détail de* $P_i^*\left(s_k, s_{k+1}, \Omega_{X,s_k}(t_X)\right)$ *en envoyant la requete* $req(s_k, s_{k+1}, \Omega_{X,s_k}(t_A))$ *à la classe* C_i

PCCDH

A partir d'une requête $Req(A, B, t_A)$ et du graphe global $G = (S, A)$, le PCCDH commence par construire le graphe d'intersection virtuel $G_{vint(A,B)}$. Il exécute ensuite l'algorithme PCCD-FIFO sur le graphe $G_{vint(A,B)}$. Ceci nous génère un chemin global optimal $Ch^*_{G_{vint(A,B)}}(A, B, t_A) = \{A = s_1, s_2.., s_m = B\}$ avec $s_k \in S_{vint(A,B)}$. Le détail des chemins $Ch_i^*(s_k, s_{k+1}, \alpha_{A_{s_k}}(t_A))$ sera déterminé à partir des différentes classes C_i en lançant une requête à la classe C_i.

Pour limiter le domaine de recherche, on s'intéresse à l'adjacence des différentes classes. Pour cela on créé un nouveau type de graphe appelé graphe d'adjacence $G_{adj} = (S_{adj}, A_{adj})$ où les nœuds de S_{adj} sont les différentes classes et les arcs de A_{adj} représentent les intersections entre les classes. Donc si deux sous graphes G_i et G_j associés respectivement aux classes C_i et C_j ont un lien qui mène d'un nœud dans G_i à un nœud dans G_j alors ceci se traduit par un lien entre les nœuds i et j du graphe G_{ajd}. Ensuite, un algorithme dit « d'inondation » sera exécuté sur le graphe d'adjacence pour trouver un ensemble de domaines de recherche. L'algorithme PCCDH sera par la suite exécuté sur chaque groupe de nœuds sélectionné.

III.6.2. Recherche des plus courts chemins dans un graphe distribué dynamique

Dans ce travail, nous nous intéressons à la recherche de plusieurs chemins pour fournir à l'utilisateur un plus grand confort dans les choix retournés.

Nous nous proposons d'étudier la problématique sous un autre angle. On suppose que l'utilisateur fournit un intervalle de temps de départ au lieu de fournir une heure de départ fixe. Par conséquent, un graphe virtuel global $G = (S, A)$, on cherche les plus courts chemins entre un nœud de départ X et un nœud d'arrivée Y en partant de X à une heure incluse dans l'intervalle de temps $T = [t_{inf}, t_{sup}]$. t_{inf} et t_{sup} sont respectivement la borne inférieur et la borne supérieure de l'intervalle.

Puisqu'on traite le temps comme discret et non pas continu, on peut écrire l'intervalle T comme suit $T = [t_{inf}, t_{sup}] = \{t_{inf} = t_1, t_2, \dots, t_p = t_{sup}\}$ avec $t_{k+1} - t_k = \tau$ et τ représente la plus petite unité de temps utilisée dans le système étudié. Dans ce qui suit τ sera égale à une minute vu que généralement on manipule des réseaux de transport urbain où les heures de départ et d'arrivée sont exprimées à une minute près.

III.6.2.1. L'algorithme Rep_n [45]

Une solution triviale au problème décrit précédemment consiste à réexécuter l'algorithme PCCDH autant de fois que d'éléments existants dans l'intervalle [**46**]. Autrement dit, si l'utilisateur fixe un intervalle de départ de p minutes $[t_1, .. t_p]$, on exécutera l'algorithme PCCDH p fois en prenant en entrée de l'algorithme le nœud de départ, le nœud d'arrivée et une heure de départ t_k avec $k \in [0, p]$ [**46**]. On obtient ainsi p chemins optimaux. Les p résultats ne sont pas strictement différents.

Nous identifions cette solution par Rep_n avec n le nombre de minutes qui sépare deux exécutions successives. Il faut remarquer que seul Rep_1 assure l'optimalité du résultat final. Les algorithmes Rep_k avec $k > 1$ proposent des solutions qui ne sont pas nécessairement optimaux vu qu'on traite un réseau non Fifo.

III.6.2.2. Complexité de l'algorithme Rep_1

La complexité de l'algorithme de calcul du plus court chemin dans un graphe dynamique distribué reste égale à la complexité de l'algorithme de calcul du plus court chemin dans un graphe statique distribué [**63**] [**64**] et qui est égale à $O(nlog(n))$ (complexité de l'algorithme de Dijkstra) [**91**].

Rep_1 étant une réexécution du PCCDH autant de fois que d'éléments existants dans l'intervalle $T = [t_{inf}, t_{sup}] = \{t_{inf} = t_1, t_2, \dots, t_p = t_{sup}\}$, il s'agit alors de réexécuter PCCDH p fois. Avec p le nombre d'éléments dans $[t_{inf}, t_{sup}]$ d'où la complexité de Rep_1 est égale à $O(p * n * log(n))$.

III.6.2.3. *Notre nouvelle approche qui parallélise la recherche des chemins*

Dans les systèmes distribués, l'échange de messages (envoi et réception de messages) entre les différents sous-systèmes et le serveur central prend plus de temps que l'exécution des instructions.

D'autre part, lors du calcul des chemins les plus rapides pour plusieurs heures de départ (cf le cas du Rep_1), le système central pourrait interroger les sous-systèmes pour une même partie d''itinéraire, mais pour des heures de départ différentes. Par exemple, le système central peut demander à un sous-système quels sont les itinéraires allant de X à Y en partant de X à t et t'. Demande que l'on peut traduire par deux requêtes distinctes $Req(X, Y, t)$, $Req(X, Y, t')$ qui devront être exécuter séparément. Partant de ce constat, nous proposons un algorithme qui regroupe, dans la limite du possible, plusieurs

requêtes en une seule, lors de l'interaction avec un sous-système. Nous proposons une solution pour trouver les chemins les plus rapides pour aller d'un nœud X à un nœud Y en fixant un intervalle de temps de départ, et ce , dans un graphe distribué. On traduit cette requête par la notation suivante $Req(X, Y, T)$ avec X nœud de départ, Y noud d'arrivé et $T = [t_{inf}, t_{sup}]$ intervalle d'heures de départ.

Notre approche est composée de 5 étapes principales [**44**]. La première sert à réduire la zone de recherche en identifiant les chaines minimales d'opérateurs pouvant résoudre le problème. Pour limiter la zone, on commence par construire le graphe d'adjacence relatif au graphe distribué. On utilise ensuite l'algorithme des K plus court chemins en considérant que le poids de chaque arc est égale à « 1 ».

On obtient ainsi des chaînes minimales de classes adjacentes qui permettent de lier le nœud de départ au nœud d'arrivée. L'ensemble de ces chaines minimales sera noté DR. Chaque chaine étant une liste de classe ordonnée on a donc $DR = \{L_1, L_2, \dots, L_n\}$ avec $L_i = (C_{k_1}, C_{k_2}, \dots, C_{i_p})$. La deuxième étape consiste à construire le graphe virtuel d'intersection étendu pour chaque liste de classe L_k dans le domaine DR défini dans la première étape. L'étape trois, sert à réduire le nombre d'éléments inclus dans l'intervalle d'heures de départ T en utilisant un algorithme *ASHDE*. La quatrième étape représente le cœur de l'approche, il s'agit d'exécuter l'algorithme TWDTA qui permet de trouver les chemins les plus courts dans chaque graphe étendu créé dans l'étape précédente. Dans l'étape cinq, chaque sous-système remplit le détail du tronçon du chemin global concerné.

Etape 1 : construire le graphe d'adjacence qui schématise la connexion des classes et exécuter l'algorithme des K plus court chemin pour déterminer $DR = \{L_1, L_2, \dots, L_n\}$ avec $L_i = (C_{k_1}, C_{k_2}, \dots, C_{i_p})$.

Etape 2 : construire le graphe virtuel d'intersection étendu relatif à chaque liste de classe L_k du domaine DR défini dans la première étape.

Etape 3: réduire le nombre d'éléments inclus dans la fenêtre de temps choisi pour les heures de départ en utilisant ASHDE

Etape 4 : pour chaque graphe virtuel d'intersection étendu de l'étape 3, on calcule les itinéraires les plus rapides à l'aide de TWDTA pour les heures de départ sélectionné à l'étape 2. Chaque chemin trouvé est constitué de nœuds d'intersection $(s_1, s_{2,} \dots, s_n)_k$

Etape 5 : Chaque sous-système remplit le détail du tronçon du chemin global concerné.

Algorithme des Chemins les Plus rapides dans un graphe Distribué avec un Intervalle d'Heure de Départ (ACPRD-IHD)

Pour bien expliquer l'utilisation de l'algorithme, nous illustrons dans ce qui suit les différentes étapes en se basant sur l'exemple du graphe distribué de la Figure III-6. Nous supposons qu'on recherche les chemins les plus rapides allant de « a » vers « z » avec des heures de départ inclus dans l'intervalle $T = [1,10]$.

Etape 1 : La première étape consiste à exécuter l'algorithme des K plus courts chemins dans le graphe d'adjacence.

Le graphe d'adjacence $G_{adj} = (S_{adj}, A_{adj})$ relatif au graphe distribué de la Figure III-6 est donné à gauche dans la Figure III-11. Il est composé de trois nœuds représentant les trois classes $C_1, C_2\ et\ C_3$ et trois arcs représentant les intersections entre ces classes.

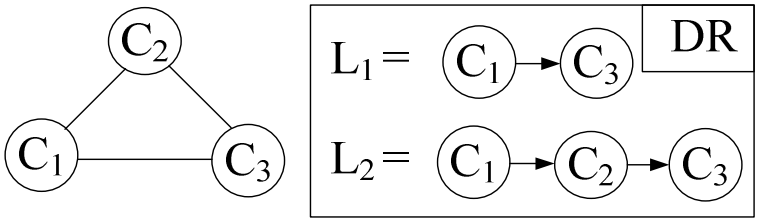

Figure III-11 - Exemple de graphe d'adjacence

En supposant que le poids de chaque arc est égal à « 1 » on obtient un graphe statique valorisé. On peut donc exécuter un algorithme de K plus courts chemins pour trouver les chaines minimales de classes liant « C_1 » à « C_3 » et par la suite les chaines de classes liant « a » à « z ».

L'exécution de l'algorithme des K plus courts chemins sans cycle nous permet d'obtenir le résultat suivant $DR = \{L_1, L_2\}$ avec $L_1 = (C_1, C_3)$ et $L_2 = (C_1, C_2, C_3)$ (voir Figure III-11)

Etape 2 : Pour chaque liste du domaine de recherche, on construit le graphe d'intersection étendu associé sans valorisation des arcs. Il s'agit d'identifier les nœuds d'intersection, les nœuds de départ et d'arrivée et les liens qui existent entre eux.

Par **exemple**, pour la liste L_2, nous construisons le graphe d'intersection complète étendu $(\mathrm{G_{eint}}(\mathrm{S_{eint}}, \mathrm{A_{eint}}))_{L_1}$ avec $\mathrm{S_{eint}} = \{\text{noeud de départ}\} \cup \{\text{noeud d'arrivée}\} \cup \{\text{noeuds d'intersection}\}$. Donc $\mathrm{S_{eint}} = \{\mathrm{a}\} \cup \{\mathrm{z}\} \cup \{\mathrm{f}, \mathrm{d}, \mathrm{h}\}$. On obtient ainsi le graphe de la Figure III-9.**Etape 3 :** On passe à la troisième étape qui sert à réduire les heures de départ possible dans l'intervalle T sélectionné. Pour cela on utilise l'algorithme (ASHDE).

III.6.2.4. Sélection des heures de départ effectives [ASHDE]

Comme décrit précédemment, à partir d'un nœud s du graphe, on a un nombre limité d'arcs sortants. Pour chaque arc, on a un nombre limité d'heures de départ. Dans un graphe distribué, un nœud peut

appartenir à plusieurs sous-systèmes (Classes) et donc à plusieurs sous-graphes (il s'agit donc d'un nœud d'intersection).

Dans ce qui suit nous utiliserons la notation suivante :

Un graphe $G_k = (S_k, A_k)$ est associé à la classe C_k qui représente le $k^{ième}$sous-sytème. Soit un nœud d'intersection s de p sous-graphes. Donc $s \in \bigcap_{k=1}^{p} G_k$. On note $A_k(s) = \{a_k^1(s), a_k^2(s), \dots, a_k^{n_k}(s)\}$ l'ensemble des arcs du graphe G_k dont le nœud source est le nœud s. L'ensemble $A_k(s)$ contient n_k éléments. A chaque arc $a_k^i(s)$ on associe un ensemble d'heure de départ (Cf. définition graphe dynamique) on notera cet ensemble $D(a_k^i(s))$. On note $HD(s)$ l'ensemble d'heures de départ possible à partir d'un nœud s toutes classes confondues donc on a $HD(s) = \bigcup_{k=1}^{p}\left(\bigcup_{i=1}^{n_k} D(a_k^i(s))\right)$.

L'ensemble des arcs sortants d'un nœud d'intersection est égal à l'union des arcs sortant de ce nœud vers les différentes classes auxquels il appartient. L'ensemble des heures de départs possibles à partir d'un nœud d'intersection est égal à l'union des heures de départ de chaque arc.

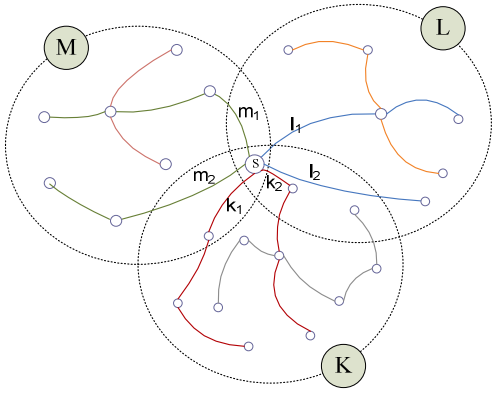

Figure III-12 - Intersection de plusieurs graphes

Prenons l'exemple du nœud s appartenant aux trois classes L, M, K. Dans chaque classe, s est l'origine d'un ensemble d'arcs. Ainsi l'ensemble global d'arcs de sortie du nœud s est l'union de $\{l_1, l_2\} \cup \{m_1, m_2\} \cup \{k_1, k_2\}$ et l'ensemble d'heures de départ possibles du nœud s est l'union des heures de départ possibles des différents arcs $l_1, l_2, m_1, m_2, k_1, k_2$. Donc l'ensemble des heures de départ possibles à partir d'un nœud s est $HD(s) = \{D(l_1), D(l_2), D(m_1), D(m_2), D(k_1), D(k_2)\}$.

Revenons maintenant à notre problématique, l'utilisateur fournit un nœud de départ, un nœud d'arrivée et une liste d'heure de départ $T = [t_{inf}, t_{sup}] = \{t_{inf} = t_1, t_2, \dots, t_p = t_{sup}\}$. Etant donné que l'on connaît la station de départ, il est possible de réduire l'intervalle d'heures de départ - en un ensemble plus restreint d'heures. En effet, il suffit de faire l'intersection de l'ensemble T avec l'ensemble $HD(s)$ pour obtenir l'ensemble d'heures de départs effectifs. On notera $HD_T(s)$ l'intersection entre l'ensemble T et l'ensemble $HD(s)$.

Nous proposons dans ce qui suit un algorithme que nous appelons ASHDE (Algorithme de Sélection d'Heures de Départs Effectifs) qui permet de déterminer pour un nœud s et un intervalle $T = [t_{inf}, t_{sup}]$, l'ensemble $HD_T(s)$ qui représente les heures de départ possibles du nœud s inclus dans l'intervalle T. Cet algorithme interroge les différents sous-systèmes associés aux sous-graphes le nœud s appartient.

1. Initialisation
   $HD_T(s) = \{\}$
2. Pour $k = 1$ à p faire
   Pour $j = 1$ à n_k faire
   Demander au sous-système k (C_k) l'ensemble $(D(a_k^j(s)) \cap T)$
   $HD_T(s) = HD_T(s) \cup (D(a_k^j(s)) \cap T)$

ASHDE

L'utilisation de cet algorithme va permettre une minimisation du nombre d'heures de départ utilisées pour le calcul des chemins les plus rapides (T). Par la suite une optimisation du temps de calcul de l'algorithme global.

En utilisant l'algorithme ASHDE, nous sommes sûre de ne pas ignorer une heure de départ possible. En effet, ASHDE prend en compte tous les arcs associés au nœud s.

Proposition : Pour une fenêtre de temps $T = [t_{inf}, t_{sup}]$ et un nœud s, le ASHDE détermine toutes les heures de départ possible incluses dans l'intervalle T et ayant s comme nœud de départ.

Preuve :

On suppose qu'il existe $t' \in [t_{inf}, t_{sup}] = T$ tel que $t' \notin HD_T(s)$. $HD_T(s)$ étant calculé avec ASHDE. On a $t' \in [t_{inf}, t_{sup}]$ donc t' est une heure de départ du nœud s ainsi $\exists r, k$ tel que $t' \in D(a_k^r(s))$ et $D(a_k^r(s)) \in \mathrm{HD(s)}$. Or $t' \in T$ donc $t' \in (D(a_k^r(s)) \cap T)$ nous en déduisons que $(D(a_k^r(s)) \cap T) \notin HD_T(s)$. Absurde vu la définition de $HD_T(s)$. ASHDE calcule donc toutes les heures de départ possibles inclus dans l'intervalle T et ayant s comme nœud de départ.

Reprenons l'exemple décrit plus haut, on recherche les chemins les plus rapides allant de « a » vers « z » avec des heures de départ inclus dans l'intervalle $T = [1,10]$. Les fiches horaires détaillant les heures de départ et d'arrivée d'un nœud à un autre dans le graphe d'intersection étendu sont fournies dans la Figure III-13.

$a \to f$		$f \to d$		$d \to f$		$d \to h$		$f \to h$	
Départ : t_a	Arrivée : $\Omega_{a,f}(t_a)$	t_a	$\Omega_{f,d}(t_a)$	t_a	$\Omega_{d,f}(t_a)$	t_a	$\Omega_{d,h}(t_a)$	t_a	$\Omega_{f,h}(t_a)$
3	9	10	17	10	17	11	19	11	15
7	12	13	27	15	26	13	20	14	17
		15	25	19	31	15	22	19	27
		19	28			18	28		
						26	32		

$h \to d$		$h \to z$		$d \to z$		$a \to d$	
t_a	$\Omega_{h,d}(t_a)$	t_a	$\Omega_{h,z}(t_a)$	t_a	$\Omega_{d,z}(t_a)$	t_a	$\Omega_{a,d}(t_a)$
16	21	16	23	11	26	4	8
18	23	30	36	22	27	39	44
38	49	34	40	24	28	52	59
				26	32		

Figure III-13 - Fiches horaires (TMT) relatives aux nœuds du graphe d'intersection étendu

Nous exécutons ASHDE sur le graphe d'intersection étendu construit préalablement et la fenêtre du temps $T = [1..10]$. Dans notre cas, $A(a) = A_1(a) = \{a_1^1(a), a_1^2(a)\} = ((a,d),(a,f))$. D'après la fiche horaire, on a $D\left(a_1^1(a)\right) = \{4,39,52\}$ et $D\left(a_1^2(a)\right) = \{3,7\}$ proposé tous les deux par la classe C_1. Par la suite, $HD_T(a) = (D(a_1^1(a)) \cap T) \cup (D(a_1^2(a)) \cap T)$ d'où $HD_T(a) = \{4\} \cup \{3,7\} = \{3,4,7\}$. Pour simplifier dans la suite de l'exemple, on notera T' l'ensemble $HD_T(a) = \{3,4,7\}$.

Etape 4 : Une fois les tris premières étapes terminées, on entame la quatrième étape qui consiste à exécuter l'algorithme TWDTA pour chaque liste de classe L_k du domaine DR défini dans la première étape.

L'algorithme TWDTA (Time-Window Departure Time Algorithme) étudie la corrélation des interactions entre le serveur central et les sous-systèmes et essaye de regrouper les requêtes tout en assurant une optimalitée du résultat final.

Notre point d'entrée pour cet algorithme est la $Req(A, B, T')$. Où T' est l'ensemble d'heures de départ sélectionnées dans l'étape 2 à partir de la fenêtre de temps $T = [t_{inf}, t_{sup}]$.

L'algorithme proposé ici construit les chemins globaux pour chaque heure de départ de T'. Il s'agit de construire les graphes partiels virtuels en parallèle minimisant ainsi le nombre de requêtes envoyées aux sous-systèmes.

On aura donc pour chaque heure de départ $t_k \in T'$ un graphe partiel virtuel $G_{vint(A,B)}^{t_k} = \left(S_{vint(A,B)}^{t_k}, A_{vint(A,B)}^{t_k}\right)$ qui permettra de trouver le chemin global allant de A vers B avec une heure de départ t_k.

Les chemins seront construits en partant du nœud source et en interrogeant les sous-systèmes. Les requêtes envoyées aux sous-systèmes seront regroupées permettant ainsi un traitement par lot.

Etant donnée que plusieurs chemins seront construits en parallèle, nous aurons à les identifier dans l'algorithme. Pour cela, on associera à chaque chemin une étiquette représentant la première heure de départ prise dans l'intervalle T'.

On reprend la notation décrite dans le premier paragraphe de ce chapitre : l'ensemble des successeurs de x sera donc noté $N^+(x) = \{v \in S/\ (x,s) \in A\}$.

On note :

1. $I(x) = \{i/\ x \in C_i\}$ l'ensemble des index des classes auxquels appartient un nœud x.
2. $\Omega_{a,b}(t_a)$ l'heure d'arrivée au plus tôt au nœud b si on part du nœud a à t_a.
3. $r_{t_a}(p)$ index de la classe utilisée pour trouver la valeur de $\Omega_p(t_a)$ si on part du nœud source à t_a.
4. $pred_{t_a}(\alpha)$ le nœud prédécesseur du nœud α si on part du nœud source à t_a.
5. $E_{courant}(t_a)$ l'ensemble de nœuds actifs pour un départ du nœud source à $t = t_a$.
6. $E^*_{courant}(t_a)$ est le nœud sélectionné de l'ensemble $E_{courant}(t_a)$ avec l'heure minimale d'arrivée pour un départ à t_a $\left(E^*_{courant}(t_a) = n\ tel\ que\ \Omega_n(t_a) = \min_{s \in E_{courant}(t_a)}(\Omega_s(t_a))\right)$.
7. $HDA[k]$ l'ensemble des Heures de Départ Actives défini pour un nœud k comme suit $HDA[k] = \{t_a\ \in T\ /\ E^*_{courant}(t_a) = k\}$.
8. Q désigne un ensemble temporaire qui contient les heures de départ utilisées dans chaque itération. Cette liste est vidée à chaque itération.

1. Initialisation

$Q = \{\}$

$Pour\ \forall i \in T\ faire$
$\Omega_X(i) = i$
$pred_i(X) = X$
$E_{courant}(i) = \{ X \}$

$Pour\ \forall \alpha \in S - \{X\}\ faire$
$Pour\ \forall i \in T\ faire$
$\Omega_\alpha(i) = \infty$
$pred_i(\alpha) = null$

2. $Pour\ \forall t_i \in T\ faire$

$HDA[E^*_{courant}(t_i)] = HDA[E^*_{courant}(t_i)] \cup \{ t_i \}$

$Sélection\ d'un\ noeud\ j\ de\ l'ensembl\ global\ des\ noeuds\ S\ tel\ que$
$card(HDA[j]) = max_{w \in S}\langle card(HDA[w])\rangle$

$Q = HDA[j]$

$HDA[j] = \{\}$

$Pour\ \forall t \in Q\ faire$

$E_{courant}(t) = E_{courant}(t) - \{j\}$

3. $Exploration\ des\ successeurs\ de\ \boldsymbol{j}$

$\forall p \in N^+(j)\ faire$

$\forall k \in Q\ faire$

$si\ (\Omega_{j,p}(\Omega_k(j)) < \Omega_k(p))\ alors$

$\Omega_k(p) = \Omega_{j,p}(\Omega_k(j))$

$pred_p(k) = j$

$r_k(p)$= index de la classe utilisée pour déterminer $\Omega_{j,p}(\Omega_k(j))$.

$Si\ (p \notin E_{courant}(k))\ alors\ E_{courant}(k) = E_{courant}(k) - \{p\}$

4. Si $\forall i \in T, E_{courant}(i) = \emptyset$ alors fin de l'algorithme sinon aller à « 2 ».

TWTDA[**44**]

Si on cherche les plus courts chemins entre un nœud de départ X et un nœud d'arrivée Y en partant de X à une heure incluse dans l'intervalle de temps $T = [t_{inf}, t_{sup}]$. On trace en retour arrière les chemins à partir de Y jusqu'à X en se basant sur les valeurs des étiquettes « $pred_p(k)$ ». Pour chaque heure de départ t inclus dans l'intervalle T, $\Omega_t(Y)$ représentera l'heure d'arrivée au plutôt à Y.

Reprenons l'exemple traité précédemment, dans la première étape on a pu identifier le domaine de recherche $DR = \{L_1, L_2\}$ avec $L_1 = (C_1, C_3)$ et $L_2 = (C_1, C_2, C_3)$.

Le calcul de T' dans la deuxième étape nous a fourni le résultat suivant $T' = \{3,4,7\}$. On aura donc à construire trois graphes partiels virtuels d'intersection en parallèle pour déterminer trois chemins optimaux.

Ainsi tout au long de l'exécution de l'algorithme, nous indexerons les variables par (3,4 ou 7) correspondant aux heures de départ initial. Ceci nous permettra d'identifier les chemins à tout instant de l'exécution de l'algorithme.

On commence par initialiser les étiquettes du nœud source a et tous les autres nœuds du graphe d'intersection étendu [Figure III-9].

t_i	3	4	7
$\Omega_a(t_i)$	3	4	7
$pred_{t_i}(a)$	a	a	a
$E_{courant}(a)$	$\{a\}$	$\{a\}$	$\{a\}$

$s \in S-\{a\}$ \ t_i	3	4	7
$\Omega_s(t_i)$	∞	∞	∞
$pred_{t_i}(s)$	$\emptyset$	$\emptyset$	$\emptyset$

On exécute ensuite **la première itération** : on calcule $E^*_{courant}$ pour toutes les heures de départ. Le calcul est simple vu que $E_{courant}(a) = \{a\}$ on aura $E^*_{courant} = \{a\}$ pour toutes les heures de départ : il s'agit du seul élément dans l'ensemble.

t_i	3	4	7
$E^*_{courant}$	a	a	a

On en déduit la nouvelle valeur de $HDA[E^*_{courant}(t_i)]$ pour tout $t_i \in T$.

t_i	3	4	7
$HDA[E^*_{courant}(t_i)]$	HDA[a]={3}	HDA[a]={3,4}	HDA[a]={3,4,7}

Suite à cette itération, on n'a renseigné que la valeur de HDA pour le nœud a, on a donc un ensemble vide pour les autres nœuds.

s_i	a	f	d	h	z
$HDA[s_i]$	{3,4,7}	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
card(HDA[s_i])	3	0	0	0	0

On sélectionne un nœud j de l'ensemble $S_{vint(a,z)} = \{a, f, d, h, z\}$ (ensemble des nœuds du graphe d'intersection étendu) tel que $card(HDA[j])$ soit maximale. Or le seul ensemble HDA valorisé est celui associé au nœud « a ». On affectera alors la valeur « a » à « j ». De plus, Q prendra le contenu de $HDA[a]$ et $HDA[a]$ sera vidé.

On désactive le nœud sélectionné (le nœud « a ») pour les heures de départ sélectionnées (se trouvant dans Q). Pour les désactiver, on les supprime des ensembles $E_{courant}(i)$ pour i ∈ {3,4,7}. On obtient alors :

j	a
Q	{3,4,7}

t_i	3	4	7
$E_{courant}(a)$	∅	∅	∅

Ensuite, on explore les nœuds successeurs du nœud « *a* ». On a $N^+(a) = \{d, f\}$ (voir Figure III-10). On commence par explorer le nœud « *d* ». Pour cela on interroge le sous-système associé à la classe la C_1 pour trouver les chemins le plus rapides liant « *a* » à « *d* » *avec une heure de* départ incluse dans *Q={3,4,7}*.

Pour aller de « *a* » à « *d* », la classe C_1 propose un départ à 4 pour une arrivée à 8 et un départ à 39 pour une arrivée à 44 (voir Figure III-13). Ainsi pour les heures de départ incluse dans *Q={3,4,7}* on obtient les réponses suivantes : $\Omega_{a,d}(3) = 8, \Omega_{a,d}(4) = 8\ et\ \Omega_{a,d}(7) = 44$. Les $\Omega_d(t_i)$ étant valorisées initialement à l'infini (∞), la condition « $\Omega_{a,d}(\Omega_i(a)) < \Omega_i(d)$ » est toujours réalisée. $\Omega_d(t_i)$ prend alors la valeur retournée par le sous-système : $\Omega_{a,d}(\Omega_i(a))$. Le prédécesseur de « d » étant « a », $pred_{t_i}(d)$ prendra la valeur « a ». On active par la suite le nœud « d » pour les heures de départ incluses dans Q en ajoutant « d » dans $E_{courant}(i)$ pour $i \in \{3,4,7\}$.

On explore de même le nœud « f » et on met à jour ses étiquettes en conséquence.

t_i	3	4	7
$\Omega_d(t_i)$	8	8	44
$pred_{t_i}(d)$	a	a	a
$E_{courant}(t_i)$	$\{d\}$	$\{d\}$	$\{d\}$
$r_{t_i}(d)$	1	1	1

t_i	3	4	7
$\Omega_f(t_i)$	9	12	12
$pred_{t_i}(f)$	a	a	a
$E_{courant}(t_i)$	$\{d, f\}$	$\{d, f\}$	$\{d, f\}$
$r_{t_i}(f)$	1	1	1

On passe à la **deuxième itération** : on calcule $E^*_{courant}$ pour toutes les heures de départ. En se basant sur les valeurs de $\Omega_d(t_i)$ et $\Omega_f(t_i)$ précédentes on trouve que l'heure minimale d'arrivée pour un départ à 3 est celle du nœud « d ». Pour un départ à 4 est aussi celle du nœud « d » mais pour un départ à 7 l'heure minimale est celle du nœud f. On calcule alors les valeurs de $HDA[E^*_{courant}(t_i)]$

t_i	3	4	7
$E^*_{courant}$	d	d	f
$HDA[E^*_{courant}(t_i)]$	HDA[d]={3}	HDA[d]={3,4}	HDA[f]={7}

On obtient donc :

s_i	a	f	d	h	z
$HDA[s_i]$	∅	{7}	{3,4}	∅	∅
card(HDA[s_i])	2	1	0	0	0

On sélectionne le nœud j en maximisant $card(HDA[j])$, le nœud sélectionné est donc le nœud « d ». On met à jour Q et $E_{courant}(i)$ pour i ∈ {3,4}.

j	d
Q	{3,4}

t_i	3	4
$E_{courant}(d)$	∅	∅

Ensuite, on explore les nœuds successeurs du nœud « d ». On a $N^+(d) = \{f, h, z\}$ (voir Figure III-10). On commence par explorer le nœud « f » en interrogeant le sous-système associé à la classe la C_1 ensuite on explore les nœuds « h » et « z » en interrogeant le sous-système relatif à C_2. Après mise à jour on obtient :

t_i	3	4	7
$\Omega_f(t_i)$	9	12	12
$pred_{t_i}(f)$	a	a	a
$r_{t_i}(f)$	1	1	1

t_i	3	4	7
$\Omega_h(t_i)$	19	19	∞
$pred_{t_i}(h)$	d	d	∅
$r_{t_i}(h)$	2	2	∅

t_i	3	4	7
$\Omega_z(t_i)$	26	26	∞
$pred_{t_i}(z)$	d	d	∅
$r_{t_i}(h)$	2	2	∅

On poursuit le déroulement de l'algorithme jusqu'à ce que $\forall i \in T'\ E_{courant}(i) = \emptyset$. On abouti alors aux valeurs des étiquettes décrites dans la Figure III-14.

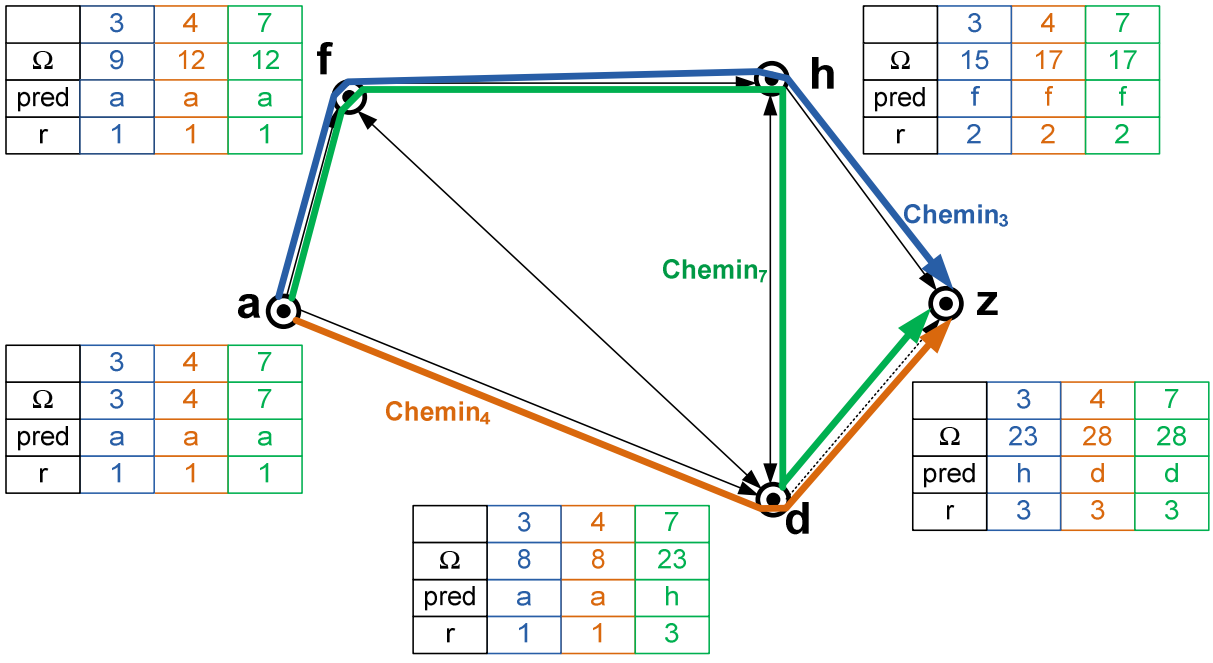

Figure III-14 - Résultat de l'exécution de TWDTA

Pour trouver les chemins le plus rapides liant « *a* » à « *z* » *avec une heure de* départ incluse dans Q, il suffit de tracer en retour arrière du nœud « *z* » jusqu'à « *a* » en se basant sur la valeur de « $pred$ ». On

obtient alors les trois chemins optimaux : $Chemin_3 = (a, f, h, z)$, $Chemin_4 = (a, d, z)$ et $Chemin_7 = (a, f, h, d, z)$.

Etape 5 : finalement, pour chaque chemin trouvé ($Chemin_3$, $Chemin_4$et $Chemin_7$) on consulte les sous-systèmes concernés pour remplir les arcs avec les informations locales pour obtenir un itinéraire complet. Par exemple pour le $Chemin_7 = (a, f, h, d, z) = ((a, f)_1, (f, h)_3, (h, d)_2, (d, z)_2)$ on récupère le détail de chaque arc en consultant la classe concernée (voir Figure III-15).

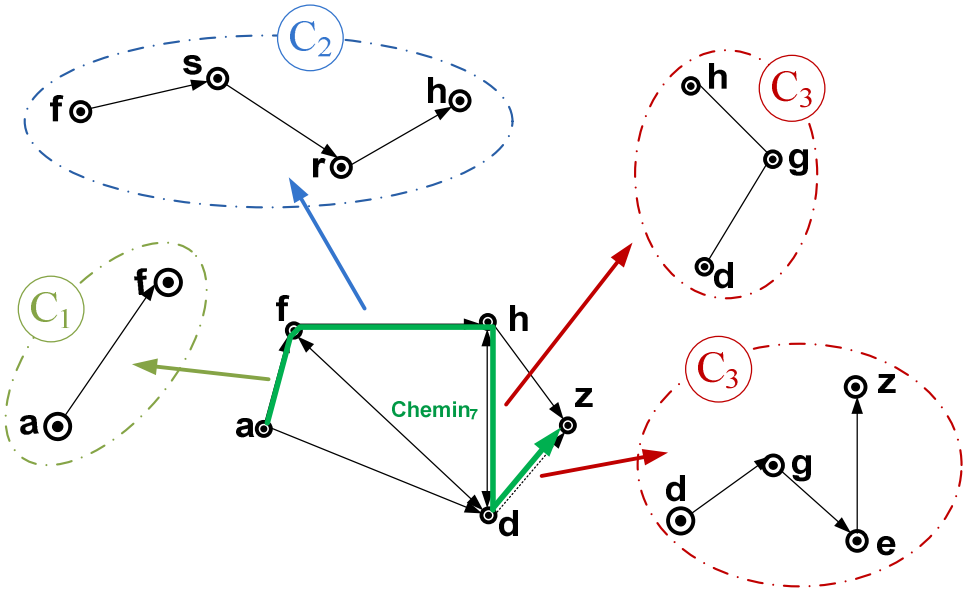

Figure III-15 - Détails du chemin globale pour l'heure de départ "7"

Théorème

L'algorithme TWTDA calcule les chemins les plus rapides dans G (V, E) pour toutes les heures de départ incluses dans l'intervalle de temps $T = [t_{inf}, t_{sup}]$.

Preuve

Comme dans la preuve de l'algorithme de Dijikstra décrit dans le paragraphe [III.1], l'optimalité du résultat est assurée par le maintien des étiquettes Ω_t et $E_{courant}$ pour tous les nœuds. Dans notre cas, nous maintenons les étiquettes Ω_t et $E_{courant}$ pour tous les nœuds et pour toutes les heures de départ possibles du nœud de départ.

En effet, le maintien de ces étiquettes garantit deux hypothèses :

- Maintenir Ω_t assure l'heure d'arrivée au plus tôt à chaque nœud et pour chaque heure de départ.

- Maintenir $E_{courant}$ assure le traçage de l'itinéraire final du nœud d'arrivée en remontant au nœud de départ. Ceci est réalisé pour chaque heure de départ.

Complexité

La complexité (dans le pire des cas) de l'algorithme ci-dessus est de $O\ (p\ *\ n\ log\ n)$. Elle est égale à la complexité de calcul de Rep1. p étant le nombre de départs effectifs du nœud source.

Dans le cas où les chemins sélectionnés sont identiques en terme d'enchainement de nœuds pour les différentes heures de départ, la complexité redescend à $O(n\ log\ n)$. Autrement dit, si pour tout t_i appartenant à T, l'itinéraire sélectionné sera $(s_0, s_1, .., s_p)$ avec $s_i \in S$ la complexité serait $O(n\ log\ n)$ puisqu'on ne fera pas p appels, mais seulement un seul appel.

Performance

Pour réaliser les tests de performance, nous avons utilisé des connexions en réseau local et une machine de 2 GHz 4 Go de mémoire.

La figure suivante montre une comparaison des traitements des différents algorithmes (Rep_1, $ASHDE + Rep_1$ et $ASHDE$ +TWTDA) détaillés précédemment. Les résultats de ces algorithmes sont basés sur des graphes connexes générés aléatoirement.

Le gain en temps de traitement est visible et il est d'autant plus important que le réseau est grand et avec un grand nombre de nœuds.

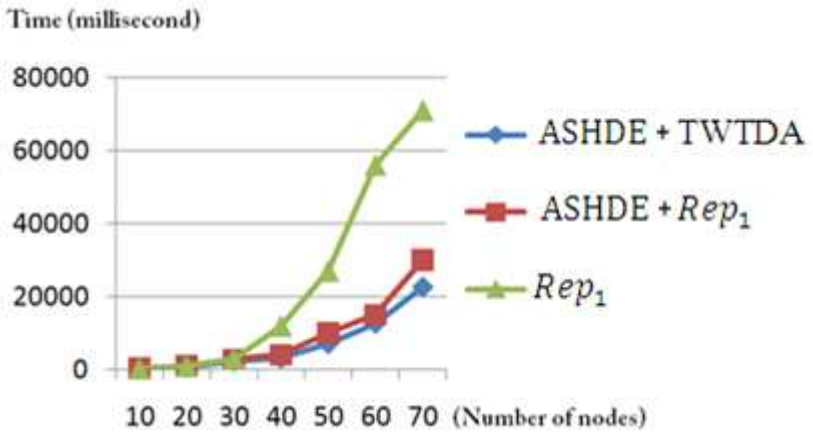

Figure III-16 - Comparaison des performances des algorithmes

III.7. Conclusion

Dans ce chapitre nous avons présenté différents algorithmes traitant plusieurs types de graphes. Ensuite, nous nous sommes focalisés sur un type en particulier, il s'agit des graphes distribués dynamiques qui représentent au mieux notre problématique. Dans ce contexte, nous avons proposé des algorithmes de recherche de/des chemin(s) le(s) plus rapides dans un environnement distribué. Ces algorithmes sont précédés par des étapes de réduction du domaine géographique et temporel permettant une optimisation du temps de recherche.

Ces algorithmes sont utilisés au niveau SMA de notre système. En effet, l'étape de limitation de la zone géographique est à la charge de l'agent jouant le rôle de « limitation de zone de recherche » et les étapes de limitation de la zone temporelle et de composition de « l'itinéraire globale » est à la charge de l'agent jouant le rôle « Composeur itinéraire ».

Nous détaillons dans le chapitre suivant le développement du système et nous l'illustrons avec des scénarios de simulation dans le cas d'exécution « normal » et « perturbé ».

IV. Implémentation, déploiement et Simulation

IV.1. Implémentation

IV.1.1. Introduction

Dans le présent chapitre, nous détaillons les différents aspects techniques du système d'information et nous exposons, module par module, les choix de langage de programmation. Nous proposons deux possibilités d'implémentations en identifiant leurs avantages et inconvénients. Nous utilisons dans certains cas le formalisme UML pour exposer des schémas afin de faciliter la compréhension statique et dynamique du système.

IV.1.2. Le niveau présentation

L'environnement utilisé pour l'interface homme/machine, est un environnement J2EE il permet d'accéder aux services offerts par le système à travers des pages JSP et des servlets écrites en langage JAVA. Dans ce qui suit, nous présenterons sommairement l'environnement J2EE. Ensuite nous exposons l'implémentation de notre système dans cet environnement.

IV.1.2.1. Environnement J2EE

L'environnement J2EE est une collection de standards permettant de développer et de déployer des applications web pour différents serveurs. Nous pouvons citer parmi ces standards :

- Entreprise Java Beans (EJB)
- Java Server Pages (JSP)
- Java Servlets
- Java Messaging Services
- Java Transaction Services
- Java Naming and directory Interface

IV.1.2.1.1. Les Java Servlets

Les servlets sont des applications Java qui tournent sur un serveur et qui permettent d'étendre les fonctionnalités de ce dernier. Les servlets peuvent traiter dynamiquement des requêtes arrivant des navigateurs web et génèrent les réponses adéquates. Les servlets tournent donc sur un serveur en attente d'une requête client. Dès qu'une requête arrive, la servlet crée un nouveau thread pour le client, dans lequel elle génère du HTML à envoyer au navigateur.

IV.1.2.1.2. Java Server Page

Le JSP est une technique de programmation qui permet la génération dynamique de pages Html. Ces pages contiennent des balises spéciales renfermant du code Java. Elles sont exécutées sur le serveur qui transforme le code java en code HTML compréhensible par les navigateurs des clients.

IV.1.2.2. Les couches du niveau présentation

Comme précisé dans le deuxième chapitre, le niveau présentation est formé de trois couches : interface, traitement et connexion [Figure II-14]. En implémentation, la couche interface est formée par les pages jsp, la couche traitement et la couche connexion sont représentées par des classes métier distinctes [Figure IV-1].

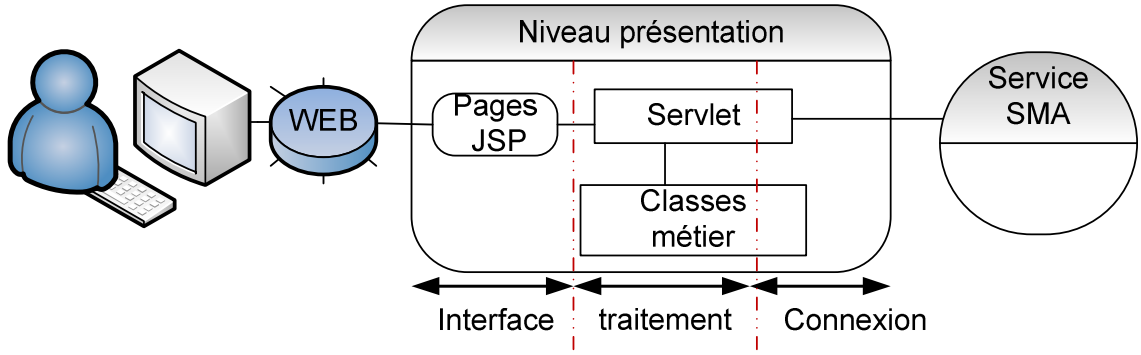

Figure IV-1 – Implémentation des trois couches du niveau présentation

IV.1.2.2.1. Les pages JSP du DPM

Ce sont les pages web constituant l'interface directe avec l'utilisateur elles constituent ainsi la **couche interface**. Ces pages proposent les services offerts par le système. Elles sont envoyées par le serveur d'application et gérées par la servlet. Les pages les plus importantes dans notre système sont les suivantes :

Page Accueil : C'est la première page qui sera affichée au démarrage du système. Elle permet d'accéder aux autres pages. Elle représente le menu principal de l'application.

Page Itinéraire optimisé : En accédant à cette page, le système nous présente les différents opérateurs et stations inscrits dans le système. C'est la page à travers laquelle on peut lancer la requête de recherche.

Page Résultat : c'est la page qui affiche le résultat de recherche sous forme de tableau.

Page Perturbation : cette page sera utilisée pour afficher les perturbations détectées durant les sept derniers jours. Les perturbations détectées au cours du déplacement de l'utilisateur lui seront directement transmises par mail ou par SMS.

IV.1.2.2.2. La 'Servlet DPM'

C'est le cœur de l'entité présentation, elle joue le rôle d'un organisateur entre les différentes parties de l'application Web. En effet, les informations recueillies par les pages JSP seront directement acheminées vers cette servlet qui décidera de leur traitement. Deux classes de traitement sont mises à la disposition de cette servlet :

Les classes métier : contenant les différents traitements à exécuter avant tout affichage des pages JSP sous forme de méthodes.

Les classes de connexion : leur rôle est la communication avec les autres niveaux.

IV.1.2.2.3. Les classes de traitement

Ce sont les classes qui contiennent tous les traitements à effectuer pour afficher les différentes pages JSP. Une fois la page et le traitement correspondants sont déterminés, la 'servlet' utilise les méthodes des classes de traitement pour exécuter les instructions nécessaires, récupère les résultats et les renvoie à la page JSP correspondante.

IV.1.3.Le niveau service

Une des parties importantes de notre projet est la communication avec l'environnement extérieur (extra) et la communication entre les différents niveaux (inter). Cette communication représente le niveau service de notre application. Ce niveau à été développé en utilisant des services web (Voir Chapitre II). Un web service, est un programme informatique qui tourne sur un serveur, qui calcule les réponses à des requêtes reçues par le serveur, et renvoie des réponses à ces requêtes assurant ainsi l'échange de données entre systèmes différents. Par exemple, on pourrait imaginer un service web "Taux de change", qui aurait une fonction "kilometre Vers Miles" prenant en entrée une distance en kilomètre, la convertirait et rendrait en distance en Miles. C'est la partie "serveur" du service web. N'importe quelle personne connaissant la fonction kilometre Vers Miles et son utilisation peut coder une partie "client", sur un autre ordinateur connecté au réseau, qui enverrait une requête à la partie serveur : il suffit pour cela de connaître l'adresse du serveur, le nom de la fonction et son utilisation, ainsi que l'adresse d'un fichier WSDL[1] (Norme des descripteurs d'un service Web).

Ce fichier WSDL (Web Services Description Language) est un fichier, respectant les normes XML[2] (XML est un langage à base de balises qui permet de hiérarchiser des informations), qui décrivent toutes les modalités de l'échange entre le client et le serveur : description des fonctions disponibles, du type de données qui transitent, etc. Ce fichier est situé sur le serveur qui contient la partie "serveur" du service web. Lorsqu'un client fait une requête vers la partie serveur, on dit qu'on "consomme" le service web. Ainsi, on peut voir le service web (partie client) comme une boîte noire : on lui envoie

[1] http://www.w3.org/TR/wsdl
[2] http://www.w3.org/TR/REC-xml/#sec-white-space

quelque chose en entrée, et on reçoit quelque chose en sortie. Le tout est totalement opaque : il n'y a aucun moyen de savoir ce qu'il y a dans la boîte, quel langage est utilisé, etc. La partie "client" du service web est en général assez courte : quelques lignes de code servent à appeler le service web, et le reste consiste principalement à trier les données reçues dans la requête, et les afficher correctement à l'utilisateur.

L'intérêt du service web est que ce système facilite grandement les échanges entre diverses applications, sans se soucier d'une possible incompatibilité entre plusieurs langages. En effet, en théorie, il n'existe presque aucune contrainte quant aux langages utilisés, que ce soit dans la partie client ou serveur. Il faut juste que ces langages rendent possible l'utilisation de services web. On peut ainsi, par exemple, avoir un service web (partie serveur) codé en Java, qui communique avec plusieurs clients, un en PHP, l'autre en Java, etc.

De plus, les services web, pour communiquer, utilisent le protocole HTTP, qui est un protocole très répandu, et utilisé notamment pour la navigation sur le net. Cela évite que la communication soit bloquée pour les utilisateurs qui, par exemple, se situent derrière un serveur proxy bloquant les protocoles peu répandus. Dans ce cas, il n'existe pas de risque : à partir du moment où l'on peut aller visiter des pages web, on peut aussi utiliser des services web.

Enfin, il existe plusieurs méthodes pour faire communiquer des services web, les plus connus sont :

REST *(Representational state transfer)* **:** REST[**48**] n'est ni un format ni un protocole, c'est un style d'architecture ou une philosophie de l'utilisation du Web de base. Il utilise les standards et protocoles du web à savoir http et URI et évite de faire appel aux couches d'abstraction proposées par SOAP et XML-RPC.

XML-RPC[1] *(XML- Remote Procedure Call)* **:** Considéré comme l'ancêtre du SOAP, il partage avec ce dernier plusieurs aspects d'architecture comme l'utilisation d'XML. Il permet d'invoquer une fonction sur un serveur distant indépendamment du langage utilisé et du type du système sur lequel on est.

SOAP[2] (*Simple Object Access Protocol) : c*'est un protocole soutenu par W3C et défini à l'origine par Microsoft et IBM. Soap est devenu maintenant une référence dans la programmation des services web. Ce protocole est composé principalement par deux parties :

L'enveloppe : une description du message lui-même pour permettre sa manipulation entre systèmes.

Un modèle de donnée : une description du format du message.

[1] http://www.xmlrpc.com/
[2] http://www.w3.org/TR/soap/

Actuellement, SOAP est le protocole le plus répandu. Toujours dans une optique d'interopérabilité, nous le choisissons pour notre projet.

Nous présentons dans ce qui suit un exemple de service web assurant la communication entre le niveau « Présentation » et le niveau « SMA ».

Supposons qu'un utilisateur saisit une requête pour rechercher le meilleur itinéraire entre les stations « 4_Contons » et « Wasquehal_Pavé_de_Lille » pour une heure de départ à 8H30. Le niveau présentation structure cette requête et appelle à travers un web service l'une des fonctions proposées par le système. Dans cet exemple, la fonction appelée est la fonction requete(Départ, Arrivée, Horaire, Nombre de sauts). Départ et Arrivée sont les noms de deux stations de métro ou bus ; horaire est l'horaire de départ ; nombre de sauts est le nombre de changements autorisés, par défaut initialisé à la valeur trois.

Consommation du service web

Pour utiliser la fonction proposée au niveau SMA du DPM, on utilise le code présenté la figure suivante. On stipule l'adresse du fichier WSDL en première ligne dans la variable correspondante ($wsdl) ; on remplit ensuite un tableau d'options contenant notamment la localisation du service web en seconde ligne. La troisième ligne sert à créer un client SOAP. Une fois le client créé, nous faisons appel au serveur à travers la fonction requete.

```
$wsdl= 'http://172.31.60.100:8080/wsig/ws/DPM_SERVICES.wsdl';
 $options=array('location' => "http://172.31.60.100:8080/wsig/ws",'uri' => "DPM_SERVICES", 'connection_timeout'=>100) ;
 $serv_customers = new SoapClient(null, $options);
$customers_arr = $serv_customers->requete('M1-4_Contons','M2-Wasquehal_Pavé_de_Lille','08:30','3');
```

Figure IV-2 - Utilisation du web service

SOAP génère une chaine XML formée à partir des paramètres fournis. Par exemple, la [Figure IV-3] représente la requête SOAP associé à l'appel donné plus haut.

```
<?xml version="1.0" encoding="UTF-8" ?>
<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:urn="urn:DPM_SERVICES">
            <soapenv:Header/>
            <soapenv:Body>
              <urn:requete soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
               <departureStation xsi:type="xsd:float">M1-4_Contons</departureStation>
               <arrivalStation xsi:type="xsd:float">M2-Wasquehal_Pavé_de_Lille</arrivalStation>
                        <departureTime xsi:type="xsd:float">08:30</departureTime>
                        <sauts xsi:type="xsd:float">3</sauts>
              </urn:requete>
            </soapenv:Body>
          </soapenv:Envelope>
```

Figure IV-3 - Requête générée

Dans le corps de la requête, nous y retrouvons la station de départ, 4 Cantons (ici, M1-4_Contons est la dénomination utilisée par le service web du DPM, le M1 signifiant que la station appartient à la ligne 1 du métro lillois. De même nous y retrouvons les balises <arrivalStation>, <departureTime>, et <sauts>, qui contiennent le reste des paramètres passés dans la fonction de départ. Cette chaine est ainsi envoyée via le protocole HTML au serveur en utilisant les coordonnées disponibles dans le fichier WSDL précisé dans l'appel de la fonction. Le serveur renvoie, après calcul, la réponse par le même canal d'informations.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Body><requeteResponse xmlns="urn:DPM_SERVICES">
                                        <heureArrivStationDepart>
                                                <station>M1-Wazemmes</station>
                                                <heure>08:30</heure>
                                        </heureArrivStationDepart>
                                        <troncon>
                                                <depart>
                                                        <station>M1-Wazemmes</station>
                                                        <heure>08:32</heure>
                                                </depart>
                                                <arrivee>
                                                        <station>Porte_des_Postes</station>
                                                        <heure>08:33</heure>
                                                </arrivee>
                                        </troncon>
```

Figure IV-4 - Réponse Soap

La [Figure IV-4] représente la réponse retournée. Elle est générée aux mêmes normes que la requête avec des informations encapsulées dans des balises. Il est nécessaire de traiter la réponse du serveur pour la mettre sous une forme compréhensible par l'utilisateur (c'est la tache réalisée entre autres par

le niveau présentation). Pour cela une opération de « parsing » est nécessaire elle consiste à parcourir le flux XML et d'en extraire les informations.

IV.1.4.Le niveau SMA

La composition d'itinéraires multi-opérateurs sous entend la disponibilité des données de chaque opérateur ce qui n'est pas toujours le cas. L'approche que nous proposons vise principalement à surmonter cette contrainte en intégrant non pas 'les données de chaque opérateur' mais 'un accès vers son système d'information'. Une telle stratégie nous permet de récupérer les données nécessaires non pas à partir de la base de l'opérateur, mais à partir de son SIAD. Une telle intégration nous permet de profiter des calculateurs de chaque opérateur pour des déplacements locaux. De plus cette stratégie met la responsabilité de mise à jour des horaires et l'indication des perturbations sur les opérateurs eux-mêmes. En effet, en mettant à jour en temps réel 'les horaires', 'les coupures' et 'les retards', notre système répond directement avec la meilleure offre disponible.

De ce fait, le problème de recherche d'itinéraires entre deux stations appartenant à des opérateurs différents est ramené à un problème de recherche d'un ensemble d'opérateurs séparant les opérateurs de départ et d'arrivée. En effet, si nous pouvions déterminer l'ensemble des opérateurs impliqués dans le déplacement ainsi que les pôles d'échange (i.e. nœuds d'intersection) entre ces différents opérateurs, la composition d'itinéraire multi-opérateur se verra traduite par une concaténation des itinéraires calculés par les processeurs locaux des opérateurs de cet ensemble.

Par ailleurs, l'itinéraire fourni par chaque SIAD local peut ne pas être unique. Ainsi, en faisant varier ces itinéraires locaux, nous obtenons des chemins différents avec des durées différentes, mais passant par un même ensemble d'opérateurs selon un même ordre. Nous utilisons donc les algorithmes décrits dans le Chapitre III pour trouver le plus court chemin dans cet ensemble.

Ces algorithmes sont mis en place sous forme d'une communication entre agents. Ceux-ci sont plus adaptés à ce type de problème : en effet, en comparant la notion d'agent à la notion d'objet nous remarquons rapidement que les agents englobent les spécificités des objets et les complètent par plusieurs autres caractéristiques comme le niveau de communication, la richesse des interactions entre les agents (protocole, typologie et ontologie) et la prise autonome de décision. Or, la mise en place d'un tel système nécessite un environnement particulier de développement et des outils bien précis. En recherchant dans les outils disponibles, nous pouvons trouver une variété de plateformes qui permettent la mise en place et le suivi du processus de développement. La plupart de ces outils sont développés avec Java. Nous donnons dans ce qui suit une brève présentation des outils les plus cités et utilisés.

IV.1.4.1. Plateformes Multi-agent

IV.1.4.1.1. Zeus

ZEUS[**84**] est un outil développé par British Telecom Intelligent System Research Lab. Il permet le développement des systèmes collaboratifs selon les quatre phases : analyse, conception réalisation et support à l'exécution. Elle commence par la description des rôles, ensuite l'organisation et enfin la coordination. L'inconvénient majeur de cet outil est sa complexité et la difficulté avant de pouvoir le maîtriser.

IV.1.4.1.2. Jade

Avant de décrire la plateforme Jade, il serait judicieux de définir la norme FIPA[1] qui est un projet de normalisation dans le domaine multi-agent. En effet, vu la diversité des travaux effectués dans la recherche multi-agents, le problème de la normalisation est devenu urgent. Plusieurs projets se sont intéressés à ce sujet dans le but unique : faciliter l'interopérabilité entre agents et entre concepteurs. Les projets les plus connus sont Agent UML[2] (AUML) et FIPA. Nous nous intéressons dans la suite à la norme FIPA (Foundation for Intelligent Physical Agents). Fondée en 1996, la FIPA offre des moyens standardisés pour une meilleure communication entre agents tout en respectant leur sens initial.

JADE[3] (Java Agent Development Framework) [**10**][**9**] a été développé à l'Université de Parme en Italie. C'est une plateforme JAVA de développement des systèmes multi-agents répondant aux normes FIPA. Jade utilise pour la communication entre agents le langage FIPA ACL. De plus, cette plateforme possède trois modules nécessaires à la conformité pour la norme FIPA.

– Le DF "Director Facilitor" est un module qui fournit un service de pages jaunes à la plateforme.

– Le ACC "Agent Communication Channel" gère les différentes communications entre les agents.

– Le AMS "Agent Management System" supervise l'enregistrement des agents, leur authentification, leur accès et l'utilisation du système.

IV.1.4.1.3. Madkit

Madkit[4] (Multi-Agent Developpment Kit) créé en 1996 par Olivier GUTKNECHT et Michel FERBER (Laboratoire d'Informatique, de Robotique et de Micro-électronique de Montpellier). MadKit est une plateforme multi-agent entièrement écrite en Java. Elle se base sur un modèle organisationnel (Agent/Groupe/Rôle)

[1] http://www.fipa.org/
[2] http://www.auml.org/
[3] http://jade.tilab.com/
[4] http://www.madkit.org

La communication dans Madkit est basée sur un mécanisme de peer to peer, et permet le développement rapide des applications distribuées en utilisant les principes multi-agent. Parmi les outils que Madkit propose, on trouve une interface graphique qui permet de supporter déférentes modes d'utilisation et d'exploitation de la plateforme.

IV.1.4.2. Choix d'une plateforme

Notre objectif premier est d'assurer une bonne portabilité et une interopérabilité optimale. En effet, au sein de notre laboratoire (LAGIS) nous travaillons dans le but de développer une plateforme générique qui regroupe plusieurs services autour du transport.

La portabilité pourrait être assurée en choisissant une plateforme développée en langage Java. En effet, le langage Java est portable ce qui facilite la portabilité de la plateforme. Or la plupart des plateformes multi-agent sont à base de java, la portabilité est donc assurée dans la majorité des cas.

Nous nous intéressons alors à l'interopérabilité. Pour cela, le choix le plus judicieux doit se conformiser à une norme de développement multi-agent. En choisissant la plateforme Jade, nous nous conformons aux normes FIPA. D'où l'orientation vers cette plateforme. De plus, plusieurs travaux dans notre équipe ont adopté cette technologie, nous l'avons donc choisi pour mieux s'intégrer dans l'existant.

IV.1.4.3. Fonctionnement de la plateforme Jade et déclaration des agents

IV.1.4.3.1. Plateforme et conteneur

Chaque instance de JADE s'exécutant simultanément sur un ou plusieurs postes est appelée conteneur (container). Un ensemble de conteneurs actifs est appelé plateforme. Pour chaque plateforme il existe toujours un et un seul conteneur principal appelé main container. Tous les autres conteneurs doivent se déclarer dans le « main container ». Le « main container » (voir Figure IV-4) déclare à son démarrage deux agents qui reprennent deux des exigences de la norme FIPA. Ces deux agents sont :

- L'agent AMS (Agent Management System) qui offre le service de nommage. C'est qui attribue un nom différent à chaque Agent dans la plateforme.
- L'agent DF (Directory Facilator) offre le service des pages jaunes. Chaque agent peut consulter l'agent DF pour récupérer la liste des services offerts par les agents actifs de la plateforme.

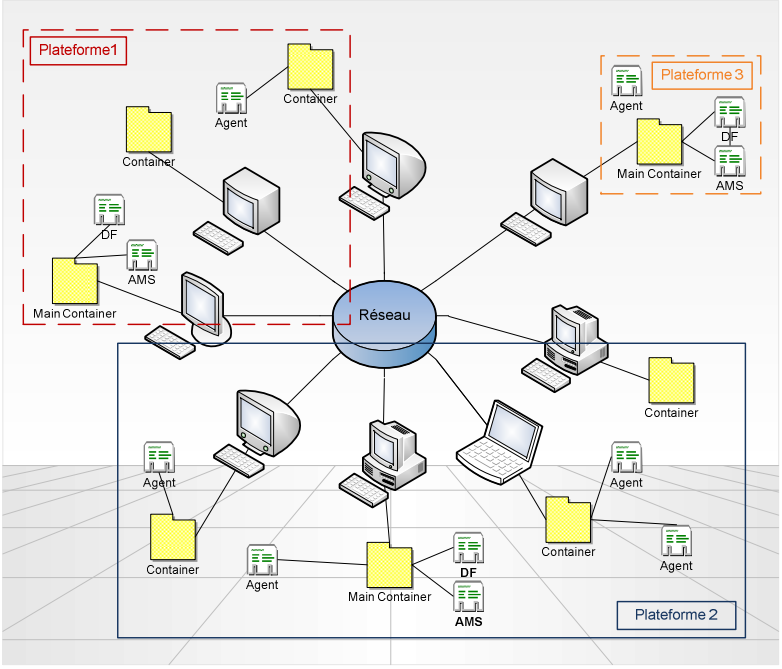

Figure IV-5 - Plateforme et conteneur JADE

IV.1.4.3.2. Structure et déclaration des agents dans Jade

Un agent dans la plateforme Jade est une classe qui hérite de « jade.core.Agent ». Cette classe doit nécessairement contenir une méthode spéciale « *setup( )* » qui constitue la partie initialisation de l'agent. Le fragment de code java suivant est un exemple de déclaration d'un agent sur plateforme jade.

```
import jade.core.Agent ;
public class monAgent extends Agent {
protected void setup()
  {
  // initialization de l'agent
   System.out.println( "MonAgent est initialisé"+getAID().getName()+" is ready.") ;
  }
 }
```

Figure IV-6 - Déclaration d'un agent dans JADE

La méthode « *getAID( )* » est une méthode pré-implémentée dans jade.core. Elle permet d'obtenir l'identifiant de l'agent qui est une instance de la classe jade.core.AID. Cet identifiant contient le nom unique qui référence l'agent.

Ce sont les « Behaviour » (comportements) qui permettent aux agents d'agir dans une plateforme. Un comportement est une tâche qu'un agent doit réaliser à un moment précis et selon des contraintes. Les

«Behaviour» sont implémentés comme des instances d'une classe qui hérite de «jade.core.behaviours.Behaviour».

Chaque «Behaviour» contient au moins deux méthodes :

- La méthode « *action ( )* » : qui définit les actions à réaliser quand ce «Behaviour» est invoqué.
- La méthode « *done( )* » : retourne un booléen qui indique si le «Behaviour» a terminé son exécution ou non.

Il existe deux principaux types de « Behaviours »

IV.1.4.3.2.1. One-shot behaviours [Figure IV-7]

Les actions de ce « Behaviour » s'exécutent une seule fois. Ce « Behaviour » se termine immédiatement après l'appel de sa méthode « *action( )* ». La méthode « *done( )* » est déjà implémentée dans le jade.core.behaviours.OneShotBehaviour. Le fragment de code suivant est un exemple d'implémentation d'un « One-shot Behaviour »

```
import jade.core.behaviours.OneShotBehaviour
public class MyOneShotBehaviour extends OneShotBehaviour
        {
          public void action()
          {
            // Instructions à réaliser qu'une seul fois pour ce behaviour
          }
        }
```

Figure IV-7 - Un exemple de code d'implémentation du « OneShotBehaviour »

IV.1.4.3.2.2. Cyclic behaviours [Figure IV-8]

Ce « Behaviour » ne se termine jamais, il continue à s'exécuter en boucle jusqu'à ce que l'agent meurt. Ces « comportements » sont généralement utilisés de la manière suivante : un « CyclicBehaviour » qui joue le rôle d'un serveur. Il boucle en vérifiant les messages reçus, vérifie leurs formes et leurs destinataires. S'ils répondent aux critères, cela signifie qu'il est bien compétent à les gérer. Il les traite donc et renvoie une réponse à leur expéditeur. De l'autre côté, un « OneShotBehaviour » s'exécute sous certaines conditions en envoyant un message vers un « CyclicBehaviour » en attente.

Il faut aussi signaler que si un agent n'a pas de comportement à exécuter, le thread qui gère l'agent en question s'endort afin de ne pas consommer du temps du processeur. Il est réveillé dès qu'il y a un « Behaviour » dans sa liste d'attente.

```
import jade.core.behaviours.CyclicBehaviour
public class MyCyclicBehaviour extends CyclicBehaviour
 {
  public void action()
  {
    //Tant que l'agent est vivant, les instructions qui se trouvent ici seront exécutées
   }
}
```

Figure IV-8 - Un exemple de code d'implémentation du « CyclicBehaviour »

IV.1.5. Deux possibilités d'implémentation

Comme décrit précédemment (Chapitre II), l'architecture de notre système se décompose en trois niveaux (présentation, service, SMA). Une première possibilité d'implémentation consiste donc à séparer totalement les trois niveaux. Dans ce cas, la communication entre le niveau présentation et SMA passe nécessairement par le niveau Service (voir [Figure II-14]).

Une deuxième possibilité consiste à fusionner une partie du « niveau service » avec le « niveau présentation » Figure IV-9. Nous nous référons à cette typologie par 'implémentation en Fusion'. La partie fusionnée fait partie du niveau service qui sert à la communication entre les deux autres niveaux. Nous avons donc toujours trois niveaux. Seulement la communication entre le niveau présentation et SMA ne passera plus par le niveau service. Cette disposition peut être mise en place à condition que les niveaux présentation et SMA soient déployés sur un même serveur ou sur des serveurs dans un même réseau local. Nous expliquons dans la suite plus en détail cette disposition.

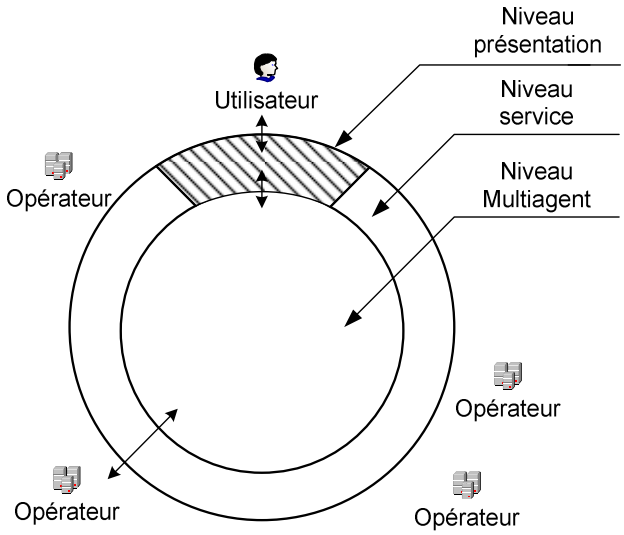

Figure IV-9 - Fusion du niveau Présentation et une partie du niveau Service

IV.1.5.1. Implémentation en trois niveaux

Les utilisateurs communiquent avec les pages JSP à travers un explorateur Web. Après avoir sélectionné les paramètres de recherche dans la page d'optimisation, une requête est générée. Elle est envoyée à la servlet. La servlet utilise les classes métier de connexion pour faire le bon appel du web service. L'appel du service web enclenche un mécanisme au niveau SMA, les différents agents exécutent alors un des algorithmes (décrit dans le chapitre III) pour cela, ils consultent des opérateurs de transport à travers des web services.

IV.1.5.2. Implémentation en Fusion

L'utilisateur communique avec le système à travers des pages ***JSP*** et une servlet constituant le niveau présentation du système. Les informations sont transférées au niveau SMA à travers une classe spéciale nommée GateWayAgent qui constitue une sorte de passerelle entre la servlet d'un côté et les agents d'un autre. GateWayAgent utilise un nouvel objet appelé « BlackBoard » pour les échanges des messages entre la servlet et les agents [Figure IV-10].

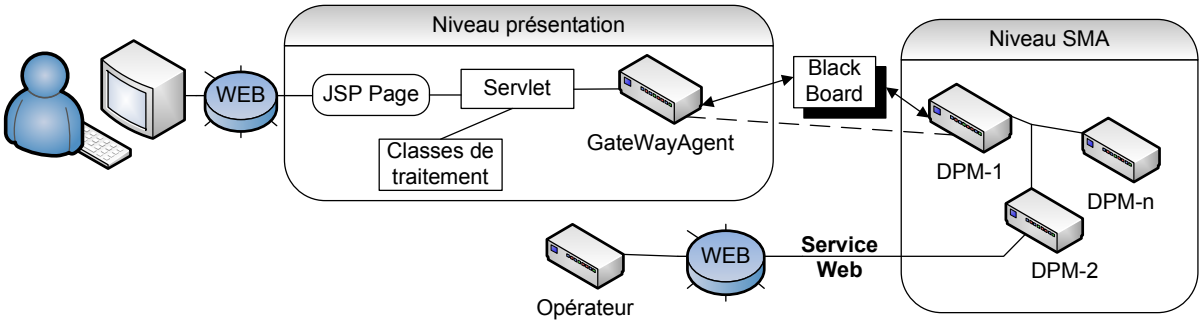

Figure IV-10 - Implémentation en Fusion

IV.2. Déploiement

IV.2.1. Introduction

La répartition des agents impacte directement le coût des communications. Le temps qui sépare l'envoi et la réception d'un message dépend avant tout de la typologie du réseau. Il est donc nécessaire de trouver une distribution qui minimise le temps consacré à la transmission des messages entre les agents.

Notre système d'information, ainsi défini, hérite des caractéristiques et autres avantages des différents agents qui le composent. La capacité qu'ont les agents de communiquer, indépendamment de la nature et de la topologie du réseau, confère à notre système une grande diversité vis-à-vis des configurations de déploiement possibles

Dans le présent chapitre, nous présentons quelques scénarios de déploiement, afin de trouver une configuration adéquate, que nous testons par la suite.

IV.2.2. **Architectures centralisées**

Dans cette configuration [Figure IV-11], tous les agents sont contenus dans un même serveur central. Les requêtes d'itinéraires sont redirigées vers ce serveur. Une telle architecture a pour avantage de réduire considérablement le coût engendré par les communications. Les messages ainsi échangés entre agents appartenant à une même plateforme, ne sont pas acheminés sur un réseau, ce qui constitue un gain considérable. De même, cette approche tend à diminuer le risque de perte de messages. Le gain en communication peut paraître considérable, mais cependant, il reste assez limité en terme de performance. En effet, initialement conçus pour fonctionner de façon parallèle, les différents agents sont exécutés sur un même processeur, ce qui réduit considérablement les performances du système.

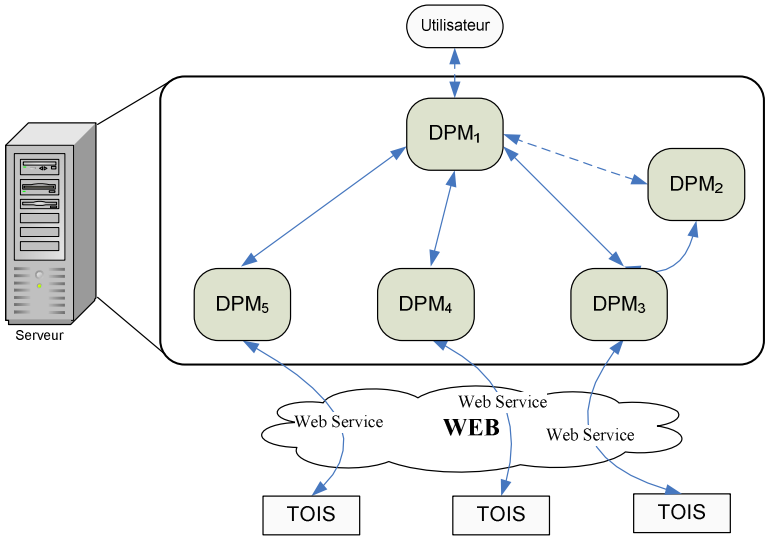

Figure IV-11 - Déploiement centralisé

Par ailleurs, une telle architecture impose une mise en place d'une file d'attente pour des requêtes provenant de plusieurs utilisateurs et d'un algorithme de gestion approprié. Une telle solution montre rapidement ses limites si nous venons à considérer un nombre important de requêtes. En effet, le traitement de chaque requête implique une monopolisation de la plateforme jusqu'à la fin de son exécution.

Cette disposition reste tout de même intéressante pour tester le système à faible cout et sans monopoliser plusieurs machines.

IV.2.3. Architectures distribuées

L'objectif principal d'une organisation distribuée est le gain de temps par une parallélisation des calculs d'itinéraire. Il est donc important de ne pas gaspiller ce gain de temps dans la communication entre les différents agents. En déployant notre système sur un réseau local, nous réduisons le coût des transferts de messages. De plus, cette distribution nous garantit une parallélisation des traitements. D'autre part, les ordinateurs et serveurs sont de plus en plus équipés de multi-processeur (Dual-Core et Quadri-Core). Plusieurs travaux de recherche ont montré l'efficacité de ce type de processeurs [**7**]. Nous essayons de les exploiter dans le déploiement de notre système. Les agents du SMA sont intelligents et peuvent détecter la présence d'un processeur « Dual-Core » ou « Quadri-Core » sur une machine. Une fois détectée, les agents s'organisent de manière à ce que chaque agent exploite un seul processeur. Les échanges de messages entre les agents tournant sur une même machine et exploitant des processeurs différents sont très rapides et leurs traitements sont totalement parallèles.

Prenons par exemple la disposition de la [Figure IV-12], cinq agents composent le système ($DPM_1, DPM_2 .. DPM_5$). Les agents $DPM_1 .. DPM_3$ s'exécutent sur trois machines différentes appartenant au même réseau local. Les deux agents DPM_4 et DPM_5 s'exécutent sur une seule machine, mais profitent du processeur Dual-Core de cette machine. En effet, l'agent DPM_4 utilise le premier processeur du serveur et DPM_5 utilise le deuxième processeur.

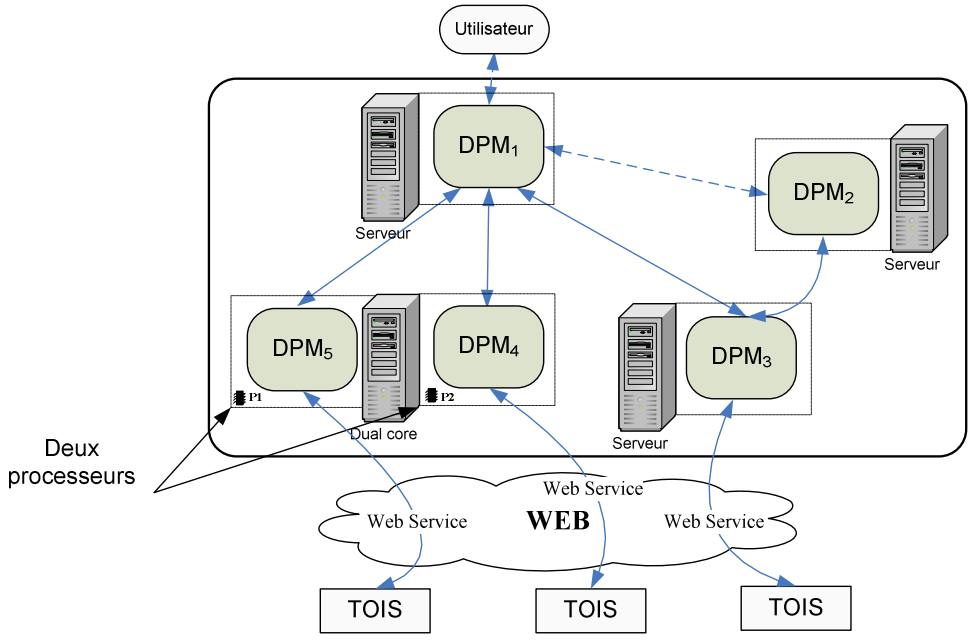

Figure IV-12 - Déploiement parallèle

IV.2.4. Conclusion

Dans ce paragraphe, nous avons présenté différentes stratégies de déploiement de notre système. Dans le paragraphe suivant, nous proposons un exemple général d'exécution.

IV.3. Une vue « services aux voyageurs » de notre système

Nous nous sommes proposé d'intégrer notre système de recherche d'itinéraire dans une application d'offres de services aux voyageurs basé sur l'utilisation de PDA (*Personal Digital Assistant).* On aboutit ainsi à un système global qui permet de guider l'utilisateur dans une ville inconnue. Il commence par géolocaliser l'utilisateur et afficher sa position dans une carte GoogleMaps (voir Figure IV-13). Ensuite il lui propose plusieurs services. Ces services peuvent aller de la recherche de commerces de proximité jusqu'à la recherche d'itinéraire.

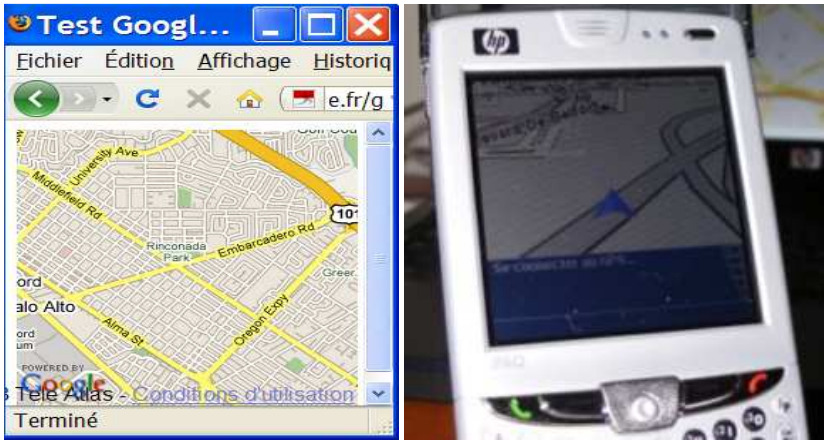

Figure IV-13 - Géolocalisation sur PDA exploitée par notre système

Notre objectif est d'offrir une alternative aux systèmes de guidage déjà existants tout en complétant largement la gamme de services. Il s'agit d'une interface ergonomique et intuitive proposant des services innovants.

Nous nous proposons alors de fournir à l'utilisateur tout ce dont il aura besoin pour se mouvoir et se sentir à l'aise dans une ville qui lui est pourtant inconnue. Il s'agit d'un outil permettant la plus grande mobilité à l'usager. Accompagner le touriste ou l'homme d'affaires dans ses moindres déplacements, en garantissant une assistance complète de l'utilisateur avec un encombrement minimal. En effet, ce système s'utilise dans un PDA (Personal Digital Assistant) l'outil adapté à ce besoin dans la mesure où il concilie portabilité, facilité d'emploi et mobilité.

Nous avons donné la priorité à l'implémentation des services suivants :

1. Accès aux commerces de proximité, et en particulier aux boulangeries. La concentration sur le seul cas des boulangeries représentera un cas d'étude intéressant. Service sur les commerces de proximité incluant les horaires d'ouverture en temps normal et en période de congés, pouvant s'adapter en temps réel aux périodes d'ouverture des commerces concernés.
2. Informations culturelles : informations pratiques sur les horaires des séances et les films diffusés, les expositions, permanentes ou non, les opéras et pièces de théâtre qui se jouent.

3. Service de recherche d'itinéraire, en privilégiant les transports en commun et la marche à pied, qui constituent en général les seuls moyens de transport locaux à disposition des personnes en déplacement, et qui présentent l'indéniable avantage d'être écologiques.

La [Figure IV-14] représente spatialement le chemin parcouru par les requêtes et les informations dans le cadre d'une communication avec le système.

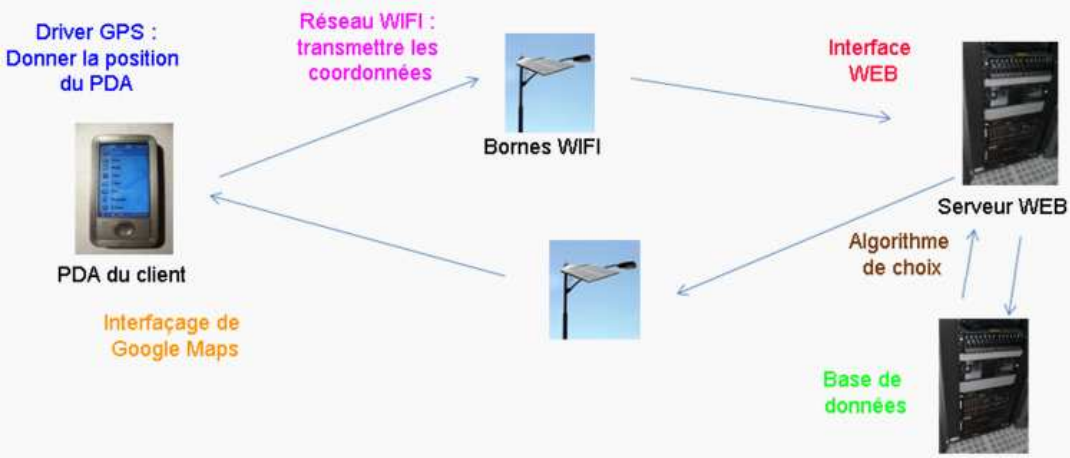

Figure IV-14 - Diagramme spatial du système

L'utilisateur interagit avec l'interface (PDA) pour s'authentifier et sélectionner le service recherché avec les bons paramètres. Les coordonnées GPS (Latitude, Longitude) de l'utilisateur seront récupérées et transmises avec la requête de l'utilisateur au serveur Web via le WIFI. Le serveur traite les informations reçues en essayant de comprendre ce que veut le client et utilise un script d'accès à la base de données pour en déduire l'ensemble des résultats possibles. Il détermine par la suite l'ensemble des éléments les plus proches de l'utilisateur. La réponse est transmise par la suite via le WIFI au PDA qui se charge d'afficher les résultats de la réponse dans l'interface sur carte Google Maps présentant la position de l'utilisateur et celle des services demandés.

Le service le plus important est le service de recherche d'itinéraire. Nous détaillerons donc dans ce qui suit l'intégration du système DPM décrit précédemment dans cet environnement.

L'utilisateur saisit, sur l'interface du PDA, sa destination et les heures de départs qui lui conviennent. Sa station de départ est directement identifiée par le calcul des coordonnées GPS. Toutes ces informations sont transmises via un service web au système DPM. Ce dernier se charge alors de calculer les itinéraires optimaux entre la position actuelle de l'utilisateur et la destination. Ensuite, il renvoie la réponse via le niveau service.

Comme on peut le voir sur la [Figure IV-15], seuls le niveau « service » et le niveau « SMA » sont exploités, le niveau présentation est dans ce contexte représenté par l'interface de communication du PDA qui permet d'accéder à plusieurs services, y compris celui de recherche d'itinéraire. La flexibilité

de notre architecture (trois niveaux indépendants) facilite le changement de disposition et l'ajout d'autres services. En effet, il suffit de se connecter directement au niveau « service » pour avoir accès aux différentes fonctionnalités de recherche d'itinéraire.

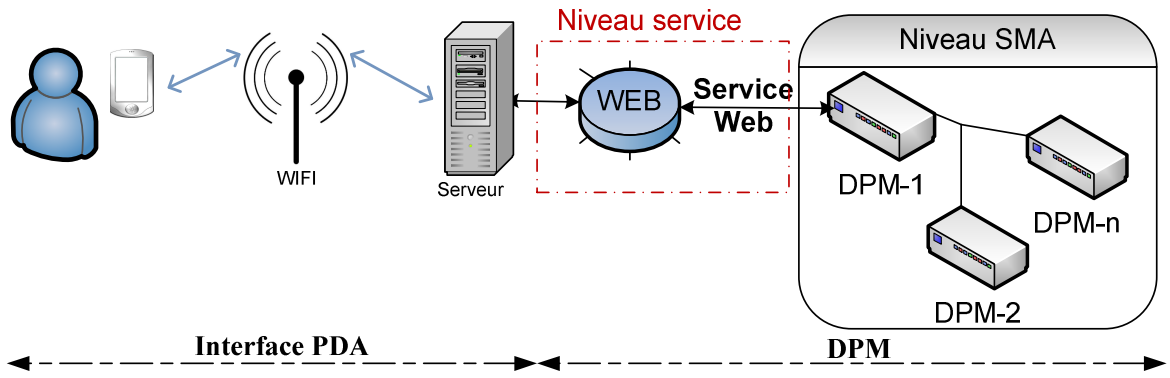

Figure IV-15 – Transit de l'information du PDA aux Agents

Cette expérience a donné une autre dimension à notre système, il nous a permis de concrétiser l'utilisation de la géolocalisation permettant ainsi d'exploiter réellement les fonctionnalités du système.

IV.4. Simulation

IV.4.1. Introduction

Dans ce paragraphe, nous présentons un exemple d'exécution, en détaillant l'aspect graphique du système. Nous commencerons par présenter les données utilisées pour la validation du système ; ensuite, nous expliquerons les étapes d'utilisation.

IV.4.2. Données utilisées

Pour valider le fonctionnement du système et les algorithmes de recherche d'itinéraire, nous avons utilisé, dans un premier temps, des données générées aléatoirement, mais qui restent proches de la réalité. Ceci nous a permis de faire des tests de performance et de charge en simulant des réseaux de transport d'une grande envergure avec des centaines de lignes et des milliers de nœuds.

Mais pour nos tests fonctionnels et pour la validation des résultats, nous avons utilisé des données réelles récupérées à partir de l'opérateur SNCF et de l'opérateur Transpôle.

Nous présentons dans ce qui suit un exemple tiré de quatre lignes. Il s'agit de deux lignes de métro (métro 1 et métro 2) et de deux lignes de bus (Bus 43 et Bus 44) de la région Lilloise. De plus, nous considérons un site internet de covoiturage qui propose deux itinéraires en départ de « G. Flandres » à 9h20 et 11h10 et à destination de « 4 Canton ».

Pour une meilleure représentation de l'aspect multi-opérateur du problème, nous considérons que chaque ligne de ce réseau représente un opérateur exploitant une seule ligne de transport. Donc, dans notre exemple, Metro1 est un opérateur de transport qui exploite la seule ligne de métro Métro1 de même pour Métro2, Bus43 et Bus44.

Le réseau constitué par ces cinq opérateurs est présenté dans la [Figure IV-16].

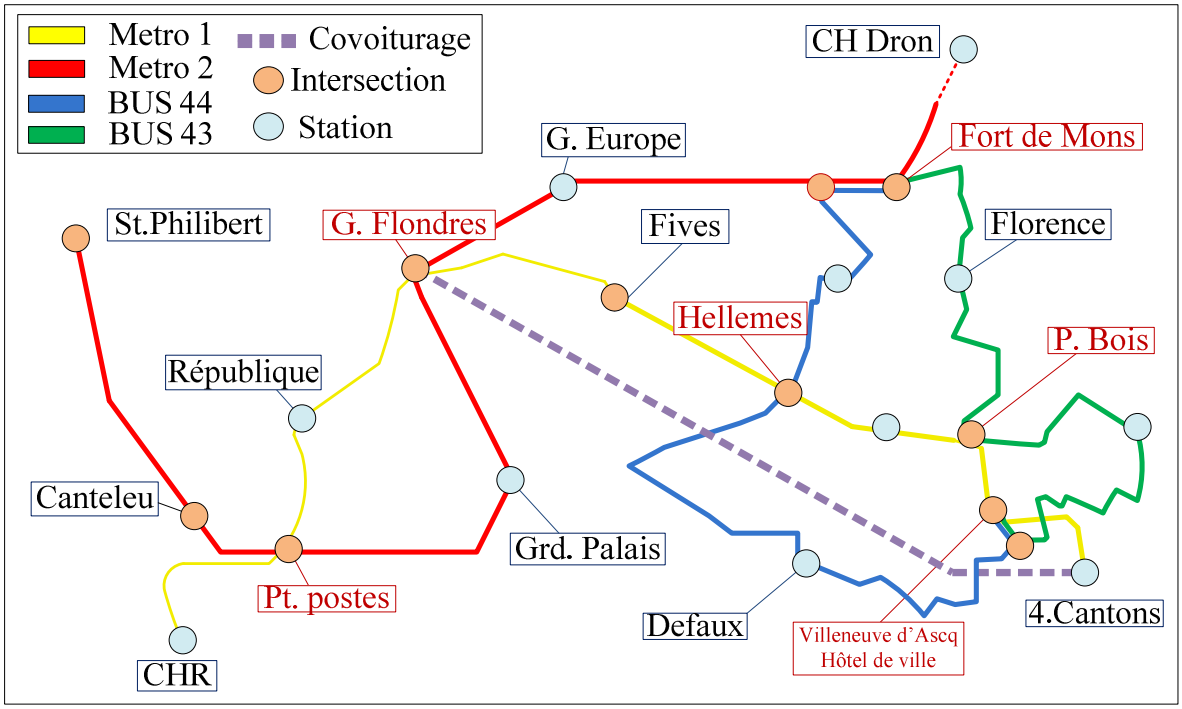

Figure IV-16 - Le réseau formé par les cinq opérateurs et leurs intersections

IV.4.3. Présentation et aspect graphique

Pour les aperçus présentés dans ce chapitre, nous avons utilisé le serveur d'application « Apatch tomcat » et l'explorateur internet *Firefox*. La première page proposée par le système est la page d'accueil présentée dans la [Figure IV-17].

Figure IV-17 - Page d'accueil

Cette page permet l'accès aux différents services offerts par le système. Nous pouvons distinguer les fonctionnalités suivantes :

- **Itinéraire heure départ** : permet d'accéder à la page de recherche d'itinéraire avec une heure de départ unique. En effet, on utilise dans ce cas l'algorithme PCCDH-FIFO décrit dans le chapitre précédent. On accède alors à la page de recherche [Figure IV-18] où l'on peut sélectionner une station de départ, une station d'arrivée et l'heure de départ.

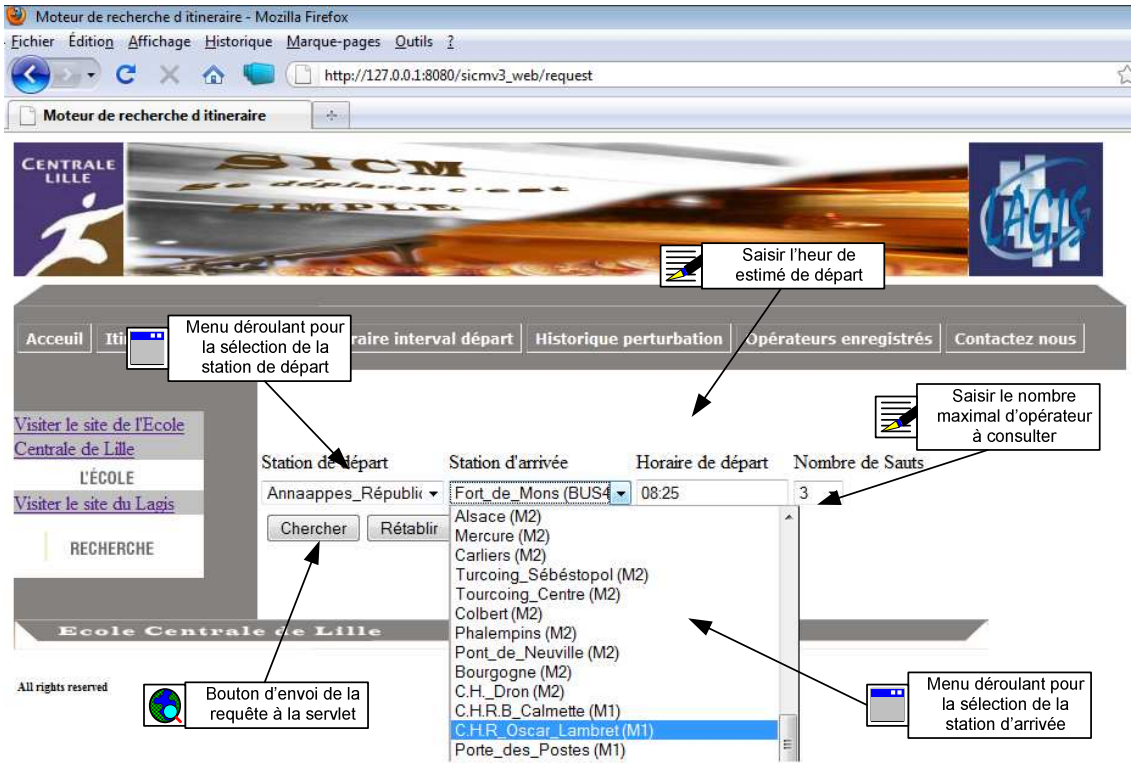

Figure IV-18 - Page de recherche

- **Itinéraire « intervalle départ »** : actionne une fonctionnalité différente. En effet, en cliquant sur ce bouton, la servlet génère une page où l'utilisateur peut sélectionner un intervalle d'heures de départ. Le système utilise ainsi un autre algorithme, il s'agit de l'algorithme TWTDA.

- **Historique « perturbation » :** on accède grâce à ce bouton à l'historique des perturbations enregistrées durant les sept derniers jours (paramètre réglable).

- **Opérateurs enregistrés :** nous fourni la liste des SIAD actuellement enregistrés dans notre système et qui seront par la suite considérés lors de l'exécution des algorithmes.

En utilisant la recherche avec ***intervalle d'heures de départ***, nous aboutissons à une page spécifique [Figure IV-19].

La page de recherche par intervalle d'heures de départ nous offre quatre champs pour saisir les détails de notre requête. Le premier champ est un menu déroulant contenant les différentes stations susceptibles d'être des stations de départ. Le deuxième champ est un autre menu déroulant qui permet de sélectionner la station de destination.

Figure IV-19 - Recherche avec intervalle de départ

Le troisième champ, est un champ « texte » qui permet de saisir la borne inférieure de l'intervalle d'heures de départ estimé. Le quatrième champ, est également un champ « texte », permet quant à lui de sélectionner la borne supérieure de cet intervalle. L'heure saisie doit être de la forme « hh:mm » c'est-à-dire les deux chiffres nous informent de l'heure, suivis des deux chiffres indiquant les minutes. En cas de saisie erronée, un message d'erreur s'affiche à l'écran nous demandant de saisir à nouveau la valeur de l'heure [Figure IV-20].

Figure IV-20 - Message d'erreur de saisie de l'heure

Une fois ce paramètre fixé, il suffit de cliquer sur le bouton « chercher » pour enclencher la recherche de chemins optimisés. Ceci génère une requête qui sera envoyée à la servlet. Cette dernière la transmet à son tour via le niveau service à la plateforme SMA.

En développement, une fenêtre de débogage apparaît lors de la recherche d'itinéraire permettant ainsi de suivre les différentes étapes de l'algorithme. Cette fenêtre est créée par l'agent débogage de la plateforme. Elle se présente comme une grande fenêtre constituant le conteneur actif de la plateforme

Jade renfermant plusieurs petites fenêtres chacune d'elles est consacrée à la description de l'état d'un agent. (Voir Figure IV-21)

Grace à cette fenêtre on peut remarquer l'avancement parallèle de l'algorithme et les communications entre les différents agents.

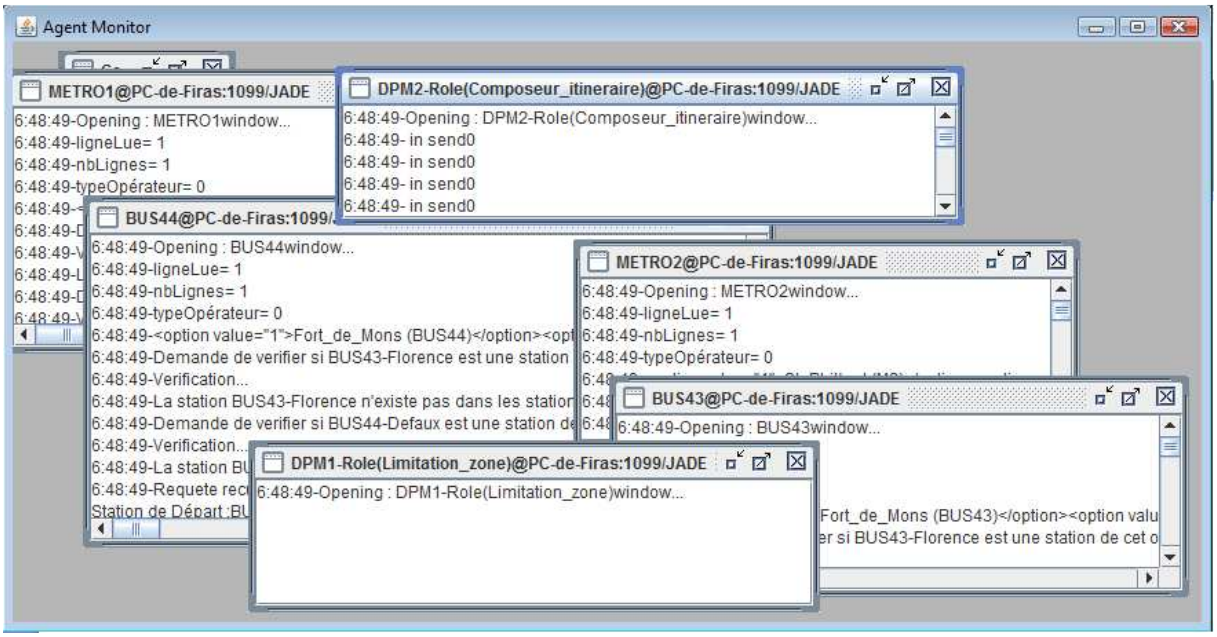

Figure IV-21 - Fenêtre de l'agent débogage

Dès que le SMA termine ses calculs, il envoie le résultat via le niveau service à la servlet, nous accédons alors à la réponse de la requête envoyée. Elle reformule le résultat pour qu'il soit exploitable par le serveur web ou le PDA.

Dans la page de résultat, on affiche pour chaque itinéraire possible un tableau indiquant les différentes étapes du trajet. Chaque étape concerne un opérateur et elle est constituée de l'heure et la station de départ, l'heure et la station d'arrivée et la ligne utilisée.

Pour mieux illustrer le fonctionnement en cas normal et en cas de perturbation, nous considèrerons deux scénarios différents. Le premier concerne une perturbation qui se produit avant le départ du voyageur et le deuxième décrit une perturbation qui se produit au cours du voyage.

IV.4.4. Scénario 1 : Une perturbation se produit avant le départ du voyageur

A 7 heures du matin, un étudiant planifie son déplacement. Habitant à proximité de la station « Florence » de la ligne du Bus 43 à Lille, il souhaite aller à la station « Defaux » de la ligne de Bus 44 avec un départ de la station « Florence » à une heure entre 8h30 et 8h40 [Figure IV-22].

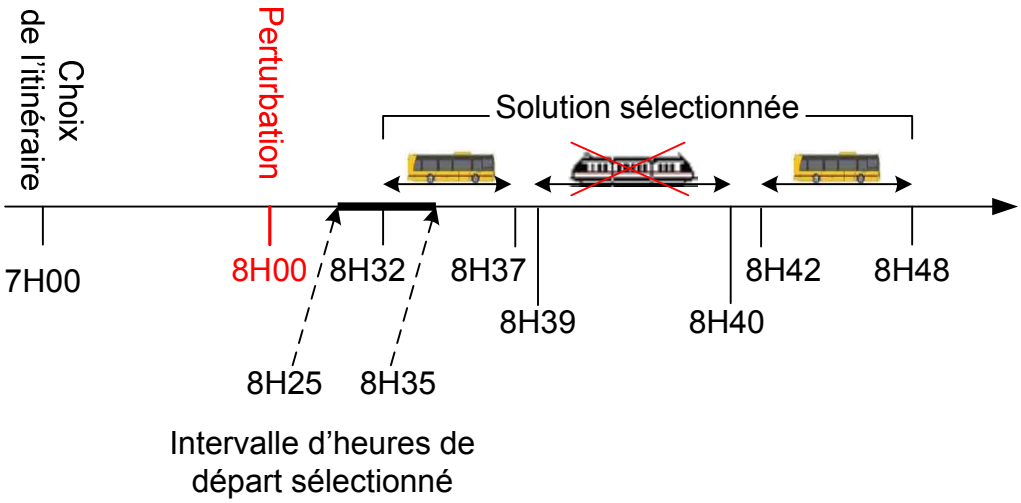

Figure IV-22 - Scénario 1

Il se connecte avec son PC au système DPM et s'identifie en indiquant son numéro de téléphone portable. Un agent du DPM (qu'on notera $Agent_{U\_Etudiant}$) sera créé et gardera en mémoire ce numéro de téléphone.

L'utilisateur recherche par la suite le chemin optimal correspondant aux critères décrits précédemment. Il accède alors à la page de recherche d'itinéraire avec intervalle de départ (voir Figure IV-19) et sélectionne la station de départ « Florence » et la station d'arrivée « Defaux ». Il pense partir entre 8H25 et 8H35, il fixe alors l'intervalle d'heures de départs de 8H25 à 8H35 et clique sur le bouton « Chercher ».

Le DPM lui propose un carnet de voyage avec plusieurs itinéraires (voir Figure IV-23). Le premier, consiste à prendre le bus 43 de Florence le départ est à 8h29 et de descendre à Fort de Mons attendre jusqu'à 8h38 et prendre le bus 44 pour arriver à destination(Défaux) à 8h53.

Le deuxième, consiste à prendre le bus 43 à 8h32 , descendre à Pont de bois, prendre le métro 1 de 8h39, descendre à « Villeneuve d'Ascq hôtel de ville » et enfin prendre le bus 44. L'arrivée à destination sera donc prévue à 8h48.

L'étudiant choisit d'emprunter la deuxième solution proposée, il la sélectionne dans l'IHM. L' $Agent_{U\_Etudiant}$ enregistre alors cette information et la lie au numéro de téléphone enregistré précédemment. De plus, suite à la sélection de l'itinéraire, l'heure de mort de cet agent est fixée à l'heure d'arrivée à destination de la solution sélectionnée qui correspond dans ce cas à 8H48.

Vers 8h, un incident technique immobilise le Métro 1. La ligne est donc coupée pour réparation pour une durée estimée à trente minutes.

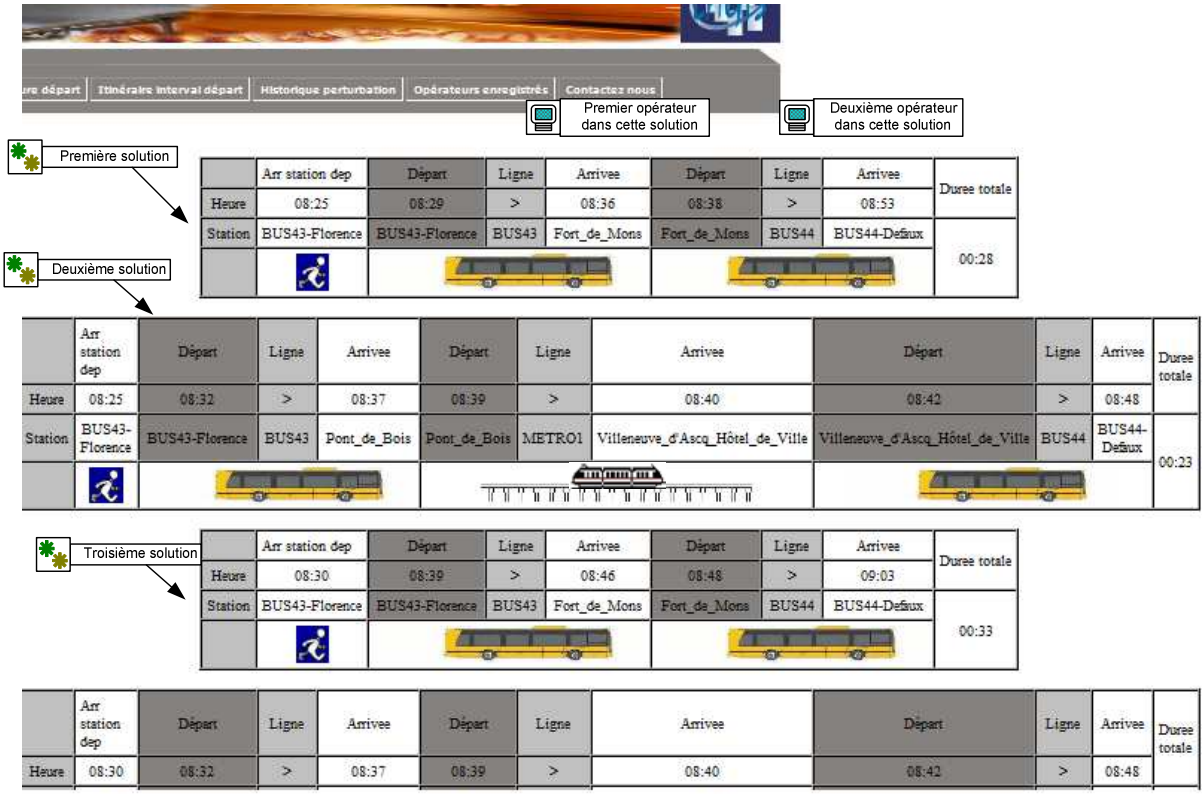

Figure IV-23 - Page résultat scénario 1

L'agent ($Agent_{S\_Metro1}$), jouant les rôles « Extraire Information » et « Détection perturbation » associé à l'SIAD de l'opérateur Métro1, détectera la perturbation dés que la mise à jour des données de l'opérateur Métro1 sera effective (L'action de mise à jour des données est sous la responsabilité de l'opérateur lui même, plutôt elle sera effectuée, meilleure sera sa qualité de service).

L'$Agent_{S\_Metro1}$ interroge l' $Agent_A$ (jouant le rôle « limitation de zone de recherche ») pour récupérer la liste des adresses des Agents vivants jouant le rôle « informer voyageur des perturbations ». Parmi les agents de la liste retournée, on retrouve l' $Agent_{U\_Etudiant}$. En effet, il est encore en vie vue que sa date de mort est fixée à 8h48.

L' $Agent_{U\_Etudiant}$ se charge alors d'estimer l'impact de la perturbation sur l'itinéraire sélectionné, il trouve qu'un tronçon de cet itinéraire va être perturbé. Il compare alors l'heure système (8h05) avec la première heure de départ (8h32), il en déduit que le voyageur n'est pas encore parti. L' $Agent_{U\_Etudiant}$ ré-exécute alors l'algorithme dynamique de recherche d'itinéraire distribué avec intervalle de départ en activant les opérateurs de transport individuels. Un SMS sera envoyé à l'étudiant pour le prévenir de la perturbation, et lui proposant un itinéraire de secours (voirFigure IV-24).

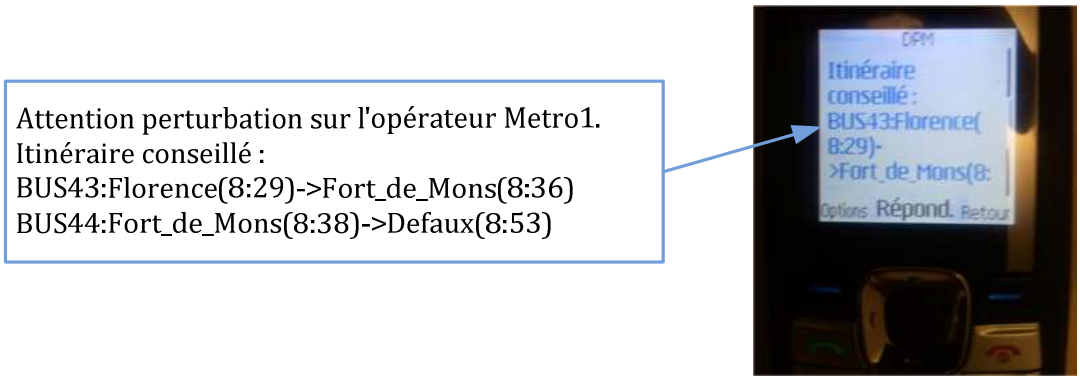

Figure IV-24 - Informer par SMS

IV.4.5. Scénario 2 : Une perturbation se produit durant le déplacement du voyageur

Il est 9h00, un homme d'affaires se trouve à proximité de la station « St. Philibert » de la ligne de Métro 2. Il décide d'aller à la station « 4 Cantons » de la ligne de Métro 1. Il utilise alors le système DPM via son PDA pour trouver un chemin optimisé. Pour cela, il se connecte en précisant son e-mail et accède à la page de recherche d'itinéraire avec intervalle de départ. Il sélectionne sur cette page la station d'arrivée « 4 Cantons», la station de départ « St Philibert » est présélectionnée automatiquement grâce à la reconnaissance des coordonnées GPS. Il fixe l'intervalle d'heures de départ de 9h à 9h15.

Le DPM lui propose un carnet de voyage avec plusieurs itinéraires [Figure IV-25]. Il choisit d'emprunter la première solution qui consiste à prendre le Metro 2 de St Philibert à 9h02 et de descendre à Porte des Postes, attendre jusqu'à 9h15 et prendre le Métro 1 pour arriver à 4 Cantons à 9h34.

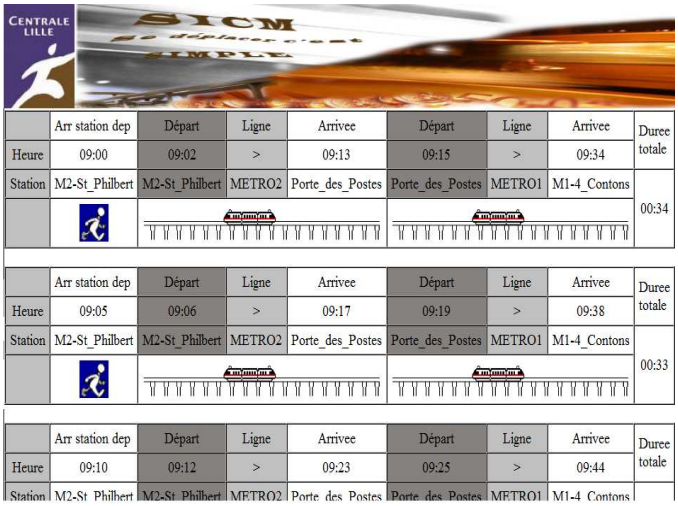

Figure IV-25 - Page résultat scénario 2

Au cours du voyage (vers 9h05), une panne provoque la coupure de la ligne de métro 1 entre les stations « Fives » et « Hellemes » , ce qui engendre une paralysie totale de cette ligne. Le temps de réparation est estimé à 30 minutes. L'agent ($Agent_{S\_Metro1}$) jouant le rôle « Détection perturbation » détecte la perturbation et informe tous les agents gérant les requêtes des utilisateurs. Parmi ces agents on trouve l'$Agent_{U\_HA}$ qui reçoit à son tour l'information de perturbation. Il commence par vérifier l'impact de cette perturbation sur l'itinéraire proposé à l'homme d'affaires. $Agent_{U\_HA}$ découvre alors que l'homme d'affaires est en cours de voyage et que le tronçon suivant de son l'itinéraire est impacté par la perturbation.

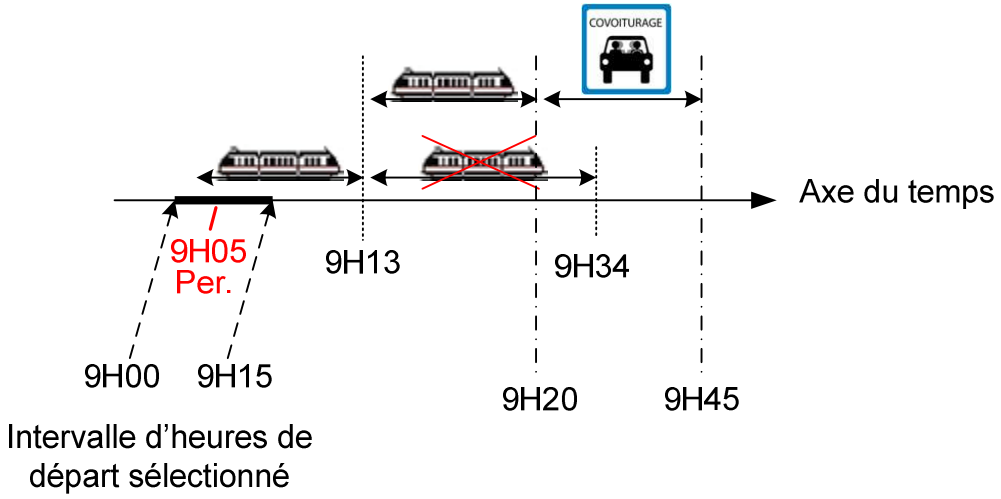

Figure IV-26 - Scénario 2

L'$Agent_{U\_HA}$ utilise alors l'algorithme dynamique de recherche d'itinéraire distribué en activant les opérateurs de transport individuels (dans notre cas le site de Covoiturage). Il recherche un itinéraire partant de « Porte des Postes » (la prochaine station du mode de transport emprunté) toujours à destination de « 4 Cantons », mais avec une heure de départ égale à l'heure d'arrivée à la station « Porte des Postes » donc à 9h13 [Figure IV-26].

Le résultat de la recherche est un itinéraire comodal constitué d'un tronçon en métro 2 pour aller de « Porte des Postes » à « G. Flandres » avec une arrivée à la gare Lille Flandres à 9h19. Et un tronçon en covoiturage pour aller de la Gare Lille Flandres à 4 Cantons avec une arrivée à destination à 9h45.

L'information de perturbation et l'itinéraire de secours sont envoyés par e-mail à l'homme d'affaires lui évitant ainsi une grande perte de temps due à ce problème technique.

IV.4.6. Conclusion

Dans ce chapitre, nous avons commencé par détailler le développement de chaque niveau du système, nous avons mis l'accent sur les différentes plateformes multi-agent et nous avons expliqué les raisons du choix de la plateforme JADE. Nous avons par la suite exposé deux possibilités de déploiement de la plateforme multi-agent en comparant les avantages et les inconvénients de chaque disposition.

Nous nous sommes intéressés par la suite à l'utilisation de la géolocalisation qui a donné une nouvelle dimension à notre système. Ceci a été réalisé par le développement d'une application PDA offrant plusieurs services (y compris un service de recherche d'itinéraire) exploitant la géolocalisation du PDA.

Finalement, nous avons présenté un exemple d'utilisation en mettant l'accent sur l'aspect graphique.

Conclusion générale

Le transport de personnes est un problème de plus en plus complexe. En effet, il faut prendre en considération plusieurs facteurs déterminants : besoin grandissant de planification des trajets, optimisation des temps de trajets et de leurs coûts, gestion efficiente des problèmes susceptibles de survenir en cours de voyage - comme les grèves et les incidents.

Nous nous sommes proposés dans ce travail de résoudre ces problèmes relatifs au transport de personnes et, plus précisément, aux services proposés aux clients des transports dans un environnement distribué où plusieurs opérateurs de transport présentent leurs offres. Le but étant de faciliter le choix d'un itinéraire en fusionnant les offres de transports en commun à celles des transports individuels (comme le covoiturage et l'AutoPartage) pour former un itinéraire comodal. Notre objectif était donc de faciliter la planification de ces itinéraires et de garantir un support aux utilisateurs au cours de leurs déplacements.

Tout au long de ce mémoire de thèse, nous avons tenté de détailler et d'expliciter les étapes suivies pour mener à bien ces travaux de recherche. De la problématique à la phase de réalisation en passant par la phase d'étude, ce mémoire a été l'occasion de synthétiser ces trois années d'investigation.

Pour réaliser ce travail, nous avons commencé par étudier l'évolution du transport des personnes, l'impact des perturbations sur le trajet et l'efficacité de l'information voyageur. Ensuite, nous nous sommes intéressés aux systèmes d'information existants et en relation avec notre problématique. Ainsi, nous avons dégagé la difficulté de la recherche d'itinéraire dans un environnement distribué où les données sont éparpillées dans plusieurs bases de données associées à différents opérateurs.

Par la suite, nous avons étudié les architectures « système » existantes pour déterminer celles qui s'adaptent le mieux à notre problématique. Nous nous sommes, par ailleurs, investis dans l'étude des méthodologies de modélisation en mettant l'accent sur la modélisation multi-agent.

Une fois les méthodologies définies, nous avons réalisé une analyse descendante de la problématique en partant des objectifs pour aboutir à un système complet. Le système ainsi défini est composé de trois niveaux : un niveau *présentation* pour la communication avec le client, un niveau *services* pour le transfert de données et un niveau *SMA* pour le calcul distribué des itinéraires.

Par la suite, nous avons détaillé chaque niveau en expliquant les choix retenus. Pour définir le niveau SMA, nous avons utilisé la méthodologie O-MaSE. Cette dernière est une méthodologie organisationnelle qui sépare les agents par rapport aux rôles qu'ils jouent, ce qui autorise une grande

flexibilité de conception. Nous avons ainsi proposé un système capable de gérer une perturbation en informant le voyageur concerné et en lui proposant des itinéraires de secours.

Pour étudier la problématique de calcul d'itinéraire dans notre contexte, à savoir dans un environnement dynamique et distribué, nous nous sommes intéressés aux différents algorithmes de recherche de plus court chemin et des chemins les plus rapides existants. Nous avons donc parcouru les algorithmes les plus connus dans des graphes statiques, dynamiques et distribués. Nous avons, par la suite, proposé un algorithme de recherche d'itinéraires dans un environnement distribué et dynamique. Cet algorithme fournit un carnet de voyage composé des chemins les plus rapides pour une station de départ et une station d'arrivée et un intervalle (fenêtre de temps) d'heures de départ choisies par l'utilisateur. L'algorithme proposé est utilisé par un agent de SMA qui joue le rôle « Composeur itinéraire ».

Nous avons ainsi proposé un système d'information à base d'agents qui permet de partir d'une requête saisie par l'utilisateur et de proposer le carnet de voyage correspondant. Ce carnet de voyage contient l'ensemble des solutions optimisées sous forme d'itinéraires entre les deux stations de départ et d'arrivée saisies par l'utilisateur.

Notre système d'information est fondé sur la communication entre les différents agents qui le composent. En effet, les agents commencent par créer une liste de chemins possibles liant la station de départ à celle d'arrivée. Chaque chemin étant constitué d'une succession d'opérateurs. Cette sélection permet de limiter le domaine de recherche ; elle diminue donc le temps de calcul. Ensuite, un agent se charge de décomposer la requête globale saisie par l'utilisateur en un ensemble de requêtes locales. Chacune de ces requêtes est traitée par un agent (qui joue le rôle d'« Extracteur d'information ») pour extraire la meilleure combinaison d'offre locale à partir du système d'information correspondant. Enfin, un agent se charge de combiner les résultats obtenus pour générer le carnet de voyage. Ce carnet transite sur les niveaux « service » et « présentation », puis il est mis en forme pour pouvoir être affiché sur le dispositif utilisé (ordinateur, téléphone portable, PDA). Ainsi, notre système arrive à composer un calcul d'itinéraire lourd et complexe, en plusieurs calculs locaux, plus simples, s'exécutant en parallèle et facilite la communication entre l'utilisateur et les systèmes des différentes situations.

Notre système a été testé sur des données générées automatiquement, mais très proches de la réalité ainsi que des données réelles de réseaux de transports existants. Les résultats obtenus nous ont permis de valider plusieurs aspects de notre système.

En outre, nous avons développé un autre système pour les PDA et téléphones portables, faisant l'interface entre l'utilisateur et divers SI extérieurs, en prenant en compte la géolocalisation (fournie par une puce GPS présente dans le PDA). Ce deuxième système propose plusieurs services distincts.

Parmi ces services, on peut citer un service de « recherche de boulangeries », « recherche de services culturels », un service de « calcul d'itinéraires » (DPM développé initialement). Ceci a permis une exploitation étendue des fonctionnalités de notre système et facilite son utilisation au cours des déplacements.

Plusieurs perspectives d'évolution et d'amélioration peuvent être envisagées si l'on prend en considération la multitude de domaines auxquels notre système peut s'appliquer. En effet, sur un plan fonctionnel, il serait intéressant d'améliorer la qualité du service de transport en étudiant la recherche d'itinéraires à moindre coût ou la recherche multicritère d'itinéraires (temps, coût, émission de gaz à effet de serre …) toujours dans un environnement dynamique distribué. En effet, notre travail se limite à la fonctionnalité de recherche des itinéraires les plus rapides et à l'assistance en cas de perturbation. Dans cette optique, un travail de recherche [**61**] financé par la région Nord-Pas-De Calais et l'ADEME[1] est en cours de réalisation au sein de notre équipe qui vise à optimiser la recherche multicritères favorisant la complémentarité entre transport en commun et éco-partage. De plus, pour renforcer le service covoiturage exploitable par notre système, un deuxième travail de recherche est actuellement en cours [**103**] et financé par le CISIT visant une meilleure gestion du service de covoiturage dynamique.

D'autre part, notre étude est fondée sur des algorithmes exacts, qui assurent l'optimalité des résultats obtenus. Il est intéressant d'avoir une vision probabiliste (graphe stochastique) sur les itinéraires, surtout en cas de perturbation. Une extension de ce travail pourrait donc être constituée par la recherche des chemins les plus rapides dans un environnement distribué dynamique stochastique. D'autre part, sur un plan technique, il serait sans doute effectivement fécond d'étudier la mise en place du système multi-agent proposé dans un environnement formé uniquement des nouveaux appareils émergents type Webphone (capable d'accueillir un ou plusieurs agents vu la puissance de leurs processeurs) et communiquant uniquement via le réseau GPRS.

[1] http://www2.ademe.fr

Bibliographie

[1] AFNOR, "Dictionnaire de l'informatique : le vocabulaire normalisé," ISO AFNOR, 1997.

[2] B.H. Ahn and J.Y. Shin, "Vehicle-Routing with Time Windows and Time-Varying Congestion," *The Journal of the Operational Research Society*, vol. 42, no. 5, pp. 393-400, 1991.

[3] ARENE, "Intermodalité, transport collectif et vélo : exemple de deux pôles vélo," ARENE – ADEME, fiche d'opération 2002.

[4] F. Bachman et al., "Technical Concepts of Component-Based Software Engineering," Software Engineering Institute, Carnegie Mellon, Pittsburgh, PA, États-Unis, Rapport Technique CMU/SEI-2000-TR-008 & ESC-TR-2000-007, 2000.

[5] B.M. Balachandran and M. Enkhsaikhan, "Development of a Multi-Agent System for Travel Industry Support," *Conference on Intelligent Agents, Web Technologies and Internet Commerce*, no. 63–63, Novembre 2006.

[6] F. Barbier et al., "Composants dans l'ingénieurie des systèmes d'information : concepts clés et techniques de réutilisation," *actes des deuxièmes assises nationales du GdR I3 (Information - Interaction - Intelligence)*, 2002.

[7] K.J. Barker et al., "A performance evaluation of the Nehalem quad-core processor for scientific computing," *Parallel Processing Letters*, pp. 453-469, 2008.

[8] B. Bauer and J. Mueller, "Methodologies and modeling languages," in *Agent-Based Software Development*. Londo: Artech House, 2004, pp. 77-131.

[9] F. Bellifemine, G Caire, A. Poggi, and G. Rima, "JADE," *A White Pape*, vol. 3, no. 3, 2003.

[10] F. Bellifemine, A. Poggi, and G. Rimassa, "JADE – A FIPA-compliant agent framework," *Proceedings of the The Fourth International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agents*, 1999.

[11] R. E. Bellman, "On a routing problem," *Quarterly Applied Mathematics*, vol. 16, pp. 87-90, 1958.

[12] JY Blais and JM Rousseau, "Overview of HASTUS current and future versions," *Computeraided Transport Scheduling (Edited by JR Daduna and A. Wren)*, pp. pp. 175-187,

1988.

[13] G. Booch, *Conception orientée objets et applications*.: Addition-Wesley, 1992.

[14] P. Borne, B. Fayech, S. Hammadi, and S. Maouche, "Decision Support System for Urban transportation networks," *IEEE SMC Part C : Applications and Reviews, Special Issue on Decision Technologies in honour of Prof Madan Singh,* vol. 33, no. 1, pp. 67-77, 2003.

[15] J.-P. Briot, "Composants logiciels et systèmes multi-agents," in *Technologies des systèmes multi-agents et applications industrielles*.: Hermès Lavoisier, 2009, ch. 5, pp. 147-187.

[16] M. Garrijo, J. Gonzalez and J. R. Velasco C. Iglesias, "Analysis and Design of multiagent systems using MASCommonKADS," *Intelligent Agents IV : Agent Theories, Architectures and Languages*, 1997.

[17] "CCITT specification and description language (SDL)," Technical report ITU-T. Z100 (1993), 1994.

[18] I. Chabini, "A New Short Paths Algorithm for Discrete Dynamic Networks," *8th IFAC Symposium on Transportation Systems*, 1997.

[19] I. Chabini, "Aalgorithms and high performance computing for dynamic shortest paths and analytical dynamic traffic assignment models," Massachusetts institute of technology, Technical Report DTRS95-G-0001, 2000.

[20] I. Chabini, "Discrete dynamic shortest path problems in transportation applications : Complexity and algorithms with optimal run time," *Transportation Research Record*, pp. 170-175, 1998.

[21] I. Chabini, A. Glenn, and S. Pallottino, "Reoptimization Algorithms for Minimum-Time Path Problems in Dynamic Networks," Università di Pisa, 1998.

[22] B. Chaib-draa, "Distributed Artificial Intelligence : An overview," in *Encyclopedia Of Computer Science And Technology*, Marcel Dekker, Ed., 1994, vol. 31, pp. 215-243.

[23] L. Cooke and E. Halsey, "The Shortest Route Through a Network with Time-Dependent Internodal Transit Times ," *Journal of Mathematical Analysis and Applications* , pp. 492-498, 1966.

[24] J. Coquio, "La performance adaptative des systèmes de transports collectifs : Modélisation, mesures de vulnérabilité et évaluation quantitative du rôle de l'information des voyageurs dans

la régulation des situations perturbées," UNIVERSITÉ FRANÇOIS - RABELAIS DE TOURS, TOURS, Thèse de doctorat 2008.

[25] R. Courcy, "Les systèmes d'information en réadaptation," *Réseau international CIDIH et facteurs environnementaux*, vol. 1-2, no. 5, pp. 7-10, 1992.

[26] J.R. Daduna and M. Mojsilovic, "Computer-aided vehicle and duty scheduling using the HOT programme system," *Computer-Aided Transit Scheduling*, pp. 133-146.

[27] J.R. Daduna and J.M.P. Paixão, "Vehicle scheduling for public mass transit – an overview," *Computer-Aided Transit Scheduling Notes de Lectures dans Economics and Mathematical System*, vol. 430, pp. 76–90, 1995.

[28] B.C. Dean, "Shortest Paths in FIFO Time-Dependent Networks : Theory and Algorithms," Massachusetts Institute Of Technology, Rapport Technique 2001.

[29] S.A. DeLoach, "Engineering Organization-Based Multiagent Systems," in *Software Engineering for Multi-Agent Systems IV.*: Springer Berlin / Heidelberg, 2006, vol. 3914/2006, pp. 109-125.

[30] S.A. DeLoach, "Engineering Organizationbased Multiagent Systems," *The 4th International Workshop on Software Engineering for Largescale multiagent Systems (SELMAS'05)*, vol. 3914, pp. 109–125, May 2005.

[31] S.A. Deloach, *Multiagent Systems Engineering : A Methodology And Language for Designing Agent Systems*., 1999.

[32] S.A. Deloach, M.F. Wood, and C.H. Sparkman, "Multiagent Systems Engineering," *International Journal of Software Engineering and Knowledge Engineering*, vol. 11, no. 3, pp. 231-258, 2001.

[33] Y. Demazeau, "Multi-Agent Systemes : MAS," *Journée Industrie-Recherche. Les Systèmes multi-agents (SMA) et leurs applications*, Novembre 2000.

[34] Business Dictionary. [Online]. www.businessdictionary.com

[35] E. W. DIJKSTRA, "A Note on Two Problems in Connection," *Numeriche Mathematik*, vol. 1, pp. 269–271, 1959.

[36] E. W. Dijkstra, "A Note on Two Problems in Connection," *Numeriche Mathematik*, vol. 1, pp.

269–271, 1959.

[37] S. Dreyfus, "An Appraisal of Some Shortest Path Algorithms," *Operations Research*, vol. 17, pp. 395-412, 1969.

[38] G. Dupuy, "La dépendance automobile : symptômes, analyses, diagnostic," *Anthropos, Ed. Economica*, p. 157, 1999.

[39] D. Eppstein, "Finding the k shortest paths," *SIAM J. Computing*, vol. 28, no. 2, pp. 652–673, 1998.

[40] European Commission, "A sustainable future for transport: Towards an integrated, technology-led and user friendly system," in *Publications Office of the European Union*, Luxembourg, 2009, p. 26.

[41] European Commission, "European transport policy for 2010: time to decide," White Paper 2001.

[42] "Examen à mi-parcours du livre blanc sur les transports publié en 2001," Journal officiel de l'Union européenne, Avis du Comité des régions 2007.

[43] M.F. Feki and S. Hammadi, "Disturbance Management in distributed Travel Information System," *18th IFAC World Congress (IWC)*, August - September 2011.

[44] M.F. Feki and S. Hammadi, "The fastest paths in time-dependent decentralized travel information system with time-window as departure time," *Computer and Simulationin Modern Science - WSEAS Press*, pp. 129-135, 2009.

[45] M.F. Feki, M.A. Kamoun, and S. Hammadi, "Advanced Travel Information System: An Agent-based Approach for Itineraries Web-Services Composition," *Buletinul U.P.G. din Ploiesti*, vol. Vol. LXI, no. 1/2009, pp. 51-64, 2009.

[46] M.F. Feki, M.A. Kamoun, and S. Hammadi, "An Agent Oriented Information System for Itineraries Search Using Web Services Composition," *Proc. IEEE International Conference on Systems, Man, and Cybernetics*, 2008.

[47] J. Ferber, *Les systèmes Multi-Agents : vers une intelligence collective*. Paris, France: InterEdition, 1995.

[48] R.T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures,"

Université de Californie, Irvine, Thèse de doctorat 2000.

[49] L. R. Ford and D. R. Fulkerson, "Flows in Networks," Princeton University, New Jersey, 1962.

[50] B. L. Fox., "k-th shortest paths and applications to the probabilistic networks," *RSA/TIMS Joint National Mtg*, 1975.

[51] GN. Frederickson, "Fast algorithms for shortest paths in planar graphs, with applications," *SIAM J Comput*, vol. 16, no. 1004-1022, 1987.

[52] R. Freling, A.P.M. Wagelmans, and J.M.P. Paixão, "An overview of models and techniques for integrating vehicle and crew scheduling," *Wilson, N.H.M. (Ed.) Computer-Aided Transit Schedul-ing. Notes de Lecture dans Economics and Mathematical Systems*, vol. 471, pp. 441–460, 1999.

[53] G. Gallo, G. Longo, S Nguyen, and S. Pallottino, "Directed hypergraphs and applications," *Discrete Applied Mathematics*, vol. 40, pp. 177-201, 1992.

[54] L. Gasser, "Agents and Concurrent Objects," *Interview par J.-P. Briot, Special series on Actors and Agents*, pp. 74–81, 1998.

[55] G. A. Giannopoulos, "The Application of Co-modality in Greece: a Critical Appraisal of Progress in the Development of Co-modal Freight Centres and Logistics Services," *Transition Studies Review*, no. Volume 15, Number 2 / septembre 2008, pp. 289-301, 2008.

[56] A. Gille, "La co-modalité outil du developpement durable," *Transports*, 2006.

[57] F. Guerriero and R. Musmanno, "Parallel asychronous algorithms for the K shortest paths problems," *Journal of Optimization Theory and Applications*, vol. 104, pp. 91 - 108, 2000.

[58] M.M. Hizem, "Recherche de chemins dans un graphe à pondération dynamique Application à l'optimisation d'itinéraires dans les réseaux routiers," École Centrale de Lille, Lille, Thèse de doctorat 2008.

[59] C.A. Iglesias, M. Garijo, and J.C. Gonz, "A methodological proposal for multiagent systems development extending CommonKADS," *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop. Track Agent-Oriented Approaches To Knowledge Engineering*, vol. 1, pp. 25–1/17, November 1996.

[60] I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard, "Object-Oriented Software Engineering.," *ACM Press*, 1992.

[61] K. JERIBI, "Gestion optimisée de véhicules partagés : vers une évaluation de services de mobilité avancée," Ecole Centrale de Lille, Lille, Thèse de doctorat prévu 2013.

[62] M.A. Kamoun, "Conception d'un système d'information pour l'aide au déplacement multimodal : Une approche multi-agents pour la recherche et la composition des itinéraires en ligne.," Ecole Centrale de Lille, Lille, Thèse de doctorat 2007.

[63] M.A. Kamoun, G. Uster, and S. Hammadi, "An Agent-Based Cooperative Information System for Multimodal Transport's Travelers Assistance," *Proc. IMACS 2005 Scientific Computation: Applied Mathematics and Simulation*, 2005a.

[64] M.A. Kamoun, G. Uster, and S. Hammadi, "An Agent-Based Cooperative Information System for Multi-modal Travelers Route Computation," *Proc. IEEE International Conference on Systems, Man, and Cybernetics*, 2005b.

[65] T Kampke and M Schaal, "Distributed generation of fastest paths," *Conference on Parallel and Distributed Computing and Systems*, pp. 172-177, 1998.

[66] D.E. Kaufman and R.L. Smith, "Fastest Paths in Time-Dependent Networks for Intelligent Vehicle-Highway Systems Application," *Journal of Intelligent Transportation Systems*, vol. 1, no. 1, pp. 1-11, 1993.

[67] Georgeff M., Rao A. Kinny D., "A methodology and modeling technique for systems of BDI agents," *Agents Breaking Away : Proceedings of the Seventh European Workshop on Modelling Autonmous Agents in a Multi-Agent World MAAMAW'96*, 1996.

[68] P. Kruchten, *The Rational Unified Process: An Introduction, Second Edition, 2nd edition.*: Addison-Wesley Longman, 2000.

[69] A. Lahlouhi, "Méthodologie De Développement D'environnements de développement de SMA," *Techniques et sciences informatiques*, vol. 1, 2001.

[70] H. Laichour, "Modélisation Multi-Agent et Aide à la Décision : Application à la régulation des correspondances dans les réseaux de transport urbains," Université des sciences et technologies de Lille, Lille, Thèse de doctorat 2002.

[71] S. Leriche, "Architectures à composants et agents pour la conception d'applications réparties adaptables," Toulouse, Thèse de doctorat 2006.

[72] "Les Transports en Nord–Pas de Calais," Conseil Économique et Social Régional Nord – Pas de Calais, Séance Plénière d'information mai 2008.

[73] Pascoal M., Santos J. Martins E., "The k shortest paths problem," CISUC, Research Report 1998.

[74] G. Marzo Serugendo, A. Karageorgos, O.F. Rana, and F. Zambonelli, "Engineering Self-Organising Systems : Nature-Inspired Approaches to Software Engineering," *Lecture Notes in Artificial Intelligence (LNAI)*, vol. 2977, 2004.

[75] P. M. Maurer, "Components: What if they gave a revolution and nobody came," *IEEE Software*, pp. 28-34, 2000.

[76] T.D. Meijler and O. Nierstrasz, "Beyond Objects: Components," *Cooperative Information Systems: Current Trends and Directions, M.P. Papazoglou and G. Schlageter (Eds.)*, pp. 49-78, 1997.

[77] Ministère des Transport de l'Equipement du Tourisme et de la Mer, "La situation des transports et de la mobilité au terme de l'année 2006," Dossier de presse, janvier 2007.

[78] E. F. Moore, "The shortest path through a maze," *International Symposium on the Theory of Switching*, pp. 285-292, 1959.

[79] A. Mouloudi, "Intégration des besoins des utilisateurs pour la conception de systèmes d'information interactifs. Application à la conception d'un système d'information voyageurs multimodal (SIVM)," Université de Technologie de Compiègne, Thèse de doctorat 2007.

[80] Kh. Nguyen Duc, "Contribution à l'étude des problèmes de ré ordonnancement en cas de perturbation," INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE, Grenoble, Thèse de doctorat 2007.

[81] M.H. Nguyen, "Génie logiciel orienté agent," L'INSTITUT DE LA FRANCOPHONIE POUR L'INFORMATIQUE, TRAVAIL D'INTÉRÊT PERSONNEL ENCADRÉ (TIPE) 2006.

[82] S. Nguyen and D. Pretolani, "Shortest hyperpath problems on oriented hypergraphs," Centre de recherche sur les transports, Rapport technique 1994.

[83] L.R. Nielsen, K.A. Andersen, and D. Pretolani, "Finding the K shortest hyperpaths : algorithms and applications," Department of operations research, University of Aarhus, Rapport technique WP-2002-2, 2002.

[84] H. Nwana, D. Ndumu, L. Lee, and J. Collis, "ZEUS: A tool-kit for building distributed multi-agent systems," *Applied Artifical Intelligence*, vol. 13 , pp. 129–186, 1999.

[85] J. Odell, "Objects and Agents Compared," *Journal of Object Technology (JOT)*, vol. 1, Mai 2002.

[86] A. Orda and R. Rom, "Shortest Path and Minimum Delay Algorithms in Networks with Time-Dependent Edge Length," *Journal of the ACM*, 1990.

[87] M.M. Ould Sidi, "Contribution à l'amélioration des systèmes d'aide à la décision dans le domaine du transport," Ecole Centrale de Lille, Lille, Thèse de doctorat 2006.

[88] M. Oussalah, *Ingénierie des composants - Concepts, techniques et outils, Vuibert*. Vuibert, 2005.

[89] L. Padgham and M. Winikoff, "Prometheus: A Methodology for Developing Intelligent Agents," in *Agent-Oriented Software Engineering III*.: Springer Berlin / Heidelberg, 2003, vol. 2585/2003, pp. 174-185.

[90] S. Pallottino and M.G. Scutella, "Shortest path algorithms in Transportation models : classical and innovative aspects," *Dans Proceedings of the Equilibrium and Advanced Transportation Modelling Colloquium*, 1998.

[91] Ch. Papadimitriou and K. Steiglitz, "Combinatorial optimization, algorithms and complexity," *Prentice- Hall*, 1998.

[92] C. Perreau, "Les Systèmes d'informations Multimodale," Institut d'Etudes Politiques de Paris, Thèse de doctorat 2002.

[93] J.-F Perrot, "Objets, classes et héritage : Définitions, chapitre de Langages et modèles à objets - État des recherches et perspectives," in *Collection Didactique*, INRIA, Ed., 1998, pp. 3–31.

[94] P.M. Ricordel, "Programmation orientée multiagent : Développement de systems multi-agents Voyelles," Institut National Polytechniques de Grenoble, Grenoble, Thèse de doctorat 2001.

[95] J. Rouillard, T. Vantroys, and V. Chevrin, *Architecture Orientée Services, Une approche pragmatique des SOA*, Editions Vuiberts, Ed., 2007.

[96] J-M. Rousseau, *Computer Scheduling of Public Transport*. North-Holland, 1985, vol. II.

[97] E. Ruppert, "Parallel Algorithms for the k Shortest Paths and Related Problems," University of Toronto, Thèse de doctorat 1996.

[98] S.J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Englewood Cliffs (New Jersey): Prentice Hall, 1995.

[99] A. Sabas, M. Badri, and S. Delisle, "A Multidimentional Framework for the Evaluation of Multiagent System Methodologies," *6th World Multiconference on Systemics, Cybernetics and Informatics (SCI-2002)*, vol. I, pp. 211-216, 2002a.

[100] A. Sabas, S. Delisle, and M. Badri, "A Comparative Analysis of Multiagent System Development Methodologies: Towards a Unified Approach," *Third International Symposium "From Agent Theory to Agent Implementation" (AT2AI-3), Sixteenth European Meeting on Cybernetics and Systems Research*, vol. II, pp. 599-604, 2002.

[101] A. Sabas, S. Delisle, and M. Badri, "Vers une unification des méthodologies de développement des systèmes multiagents," *Actes des Journées francophones pour l'intelligence artificielle distribuée et les systèmes multi-agents*, pp. 327-330, 2001.

[102] A. Schreiber, B. J. Wielinga, J. M. Akkermans, and W. Van de Velde, "CommonKADS : A comprehensive Methodology for KBS Development," *IEEE Expert: Intelligent Systems and Their Applications*, vol. 9, no. 6, pp. 28-37, 1994.

[103] M. Sghaier, "Aide informatique à la gestion de véhicules partagés et de services de mobilité avancée," Ecole Centrale de Lille, Lille, Thèse de doctorat prévu 2012.

[104] Sh. Subramanian, "Routing Algorithms for Dynamic, Intelligent Transportation Networks," Faculty of the Virginia Polytechnic Institute and State University, Blacksburg, Virginia, Thèse de Doctorat 1997.

[105] C. Szypersky, *Component Software – Beyond Object-Oriented Programming*.: Addison-Wesley, 1998.

[106] J. Tranier, "Vers une vision intégrale des systèmes multi-agents," Université Montpellier II,

Montpellier, Thèse de doctorat 2007.

[107] G. Uster, "Développement de l'information multimodale en France : quels leviers actionner ?," *Annales des Ponts et Chaussées n°98*, pp. 22-26, 2001.

[108] G. Uster, "État des réflexions sur le développement de l'information multimodale en France," *Congrès UITP*, 2000.

[109] G. Uster, "Information Multimodale voyageurs aspects Institutionnels et Juridiques," *Centre de documentation de l'INRETS*, no. LI-UTG0078, 2001.

[110] M. Viroli, A. Ricci, and A. Omicini, "Operating instructions for intelligent agent coordination," *The Knowledge Engineering Review*, pp. 49-69, 2006.

[111] J. Wang and T. Kaempke, "Shortest route computation in distributed systems," *Computers and opérations research*, vol. 31, pp. 1621-1633, 2004.

[112] J. Wang and T. Kaempke, "The fastest itinerary in time-dependent decentralized travel information systems," *Journal of Combinatorial Optimization*, vol. 12, no. 3, 2006.

[113] G. Weiss, "Agent orientation in software engineering," *The Knowledge Engineering Review*, vol. 16, no. 4, pp. 349 - 373, 2001.

[114] G. Weiss, *Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence*. Cambridge, Massachusetts: The MIT Press, 1995.

[115] K. Wilber, *A Theory of Everything : An Integral Vision for Business, Politics, Science and Spirituality*. Shambhala, 2001.

[116] M. Wooldridge, N.R. Jennings, and D. Kinny, "The Gaia Methodology For Agent-Oriented Analysis and Design," *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 3 (3), pp. 285-312, 2000.

[117] A. Wren, *Computer Scheduling of Public Transport*. North Holland, 1981.

[118] A. Wren and M. Chamberlain, "The development of Micro-BUSMAN: scheduling on micro-computers," *Computer-aided transit scheduling*, pp. 160-174, 1988.

[119] A. Wren and J.-M. Rousseau, "Bus Driver Scheduling – An Overview," *Workshop on Computer Aided Scheduling of Public Transport*, pp. 173-187, 1993.

[120] A. Wren and BM. Smith, "Experiences with a crew scheduling system based on set covering," *Computeraided Transit Scheduling (Edited by JR Daduna and A. Wren)*, pp. 104-118, 1988.

[121] Q. Wu and J. Hartley, "Using k-shortest paths algorithms to accommodate user preferences in the optimization of public transport travel," *UKSIM*, pp. 113–117, 2004.

[122] QIUJIN WU and JOANNA HARTLEY, "Using k-shortest paths algorithms to accommodate user preferences in the optimization of public transport travel," *UKSIM*, pp. 113–117, 2004.

[123] J. Y. Yen, "Finding the K shortest loopless paths in a network," *Management Science*, vol. 17, pp. 712–716, 1971.

[124] F. Zambonelli, N.R. Jennings, and M.J. Wooldridge, "Developing Multiagent Systems: the Gaia Methodology," *ACM Transactions on Software Engineering and Methodology*, vol. 12, no. 3, July 2003.

[125] W.T. Zaumen and J.J. Garcia-Luna Aceves, "Dynamics of distributed shortest-path routing algorithms," *ACM SIGCOMM Computer Communication Review*, vol. 21, no. 4, pp. 31-42, 1991.

[126] H. 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," Ecole Centrale de Lille, Lille, Thèse de doctorat 2007.

[127] S. Zidi, "SARR : Système d'aide a la régulation et la reconfiguration des reseaux de transport multimodal," Ecole centrale de Lille, Thèse de doctorat 2007.

[128] K. Zidi, "Système Interactif d'Aide au Déplacement Multimodal (SIADM)," Ecole Centrale de Lille, Lille, Thèse de doctorat 2006.

[129] K. Zidi and S. Hammadi, "Algorithme génétique avec contrôle des opérateurs pour l'optimisation multicritère d'un déplacement dans un réseau de transport multimodal," *Revu électronique e-STA*, vol. 2, 2005.

[130] A. Ziliaskopoulos and H.S. Mahmassani, "Time-Dependent Shortest Path Algorithm for Real-Time Intelligent Vehicle Highway System Applications," *Transportation Research Record*, vol. 1408, pp. 94-100, 1993.

TITRE DE LA THESE :

Optimisation distribuée pour la recherche des itinéraires multi-opérateurs dans un réseau de transport co-modal

RESUME :

La politique des transports dans le monde et en Europe évolue vers une vision co-modale. Cette nouvelle politique n'oppose plus la voiture au transport public mais encourage une combinaison de tous les modes de transport en espérant ainsi assurer un développement rentable et durable.

Nous focalisons notre étude sur le service transport de personnes qui s'inscrit au cœur des politiques co-modales en combinant tous les modes de transport en commun (métro, bus..) et promeut de nouveaux modes d'utilisation de la voiture particulière comme le covoiturage (partage d'un véhicule personnel) ou l'AutoPartage (voiture en libre-service).

Toutefois, pour générer un itinéraire exploitant les services de plusieurs opérateurs de transport, il faut consulter plusieurs sites internet. Selon le déplacement à réaliser, cette tâche de planification complexe peut être très difficile à réaliser et ne garantit pas l'optimalité de l'itinéraire sélectionné.

Nous nous sommes donc intéressés à la conception d'un système d'aide au déplacement capable de fournir une information voyageur (co-modale) en mettant en relation plusieurs opérateurs de transport (en commun et individuel). Le système en question doit être capable d'assister l'utilisateur dans la phase de planification par la constitution d'un carnet de voyage proposant plusieurs itinéraires multi-opérateurs. De plus, il assiste l'utilisateur en cas de perturbation en l'informant et en lui proposant des itinéraires de secours.

Ce travail est basé sur des avancées technologiques qui facilitent l'optimisation dans un environnement distribué (Multi-agent - SOA) et rendent l'information accessible grâce à un grand nombre de médias (téléphone, PDA..)

TITLE :

Distributed optimization for multi-operator routes search in co-modal transport network.

ABSTRACT :

Today's transport policy in the world, and more specifically in Europe, is moving towards a co-modal vision. This new policy does not oppose private to public transport but encourages a combination of all modes of transportation in the hope of assuring a lasting development. We focus our study on one special service: people transport which combines all modes of public transport (train, subway, tram, bus ..) and integrates new ways of car use such as carpooling (sharing a personal vehicle and the travel cost between drivers and passengers) and the EcoPartage (self-service cars in town).

However, to generate a route using several transport operators, we have to check different Internet websites. This complex planning task can be very difficult and does not guarantee the optimality of the selected route.

We are therefore interested in designing a support information system capable of providing comodal information linking several transport operators (public and individual).

The system in question must be able to assist the user in the planning phase by offering a travelogue suggesting several possible routes; each route may include one or more transport operator (public or individual ones).

In addition, it assists the user during the trip by informing him in case of disturbance and by proposing alternatives routes if necessary.

This work is based on various technological developments in order to facilitate the information optimization in a distributed environment (Multi-agent - SOA) and make the information accessible to the user through a large number of media (telephone, mobile, PDA ...).