THESE DE DOCTORAT DE L'UNIVERSITE PARIS VI

Spécialité :
Informatique

Présentée pour obtenir le titre de
DOCTEUR DE L'UNIVERSITE PARIS VI

par
Walter MERLAT

TITRE
Adaptation dynamique de l'organisation dans les Systèmes Multi-Agents. Application à la conception des Systèmes Coopératifs Distribués et Ouverts.

Soutenue le 17 Décembre 1998.

Version définitive.

JURY

M. Pascal ESTRAILLIER,	Président
M. Yves DEMAZEAU,	Rapporteur
M. Joel QUINQUETON,	Rapporteur
M. Jean-Pierre BRIOT,	Directeur
M. Jacques FERBER,	Co-directeur
M. Thierry BOURON,	Examinateur
M. Paul O'BRIEN,	Examinateur
M. Jean-François PERROT,	Examinateur

Résumé / Abstract.

L'examen des SMAs de type «Systèmes Coopératifs Distribués et Ouverts» (SCDOs), nous permet d'identifier une propriété importante des relations de dépendance liant les agents les uns aux autres: leur corrélation.
L'analyse critique des modèles d'agents, et en particulier du modèle des «dépendances sociales» de Castelfranchi, montre que ces modèles ne rendent pas compte de cette propriété de manière satisfaisante.
Nous proposons un mécanisme d'interaction qui tienne compte de la corrélation des relations de dépendance. Ce mécanisme utilise des techniques de satisfaction de contraintes, et il est mis en œuvre par des agents mobiles.
Ce mécanisme est testé sur un SCDO type: le système de «l'Agence de Voyage». Une plate-forme d'exécution d'agents mobiles est également développée pour l'occasion, intitulée JavaNetAgents.

Mots clefs: systèmes multi-agents, interaction, agents mobiles, systèmes coopératifs distribués et ouverts.

By examining a kind of Multi-Agent System, «Cooperative Distributed and Open Systems» (CDOSs), we identify an important property of the dependence relationships that tie agents together: they are correlated.
A survey of agent models, and in particular the Castelfranchi «social dependence» model, shows that they fail to address this issue properly.
We propose an interaction procedure that takes into account the correlation of dependence relationships. It uses constraint satisfaction concepts, and is performed by mobile agents.
The procedure is tested on a typical CDOS: the Travel Agency system. A mobile agent platform is also built for this purpose.

Keywords: multi-agent systems, interaction, mobile agents, cooperative distributed and open systems.

Remerciements.

Ce travail n'aurait certainement pas vu le jour sans l'aide de nombreuses personnes.

En premier lieu, je tiens à remercier vivement et sincèrement Jean-Pierre Briot, mon directeur de thèse. Son aide a été inestimable pour transformer ce travail de trois années en une thèse.

Je remercie ensuite Yves Demazeau pour le temps qu'il a consacré à la lecture du mémoire, et les remarques enrichissantes qu'il a faites à son sujet. Je sais comme son temps lui est très précieux, et j'apprécie d'autant son implication en tant que rapporteur dans mon jury.

Je remercie également Joel Quinqueton pour l'intérêt qu'il a porté à ce travail, et les commentaires enrichissants qu'il a faits à son sujet. Je lui suis reconnaissant d'avoir pris de son temps (qui est également très précieux !) pour rapporter sur ma thèse.

Pascal Estraillier m'a fait un grand honneur en acceptant de présider mon jury de thèse. Ses commentaires judicieux sur mon travail soulignent la grande similitude des systèmes multi-agents et des systèmes distribués.

Je remercie Jean Erceau de m'avoir accueillit à l'ONERA et de m'avoir laissé une grande liberté dans mon travail. J'espère qu'il ne le regrette pas (trop). Mon contrat de doctorant ONERA spécifiait la tâche qui m'était confiée : « la réalisation d'outils en MeringIV pour la conception de systèmes distribués et coopératifs ». Mission presque accomplie : mes agents mobiles sont écrits en Java !

Je remercie Jacques Ferber de m'avoir accueillit dans l'équipe MIRIAD et de m'avoir introduit au monde des systèmes multi-agents.

Je remercie Thierry Bouron d'avoir accepté de faire partie de mon jury. Il n'imagine pas à quel point cela valorise ma thèse aux yeux de nos amis des BT Labs.

I would like to thank Paul O'Brien for listening carefully and showing interest during my 45 minute defence in French. He is very far from imagining how much more appealing this makes my PhD to the eyes of our friends from France Telecom.

Jean-François Perrot a été d'un grand secours dans mon jury, par sa seule présence!

Je remercie Jean-Pierre (encore !) et Madeleine Girard-Faugère pour m'avoir aidé à rédiger ce qui finalement devint un mémoire de thèse.

Durant mon DEA et les trois années qui suivirent, les membres de l'équipe MIRIAD m'ont fournit un support et des conseils particulièrement judicieux, fruits d'une expérience longuement mûrie. Les plus expérimentés d'entre eux m'ont prévenu de ce qui m'attendait et conseillé de ne pas faire de thèse. Comme à mon habitude, j'ai écouté attentivement les conseils des anciens... puis j'ai entrepris de me faire une opinion par moi-même. Les plus jeunes, en particulier Min, m'ont accompagné dans cette grande aventure qu'est une thèse dans l'équipe MIRIAD. Je ne citerai pas nommément tous les autres membres de l'équipe —passés et présents—, mais je leur adresse ici un remerciement collectif : ce fut agréable de ramer en cadence avec eux !

Je remercie les membres de la division des Systèmes Informatiques Avancés de l'ONERA, pour leur accueil et leur tolérance à mon égard. Je suis persuadé qu'ils trouveront quelque chose à faire d'un système d'agence de voyage avec des agents mobiles.

Finally, I would like to thank members of the Intelligent Business Systems Research Group at BT Labs, for their very warm welcome (so typically British!), and the freedom they allowed me in order to finish my PhD dissertation.

Table des matières.

Table des illustrations et des tableaux.

1. Introduction

Les travaux dans le domaine des Systèmes Multi-Agents (SMA) concernent l'étude des organisations d'agents. Un agent est une entité autonome et dépendante des autres agents du système dans lequel il se trouve. Autonomie et dépendance sont deux concepts clés, dont la conjugaison est à la base de la « problématique Multi-Agents » :

- d'une part, chaque agent dispose de ses propres règles de contrôle[1] (on parle de « contrôle local » à l'agent). Ces règles déterminent les désirs de l'agent, i.e. les objectifs qu'il adopte ou les actions qu'il souhaite réaliser. Elles assurent en général la maximisation d'une fonction de satisfaction ou la poursuite d'un objectif particulier, qui sont confiés à l'agent au moment de sa création et par son créateur (son « propriétaire humain » en général). Dans l'exemple de l'Agence de Voyage (présenté au §2.3), l'agent AgVoy, représentant une agence de voyage, dispose de ses propres règles de contrôle déterminant ses objectifs : AgVoy n'organise un voyage qu'à la demande d'un agent (le client), et si cet agent le paie suffisamment (règle de contrôle de type commercial).
- d'autre part, des relations de dépendance lient chaque agent à d'autres agents du SMA. Ces relations de dépendance sont similaires à des relations client/serveur : le besoin du client est satisfait par un service fourni par le serveur. Les relations de dépendance restreignent la capacité d'action d'un agent, en reliant ses besoins aux agents serveurs compétents : l'agent client ne peut agir que si les serveurs lui fournissent les services dont il a besoin. Les relations de dépendance déterminent ainsi quels objectifs de l'agent sont satisfiables —et lesquels ne le sont pas— en fonction de la composition qualitative du système (présence ou absence d'agents disposant de compétences particulières qui sont nécessaires pour atteindre certains objectifs). Dans l'exemple de l'Agence de Voyage, l'agent AgVoy ne dispose d'aucune ressource propre —ni trains, ni avions—. Pour organiser des voyages, il dépend donc d'autres agents du système, précisément les agents représentant des compagnies de transport (qui possèdent et gèrent leurs propres flottes de trains ou d'avions) : en l'absence d'agents fournissant des services de transport intérieurs en France, AgVoy ne peut satisfaire ses objectifs d'organiser des voyages au départ de villes françaises quelconques (il peut *adopter* de tels objectifs car il est autonome, mais il ne peut pas les *satisfaire*). En revanche, la présence de l'agent SNCF dans le système multi-agents, permet à AgVoy de réaliser de tels objectifs.

---

[1] « autonomie » vient du grec auto-nomos (αυτο νωμοζ) : « qui dispose de ses propres lois ».

Le rôle des relations de dépendance entre agents n'est donc pas négligeable dans un SMA : l'organisation, définie comme l'agencement des relations entre les agents, en est une caractéristique essentielle. Idéalement, l'organisation du système est adaptée aux agents et tous leurs objectifs peuvent être satisfaits car leurs besoins (les pré-requis associés aux objectifs) sont tous satisfaits : le système contient les agents serveurs compétents pour cela, et ces agents fournissent les services appropriés aux agents qui en ont besoin.

Cependant plusieurs événements survenant en phase d'exploitation d'un SMA peuvent provoquer l'inadaptation de son organisation. Les deux principaux sont :

- la génération d'un nouvel objectif par le contrôle local d'un agent (en particulier suite à un stimulus extérieur, comme par exemple la réception d'un message de requête de la part d'un agent client). La génération de cet objectif ajoute de nouveaux besoins à l'agent, que l'agent va essayer de satisfaire en obtenant des services de la part d'autres agents du système.
- la défaillance d'un agent serveur, qui le conduit à cesser prématurément et spontanément de fournir un service à l'un de ses client (pour cause de panne de son environnement d'exécution —machine, réseau—, ou pour toute autre raison). Cet événement rend l'organisation inadaptée puisque précisément le besoin satisfait par ce service, ne l'est plus.

De tels événements sont susceptibles de survenir à tout moment en phase d'exploitation d'un SMA, mais leur occurrence n'est pas prévisible lors de la phase de conception. Cette très forte indétermination, caractéristique de l'exploitation d'un SMA, explique qu'une organisation adaptée à un moment donné ne l'est vraisemblablement plus au moment suivant. En conséquence, les « bonnes » organisations ne sont pas statiques : elles ***s'adaptent dynamiquement*** aux évolutions du système. Une des principales tâches du concepteur d'un SMA est alors de concevoir des mécanismes d'auto-adaptation de l'organisation, qui sont ensuite activés en fonction des événements survenant en cours d'exploitation. Ce problème définit la « problématique Multi-Agents ».

La solution à ce problème prend la forme générale d'un échange de messages, principalement entre l'agent client et des agents serveurs qui sont susceptibles de satisfaire ses besoins. Ces échanges constituent une interaction. Sans entrer dans les détails (car l'objet de ce travail de thèse est justement de concevoir un mécanisme d'interaction !) l'interaction se déroule en deux étapes :

1. identification des agents serveurs compétents pour satisfaire les besoins,
2. négociation avec ces agents, pour que les services nécessaires soient effectivement fournis.

L'indétermination rapproche les domaines des Systèmes Multi-Agents de ceux des Systèmes Dynamiques et des Systèmes Complexes. Elle explique le rôle central —et commun à ces domaines— des principes d'*auto*-organisation, d'*auto*-régulation, ou encore d'*auto*-adaptation. Deux autres caractéristiques justifient la proximité de ces domaines :

- la complexité. Les relations de dépendance liant les agents au sein d'un SMA sont nombreuses et corrélées : elles ne peuvent être définies qu'ensemble, et non pas individuellement. Ceci constitue d'ailleurs un point central du problème traité dans cette thèse. Dans de telles configurations de relations de dépendance inter-agents, caractérisées par un réseau de boucles de rétroaction particulièrement dense, la moindre perturbation locale a des conséquences globales difficiles à évaluer. Ce phénomène peut conduire à une évolution chaotique du SMA, identique à celles observées et étudiées dans le cadre des Systèmes Dynamiques.

- le concept de collectif. Les Systèmes Multi-Agents sont le siège de phénomènes collectifs (i.e. impliquant plusieurs agents). L'importance du collectif apparaît clairement dans ce travail de thèse, où des regroupements d'agents serveurs sont formés par interaction entre divers agents. De manière similaire, l'observation de Systèmes Dynamiques et de Systèmes Complexes permet d'identifier des structures d'ordre de haut niveau qui « émergent » à partir des interactions entre entités de base.

Les domaines d'application des Systèmes Multi-Agents sont nombreux, en particulier en informatique et en télécommunications, où la problématique Multi-Agents facilite grandement la conception des Systèmes Coopératifs Distribués et Ouverts (SCDO). Les conditions d'exploitation particulières des SCDO expliquent en partie l'intérêt des concepts Multi-Agents pour la conception de tels systèmes : la dispersion géographique, les pannes machines/réseau rendent difficiles —voire momentanément impossibles— les communications entre applications, ce qui justifie l'intérêt de la décentralisation du contrôle et du calcul local, deux concepts clés des SMA. De plus, l'introduction et le retrait fréquents d'applications nécessitent l'adaptation dynamique de l'organisation du système. Un exemple représentatif de SCDO est l'Agence de Voyage, auquel il est fait continuellement référence au cours de ce mémoire. Dans l'industrie aérospatiale, les systèmes d'aide à la conception coopérative de satellite (et d'une manière plus générale, les systèmes d'aide au travail coopératif) sont également des SCDO.

Des travaux ont déjà réalisés dans le domaine des SMA, qui portent principalement sur la modélisation des mécanismes de contrôle local des agents. Il explicitent la manière dont les objectifs des agents sont générés : par exemple par le moyen de règles d'inférences sur des états mentaux pour les modèles d'agents intentionnels.

Cependant, ces modèles sont essentiellement « orienté agent », et négligent le rôle et les caractéristiques des relations de dépendance entre agents. En particulier, deux problèmes essentiels ne sont pas abordés dans ces travaux :

1. la prise en compte de la **corrélation des besoins** des agents. Les besoins d'un même agent ne peuvent en effet pas être définis indépendamment les uns des autres, et n'ont de sens que définis —et satisfaits— ensemble. Ce fait explique le nécessaire regroupement d'agents serveurs pour satisfaire les besoins d'un agent client : ils se regroupent et agissent d'une manière coordonnée afin de respecter la corrélation entre les besoins. La corrélation des besoins rend particulièrement délicate la conception de mécanismes d'adaptation dynamique de l'organisation, puisqu'il ne faut pas adapter l'organisation pour satisfaire

chaque besoin « au coup par coup », mais au contraire satisfaire tous les besoins corrélés « d'un coup ».
2. la prise en compte des **conditions d'exploitation** particulières des SMA de type SCDO (dispersion géographique, pannes machines/réseau). Il est particulièrement difficile de concevoir des mécanismes d'adaptation dynamique qui soient efficaces dans de telles conditions.

Notre travail de thèse a pour objectif de définir un mécanisme d'interaction efficace qui assure l'adaptation dynamique de l'organisation des SMA, et plus particulièrement des SMA de type SCDO. Il comblera donc les lacunes qui viennent d'être exposées. En nous basant sur l'exemple de l'Agence de Voyage —simple, mais très complet et représentatif—, nous déterminons un mécanisme d'interaction qui assure :
1. le regroupement d'un ensemble d'agents serveurs pour satisfaire les besoins corrélés d'un agent client, puis
2. l'adaptation dynamique de l'organisation interne de ce regroupement.

L'implémentation informatique de l'Agence de Voyage sert de banc de test pour ces concepts. En outre, la technique novatrice de la mobilité de code est proposée pour une implémentation efficace du mécanisme d'interaction en particulier dans les conditions d'exploitation particulières des SMA de type SCDO. Nous avons pour ce faire développé une plate-forme prototype d'exécution d'agents mobiles (intitulée JavaNetAgents).

Ce mémoire débute au chapitre §2 par la présentation des Systèmes Multi-Agents de type « Systèmes Distribués Coopératifs et Ouverts », et en particulier du système de l'Agence de Voyage, illustré par un scénario d'utilisation. L'analyse de ce scénario permet ensuite de dégager la problématique de notre travail de thèse (chapitre §3).
L'état de l'art suit (§4). Il est en deux parties. La première partie (§4.1) décrit les modèles « traditionnels » de Systèmes Multi-Agents. La deuxième partie (§4.2) présente les architectures de systèmes distribués « traditionnels » et introduit les architectures à base d'agents mobiles.
Le chapitre §5 décrit le mécanisme d'interaction entre agents, qui utilise le concept novateur de mobilité de code.
Le chapitre §6 décrit JavaNetAgents, la plate-forme d'exécution d'agents mobiles développée pour la réalisation informatique du mécanisme d'interaction, et du système complet de l'Agence de Voyage.
L'implémentation du système que nous proposons est présentée au chapitre §7 afin d'illustrer le mécanisme d'interaction et de montrer expérimentalement ses fonctionnalités..
L'annexe 1 donne des extraits du code source de l'implémentation de l'Agence de Voyage.

2. Les Systèmes Coopératifs Distribués et Ouverts

Ce chapitre présente les applications pratiques de cette thèse : l'adaptation dynamique de l'organisation dans les Systèmes Coopératifs Distribués et Ouverts (SCDOs). Les applications pratiques sont ainsi présentées avant le problème scientifique traité dans cette thèse. Ce choix de présentation facilite la description et l'analyse du problème scientifique (au §3) en utilisant l'exemple du SCDO « Agence de Voyage » détaillé dans ce chapitre.

Dans ce chapitre sont d'abord introduits les Systèmes Coopératifs Distribués et Ouverts. Deux exemples sont ensuite présentés. Le deuxième, celui de l'Agence de Voyage, est ensuite détaillé car il sert de référence durant tout le reste du mémoire.

2.1 Les SCDOs : les systèmes d'information des entreprises du 21ème siècle

2.1.1 Les entreprises virtuelles (EVs)

La compétition grandissante entre entreprises, l'évolution de plus en plus rapide des marchés, provoquent des changements constants dans les besoins des entreprises. En particulier, les entreprises relient de plus en plus souvent leurs systèmes d'information afin de coopérer plus étroitement et plus efficacement sur un projet commun. Le résultat est la constitution d'entreprises virtuelles (notées EVs dans la suite).

Du fait de la nature très compétitive de leur marchés respectifs, et de l'évolution très rapide de leur technologies, les industries automobile et aérospatiale (cf. la conception de satellite §2.2), ou l'industrie des télécommunications (fourniture de services à valeur ajoutée, comme les services de voyage clés en main « train + avion + hôtel », cf. §2.3) jouent un rôle pionnier dans l'apparition des EVs.

Les EVs se forment (et se dissolvent) au gré de projets communs, qui peuvent être des projets à long terme (conception/fabrication d'un satellite), ou bien très brefs (fourniture d'un voyage[2]).
Au sein d'une EV, chaque membre joue un rôle particulier, par exemple :

- conception/fabrication d'une pièce détachée entrant dans la composition d'une automobile ou d'un satellite, ou
- fourniture d'un service « élémentaire » de voyage : réservation de train, ou d'une chambre d'hôtel.

La composition des rôles de chaque membre permet à l'EV de fournir un service global à son ou ses clients : conception d'un nouveau type de voiture, ou fourniture d'un voyage « clés en main ».

L'organisation d'une EV est particulièrement dynamique et elle évolue continuellement durant son existence : de nouveaux membres peuvent être admis, d'autres peuvent en sortir. Ce dernier cas se produit notamment en cas de défaillance d'un membre de l'EV : s'il ne peut remplir le rôle qui lui est assigné (par exemple en cas de grève du personnel ou de panne d'un avion chez une compagnie aérienne), il est remplacé par un autre. En outre, les rôles peuvent également changer. Par exemple si un client décide de faire un détour au cours d'un voyage, l'EV doit modifier les réservations déjà faites (ce qui revient à modifier les rôles de ces membres), et éventuellement intégrer de nouveaux membres (pour fournir des billets de transport sur le trajet spécifique du détour).

Un point important à noter est la corrélation des rôles. Dans le cas de la conception d'une voiture ou d'un satellite, les spécifications des différentes pièces détachées sont mutuellement corrélées car des contraintes de compatibilité géométrique, de masse maximale, de budget maximal lient les pièces les unes aux autres. Dans le cas de services de voyage, des contraintes de compatibilité horaires (aux escales), et de budget lient de manière évidente les différents services « élémentaires » qui sont composés pour fournir un voyage « clés en main ».
La corrélation des rôles est importante : sa prise en compte nécessite la coopération étroite des différents membres de l'EV.

[2] Même si ce n'est pas encore le cas actuellement, il est tout à fait envisageable dans un avenir proche, de voir des EV se former pour la satisfaction *d'une requête unique*. Le développement de ce type d'EV dépend essentiellement de la durée et du coût de la constitution d'une EV. Ces paramètres sont est encore trop importants actuellement. Par exemple, de longues négociations juridiques précédent nécessairement la création d'un EV dans l'industrie aérospatiale ou automobile, à cause des budgets en jeu). Dans le cas de service de voyage cependant, les infrastructure de « commerce électronique » (établissement et certification « électronique » de contrats par ex.) permettront vraisemblablement de réduire la durée et le coût de la corrélation d'un EV dans le cas de services nécessitant un moindre investissement —et donc un moindre risque financier—, comme des voyages par exemple..

2.1.2 Les systèmes d'information des EVs : les SCDOs

L'organisation des EVs nécessite un système d'information approprié, qui facilite la coopération entre les membres de l'EV, mais aussi l'évolution de l'organisation de l'EV. Les Systèmes Coopératifs, Distribués et Ouverts (notés SCDOs par la suite) sont particulièrement adaptés à cette tâche.

Les SCDOs sont constitués du rassemblement des systèmes d'information individuels de chaque entreprise. La « valeur ajoutée » du SCDO est l'organisation des systèmes d'information individuels afin de faciliter leur coopération.

La Figure 1 présente une EV fournissant des services de transport « clés en main ». L'EV est constituée du rassemblement de deux compagnies de transports SNCF et AirFrance et d'une agence de voyage. Le SCDO regroupe leur systèmes d'information individuels respectifs, qui sont des systèmes de réservation de billets. Sa fonction est de faciliter la constitution de services de transport « clés en main » à partir des services de transport élémentaires fournis par AirFrance et SNCF.

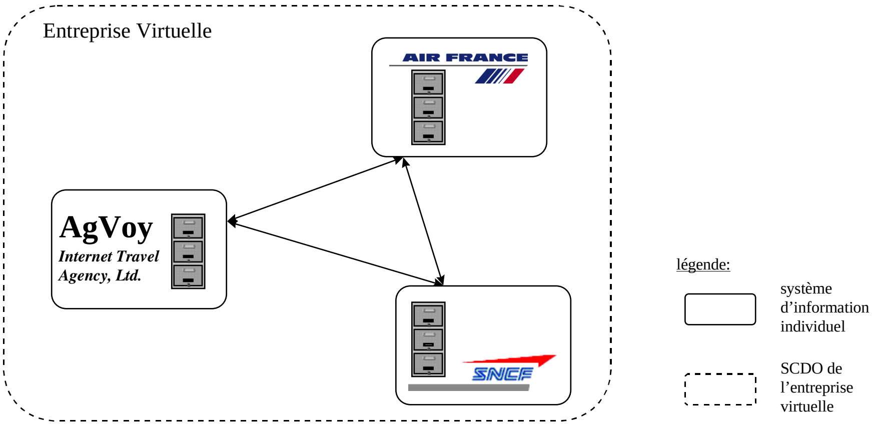


Figure 1. Exemple général de SCDO.

2.1.3 Les SCDOs sont des SMAs

La principale qualité d'un SCDO est de faciliter l'évolution dynamique de l'organisation de l'EV. Dans cette perspective, le paradigme Multi-Agents propose des concepts particulièrement intéressants pour la conception de SCDOs: organisation dynamique, autonomie de contrôle, décentralisation, négociation, coopération, etc.

En fait, les SCDOs sont un type particulier de SMA ([Ferber 95]). En effet, les systèmes d'information individuels des membres de l'EV sont des agents. En effet ils sont :

- autonomes, car ils mettent en oeuvre la politique commercial et industrielle de leur propre entreprise.
- et en interaction, puisqu'ils coopèrent au sein de l'EV.

2.2 Un SCDO pour la conception de satellite

De nombreuses études ont été menées sur les EVs dans l'industrie aérospatiale ([Park et al. 94], [NIIIP 95] [Herbert94], [I3 95], [Hale et al. 94]).

Un SCDO pour la conception coopérative de satellite a été l'objet du projet EDID mené à l'ONERA ([Barat et al.92]). Sa configuration est décrite sur la Figure 2 :

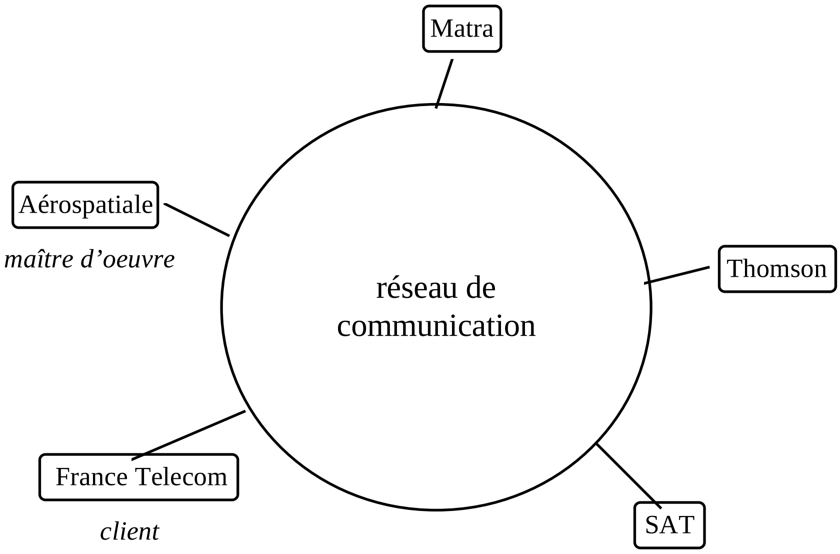


Figure 2. SCDO pour la conception de satellite.

Chaque entreprise est chargée de la conception d'une partie du satellite. Le client initial (France Telecom) commande le satellite et fournit donc un cahier des charges décrivant ses caractéristiques de manière très imprécise : durée de vie, nombre de communications simultanées, zone de couverture géographique, etc. Un maître d'oeuvre dispose des compétences pour analyser ce cahier des charges et décomposer le problème en un ensemble de sous-problèmes qui correspondent à la conception de différents composants du satellite, et qu'il sous-traite à des entreprises compétentes. De ce fait, un groupe d'entreprises se forme pour répondre à la requête initiale du client.

En l'occurrence, sur l'exemple de la Figure 2, trois entreprises interviennent et forment avec Aérospatiale une EV pour la conception du satellite[3] :

- Thomson est chargé de la conception de l'antenne parabolique.
- Matra fait les accumulateurs.
- SAT s'occupe des panneaux solaires.

[3] Le problème de la conception du satellite est bien évidemment exposé d'une manière très simplifiée. Les industriels cités ne le sont qu'à titre illustratif.

La corrélation des caractéristiques des différents composants du satellite est évidente, du fait que ces composants seront assemblés pour former le satellite. Les corrélations sont représentées sur la Figure 3 par un réseau de contraintes liant les paramètres caractéristiques des différents composants.

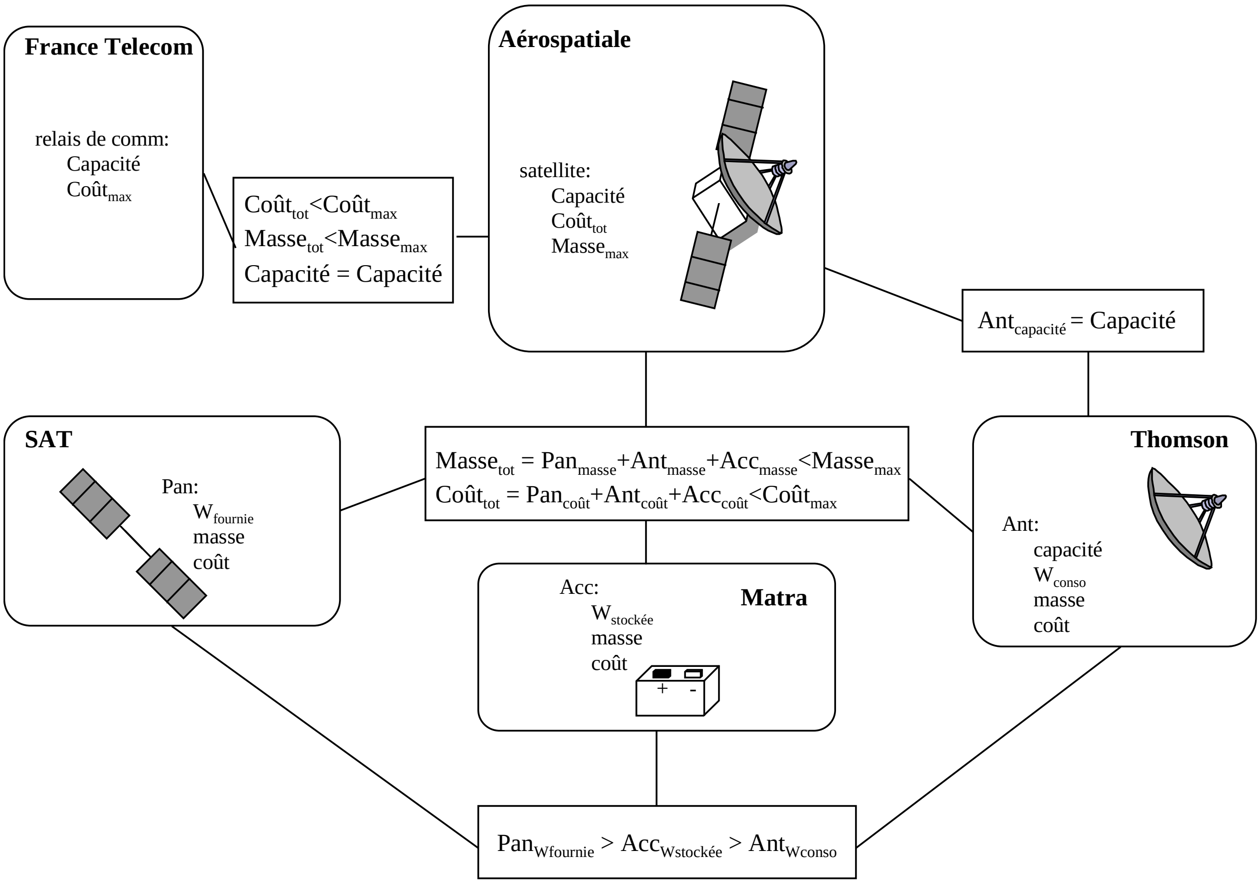


Figure 3. Corrélation des services dans l'exemple de la conception coopérative de satellite.

La prise en compte de ces corrélations impose une étroite coopération entre les membres de l'EV, afin de produire des composants mutuellement compatibles. C'est en fait le principal problème lié à la conception coopérative de satellite. En effet, le problème n'est pas de déterminer comment un intervenant s'y prend pour concevoir le composant qu'on lui a sous-traité (par exemple, comment concevoir une antenne parabolique qui vérifie un certain nombre des caractéristiques), mais de déterminer les caractéristiques précises du composant en tenant compte des corrélations avec les autres composants. Ce problème est d'autant plus difficile à résoudre, que :

- les caractéristiques précises des composants ne sont évidemment pas connues au début du projet. Elles sont déterminées progressivement au cours du projet.
- même une fois fixées, certaines caractéristiques peuvent être modifiées plusieurs fois au cours du projet. C'est le cas quand un intervenant s'aperçoit qu'il ne peut réaliser le composant avec

les caractéristiques prévues dans le contrat initial, et qu'il demande donc au maître d'oeuvre de « relâcher » les contraintes portant sur ce composant. En outre, le client France Telecom peut décider, en cours de projet, de modifier le cahier des charges initial : augmenter la capacité du satellite, ou sa durée de vie, etc. Ces modifications ont bien sûr des conséquences sur tous les composants du satellite.

L'indétermination portant sur les paramètres caractéristiques des composants, imprécision et incertitude sur leur valeurs précises, nécessite une grande flexibilité de l'organisation de l'EV. En effet, les capacités d'adaptation dynamique de l'organisation permettent à l'EV :

- de se constituer rapidement, pour répondre au plus vite et au mieux, à l'appel d'offres du client France Telecom.
- de réagir rapidement à toute modification imprévue dans les caractéristiques du satellite ou d'un de ses composants.
- d'intégrer facilement en cours de projet, de nouveaux intervenants qui apportent de nouveaux savoir-faire et/ou de nouvelles technologies.

2.3 Un SCDO pour « l'Agence de Voyage »

2.3.1 Présentation de l'exemple de l'Agence de Voyage

L'exemple d'Agence de Voyage sert de référence tout au long de ce mémoire. C'est un exemple simple, intelligible, et pourtant très représentatif des SCDOs. Ce paragraphe en fait une présentation générale. Le système est détaillé par la suite en §2.4.

Le système regroupe plusieurs prestataires de services de l'industrie du tourisme, dispersés géographiquement et dont les systèmes d'information sont reliés par un réseau de communication (Internet par exemple). Il a pour fonction de fournir des services de voyages « clés en mains » : train + avion, ou train + avion + hôtel + location de voiture + réservation de salle de conférence par exemple. Les services « élémentaires » qui composent un voyage sont corrélés par des contraintes de compatibilité horaire aux escales, des contraintes de budget, etc.

Par exemple, le système précis étudié dans le cadre de cette thèse (décrit au §2.4) implique une agence de voyage[4] (AgVoy) et ceux de compagnies de transport (SNCF, AF, TWA). L'agenda électronique de M. Dupond est un client qui émet une requête concernant un transport particulier. Le système Agence de Voyage est illustré par la Figure 4.

---

[4] Pour simplifier la présentation, nous assimilons les membres de l'EV à leur systèmes d'information. Par exemple nous parlons de « l'agence de voyage » au lieu du « système d'information de l'agence de voyage ».

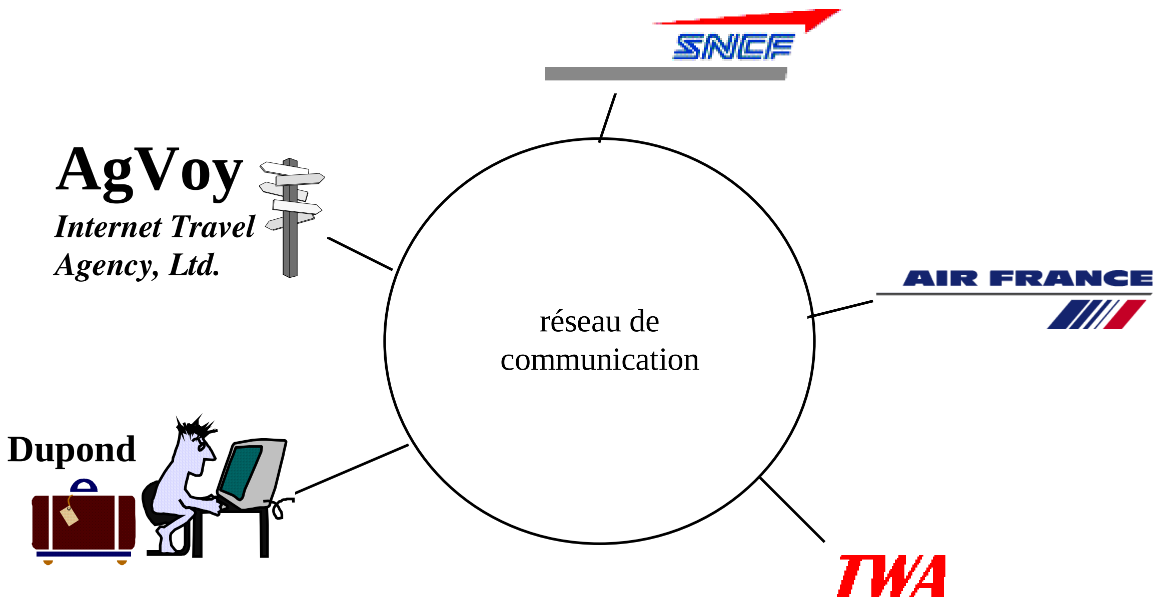


Figure 4. Le système « Agence de Voyage ».

L'agenda électronique gère l'emploi du temps de Dupond et planifie les voyages de Dupond en achetant des services.
A l'exception de l'agence de voyage, les prestataires de services (plus précisément, leur systèmes d'information) fournissent des services qui correspondent à l'exploitation des ressources propres de la compagnie —avions, trains, chambre d'hôtel, etc.—.

L'agence de voyage fournit des services de voyage. Cependant, contrairement aux autres prestataires de services, elle ne possède elle-même aucune ressource matérielle (avions, trains, etc.). Son expertise porte sur *l'organisation* de services qui composent un voyage. Elle intervient ainsi comme intermédiaire entre les clients et les autres prestataires de services :

- elle assiste les clients pour les aider à formuler leur requête (le cahier des charges portant sur le service recherché).
- elle crée une EV pour satisfaire la requête du client. Elle décompose le service en un ensemble de services « élémentaires » : transport, réservation de chambres d'hôtel, etc. Puis elle identifie des prestataires compétents, auxquels elle sous-traite la fourniture de ces services. L'EV est constituée de l'ensemble des prestataires sélectionnés plus l'agence de voyage elle-même.
- elle facilite la coopération des prestataires, pour fournir des services mutuellement compatibles en regard des contraintes qui les lient du fait de leur corrélation.

Une forte indétermination caractérise les conditions d'exploitation de ce système. En particulier une grande incertitude porte sur les horaires des différents trajets, car le client peut retarder son voyage, prolonger ou annuler une escale, etc. De plus, une compagnie de transport peut retarder ou annuler son vol ou son train. Dans ces conditions, il est nécessaire que l'organisation de l'EV s'adapte dynamiquement.

2.3.2 Comparaison avec l'« Assistant Personnel de Voyage » de la FIPA

Il est à noter qu'un scénario d'application très proche de l'Agence de Voyage a été choisi par la FIPA (*Foundation for Intelligent and Physical Agents*, regroupant des centre de recherches universitaires et industriels, avec l'objectif de définir des concepts et des technologies « standards » dans le domaines des agents et des systèmes multi-agents). L'« Assistant Personnel de Voyage » (*Personal Travel Assistant*). [FIPA 97] définit la spécification de ce système.

L'Assistant Personnel de Voyage travaille pour le compte d'agents qui représentent les intérêts d'utilisateurs humains (des « agents-secrétaires »). Il les assiste dans la planification préalable de voyages, ainsi que durant le déroulement de ces voyages.
D'après [FIPA 97], « ces agents interagiront avec l'utilisateur et avec d'autres agents représentant les services de voyages disponibles. Le système multi-agents est chargé de la configuration et de la fourniture — au bon moment, avec une bonne qualité de service et dans des conditions de sécurité et de confidentialité appropriées — d'un service de voyage et des services attenants (réservation de nuits à l'hôtel et de places de parking, service de cartographie, guidage personnalisé dans le trafic autoroutier, informations touristiques, réservation d'avions, guidage dans le métro, conditions météorologiques, événements particuliers, etc.). Par ailleurs, des interactions auront lieu avec d'autres agents, fournissant des services de médiation (pages blanches et pages jaunes), et des courtiers en information qui recherchent, évaluent et raisonnent sur l'information. »

L'objectif de la FIPA est de vérifier, sur un exemple réel et opérationnel, l'intérêt des concepts et des technologies dans les domaines agents et multi-agents.

Le problème de l'Assistant Personnel de Voyage est d'obtenir un ensemble de (réservation de) services qui, composés ensemble, réalisent un voyage « complet » (avec séjour à l'hôtel, location de voiture, etc.). Ce problème est très semblable à celui de l'agent AgVoy dans notre application. Ce qui justifie :

1. l'intérêt de la problématique de cette thèse
2. l'utilisation de l'application Agence de Voyage comme illustration de cette problématique

Le projet d'Assistant Personnel de Voyage est actuellement en phase de spécification. Un premier démonstrateur devrait être prêt en 1998. Il ne validera cependant qu'une partie des spécifications de l'Assistant Personnel de Voyage.

2.4 Le système de l'Agence de Voyage

Cette section décrit précisément le système d'Agence de Voyage utilisé comme référence dans cette thèse. En particulier sont énumérées les hypothèses visant à simplifier la présentation détaillée du système. Un scénario d'utilisation est également présenté pour illustrer le fonctionnement du système.

2.4.1 Composition du système.

Le système contient les agents Dupond, AgVoy et les agents de compagnies de transport précédemment cités (§2.3.1). La Figure 4 en donne une illustration.

Les agents représentant de compagnies de transport proposent une liste de services (réservations de billets TGV ou vols transatlantiques. Le §2.4.2 précise la nature exacte des services de type transport). Ces services constituent leurs compétences et sont listés au Tableau 1.

	SNCF	**SNCF**	**SNCF**	**AirFrance**	**AirFrance**	**TWA**
TRANSPORT	s_{SNCF1}	s_{SNCF2}	s_{SNCF3}	s_{AF1}	s_{AF2}	s_{TWA}
lieuDep	Dijon	Dijon	Dijon	Paris	Paris	Paris
lieuDest	Paris	Paris	Paris	NY	NY	NY
dateDep	16h30	18h30	20h30	19h00	21h00	21h00
dateDest	18h00	20h00	22h00	01h00	03h00	03h00
prix	320	320	320	5000	5100	5200

Tableau 1. Compétences des agents SNCF, AirFrance et TWA.

Note : il est évident que dans la réalité, les compagnies de transport proposent un plus grand nombre de services. Néanmoins, pour des raisons de clarté, le Tableau 1 ne liste que les services qui sont susceptibles d'être utilisés dans le scénario présenté ultérieurement dans ce mémoire.

2.4.2 Simplification du système.

Par rapport au système décrit en §2.3, plusieurs simplifications sont faites pour rendre la présentation plus claire. Ces simplifications ne limitent pas la portée du travail réalisé.

2.4.2.1 Un seul type de services : le service TRANSPORT.

Seule la fourniture de services de transport est étudiée. Les autres types de services sont négligés (hôtel, location de voiture, etc.).

Un service de transport est caractérisé par un ensemble de paramètres :

- lieu et date de départ, notés respectivement « lieuDep » et « dateDep ».
- lieu et date d'arrivée, notés respectivement « lieuDest » et « dateDest ».
- prix, noté « prix ».

Chaque service est décrit sous la forme d'un « tuple » qui indique les valeurs de ses paramètres : « TRANSPORT, lieuDep, dateDep, lieuDest, dateDest, prix ».

Un service de transport peut résulter de la composition de plusieurs services de transport (ces derniers sont alors appelés « services élémentaires ») : c'est le cas lorsque des escales ont lieu.

Les corrélations des services de transport élémentaires sont représentées par un réseau de contraintes qui lient les paramètres de ces services (cf. §2.4.3.2).

Les services de transport « élémentaires » sont fournis par des compagnies de transport : compagnies aériennes (Airfrance, TWA), ou ferroviaires (SNCF). Ils correspondent à l'exploitation de ressources propres de l'entreprise —avion ou train—. L'agence de voyage fournit des service de transport qui sont composés des services élémentaires fournis par les compagnies de transport.

Un ensemble de services de transport peut être décrit par *intension*, i.e. par la donnée de domaines de valeurs pour les paramètres des services qu'il contient (i.e. par la donnée d'une contrainte que les valeurs doivent vérifier, exemple « prix≤10000fr »). Cette possibilité est exploitée dans les requêtes de clients qui spécifient un cahier de charges au lieu d'un service précis : le cahier des charges définit *l'ensemble des services acceptables par le client*, et le client demande au prestataire de fournir l'un quelconque des services appartenant à cet ensemble. Le §2.4.3.1 donne un exemple de cahier des charges.

2.4.2.2 Seule la planification du transport est envisagée.

Seule la planification du transport est envisagée. La supervision de son exécution n'est pas considérée dans le système.
De ce fait, **les services proposés sont la *réservation* de billets de transport** : places sur des vols aériens et places dans un train. La fourniture de ces services représente la réservation effective d'une place sur un vol ou un train particulier, avec horaires et lieu de départ et d'arrivé fixés.
Lorsque AirFrance, TWA ou SNCF fournissent des services à AgVoy, il s'agit de places « bloquées » sur un vol ou un TGV particulier. De même AgVoy fournit une *réservation* à Dupond. L'exécution du transport à proprement parler, n'est pas modélisée. Par la suite, on parlera de « service de transport » au lieu de dire « service de réservation de billets de transport ».

2.4.2.3 Les réservations sont toujours possibles.

Toujours pour des raisons de simplicité de présentation, nous n'envisageons pas la possibilité qu'un vol (ou un TGV) soit complet, empêchant par là un client de faire une réservation. Ce choix facilite grandement le gestion des appels d'offres émis par le client : **les réponses des serveurs de compagnies de transport aux appels d'offres d'un client, restent valables indéfiniment**.
La prise en compte de la possibilité d'un vol (ou d'un train) plein, ne présente pas de difficulté majeure dans notre travail. Elle en complique seulement la description.

2.4.3 Analyse d’une requête client.

Une requête typique est présentée ici : elle porte sur la fourniture d’un service de transport de Dijon à NewYork. Une EV est créé pour la satisfaire. L’analyse de cette requête permet d’identifier clairement les caractéristiques des SCDOs et des EVs.

L'agenda (Dupond) doit planifier une réunion de travail à NewYork pour le compte de son propriétaire humain, M. Dupond, qui vit à Dijon. Pour ce faire, l’agent Dupond a besoin de réserver un transport Dijon-NewYork avec un cahier des charges particulier.

La réservation de ce voyage se déroule en plusieurs étapes :

1. l’émission d’une requête par Dupond à AgVoy, concernant le service de transport de Dijon à NewYork (noté en abrégé « TRANSPORT Dijon NY »).
2. la décomposition par AgVoy du service demandé en un ensemble de services « élémentaires » : « TRANSPORT Dijon Paris » et « TRANSPORT Paris NY ».
3. la création par AgVoy, d’une EV pour satisfaire la requête de Dupond. Les membres de l’EV fournissent les services élémentaires identifiés par AgVoy.
4. l’adaptation dynamique de cette EV à des événements imprévus, et notamment leur réaction à la défaillance de AirFrance dont le vol « Paris NY » est annulé.
5. dissolution de l’EV lorsque son existence ne se justifie plus (Dupond commence son voyage).

Ces différentes étapes sont décrites dans ce qui suit, et elles sont illustrées par la Figure 5.

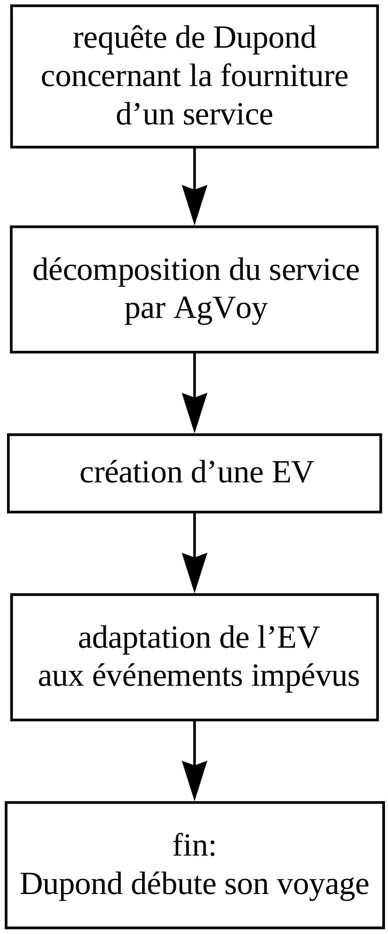


Figure 5. Etapes successives du traitement d’une requête client.

2.4.3.1 Requête de Dupond à AgVoy.

Dupond émet dans le système une requête avec le cahier des charges suivant (cf. Figure 6) :
« TRANSPORT, lieuDep=Dijon, dateDep>12juin16h00, lieuDest=NY, dateDest<13juin08h00, prix<7KF »

Ce cahier des charges décrit les dates limites de départ et d’arrivée ainsi qu’un prix maximum pour le service demandé par Dupond à AgVoy. AgVoy peut fournir n’importe quel service (i.e. n’importe quelle réservation sur pour un transport de Dijon à NewYork) respectant ces contraintes.

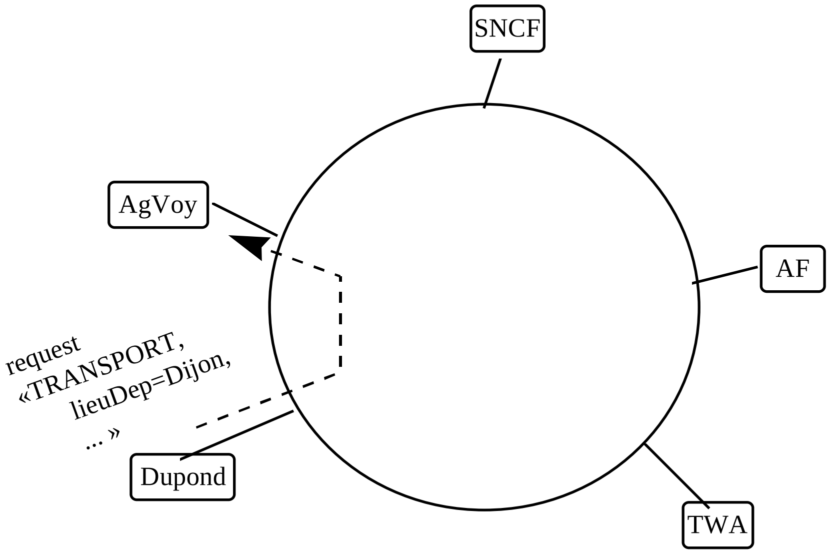


Figure 6. Emission d’une requête par Dupond.

2.4.3.2 Décomposition du service.

En utilisant son expertise propre, l’agence de voyage décompose le voyage en un ensemble de trajets qui correspondent à autant de services TRANSPORT.

En l’occurrence, AgVoy détermine qu’une escale à Paris est nécessaire[5]. Le service demandé par Dupond est donc décomposé en deux services de type TRANSPORT s1 et s2. s1 et s2 sont liés par une relation de compatibilité, représentées par un réseau de contraintes portant sur les paramètres de ces deux services

[5] Le choix d’une escale à Paris est décidé par AgVoy. Le choix de Paris est réaliste compte tenu de la topologie du réseau de transport français et de la position centrale de Paris. AgVoy pourrait rechercher des services « TRANSPORT Dijon * » et « TRANSPORT * NewYork », où * représente une escale indéterminée. Paris, Lyon, ou Genève pourraient être identifiés comme escales possibles. Bien que le mécanisme d’interaction proposé dans cette thèse permette une telle recherche, le choix a priori de Paris en simplifie la description.

Cette relation traduit notamment (cf. les deux dernières contraintes) les problèmes d'horaire à l'escale et une contrainte sur le prix total[6] :
R = (s1.type=TRANSPORT)
∧ (s1.lieuDep = Dijon)
∧ (s1.dateDep>12juin16h00)
∧ (s1.lieuDest=Paris)
∧ (s2.type=TRANSPORT)
∧ (s2.lieuDep=Paris)
∧ (s2.lieuDest=NY)
∧ (s2.dateDest<13juin08h00)
∧ (s1.dateDest<s2.dateDep)
∧ (s1.prix+s2.prix<7KF)

On peut remarquer que la relation de compatibilité contient des contraintes unaires, i.e. ne portant que sur un des deux services. Ces contraintes sont utilisées par la suite pour définir des cahiers des charges dans les appels d'offres diffusés dans le système afin d'identifier les agents serveurs compétents pour fournir s1 ou s2.

Cette décomposition donne à AgVoy l'ensemble des services à sous-traiter à des compagnies de transport. La corrélation entre ces services (définie par le réseau de contrainte, cf. Figure 7) justifie la coopération des compagnies impliquées : elle est nécessaire pour fournir des services de transport respectant ces contraintes d'horaires et de prix.

[6] La notation s.n signifie « l'attribut de nom n du service s ». Par exemple « s1.dateDépart » représente la date de départ du service s1.

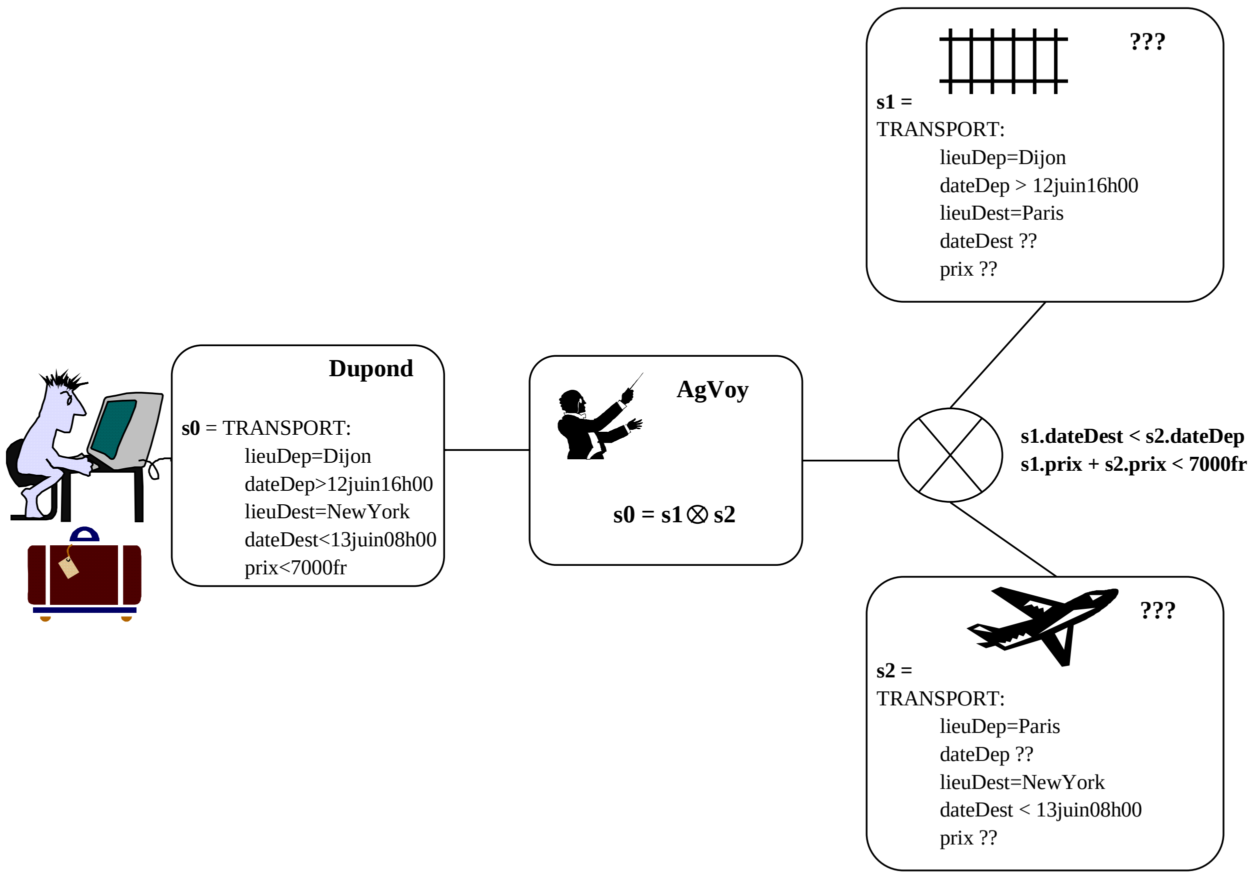


Figure 7. Corrélation entre services.

2.4.3.3 Création d'une EV.

Par la diffusion d'appels d'offres, AgVoy identifie les agents présents dans le système qui sont compétents pour fournir les services s1 ou s2 [7]. Cela lui permet de déterminer les organisations possibles de l'EV.

L'organisation (§3.3) définit les rôles de chaque membre de l'EV. En pratique, les un rôle correspond à la fourniture d'un service de transport précis (sélectionné les compétences des agents de compagnies de transport, cf. Tableau 1).

Chaque organisation possible correspond ainsi à un regroupement de compagnies de transport dont les services respectifs sont mutuellement compatibles (en regard de la relation de compatibilité R). Ces regroupements sont listés au Tableau 2.

[7] Il faut noter que la gestion des procédures d'appels d'offres est rendue complexe du fait de la corrélation des services recherchés.

groupes	serveur pour s1	service pour s1	serveur pour s2	service pour s2
((s1, s_{SNCF1}), (s2, s_{AF1}))	SNCF	s_{SNCF1}	AirFrance	s_{AF1}
((s1, s_{SNCF1}), (s2, s_{AF2}))	SNCF	s_{SNCF1}	AirFrance	s_{AF2}
((s1, s_{SNCF1}), (s2, s_{TWA}))	SNCF	s_{SNCF1}	TWA	s_{TWA}
((s1, s_{SNCF2}), (s2, s_{AF1}))	SNCF	s_{SNCF2}	AirFrance	s_{AF1}
((s1, s_{SNCF2}), (s2, s_{AF2}))	SNCF	s_{SNCF2}	AirFrance	s_{AF2}
((s1, s_{SNCF2}), (s2, s_{TWA}))	SNCF	s_{SNCF2}	TWA	s_{TWA}

Tableau 2. Les regroupements de serveurs mutuellement compatibles.

Parmi toues les organisations possibles, AgVoy en choisit une en utilisant une « fonction de préférence ». En l'occurrence, il choisit la moins chère, i.e. celle correspondant au regroupement dont la somme des prix des services est minimale). D'après le Tableau 1, la moins chère est ((s1, s_{SNCF1}), (s2, s_{AF1})).

L'EV correspondante est alors créée : elle est composée, en plus de AgVoy, de SNCF —dont le rôle est la fourniture de s_{SNCF1}— et de AirFrance—dont le rôle est la fourniture de s_{AF1}—. La Figure 8 en donne l'illustration.

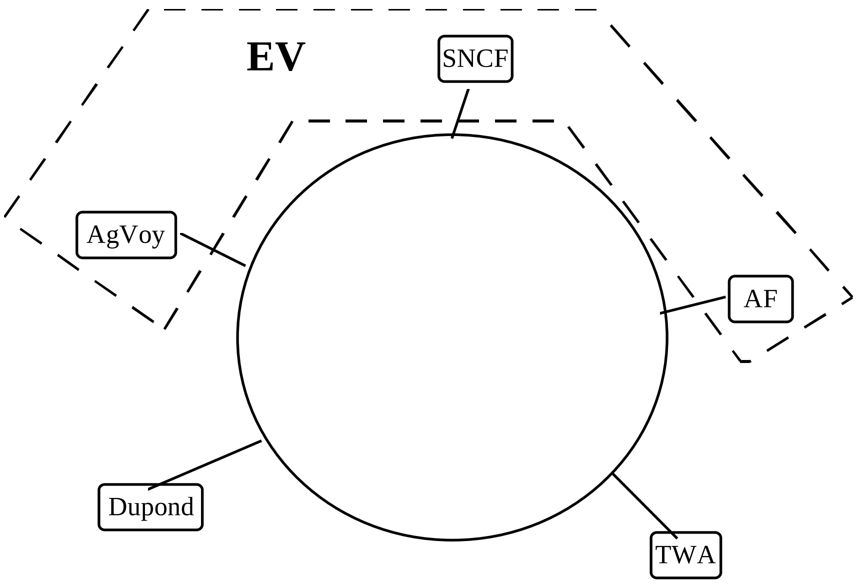


Figure 8. EV formée pour satisfaire la requête de Dupond

2.4.3.4 Adapter l'organisation de l'EV aux imprévus.

D'importantes incertitudes caractérisent les services de transport : annulation ou retard de vols ou de trains dus à des grèves ou des pannes, etc. De tels événements sont susceptibles de causer la défaillance d'un des membres de l'EV, en l'empêchant de fournir le service correspondant à sont rôle.

L'EV adapte dynamiquement son organisation pour pallier à une telle situation, et assurer à Dupond une certaine « continuité de service ».

Par exemple si une grève survient à AirFrance et provoque l'annulation du vol transatlantique, alors les rôles de AirFrance et de SNCF sont redéfinis : les réservations de Dupond sont finalement transférées sur un autre train et un autre vol.
Dans certains cas —par exemple un grève totale et sans limite de AirFrance—, les changements de l'organisation peuvent conduire à exclure la compagnie défaillante et à la remplacer par une autre (par exemple remplacer AirFrance par TWA).
Il est important de noter l'importance de la corrélation entre service : un événement local —la défaillance d'un membre—, peut avoir des conséquences globales sur l'EV, en l'occurrence sa réorganisation complète.

2.4.3.5 Dissolution finale de l'EV.

L'EV est dissoute :

- lorsque Dupond débute son voyage à Dijon[8],
- OU lorsque Dupond annule sa réservation (si sa réunion à NewYork est annulée),
- OU enfin si l'EV ne peut s'adapter à un changement, par exemple la disparition de SNCF qui supprime toute possibilité d'aller de Dijon à Paris (aucune autre compagnie de transport de dessert Paris depuis Dijon, dans ce scénario). Dans ce cas, l'EV se dissout sur un échec car elle n'a pu satisfaire la requête de Dupond.

2.5 Conclusion.

Une requête typique a été présentée. Elle constitue un scénario typique d'utilisation du système de l'Agence de Voyage. Il en sera fait référence tout au long de ce mémoire.

L'analyse de cette requête permet d'identifier deux caractéristiques souhaitables de l'organisation d'une EV :

- l'organisation est dynamique pour s'adapter aux changements qui sont susceptibles de survenir à tout moment, à cause de la forte indétermination caractéristique des conditions d'exploitation des EVs.
- l'organisation est décentralisée pour respecter l'autonomie de chaque entreprise membre de l'EV, et également pour minimiser les communication longues distances —particulièrement coûteuses dans ces conditions d'exploitation—.

La principale fonction d'un SCDO est de gérer l'organisation de l'EV : sa formation, mais aussi —et surtout— son adaptation dynamique en cas de défaillance d'un des membres de l'EV. Un SCDO permet d'apporter aux EVs :

- réactivité, i.e. des temps de réponse et de prise de décision réduits

[8] Le rôle de l'EV se limite en effet à la planification du voyage. Nous ne traitons pas de *son exécution*.

- adaptabilité, qui permet le maintien, au mieux l'amélioration, et au pire la lente dégradation des performances du système lorsque les besoins de l'entreprise évoluent, ou lorsque les conditions d'exploitation changent (pannes, etc.).

Ces qualités sont indispensables aux entreprises, virtuelles ou non, qui souhaitent développer, fournir et maintenir des services à une vitesse, avec une flexibilité, une qualité et à un coût compatibles avec les exigences toujours grandissantes du marché.

La principale difficulté de la gestion de l'organisation d'une EV vient de la corrélation des rôles de ces membres de l'EV : les services qu'ils fournissent sont mutuellement contraints par un réseau de contraintes. La corrélation rend délicate la détermination d'une organisation car il faut identifier des rôles mutuellement compatibles. De plus, la défaillance d'un membre de l'EV peut conduire à une réorganisation complète de l'EV, afin de préserver la corrélation des services fournis.

La corrélation des services est en fait la raison de la formation de l'EV : sa prise en compte nécessite la coopération étroite des entreprises qui fournissent les services. Et c'est pour mieux coopérer qu'elles se regroupent au sein d'une EV.

Dans cette perspective, les concepts Multi-Agents sont particulièrement utiles (autonomie, décentralisation, coopération, etc.). En fait, un SCDO peut être considéré comme un SMA, dont les agents sont les membres de l'EV (ou plus précisément, leurs systèmes d'information).

3. Le Problème de l’Adaptation Dynamique de l’Organisation

Ce chapitre présente le problème traité dans ce travail de thèse, en le situant clairement dans le domaine d’étude des Systèmes Multi-Agents.

Tout d’abord, l’analyse du scénario de l’Agence de Voyage permet de confronter deux propriétés caractéristiques des agents : l’autonomie de contrôle de chaque agent d’une part, et d’autre part les relations de dépendance qui lient les agents les uns aux autres.
Les groupes d’agents, qui constituent l’organisation du SMA, sont ensuite introduits à partir du constat de la corrélation des liens de dépendance entre agents.
Le problème, qui découle de cette confrontation autonomie de contrôle/dépendance envers les autres agents, est ensuite décrit : c’est celui de l’adaptation dynamique de l’organisation du SMA. L’intérêt de ce problème est expliqué ensuite.

3.1 Autonomie de contrôle

Un agent est une entité autonome : il dispose de son propre mécanisme de contrôle, qui détermine les actions qu’il entreprend ([Ferber 95]). On parle de « contrôle local ». Cette propriété peut s’exprimer par le fait que « l’agent détermine en son âme et conscience les actions qu’il entreprend afin de remplir sa fonction ».

Dans le cas des agents-logiciels[9], les actions entreprises correspondent à la fourniture de services, et le contrôle local est relié au module de communication de l’agent, afin de recevoir/émettre des messages aux autres agents du système. En particulier il reçoit les requêtes provenant des autres agents, et il les traite en décidant ou en refusant de fournir le service demandé. La Figure 9 illustre la structure de l’agent et souligne la position centrale du mécanisme de contrôle local (décrit dans la partie état de l’art §4.1).

---

[9] Dans le cadre de cette thèse, nous ne nous intéressons qu’aux SMAs de type SCDOs dont les agents sont des programmes informatiques reliés par un réseau de communication. D’autres types de SMAs existent, par exemple les système de robots footballeurs ou les fourmilières. Leurs agents peuvent avoir d’autres types de capteurs et d’effecteurs.

Dans le cas de L'Agence de Voyage, le contrôle local met en oeuvre une politique de type « commerciale » : à chaque service est associé un « prix ». Pour que l'agent serveur accepte une requête émise par un client (et donc qu'il décide de fournir le service demandé par le client), il faut et il suffit que le client paye le serveur d'un montant égal au prix du service demandé[10], comme illustré sur la Figure 9. Pour cette raison, on qualifiera ces agents « d'agents commerciaux ».

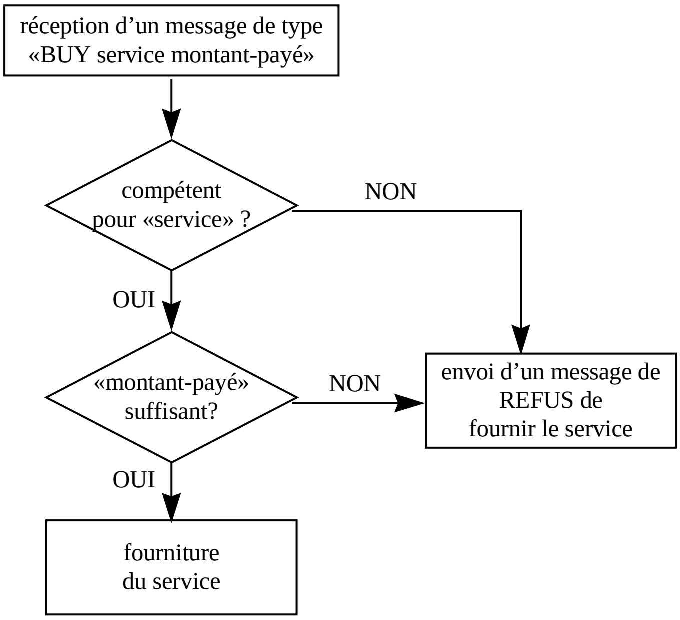


Figure 9. Réaction à la réception d'une requête par un agent commercial.

3.2 Dépendance inter-agents

Au sein d'un SMA, les agents sont liés les uns aux autres par des relations de dépendance, qui correspondent aux relations client/serveur identifiées entre l'agence de voyage et les compagnies de transport au §2.3.1 : à certains services sont associées des conditions que l'environnement de l'agent doit vérifier pour que le service s'exécute correctement. Ces conditions définissent les besoins de l'agent concernant la fourniture du service. Ces besoins sont satisfaits par d'autres agents du système. Dans le reste du chapitre, l'agent dépendant sera appelé agent client[11], et l'agent dont il dépend sera appelé agent serveur. Les relations de

---

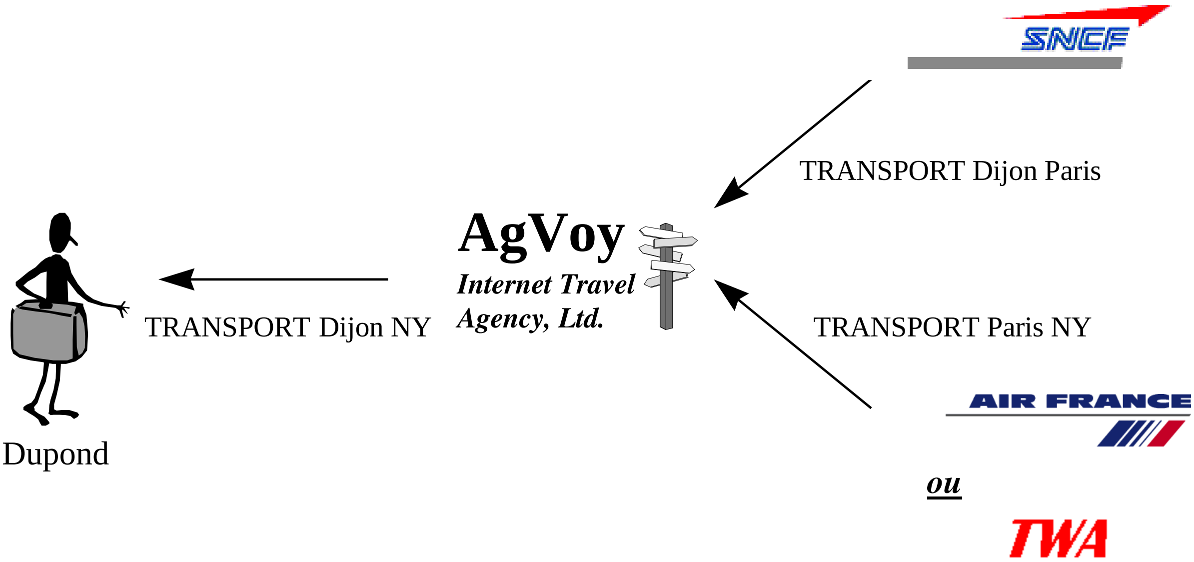
[10] Dans le reste du mémoire, et en particulier dans l'implémentation informatique de l'Agence de Voyage, l'acte de paiement est ignoré pour des raisons de simplicité.

[11] L'agent est client dans le cadre de cette relation, mais il est également serveur, puisque ses besoins sont associés à la fourniture par lui-même d'un service. Cette situation est illustrée par l'agent AgVoy dans l'exemple de l'Agence de Voyage.

dépendance traduisent la relation entre chaque agent et son « environnement » [12] : l'agent ne peut être défini indépendamment de son environnement car sa capacité à fournir ses propres services dépend des services fournis par d'autres agents.

Dans l'exemple de l'Agence de Voyage, l'agent AgVoy ne dispose pas des ressources nécessaires à la fourniture de ses propres services (trains, avions, etc.). Il est de ce fait dépendant des agents qui disposent de ces ressources : AgVoy ne peut réellement agir (fournir ses services de transport) que si d'autres agents fournissent des services de transport. Comme remarqué au paragraphe précédent (en note de page), l'agent AgVoy est donc à la fois serveur —de ses propres clients, comme Dupond— et client —des agents représentant les compagnies de transport—.

La Figure 10 rappelle les liens de dépendance entre les agents du système Agence de Voyage.



«TRANSPORT Dijon NY» = «TRANSPORT Dijon Paris» + «TRANSPORT Paris NY»

Figure 10. Dépendances de l'agent AgVoy pour fournir un de ses services.

Le cas de l'agent AgVoy montre l'importance de la prise en compte par le contrôle local de l'agent, des relations de dépendance qui le lient aux autres agents du système. L'état de l'art présente le modèle de contrôle local de Castelfranchi (§4.1.3), qui prend effectivement en compte les relations de dépendance inter-agents.

[12] Nous ne tenons pas compte ici de l'environnement d'exécution de l'agent (comme par exemple l'environnement physique pour un agent robot, ou un réseau d'ordinateurs pour un agent-logiciel). Le terme « environnement » se rapporte uniquement aux autres agents du système. Cette notion d'environnement est parfois appelée « co-agent ».

Il est important de noter la corrélation des différents liens de dépendance. Dans le cas de AgVoy, les services de transports sont liés par un ensemble de contraintes :

- les contraintes d'horaire aux escales : M. Dupond doit *arriver* à une escale —par exemple Paris—, *avant d'en repartir*. Cela se traduit par une contrainte liant les deux services de transport, et portant plus précisément sur la date d'arrivée du premier et la date de départ du second.
- contrainte de rentabilité pour AgVoy: pour faire un bénéfice sur la fourniture du service de transport, la somme des prix des services de transport doit être inférieure au prix du service de transport —celui que paye effectivement Dupond—. La différence constitue la marge de AgVoy.
- d'autres contraintes peuvent exister, mais nous n'en tiendrons pas compte.

Cet ensemble définit un réseau de contraintes dont les variables sont les différents services de transport (il est illustré par la **Figure 7** page 19). Le réseau définit la relation de compatibilité qui lie les différents services : pour que l'agent AgVoy puisse effectivement fournir son propre service de transport, il est nécessaire que :

- les services de transport soient fournis par des agents dans le SMA. ET
- ces services soient mutuellement compatibles en regard de la relation de compatibilité.

Remarque : le cas des agents AirFrance, TWA et SNCF est particulier, car ils disposent de ressources propres (avions, trains) qui les rendent indépendants de leur environnement. C'est toutefois une hypothèse très simplificatrice : en réalité les compagnies aériennes assurent fréquemment des vols sur des avions qui ne leur appartiennent pas, et qu'elles louent à des sociétés spécialisées ou à d'autres compagnies aériennes. Des mécanismes similaires existent pour les services ferroviaires dans les pays où ces services sont privatisés et « déréglementés », comme au Royaume-Uni par exemple. Dans un souci de simplicité et de clarté, les agents représentant les compagnies de transports sont donc considérés comme indépendants de leur environnement.

3.3 Regroupements d'agents serveurs et organisation du SMA

3.3.1 Regroupement d'agents serveurs = solution du réseau de contraintes

Chaque solution au réseau de contraintes définit un ensemble de services précis, i.e. service de transport avec une valeur précise pour chacun des paramètres : date et lieu de départ, d'arrivée, prix, etc. Par souci de clarté, nous supposerons que chaque service est fourni par un agent différent (en réalité, un même agent peut bien sûr fournir plusieurs services). Chaque solution définit de ce fait un groupe d'agents serveurs, chacun ayant un rôle précis dans le groupe : le service qu'il fournit. La structure du groupe (les relations entre les rôles) est donnée par le réseau de contraintes.

La nature des corrélations entre services dans le système « Agence de Voyage », des contraintes portant sur les paramètres des services, permet d'établir l'analogie suivante (Tableau 3) :

Point de vue SCDO	Point de vue « réseau de contraintes »
service recherché par le client	variable
services proposés par les serveurs	valeurs possibles pour la variable
groupes d'agents serveurs	solution du réseau de contraintes
rôle d'un membre du groupe	valeur de la variable correspondante dans la solution
structure du groupe	réseau de contraintes

Tableau 3. Analogie avec le problème de satisfaction de contraintes.

Dans l'exemple de l'Agence de Voyage, les services s_{SNCF1} et s_{AF1} satisfont la relation de compatibilité R de AgVoy, les agents SNCF et AirFrance forment donc un groupe d'agents serveurs, dont AgVoy est le client. D'autres groupes existent dans le système, leur liste est donnée au Tableau 2.

3.3.2 Caractère « vue de l'esprit » des regroupements d'agents serveurs

Il est important de noter que les agents serveurs n'ont a priori pas « conscience » de faire partie d'un groupe, et l'agent client ne connaît pas a priori les groupes d'agents serveurs dont il est client potentiel. Ce point est particulièrement clair dans l'exemple de l'Agence de Voyage :

1. au début du scénario, l'agent AgVoy ne connaît pas nécessairement la présence des agents SNCF, AirFrance, TWA dans le système[13]. Il ne peut donc *a fortiori* pas connaître les services précis qu'ils sont susceptibles de fournir, et donc les groupements potentiels qu'ils forment éventuellement. Une des principales fonctions d'une interaction est justement d'identifier les regroupements d'agents serveurs potentiels, afin d'en activer un par la suite.
2. les agents serveurs ne font partie d'un groupe que parce qu'ils fournissent des services compatibles *du point de vue du client* (i.e. en regard de la relation de compatibilité de AgVoy). Au début du scénario, ils ne peuvent donc pas savoir s'ils font partie d'un (ou de plusieurs) regroupement(s).

Le concept de groupe d'agents serveurs souligne le lien entre l'agent et son environnement, i.e. les autres agents du SMA : si le contrôle local de l'agent détermine de manière autonome quel service fournir (en acceptant de satisfaire le requête correspondante), en revanche la

[13] C'est la conséquence des conditions d'exploitation caractéristiques des SMA de type SCDO : les agents sont susceptibles d'entrer/sortir à tout moment du système, de sorte qu'aucun agent ne peut avoir de vue global et à jour de la composition du SMA. En conséquence, AgVoy n'est pas censé connaître tous les agents représentant des compagnies de transport et qui sont présent dans le système.

capacité à fournir ces services est fonction de son environnement, et plus précisément de l'adaptation de l'organisation du SMA. Ce point fondamental est détaillé au paragraphe suivant (§3.4).

Enfin, il est essentiel de noter que *la corrélation des liens de dépendance est indispensable pour définir des groupes d'agents serveurs* : en l'absence de corrélation —i.e. sans relation de compatibilité—, chaque agent serveur fournirait un service indépendant des autres. Le concept de « regroupements d'agents serveurs » perdrait alors de son intérêt. La prise en compte explicite de la corrélation des liens de dépendance est un « plus » par rapport au modèle de dépendance sociale de Castelfranchi (§4.1.3)

3.4 Interaction pour l'adaptation dynamique de l'organisation

L'interaction est initiée par l'agent client, qui indique les services dont il a besoin et leur corrélation.
La fonction de l'interaction est de provoquer la formation d'un regroupement d'agents serveurs, voire de plusieurs successivement en cas de défaillance de certains serveurs.

Si l'interaction réussit, alors un regroupement d'agent serveurs est formé pour fournir au client les services corrélés dont il a besoin. En revanche si l'interaction échoue, alors l'agent client ne peut obtenir tous les services dont il a besoin. Dans l'exemple de l'Agence de Voyage, l'échec de l'interaction initiée par AgVoy, conclurait à l'incapacité d'AgVoy de fournir un service « TRANSPORT Dijon NewYork » à Dupond.

L'interaction se déroule en plusieurs étapes (Figure 11), qui peuvent être rapprochées des étapes du traitement de la requête de Dupond (§2.4.3) :

1. identification de groupes d'agents serveurs.
2. sélection et activation d'un des groupes identifiés.
3. remplacement du groupe sélectionné par un autre, au cas où un de ses membres est défaillant.

L'interaction cesse quand l'agent client n'a plus besoin des services fournis par les agents serveurs.

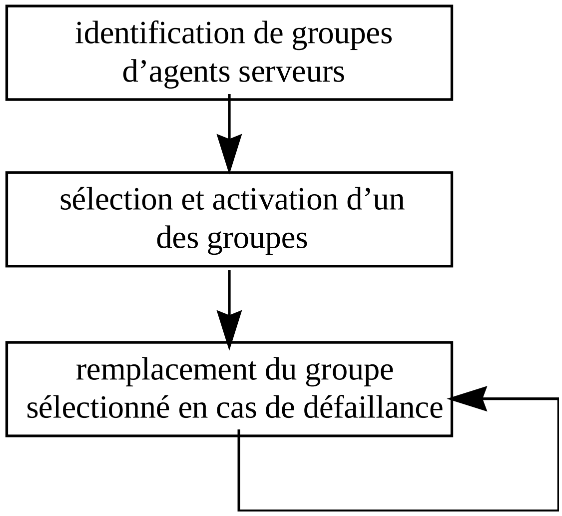


Figure 11. Etapes successives du déroulement d'une interaction

3.5 Quel est le problème, et pourquoi en est-ce un ?

L'organisation d'un SMA évolue en permanence pour différentes raisons :

- l'introduction et le retrait fréquent d'agents du système en modifie la composition et a pour conséquence la création ou la disparition de groupes d'agents serveurs.
- certains agents — c'est le cas de Dupond, agenda électronique de M. Dupond — engendrent spontanément des requêtes (dans le scénario, il s'agit d'un requête pour obtenir un service de transport de Dijon à New York). Ce phénomène s'explique par le fait que M. Dupond « entre » manuellement dans l'agenda une réunion à New York : l'agenda prend alors en charge la préparation de la réunion, et en particulier la planification du transport.

Ces deux facteurs créent une forte indétermination, sur d'une part les groupes d'agents serveurs présents dans le SMA, et d'autre part les requêtes émises par les agents en cours d'exploitation du SMA. Il est clair qu'aucune de ces indétermination ne peut être levée en phase de conception. **Une des tâches du concepteur d'un SMA est donc de construire un mécanisme d'auto-adaptation de l'organisation, qui est ensuite activé en cours d'exploitation du système.** Plus précisément, il est activé par agent client qui souhaite obtenir la fourniture de plusieurs services. Ce mécanisme est en fait une interaction car il permet à l'agent client d'influencer d'autres agents — les serveurs — .

La détermination d'un mécanisme d'interaction *efficace* dans les conditions d'exploitation des SMA de type SCDO, pose deux problèmes majeurs concernant :

- **la relation de compatibilité entre les services fournis par les agents serveurs.** La moindre modification d'un paramètre d'un service peut avoir des conséquences globales, portant sur l'ensemble des services fournis, et donc potentiellement sur l'organisation entière du système. C'est le cas dans le scénario de l'Agence de Voyage, où l'annulation du vol d'AirFrance conduit à la modification du service fourni par SNCF (la réservation TGV de Dupond est transférée sur un autre train).

- **les conditions d'exploitation particulières des SMA de type SCDO.** Ces mécanismes doivent notamment présenter des qualités d'extensibilité et de résistance aux pannes afin de garantir une dégradation acceptable de leur fonctionnement en cas d'augmentation importante de la taille du système (passer de 100 à 10000 agents par exemple) et/ou de la fréquence des pannes. De plus, la dispersion géographique rend les communications distantes coûteuses voire momentanément impossibles en cas de panne de réseau.

3.6 Intérêts de la Problématique

Le problème abordé dans cette thèse est original dans le sens où la plupart des travaux portant sur l'adaptation dans un SMA, traitent de l'adaptation de l'agent au système, et non l'inverse.

3.6.1 Apport à la conception de SMA

3.6.1.1 Les SMA de type SCDO

Un groupe d'agents serveurs représente en fait une organisation possible de l'EV formée pour satisfaire le client : les membres de l'EV sont les compagnies représentées par les agents. De ce point de vue, l'interaction initiée par le client assure la création et l'adaptation de l'EV. Le mécanisme d'interaction est donc un composant clé des systèmes d'informations des EVs, les SCDOs.
En particulier, l'adaptation dynamique de L'EV (réalisée par le remplacement d'un groupe d'agents serveurs par un autre, au cours de la troisième et dernière étape de l'interaction) lui permet d'assurer la continuité de service en cas de défaillances de certains de ses membres. L'adaptation prend plusieurs formes : intégration de nouveaux membres (les nouveaux serveurs membres du groupe de remplacement), départ d'anciens membres (les serveurs membres du groupe défaillant, *et qui ne sont pas membre du groupe de remplacement*), et maintien de certains membres (ceux qui appartiennent aux deux groupes). En outre, l'adaptation concerne également les rôles des membres de l'EV (les services qu'ils fournissent). Les rôles peuvent être modifiés en cas de défaillance du groupe actif (les membres du groupe de remplacement ne fournissent pas nécessairement les mêmes services que ceux du groupe défaillant).

L'adaptation dynamique de l'EV est illustrée sur les figures Figure 12 et Figure 13 (l'EV est définie par le rectangle en pointillé. Sous chaque agent serveur membre de l'EV, est indiqué son rôle, c'est-à-dire le service qu'il fournit au client de l'EV, qui est AgVoy). La Figure 12 montre l'état de l'EV lorsque le groupe actif est constitué de SNCF fournissant s_{SNCF1} et de AirFrance avec s_{AF1}(voir le Tableau 1 pour la description exacte des services de transports proposés par les agents serveurs). Suite à une défaillance de AirFrance (grève des pilotes par exemple), le groupe est désactivé et remplacé par le groupe constitué de SNCF, *mais avec le nouveau rôle* s_{SNCF2}, et de TWA avec s_{TWA} (cf. Figure 13). L'EV s'est adaptée à la défaillance de AirFrance : AirFrance l'a quittée, TWA y est entré, et SNCF est resté mais a changé de rôle. Il est à noter que TWA n'a pas « remplacé » simplement AirFrance, car le service fourni par TWA n'est pas identique à celui qui était fourni par AirFrance.

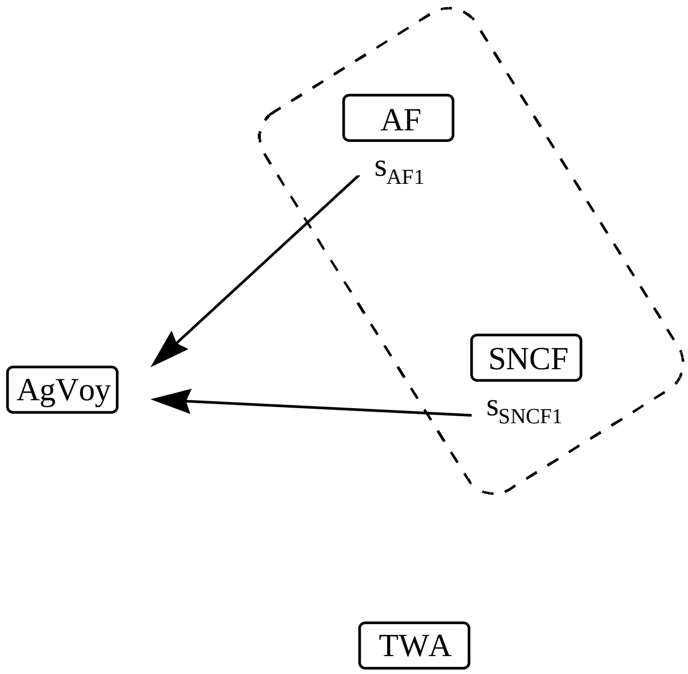


Figure 12. Etat de l'EV avant défaillance de AirFrance.

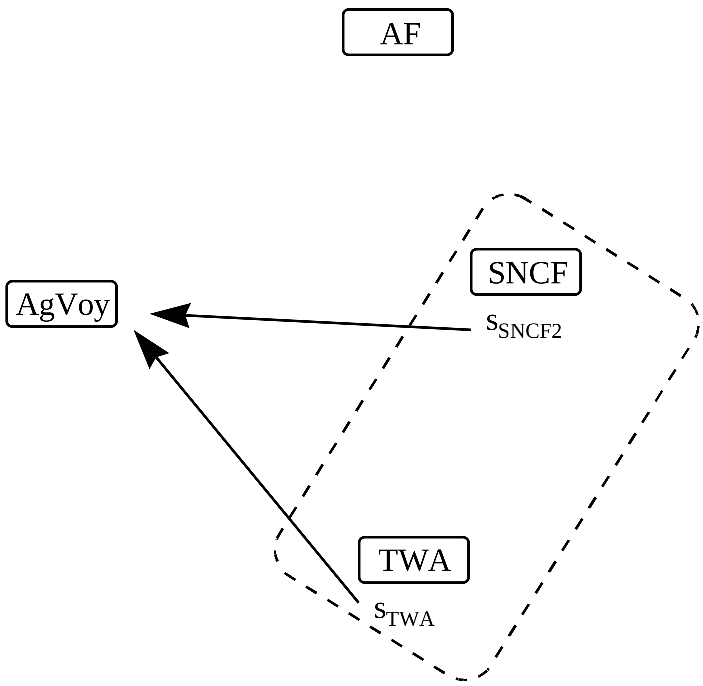


Figure 13. Etat de l'EV après adaptation à la défaillance de AirFrance.

3.6.1.2 Extension à d'autres types de SMA

Le problème de l'adaptation dynamique de l'organisation est l'un des principaux problèmes auquel le concepteur d'un système multi-agents doit faire face, quel que soit le type du SMA qu'il conçoit.
La fourniture d'un mécanisme d'interaction permettrait au concepteur d'un SMA de se concentrer sur d'autres problèmes liés à la conception d'un SMA, et en particulier ceux cités ci-après (§3.6.2). Ce travail de thèse porte exclusivement sur les SMA de type SCDO, mais le mécanisme d'interaction proposé peut être utilisée pour d'autres types de SMA (des équipes de robots par exemple).

3.6.2 Limitations de notre apport à la conception de SMA

3.6.2.1 La relation de compatibilité est prédéfinie

Dans notre travail, nous supposons que la relation de compatibilité est prédéterminée et connue du client. En d'autres termes, le mécanisme d'interaction résout un problème de satisfaction de contraintes prédéfini, dans la mesure où le réseau de contraintes est connu a priori par AgVoy.

Dans le scénario de l'Agence de Voyage, l'agent AgVoy connaît les contraintes liant les deux services « TRANSPORT Dijon Paris » et « TRANSPORT Paris NewYork » dont il a besoin. Cette hypothèse signifie que AgVoy sait à l'avance qu'il faut décomposer un transport « Dijon NewYork » en deux trajets, et donc qu'une unique escale suffit.

Ce choix est peu réaliste, mais il permet de simplifier le mécanisme d'interaction qui est développé dans cette thèse. Cependant il est possible de s'affranchir de cette hypothèse. Pour ce faire, l'interaction doit d'abord définir un réseau de contrainte, puis ensuite seulement essayer de le résoudre. En fait ces deux étapes sont interdépendantes car bien sûr il faut définir un réseau de contraintes qui soit satisfiable avec les services proposés par les agents présents dans le système. Par exemple d'abord tenter de résoudre un réseau de contraintes du type « TRANSPORT Dijon X » et « TRANSPORT X NewYork ». Puis si aucune solution n'est trouvée alors rajouter une escale supplémentaire, ce qui revient à rajouter une variable au réseau de contraintes qui devient : « TRANSPORT Dijon X » « TRANSPORT X Y » « TRANSPORT Y NewYork ». Cette procédure est répétée ainsi jusqu'à obtenir un nombre d'escales suffisant qui permette de trouver un trajet réalisable à partir des propositions des compagnies de transport (le nombre d'escale dépend de la desserte des villes de départ et d'arrivée : moins ces villes sont desservies, plus il faut d'escale intermédiaires).
Pour être réalisée efficacement, une telle procédure nécessite l'utilisation d'algorithme de satisfaction de contraintes dynamique (le réseau peut être modifié dynamiquement en cours de résolution, par ajout / suppression / modification de contraintes et/ou de variables. L'algorithme calcule une solution au nouveau réseau de contrainte par adaptation des solutions aux précédent réseau, en limitant ainsi les calculs nécessaires). Le §5.10.5.2 discute cette extension possible du mécanisme d'interaction.

Des heuristiques spécifiques au domaine du voyage permettraient certainement d'accélérer la résolution du problème, en créant plus rapidement un réseau de contrainte satisfiable (par exemple, toujours faire escale à Paris lorsque le départ ou la destination est une ville de province en France, etc.).

3.6.2.2 Problèmes d'ontologies lors des échanges d'information entre agents

Les interactions nécessitent l'échange d'informations entre différents agents qui sont fortement hétérogènes :

- chaque agent a son propre métier, ou domaine de compétence. A chaque métier correspond une ontologie particulière, c'est-à-dire des concepts et des formalisme de représentation de ces concepts, différents. Ainsi, AirFrance et AgVoy ne décrivent pas nécessairement un transport en avion de la même manière. Pour AirFrance, la description du service contient certainement le numéro matricule de l'avion, ainsi que les noms et grades des pilotes. Pour AgVoy en revanche, ces informations sont non pertinentes et ne sont donc pas représentées. De la même manière un service de type transport de SNCF est certainement très différent de celui fourni par AirFrance, puisque l'un est un transport par train et l'autre par avion.
- les agents ont été conçu par des développeurs différents. Même si les agents AirFrance et TWA ont le même métier (ce sont tous les deux des transporteurs aériens), il est vraisemblable que leurs systèmes d'information n'utilisent pas les mêmes formalismes pour représenter la même information. Des problèmes très graves peuvent apparaître si l'un des systèmes représente le nom des passagers en donnant d'abord le prénom puis le nom (notation anglo-saxonne), alors que l'autre fait l'inverse (notation française)[14]....

Ces problèmes d'ontologies sont traités par l'adoption d'une ontologie commune pour tous les agents du système, ou bien l'utilisation de traducteurs d'ontologies. Ces problèmes ne sont pas abordés dans ce travail.

3.7 Conclusion : objectifs du travail et démarche.

Ce travail a pour objectif d'utiliser les concepts des Systèmes Multi-Agents pour résoudre le problème de l'adaptation dynamique de l'environnement d'un agent. Dans le cadre de ce travail, nous limiterons notre étude de l'adaptation dynamique aux SMA de type SCDO. Nous nous concentrerons donc sur la définition d'un mécanisme d'interaction efficace dans les conditions d'exploitation particulières de ces systèmes.

[14] Ce problème est malheureusement tout à fait réaliste, même au sein d'une unique entreprise : le système d'information de BT (British Telecom) stocke les noms des usagers britanniques sous plusieurs dizaines de formats différents, ce qui pose évidemment de gros problèmes d'interopérabilité entre sous-systèmes...

L'analogie entre groupes d'agents serveurs et solution d'un réseau de contrainte est très utile. Elle permet de modéliser précisément le problème, et suggère également une solution : les mécanismes d'interaction peuvent s'inspirer des techniques utilisées pour résoudre les problèmes de satisfaction de contraintes. Toutefois le problème de cette thèse ne se ramène pas entièrement à un problème de satisfaction de contraintes. En effet, les valeurs possibles des variables (i.e. les services que peuvent fournir les agents serveurs) ne sont pas connues initialement ; il faut donc les rechercher dans le réseau (c'est la diffusion des appels d'offres lors de la première étape de l'interaction). En outre, ces valeurs sont « dispersées » sur le réseau puisque les services correspondant sont fournis par des serveurs distants les uns des autres. En conséquence, la recherche des valeurs d'une variable, et le choix d'une valeur pour cette variable (« l'instanciation » de la variable) sont des opérations délicates, qui ne sont absolument pas traitées par les algorithmes de satisfaction de contraintes[15].

Dans l'étude de l'Agence de Voyage, l'attention sera focalisée sur la manière dont AgVoy interagit avec les agents « compagnies de transport » pour acquérir la capacité de réaliser son objectif, et comment ces agents adaptent dynamiquement leur organisation à l'évolution de l'environnement (annulation de vols/trains, etc.).

Le choix du système de l'Agence de Voyage « au détriment » de celui de la conception de satellite, se justifie par la plus grande simplicité et la plus grande intelligibilité du problème de l'Agence de Voyage. Néanmoins les solutions définies dans le cadre de l'Agence de Voyage sont aisément transposables à la conception de satellite. De plus, ces résultats peuvent vraisemblablement être appliqués à d'autres types de SMAs (par exemple des équipes de robots footballeurs, etc.), mais ce point n'a pas été approfondi.

Dans la suite de ce mémoire, un état de l'art présente les modèles traditionnels de contrôle local d'agent, et nous montrerons surtout leurs limitations concernant l'expression du problème traité dans cette thèse. Nous proposons ensuite un mécanisme d'interaction (chapitre §5) permettant l'adaptation dynamique de l'organisation du SMA. Une implémentation à base d'agents mobiles est réalisée (chapitre §7). Cette réalisation informatique utilise la plate-forme d'exécution d'agents mobiles JavaNetAgents (décrite au §6) que nous avons développé pour l'occasion.

[15] Même les algorithmes décentralisés ([Ghedira 93], [Ghedira 94], [Yokoo et al. 92], [Yokoo 95]) font l'hypothèse que toutes les valeurs d'une variable sont stockées sur un même noeud du réseau, ce qui élimine le problème de recherche les valeurs et facilite grandement la procédure d'instanciation.

4. Etat de l'Art.

L'état de l'art est composé de deux parties :

1. une présentation des modèles traditionnels de contrôle d'agents. Une attention particulière est portée à la manière dont ces modèles prennent en compte les relations entre agents, afin de juger de leur adéquation à modéliser le problème abordé dans cette thèse.
2. une présentation des technologies à base d'agents mobiles. Elles seront particulièrement utiles pour réaliser des implémentations efficaces (et en particulier, des implémentations très flexibles et ouvertes) de systèmes multi-agents du type systèmes coopératifs distribués et ouverts.

4.1 Les modèles de contrôle d'agents.

Dans cette partie son successivement décrits :

1. le principe général du contrôle d'un agent, et l'un des modèles de contrôle les plus couramment utilisés : le modèle intentionnel.
2. le modèle de dépendance sociale de Castelfranchi, qui étend le modèle intentionnel et rend compte des relations entre agents, et de la manière dont ces relations sont prises en compte par le contrôle local de chaque agent.

4.1.1 Contrôle local

La structure générale d'un agent est illustrée sur la Figure 14 (en prenant l'exemple de l'agent SNCF). Un agent dispose :

1. d'un gestionnaire de communications, qui assure la réception (resp. l'émission) des messages en provenance (resp. à destination) des autres agents du système. Ce gestionnaire permet à l'agent de recevoir (et d'émettre) des informations vers son environnement[16].
2. de comportements, qui correspondent aux compétences de l'agent. Dans l'exemple de l'Agence de Voyage, les comportements des agents correspondent à la fourniture de services de transport (réservations de places sur des trains). Par exemple pour l'agent

---

[16] D'une manière plus générale, l'agent peut disposer d'autres sources d'informations : système de vision, détecteur de phéromone, etc. Dans le cadre de cette thèse, nous considérerons que les seuls échanges entre l'agent et son environnement (c'est-à-dire les autres agents du SMA) se font par le moyen de messages.

SNCF, ces comportements correspondent chacun à des réservations sur des TGV « Dijon Paris » différents (SNCF dispose de trois comportements s_{SNCF1}, s_{SNCF2}, et s_{SNCF3} cf. §2.4).

3. d'un mécanisme de contrôle qui détermine quel(s) comportement(s) l'agent active. Dans le cas de l'agent SNCF, activer un comportement revient à fournir le service correspondant, c'est-à-dire enregistrer une réservation sur un train.

Parce qu'ils sont autonomes, les agents ont chacun un mécanisme de contrôle qui leur est propre. On parle donc de « contrôle local ». Le mécanisme de contrôle local utilise les informations données par le gestionnaire de communication, et détermine quels comportements de l'agent sont activés. Dans ce processus, le contrôle local tient également compte de la finalité de l'agent (ses objectifs propres, la fonction qu'il remplit, etc.).

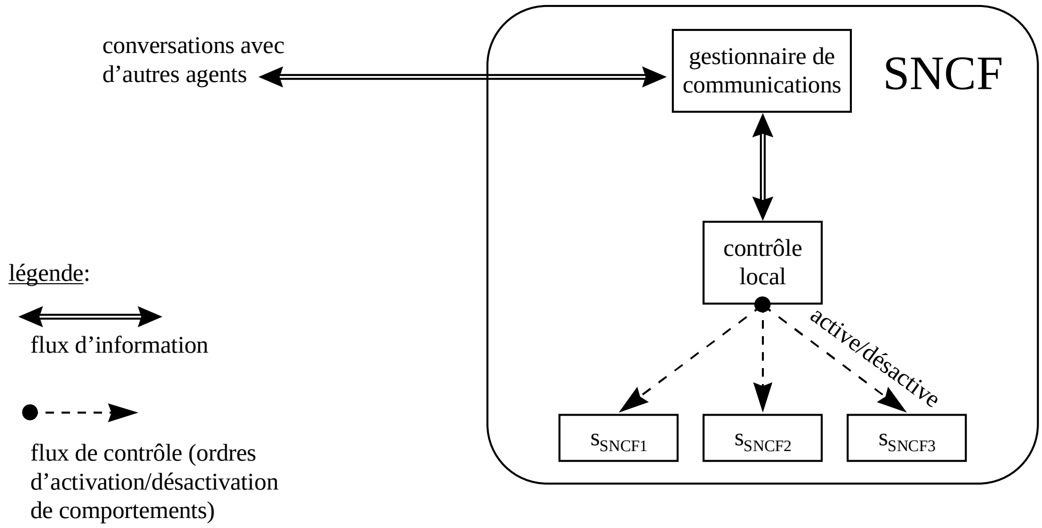


Figure 14. Structure générale d'un agent

La Figure 15 décrit un scénario simple de fonctionnement de cette structure. Ce scénario comporte cinq étapes (indiquées par les numéros sur la figure) :

- étape 1 : réception par le récepteur de messages de SNCF d'un message de l'agent AgVoy. Ce message est un ordre d'achat d'un service (en l'occurrence s_{SNCF3}). Un acte de paiement électronique de 320fr accompagne ce message.
- étape 2 : le récepteur de messages transmet ce message au contrôle local de l'agent SNCF. Comme le paiement est d'un montant suffisant, le contrôle local décide d'accepter la requête de AgVoy.
- étape 3 : le contrôle local envoie un ordre d'activation au comportement. A partir de ce moment, AgVoy dispose effectivement d'une réservation sur le TGV correspondant.
- étape 4 : le contrôle local confirme la réservation à l'agent AgVoy.
- étape 5 : le message de confirmation est effectivement envoyé par l'émetteur de message de l'agent SNCF, à destination de l'agent AgVoy.

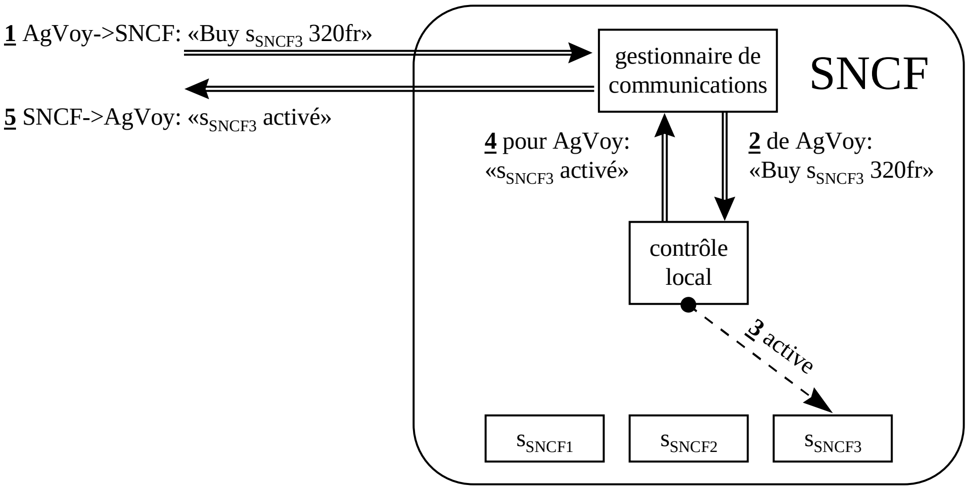


Figure 15. Scénario de fonctionnement d'un agent

Il faut noter que la structure d'agent utilisée (gestionnaire de communication - contrôle local - comportements) est très générale, et que le modèle de contrôle local (modèle intentionnel à base d'états mentaux comme pour Agent0, ou modèle réactif, etc.) est indépendant de cette structure. Le paragraphe suivant détaille un type de modèle de contrôle très utilisé, les modèles intentionnels.

4.1.2 Modèles de contrôle intentionnel

4.1.2.1 Les états mentaux

Les modèles intentionnels de contrôle local utilisent des logiques multi-modales temporelles. Ce type de modélisation affecte à l'agent des états mentaux ([Shoham 93]). Les principaux types d'états mentaux[17] sont les croyances (notées BEL, pour *belief*) et les engagements (notés CMT, pour *commitment*). Les engagements portent sur l'activation d'un comportement précis, à une date précise. A la date indiquée dans l'engagement, le contrôle local fait activer le comportement spécifié. La Figure 16 illustre le fonctionnement générale d'un agent dont le contrôle local est de type intentionnel.

[17] Cette présentation des modèles intentionnels de contrôle utilise comme référence le modèle Agent0 ([Shoham 93]). Ce modèle est introduit par Shoham dans le cadre de l'Agent Oriented Programming (AOP). D'autres modèles existent, qui diffèrent dans le choix des états mentaux principaux : dans le modèle « Cohen&Levesque » ([Cohen et al. 95]) les engagements sont remplacés par les objectifs). Toutefois les objectifs peuvent être définis comme des engagements particuliers, et réciproquement. Ces différences sont donc ici négligeables. Le choix du modèle de Shoham est justifié par le fait que ce modèle décrit explicitement une boucle de contrôle (cf. Figure 18), tandis que le modèle de Cohen&Levesque reste très vague sur la manière dont sont exploités les états mentaux.

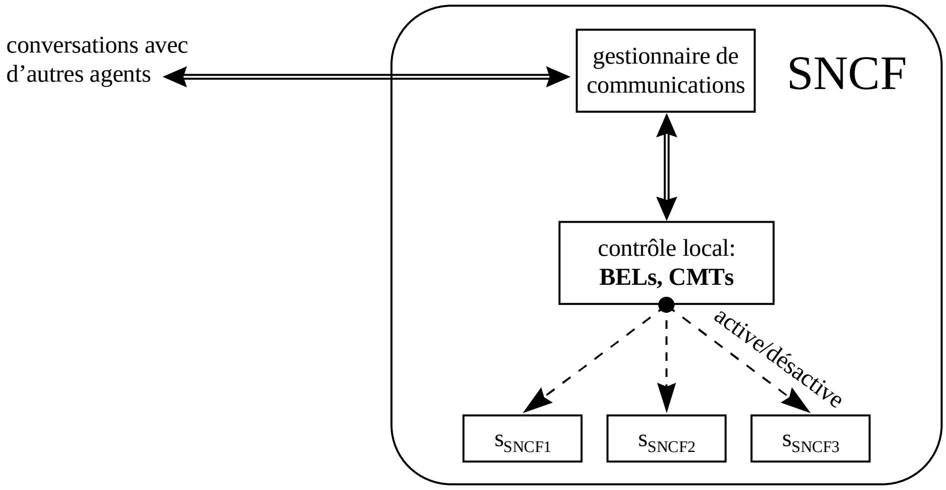


Figure 16. Structure générale d'un agent de type « intentionnel ».

Le contrôle local met en permanence à jour l'ensemble des états mentaux de l'agent, ajoutant/supprimant des croyances et des engagements. Par exemple la réception d'un message d'information « il fait beau » (INFORM « il fait beau ») provoque l'ajout d'une croyance portant sur le fait qu'il fait beau (BEL « il fait beau »). Par inférence sur l'ensemble courants des croyances et des engagements de l'agent, le mécanisme de contrôle local peut ajouter/supprimer des croyances ou des engagements (par exemple l'engagement CMT « aller se promener cet après-midi »).

Le contrôle local peut ainsi être représenté une « boucle », répétée indéfiniment, et constituée de 2 étapes (cf. Figure 17) :

1. interprétation des messages reçus, mise à jour des croyances et des engagements.
2. exécution des engagements qui doivent l'être à l'instant présent, à condition que l'agent ait la capacité de les exécuter (sinon, l'agent les abandonne).

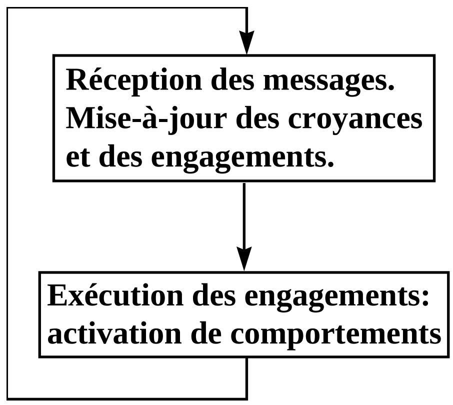


Figure 17. Boucle de contrôle intentionnel

4.1.2.2 Prise en compte des capacités de l'agent

La capacité d'un agent Agent0 à activer un de ses comportement est représentée par un type d'état mental (noté CAN, cf. [Shoham 93]) comme suit :
(CAN p) ≡ ((CH p) ⇒ p)
Ce qui signifie « l'agent peut faire p, si p devient vrai lorsque l'agent décide qu'il l'est ».

Cette définition lie la capacité d'activer un comportement à l'agent qui « possède » ce comportement, *indépendamment de l'environnement de cet agent*. De ce fait, les relations de dépendance entre agents ne sont pas explicitées. Elles ne sont pas prise en compte par le mécanisme de contrôle local de chaque agent.

4.1.2.3 Conclusion

Les mécanismes de contrôle déterminent quels comportement l'agent désire activer, sans tenir compte des capacités effectives de l'agent. Plus précisément, ces modèles considèrent que la capacité à activer un comportement est une propriété interne de l'agent, d'où sa modélisation par un état mental. Ce point de vue néglige complètement le rôle de l'environnement de l'agent, qui est pourtant fondamental, comme le démontre l'analyse de l'exemple de l'Agence de Voyage.

Ces modèles sont donc essentiellement « orienté agent ». Ils n'accordent pas suffisamment d'importance aux relations entre agents pour rendre compte de manière satisfaisante du problème abordé dans cette thèse.

Dans ce qui suit, le modèle de Castelfranchi est décrit. Il présente une extension de ces modèles, qui prend en compte les relations de dépendance entre agents.

4.1.3 Le modèle des dépendances sociales de Castelfranchi

4.1.3.1 Présentation

Le modèle de « dépendance sociale » de Castelfranchi ([Castelfranchi&al.93]) définit une extension des modèles de contrôle intentionnels, pour la prise en compte explicite des relations de dépendance entre agents (notées DEP).

L'objectif de Castelfranchi (qui fait l'originalité de son travail) est de montrer comment la simple prise en compte de liens de dépendance par les agents (i.e. par leur mécanisme de contrôle local) peut expliquer leur coopération. Castelfranchi place ainsi le concept de relation de dépendance à la base de son modèle, et à la base de toute sa théorie sur la coopération entre agents.

Note : le modèle de Castelfranchi est en fait basé sur celui de Cohen&Levesque, de ce fait il utilise les croyances et les objectifs comme « états mentaux de base », lieu de des croyances et engagements comme dans le modèle de Shoham. Néanmoins, le modèle de Castelfranchi est aisément et directement transposable au modèle de Shoham. En effet les dépendances entre engagements peuvent être analysées de la même manière que Castelfranchi traite les dépendances entre objectifs.

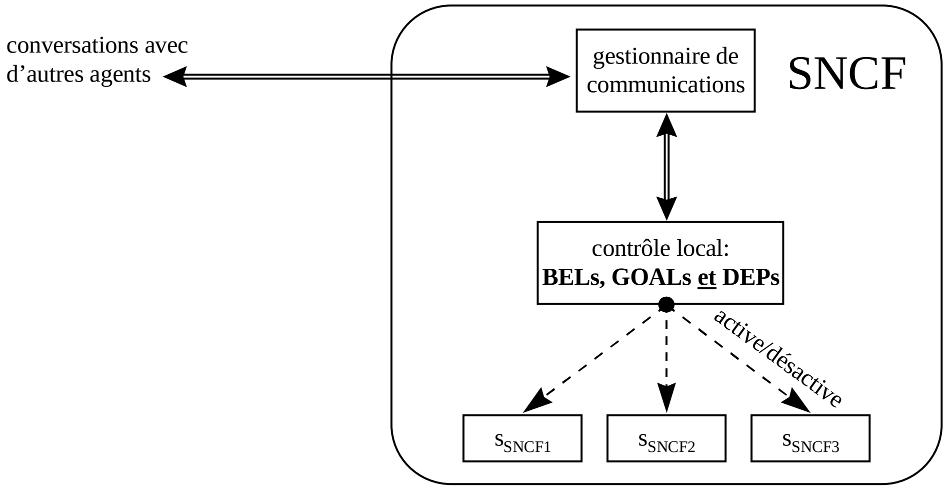


Figure 18. Boucle de contrôle d'un agent « de Castelfranchi »

4.1.3.2 Relations de dépendance entre agents

La relation de dépendance entre un agent A et un agent B vis-à-vis d'un objectif G_A de A (Figure 19), découle de l'incapacité de A à atteindre G_A, et de la capacité de B à atteindre G_A . En fait, A est dépendant de tous les agents capable d'atteindre G_A .

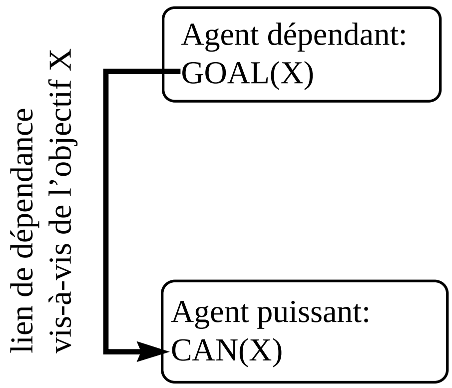


Figure 19. Relation de dépendance

Dans l'exemple de l'Agence de Voyage (Figure 20), AgVoy est dépendant de SNCF pour son objectif « réserver un billet Dijon Paris départ 8h30 arrivée 10h37 ». De même, AgVoy est dépendant de AirFrance et de TWA pour son objectif correspondant au trajet Paris NewYork. Et Dupond est dépendant de AgVoy pour le trajet complet Dijon NewYork.

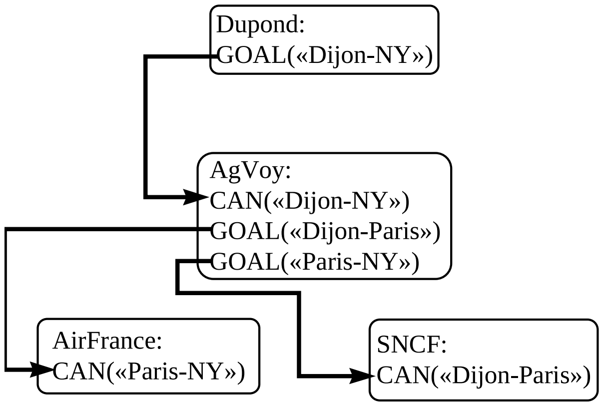


Figure 20. Relations de dépendance dans l'exemple de l'Agence de Voyage

Il faut noter que, contrairement aux états mentaux, les dépendances sont objectives : elles sont liées à l'état du monde et existent indépendamment de la conscience qu'en ont les agents. Cependant la plupart des propriétés dérivées des relations de dépendances ne valent que si les agents ont conscience d'être dépendants. Ces propriétés s'expriment donc par des croyances (BEL) ou des objectifs (GOAL).

4.1.3.3 Prise en compte des relations de dépendance par le contrôle local de chaque agent

Castelfranchi définit des « attitudes » qui caractérisent la manière dont le contrôle local d'un agent prend en compte les relations de dépendance liant l'agent aux autres agents du système. Trois attitudes sont détaillées dans [Castelfranchi&al.93] :

1. la relation de pouvoir
2. l'échange social
3. la coopération

Elles sont présentées dans ce qui suit.

4.1.3.3.1 La relation de pouvoir

La relation de pouvoir (Figure 21) est en fait symétrique de la relation de dépendance : si un agent A dépend d'un agent B vis-à-vis d'un objectif G_A, alors B dispose d'un pouvoir d'influence sur A.

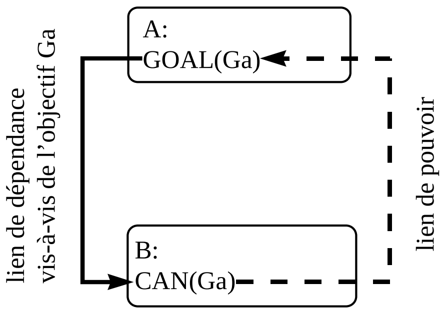


Figure 21. Relation de pouvoir

Le contrôle local de B peut exploiter cette relation de pouvoir en faisant adopter un objectif de son choix par A : [le contrôle de] B adopte l'objectif G_A à la condition que [le contrôle de] A adopte un objectif choisi par B. Par cette attitude, B influence donc le contrôle de A, ce qui est impossible sans une relation de dépendance de A envers B (car A est autonome).

4.1.3.3.2 L'échange social

L'échange social (Figure 22) survient lorsque deux agents sont en relation de dépendance réciproque, c'est-à-dire lorsqu'ils dépendent l'un de l'autre vis-à-vis d'objectifs distincts.

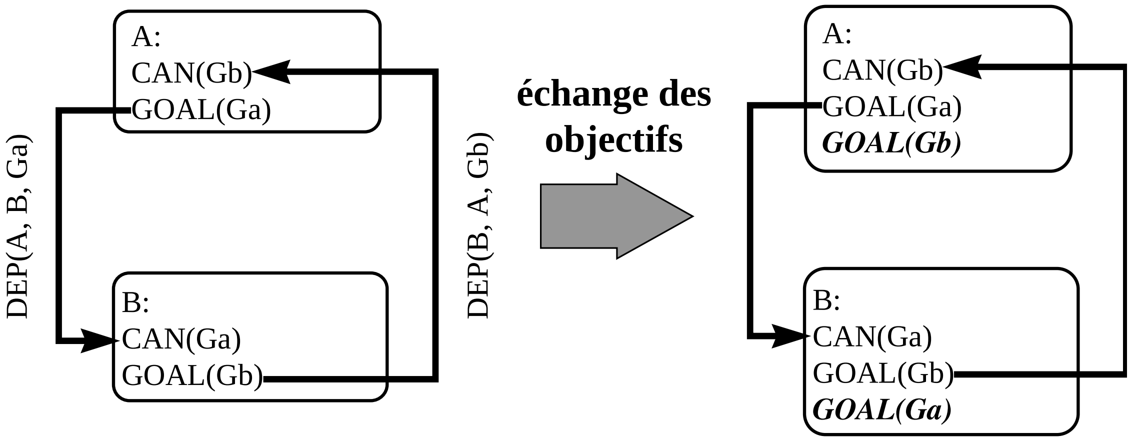


Figure 22. Echange social

L'échange social consiste en l'adoption de par G_B le contrôle local de l'agent A à la condition que B adopte G_A, et réciproquement : B adopte G_A à condition que A adopte G_B.

4.1.3.3.3 Coopération

La coopération entre agents est basée sur la dépendance mutuelle (Figure 23).
Les agents A et B sont en relation de dépendance mutuelle vis-à-vis d'un objectif G quand ils dépendent l'un de l'autre pour le même objectif G. L'objectif G est décomposé en deux sous-objectifs $G=G_A \wedge G_B$. L'agent A capable d'atteindre G_A et pas G_B, tandis que l'agent B peut atteindre G_B mais pas G_A.

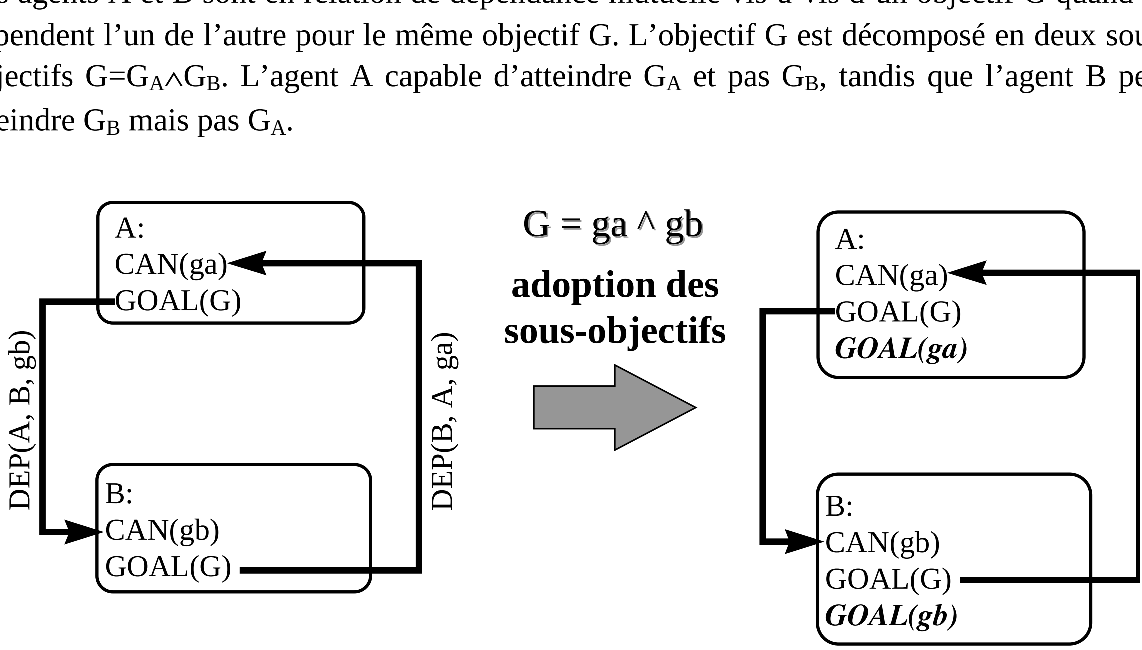


Figure 23. Coopération

L'attitude de coopération consiste en l'adoption de G_A le contrôle local de l'agent A à la condition que B adopte G_B, et réciproquement : B adopte G_B à condition que A adopte G_A.

[Sichman 98] décrit comment la coopération mène à la formation de coalition entre agents : chaque agent de la coalition adopte l'objectif d'un autre membre, et « en retour » il obtient la satisfaction d'un de ses propres objectifs par un autre membre. [Sichman 98] traite le cas de coalition de deux agents, mais ce résultat peut aisément être étendu à des coalitions de plus grande taille.

4.1.3.4 Remarque : une perspective économique sur l'échange social

Il est intéressant de noter que l'échange social (et par extension, la coopération) correspondent, d'un point de vue économique, à un troc d'objectifs.
Cette forme d'échange est la première apparue dans les sociétés humaines. C'est une forme primitive qui rend difficile les échanges entre individus : un agent souhaitant troquer son objectif risque de ne pas trouver de partenaire (i.e. un agent (1) capable d'atteindre cet objectif, et (2) souhaitant troquer un objectif que le premier agent est lui-même capable d'atteindre), dans ce cas l'objectif n'est pas atteint. C'est pour cette raison que dans les sociétés humaines, le troc a été quasi-systématiquement abandonné au profit de l'échange commercial, qui utilise un bien dont la valeur est reconnue et appréciée de tous : la monnaie[18].
Il est alors plus facile à un agent de trouver un partenaire d'échange : n'importe quel agent compétent atteindra l'objectif... à condition d'être payé un prix suffisant.
Bien que les échanges commerciaux puissent être décrits sous la forme de trocs dont un des biens est de la monnaie (l'agent serveur adopte l'objectif de fournir un service à l'agent client, qui en échange adopte l'objectif de payer l'agent serveur), cette représentation est inutilement lourde. Le modèle de Castelfranchi est donc inutilement lourd lorsqu'il est appliqué au cas d'agents commerciaux (cf. §3.1), dont les échanges sont *de facto* des échanges commerciaux.

4.1.3.5 Conclusion

Le modèle des dépendances sociales de Castelfranchi rend compte du rôle des relations de dépendance dans le contrôle local d'un agent.
Plus précisément, Castelfranchi décrit :

- comment le contrôle local d'un agent exploite les relation de dépendance (exploitation d'une relation de pouvoir, échange social, coopération)
- comment les relations de dépendance influent sur le contrôle local de l'agent (politique d'adoption d'objectif « étrangers »)

---

[18] Le problème de l'établissement des prix des objectifs est un problème extrêmement intéressant. Les prix indiquent la valeur comparée d'objectifs. Ils permettent d'identifier par exemple des compétences très demandées mais peu présentes dans le SMA (ce sont celles nécessaires pour atteindre les objectifs les plus chers). L'introduction des concepts micro-économiques permettrait vraisemblablement des développement intéressants de la théorie de Castelfranchi... totalement hors sujet dans le contexte de cette thèse.

Le modèle résultant est complet : il traite à la fois du fonctionnement interne des agents (puisqu'il étend un modèle de contrôle local « orienté agent ») et de leurs relations. Ce modèle est donc particulièrement puissant.

Le modèle de Castelfranchi présente toutefois deux défauts.
Le premier est lourdeur inutile lorsqu'il est appliqué à des agents commerciaux, qui achètent ou vendent leurs services, au lieu de pratiquer un troc plus général comme le prévoit Castelfranchi.

Le second et principal défaut du modèle de Castelfranchi est la non prise en compte de corrélations entre les objectifs des agents[19]. Dans les exemples cités ci-dessus, les objectifs G_A et G_B sont non-corrélés : ils peuvent être spécifiés et atteints indépendamment l'un de l'autre.
Or au §3.5 sont identifiés d'importants problèmes liés à la corrélation (il s'agit alors de la corrélation des services fournis par des agents, mais des problèmes identiques se posent dans le cas de la corrélation d'objectifs). La prise en compte de la corrélation des objectifs nécessite la définition d'une attitude de coopération plus sophistiquées de celles citées par Castelfranchi. En particulier, les agents peuvent être amenés à adapter leurs objectifs propres aux modifications ou à l'état (atteint / non atteint) des autres objectifs.

4.1.4 Conclusion générale

Ces modèles traditionnels sont « orientés agents » et n'explicitent pas la manière dont les relations de dépendance entre agents sont prises en compte par le contrôle local de chaque agent. Ils traitent de la génération des objectifs de l'agent, mais sans tenir compte de ses capacités, qui sont fonction de ses dépendance envers d'autres agents du système.

Le modèle de dépendance sociale de Castelfranchi étend ces modèles pour rendre compte du rôle joué par les relations entre agents dans le contrôle local. Cette approche fournit un modèle complet (explicitant à la fois le contrôle local –interne– de l'agent et le rôle des relation entre agents). Mais, il est lourd et néglige la corrélation des objectifs. De ce fait, le modèle de Castelfranchi souffre d'être encore trop « orienté agent » et pas assez « orienté relations ».

4.2 Les architectures à base d'agents mobiles

Cette partie de l'état de l'art introduit les architectures à base d'agents mobiles. L'objectif est de montrer leurs apports aux architectures traditionnelles, qui justifient notre choix de réaliser avec des agents mobiles l'application informatique Agence de Voyage.

---

[19] Les travaux de Sichman ([Sichman 98]) présentent ce même défaut.

Dans ce qui suit, nous présentons tout d'abord les limitations des architectures client/serveurs traditionnelles en nous basant sur l'exemple très représentatif de OMA/CORBA. Puis sont introduits le principe de la mobilité de code et les trois principaux paradigmes de son exploitation. Ensuite nous détaillons les technologies liées aux agents mobiles. Enfin nous comparons les avantages (et leurs inconvénients) des architectures à base d'agents mobiles face aux architectures traditionnelles.

4.2.1 Conventions graphiques

Dans ce chapitre, les conventions graphiques suivantes sont appliquées (Figure 24) :

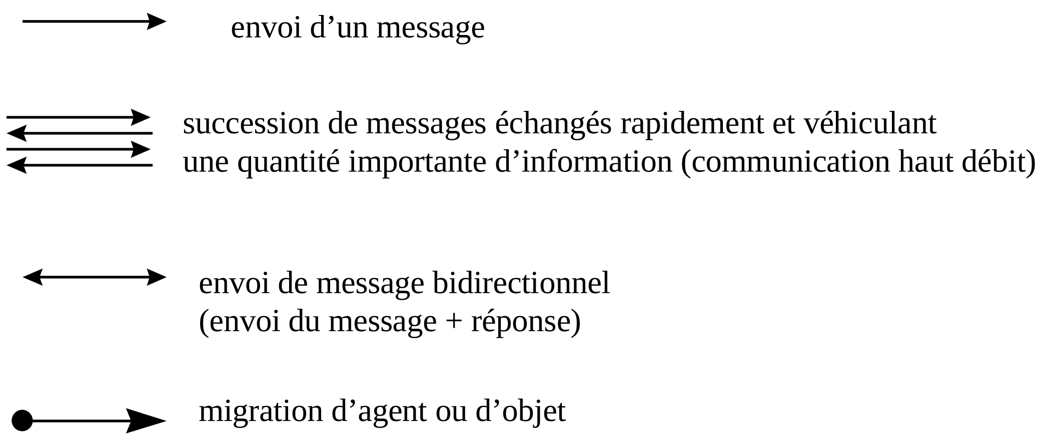


Figure 24. Conventions graphiques utilisées dans ce chapitre.

4.2.2 Les architectures « traditionnelles » de système distribués.

4.2.2.1 Exemple de l'architecture OMA/CORBA

4.2.2.1.1 Présentation

L'architecture CORBA (*Common Object Request Broker*) a été définie par l'OMG *(Object Management Group*) pour faciliter le développement de systèmes distribués hétérogènes dans un cadre orienté objet ([OMG92]).
L'architecture OMA (*Object Management Architecture*) est une surcouche de CORBA. Elle définit un ensemble de services accessibles à tout objet CORBA. Ces services sont variés, et s'avèrent très utiles à la conception et à la réalisation de systèmes distribués hétérogènes.
Nous appellerons OMA/CORBA l'architecture résultante de la « composition » de OMA et de CORBA. Cette architecture constitue le standard des architectures de systèmes distribués hétérogènes. C'est pourquoi nous la prenons comme référence pour présenter les architectures traditionnelles de systèmes distribués.

4.2.2.1.2 Les objets CORBA

Un objet-application CORBA (ou *application object*) encapsule une application logicielle quelconque, et lui permet « d'exporter » ses services auprès des autres objets-applications. Pour ce faire, les services sont décrits dans un langage normalisé de description d'interface (IDL, *interface description language*). Tout objet-application peut alors (1) découvrir les services offerts par l'application logicielle, et (2) invoquer ces services au moyen d'un simple appel de méthode sur l'objet-application qui l'encapsule. Il s'agit d'une interaction de type « client/serveur » : l'objet appelant est le client de l'objet destinataire, qui est un serveur.

La Figure 25 illustre l'encapsulation d'un serveur de réservation (d'une compagnie de transport) par un objet-application CORBA.

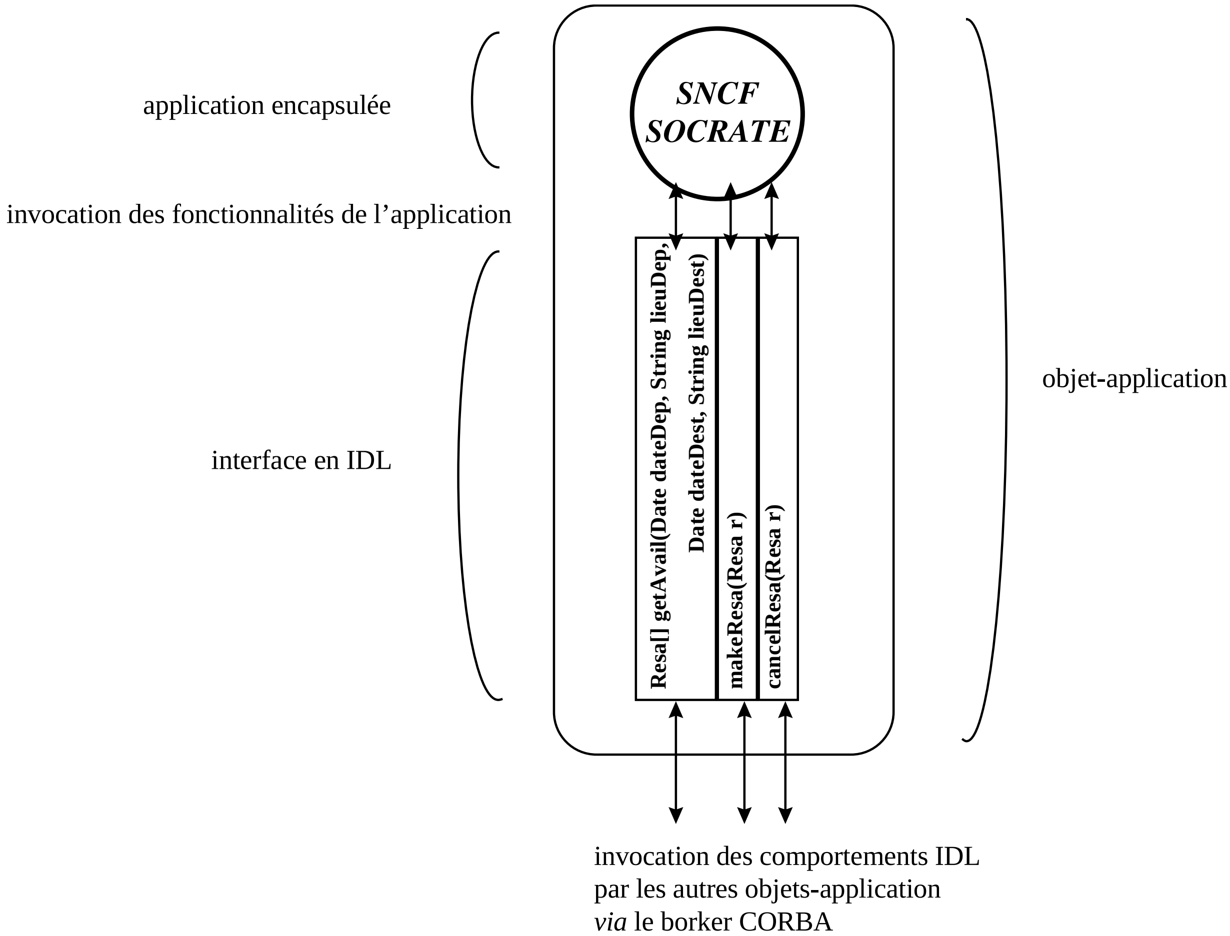


Figure 25. encapsulation du serveur de réservation SNCF dans un objet-application CORBA

4.2.2.1.3 Le bus à objets CORBA

Les communications inter-objets sont assurées via un « bus à objets », autrement appelé Object Request Broker ou Broker CORBA (Figure 26). Le broker véhicule les messages de l'objet-application émetteur vers l'objet-application récepteur, et qui correspondent à une invocation d'un service de l'application encapsulé par le receveur. L'envoi de message est de type « synchrone bidirectionnel » : l'objet-application émetteur reçoit en réponse à son message la valeur de retour de l'invocation du service concerné.

Le broker assure la communication des objets situés sur une même machine, mais également sur des machines distantes : sur chaque machine se trouve un bus à objets, et tous ces bus communiquent via Internet grâce au protocole IIOP (*Internet Inter-ORB Protocol*).

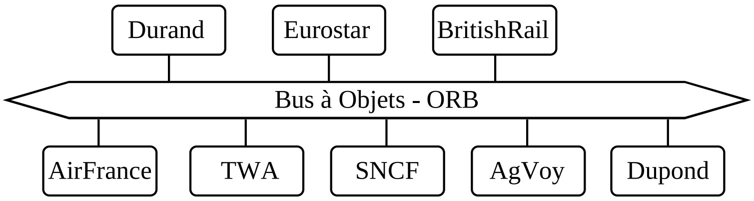


Figure 26. Bus à Objets CORBA reliant divers objets-applications

Grâce au broker CORBA, l'appel à un objet « distant » (i.e. situé sur une autre machine que l'objet appelant, et donc géré par un autre bus à objet) est réalisé de la même manière qu'un appel à un objet « local » (i.e. situé sur la même machine que l'objet appelant). De ce fait, tous les objets-applications « apparaissent » locaux, même ceux qui sont situés physiquement sur des machines très éloignées, comme illustré sur la Figure 27. La dispersion géographique est totalement cachée. CORBA facilite ainsi la conception de systèmes distribués à forte dispersion géographique.

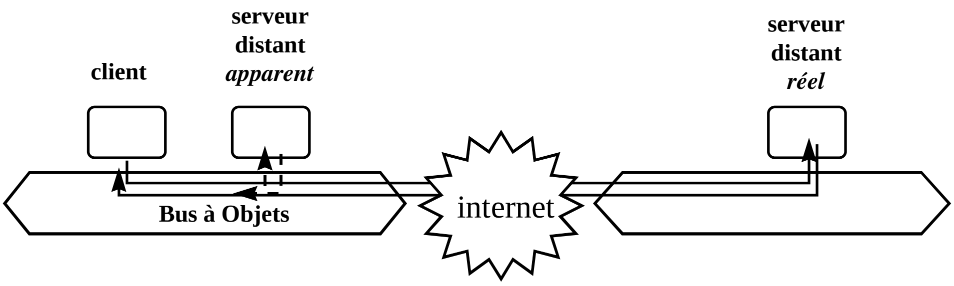


Figure 27. Invocation d'un serveur distant

4.2.2.1.4 L'architecture OMA/CORBA

OMA est une « surcouche » de CORBA. Elle introduit des objets serveurs qui proposent des services spécialisés accessibles aux applications : les services communs (*common facilities*), cf. Figure 28.

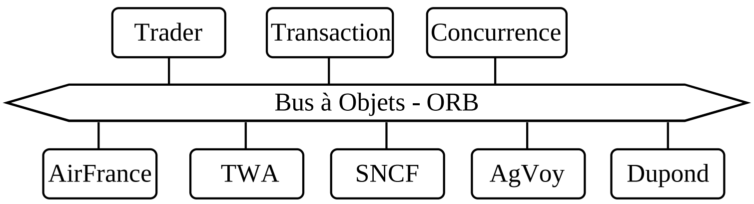


Figure 28. L'architecture OMA = CORBA enrichi par les services communs

Les services communs facilitent la communication et les interactions entre objets CORBA. Ils sont nombreux. Citons par exemple ([OMG92]) :

- cycle de vie, qui permet la création de nouveaux objets-applications et gère leur destruction également.

- nommage, qui permet à un objet-application d'envoyer un message à un autre objet-application. C'est une sorte d'annuaire « page blanche » qui répertoire la localisation exacte de chaque objet-application connecté au broker.
- courtage (en anglais « trading »), route les requêtes d'objets-applications clients vers des objets-applications serveurs capables de les traiter. C'est une sorte d'annuaire « page jaune » qui répertorie les comportements de chaque objet-application connecté au broker (i.e. les services qu'ils sont susceptibles de fournir).
- transaction, qui assure le respect de contrainte d'intégrité nécessaires dans le cadre de transactions client/serveur.
- persistance, qui permet la sauvegarde des objets-applications.
- événements, qui assure la diffusion d'informations entre objets-applications. Ce service est similaire au service courtage, à la différence que les messages routés sont des notifications d'événements, et non pas des requêtes d'exécution de services. C'est un service de « push » d'information.
- *workflow*, qui assure l'exécution successive et/ou concurrente d'un ensemble de services, selon un graphe d'exécution donné (un *business process*).
- etc.

Ces services s'avèrent *très* utiles à l'implémentation de systèmes distribués en général, et donc à celle des SMA de type SCDO en particulier.

4.2.2.2 Limitations des architectures « traditionnelles » de systèmes distribués

Les architectures « traditionnelles » de systèmes distribués présentent des limitations qui sont particulièrement pénalisantes dans le cadre de l'implémentation de systèmes multi-agents. Ces limitations sont nombreuses :

1. interactions synchrones[20] : l'objet client est bloqué en attente du résultat de l'exécution du service. Comme aucune garantie n'est donnée sur le temps d'exécution du service, l'objet client peut ainsi être bloqué pendant longtemps.
2. sensibilité aux pannes réseau : du fait du caractère synchrone de l'invocation de service, l'invocation du service échoue en cas de panne réseau. Elle doit être réitérée une seconde fois, même si la panne réseau a lieu pendant l'exécution du service, c'est-à-dire dans une période ou le client et le serveur n'échangent pas d'informations !
3. coût en bande passante : le message d'invocation d'un service, et de résultat de son exécution, peuvent représenter une important quantité de donnée. C'est le cas pour les services de traitement d'image, de son, de vidéo, etc. L'invocation de tels services est de ce fait très coûteuse en bande passante.
4. architecture figée : la spécification des services communs de OMA constitue un standard (le standard OMA, justement). Par voie de conséquence ces services sont figés. Ils ne peuvent être dynamiquement modifiés en phase d'exploitation du système distribués, car il faudrait alors mettre à jour tous les objets-applications faisant appel à ces services. Cette

[20] Certaines implémentations de l'architecture OMA/CORBA fournissent la possibilité d'appels asynchrones. C'est notamment le cas de celle commercialisée par Orbix (de Iona, http://www.iona.com). Le produit Orbix est cependant une exception qui confirme la règle.

limitation empêche l'adaptation dynamique de ces services à l'évolution des conditions d'exploitation courantes (pannes réseaux, etc.), ou à l'introduction dans le système de nouvelles applications logicielles ayant des besoins spécifiques. Ces deux types d'événements sont pourtant courants dans les conditions d'exploitation des SMA de type SCDO.

4.2.2.3 Conclusion

Les architectures « traditionnelles » de systèmes distribués présentent de nombreuses limitations qui les rendent peu adaptées à l'implémentation de SMAs de type SCDO.

Les conditions d'exploitation de ces systèmes sont en effet caractérisées par une forte dispersion géographique et par des possibilités de pannes réseau et machines. De plus l'architecture informatique qui sert de support à l'implémentation de ces systèmes doit disposer de la flexibilité suffisante pour faciliter l'adaptation dynamique de l'organisation des agents.

C'est pourquoi les architectures « traditionnelles » de systèmes distribués ne sont pas adaptées à l'implémentation des SMAs de type SCDO, et donc du système de l'Agence de Voyage développé dans cette thèse.

4.2.3 La mobilité de code

Le développement des systèmes distribués fonctionnant dans des conditions d'exploitation caractérisées par une forte dispersion géographique et de fréquentes pannes de réseau et de machines, comme les systèmes distribués coopératifs ouverts, a conduit récemment à l'introduction de la technologie de mobilité de code.

La mobilité de code définit la capacité de « déplacer » les composants logiciels d'un système distribué d'un noeud du système à un autre ([Telescript95], [Knabe 95], [Harrison et al. 95]). Elle se présente ainsi comme complémentaire des techniques utilisées dans les systèmes distribués « traditionnels », fonctionnant sur le mode « client/serveur ». La mobilité de code peut même être vue comme une alternative à ces techniques : au lieu d'envoyer des données à un serveur pour qu'il les traite avec son expertise et ses capacités propres, c'est le serveur qui envoie ses procédures s'exécuter là où se trouvent les données. Le serveur devient ainsi le client et réciproquement.

4.2.4 Paradigmes d'exploitation de la mobilité de code

Bien que ces techniques soient encore très récentes, trois paradigmes d'utilisation de la mobilité de code se dégagent déjà ([Pitoura et al. 95]) :

4.2.4.1 Le code à la carte

Ce paradigme (en anglais *code on demand*) est utilisé pour enrichir dynamiquement les noeuds du système distribué, soit en y installant de nouveau composants, soit en mettant à jour les composants déjà installés. Dans le deuxième cas, le composant logiciel télécharge la nouvelle version de son propre code (auprès d'un serveur dédié), et remplace en cours d'exécution l'ancienne version de son code par la nouvelle. Cette opération ne nécessite pas la suspension de l'activité du composant, elle n'impose donc pas rupture de service.
La Figure 29 illustre ce paradigme avec le cas d'un application téléchargement différents modules graphiques pour la visualisation de résultats de calculs.

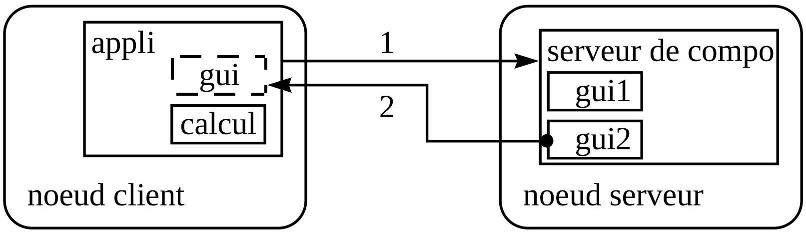


Figure 29. Code à la carte

C'est le paradigme utilisé par les *applets* Java ([Java 95]).

4.2.4.2 L'évaluation distante

Ce paradigme ([Goldszmidt et al. 95]) est le prolongement de l'appel de méthode distant, utilisé dans les architectures client/serveur distribuées traditionnelles. Il permet au client d'invoquer un service distant en passant en paramètre la procédure à exécuter par le serveur. La procédure décrit les opérations à effectuer (par le serveur) sur les ressources offertes par le serveur. En pratique, cette technique revient à passer en argument d'appel d'une procédure distante, non plus seulement des objets de types de base (nombres, chaînes de caractères, etc.) ou comme c'est le cas dans l'appel de procédure distante « traditionnel », mais des objets « complexes ». Ces objets implémentent la procédure à exécuter.
La Figure 30 illustre ce paradigme dans le cas d'une requête de traitement sur des données multimédia, stockées sur un serveur distant.

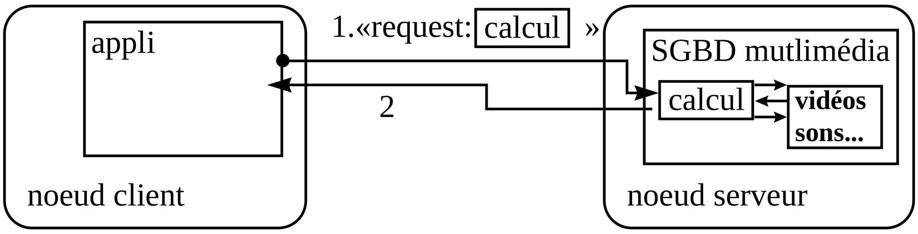


Figure 30. Evaluation distante.

Ce paradigme est utilisé par les *server-side applets*, ou *servlets* Java. Leur usage est encore peu répandu, mais néanmoins promis à un brillant avenir (les « grands » serveurs Web du marché —ceux de Netscape et de Microsoft— ainsi que Sun, proposent chacun une implémentation différente des servlets. Un processus de standardisation est nécessaire avant que les servlets ne soient utilisées à plus grande échelle).

4.2.4.3 Les agents mobiles

Le terme « agent » n'est pas à prendre ici au sens qui lui est donné dans le reste de cette thèse. Il s'agit « simplement » d'une unité d'exécution, possédant ses propres ressources et sa propre expertise ([Lange et al. 96], [Acharya 96], et en particulier le très complet [Knabe 95]). L'agent est placé dans un environnement d'exécution, appelé place, qui lui fournit ressources CPU et également l'accès au ressources locales (e.g. fichiers, sous réserve qu'il ait les droits d'accès suffisant), et aux agents locaux (i.e. ceux résidant dans le même environnement d'exécution). La mobilité de l'agent traduit sa capacité à changer dynamiquement (en cours d'exécution) d'environnement d'exécution, donc de places (Figure 31). L'agent peut exploiter les ressources locales à ces places ou invoquer des services des serveurs locaux. L'utilisation d'agents mobiles nécessite la présence sur chaque noeud du système distribué, d'une place pour les accueillir.

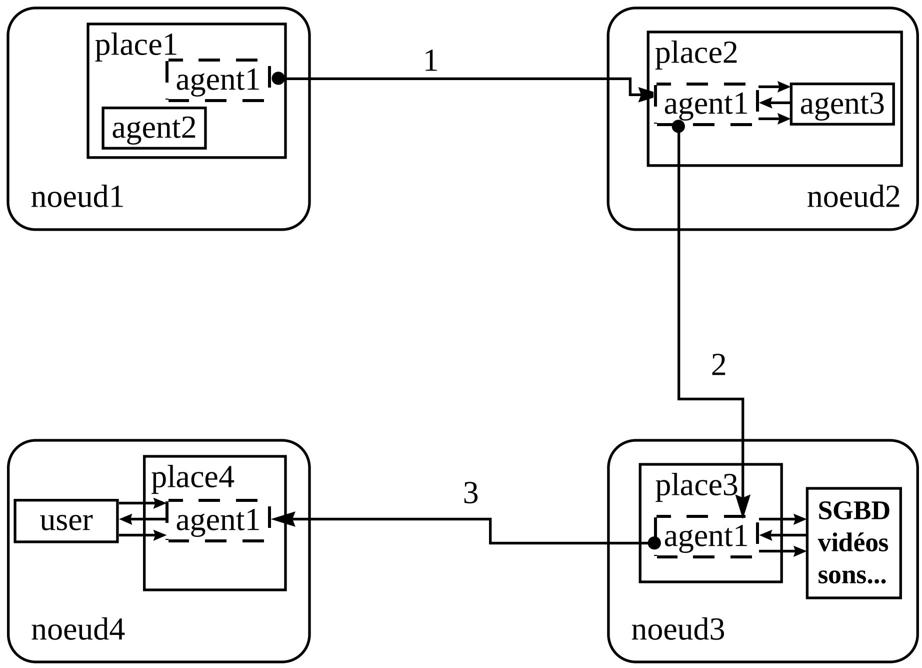


Figure 31. Agents mobiles.

Le paradigme d'agents mobiles est le plus puissant : les deux autres en sont des versions allégées (ou dégradées). C'est pourquoi c'est celui-ci que nous décrirons dans la suite. Insistons encore : le terme « agent » n'est pas à prendre ici comme dans le reste de cette thèse.

Le paradigme d'« agent mobile » se situe au niveau implémentation, c'est-à-dire au niveau « objet » et non pas au niveau « agent ». En toute rigueur, on ne devrait donc pas parler « d'agent mobile », mais « d'agent implémenté par un objet mobile ». Cependant nous avons choisi de conserver le terme « agent mobile », au lieu de celui de « code mobile » ou d' « objet mobile » par souci de ... clarté. En effet, c'est ce terme qui est le plus souvent utilisé dans les travaux relatifs à la mobilité de code.

4.2.5 Les architectures à base d'agents mobiles

Les architectures à base d'agents mobiles sont caractérisées par une très grande flexibilité d'exploitation, et une grande tolérance à la dispersion géographique et aux pannes réseau. Elles sont particulièrement intéressantes pour servir de base à l'implémentation de systèmes multi-agents de type systèmes coopératifs distribués et ouverts.

4.2.5.1 Principe

Les architectures à base d'agents mobiles sont constituées de deux types de composants :

- les agents mobiles.
- les places, qui constituent les environnements d'exécution des agents mobiles.

Ces deux composants sont complémentaires et fortement couplés : en général une place n'accepte qu'un type d'agents mobiles, et une classe d'agent mobile ne peut aller que sur un type de places. Cette limitation est due à un manque de standards dans la spécification des classes d'agents et de places. Elle sera levée lorsque des interfaces standards seront définies pour les places et les agents, ce qui est en cours de réalisation notamment dans le cadre des projets intégrant la mobilité à des architectures de SMA ([FIPA 98]) ou à des architectures de systèmes distribués (projet MASIF pour OMA/CORBA [Milojicic et al. 98]).

4.2.5.2 Classification des différents niveaux de mobilité d'agents

Il existe différents types de mobilité qui se différencient par leur facilité d'implémentation et leur flexibilité. Elles se définissent par le type d'information transmise.

4.2.5.2.1 Niveau 0 : transmission des données seulement.

On ne transmet que des données. Les procédures s'appliquant à ces données doivent être préalablement définies sur tous les sites d'exécution. Il s'agit donc d'une instanciation à distance, qui peut être assimilée à une forme dégradée de mobilité.
Cette mobilité donne une structure figée au système, car elle ne permet pas de diffuser des agents qui ont de nouveaux comportements (i.e. des instances de nouvelles classes).

4.2.5.2.2 Niveau 1 : transmission des données + des procédures.

On transmet les procédures associées aux données. On peut donc créer dynamiquement de nouvelles classes et diffuser leurs instances dans le réseau. Le transfert des procédures est effectué de manière paresseuse : seules sont transmises les définitions des classes que la place réceptrice ne connaît pas déjà.

4.2.5.2.3 Niveau 2 : transmission des données + des procédures + du vecteur d'état.

Le transfert de l'objet se réalise comme au niveau 1, sauf que l'on transmet également le vecteur d'état décrivant l'état précis de l'objet juste avant la migration. Ce niveau apporte la plus grande flexibilité : après avoir migré, l'objet reprend l'exécution de son comportement à l'endroit précis où il s'était arrêté. Ce niveau de migration est difficile à implémenter, car il nécessite d'avoir accès aux vecteurs d'état. Il est utilisé en Telescript ([Telescript95]).

Info transmises \ niveau de mobilité	Niveau 0	Niveau 1	Niveau 2
données	√	√	√
procédures		√	√
vecteur d'état			√
commentaires	Niveau de mobilité nul. Pas d'enrichissement dynamique des noeuds du système.	Bon compromis puissance/facilité d'implémentation. C'est le niveau des plates-formes en Java.	Difficile à implémenter. Pas toujours utile.

Tableau 4. Classification des différents niveaux de mobilité.

Le niveau 1 de migration présente le meilleur compromis puissance/facilité d'implémentation :

- le niveau 0 est inintéressant (il présente les mêmes défauts que les architectures de systèmes distribués « traditionnelles » comme OMA/CORBA)
- l'utilisation de Java comme langage d'implémentation facilite beaucoup l'implémentation de la migration de niveau 1. De fait, la totalité des plates-formes d'exécution d'agents mobiles écrites en Java offrent ce niveau de mobilité (à l'exception de Sumatra, voir plus bas). Notre expérience pratique, la réalisation de l'application « Agence de Voyage » sur la plate-forme JavaNetAgents, suggère qu'une migration de niveau 1 est suffisante.
- le niveau 2 est complexe à réalisé car il nécessite d'avoir accès au vecteur d'état. Peu d'implémentations permettent une manipulation aisée de cette donnée. L'implémentation en Java de mécanismes de migration de niveau 2 est possible, mais nécessite de modifier la machine virtuelle de Java. A notre connaissance, seul le système Sumatra ([Acharya 96]) réalise ce niveau en Java.

Dans la suite, le terme « agent mobile » se réfère exclusivement à des agents mobiles de type 1 ou 2.

4.2.5.3 Les agents mobiles

Un agent mobile est une unité d'exécution qui possède ses propres ressources et sa propre expertise.

Dans le cas d'une architecture « orientée objet » (la quasi totalité des architectures existantes le sont), un agent est instance d'une classe « Agent », ou d'une de ses sous-classes. Ses ressources propres sont ses attributs, et son expertise propre est constituée de ses comportements.

La classe « Agent » définit les comportements de base que tout agent possède. Quatre comportements sont à distinguer, car ils marquent les transitions importantes dans le cycle de vie d'un agent mobile (Figure 32) :

agentInit()	Initialisation de l'agent
agentRun()	Comportement de l'agent
agentEnd()	Méthode exécutée avant la mort de l'agent
agentAwake(int cpte-rendu)	Méthode exécutée après une migration

Note : les noms exacts des méthodes sont ceux des agents JavaNetAgents (cf §6). Ils diffèrent d'une plate-forme d'exécution d'agents mobiles à une autre, mais leur sens est similaire. Cette similarité est justifiée par des choix communs dans l'implémentation, par exemple l'utilisation du langage Java et l'exploitation de ses spécificités. Les détails techniques sur la classe « Agent » sont donnés dans la partie consacrée à JavaNetAgents.

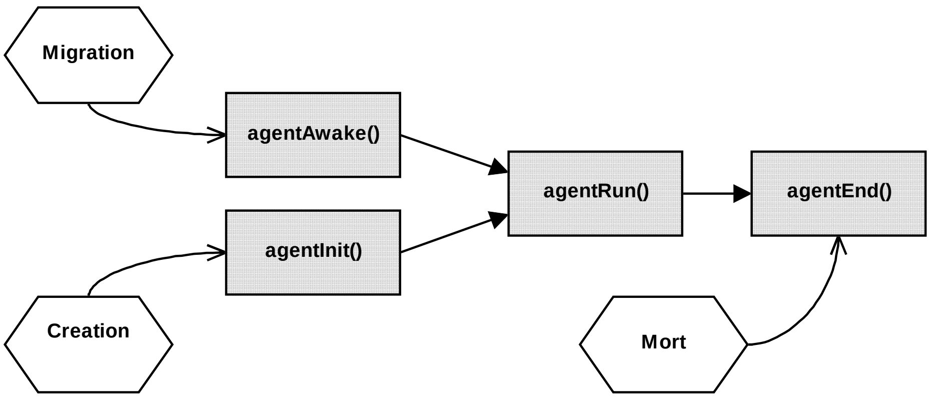


Figure 32. cycle de vie d'un agent.

La méthode `agentAwake(int cpte-rendu)` est le point de réveil d'un agent après la migration. L'entier passé en paramètre indique la manière dont s'est déroulée la migration (succès, échec, etc.).

En plus de ces comportements de base, un agent peut disposer d'autres comportements (qui sont définis dans les sous-classes de la classe « Agent »). Ces comportements correspondent à l'expertise propre de l'agent.

Les activités d'un agent correspondent à ses comportements actifs à l'instant c courant. Ce sont les méthodes « instanciées » et en cours d'exécution. La notion d'activité est importante, car au moment de la migration de l'agent, ses activités courantes sont suspendues et sont parfois relancées (cela dépend du type de migration mis en oeuvre). Les détails de la procédure de migration sont décrits dans un paragraphe ultérieur.

4.2.5.4 Les places

Une place est un environnement d'exécution pour agents mobiles ([Telescript95]). Elle accueille des agents mobiles et leur fournit des ressources CPU. Les places sont reliées les unes aux autres par un réseau de communication, cet ensemble forme une architecture à base d'agents mobiles (Figure 33).

Chaque place fournit également aux agents qu'elle héberge (Figure 34) :

- un accès au ressources locales (e.g. fichiers, sous réserve qu'il ait les droits d'accès suffisant), sous condition que les agents aient les droits d'accès suffisants. Les droits d'accès sont gérés par la place, qui les attribue de manière personnalisée à chaque agent lorsqu'il arrive chez elle. Les droits attribués à un agents peuvent dépendre de sa place d'origine, de son propriétaire « humain », etc.
- un accès aux autres agents locaux résidant sur la place : les agents locaux.

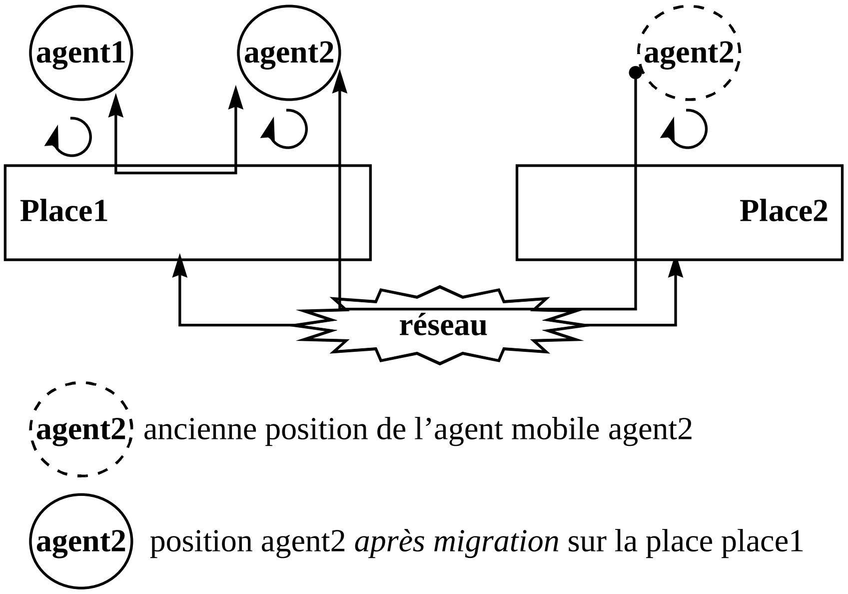


Figure 33. Architecture à base d'agents mobiles.

Enfin il faut noter que chaque place cache l'hétérogénéité des machines hôtes. De ce point de vue une place est très proches d'une machine virtuelle. C'est pour cela que des langages « à base de machines virtuelles » comme Java ou Smalltalk s'avèrent particulièrement utiles pour l'implémentation d'agents mobiles.

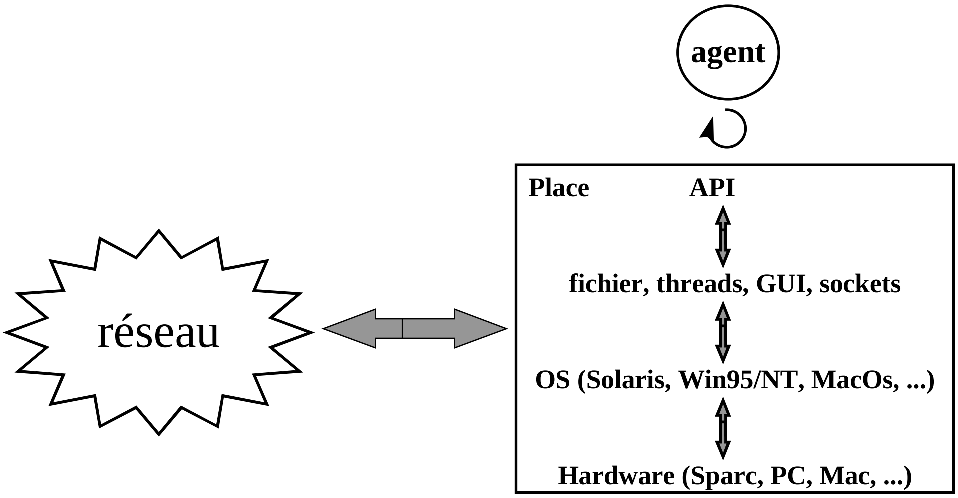


Figure 34. Structure d'une place.

4.2.5.5 Exemples d'utilisation

La technologie des agents mobiles est en plein essor actuellement. Son jeunesse ne lui permet pas encore d'être utilisée sur des systèmes commerciaux, mais des expérimentations « en grandeur nature » —par opposition à des simples « applications jouets »— ont déjà été réalisées. Parmi celles-ci, peuvent être cités :

- traitements de données dans les bases de données multimédia distribuées ([Oracle95]). Un agent mobile se déplace de place en place, où il interroge des bases de données multimédia locales. De par la nature multimédia des informations stockées dans ces bases de données, chaque requête renvoie une important quantité d'informations. Toutefois, l'agent procède *localement* à leur traitement, ce qui économise d'autant la bande passante nécessaire.
- la supervision de réseau ([Tennenhouse et al. 96]). Chaque noeud du réseau dispose d'une (petite) place d'accueil d'agent mobile. Chacune de ses places accueille un agent mobile, envoyé par le superviseur humain du réseau et chargé de superviser le noeud où il se trouve. L'utilisation d'agents mobiles permet une forte économie en bande passante et une meilleure dégradation des performances en cas de surcharge réseau, et une plus grande tolérance aux pannes.
- la prise en compte des usagers mobiles ([Nielsen 95]). Les agents mobiles facilitent la configuration du poste de travail d'un utilisateur itinérant. Cet utilisateur se déplace physiquement, des postes de travail et se connecte donc régulièrement à des points différents du réseau.

Toutefois il est important de noter l'absence de résultats expérimentaux publics validant ces affirmations. Cette lacune est sans doute due à la jeunesse de cette technologie. Les seuls résultats concrets à ma connaissance sont ceux d'un expérience de gestion de réseau SNMP[21]

[21] Le problème de la gestion de réseau SNMP est identique à celui de la recherche d'informations dans une base de données distribuée.

réalisée avec des agents mobiles (utilisant la plate-forme Aglets d'IBM, [Lange et al. 96]). Ces résultats montrent :

- une considérable amélioration de la consommation en bande passante d'un facteur 1000 à 100000.
- une forte réduction du taux de pannes du réseau, due précisément à la diminution importante de la consommation en bande passante.

La conclusion de cette expérience —aux dires de ces auteurs— est double est extrêmement intéressante et double : OUI la mobilité est intéressante, MAIS *elle est un simple outil d'implémentation, qui —en tant que tel— doit « servir » un algorithme décentralisé efficace*.
En l'occurrence les agents Aglets implémentaient un algorithme décentralisé de jointure de base de données, qui réduit fortement la quantité de données à transmettre entre deux noeuds pour réaliser la jointure de tables situées sur ces noeuds. Les qualités résultantes du réseau sont à mettre sur le compte de cette algorithme, au moins autant que sur celui de la mobilité, si ce n'est plus.
Dans un avenir proche, d'autres expérimentations devraient sans aucun doute corroborer ces résultats.

4.2.6 Couplages avec OMA/CORBA

Les agents mobiles sont en fait complémentaires des architectures « traditionnelles » de systèmes distribués comme OMA/CORBA. En effet, un agent mobile est en fait un objet qui est placé dans un environnement d'exécution distribué (cf. §4.2.4.3). C'est pourquoi l'agent mobile a besoin des services définis (et fournis) par OMA/CORBA : service courtage, service nommage, service persistance, etc.
Les architectures à base d'agents mobiles gagneraient ainsi à être couplées à OMA/CORBA. Ce couplage est réalisé par l'encapsulation dans un objet-application CORBA, de chaque place d'accueil d'agents mobiles. Les agents mobiles accèdent aux services communs OMA/CORBA par l'intermédiaire d'agents stationnaires, qui résident de manière permanente sur chaque place. Ces agents représente les serveurs OMA et fournissent chacun un service OMA/CORBA (Figure 35) : un agent trader représente le trader OMA/CORBA, un agent Workflow représente le service Workflow, etc. Ces agents ne migrent jamais : même s'ils en ont la capacité, leur migration n'a aucun intérêt compte tenu du service qu'ils fournissent.

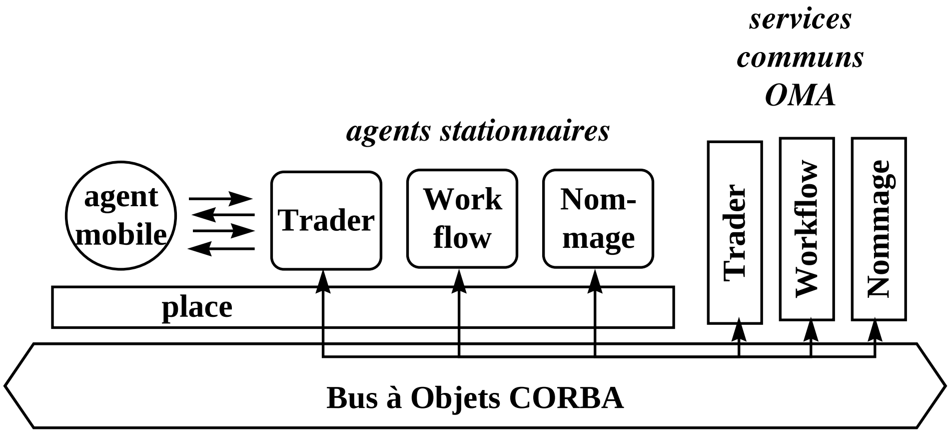


Figure 35. Intégration de la mobilité dans l'architecture OMA/CORBA.

Cependant l'intégration de la mobilité dans une architecture OMA/CORBA nécessite l'adaptation de certains services, et notamment le service nommage : ce service doit être notifié à chaque fois qu'un agent mobile change d'adresse en migrant, et mettre à jour sa table d'adressage.

L'intégration des agents mobiles à l'architecture OMA/CORBA pose donc quelques problèmes. Néanmoins elle est d'un si grand intérêt que l'OMG a diffusé un appel à propositions sur ce sujet (RFP MASIF, *Request For Proposals on Mobile Agent Facility*, [Milojicic et al. 98]).

4.2.7 Conclusion

Les agents mobiles constituent un outil d'implémentation particulièrement intéressant pour les SMA de type SCDO.

Leur utilisation présente en effet de nombreux avantages dans les conditions d'exploitation caractéristiques de ces systèmes :

- flexibilité : la mobilité de code permet d'enrichir les différents composants du système, durant la phase d'exploitation et sans suspendre leurs activités.
- économie en bande passante : les procédures sont acheminées là où se trouvent les données, et non l'inverse[22]. Cela permet des économies importantes dans le cas de manipulation de grosses quantités de données (des facteurs de 10000 à 100000 ont été observés dans des expériences de supervision de réseau SNMP)
- équilibrage de charges : la mobilité permet la mise en œuvre d'algorithmes décentralisés qui répartissent les calculs sur le réseau.

---

[22] Les niveaux de migration 1 et 2 permettent le transfert des procédures près des données. Le niveau 0 ne le permet pas, car les procédures doivent être pré-installées partout où elles sont utilisées. Le niveau 2 permet en outre le transfert d'une procédure en cours d'exécution (avec préservation des valeurs des variables temporaires), toutefois cette fonction n'est pas toujours nécessaire. C'est pourquoi le niveau 1 est le plus souvent suffisant.

- transaction en mode déconnecté : dès qu'il est créé par l'utilisateur, l'agent migre sur un autre noeud du réseau (il ne reste pas sur l'ordinateur de l'utilisateur) et exécute le traitement pour le compte de l'utilisateur, même lorsque celui-ci est déconnecté du réseau.
- tolérance aux pannes : la mobilité permet la mise en œuvre d'algorithmes qui répartissent les calculs sur le réseau et qui résistent ainsi mieux aux pannes d'une machine ou d'un lien de communication.
- vitesse : la diminution des échanges de messages distants se traduits également par une augmentation de la vitesse du traitement, car les temps de communications sont importants dans les réseaux à forte dispersion géographique.

Cependant, il faut garder à l'esprit que les agents mobiles ne sont qu'un outil d'implémentation, et que nombre des qualités qui viennent d'être énumérées dépendant également des algorithmes mis en oeuvre. Par exemple la tolérance aux pannes, et l'économie en bande passante dépendent essentiellement de la nature décentralisée de l'algorithme. Mais les agents mobiles sont très appropriés, voire même indispensables pour une implémentation flexible d'algorithmes décentralisés, car il faut alors mettre à jour régulièrement des procédures qui sont dispersées dans le réseau. Les techniques traditionnelles, sans mobilité de code, ne permettent pas l'implémentation efficace de ces algorithmes.

Toutefois nous ne prétendons pas que la technologie des agents mobiles est la panacée pour implémenter des systèmes distribués. [Chess et al. 95] présente une intéressante confrontation entre les agents mobiles et les technologies plus traditionnelles des systèmes distribués (comme celles mises en œuvre dans l'architecture OMA/CORBA). Le développement d'applications à base d'agents mobiles, actuellement encore peu nombreuses à cause de la jeunesse de cette technologie, permettra sans doute de préciser davantage le « domaine d'efficacité » des agents mobiles.

Le principal inconvénient de la migration d'agents est la nécessité de mécanismes de « surveillance » des agents, pour garantir un certain niveau de sécurité des noeuds hôtes ([Tardo et al. 95]). La sécurité est assurée par chaque place, en utilisant deux types de techniques :

- les techniques cryptographiques, garantissant la confidentialité, l'authentification et la certification de code.
- des techniques plus dépendantes de l'implémentation. En Java par exemple, des mécanismes de sécurité sont implémentés « en natif » dans la machine virtuelle. Ils permettent d'associer à chaque agent des droits d'accès aux ressources locales (fichier, sockets réseau, threads d'exécution, etc.).

4.3 Conclusion générale.

Les modèles traditionnels de contrôle d'agent ne tiennent pas compte des relations de dépendance entre agents. Le modèle des dépendances sociales de Castelfranchi fait exception, mais il ne tient pas compte de la corrélation entre liens de dépendance, dont l'importance essentielle a été identifiée au §3.5. Ces modèles ne conviennent donc pas pour l'étude du problème traité dans cette thèse. Dans la partie suivante, nous introduisons un mécanisme d'interaction qui prend en compte la corrélation des liens de dépendance.

Les architectures à base d'agents mobiles présentent de bonnes qualités pour l'implémentation des SMA de type SCDO. Nous les utiliserons donc pour réaliser le système de l'Agence de Voyage, au §7. Toutefois, en raison de la très grande jeunesse de la technologie de mobilité de code, aucune plate-forme d'exécution d'agent mobile n'existait au début de cette thèse, à part Telescript (très lourd et difficile à manipuler. Cf. [Telescript95]), et Mole (non utilisable à l'ONERA[23]. Cf. [Straßer 96]). C'est pourquoi la plate-forme JavaNetAgents a été développée, elle est décrite au §6.

---

[23] Mole nécessite un serveur Web qui donne l'accès à l'espace disque où se trouve le bytecode Java des agents mobiles. Un tel serveur ne pouvait pas être installé à l'ONERA pour des raisons de sécurité.

5. Interaction par agents mobiles

Ce chapitre décrit un mécanisme d'interaction efficace dans les conditions d'exploitation des SMA de type SCDO. Ses principales caractéristiques sont sa forte décentralisation et son implémentation par des agents mobiles.

Ce chapitre débute par un bref rappel de la fonction de l'interaction et des raisons pour lesquelles la conception d'un tel mécanisme est un problème. Le déroulement complet de l'interaction est ensuite décrit. Le §5.10 analyse les qualités et les défauts du mécanisme proposé, et en propose une extension encore plus décentralisée.

5.1 Contexte - Rappel

L'agent client souhaite obtenir un ensemble de services (s1, s2, ..., sk) fournis par d'autres agents du système, les agents serveurs. Pour des raisons de simplicité, nous supposerons que chaque service est fourni par un agent différent.

L'agent client initie une interaction pour obtenir la fourniture de ces services. L'interaction est réalisée en plusieurs étapes :

1. identification de groupes d'agents serveurs.
2. sélection et activation d'un des groupes identifiés.
3. remplacement du groupe sélectionné par un autre, au cas où un de ses membres est défaillant.

La difficulté de la réalisation d'un mécanisme d'interaction est due à plusieurs facteurs essentiels :

- les services sont liés par une relation de compatibilité R, définie comme un réseau de contraintes dont les variables sont les services. Cette corrélation empêche la recherche, et l'obtention des services « isolés », i.e. *considérés indépendamment les uns des autres*. Ce point est approfondi en commentaires (§5.10.1).
- les agents sont dispersés sur un réseau et l'agent client ne connaît pas les comportements de tous les agents du système. De plus, des pannes machines/réseau peuvent rendre les communications difficiles voire impossibles. Un mécanisme d'interaction efficace

minimise les communications distantes. Il minimise également les traitements centralisés, qui sont autant de « points de rupture » fatals pour l'interaction : l'interaction échoue si la machine exécutant un de ces traitement tombe en panne, ou si elle est isolé du réseau à la suite d'une panne de communication.

- les agents eux-mêmes peuvent être victimes de défaillances et ne plus pouvoir fournir le service convenu avec l'agent client, i.e. remplir son rôle dans le groupe. Dans ce cas, l'interaction doit trouver un groupe de remplacement.

5.2 Déroulement général de l'interaction

Le mécanisme d'interaction proposé utilise la technologie des agents mobiles. Les agents mobiles permettent la décentralisation de la majeure partie de l'interaction, et lui confèrent de ce fait certaines qualités détaillées plus loin.

Pour décrire le mécanisme d'interaction avec plus de clarté, les trois étapes de son déroulement sont décomposées en plusieurs sous-étapes comme illustré sur la Figure 36.

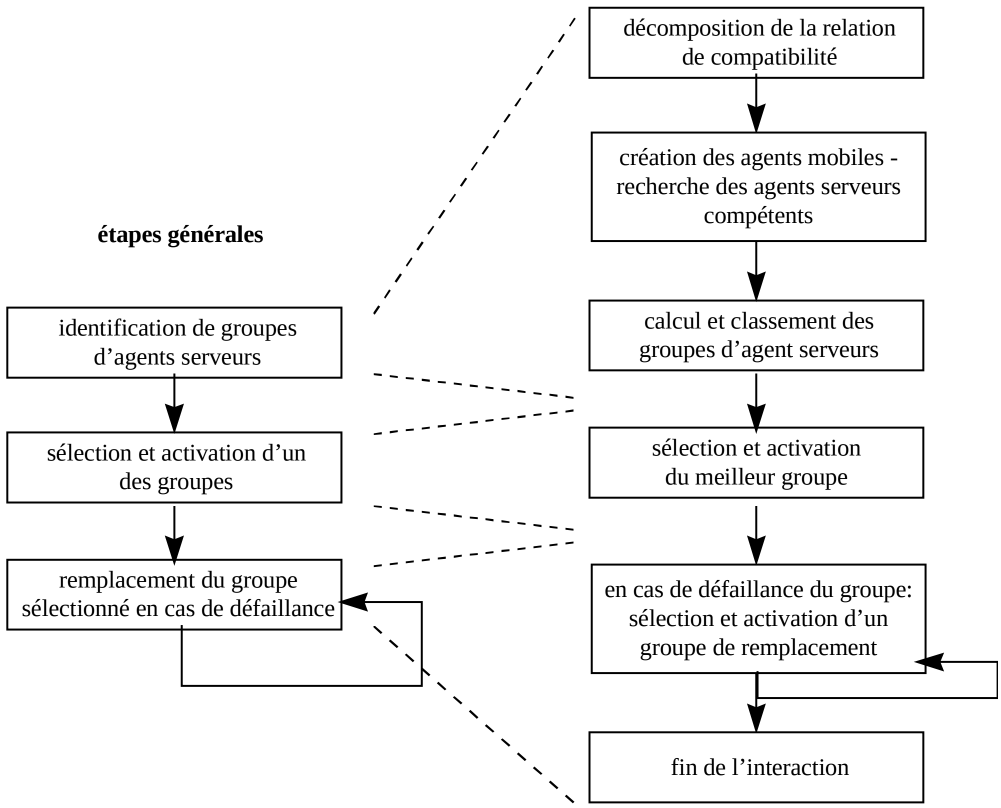


Figure 36. Déroulement d'une interaction.

L'utilisation d'agent mobiles confère certaines qualités au mécanisme d'interaction :

- la réaction aux défaillance est décentralisée. L'agent client n'intervient pas pour choisir et faire activer le groupe de remplacement. Aucune communication n'a lieu avec l'agent client durant cette phase. Cela réduit le délai de réaction, ainsi que le coût en bande passante. Cette phase est également résistance aux pannes de la machine accueillant l'agent client et de lignes de communication reliant cette machine au réseau.

- la recherche d'agent serveurs peut utiliser de nombreux protocoles de diffusion d'information dans un « réseau actif » [Tennenhouse et al. 96]. Dans ces réseaux les messages sont encapsulés dans des « agents mobiles » qui exécutent certaines opérations à leur passage sur les noeuds du réseau. En l'occurrence, le message est ici un appel d'offre pour la fourniture d'un service. L'utilisation d'un agent mobile (ce qui est identique à un objet mobile, cf. §4.2.5.3) permet une grande flexibilité dans les protocoles de diffusion du message, car le protocole est codé dans l'agent mobile lui-même, et non pas dans chaque noeud du réseau. Chaque message peut ainsi disposer de son propre protocole de diffusion (broadcast, multi-cast, diffusion épidémique [Demers et al. 87]), selon le type d'agent mobile qui l'encapsule. Il peut même en changer dynamiquement en cours de diffusion : il lui suffit de changer d'agent mobile « véhicule ». Malheureusement par manque de temps, cette fonctionnalité n'a pas pu être pleinement exploitée dans l'implémentation de l'Agence de Voyage : une seule classe d'agent mobile a été définie, et donc tous les message d'appel d'offre sont diffusés selon le même protocole.

5.3 Convention graphique adoptée dans le reste du chapitre.

Les signes utilisés dans les figures contenues dans ce chapitre sont rappelés par la Figure 37.

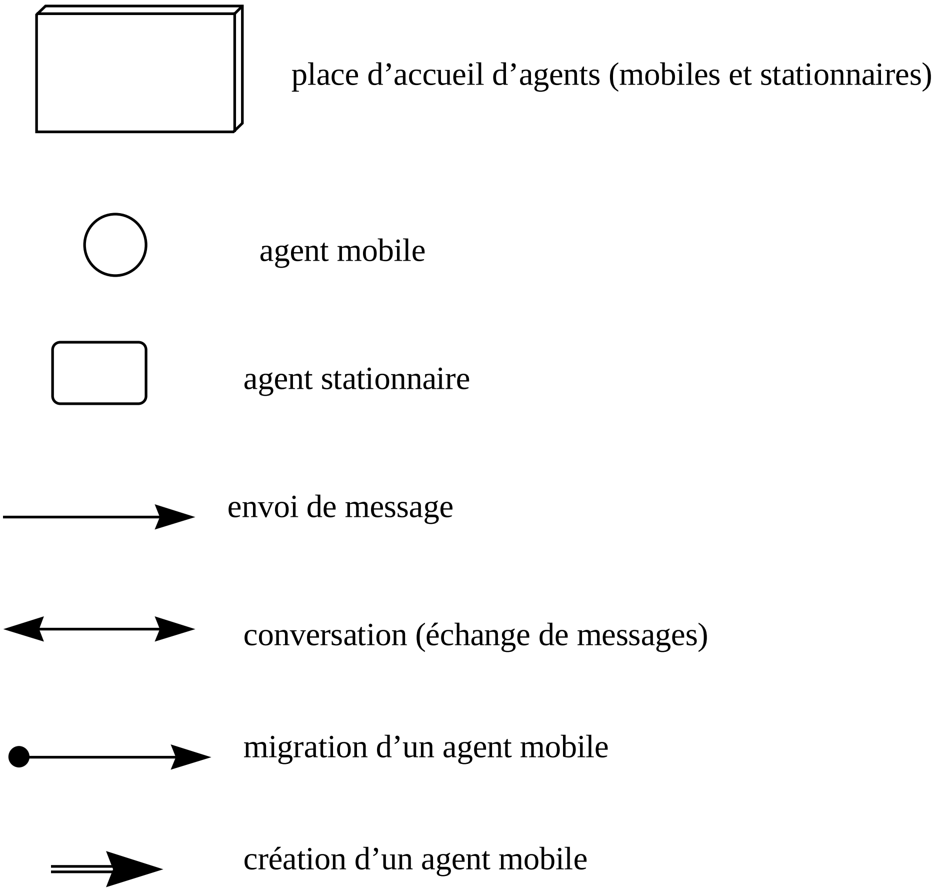


Figure 37. Convention graphique adoptée dans ce chapitre.

5.4 Décomposition de la relation de compatibilité

Dans cette étape, l’agent client décompose la relation de compatibilité pour obtenir des cahiers des charges portant les k services à rechercher.

La relation de comptabilité, notée R, traduit la corrélation entre les différents services dont l’agent client à besoin. Elle est définie par un réseau de contraintes, portant sur les paramètres des différents services.

Afin de définir des cahiers des charges portant sur chaque service indépendamment des autres services recherchés, R est décomposée en une conjonction de contraintes mono-service (notées R_i $1 \leq i \leq k$), et d’une contrainte (notée $R_{corrélation}$) liant effectivement tous les services :
$R(s1, s2, ..., sk) = R_1(s1) \wedge R_2(s2) \wedge ... \wedge R_k(sk) \wedge R_{corrélation}(s1, s2, ..., sk)$
Chaque contrainte mono-service porte sur un service indépendamment des autres : c’est une conjonction de contraintes unaires portant sur la variable représentant ce service. La contrainte exprime la corrélation entre les services : c’est une conjonction de contraintes n-aires où $2 \leq n \leq k$.

Dans l'exemple de l'Agence de Voyage, la relation de contrainte est :
$R(s1, s2) = R_1(s1) \wedge R_2(s2) \wedge R_{corrélation}(s1, s2)$
où
$R_1(s1) = (s1.type=\text{TRANSPORT}) \wedge (s1.lieuDep = Dijon) \wedge (s1.dateDep>12juin16h00) \wedge (s1.lieuDest=Paris)$

$R_2(s2) = (s1.type=\text{TRANSPORT}) \wedge (s2.lieuDep=Paris) \wedge (s2.lieuDest=NY) \wedge (s2.dateDest<13juin08h00)$

$R_{corrélation}(s1, s2) = (s1.dateDest<s2.dateDep) \wedge (s1.prix+s2.prix<7KF)$

Les contraintes R_i définissent des cahiers des charges décorrélés sur les différents services. Ils sont utilisés dans les appels d'offres transmis aux agents serveurs. Les réponses des serveurs, des propositions de services, vérifient ces cahiers des charges. Une fois les propositions de services récupérées, elles seront bien sûr appariées selon la contrainte $R_{corrélation}$ (cette étape est réalisée par l'agent client).

5.5 Création des agents mobiles - Recherche des agents serveurs compétents

Cette étape est illustrée par la Figure 38, et se décompose comme suit :
1. l'agent client crée un agent mobile pour chaque service à obtenir. Chaque agent mobile dispose d'une contrainte R_i, et il est chargé de d'identifier les services satisfaisant cette contrainte.
2. les agents mobiles se déplacent sur le réseau, à la recherche d'agent serveurs capables de fournir ces services. Pour ce faire, ils interrogent des traders en leur indiquant le type de service qu'ils recherchent (par exemple TRANSPORT). Les traders répondent en donnant une liste d'adresses de serveurs compétents.
3. ils interrogent les serveurs indiquées par les traders, en leur soumettant un appel d'offres dont le cahier des charges est R_i. Ils récupèrent une liste de propositions de services, qu'ils renvoient finalement au client.

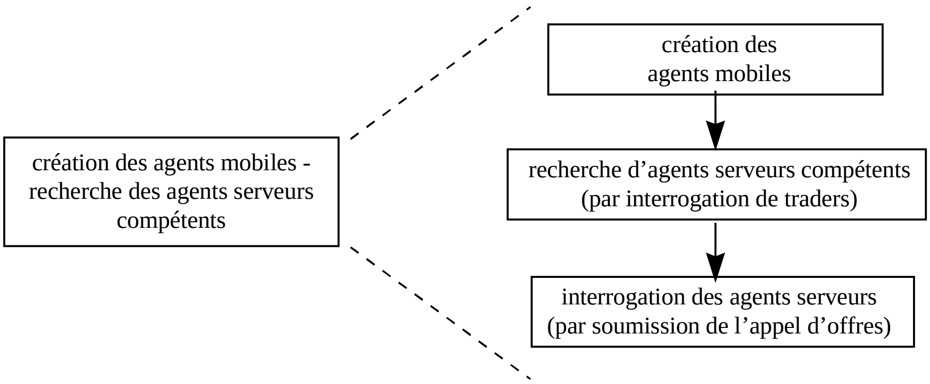


Figure 38. Recherche des agents serveurs compétents.

La Figure 39 illustre la recherche et l'interrogation d'agents serveurs sur une place contenant un trader : (1) l'agent mobile arrive sur la place, (2) il interroge le trader qui lui indique les agents serveurs compétents résidant sur la place, (3)-(4) l'agent mobile interroge successivement tous les agents serveurs—supposons ici que AirFrance et TWA résident sur cette place—, puis (5) il migre vers une autre place du réseau.

Note : c'est exemple n'est pas réaliste et n'a qu'un rôle d'illustration du comportement des agents mobiles. En effet, en pratique les serveurs AirFrance et TWA sont situés sur des places distinctes (chaque serveur « administre » sa propre place d'accueil, où vienne les agents mobiles qui souhaitent négocier avec lui). La présence du trader sur la même place que les serveurs s'explique par l'architecture particulière de la plate-forme JavaNetAgents, sur laquelle l'application « Agence de Voyage » est construite : chaque place d'accueil d'agents mobile dispose d'un trader « local », qui répertorie la liste des serveurs situés sur cette place (cf. §5.5.1).

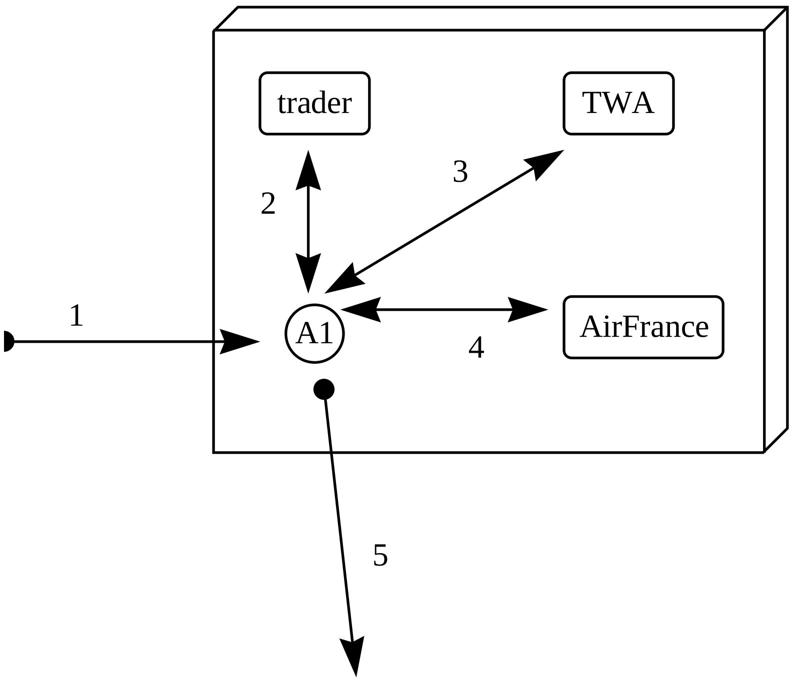


Figure 39. Recherche et interrogation d'agents serveurs compétents.

A la fin de cette étape, les agents mobiles restent sur la dernière place qu'ils ont visitée. Chacun envoie à l'agent client la liste de propositions de services qu'il a obtenu par interrogation successive des agents serveurs compétents. Puis tous les agents mobiles attendent la réponse de l'agent client (cf. l'étape suivante §5.6).

Plusieurs remarques sont à faire concernant le déroulement de cette étape, concernant le parcours des agents mobiles à la recherche d'agents serveurs compétents d'une part, et le protocole d'appel d'offres utilisé pour interroger les serveurs d'autre part.

5.5.1 Parcours des agents mobiles à la recherche d'agents serveurs compétents

Le problème du parcours des agents mobiles étant identique à celui de la diffusion d'un message d'appel d'offres dans un réseau, nous n'avons pas innové dans ce domaine et nous sommes contentés de reprendre les techniques utilisées dans les systèmes distribués. En l'occurrence, deux principales techniques de recherche existent :

- parcours exhaustif du réseau et interrogation de tous les agents : l'agent se déplace de place en place, et interroge tous les agents sur chaque place. Cette technique est aussi simple qu'inefficace (elle est très coûteuse en bande passante).
- utilisation du service de courtage, si celui-ci est fourni par la plate-forme d'exécution d'agents mobiles utilisée. Les courtiers (*trader* ou *directory facilitator* en anglais) sont présents dans certaines plates-formes d'exécution d'agents mobiles (c'est le cas en JavaNetAgents, mais pas en Voyager par exemple[24]). Ils fournissent un service « d'annuaire page jaune », en répertoriant les comportements des agents situés dans leur zone de contrôle. La zone de contrôle d'un courtier est typiquement, soit le système tout entier (dans ce cas le courtier est unique et central), soit la place (c'est la solution adoptée dans JavaNetAgents[25]). Dans ce dernier cas, chaque place dispose d'un courtier, qui connaît les comportements des agents « locaux ». L'agent mobile se déplace alors sur toutes les places du système, mais n'interroge que les agents répertoriés par le trader pour le type de service qu'il recherche.

Des techniques plus sophistiquées existent et nécessitent la mise en place de fédérations de courtiers organisées dans une hiérarchie (d'autres structures sont aussi utilisées). Elles dispensent les agents mobiles de parcourir toutes les places. Par manque de temps, de telles fédérations de courtiers JavaNetAgents n'ont pu être réalisées.

5.5.2 Protocole d'appel d'offres

Le protocole de la conversation entre un agent mobile et un agent serveur est très simple et ne comporte que deux étapes (Figure 40, l'agent serveur est SNCF sur cette figure) :

[24] Dans ce cas, l'implémentation et l'installation des courtier est à la charge du concepteur du SDCO.

[25] Dans JavaNetAgents, le courtier est implémenté sous la forme d'un agent stationnaire, i.e. un agent non mobile qui réside en permanence sur la même place d'accueil. Le fait que le service de courtage soit fourni par un agent, au lieu d'être un simple appel système à la plate-forme d'exécution, n'a ici aucune importance.

1. l’agent mobile initie la conversation en transmettant un appel d’offre à l’agent serveur. Cet appel d’offre contient le cahier des charges décorrélé portant sur le service recherché par l’agent mobile (la contrainte R_i).
2. l’agent serveur répond en donnant une liste de services satisfaisant ce cahier des charges. La liste peut être vide dans le cas où le serveur n’est pas compétent.

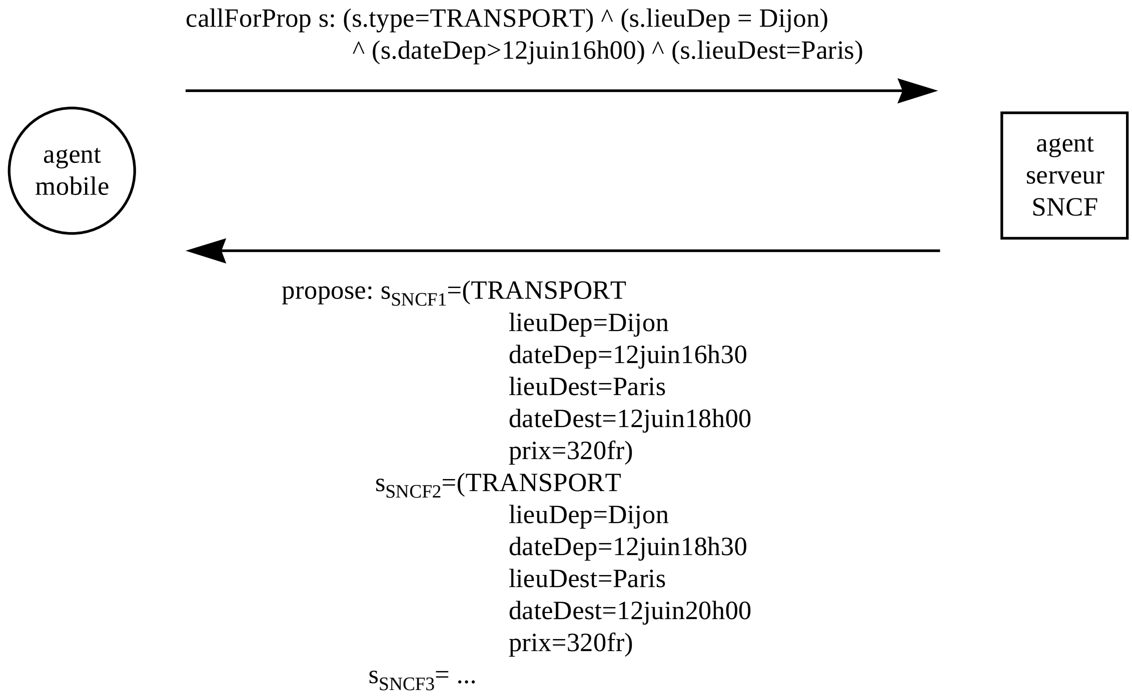


Figure 40. Protocole d’appel d’offres avec les agents serveurs

Plusieurs points distinguent le protocole de la conversation entre l’agent mobile et l’agent serveur, de celui du classique ContractNet :

- l’agent mobile (qui représente le « client » du ContractNet) ne sélectionne aucune des propositions faites par l’agent serveur. En effet, ces propositions doivent être d’abord comparées aux proposions collectées par les autres agents mobiles (cf. §5.6) avant qu’un choix ne soit fait.
- un seul serveur est impliqué dans cette conversation, tandis que le ContractNet implique typiquement plusieurs serveurs. En fait l’agent mobile initie plusieurs fois ce type de conversation, avec des serveurs différents à chaque fois. Il apparaît que **ce choix de conception (distinguer les conversations « 1-1 » entre le « client » et chaque serveur) facilite grandement l’implémentation des conversation.**

Dans l’implémentation de l’Agence de Voyage en JavaNetAgents, les agents mobiles réalisent un parcours exhaustif du système (tout en s’aidant des courtiers pour ne pas interroger tous les agents). La phase « recherche des serveurs compétents » s’arrête donc lorsque tous les agents mobiles ont parcouru l’ensemble des places du systèmes. Ce critère d’arrêt est critiquable puisque la taille d’un SMA de type SCDO est variable et *a priori* inconnue. C’est pourquoi un autre critère d’arrêt est rajouté : les agents mobiles s’arrêtent lorsqu’ils ont trouvé un nombre prédéterminé de propositions (dans l'implémentation de l'Agence de Voyage, le nombre est fixé à 10).

5.6 *Calcul et classement des groupes*

Dans cette étape, seul l'agent client est actif. Les agents mobiles attentent, immobiles, sur la dernière place visitée.
L'agent client effectue les opérations suivantes (résumées sur la Figure 41) :

1. l'agent client utilise un algorithme de satisfaction de contraintes pour apparier les différents services et former des groupes de services mutuellement compatibles. L'algorithme utilisé est quelconque car cette opération est exécutée par l'agent client, donc de manière centralisée. Au §5.10.3 est discuté l'utilisation d'algorithmes décentralisés pour pallier ce défaut.
2. l'agent client construit une liste de groupes, classés par ordre de préférence. Dans le cas de l'Agence de Voyage, les groupes les plus rapides (i.e. ceux dont la durée totale du trajet est la plus faible) sont privilégiés. A rapidité égale, les groupes les moins chers sont privilégiés.
3. l'agent client envoie cette liste aux agents mobiles, afin qu'ils procèdent (à l'étape suivante) à l'activation du meilleur groupe (le groupe placé en tête de la liste).

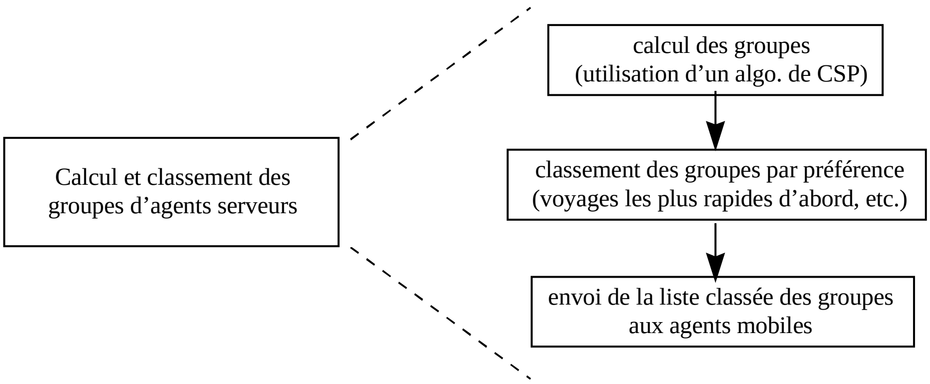


Figure 41. Calcul des groupes d'agents serveurs.

5.7 *Activation du meilleur groupe*

Dans cette étape, les agents mobiles procèdent à l'activation du meilleur groupe d'agents serveurs. Cette étape est illustrée par la Figure 42 (cette figure montre les opérations effectuées par chaque agent mobile pour obtenir le service dont il a la charge).

Chaque agent mobile examine le premier groupe de la liste et identifie l'agent serveur fournissant le service dont il a la charge. Puis il migre sur la place où réside cet agent et initie avec lui une négociation[26] pour obtenir la fourniture du service. L'agent mobile reste sur cette place pendant toute la durée de fourniture du service, afin d'être informé —et de gérer— toute éventuelle défaillance (voir étape suivante).

Une fois que tous les agents mobiles ont obtenu la fourniture de chaque service, le groupe d'agent serveur est actif : **l'EV est créé et fournit les services nécessaires à l'agent client**.

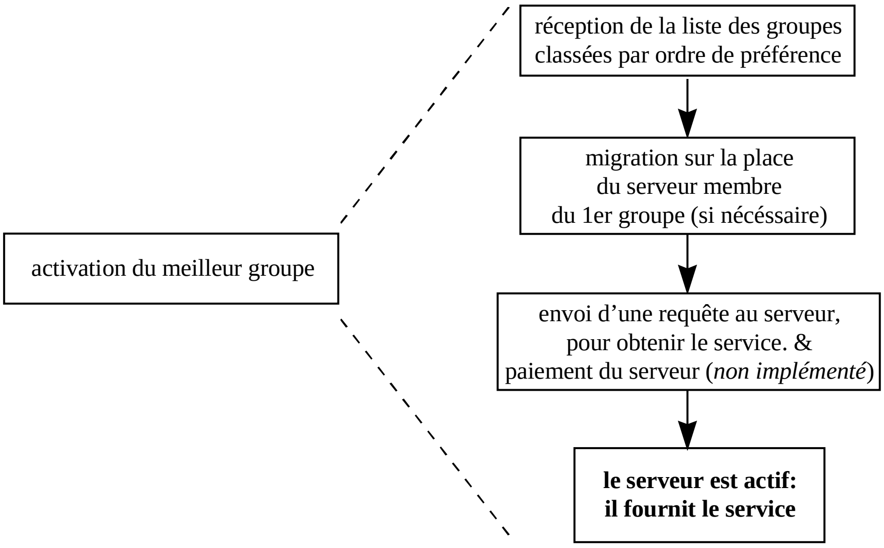


Figure 42. Procédure suivie par chaque agent mobile pour activer le meilleur groupe.

Dans l'exemple de l'Agence de Voyage (illustré sur la Figure 43), l'EV contient SNCF, qui a pour rôle (i.e. fournit le service) s_{SNCF1}, et AirFrance, qui a pour rôle (i.e. fournit le service) s_{AF1}.

[26] La négociation est en fait un simple ordre d'achat du service : l'agent mobile paie le serveur et le serveur fournit en retour le service.

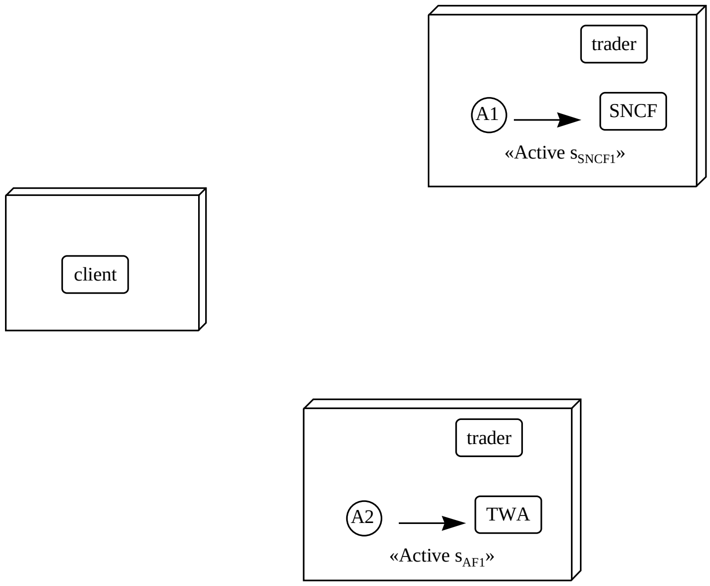


Figure 43. Activation du groupe d’agents serveurs (SNCF s_{SNCF1}, AirFrance s_{AF1}).

5.8 Réaction à la défaillance du groupe actif

Dans cette étape, les agents mobiles réagissent à la défaillance d’un des agents membres du groupe actif (activé à l’étape précédente), et procèdent à l’activation d’un groupe de remplacement.

La défaillance d’un agent serveur membre du groupe actif (i.e. celui qui a été activé à l’étape précédente), rend ce groupe « ineffectif » (il ne satisfait plus les besoins de l’agent client). L’interaction doit procéder à l’activation d’un autre groupe en remplacement. Cette étape est complètement décentralisée car elle est réalisée intégralement par les agents mobiles sans intervention de l’agent client.

En cas de défaillance d’un agent serveur, l’agent mobile chargé de ce service est averti par l’agent serveur. Cette opération est aisée car l’agent mobile se trouve toujours sur la place de l’agent qui lui fournit le service. Cet agent mobile sélectionne le groupe de remplacement parmi les groupes contenus dans la liste qu’il a reçu de l’agent client : c’est le meilleur groupe qui ne contienne pas le service défaillant. L’agent mobile envoie un message aux autres agents mobiles. Ce message les notifie de la défaillance du serveur, et indique le nouveau groupe sélectionné. Cette procédure est résumée par la Figure 44.

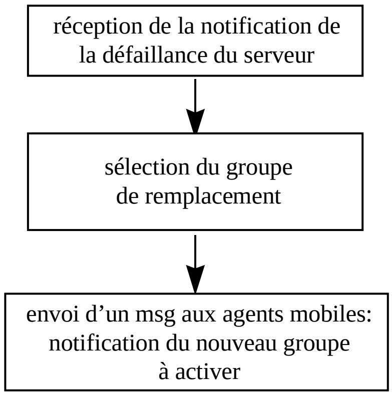


Figure 44. Réaction de l'agent mobile à la notification de la défaillance du serveur associé.

A réception de ce message, chaque agent mobile procède comme à l'étape précédente : il migre sur la place de l'agent serveur fournissant le service dont il a la charge, et négocie avec cet agent la fourniture du service.

Les Figure 45 et Figure 46 illustrent ce processus sur le scénario de l'Agence de Voyage : lorsque AirFrance est victime d'une défaillance et désactive prématurément son service s_{AF1} : l'agent mobile A2 est notifié de la défaillance (étape 1 sur la Figure 45), il calcule le groupe de remplacement et le communique à l'agent mobile A1 (étape 2 sur la Figure 45), qui demande à SNCF de désactiver le service s_{SNCF1} (qui ne fait pas partie du groupe de remplacement).

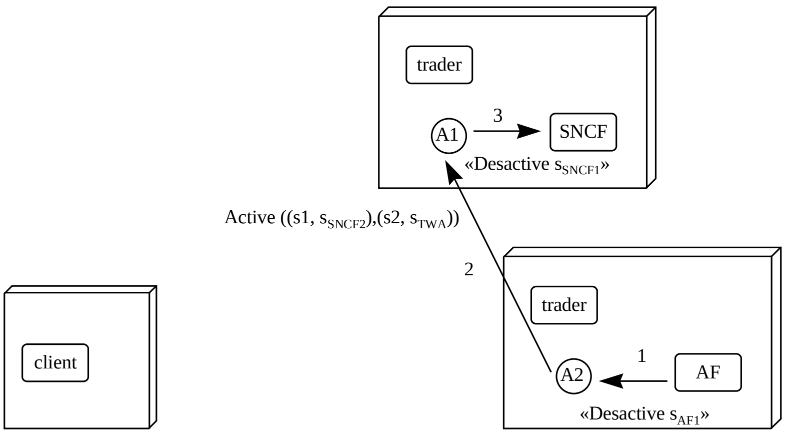


Figure 45. Désactivation du groupe suit à la défaillance de AF.

Sur la Figure 46, l'agent mobile A2 migre sur la place occupée par le serveur TWA (étape 1 sur la figure) afin de négocier avec lui la fourniture du service s_{TWA} (étape 2 sur la figure). Simultanément (étape 2' sur la figure), l'agent mobile A1 négocie avec SNCF la fourniture du service s_{SNCF2} (A1 n'a pas eu besoin de migrer car c'est la même agent serveur qui fournit le service « TRANSPORT Dijon Paris » dans le groupe de remplacement).

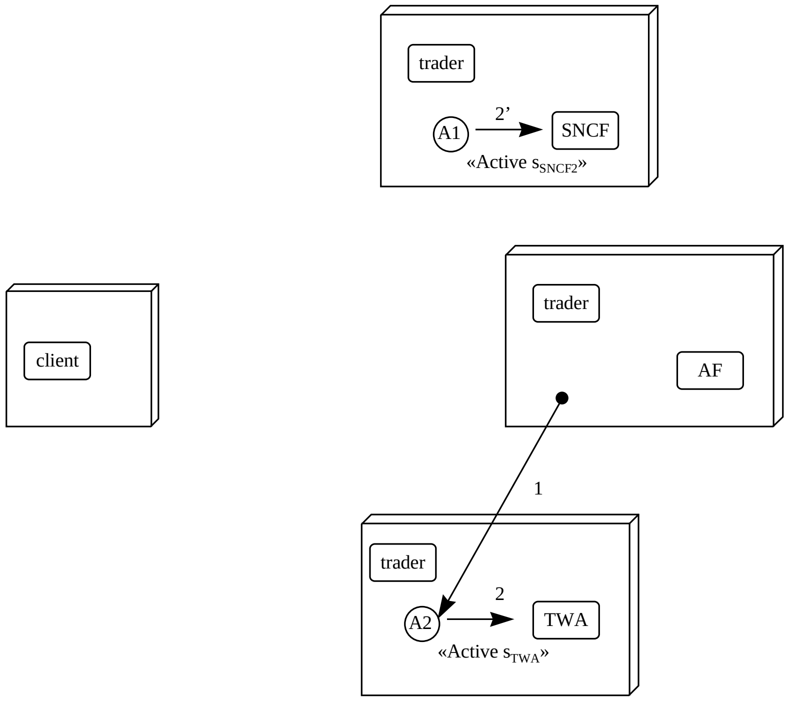


Figure 46. Activation du groupe de remplacement.

Lorsque tous les agents mobiles ont obtenu les services mentionnés dans le groupe de remplacement, alors les besoins de l’agent client sont à nouveau satisfait.

5.9 Fin de l’interaction

Dans cette étape, l’interaction se termine et les agents mobiles désactivent le groupe actif.

L’interaction se termine soit à la demande de l’agent client, soit au bout d’une durée prédéterminée.

L’agent client demande la fin de l’interaction dans le cas où il n’a plus besoin des services fournit par le groupe d’agent serveur. Dans l’exemple de l’Agence de Voyage, ce cas se produit lorsque Dupond décide d’annuler définitivement son voyage. L’agent client envoie un simple message à tous les agents mobiles. Les agents mobiles demandent alors à chaque agents serveur de cesser de fournir leur services.

Certains types de services ont une durée de vie finie. Par exemple les services de transport s’arrêtent quand le client est arrivé à destination. Quand un service cesse d’être fourni dans ces conditions, l’agent mobile envoie un message de notification de fin de service à l’agent client. Le client « attend » que tous les services soient terminés pour considérer que l’interaction est terminée.
A la fin de l’interaction, les agents mobiles deviennent inutiles et cessent d’exister : ils s’auto-détruisent.

5.10 Commentaires

Ce paragraphe présente plusieurs commentaires importants concernant le mécanisme d'interaction proposé. Le mécanisme d'interaction est d'abord comparé au protocole de ContractNet. Puis la généralité et la décentralisation du mécanisme d'interaction, l'intérêt de l'implémentation par agents mobiles sont soulignés. Enfin est examinée l'extension du mécanisme d'interaction par l'utilisation d'algorithmes de satisfaction de contraintes décentralisés et dynamiques.

5.10.1 Comparaison avec le ContractNet.

Une comparaison du protocole d'appels d'offres avec les agents serveurs d'une part, et le ContractNet d'autre part a déjà été faite au §5.5.2. Nous comparons ici le ContractNet au mécanisme d'interaction dans son ensemble.

Le ContractNet est un protocole d'interaction mono-service de type « 1 - 1 » : un client, un serveur[27].
Par comparaison, le mécanisme d'interaction décrit dans ce chapitre peut être vu comme un protocole de négociation multi-services du type « 1 - N » : un client, plusieurs serveurs[28]. La prise en compte de la corrélation des différents services constitue la principale difficulté de la conception d'un tel protocole.

Toutefois, tel qu'il est implémenté, ce mécanise d'interaction peut être considéré comme la combinaison de plusieurs négociations de type ContractNet, synchronisées à l'étape de calcul des groupes : à cette étape en effet, l'agent client est le seul à « avoir la main » (il joue donc le rôle de point de synchronisation) et son calcul des groupes permet de sélectionner des services mutuellement compatibles. Cependant, il est possible de supprimer ce point de synchronisation, en faisant calculer les groupes par l'ensembles des agents mobiles eux-mêmes, sans faire intervenir le client. Cette extension du mécanise d'interaction est intéressante car elle permet de le décentraliser complètement. Elle est discutée en §5.10.3.

5.10.2 Généralité du mécanisme d'interaction

Le mécanisme d'interaction proposé dans ce chapitre fonctionne pour tous types de SMAs de type SCDO. Aucune hypothèse n'est faite sur la nature des services fournis par les agents : les services sont décrits par un type et un ensemble de paramètres quelconques. L'algorithme de CSP utilisé n'impose aucune contrainte sur le domaine de valeurs des paramètres.
Ainsi, bien qu'il soit décrit sur l'exemple de l'Agence de Voyage, ce mécanisme d'interaction est très général.

---

[27] Cette notation ne tient compte que des serveurs effectivement impliqués dans la fourniture d'un service, et non pas de tous les serveurs contactés lors de la phase initiale d'appel d'offres.

[28] Idem. Les serveurs répertoriés sont ceux qui fournissent un service au client.

5.10.3 Décentralisation du mécanisme d'interaction

Le contrôle de l'interaction est décentralisé sauf pour le calcul des groupes (qui est réalisé par l'agent client). Ce point de synchronisation correspond à la prise en compte de la corrélation des services à obtenir.

La décentralisation du mécanisme d'interaction est particulièrement intéressante car elle garantit son extensibilité : ce mécanisme fonctionne aussi bien pour la recherche de deux services que pour cent ou mille, ou un quelconque ordre de grandeur supérieur. De plus, elle assure une certaine résistance aux pannes du réseau de communication, puisque les décisions sont prises « localement », i.e. individuellement par chaque agent mobile, sans nécessiter de concertation (par échanges de messages distants) entre les agents mobiles ou de communication (distante) avec l'agent client.

Il est à noter que l'algorithme présenté ne réagit pas correctement à plusieurs défaillances simultanées : chaque agent mobile recevant un message de notification de défaillance, applique localement la procédure décrite §5.8 (Figure 44) sans tenir compte des autres défaillances (puisqu'il n'en a pas encore été averti). Le groupe de remplacement choisi peut donc contenir un des serveurs défaillants, ce qui ne convient pas. Ce problème peut être aisément résolu par l'utilisation combinée d'une technique de « verrouillage en deux phases » et un classement des agents mobiles par ordre de priorité. Le verrouillage à deux phases permet de vérifier le consensus de tous les serveurs pour un groupe de remplacement. Si plusieurs défaillances ont lieu simultanément, un conflit est détecté par le protocole de verrouillage à deux phases. Dans ce cas, l'agent le plus prioritaire parmi ceux dont les services sont défaillants choisi un groupe de remplacement qui tienne compte de toutes les défaillances, et le protocole de verrouillage à deux phases est appliqué à nouveau pour vérifier le consensus sur ce groupe (i.e. vérifier qu'aucune autre défaillance n'a eu depuis que le conflit à été détecté par la première itération du protocole). Cette technique est répétée jusqu'à ce qu'un consensus soit obtenu sur un groupe de remplacement.

5.10.4 Apports des agents mobiles

En ce qui concerne le choix des agents mobiles pour l'implémentation des interactions de base, il faut noter que l'utilisation de la mobilité n'est pas indispensable : les agents mobiles pourraient rester sur la place du programme client (là où ils ont été créés) et envoyer des messages aux agents situés sur d'autres noeuds du réseau.

Néanmoins, la mobilité apporte de nombreuses qualités à l'implémentation de notre mécanisme d'interaction :

- économie en bande passante : les communications distantes sont réduites car les agents mobiles se déplacent sur les places d'accueil des agents serveurs pour leur transmettre l'appel d'offres et recueillir leurs propositions. Ce gain est surtout notable dans le cas où le cahier des charges est particulier et ne peut être exprimé de manière déclarative (par de

« simples » expressions en SQL par exemple[29]), mais uniquement de manière procédurale[30]. La mobilité de code permet d'amener au serveur, la définition procédurale du cahier des charges.

- résistance aux pannes : des pannes de réseau, coupant momentanément les communications entre le client et les agents mobiles, ne remettent pas en cause le bon fonctionnement du système.
- opérations en mode « déconnecté » : le client peut être situé sur une place connectée au réseau seulement par intermittence. C'est le cas d'une place située sur un ordinateur portable : elle n'est connectée au réseau que lorsque le portable l'est. Le reste du temps, les agents situés sur la place (et le client) ne peuvent communiquer avec des agents sur d'autres places. La mobilité de code facilite ainsi le nomadisme des utilisateurs.

Il est cependant à noter que les propriétés qui viennent d'être énoncées ne sont pas uniquement le fait de la mobilité, elles proviennent également de la grande décentralisation du contrôle du mécanisme d'interaction. En fait, décentralisation du contrôle et implémentation par agents mobiles s'avèrent complémentaires : l'un ne peut être pleinement exploité sans l'autre.

Le manque de temps ne m'a pas permis d'obtenir des résultats *expérimentaux* pour confirmer ces affirmations. Toutefois, et bien que cela ne constitue pas une excuse, il faut noter l'absence de résultats publiés portant sur des expériences grandeur nature de système à base d'agents mobiles. Cette lacune est sans doute à mettre sur le compte de l'extrême jeunesse de cette technologie. Au §4.2.5.5 sont mentionnés les seuls résultats expérimentaux que je connaisse à ce jour.

Un autre point intéressant est à noter, concernant l'utilisation d'agents mobiles pour l'implémentation des interactions. Les agents mobiles étant « physiquement » situés sur la place d'accueil des agents serveurs, tout se passe comme si ces agents serveurs « coopéraient entre eux » afin d'activer d'un groupe de comportements satisfaisant les besoins associés au comportement de l'agent client. *Le protocole de coopération utilisé par les serveurs est implémenté par les agents mobiles : il est donc propre à chaque classe d'agents mobiles et indépendant des agents serveurs*. Cette propriété permet une totale flexibilité : différentes

---

[29] Une requête SQL ne peut être considérée comme une forme « dégradée » de code mobile. Toutefois ce n'est pas un agent mobile. D'une part, la requête SQL ne dispose donc pas de module de communications et ne peut ni recevoir ni émettre de messages à destination du programme qui l'a émise (ou de tout autre programme). D'autre part, une fois exécutée par la base de données, elle n'est ni réutilisée (sauf dans le cas particulier des procédures stockées) ni transmise à une autre base de données pour y être exécutée sur un autre ensemble de données.

[30] C'est le cas pour certains service telecom, dont les cahiers de charges spécifient des contraintes très spécifiques portant sur la qualité de service demandée par le client. Cependant il est clair que dans l'exemple de l'agence de Voyage, tel qu'il est présenté en §2.4, la description des services VOYAGE et TRANSPORT est trop simple pour que l'intérêt des agents mobiles soit évident. Le recours à un exemple de service plus complexe aurait nuit à la lisibilité de ce travail.

classes d'agents mobiles implémentent des protocoles de coopération différents. L'agent client choisirait la classe la plus adaptée aux conditions d'exploitation courantes et aux besoins à satisfaire (sans avoir à modifier les serveurs).
De ce point de vue, les agents mobiles peuvent être considérés comme des composants logiciels ajoutés aux agents serveurs de manière à enrichir dynamiquement leur capacité de coopération.

5.10.5 Extensions du mécanisme d'interaction

5.10.5.1 Décentralisation complète du mécanisme d'interaction

Le principal défaut du mécanisme d'interaction proposé, est le recours à l'agent client pour collecter les propositions de services et calculer les groupes d'agents serveurs.

L'utilisation d'un algorithme de satisfaction de contrainte *décentralisé* permettrait de répartir le calcul des groupes sur l'ensemble des agents mobiles. L'algorithme de Ghédira ([Ghedira 94]) est un bon candidat : il est déjà implémenté sous forme d'un système multi-agents avec des « agents-variables » et des « agents contraintes ». En outre, la correspondance entre les agents-variables et agents-contraintes de Ghédira, et les agents serveurs et agents mobiles de notre mécanisme d'interaction, est directe :

- chaque agent-variable est chargé de « fournir » la valeur d'un variable particulière. Il a donc le même rôle qu'un agent serveur qui détermine le service précis — en terme de lieu et d'horaire de départ et d'arrivé, et de prix — qui est fourni au client.
- chaque agent-contrainte négocie avec des agents-variables pour obtenir un ensemble de valeurs compatibles en regard de la contrainte qu'il représente. Si la contrainte est mono-service, alors l'agent représente une partie[31] d'un cahier des charges. La compatibilité entre les différents services, est assurée *de manière décentralisée* par l'ensemble des agents-contraintes n-aires avec n>1.

Le problème de la localisation des agents-contraintes n-aires se pose : par exemple où doit se trouver l'agent-contrainte binaire qui assure la compatibilité d'horaire à l'escale, entre le service « TRANSPORT Dijon Paris » de SNCF et le service « TRANSPORT Paris NewYork » de AirFrance : sur la place de l'agent serveur SNCF, sur celle de l'agent serveur AirFrance, ou bien sur une autre place ? Une analyse plus fine des communications entre l'agent-contrainte et les agents-variables (les agents serveurs) est nécessaire. Une heuristique simple est de placer l'agent mobile sur la place de la variable « la plus chaude » : l'algorithme de Ghédira utilisant la technique du recuit simulé, le température d'un agent-variable définit sa propension à accepter à changer de valeur[32]. Plus l'agent-variable est « chaud », plus il est

[31] Plusieurs agents-contraintes unaires peuvent porter sur la même variable : chacun définit alors une partie seulement du cahier des charges sur le service représenté par cette variable.

[32] Plus précisément, une variante décentralisée du recuit simulé est utilisée : chaque agent-variable a une température propre. Chaque agent-variable contrôle de manière autonome l'évolution de la température propre. Il

facile de lui faire changer de valeur. Donc l'agent-contrainte a plus de chance de réussir à « valider sa contrainte » (i.e. obtenir un couple de valeurs compatibles en regard de la contrainte qu'il représente) en négociant un changement de valeur avec l'agent-variable le plus chaud qu'avec les autres agents-variables. L'agent-contrainte communique donc plus avec cet agent qu'avec les autres. Il en découle que pour minimiser les communications, l'agent-contrainte doit être localisé sur le site de cet agent-variable le plus chaud.

5.10.5.2 Détermination dynamique du réseau de contraintes

Une autre qualité de l'algorithme de Ghédira est d'autoriser la modification dynamique du réseau de contrainte en cours de résolution : des contraintes peuvent être ajoutées / supprimées, et/ou modifiées, de même des variables peuvent être ajoutées ou supprimées. Le processus de résolution du problème ne recommence pas « à zéro » la résolution complète lorsque de telles modifications sont apportées à l'énoncé du problème : il adapte la solution à l'ancienne version de l'énoncé, et obtient une solution au nouvel énoncé en minimisant les modifications[33].

Cette propriété est particulièrement intéressante, car elle permet de réaliser un mécanisme d'interaction qui « découvre » dynamiquement la relation de compatibilité. A l'initiation de l'interaction, l'agent client ne donnerait que des contraintes génériques, comme par exemple :

- une contrainte d'horaires à l'escale portant sur deux services successifs quelconques, sans préciser quel service (du type « s_i.dateArr < s_{i+1}.dateDep », où s_i et s_{i+1} dénotent deux services quelconques successifs dans le transport)
- la contrainte sur le prix total du transport, sans spécifier le nombre exact de services composant ce transport. Cette contrainte s'exprimerait par un « Σs_i.prix < prixMax, où i parcourt l'ensemble des services *qui est inconnu à priori* »

Ces contraintes définissent un réseau générique, au sens où le nombre de variables est inconnu.

Le mécanisme d'interaction pourrait débuter en supposant que le réseau ne contient qu'une seule variable (i.e. que le transport est sans escale), puis, si aucune solution n'est trouvée, ajouter une nouvelle variable et ses contraintes associées (i.e. introduire une escale), et résoudre la nouvelle configuration du réseau ainsi obtenue[34]. Le mécanisme d'interaction pourrait ainsi trouver une solution sans faire l'hypothèse a priori sur le nombre de sous-services composant le service de transport.

---

n'y a donc pas de température globale à l'ensemble du système. L'algorithme gagne ainsi en décentralisation, mais perd la validation formelle de sa convergence.

[33] Cette adaptation est simplement mise en œuvre par un « réchauffage » local du réseau de contrainte, précisément aux endroits où des modifications ont eu lieu. Le système se relaxe alors à nouveau, et converge vers une solution proche de l'ancienne.

[34] Même si aucune solution n'a été obtenue pour l'ancienne configuration du réseau de contraintes, une partie des calculs qui ont été fait ne sont pas répétés pour calculer la solution à la nouvelle configuration du réseau.

En outre, l'interaction pourrait identifier plusieurs décompositions valides (i.e. trouver des solutions à plusieurs configurations différentes du réseau de contraintes), et sélectionner les meilleures d'entre elles.

Par manque de temps, il n'a pas été possible de réaliser l'implémentation de l'algorithme de Ghédira sur JavaNetAgents, ni a fortiori de le tester sur le système de l'Agence de Voyage pour valider (1) l'intérêt d'un algorithme de satisfaction de contrainte décentralisé, (2) l'heuristique de placement des agents-contraintes n-aires, et enfin (3) l'exploitation de la « dynamicité » de cet algorithme. Mais des travaux similaires, menés aux BTLabs, ont conclu à l'intérêt de cette approche.

5.11 Conclusion

L'interaction provoque l'activation d'un groupe d'agents serveurs (ou de plusieurs groupes successivement) satisfaisant le client. En d'autres termes, l'interaction provoque l'adaptation dynamique de l'organisation du SMA, afin qu'un agent (l'initiateur de l'interaction) acquiert la capacité de réaliser un de ses objectifs (fournir un service à l'un de ses propres clients)..

La décentralisation du mécanisme d'interaction s'avère très intéressante dans les conditions d'exploitation particulières des SMA du type SCDO. Elle permet notamment une grande réactivité de l'interaction aux changements (défaillance d'agents serveurs). Elle assure aussi l'extensibilité du mécanisme d'interaction : l'exemple de l'Agence de Voyage ne fait intervenir que des groupes de deux services, mais —du fait de sa décentralisation— le mécanisme fonctionne aussi bien pour des groupes plus importants.

L'utilisation d'agents mobiles présente également un grand intérêt. En particulier les agents mobiles garantissent une grande résistance aux pannes machines/réseau. Ils permettent aussi des économies de bande passante puisque les négociation entre agents mobiles et agents serveurs consistent essentiellement en des échanges de messages locaux et ne passent donc pas par le réseau. Mais le principal avantage des agents mobile est la possibilité pour l'agent client de choisir le mécanisme d'interaction qui lui convient, voire même d'en changer en cours d'interaction.

De ce point de vue, les agents mobiles peuvent être considérés comme des composants logiciels ajoutés aux agents serveurs de manière à enrichir dynamiquement leur capacité de coopération.

6. JavaNetAgents : une plate-forme d'exécution d'agents mobiles

Ce chapitre présente JavaNetAgents (JNA), la plate-forme d'exécution d'agents mobiles développée dans le cadre de cette thèse, et utilisée pour réaliser le système de l'Agence de Voyage.

6.1 présentation

JavaNetAgents a été réalisée pour permettre l'implémentation du système de l'Agence de Voyage et du mécanisme d'interaction proposé au chapitre §5. JNA a été réalisé avec le concours de Claude Seyrat lors de son stage de DEA au LIP6 ([Seyrat 96]).

En plus de l'objectif « pratique » d'implémenter le système de l'Agence de Voyage avec des agents mobiles, l'objectif de JNA est également d'analyser les enjeux, les difficultés et l'apport réel des techniques et des concepts liés à la mobilité de code. Cette plate-forme a donc une vocation essentiellement « expérimentale » et n'a de ce fait pas vocation à se comparer aux plates-formes « commerciales ». JNA peut paraître —et est effectivement !— dépassé actuellement[35]. En effet de nombreuses plates-formes existent désormais, et parmi celles-ci certaines offrent des fonctionnalités de haut niveau qui font défaut à JNA (notamment un service de nommage, qui est particulièrement important, cf. §7.6). Toutefois, il faut garder à l'esprit qu'au début de cette thèse, très peu de plates-formes d'exécution d'agents mobiles existaient : la conception de JNA a débuté alors que n'existaient « sur le marché » que Telescript ([Telescript95]. Telescript est lourd et difficile à manipuler), et la pré-version « alpha » de Mole ([Straßer 96]. Mole est impossible à installer à l'ONERA, car Mole ne fonctionne qu'associé à un serveur Web, qui ne peut être installé à l'ONERA pour des raisons de sécurité).

35 A l'exception notable de son système de sécurité, car encore actuellement aucune plate-forme n'offre de contrôle d'exécution des agents aussi fins que le système de sécurité de JNA.

6.2 Le noyau JNA

6.2.1 Description.

La plate-forme JavaNetAgents (JNA) fournit une structure d'accueil aux agents mobiles de niveau de mobilité 1 (cf. §4.2.5.2). Elle leur permet de s'exécuter, d'interagir entre eux, et d'entrer en contact avec des applications locales. Elle masque l'hétérogénéité des systèmes informatiques hôtes. Elle assure également une fonction de sécurisation en contrôlant strictement les comportements de chaque agent.

Comme toute architecture à base d'agents mobiles (cf. §4.2.5), l'architecture de JNA a deux composants principaux : les places et les agents (Figure 47).

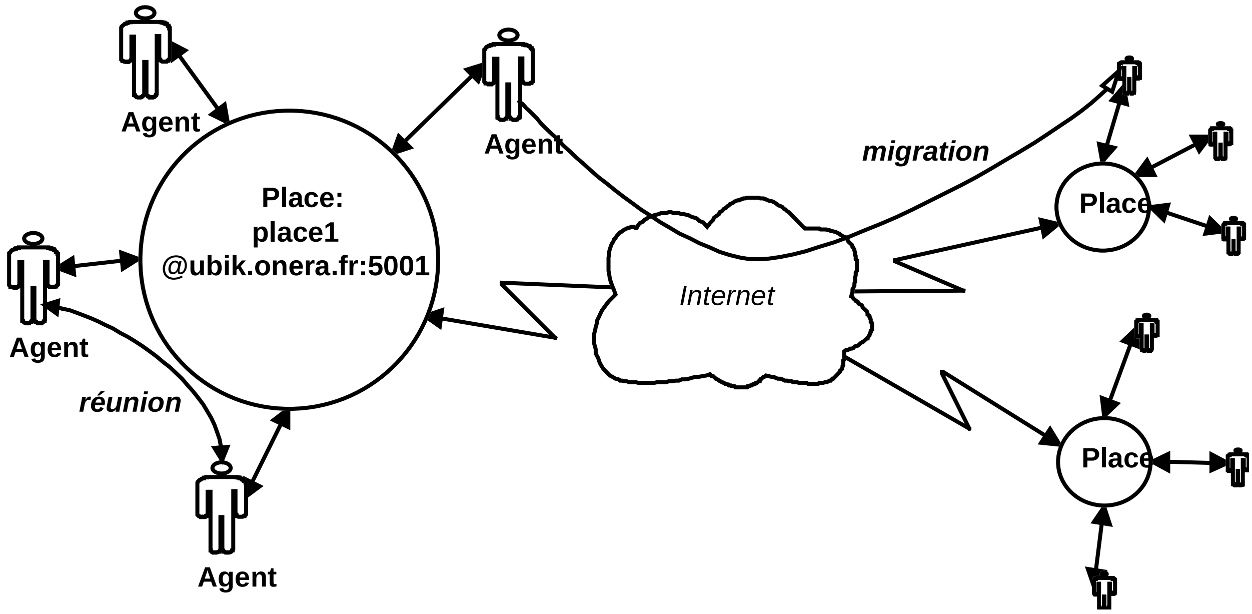


Figure 47. architecture générale de JNA.

Les places accueillent des agents (Figure 48), leur fournissent un accès sécurisé aux ressources locales (fichiers, sockets, graphiques, threads), assurent leurs interactions (communications ou réunions), et leur permettent de migrer sur d'autres places. Les agents sont des objets Java actifs (possédant leur propres threads d'exécution) capables de se déplacer de places en places.

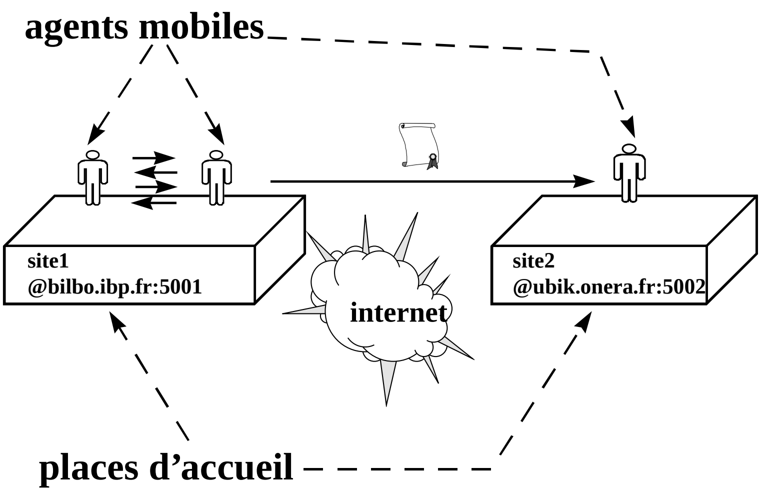


Figure 48. Les deux principaux composants de JNA : place et agent.

6.2.1.1 Les places.

Chaque place est située sur une machine hôte (il peut y avoir plusieurs places sur une même machine). C’est un moteur d’exécution sécurisée d’agents mobiles.
Elle fournit aux agents un accès sécurisé aux ressources locales : chaque agent peut utiliser des fichiers, des sockets, des threads d’exécution ou des fenêtres graphiques en fonction de ses droits propres. Ces droits lui sont donnés par la place à son arrivée (il peut en réclamer de nouveaux en cas de besoin).
La place assure également les interactions entre les agents : soit en transmettant leurs messages, soit en préparant leurs réunions (voir §6.2.2.2). Il faut noter que l’utilisation d’objets mobiles dispense les places d’avoir à connaître les protocoles d’interaction : chaque message et chaque agent implémente son propre protocole de transmission/migration. Cette caractéristique permet d’optimiser dynamiquement les interactions entre agents.

Du point de vue implémentation informatique, chaque place est exécutée sur une machine Java, et est à l’écoute d’un port de communication de l’ordinateur hôte (Figure 49). Par ce port arrivent les demandes de connexion de places distantes pour la transmission d’un message ou d’un agent. L’adresse de la place (qui est unique) est définie par l’adresse Internet de la machine hôte suivie du numéro de port. Les places situées sur une même machine se différencient par leur numéro de port.

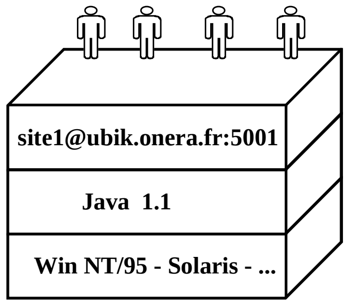


Figure 49. La place est placée dans une machine virtuelle Java.

6.2.1.2 Les agents

JNA n'impose aucune architecture interne particulière pour les agents : l'architecture peut être du type cognitif, réactif, à base de règles ou de réseaux de neurones, etc. En fait, tout programme Java peut être encapsulé dans un agent.

En revanche, JNA spécifie les interactions possibles entre agents : par envois de message ou par réunion (voir plus loin). Les interactions ont lieu par l'intermédiaire des places.

Du point de vue implémentation informatique, un agent JNA est un ensemble d'objets sans référence vers l'extérieur. On peut définir un agent comme étant la fermeture transitive sur l'ensemble des objets que contient l'objet instance de la classe « Agent » (ou d'une sous-classe). En d'autres termes, un agent ne « connaît » que ce qu'il « possède ». Cette définition facilite la migration de l'agent (sa fermeture – tous les objets qu'il « possède » -- migre avec lui).

Chaque agent dispose d'un nom (qui peut ne pas être unique), d'une adresse (déterminée par son nom et l'adresse de la place où il se trouve. ex : TalkAgent@bilbo.ibp.fr:5000), d'un propriétaire (le nom d'un utilisateur humain, en général le créateur de l'agent), et de droits.

Les droits de l'agent spécifient les ressources locales auxquelles il a accès (fichiers, threads, sockets, fenêtres), et le type d'opération qu'il peut faire dessus (création, lecture, écriture, etc.). Actuellement on ne peut limiter la consommation de ressources CPU d'un agent (les futures versions de Java devraient résoudre ce problème). Ses droits lui sont alloués dynamiquement par la place, en fonction de ses besoins, de son propriétaire et d'autres caractéristiques.

Le comportement d'un agent est essentiellement programmé par quatre méthodes (Figure 50), que tout concepteur d'agent peut redéfinir pour obtenir des agents spécialisés :

- `agentInit()` Initialisation de l'agent
- `agentRun()` Comportement de l'agent
- `agentEnd()` Méthode exécutée avant la mort de l'agent
- `agentAwake(int cpte-rendu)` Méthode exécutée après une migration

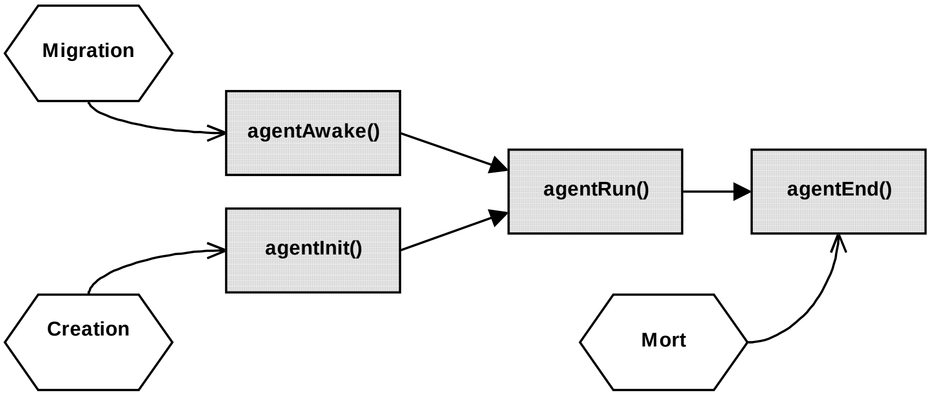


Figure 50. cycle de vie d'un agent.

Le vecteur d'état de l'agent mobile n'est pas sauvegardé lors de la migration (le niveau de mobilité est 1, cf. §4.2.5.2). En conséquence, après avoir migré l'agent ne reprend pas son exécution à l'endroit précis où il s'était arrêté avant de migrer : son « point de réveil » est la méthode `agentAwake(int cpte-rendu)`. Elle est automatiquement invoquée à la fin du processus de migration, lorsque l'agent a été reconstruit sur la place réceptrice. L'entier passé en paramètre indique la manière dont s'est déroulée la migration (succès, échec, etc.).

6.2.1.3 choix de l'implémentation en Java

Java présente de nombreux atouts pour la programmation d'architectures à base d'agents mobiles :

- **machine virtuelle indépendante de la plate-forme d'exécution :** les programmes (et en particulier les agents) sont ainsi interprétés d'une manière identique sur toutes les plates-formes. Il existe maintenant des machines virtuelles Java sous Unix (Solaris, Linux, AIX, HP-UX, Irix, etc.), Windows 95 et Macintosh OS.
- **multi-threaded :** Java fournit de tous nombreuses fontionnalités pour la gestion des threads d'exécution.
- **sécurité :** Un code « inconnu » peut être exécuté avec sécurité. En effet, l'accès aux ressources sensibles du système hôte est surveillé par un système de sécurité.
- **persistance :** presque tout objet Java peut être « sérialisé » i.e. transformable en une représentation sous forme d'un ensemble d'octets. Ainsi, un objet Java pourra être sauvegardé ou transféré via le réseau.
- **chargement dynamique de classe dans une machine virtuelle :** Java dispose de fonctionnalité permettant le chargement dynamique d'une classe dans une machine virtuelle, sans avoir à stopper la machine.

Ces propriétés font de Java un très bon candidat pour le développement d'une plate-forme d'exécution d'agents mobiles. Les agents mobiles sont implémentés sous forme d'objets « sérialisables ». Leur migration est réalisée en trois étapes :

1. sérialisation de l'agent
2. transfert de la version sérialisée de la place de départ à la place destination

3. reconstruction de l'agent par « désérialisation » de la version sérialisée. Cette étape peut éventuellement nécessiter le transfert de définition de classes Java.

Le protocole précis de migration d'un agent est décrit au §6.2.3.2.

6.2.2 Les interactions en JNA.

JNA permet 2 types d'interactions entre agents : l'envoi de messages asynchrones et l'appel fonctionnel (synchrone).

6.2.2.1 L'envoi de messages.

Les messages permettent la communication asynchrone entre agents distants.
Ce sont des objets mobiles passifs : ils n'ont pas de *thread* d'exécution propre. Contrairement aux messages « traditionnels » qui ne véhiculent que des données, les messages JNA sont des objets : ils encapsulent les données (le contenu du message) et des procédures qui définissent le protocole de communication utilisé pour la transmission du contenu. Le protocole définit ce qu'il faut faire si la migration échoue, si elle dure trop longtemps, si l'agent destinataire ne se trouve pas à l'adresse indiquée, etc.

Il est important de noter que, les places étant insensibles aux protocoles utilisés pour la transmission des messages, les concepteurs de classes de messages (et d'agents) ont une grande liberté pour introduire dynamiquement dans le SMA des protocoles de communication nouveaux et optimisés, par spécialisation de la classe « Message ».

Il est par exemple possible de réaliser facilement des messages qui peuvent être émis vers un destinataire dont l'adresse est inconnue. Ce type de message est particulièrement intéressant quand le destinataire est un agent mobile dont l'adresse change à chaque fois qu'il se déplace dans le réseau. Dans ce cas, les autres agents ne connaissent pas son adresse exacte au moment où ils lui envoient un message[36]. Le message est alors diffusé dans le réseau, à la recherche de l'agent destinataire (cf. Figure 51). Le mode de diffusion utilisé correspond à l'inondation du réseau : le message est dupliqué à leur création (i.e. au moment de leur émission par l'agent émetteur). Les copies sont diffusées sur les places voisines (i.e. celles qui sont référencées auprès du serveur « Concierge » local. cf. §6.4.2). Ce processus est répété avec chaque clone, de sorte que toutes les places sont atteintes de proche en proche. Des mécanismes de trace et de « time out » permettent de stopper la diffusion lorsque toutes les places ont été visitées, ou à défaut au bout d'une durée fixée.

---

[36] Si la plate-forme d'exécution d'agents mobiles fournit un service nommage, alors ce problème n'a pas lieu d'être : par simple interrogation de ce service, tout agent émetteur peut connaître l'adresse courante d'un agent —mobile ou non— auquel il veut envoyer un message. Cependant peu de plates-formes d'exécution d'agents mobiles proposent actuellement un service nommage. En particulier JNA n'en a pas., et c'est pourquoi ces messages sont beaucoup utilisés dans l'implémentation du système d'Agence de Voyage.

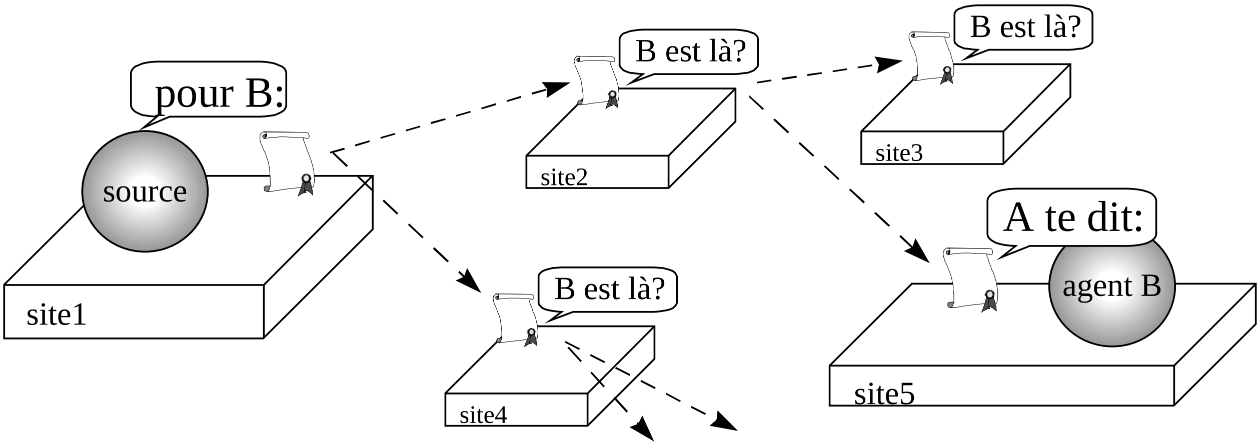


Figure 51. Diffusion d'un message JNA par inondation du réseau.

Ce type de diffusion est bien sûr très coûteux en bande passante, mais il assure une forte résistance aux pannes de réseau. Dans le même ordre d'idée, on peut également utiliser un mode de diffusion épidémique des messages ([Demers et al.87]) : chaque message se clone et se diffuse sur une quelques places voisines[37], à la manière d'un virus. Des techniques d'analyse stochastique permettent de calculer les performances et de valider le mode de diffusion épidémique ([Demers et al.87] et [Pittel 87]).

Il est intéressant de noter que du fait de leur nature d'objet mobile, les messages JNA sont très comparables aux messagers de [Di Marzo et al.95].

6.2.2.2 Les réunions

Les réunions permettent à deux agents d'interagir par appels fonctionnel (Figure 52). Ce mécanisme correspond au mode d'interaction traditionnel dans le monde objet, il est donc plus efficace en terme d'implémentation que l'envoi de message. Il ne peut avoir lieu qu'entre 2 agents situés sur la même place (pour l'instant, les appels fonctionnels distants ne sont pas implémentés).

Une réunion est établie par la place, à la demande d'un des agents. L'autre agent peut refuser la réunion. La demande d'une réunion spécifie un couple d'interfaces Java (i.e. deux ensemble d'API Java), sur la Figure 52 c'est « Interf1, Interf2 » :

- Interf1 est l'interface implémentée par l'agent demandeur de la réunion (agent1), elle définit l'ensemble des méthodes que l'agent agent2 peut appeler sur l'agent agent1.
- Interf1 est l'interface implémentée par l'autre agent (agent2), elle définit l'ensemble des méthodes que l'agent agent1 peut appeler sur l'agent agent2.

Ces interfaces correspondent, en quelque sorte, aux rôles de chaque interlocuteur dans la conversation

[37] Ce point constitue la principale différence avec la diffusion par inondation : le message ne se propage pas sur toutes les places voisines, mais seulement sur un sous-ensemble (calculé de manière stochastique). La taille du sous-ensemble est fonction du nombre de places voisines déjà « contaminées » (i.e. qui ont déjà reçu un autre clone du même message). Avec le temps et la multiplication des clones, toutes les places finissent par être contaminées avec une probabilité égale à 1, et la diffusion s'arrête. Toute les places ont alors reçu au moins un clone du message.

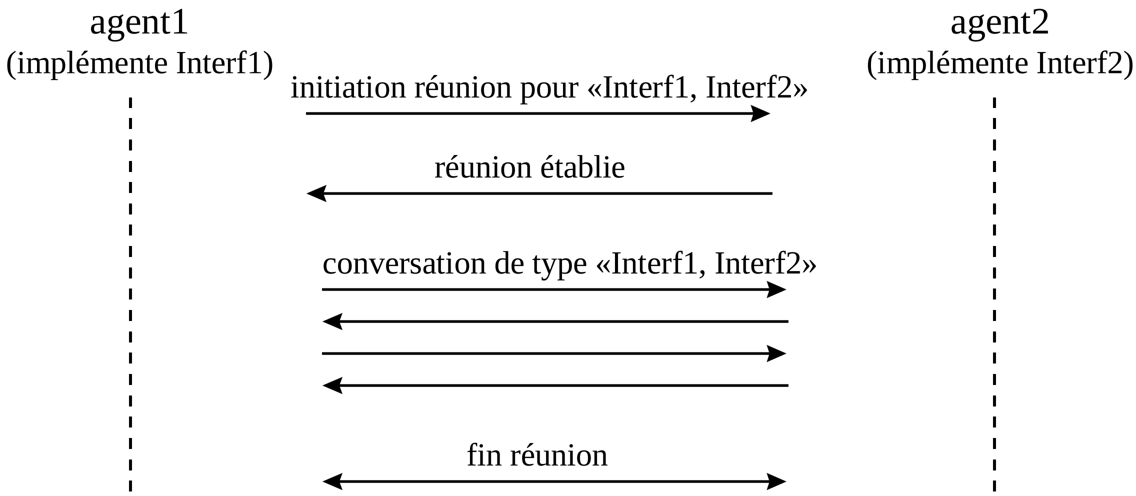


Figure 52. Conversation « Interf1, Interf2 ».

Remarques :

- l'agent « appelé » agent2 peut refuser la réunion (par exemple s'il n'implémente pas l'interface Interf2). Dans ce cas il répond « réunion refusée » à la place de « réunion établie ».
- l'interface Interf1 peut être vide (dans ce cas, le paramètre Interf1 dans la requête d'initiation de la réunion, vaut *null*). Ce cas correspond à une conversation au cours de laquelle l'agent agent2 se contente de « répondre » à l'agent agent1 : les messages envoyés de l'agent2 à l'agent1 sont les valeurs de retour des appels de méthode effectués par agent1 sur agent2. A aucun moment agent2 « ne prend l'initiative » d'envoyer un message à agent1 (en lui faisant un appel de méthode), c'est pourquoi agent2 n'a pas besoin des « points d'entrée » sur agent1 que définit l'interface Interf1.
- en toute rigueur, la donnée de deux interfaces n'est pas suffisante pour définir une conversation : il manque le protocole, i.e. les règles spécifiant le ou les enchaînements légaux de messages entre les deux agents. Toutefois, dans le cas de conversations suffisamment simples, le protocole peut être omis. C'est le cas dans l'implémentation de l'Agence de Voyage, où le protocole est de type « question - réponse » : les conversations n'impliquent alors que l'échange de deux messages entre les agents.

6.2.3 La migration d'agents.

6.2.3.1 Ticket.

La migration de l'agent est supervisée par un objet « ticket » qu'il crée juste avant de migrer. Le ticket reste sur le site émetteur (il se détruit à la fin de la migration). Il met en œuvre un protocole de migration qui définit ce qu'il faut faire si la migration échoue, si elle dure trop longtemps, si la place destination n'est pas accessible, etc. Si la migration réussit, le ticket génère un compte-rendu de migration qui est passé en paramètre de la méthode agentAwake() au réveil de l'agent. Si elle échoue, l'agent est réveillé sur son site de départ.

Comme pour la transmission de messages, un concepteur d'agent peut définir des protocoles optimisés pour ses agents, par surcharge de la classe « Ticket ». De même, un agent peut changer dynamiquement de protocole de migration (sauf lorsqu'il est en phase de migration), en changeant de ticket.

6.2.3.2 Ticket standard.

Le ticket standard définit le protocole de migration « par défaut » des agents mobiles JNA.

La migration « standard » d'un agent d'un site A à un site B (Figure 53 à Figure 58) se déroule en plusieurs étapes :

1. création du ticket pour gérer la migration (Figure 54). Le ticket suspend les activités de l'agent et le « sérialise ».
2. ouverture d'une connexion avec le site destination, et transmission de la version sérialisée de l'agent (Figure 55). Le Ticket dispose alors d'une marge de contrôle : si le temps de transfert est trop long ou si le site n'est pas joignable il peut rerouter l'agent ou bien annuler la migration. En cas d'annulation de la migration, le ticket « réveille » l'agent sur le site de départ, en lui indiquant un code de réveil spécial (envoi du message « agentAwake(code_réveil) »).
3. reconstruction de l'agent par « désérialisation » de la version sérialisée de l'agent (Figure 56). Durant cette phase, le site récepteur peut demander au site émetteur, les définitions de classes dont il a besoin (des classes faisant partie de la définition de l'agent, mais dont le site récepteur ne dispose pas en local), cf. Figure 57. Le site émetteur les lui envoie.
4. une fois l'agent reconstruit (mais pas encore activé), le site récepteur accepte ou refuse l'agent. Si le site récepteur refuse l'agent, l'agent est « détruit » (les objets qui le composent sont « garbagés »). Qu'il accepte l'agent ou le refuse, le site récepteur envoie un compte rendu de migration au site émetteur : migration_acceptée, migration_refusée ou erreur_de_migration (si l'agent n'a pu être reconstruit correctement). Puis le ticket est détruit sur le site émetteur.
5. si le site accepte l'agent, alors il lui crée les ressources locales personnelles (une fenêtre pour que l'agent y place son interface graphique, un « groupe de threads » pour les activités de l'agent) et il réveille l'agent (envoi du message « agentAwake(code_réveil) » avec en paramètre le « code de réveil » envoyé par le ticket à l'étape précédente (Figure 58).
6. si la migration ne s'est pas déroulée correctement (si l'agent a été refusé par la site récepteur, ou bien si le site récepteur n'a pas pu reconstruire l'agent), alors l'agent est réveillé sur le site émetteur avec un code de réveil particulier (Figure 59).

Figure 53. L’agent désire migre du site A vers le site B

Figure 54. Un objet Ticket est créé. L’agent est suspendu puis « sérialisé »

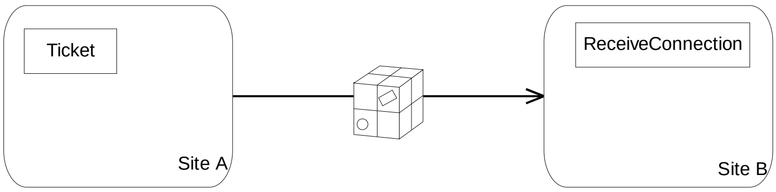


Figure 55. La version « sérialisée » est transmise via le réseau.

Figure 56. L’agent est « désérialisé » sur le site récepteur

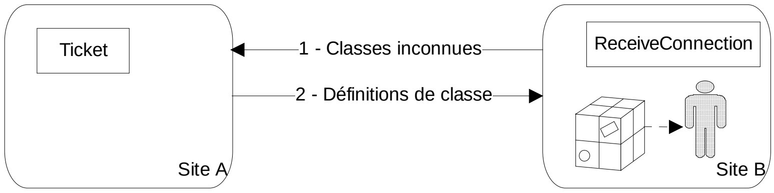


Figure 57. Récupération des classes non connues du site récepteur.

Figure 58. L'agent sur le site B est relancé par la méthode « agentAwake ».

Figure 59. En cas de problème lors de la migration, l'agent est réveillé sur le site A.

6.2.4 L'interface graphique

Chaque place dispose d'une interface graphique, qui permet de gérer l'ensemble des agents qu'elle accueille (les tuer, en créer de nouveaux, accéder à leurs interfaces graphiques propres, etc.).

Sur chaque place, existe un agent particulier, le GraphAgent (décrit au §6.4.3), dont la fonction consiste à représenter graphiquement sous forme « attrayante » les agents présents sur la place (Figure 60. Sur cette figure, le lien représente une réunion en cours entre l'agent « select-Voy » et l'agent « AirFrance »). Des captures d'écran de la fenêtre du GraphAgent serviront au §7 pour décrire le déroulement du scénario de l'Agence de Voyage.

Chaque agent peut également disposer d'une interface graphique propre (voir ex. Figure 60 et Figure 61), qui s'inscrit dans des fenêtres indépendantes. Pour des raisons de sécurité, les droits par défaut d'un agent lui interdisent de créer lui-même des fenêtres (pour éviter que des agents indélicats ne « polluent » l'écran avec de multiples fenêtres). A l'arrivée sur une place, chaque agent se voit confier une fenêtre « non redimensionnable », dans laquelle il peut ajouter ses propres éléments graphiques (boutons, listes, zones de textes, etc.). Cette fenêtre affiche également les caractéristiques de l'agent (nom, droits, etc.) et indique clairement que les autres informations qu'elle donne sont sous la responsabilité exclusive de l'agent (une bande colorée barre le coté supérieur de la fenêtre et avertit l'utilisateur que la fenêtre appartient à un agent, et qu'il devrait lui accorder une confiance limitée).

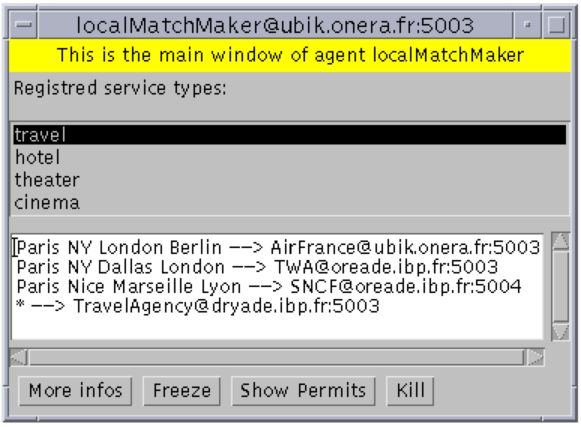


Figure 60. fenêtre du Trader de la place située sur la machine ubik à l'ONERA.

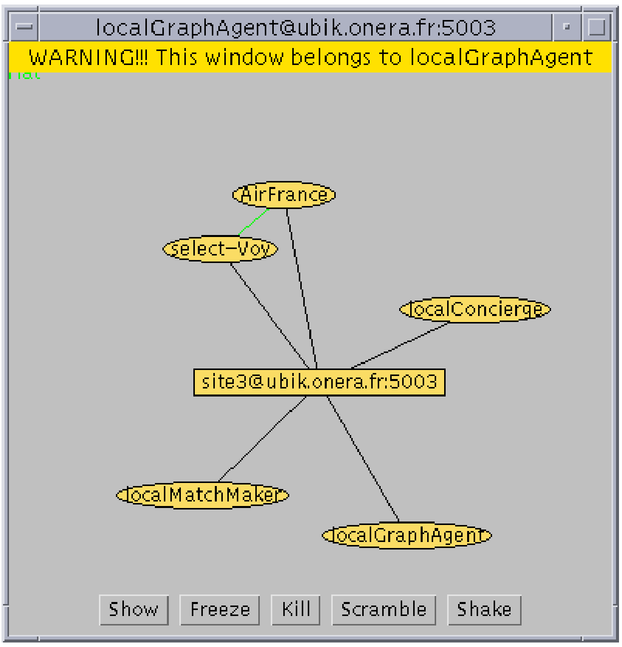


Figure 61. fenêtre d'un GraphAgent montrant tous les agents sur une place.

6.2.5 Interface Web

Chaque site est également un serveur Web, il reçoit et traite les requêtes formulées en HTTP. Les agents sont alors vus comme des pages Web dynamiques :

- leur contenu varie en fonction du temps (l'agent peut la régénérer à tout moment, pour mettre à jour les informations qui s'y trouvent)
- leur urls changent lorsque les agent migrent

L'url de la page web d'un agent est du type
http://nom_de_machine.nom_de_domaine.fr:port_écoute/~nom_d_agent, où :

- nom_de_machine est le nom de la machine hôte du site d'accueil de l'agent,
- nom_de_domaine et le nom du domaine Internet où cette machine se trouve,
- port_écoute est le numéro de port d'écoute du site
- nom_d_agent est le nom de l'agent

Tout utilisateur disposant d'un browser web peut, d'où qu'il se trouve, interagir avec un agent en visualisant son interface graphique « web » sur son browser (Figure 62).

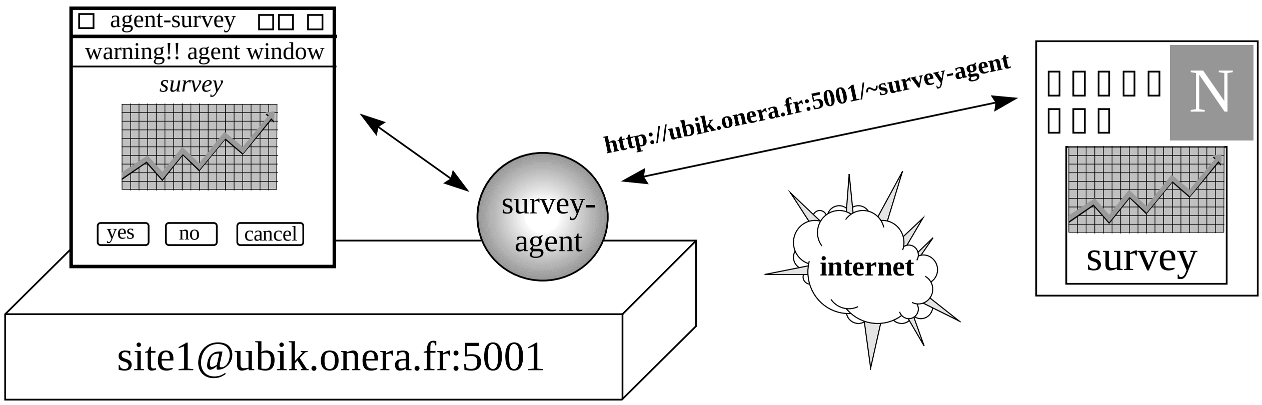


Figure 62. Les deux interfaces graphiques de l'agent « survey-agent »

6.3 Comparaison avec d'autres plates-formes existantes.

6.3.1 Telescript.

Telescript ([Telescript95]) est un langage d'objets mobiles développé par *General Magic* pour la réalisation de Systèmes Multi-Agents mobiles sur Internet. Il propose une architecture basée sur la notion de place de marché, où se réunissent des agents pour négocier des services. Cette

architecture nous paraît très judicieuse : nous en retrouvons beaucoup des caractéristiques dans JNA (place, réunion, notion de propriétaire d'un objet, etc.). La principale différence tient à la flexibilité de la migration des objets Telescript (la migration est de niveau 3), qui se paye par une lourdeur excessive de son implémentation.
Telescript est particulièrement lourd et difficile à manipuler. Il fut un pionnier en son temps, mais son développement est désormais arrêté.

6.3.2 Aglets.

Aglets ([Lange et al. 96]) est une plate-forme d'exécution d'agents mobiles écrite en Java par IBM. Ses caractéristiques techniques sont assez proches de JNA. La migration est de niveau 2. Les aglets peuvent interagir par échange de messages. Mais les messages échangés ne sont pas actifs comme dans JNA : le protocole de communication dans aglets est figé. Les aglets peuvent également interagir par appel fonctionnel.
Les mécanismes de sécurité en Aglets sont moins sophistiqués que dans JNA. Une distinction est faite entre les aglets « de confiance » (*trusted aglets*) et les autres (*untrusted aglets*) : dans chaque catégorie, l'administrateur d'un « site aglets » peut donner des droits en lecture et des droits en écriture sur un ensemble de fichiers locaux. La sécurité est donc semi-collective, et non pas personnalisée à chaque agent comme en JNA. De plus, il n'est pas possible de restreindre l'accès aux autres types de ressources locales comme les *sockets*, les *threads* et les fenêtres.

L'objectif à terme d'IBM est de définir un ensemble d'API standards pour les environnements d'exécution d'agents mobiles. Toute plate-forme implementant ces APIs peut accueillir des agents de type aglets.

6.3.3 Voyager

Voyager ([Voyager 97]) est une plate-forme d'agents mobiles en Java, qui se distingue des autres par sa grande intégration avec les technologies objets distribuées, et notamment les bus à objets (Object Request Brokers, ou ORB). En fait, la plate-forme Voyager est un ORB à part entière. Elle ne se conforme pas, pour l'instant au standard CORBA, mais son intégration à CORBA est en cours de réalisation.
Plusieurs caractéristiques de Voyager en font une plate-forme très confortable d'utilisation pour la programmation d'applications à base d'agents mobiles :

1. conservation de la syntaxe Java pour les communications distantes avec des agents mobiles : contrairement à JNA et à Aglets, les agents Voyager communiquent par « appel de méthode », c'est-à-dire exactement comme des objets Java quelconques. Ce service rend plus compact (et donc plus lisible, et plus facile à corriger) le code d'une application à base d'agents mobiles. Dans les plates-formes Java traditionnelles, comme JNA ou Aglets, l'envoi d'un message à un agent mobile nécessite souvent de nombreuses lignes de code (instanciation, initialisation et envoi du message. Puis attente et traitement de la réponse), ce qui rend ce code souvent difficile à réaliser et à corriger.
2. préservation des canaux de communication lorsque les agents migrent. L'ORB Voyager dispose d'un service de communication gérant la mobilité : lorsqu'un agent migre, son

adresse change et les messages qui sont acheminés à son ancienne adresse sont routés vers sa nouvelle adresse. Simultanément, la « table d'accointances » de l'agent émetteur du message, est mise à jour, afin que les prochains messages envoyés par cet agent, soient directement acheminés à la nouvelle adresse. Ce service s'avère extrêmement utile en pratique, car la gestion des tables d'accointances pour agents mobiles est une opération particulièrement fastidieuse pour le programmeur d'applications à base d'agents mobile.

3. mise en œuvre d'un mécanisme de GC (« ramasse miettes ») distribués et gérant la mobilité. Voyager est, à notre connaissance, la seule plate-forme à intégrer une telle fonctionnalité, pourtant indispensable pour la réalisation d'applications « sérieuses ».

Grâce à ces caractéristiques, Voyager est actuellement à notre avis la meilleure plate-forme existante pour la réalisation d'applications à base d'agents mobiles. Il faut cependant noter que c'est une plate-forme très récente, qui tire pleinement profit des fonctionnalité apportées par les dernières versions de Java, tandis que JNA et Aglets ont été conçues avec des versions antérieures de Java et, sur certains points, plus limitées.

6.3.4 Bilan du comparatif.

Sur le tableau ci-dessous figure un comparatif avec d'autres plates-formes semblables (nous avons inclus JavaAgent qui permet la migration d'objets passifs, mais pas d'agents) :

	Mobilité d'agents	**Envoi de msg**	**Réunion**	**Sécurité**
JavaNetAgents	oui	oui	oui	individualisée[1]
Mole ([Straßer et al. 96])	oui	oui	non	collective[2]
JavaToGo ([Li et al.96])	oui	oui	non	??
JavaAgent ([Frost 95])	non	msg. KQML	non	collective
Aglets	oui	oui	non	semi-collective
Voyager	oui	oui[4]	oui[4]	paramétrable[3]
Telescript	oui	non	oui	individualisée

[1]Une sécurité individualisée associe à chaque agent des droits propres. Cette sécuritée est obtenue par d'importantes modifications du gestionnaire de sécurité de Java.

[2]Sécurité collective signifie que le contrôle d'accès aux ressources locales est le même pour tous les agents (il est en fait effectué par le gestionnaire de sécurité natif de Java).

[3]Voyager permet de définir un système de sécurité, mais aucune aide n'est fournie pour assurer une politique de sécurité particulière (collective, individualisée ou autre).

[4]Les envois de messages distants et les réunions en Voyager s'écrivent avec la même syntaxe : la syntaxe Java « traditionnelle » pour l'appel de méthode. Cette technique facilite grandement la programmation des agents mobiles.

6.4 L'architecture Multi-Agents de JNA.

Pour réaliser l'application « Agence de Voyage » une architecture Multi-Agents est ajoutée au dessus de JNA. Cette architecture définit un ensemble de services de base, accessibles à tout

agent JNA. Chaque service est fourni par une classe d'agents particulière, dont une instance se trouve sur chaque place.
Actuellement trois services ont été réalisés :

6.4.1 Service de « pages jaunes ».

Ce service est fourni sur chaque place par un agent Trader ([Kuokka et al. 95]) dont le nom est par convention localTrader. Ce service facilite l'établissement des relations client/serveur. Tout agent JNA peut y accéder en entamant une conversation (dans le cadre d'une réunion préalablement établie) avec le localTrader de sa place.

Deux types de conversation sont possibles avec un trader :

- conversation « Advertise », pour indiquer au localTrader le type de service qu'il est susceptible de fournir (Figure 63).
- conversation « Serveur » pour demander au localTrader des adresses d'agents compétents pour satisfaire un besoin donné (Figure 63).

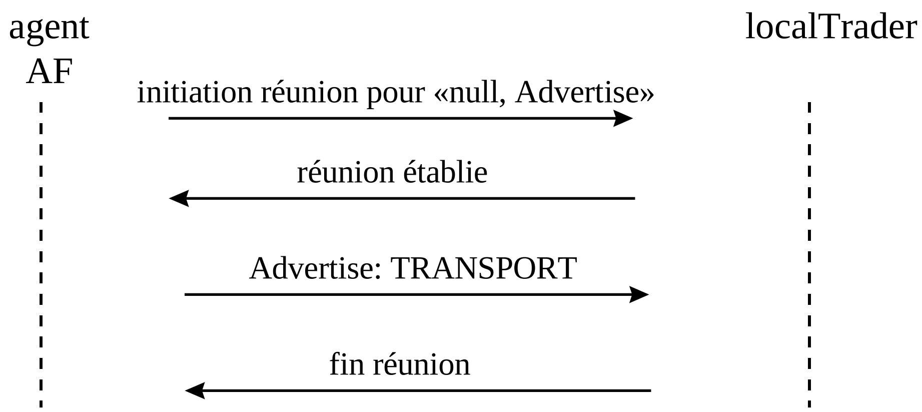


Figure 63. Conversation pour notifier le trader d'un comportement.

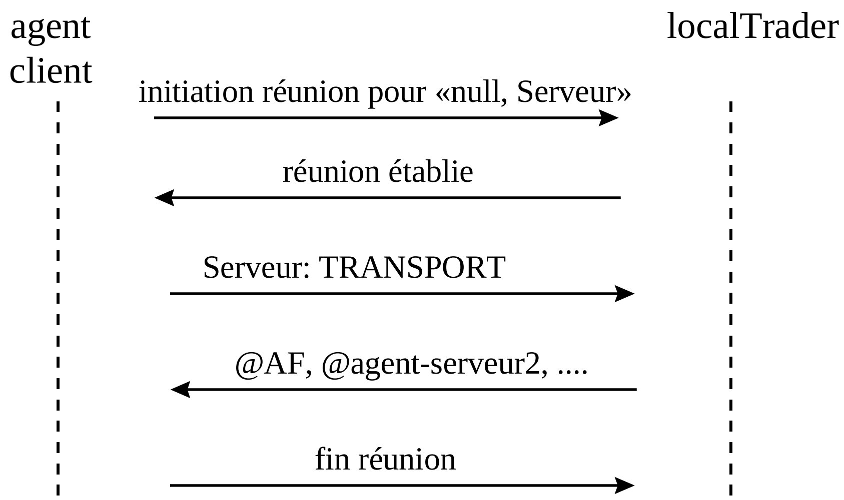


Figure 64. Conversation pour demander au trader une liste d’agents compétents.

La Figure 60 montre la fenêtre d’un localTrader dans le scénario de l’Agence de Voyage. On observe une liste de serveurs compétents pour le service « TRANSPORT ».

Dans l’état actuel de l’implémentation des traders JNA, les traders ne communiquent pas ensemble pour échanger leurs connaissances (contrairement aux concierges, décris dans le paragraphe suivant). En conséquence, chaque trader ne connaît que les comportements des agents locaux (i.e. situés sur la même place qui lui-même). Une extension (importante) des traders est de les faire communiquer pour diffuser les connaissances concernant les comportements de chaque agents dans le système. Pour ce faire, la technique de diffusion épidémique par agents mobiles pourrait être utilisée, comme c’est le cas déjà avec les concierges.

La classe Java des traders est la suivante :

```
public class Trader extends Agent implements Reunion, Advertise, Serveur {
private Hashtable registre = null ;   // contient des paires (comportements, adresses d’agents compétents)
public void advertise(String competence, AgentAddress serveur) ;        //stocke le couple (competence, serveur)
dans l’attribut registre
public Vector serveur(String competence) ;  //renvoie une liste d’adresses d’agents compétents
...
}
```

Remarques :

- par commodité de programmation, la classe Trader est sous-classe de Agent. En conséquence, les traders sont des agents mobiles. Cependant, de part leur fonctionnalité, ces agents ne migrent pas.
- l’interface Reunion est l’interface de base que tout agent implémente : elle définit la conversation permettant l’initiation (et la terminaison) d’une réunion : les messages « debutRéunion », « réunion établie », et « finRéunion ». Ces méthodes sont en fait définies dans la classe Agent, donc toute sous-classe de Agent implémente l’interface Reunion.

Cependant en Java, l'ensemble des interfaces implémentées par une classe doit être rappelé à la définition de cette classe : il n'y a pas « d'héritage » des *déclarations* d'interfaces, bien qu'il y ait héritage des méthodes qui sont spécifiées par ces interfaces ...

6.4.2 Service de voisinage.

Ce service permet de diffuser des informations concernant la topologie du réseau. Il permet à tout agent de connaître l'adresse de places distantes de celle où il se trouve, et d'avoir une indication de la distance de ces places (la distance inter-place est la mesure du coût du transfert entre 2 places). Ce service est assuré sur chaque place par un agent localConcierge.

La conversation entre un agent et un concierge est décrite à la Figure 65.

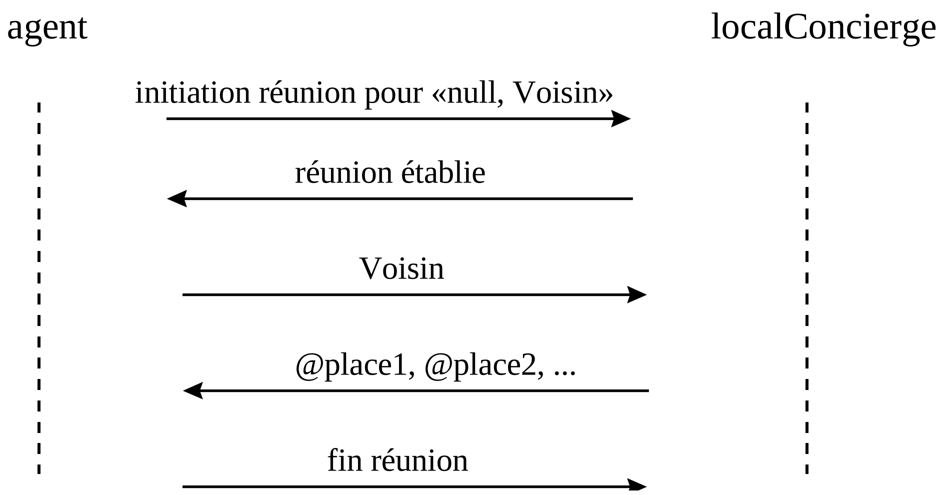


Figure 65. Conversation pour obtenir la liste des places voisines de la place courante.

Les localConcierges sont en conversation permanente pour s'échanger des adresses de nouvelles places (quand une place est créée, son « créateur » lui donne une liste d'adresses de places déjà existantes. Le localConcierge de la nouvelle place envoie des messages à ses collègues sur les autres places pour se faire connaître). Les messages échangés sont dupliqués et se propagent de proche en proche à tous les localConcierges qu'ils rencontrent. Ce protocole est particulièrement efficace dans des conditions dynamiques, quand des places sont souvent rajoutées ou supprimées du système.

La classe Java des concierges est la suivante :

```
public class Concierge extends Agent implements Reunion, Voisin {
private Vector voisinnage = null ;   // l'attribut contient les adresses des places voisines
public Vector voisin() ;     //renvoie la liste des places voisines
public void addVoisin(AgentAddress place) ;          //rajoute l'adresse d'une nouvelle place
public void delVoisin(AgentAddress place) ;          //supprime l'adresse d'une place
...
}
```

Remarque :

- comme les traders, les concierges sont définis comme agents mobiles, mais ils ne migrent jamais.
- les méthodes « addVoisin » et « delVoisin » sont utilisées par des agents mobiles chargés de diffuser à l'ensemble des agents concierges du système l'information concernant l'ajout ou le retrait d'une place. Le déroulement précis de la mise à jour des connaissances des concierges ne sera pas détaillé davantage.

6.4.3 Service de visualisation graphique.

Un agent de type GraphAgent, et de nom localGraphAgent, affiche dans sa fenêtre une représentation graphique de la place (Figure 61). Cet agent est un agent « système » : il a accès à des informations protégées (entre autres, il a des références directes sur tous les agents de la place). Pour des raisons de sécurité, ces agents ne peuvent être créés que par une place, et ils ne peuvent pas migrer ailleurs (les autres places refusent de les accueillir). Ce service sert en fait au développeur d'agents, plus qu'aux agents eux-mêmes.

6.5 Les agents stationnaires

6.5.1 présentation

Les agents du système multi-agents Agence de Voyage sont des agents *stationnaires* (i.e. non mobiles), dispersés sur plusieurs noeuds du réseau.

Ce choix se justifie par le fait que ces agents encapsulent (et représentent) des applications quelconques, comme des SGBD par exemple. Il n'est pas envisageable de faire migrer ces agents, car les applications qu'ils encapsulent ne sont pas elles mêmes mobiles. Ce sont des applications de type *legacy systems*, qui ne fonctionnent que dans un environnement matériel (type de machine) et logiciel (type de système d'exploitation) précis.
Par ailleurs, certaines de ces applications contiennent une quantité importante de données (le SGBD d'une compagnie aérienne contient la liste de tous les vols que propose la compagnie, et également l'ensemble des réservations déjà effectuées par des clients). Dans ces conditions, la migration de l'application serait d'un coût prohibitif en bande passante !

Pour des raisons de simplicité de l'implémentation, les agents stationnaires sont implémentés par... des agents mobiles qui restent constamment sur la même place (leur classes sont sous-classes de classes d'agents mobiles, mais ces agents ne sont pas programmés pour tenter de migrer). Ce choix facilite les communications entre agents stationnaires et agents mobiles.

6.5.2 appel d’offres à un agent serveur

La conversation avec un agent serveur a pour fonction de donner à l’agent une liste de services précis (type + valeurs des attributs) de l’agent serveur, chacun des services vérifiant bien sûr le cahier des charges.

L’agent client initie la conversation par un message « AppelDOffres SERVICE » (Figure 66), où SERVICE est une classe de services (ex. « TRANSPORT »). L’agent répond une liste de ses services du type SERVICE (si le serveur n’est pas compétent, il répond au client une liste vide, sinon il répond la liste des de ses services qui satisfont le cahier des charges : {s_1, s_2, s_3, s_4...} sur la Figure 66).

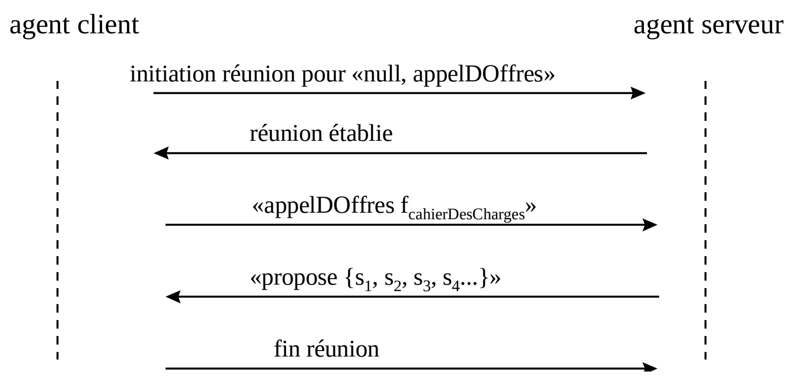


Figure 66. Conversation de base entre l’agent client et un agent serveur

Lorsqu’il reçoit les propositions de l’agent serveur interrogé, l’agent client utilise la fonction « cahier des charges » $f_{cahierDesCharges}$ pour les filtrer, afin de déterminer celles qui satisfont le cahier des charges.
La fonction $f_{cahierDesCharges}$ est donnée sous forme procédurale : c’est un objet Java, qui a une méthode `boolean match(Services service)`(la classe de l’objet implémente une interface —au sens « Java » du terme— définissant la spécification de cette méthode)
La méthode `match` prend en paramètre un service (un comportement de l’agent serveur) et renvoie vrai si elle vérifie le cahier des charges.
Cette technique dispense d’avoir à définir un langage de description de contraintes pour décrire le cahier des charges de manière déclarative.

6.5.3 Obtention d’un service

La conversation pour l’obtention d’un service est en plusieurs étapes (Figure 67) :

- l’agent client initie la conversation (il initie une réunion avec l’agent serveur). Le client précise quel service il souhaite que le serveur active. Dans cette partie, le client paie le serveur une somme précise, mais les paiements n’ont pas été implémentés.
- le serveur décide d’accéder à la requête du client ou non (en théorie, ce choix est par exemple déterminé par le montant du paiement effectué par le client. En pratique, dans l’implémentation actuelle, le serveur accepte toujours la requête du client). Si le serveur

refuse de fournir le service au client, alors la conversation suit la partie 2.1 puis s'arrête. Si le serveur accepte, alors la conversation suit la partie 2.2.1 ou 2.2.2.

- si le client décide de la désactivation du service, alors c'est la conversation 2.2.1 qui est suivie
- si le serveur désactive son service de sa propre initiative (pour cause de panne par exemple, alors c'est la conversation 2.2.2 qui est suivie.

Cette conversation est la plus complexe qui ait été implémentée en JNA. C'est en effet la seule dont le protocole ne soit pas du type « question - réponse ». Pour plus de clarté, le protocole est représenté en Figure 68 (les messages pour l'initiation de la réunion et pour sa terminaison n'ont pas été représentés).

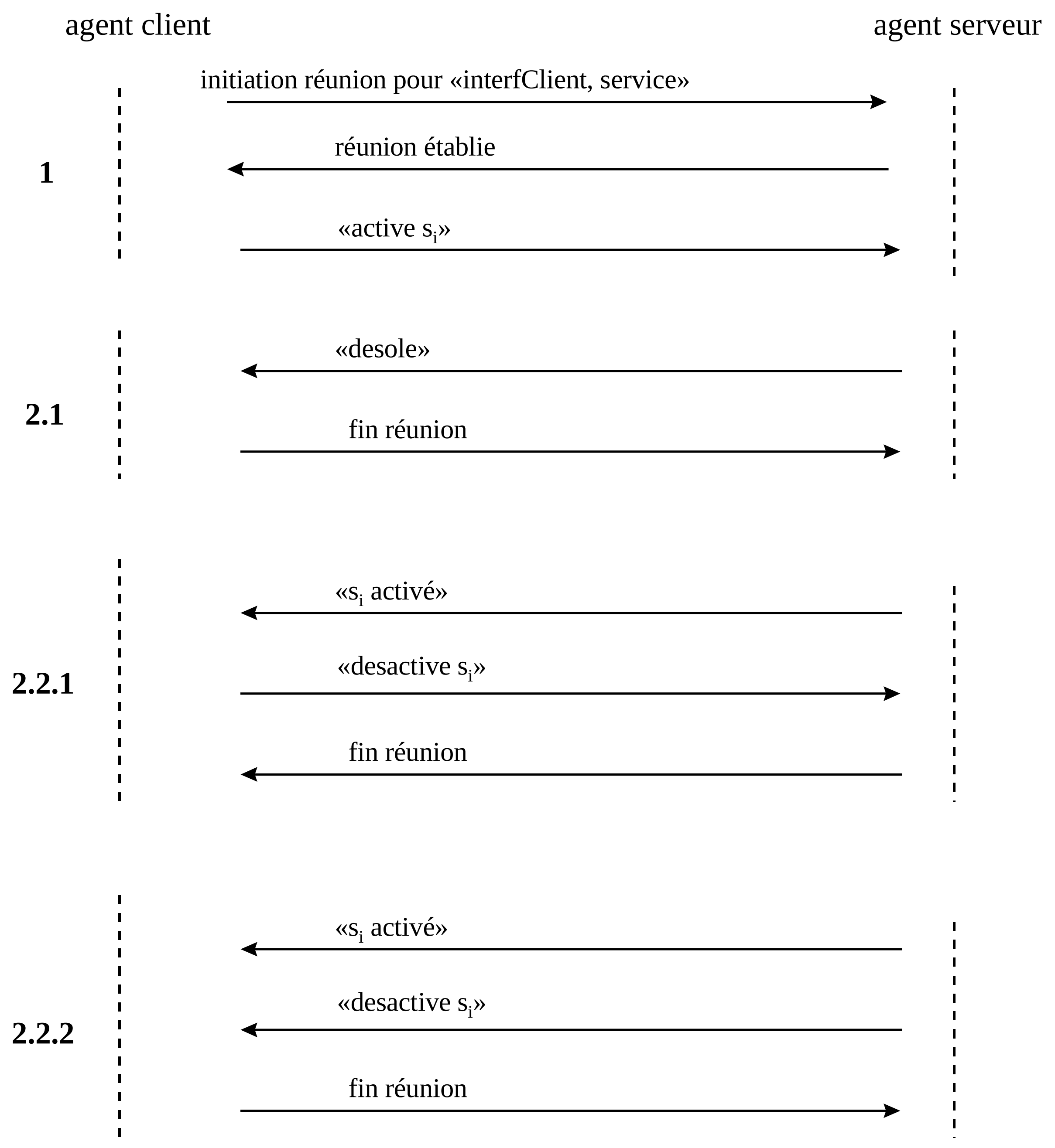


Figure 67. Conversation entre un agent client et un agent serveur.

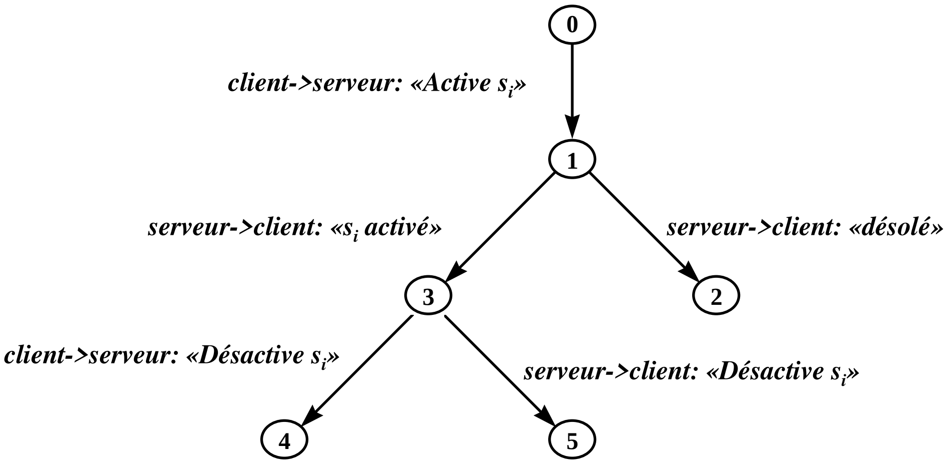


Figure 68. Protocole de la conversation entre le client et l'agent serveur.

Remarque : l'interface du client est « InterfClient », elle permet à l'agent serveur de :
1. renvoyer un message « Désolé » au client si le serveur refuse de fournir le service demandé.
2. prévenir le client de la désactivation du service, en lui envoyant un message « Désactive s_i ».

6.6 Conclusion

La plate-forme JavaNetAgents fournit un environnement d'exécution sécurisé pour agents mobiles. Tout programme Java peut être facilement encapsulé dans un agent. Le choix d'une migration de niveau 2 semble un judicieux compromis entre facilité d'implémentation, et utilité.

JNA a été testé sur Windows95 et sur Solaris, entre l'ONERA et le LIP6.

JNA présente un défaut (qu'elle partage avec toutes les autres plates-formes d'exécution d'agents mobiles) : le manque de services de haut niveau pour les agents mobiles. JNA n'offre aucun service mis à part les services de courtage et de voisinage. La programmation d'applications à base d'agents mobile est pourtant grandement facilitée si les agents mobiles peuvent accéder à des services tels que :

- un service de nommage intégrant la mobilité. Le service de nommage est un annuaire « page blanche », qui donne l'adresse courante d'un agent à partir de la seule donnée du nom de cet agent. Ce service facilite grandement les communications entre agents mobiles : il dispense l'agent émetteur de connaître l'adresse courante du destinataire. Dans le cas d'agents mobiles, les adresses changent souvent ; cette connaissance doit donc être mise à jour en permanence, d'où un énorme surcroît de travail —et une énorme source de bugs potentiels— pour le dévelopeur d'un SCDO qui ne dispose pas d'un service nommage fourni d'office par la plate-forme d'exécution.

- un service « cycle de vie », qui gère la création, le clonage, et la suppression d'agents mobiles.
- etc.

En fait ce sont l'ensemble des services fournis par OMA /CORBA qu'il faudrait recréer pour les agents, en tenant compte de la mobilité des agents. De ce point de vue, le projet d'intégration des agents mobiles à OMA/CORBA (le projet MASIF, introduit plus bas) est particulièrement intéressant.

Ce défaut rend particulièrement fastidieuse la programmation d'application à base d'agents mobiles. A la décharge de JNA (et de ses auteurs ! !), il faut noter la nature expérimentale et *profondément* exploratoire de cette plate-forme. L'objectif de JNA était d'analyser les enjeux, les difficultés et l'apport réel des techniques et des concepts liés à la mobilité de code. De ce point de vue, JNA a pleinement remplit son rôle.

Peu de plates-formes disponibles actuellement offrent des services de haut niveau satisfaisants (la programmation des mêmes applications en Aglets ou en Mole présenterait exactement les mêmes difficultés). Voyager est cependant celle qui se rapproche le plus de la plate-forme idéale.

Deux projets visent à fournir des services de hauts niveaux aux agents mobiles :

1. le projet MASIF de l'OMG, qui a pour objet d'intégrer les agents mobiles à l'architecture OMA/CORBA, *Request For Proposals on Mobile Agent Facility*, [Milojicic et al. 98])
2. Le projet « agents mobiles » de FIPA ([FIPA 98]).

Ces approches semblent les plus prometteurs, car les plates-formes résultantes (OMA/CORBA pour MASIF, et la plate-forme d'exécution d'agent de FIPA) fournissent déjà de nombreux services de haut niveaux.

Bien qu'ayant un objectif commun, ces deux projets s'avèrent plus complémentaires que concurrent. Mais ils sont à un stade trop peu avancé pour qu'un discussion puisse être menée à leur sujet, dans ce mémoire de thèse.

7. Réalisation informatique de l'Agence de Voyage

Ce chapitre présente l'implémentation informatique du système d'Agence de Voyage. Le déroulement du scénario (introduit en §2.4) y est présenté.

7.1 Introduction

Le système Agence de Voyage sert d'exemple de référence tout au long de cette thèse. Logiquement, il sert également de « banc de test » pour le mécanisme d'interaction proposé au §5.

Par manque de temps, et également par souci de simplicité, le système de l'Agence de Voyage n'a été implémenté et testé dans sa totalité. Seuls l'agent AgVoy et les agents représentant les compagnies de transport ont été implémentés. L'agent Dupond ne l'a pas été. Ce choix permet d'expérimenter l'interaction entre AgVoy et les agents de compagnies de transport, ce qui est le fond de notre travail.

Cette implémentation permet d'exécuter le scénario décrit en §2.4, à une restriction près : l'absence implémentation de l'agent Dupond impose de faire démarrer le scénario lorsque AgVoy initie l'interaction pour obtenir les deux services de transport. Le scénario s'arrête :

- lorsqu'il n'est plus possible d'activer un groupe d'agents serveurs, parce que tous les groupes identifiés ont été victimes de défaillances. Dans ce cas l'interaction se termine sur un échec. ou
- ou bien, au bout d'une durée fixée à l'avance (AgVoy envoie alors un message de fin d'interaction aux agents mobiles). Dans ce cas l'interaction se termine sur un succès.

Enfin, il faut noter que l'implémentation a été également minimaliste dans le sens où seules les fonctionnalités indispensables des agents ont été réalisées : en particulier les interfaces graphiques des agents sont très simples, voire inexistantes. Toutefois, le déroulement du scénario peut aisément être suivi grâce aux interfaces graphiques sophistiquées de la plate-forme d'exécution des agents (JNA). Dans le reste de ce chapitre, des copies d'écran de l'agent GraphAgent (§6.4.3) permettent de suivre la progression du scénario.

7.2 *Description générale de l'infrastructure informatique*

Ce paragraphe décrit la configuration générale du système, tel qu'il a été installé sur les sites du LIP6 (Laboratoire d'Informatique de Paris6, à Jussieu) et de l'ONERA (Office National d'Etudes et de Recherches Aérospatiales, à Chatillon).

Lors de son utilisation, cinq places sont crées :

- trois au LIP6, sur les machines oreade, dryade et bilbo. Dans la suite de ce chapitre, elles sont désignées par leurs adresses : oreade.lip6.fr, dryade.lip6.fr, bilbo.lip6.fr
- deux à l'ONERA, sur les machines ubik et talon. Leur adresses sont ubik.onera.fr et talon.onera.fr

Nous avons été obligé de « tricher » au sujet des agents concierges. En effet, l'infrastructure décrite est de taille très réduite : seulement 5 places, réparties sur deux « pôles » (le LIP6 et l'ONERA), eux-mêmes distants de quelques kilomètres seulement (Jussieu - Chatillon). En pratique, les mécanismes de mise à jour automatique des tables d'accointances des agents concierges mènent très rapidement à une situation où tous les concierges connaissaient toutes les places distantes. C'est pourquoi nous avons imposé une « topologie » du système définie sur la Figure 69. Cette topologie définit les accointances des agents concierges. Elle « partitionne » le système en deux : la partie LIP6 et la partie ONERA. Les deux partitions sont reliées par l'intermédiaire des machines dryade au LIP6 et ubik à l'ONERA. Cette topologie rend le scénario plus intéressant, puisque les agents mobiles découvrent « progressivement » l'ensemble des places du système, et non pas instantanément en interrogeant le premier concierge. La Figure 70 est une capture d'écran de la fenêtre graphique du concierge de la place dryade.lip6.fr.

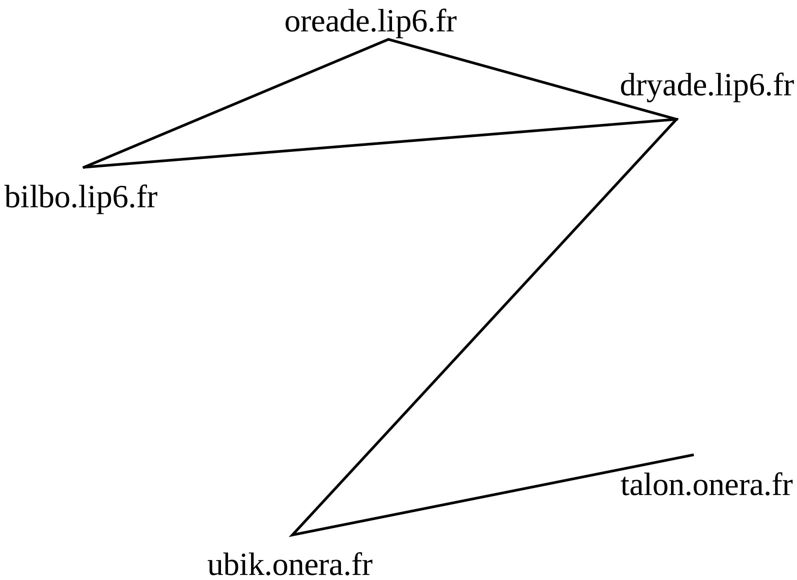


Figure 69. Topologie du réseau de communication

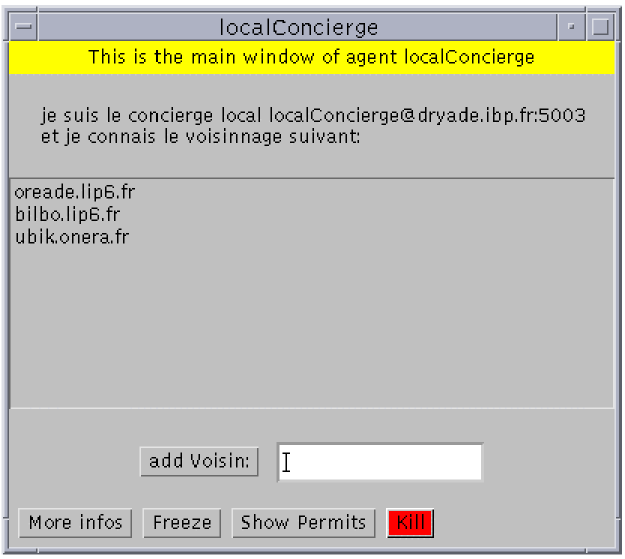


Figure 70. Le concierge de la place dryade.lip6.fr

7.3 L'agent AgVoy

L'agent AgVoy est installé sur oreade.lip6.fr (cf. la Figure 71, qui est une capture d'écran de la fenêtre du GraphAgent de cette place). Son rôle dans le scénario est celui de l'agent client : il initie une interaction pour obtenir deux services de transport corrélés (les s1 et s2 du scénario §2.4).

Note : dans toutes les captures d'écran présentées dans ce chapitre, l'agent AgVoy apparaît sous le nom « capa ». Pour des raisons techniques, il ne m'a pas été possible de refaire des captures d'écran de JNA aux BTLabs afin de mettre à jour le nom de l'agent représentant l'agence de voyage.

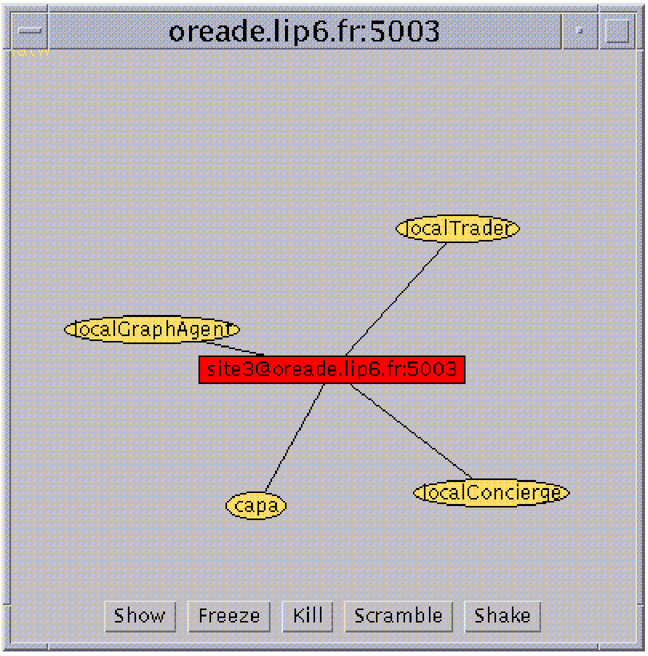


Figure 71. L'agent AgVoy sur la place oreade.lip6.fr.

7.4 Les serveurs de compagnies de transport

Les serveurs sont installés à raison de un par place :

- AirFrance sur dryade.lip6.fr (la Figure 72 est la capture d'écran de la fenêtre du GraphAgent de cette place. La Figure 73 est la capture d'écran de la fenêtre du trader local à cette place.)
- TWA sur ubik.onera.fr
- SNCF sur talon.onera.fr
- AirBourgogne sur bilbo.lip6.fr

Le serveur AirBourgogne (noté AB) a été rajouté pour enrichir le scénario : c'est une compagnie de transport aérien régionale (il existe en fait un petit aéroport à Dijon, qui peut accueillir de petits avions mais qui n'offre pas de desserte internationale). AirBourgogne concurrence SNCF sur le trajet Dijon-Paris.

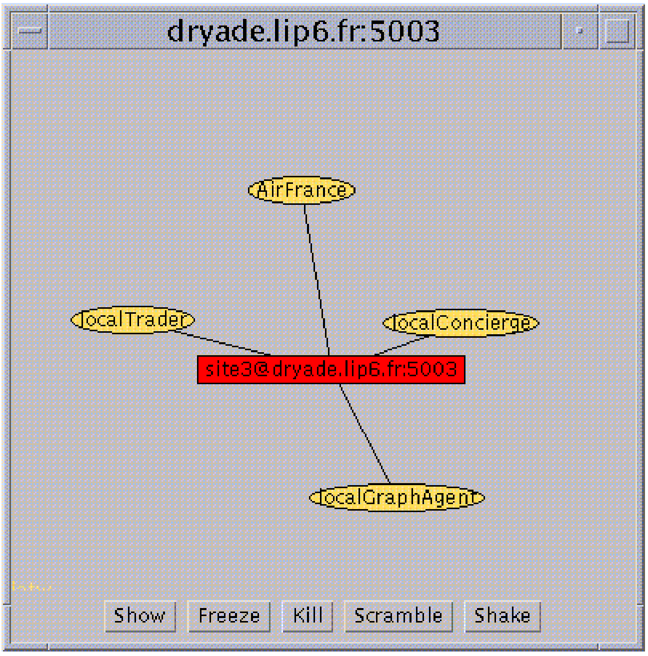


Figure 72. Place dryade.lip6.fr

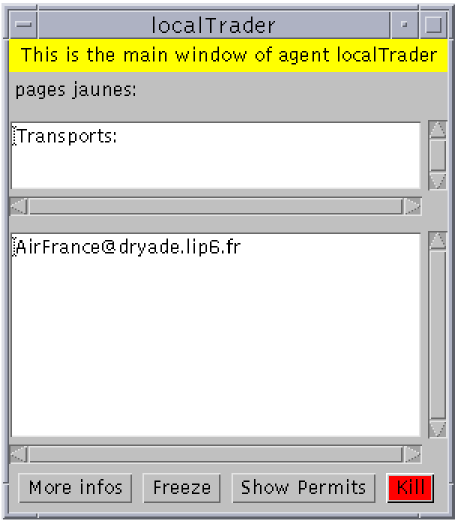


Figure 73. Fenêtre du trader local de la place dryade.lip6.fr

Les services proposés sont rappelés dans le tableau suivant (Tableau 5):

	SNCF	**SNCF**	**SNCF**	**AirBourgogne**
TRANSPORT	s_{SNCF1}	s_{SNCF2}	s_{SNCF3}	s_{AB}
lieuDep	Dijon	Dijon	Dijon	Dijon
lieuDest	Paris	Paris	Paris	Paris
dateDep	16h30	18h30	20h30	19h00
dateDest	18h00	20h00	22h00	19h30
prix	320	320	320	1200

	AirFrance	**AirFrance**	**TWA**
TRANSPORT	s_{AF1}	s_{AF2}	s_{TWA}
lieuDep	Paris	Paris	Paris
lieuDest	NY	NY	NY
dateDep	19h00	21h00	21h00
dateDest	01h00	03h00	03h00
prix	5000	5100	5200

Tableau 5. Services proposés par les agents serveurs.

Les groupes satisfaisant les besoins d'AgVoy, classés par ordre de préférence (les plus rapides d'abord) sont :

1. $((s1, s_{SNCF1}), (s2, s_{AF1}))$
2. $((s1, s_{SNCF2}), (s2, s_{AF2}))$
3. $((s1, s_{SNCF1}), (s2, s_{AF2}))$
4. $((s1, s_{SNCF1}), (s2, s_{TWA}))$
5. $((s1, s_{SNCF2}), (s2, s_{TWA}))$
6. $((s1, s_{AB}), (s2, s_{AF2}))$
7. $((s1, s_{AB}), (s2, s_{TWA}))$

7.5 Le scénario complet

Dans ce scénario, AgVoy crée deux agents mobiles : A1 et A2 (cf. Figure 74. Sur cette figure, A1 et A2 sont chacun en réunion avec AgVoy pendant lesquelles A1 et A2 reçoivent de AgVoy le cahier des charges du service à rechercher).

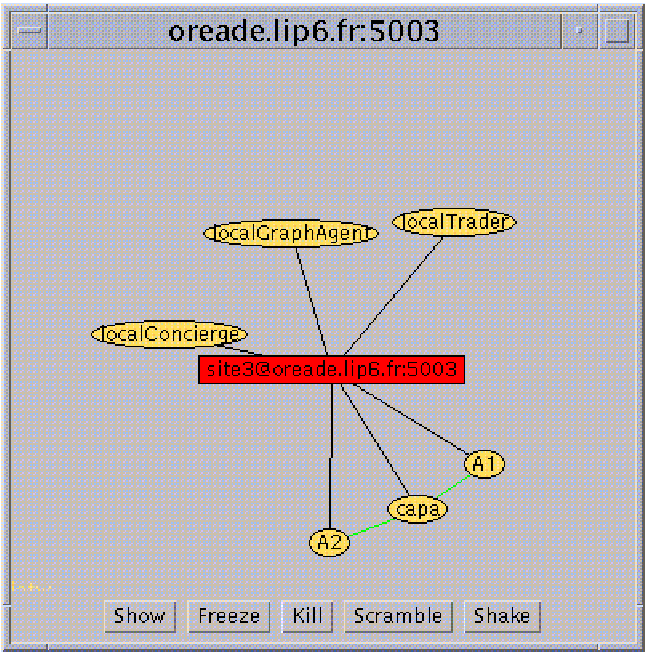


Figure 74. Création des deux agents mobiles A1 et A2.

7.5.1 Recherche d'agents compétents.

A1 et A2 recherchent des agents compétents. A1 et A2 utilisent les cahiers des charges décorrélés décrits en §2.4

- pour s1 : type=TRANSPORT ∧ 12juin10h30<s1.dateDep ∧ s1.lieuDep=Dijon ∧ s1.lieuDest=Paris ∧ s1.dateDest<13juin8h00 ∧ s1.prix<7000fr
- pour s2 : type=TRANSPORT ∧ s2.dateDest<13juin8h00 ∧ s2.lieuDep=Paris ∧ s2.lieuDest=NewYork ∧ 12juin10h30<s2.dateDep ∧ s2.prix<7000fr

Chaque agent mobile se déplace sur le réseau, interrogeant concierges et traders locaux, puis les agents serveurs locaux qui ont été désignés par les traders. Dans l'implémentation actuelle des agents mobiles, leur parcours est exhaustif et du type « premier agent cité, premier interrogé » (décrit en §7.6.1).

L'itinéraire suivi par A1 est décrit dans ce qui suit. Les numéros d'étapes indiqués entre crochet correspondent à ceux indiqués sur la Figure 75 (pour des rasions de clarté, l'agent A2 n'apparaît pas sur la figure. Il est créé en même temps que A1 et effectue un travail similaire à A1).

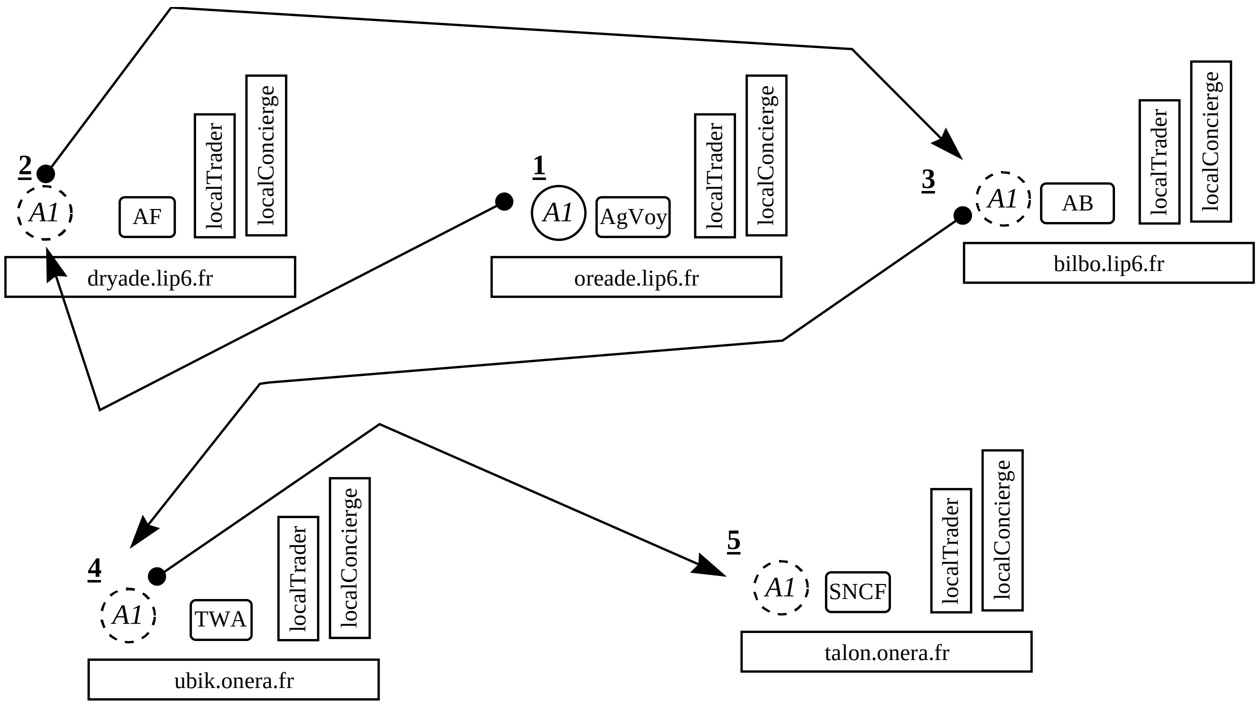


Figure 75. Itinéraire de A1 dans la recherche d'agents compétents.

oreade.lip6.fr [étape 1]
C'est la place de création de A1. A1 interroge successivement :

1. le concierge : il récupère les adresses dryade.lip6.fr et bilbo.lip6.fr.
2. le trader : il ne récupère aucune adresse.

Les connaissances de A1 sont alors :

places distantes connues	serveurs signalés	comportements satisfaisants identifiés
oreade.lip6.fr (place de départ) dryade.lip6.fr bilbo.lip6.fr	aucun	aucun

dryade.lip6.fr [étape 2]
A1 interroge successivement :

1. le concierge : il récupère les adresses oreade.lip6.fr, bilbo.lip6.fr, et ubik.onera.fr (le meeting entre A1 et le concierge est visualisé dur la Figure 76, qui la capture d'écran du GraphAgent local, prise au moment même de la réunion. Un lien relie les deux agents pour marquer la réunion en cours).
2. le trader : il récupère l'adresse d'AirFrance (cf. Figure 77)
3. AirFrance : il récupère les propositions s_{AF1} et s_{AF2}. Aucune ne convient (cf. Figure 78).

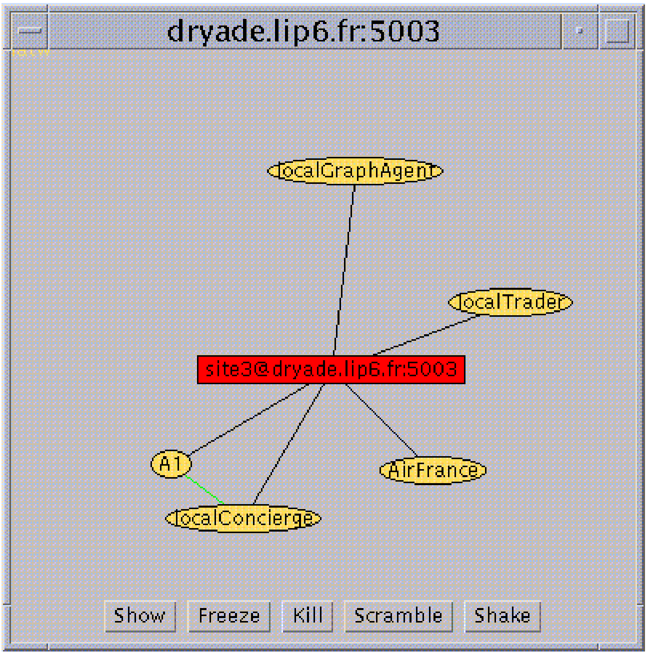


Figure 76. Réunion entre A1 et le concierge local.

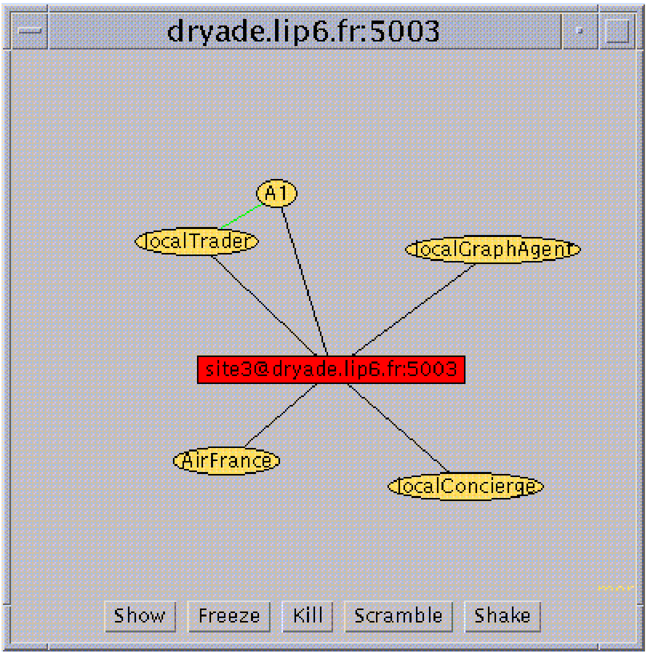


Figure 77. Réunion entre A1 et le trader local.

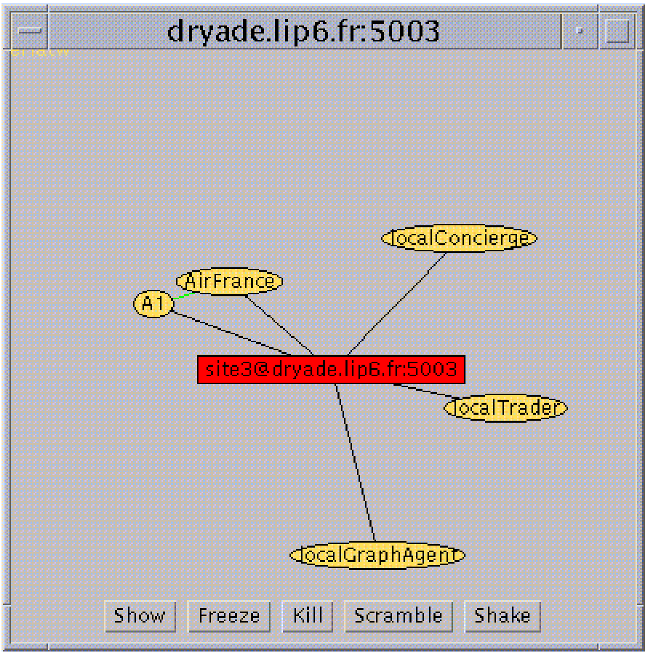


Figure 78. Réunion entre A1 et l'agent AirFrance.

Les connaissances de A1 sont alors :

places distantes connues	serveurs signalés	comportements satisfaisants identifiés
oreade.lip6.fr (place de départ) dryade.lip6.fr (visitée) bilbo.lip6.fr ubik.onera.fr	AF	

bilbo.lip6.fr [étape 3]

A1 interroge successivement :

1. le concierge : il récupère les adresses oreade.lip6.fr et dryade.lip6.fr. Aucune adresse qu'il ne connaissait déjà.
2. le trader : il récupère l'adresse d'AirBourgogne
3. AirBourgogne (Figure 79) : il récupère les propositions s_{AB}. Cette proposition convient : A1 vient d'identifier un premier agent serveur compétent.

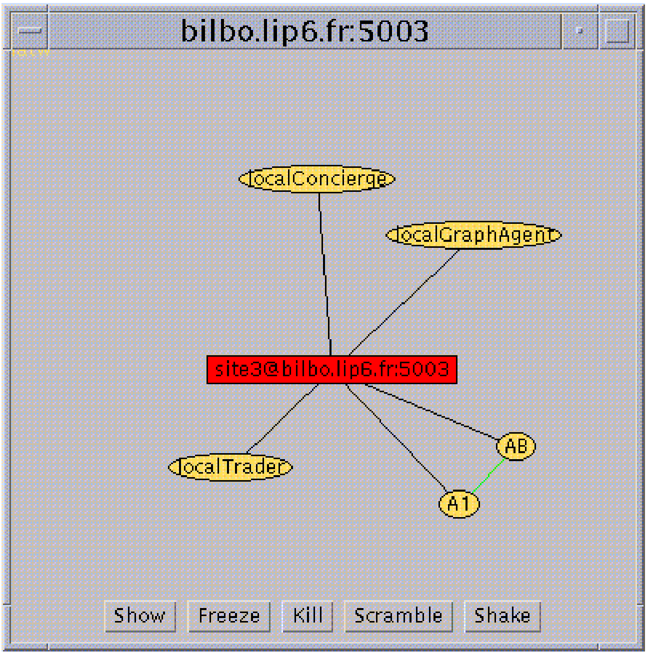


Figure 79. Réunion entre A1 et l'agent AirBourgogne.

Les connaissances de A1 sont alors :

places distantes connues	serveurs signalés	comportements satisfaisants identifiés
oreade.lip6.fr (place de départ) dryade.lip6.fr (visitée) bilbo.lip6.fr (visitée) ubik.onera.fr	 AF AB	 s_{AB}

ubik.onera.fr [étape 4]

A1 interroge successivement :

1. le concierge : il récupère les adresses talon.onera.fr et dryade.lip6.fr.
2. le trader : il récupère l'adresse de TWA
3. TWA (Figure 80) : il récupère les propositions s_{TWA}. Cette proposition ne convient pas.

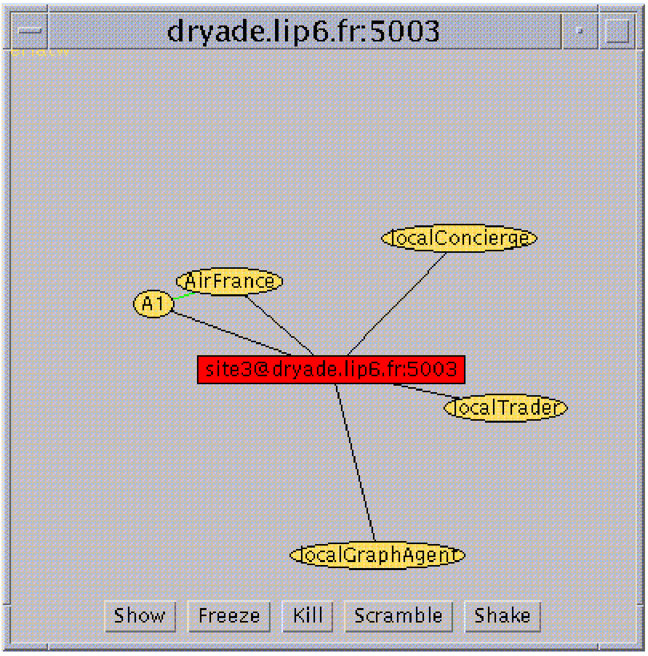


Figure 80. Réunion entre A1 et l'agent TWA.

Les connaissances de A1 sont alors :

places distantes connues	serveurs signalés	comportements satisfaisants identifiés
oreade.lip6.fr (place de départ) dryade.lip6.fr (visitée) bilbo.lip6.fr (visitée) ubik.onera.fr (visitée) talon.onera.fr	 AF AB TWA	s_{AB}

talon.onera.fr [étape 5]

A1 interroge successivement :

1. le concierge : il récupère les adresses ubik.onera.fr .
2. le trader : il récupère l'adresse de SNCF.
3. SNCF (Figure 81) : il récupère les propositions s_{SNCF1} , s_{SNCF2} , s_{SNCF3}. Ces propositions conviennent.

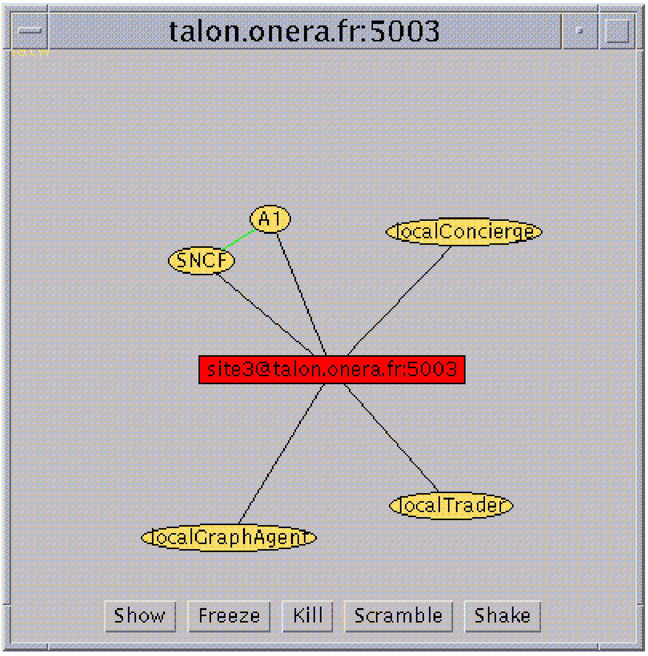


Figure 81. Réunion entre A1 et l'agent SNCF.

Les connaissances de A1 sont alors :

places distantes connues	serveurs signalés	comportements satisfaisants identifiés
oreade.lip6.fr (place de départ)		
dryade.lip6.fr (visitée)	AF	
bilbo.lip6.fr (visitée)	AB	s_{AB}
ubik.onera.fr (visitée)	TWA	
talon.onera.fr (visitée)	SNCF	s_{SNCF1} , s_{SNCF2} , s_{SNCF3}

Toutes les places connues ont été visitées : la phase « recherche d'agents compétents » est finie pour A1.

A la fin de cette phase, A1 et A2[38] se trouvent sur talon.onera.fr et ont identifiés les comportements satisfaisants —avec les agents serveurs qui les proposent— suivants (ordonnés par ordre de préférence) :

pour A1 :	pour A2 :
s_{SNCF1} , s_{SNCF2} , s_{SNCF3} , s_{AB}	s_{AF1} , s_{AF2} , s_{TWA}

A1 et A2 envoient alors chacun leur liste à AgVoy au moyen d'un message distant (i.e. A1 et A2 ne migrent pas vers AgVoy mais se content de lui envoyer un message).

7.5.2 Calcul des groupes d'agents serveurs.

A réception des deux listes, AgVoy calcule les groupes et les classe par ordre de préférence :

1. ((s1, s_{SNCF1}), (s2, s_{AF1}))
2. ((s1, s_{SNCF2}), (s2, s_{AF2}))
3. ((s1, s_{SNCF1}), (s2, s_{AF2}))
4. ((s1, s_{SNCF1}), (s2, s_{TWA}))
5. ((s1, s_{SNCF2}), (s2, s_{TWA}))
6. ((s1, s_{AB}), (s2, s_{AF2}))
7. ((s1, s_{AB}), (s2, s_{TWA}))

AgVoy envoie ensuite cette liste aux deux agents A1 et A2, qui attendent sur talon à l'ONERA.

[38] A2 a en fait suivi le même itinéraire que A1 (oreade - dryade - bilbo - ubik - talon), et il se trouve donc sur la même place que A1 à la fin de cette phase. Cela est dû au fait que les deux agents mobiles recherchent le même type de service (des services de types TRANSPORT), et qu'ils visitent les places dans l'« ordre d'acquisition » auprès des agents concierges : la première place citée par le premier concierge est visitée en premier, etc. Si les deux agents mobiles cherchaient des services de types différents, ils parcourraient un itinéraire différent.

7.5.3 Sélection et activation d'un groupe

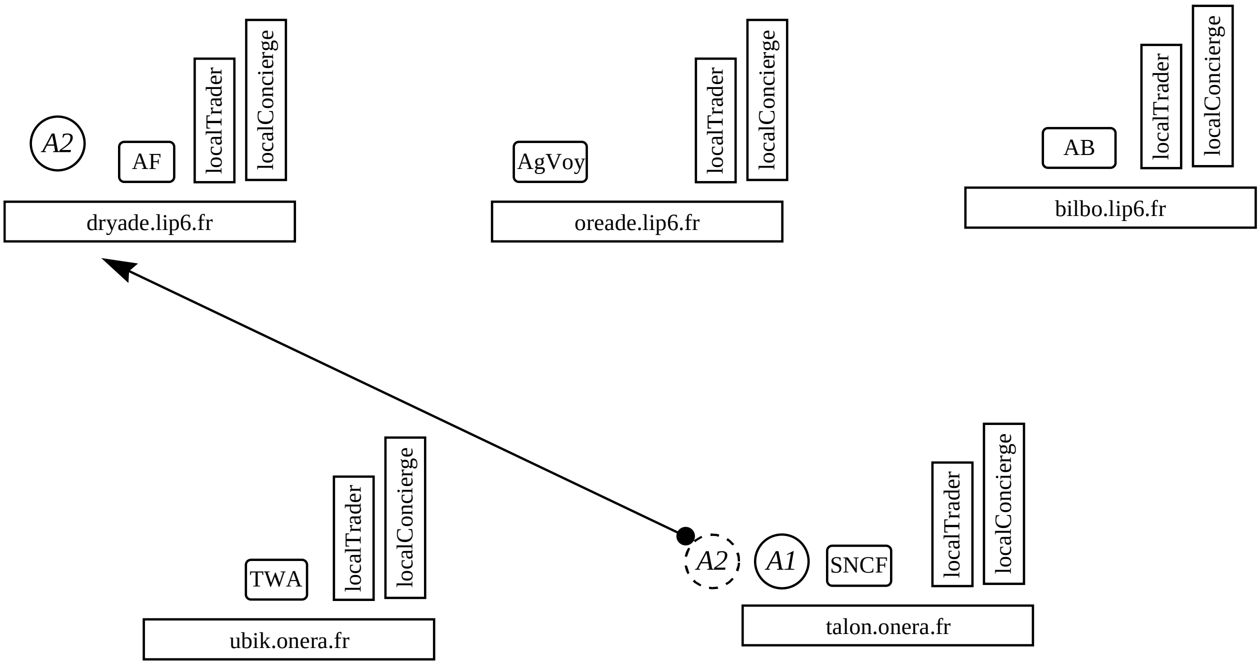


Figure 82. Activation du groupe ((s1, s_{SNCF1}), (s2, s_{AF1})).

A réception de la liste des groupes, chaque agent mobile sélectionne le premier groupe et fait activer le comportement qui le concerne [étape 6 sur la Figure 82] :

1. A1 n'a pas besoin de migrer puisqu'il est déjà sur talon. Il initie une conversation `Active` (cf. §6.5.3) avec SNCF pour faire activer s_{SNCF1}. Puis A1 envoie un message à AgVoy pour le notifier de l'activation de s_{SNCF1}.
2. A2 migre sur dryade, et fait activer s_{AF1} par AirFrance en initiant avec lui une conversation `Active`. Puis A2 envoie un message à AgVoy pour le notifier de l'activation de s_{AF1}.

Deux défaillances successives de AirFrance conduisent A2 a sélectionner les groupes de remplacements :

1. ((s1, s_{SNCF2}), (s2, s_{AF2})). C'est le groupe suivant dans la liste. A2 envoie un message à A1 pour lui demander de faire activer s_{SNCF2} à la place de s_{SNCF1}. Simultanément, A2 fait désactiver s_{AF1} et fait activer s_{AF2} en remplacement. Quand A2 a confirmation de l'activation de s_{SNCF2} de la part de A1, il avertit AgVoy du changement de groupe de serveurs. Lorsque ce groupe est activé, les agents mobiles ont toujours la même position que sur la Figure 82.
2. ((s1, s_{SNCF1}), (s2, s_{TWA})). Le groupe ((s1, s_{SNCF1}), (s2, s_{AF2})) n'a pas été sélectionné, bien qu'il soit plus « prioritaire » que ((s1, s_{SNCF1}), (s2, s_{TWA})), car le précisément le service s_{AF2} est défaillant. A2 migre sur la place d'accueil de TWA (ubik.onera.fr) et initie une conversation `Active` avec TWA pour faire activer s_{TWA}. Puis il avertit AgVoy du changement de groupe. Lorsque ce groupe est actif, les agents mobiles ont la position illustrée par la Figure 83.

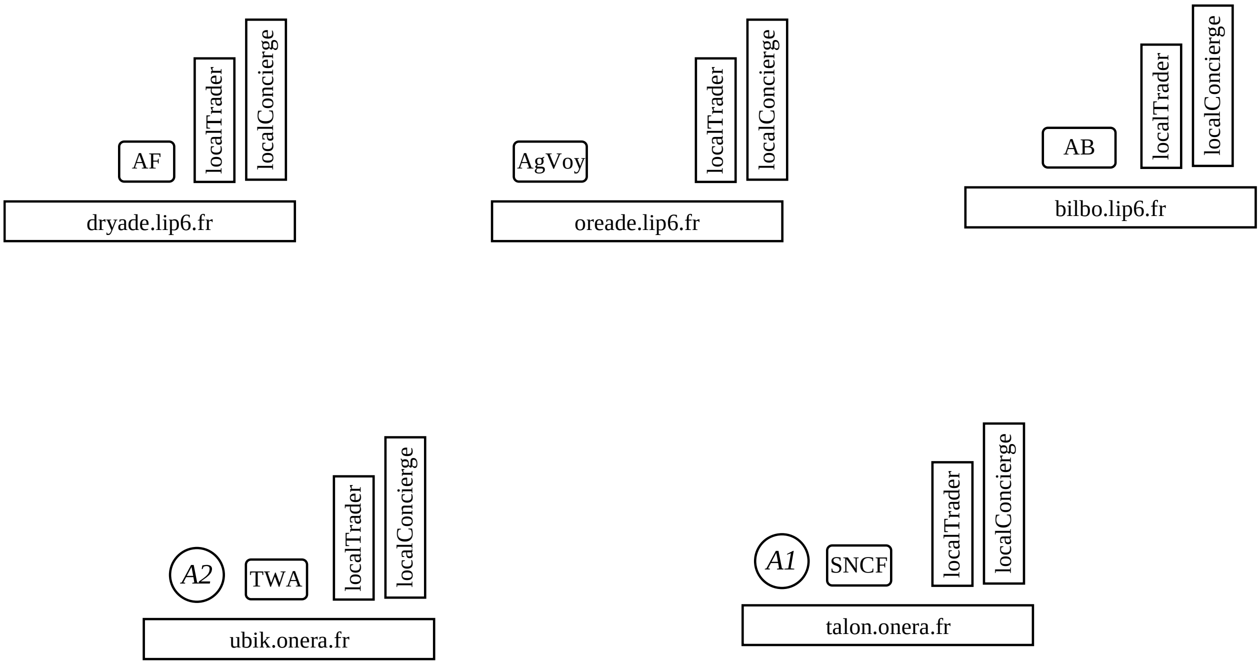


Figure 83. Position des agents mobiles lorsque le groupe ((s1, s_{SNCF1}), (s2, s_{TWA})) est actif.

Une défaillance force SNCF à désactiver s_{AF2} . A1 répète la procédure déjà appliquée deux fois par A2 :

1. sélection d'un groupe de remplacement : c'est ((s1, s_{AB}), (s2, s_{TWA})).
2. notification à A2, qui *n'annule pas* s_{TWA} puisque le comportement est également présent dans le groupe de remplacement..
3. simultanément à (2), A1 migre sur la place d'accueil de AirBourgogne (bilbo.lip6.fr) et initie une conversation `Active` avec AB pour faire activer s_{AB}.
4. lorsque A1 reçoit confirmation de l'activation de s_{AB}, alors il avertit AgVoy du changement de groupe (inutile d'attendre de message de la part de A2 puisque A2 n'a pas à changer de comportement).

Lorsque ce groupe est actif, les agents mobiles ont la position illustrée par la Figure 84.

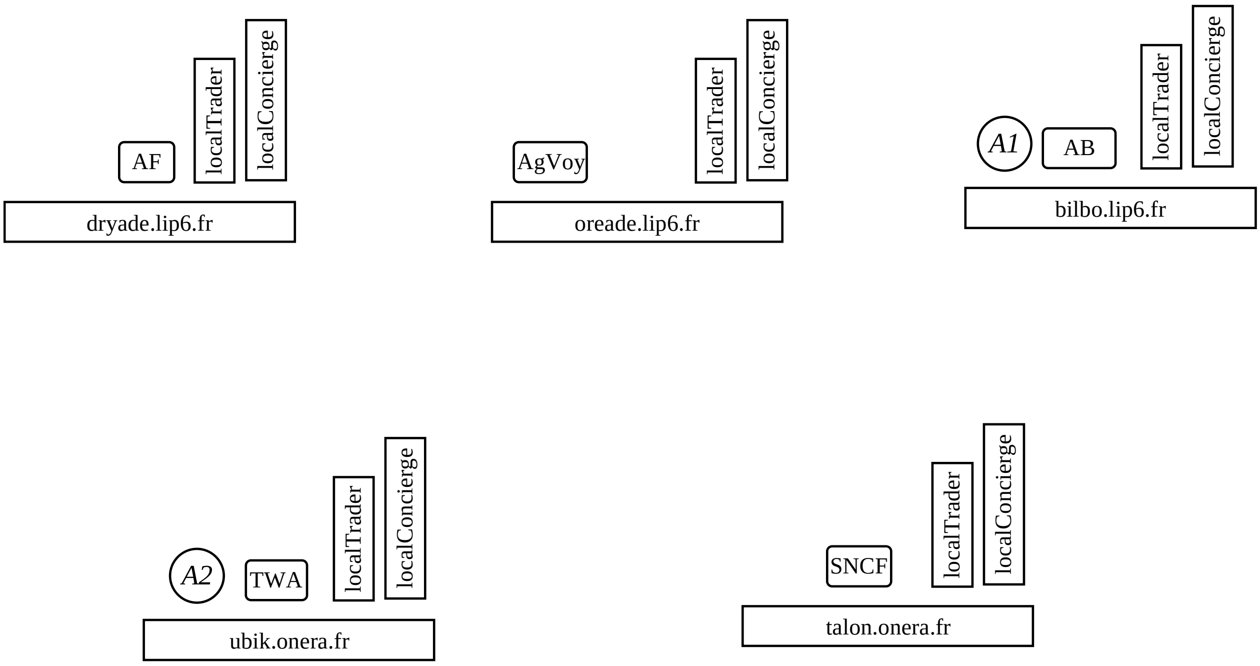


Figure 84. Position des agents mobiles lorsque le groupe ((s1, s_{AB}), (s2, s_{TWA})) est actif.

Enfin, l'interaction est finie : AgVoy envoie un message à A1 et A2. Les agents mobiles font désactiver les comportements s_{AB} et s_{TWA} et envoient chacun un message de confirmation à AVoy, avant de disparaître.

7.6 Remarques

Plusieurs remarques sont à faire concernant l'implémentation actuelle du système de l'Agence de Voyage :

1. la simplicité de stratégie de parcours du réseau, appliquée par les agents mobiles lorsqu'ils sont à la recherche d'agents serveurs compétents.
2. l'importance primordiale d'un service de nommage permettant de joindre (i.e. envoyer un message) facilement tout agent mobile, où qu'il soit dans le réseau et même s'il se déplace en permanence.
3. la portabilité du mécanisme d'interaction, puisqu'une implémentation identique a été réalisée en utilisant la plate-forme d'exécution d'agents mobiles Voyager.

Ces points sont détaillés dans ce qui suit.

7.6.1 Simplicité la stratégie de parcours du réseau

Dans l'implémentation actuelle des agents mobiles, leur parcours est exhaustif et du type « premier cité, premier interrogé ». L'itinéraire des agents mobiles, i.e. la succession de places qu'ils visitent, est déterminé par l'ordre de citation des places : chaque agent va d'abord sur la place qui lui a été indiquée en premier par le premier agent concierge qu'il a interrogé, puis sur la deuxième place indiquée par ce concierge, etc. D'autre part, sur chaque place, les agents mobiles interrogent les agents serveurs par ordre de citation du trader : ils interrogent d'abord le premier agent cité par le trader, puis le second etc. Cette procédure simpliste explique pourquoi les deux agents mobiles A1 et A2 suivent exactement le même parcours...

Des procédures plus sophistiquées, et donc plus efficaces, sont bien sûr souhaitables : les agents mobiles pourraient aller d'abord vers les places où se trouvent le plus de serveurs compétents, etc. Mais de telles procédures nécessitent l'utilisation de traders globaux, qui connaissent les comportements d'agents serveurs situés sur des places voisines en plus des comportements des serveurs locaux. Les traders de JavaNetAgents ne sont pas assez sophistiqués. L'extension de la fonctionnalité des traders, et l'utilisation de stratégies de parcours de réseau plus sophistiquées sont bien sûr au programme des améliorations à apporter au système de l'Agence de Voyage.

Toutefois, il est important de noter que le mécanisme d'interaction proposé dans cette thèse est totalement indépendant de la stratégie utilisée par les agents pour la recherche des agents serveurs compétents.

7.6.2 Importance du service de nommage

Les problèmes rencontrés lors de l'implémentation de l'Agence de Voyage illustrent l'importance des services de haut niveau fournis par la plate-forme d'exécution d'agents mobiles, et en particulier du service de nommage. En effet, les communication avec les agents mobiles sont particulièrement difficiles à mettre en oeuvre en pratique, précisément parce que ces agents se déplacent et changent alors d'adresse.

JavaNetAgents ne dispose pas actuellement de service de nommage, c'est pourquoi la réalisation du scénario complet est extrêmement fastidieuse sur cette plate-forme : sans service de nommage, il faut gérer « à la main » tous les changements d'adresse —fréquents pour les agents mobiles ! —. Cette contrainte est lourde et source potentielle de nombreux *bugs*... Dans l'implémentation réalisée, ce problème a été contourné ainsi : l'envoi de message aux agents mobiles est réalisé par *broadcast* : chaque message est en fait dupliqué et ses copies sont envoyées sur toutes les places du système[39] ! Cette solution est bien évidemment très mauvaise en terme de bande passante consommée, mais présente l'avantage d'être rapide à programmer[40] : par manque de temps, il n'a pas été possible de réaliser un service de nommage sur JNA.

---

[39] Les messages qui n'arrivent pas à la « bonne » place sont simplement détruits.

[40] Il est d'autant plus facile de programmer ce type de diffsuon de message, que les messages JNA sont en fait des objets mobiles. La classe BoradcastMessage est sous classe de la classe Message (elle-même sous-classe de MobileObject), et les méthodes send et noReceiver sopnt modifiéés de manière à

7.6.3 Portabilité du système.

A titre d'information, une autre implémentation du scénario a été réalisée, en Voyager cette fois-ci. Voyager disposant d'un service de nommage, cette implémentation a été beaucoup plus facile et rapide. Cependant par manque de temps, elle ne dispose pas d'interface graphique, et seule une « trace » texte est disponible pour suivre l'exécution du scénario. Nous ne la détaillerons pas ici, car elle est fonctionnellement semblable à la première. Son seul intérêt est de prouver la « portabilité » du système d'Agence de Voyage et du modèle d'interaction proposé dans cette thèse.

7.7 Conclusion

Ce scénario illustre bien le processus d'interaction : la recherche, la sélection, l'activation et la réaction à la désactivation d'un groupe (ou de plusieurs) pour obtenir des services corrélés. Les avantages liés à l'utilisation d'agents mobiles ont été soulignés dans §5.10.4.

De très nombreuses améliorations peuvent bien sûr être apportées au système de l'Agence de Voyage (stratégie de parcours du réseau à la recherche d'agents serveurs compétents, etc.). Toutefois celles-ci ne sont pas prioritaires, compte tenu que le mécanisme d'interaction est indépendant de ces « détails » d'implémentation. En outre, l'implémentation réalisée n'est en rien spécifique à une application du type « Agence de Voyage » : d'autres types de SCDOs peuvent être facilement implémentés en réutilisant la méthode est même une partie du code développé ici.

En ce qui concerne JNA, la principale amélioration à apporter à cette plate-forme concerne l'ajout d'un service de nommage qui facilite les communications avec les agents mobiles. Les agents mobiles étant de simples outils d'implémentation, il est indispensable que leur emploi soit le plus « transparent » possible. De ce point de vue, la plate-forme JNA, dans sa version actuelle, est largement insuffisante... comme la quasi-totalité des plates-formes d'exécution d'agents mobiles en Java[41] ! ! ! Toutefois, il est important de noter que JNA n'a été développé que parce que, au début de ce travail de thèse, aucune plate-forme satisfaisante n'existait. Comme ce n'est plus le cas aujourd'hui, le développement de JNA est arrêté : la plate-forme Voyager, l'une des meilleures disponibles actuellement, dispose de toutes les fonctionnalités qui manquaient à JNA.

---

[41] Seules la plate-forme Voyager, et la dernière version de Mole, fournissent un service de nommage. Les autres n'offrent aucun service de ce type, qui apparaît pourtant indispensable, à la lumière de notre expérience.

8. Conclusion

Ce travail de thèse porte sur l'adaptation dynamique des organisations de SMA. Le premier apport de ce travail est la mise en évidence de l'importance de la corrélation entre les besoins d'un agent client (les services qu'il demande) : leurs descriptions sont mutuellement liées, de sorte qu'un besoin ne peut être défini indépendamment des autres. Il en résulte que les besoins ne peuvent pas être satisfait indépendamment les uns des autres : les agents serveurs satisfaisant chacun des besoins doivent *se regrouper* et agir *en corrélation*. La corrélation des besoins permet de ce fait de rendre compte précisément de la nature exacte et du rôle des phénomènes collectifs que sont les regroupements dynamiques d'agents au sein d'un SMA.

Un mécanisme d'interaction a été proposé, afin d'assurer l'adaptation dynamique de l'organisation du SMA. L'interaction est initiée par un agent client afin de satisfaire ses besoins. Elle conduit au regroupement d'un ensemble d'agents serveurs (définis par analogie avec un problème de satisfaction de contraintes), qui chacun fournit un service qui satisfait un des besoins du client. L'interaction assure l'adaptation dynamique du regroupement en cas de défaillance de certains de ces membres. Dans cette situation, un autre regroupement le remplace. Ce processus est très largement décentralisé et il est exécuté par un ensemble d'agents mobiles : l'agent client n'intervient qu'une seule fois après avoir initié l'interaction.

L'utilisation d'agents mobiles contribue beaucoup à l'efficacité de ce mécanisme d'interaction dans les conditions d'exploitation caractéristiques des SMA de type SCDO : il est économe en bande passante (peu de messages distants sont échangés), et tolérant aux pannes machines/réseau. Il est à noter que le choix de cette technologie novatrice a nécessité de construire une plate-forme d'exécution d'agents mobiles. Au moment où ce travail de thèse a commencé, aucune plate-forme n'était en effet disponible.

Les limitations de notre travail sont nombreuses. D'une part, le calcul des regroupements d'agents serveurs est fait de manière centralisée. L'utilisation d'algorithmes de CSP décentralisés et dynamiques, comme l'algorithme de Ghédira, n'a pas pu être réalisée par manque de temps.
D'autre part, l'implémentation de l'Agence de Voyage a permis de mieux cerner les besoins d'un développeur de SMA de type SCDO. En particulier, il apparaît indispensable qu'une plate-forme

d’exécution du SMA fournisse des services « de haut niveau ». Parmi ceux-ci, outre la mobilité d’agents, des services de nommage et de courtage sont également essentiels. Ce constat permet d’envisager l’utilisation d’architectures de systèmes distribués comme OMA/CORBA, pour servir de base à l’implémentation des SMA. Ces architectures fournissent en effet déjà de nombreux services de haut niveau. Ce constat confirme également l’intérêt de l’architecture FIPA, qui fournit elle aussi les services de haut niveau souhaités. Dans cette perspective, il est particulièrement important d’identifier les conséquences de l’introduction de la mobilité dans ces architectures. Les travaux portant sur le support de la mobilité dans l’architecture FIPA sont de ce fait très prometteurs.

Enfin, de nombreuses applications industrielles de ce travail sont envisageables concernant ce travail. Les entreprises du 21ème siècle seront « virtuelles » : elles seront constituées du regroupement dynamiques de plusieurs entreprises de petites tailles, chacune spécialisée dans un domaine particulier. La compétitivité d’une entreprise virtuelle tient à son « agilité », c’est-à-dire à sa capacité à s’adapter dynamiquement afin de réagir aux évolutions rapides du marché. L’élément moteur de sa dynamique est bien sûr son système d’information, un SMA de type SCDO. De ce point de vue, l’interaction assure (1) la création de l’entreprise virtuelle pour satisfaire les besoins du client, et (2) l’adaptation dynamique de son organisation en réaction à des événements imprévus (défaillance de certains de ses membres). D’une manière plus générale, les concepts multi-agents développés dans cette thèse, et la technologie des agents mobiles, joueront vraisemblablement un rôle important dans ces domaines.

9. Références bibliographiques.

[Acharya 96] Ranganathan, Acharya, Shamik Sharma, Saltz. *Network-aware Mobile Programs*. Department of Computer Science, Université du Maryland

[Agrawal et al. 91] D. Agrawal, A. Malpani. *Efficient Dissemination of Information in Computer Networks.* The computer Journal. 1991.

[Austin 63] J. Austin. *How To Do Things With Words?*. Oxford University Press. 1963.

[Barat et al. 92] M. Barat, J. Erceau, M. Castellanet. *Conception Distribuée Coopérative. Application à la Conception de Satellite*. rapport ONERA. 1992

[Barbuceanu et al. 94] M. Barbuceanu, M. Fox. *The Information Agent: An Infrastructure Agent Supporting Collaborative Entreprise Architectures*. 3rd Workshop on Enabling Technologies: Infrastructure for Collaborative Entreprises. 1994.

[Berlandier et al. 92] P. Berlandier, B. Neveu, B. Trousse. *PROSE: un Langage de contraintes Intégrable dans un Système à base de Connaissances*. Avignon'92.

[Berlandier et al. 93] P. Berlandier, B. Trousse. *Expression et Interprétation de MetaContraintes pour le maintient de Solution en Conception*. Avigon'93.

[Bharat et al. 96] K. Bharat, Cardelli. *Migratory Applications*. SRC Research Report. Digital Equipment Company. 1996

[Castelfranchi et al.93] C. Castelfranchi, A. Cesta, R. Conte, M. Miceli. *Foundations for Interactions: the Dependance Theory*. AA*IA'93. LNCS 728.

[Castelfranchi 94] C. Castelfranchi. *Guarantees for Autonomy in Cognitive Agent Architecture*. ECAI Workshop on Agent Theories, Architectures, and Languages 1994: 56-70

[Chess et al. 95] D. Chess, C. Harrison, A. Kershenbaum. *Mobile Agents: Are they a good idea?* IBM tech report. disponible sur http://www.ibm.com.

[Cohen et al. 95] P. Cohen, H. Levesque. *Communicative Actions for Artificial Agents*. Comptes-rendus de ICMAS'95.

[Conte et al. 91] R. Conte, M. Miceli, C. Castelfranchi. *Limits and levels of cooperation: Disentangling various forms of social interaction*. In Y. Demazeau & J.P. Mueller (Eds.) Decentralized AI - 2, Amsterdam, Elsevier, 1991.

[Cutkosky et al. 93] M. Cutkosky, R. Engelmore, R. Fikes, M. Genesereth, T. Gruber, W. Mark, J. Tenenbaum, J. Weber. *PACT: An Experiment in Integrating Concurrent Engineering Systems.* IEEE Computer Jan 1993.

[Davis et al. 83] R. Davis, R. Smith. *Negociation as a Metaphor for Distributed Problem Solving.* Articial Intelligence vol 3. 1983.

[Demers et al. 87] Demers, Greene, Hauser, Irish, Larson, Shenker, Sturgis, Swinehart, Terry. *Epidemic Algorithms for Replicated Database Maintenance*. Proceedings of the 6th Symposium on Distributed Computing.

[Demers et al. 95] Demers, Petersen, Spreitzer, Terry, Theimer, Welch. *The Bayou Architecture: Support for Data Sharing among Mobile Users*. Tech Reprt. Computer Science Lab. Xerox PARC. diposnible sur http://www.parc.xerox.com

[Di Marzo et al.95] Di Marzo, Muhugusa, Tschudin, Harms. *The Messenger Paradigm and its Implications on Distributed Systems.* Comptes-rendus de the ICC'95 Workshop ICC (Intelligent Computer Communication).

[Ferber 95] J. Ferber. *Les Systèmes Multi-Agents*. InterEditions. 1995.

[Finger et al. 94] S. Finger, S. Konda, F. Prinz, D. Siewiorek, E. Subrahmanian, M. Tenenbaum, M. Cutkosky, L. Leifer, R. Bajcsy, W. Birmaingham. *ACORN: Creating an Advanced Collaborative Open Ressource Network.* 6th International ACME Conf. on Design Theory and Methodology. 1994

[Finin et al. 94] T. Finin, R. Frotzson, D. McKay, R. McEntire. *KQML as an Agent Communication Language*. 3rd International Conference on Information and Knowledge Managment. 1994

[FIPA 97] Foundation for Intelligent Physical Agents. *Personal Travel Assistance*. Draft Specification Part 4. 1997.

[FIPA 98] Foundation for Intelligent Physical Agents. *Agent Management Support for Agent Mobility* (Dublin July 1998). Disponible sur http://drogo.cselt.stet.it/fipa/

[Franklin et al. 96] S. Franklin, A. Graesser. *Is it an Agent, or just a Program? A Taxonomy for Autonomous Agents*. rapport technique univ. de Memphis, Tennessee. disponible sur http://www.msci.memphis.edu/~franklin

[Frost 95] Frost. *Java(tm) Agent Template, Version 0.3.* Stanford University. Disponible sur http://cdr.stanford.edu/ABE/JavaAgent.html

[Genesereth et al. 92] M. Genesereth, R. Fikes. *Knowledge Interchange Format. Version 3. Reference Manual*. rapport technique univ. Stanford. disponible sur http://ksl.stanford.edu. 1992

[Genesereth et al. 93] M. Genesereth, Ketchpel. *Software Agents*. Comm. of the ACM, 1993.

[Genesereth et al. 94] M. Genesereth, N. Singh. *A Knowledge Sharing Approach to Software Interoperation*. rapport technique univ. Stanford. 1994

[Ghedira 93] K. Ghedira. *MASC: une approche Multi-Agents des problèmes de satisfaction de contraintes*. Thèse de l'ENSAE. 1993

[Ghedira 94] K. Ghedira. *Partial Constraint Satisfaction by a Multi-Agent-Simulated Annealing Approach*. Avignon'94.

[Ghedira et al. 92] K. Ghedira, G. Verfaillie. *Approche Multi-Agents d'un Problème de Satisfaction de Contraintes: Optimalité et Réactivité*. Avigon'92.

[Ginberg 91] M. Ginsberg. *Knowledge Interchange Format: The KIF of Death.* AI Magazine. Fall 1991.

[Girard-Faugère 98] M. Girard-Faugère. *ADAM : un gestionnaire d'applications réparties en environnement dynamique*. Thèse de l'Université de Paris 6. 1998

[Goldszmidt et al. 95] G. Goldszmidt, Y. Yemini. *Distributed Managment by Delegation*. 15th International Conf. on Distributed Computing Systems. 1995.

[Hale et al 94-2] M. Hale, J. Craig. *Preliminary report of Agent Technologies for a Design Integration Framework*. 5th Symposium on Multi-Disciplinary Analysis and Optimization. 1995.

[Hale et al. 94] M. Hale, J. Craig. *IMAGE: A Design Integration Framework Applied to the High Speed Civil Transport*. Aerospace Systems Design Lab. Tech Report. Georgia Institute of Technology.

[Herbert 94] A. Herbert. *An ANSA Overview*. IEEE Network. Jan/Feb 1994.

[I^3 95] Program on I^3 ARPA. *Reference Architecture for the Intelligent Integration of Information (I^3)*. disponible sur http://www-db.stnford.edu. 1995

[Java 95] Sun Microsystem. *Java: An Overview*. Disponible sur http://www.javasoft.com. 1995.

[Jennings et al. 96] N. Jennings, P. Faratin, M. J. Johnson, P. D. O'Brien, M. E. Wiegand. *Using Intelligent Agents to Manage Business Processes*. PAAM'96.

[Kafura & al.98] D. Kafura, J.-P. Briot. *Introduction to the Special Series on Actors and Agents*. IEEE Concurrency, 1998.

[Keller 95] A. Keller. *CIT's Work on CommerceNet*. rapport technique univ. Stanford. disponible sur http://cit.stanford.edu. voir aussi http://www.commercenet.org

[Knabe 95] F. Knabe. *Language Support for Mobile Agents*. PhD Thesis. Carnegie Mellon University. 1995.

[KQML 92] KQML Advisory Group. *An Overview of KQML: A Knowledge Query and Manipulation Language.* rapport technique univ. Stanford. 1992

[Kuokka et al. 94] D. Kuokka, B. Livezey. *A Collaborative Parametric Design Agent*. AAAI'94.

[Kuokka et al. 95] D. Kuokka, L. Harada. *Matchmaking for Information Agents*. Comptes-rendus de IJCAI'95.

[Lange et al. 96] D. Lange, Chang. *IBM Aglets Workbench. A White Paper*. Disponible sur http://www.trl.ibm.co.jp/aglets/

[Levesque et al. 90] H. Levesque, R. Cohen. *On Acting Together*. AAAI'90.

[Lhuillier 98] M. Lhuillier. *Une approche à base de composants logiciels pour la conception d'agents. Principes et mise-en-œuvre à travers la plate-forme MALEVA*. Thèse de l'Université de Paris 6. 1998

[Li 96] Li, Messerschmitt . *Java-To-Go : Itinerative Computing Using Java* . Disponible sur http://ptolemy.eecs.berkeley.edu/dgm/javatools/java-to-go/

[Linden 93] R. van der Linden. *An Overview of ANSA*. disponible sur http://www.ansa.org. 1993

[Mayfield et al. 95] J. Mayfield, Y. Labrou, T. Finin. *Desiderata for Agent Communication Language*. ICMAS'95.

[McGuire et al. 93] J. McGuire, D. Kuokka, J. Weber, J. Tenebaum, T. Gruber, G. Olsen. *SHADE Technology for knowledge-based cooperative engineering. Concurrent Engineerring: Research & Applications*. 1(3). 1993.

[Merlat et al. 97] W. Merlat, C. Seyrat, J. Ferber, *Mobile-Agents for Dynamic Organizations: The Conversational-Agent Paradigm*, Poster à 8th European Workshop on Modeling Autonomous Agents in a Multi-Agent World (MAAMAW'97), 1997

[Merlat et al. 97-2] W. Merlat, C. Seyrat, *JavaNetAgents : Une Plate-forme d'Exécution d'Agents Mobiles pour le Développement de Systèmes Multi-Agents sur Internet.* JFIADSMA'97

[Merlat 96] W. Merlat *Intérêts de l'Approche Multi-Agents pour la Conception des Réseaux Intelligents*. RJCIA'96

[Milojicic et al. 98] D. Milojicic, M. Breugst, L. Busse, J. Campbell, S. Covaci, B. Friedman, K. Kosaka, D. Lange, K. Ono, M. Oshima, C. Tham, S. Virdhagriswaran, J. White. *MASIF: The OMG Mobile Agent System Interoperability Facility*. Mobile Agent MA'98, 1998.

[Müller 95] J. Müller. *A Markovian Model for Interaction among Behavior-Based Agents*. IJCAI'95 Workshop on Intelligent Agents. 1995.

[Nielsen 95] J. Nielsen. *Transactions in Mobile Computing*. Skrifligt arbejde no. 95-2 11. DIKU, forår 1995.

[NIIIP95] Le consortium NIIIP. *NIIIP: An Overview*. Diponible sur http://www.niiip.org

[O'Brien et al. 97] P. D. O'Brien, M. E. Wiegand, B. Odgers, C. Voudouris, D. Judge. *Using Software Agents for Business Process Management*. BT Technical Journal, vol. 15, 1997.

[OMG92] Object Managment Group. *Object Managment Architecture Guide*. Disponible sur http://www.omg.org

[Oracle95] Oracle Company. *Oracle Mobile Agents*. *Technical Product Summary*. Disponible sur http://www.oracle.com

[Pan et al. 91] J. Pan, J. Tenenbaum. *An Intelligent Framework for Entreprise Integration*. IEEE Trans. on Systems, Man and Cybernetics. Nov/dec 1991.

[Park et al. 94] H. Park, J. Tenenbaum, R. Dove. *Agile Infrastructure for Manufacturing Systems*. Lockheed Tech Report. disponible sur http://space.hitchhiker.lockheed.com.

[Pitoura et al. 95] E. Pitoura, B. Bhargava. *Revising Transaction Concepts for Mobile Computing.* Position Paper. Purdue University. 1995.

[Pittel 87] B. Pittel. *On Spreading a Rumour*. SIAM Journal of Applied Math. Feb. 1987.

[Seyrat 96] C. Seyrat. *JavaNetAgents : Un environnement d'exécution d'agents mobiles pour le développement de systèmes multi-agents coopératifs sur Internet.* Rapport de stage de DEA/IARFA. 1996

[Shoham 93] Y. Shoham. *Agent-Oriented Programming*. Artificial Intelligence. 60. 1993.

[Sichman 98] J. Sichman. *DEPINT : Dependance-Based Coalition Formation in an Open Multi-Agent Scenario*. Journal of Artificial Societies and Social Simulation vol . 1, no. 2. 1998.

[Singh et al. 95] N. Singh, M. Gisi. *Coordinating Distributed Objects with Declarative Interface*. rapport technique univ. Stanford. 1995

[Smith 87] Smith. *The ContractNet Protocol. A High Level Negociation Protocol for Distributed Problem Solving*. *in* Readings in DAI 87.

[Straßer 96] Markus Straßer, Joachim Baumann, Fritz Hohl. *MOLE - A Java Based Mobile Agent System*. Disponible sur http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole.html

[Tardo et al. 95] J. Tardo, L. Valente. *Mobile Agent Security and Telesccript*. General Magic Tech Report. disponible sur http://cnn.genmagic.com. 1995.

[Telescript95] General Magic. *Telescript: An Overview*. disponible sur http://www.genmagic.com

[Tennenhouse et al. 96] D. L. Tennenhouse, D. Wetherall. *Towards an Active Network Architecture*. Proceedings of Multimedia Computing and Networking. 1996.

[Terry et al. 95] Terry, Theimer, Petersen, Demers, Spreitzer, Hauser. *Managing Update Conflict in Bayou, a Weakly Connected Replicated Storage System*. Tech Reprt. Computer Science Lab. Xerox PARC. diposnible sur http://www.parc.xerox.com

[Terry et al. 95-2] Terry, Demers, Petersen, Theimer, Spreitzer, Welch. *Session Guaranties for Weakly Consistent replicated Data*. Tech Reprt. Computer Science Lab. Xerox PARC. diposnible sur http://www.parc.xerox.com

[Theimer et al. 95] Theimer, Demers, Petersen, Spreitzer, Terry, Welch. *Dealing with Tentative Data Values in Disconnected Work Groups*. Tech Reprt. Computer Science Lab. Xerox PARC. diposnible sur http://www.parc.xerox.com

[Tschudin 93] C. Tschudin. *On the Structuring of Computer Communication*. Thèse de l'Université de Genève. 1993

[Tschudin 95] C. Tschudin. *OO-Agent and Messengers*. ECOOP'95 Workshop on Objects ans Agents. 1995.

[Voyager 97] ObjectSpace company. *The Voyager Whitepaper*. Disponible sur http://www.objectspace.com. 1997

[Yoo et al. 97] M. Yoo, W. Merlat, J.-P. Briot. *Modelling and Validation of Mobile Agents on the Web*. WebSim’97.

[Yokoo et al. 92] M. Yokoo, E. H. Durfee, T. Ishida, K. Kuwabara. *Distributed Constraint Satisfactionfor Formalizing Distributed Problem Solving*. Proceedings of the 12th International Conference on Distributed Computing Systems. June 9-12, 1992,Yokohama, Japan. IEEE Computer Society Press, 1992, ISBN 0-8186-2865-0

[Yokoo 95] M. Yokoo. *Asynchronous Weak-commitment Search for Solving Distributed Constraint Satisfaction Problems*. First International Conference, CP'95, Cassis, France, September 19-22, 1995, Proceedings. Lecture Notes in Computer Science, Vol. 976, Springer 1995, ISBN 3-540-60299-2

10. Glossaire.

CORBA	Common Object Request Broker Architecture. Architecture objet assurant l'interopérabilité d'applications hétérogènes et utilisée comme composant de base de l'architecture de nombreux systèmes distribués. Cf. §4.2.2.1
CSP	Constraint Satisfaction Problem. Problème de satisfaction de contraintes.
Environnement	L'environnement d'un agent est constitué des autres agents du SMA (il ne s'agit pas de l'environnement « physique » ou informatique dans lequel se trouve l'agent).
EV	Entreprise Virtuelle, cf. §2.1
Facilitateur	voir trader
Matchmaker	voir trader
OMA	Object Management Architecture. Surcouche de CORBA, qui ajoute de nombreux services indispensables à l'implémentation de systèmes distribués (par exemple un service trader).
SCDO	Système Coopératif Distribué et Ouvert, cf. §2.1
SMA	Système Multi-Agents
Trader	service « page jaune » qui répertorie les agents en les classant par comportements. Ce service est souvent fourni par un agent spécialisé, appelé trader (dans la norme CORBA), ou matchmaker (dans la norme KQML), ou encore facilitateur (dans certains travaux liés à KQML).

11. Annexe 1 : Un peu de code...

11.1 Introduction

Cette annexe présente des extraits du code source réalisé pour l'implémentation du système Agence de Voyage. Le code est en Java à la norme 1.1.
L'implémentation réalisée est très simple car elle fait essentiellement appel aux fonctionnalités fournies par la plate-forme d'exécution (JNA). Des interfaces graphiques très simples ont été réalisées, mais leur code n'est pas décrit dans ce chapitre.

11.2 La classe Service

Les services ont un prix et l'activation du service est conditionnée par le paiement du serveur d'un montant égal à ce prix.
La classe Service contient les attributs suivants :

```
public class Service {
        String nom ;
        int prix ;
        ...
}
```

La classe Transport est sous-classe de Service :

```
public class Transport extends Service {
        String lieuDep ;
        Date dateDep ;
        String lieuDest ;
        Date dateDest ;
        ...
```

```
}
```

11.3 *Offres et réservations de service*

Un agent serveur répond une liste d'offres à un appel d'offres effectué par un agent mobile. Une offre est un couple (service, agent proposant de fournir ce service).
La classe Offre est :

```
public class Offre {
        Service service;
        Serveur serveur ;
        ...
}
```

Après avoir sélectionné une offre parmi celles qui lui ont été faite par les agents serveurs qu'il a rencontrés, un agent mobile négocie le service correspondant auprès de l'agent qui a fait l'offre (voir conversation « Active »). Rappelons que la négociation est un simple acte de paiement électronique (l'agent mobile paie le serveur d'un montant égal au prix d'activation du service. L'acte de paiement n'est pas implémenté lui-même). L'agent mobile obtient une réservation sur le service : une réservation est un triplet (service, agent serveur, agent client), l'agent client est ici l'agent mobile !
La classe Reservation est :

```
public class Reservation {
        Service service;
        Serveur serveur ;
        Agent client ;    //le client est en fait l'agent mobile !
        ...
}
```

11.4 *Implémentation des cahiers des charges et des fonctions de préférence*

11.4.1 L'objet cahier des charges

Les cahiers des charges sont implémentés par des objets « filtreurs de services », dont la classe implémente l'interface Java ServiceFiltre.

```
public interface ServiceFiltre {
        public boolean convient(Offre o) ;
```

```
}
```

Une classe de cahiers des charges a été définie : la classe TransportDeA. Ces cahiers des charges portent sur des services de type Transport, et expriment des contraintes portant sur les dates et lieux de départ et d'arrivée.

```
public class TransportDeA implements ServiceFiltre {
        String lieuDep ;  //le lieu de départ
        Date dateDepMin ; //la date de départ au plus tôt
        String lieuDest ; //le lieu d'arrivée
        date dateDestMax ; //la date d'arrivée au plus tard

        // le constructeur de la classe
        public TransportDeA (String lieuDep, Date dateDepMin,
                                String lieuDest, date dateDestMax) {
              ...
        }

        public boolean convient(Offre o) {
              //le service mentionné dans l'offre
              Service service = o.service ;

              return (    (lieuDep.equals(service.lieuDep))
                          && (dateDepMin.before(service.dateDep))
                          && (lieuDest.equals(service.lieuDest))
                          && (dateDestMax.after(service.dateDest)) ) ;
              }
        }
```

11.4.2 Classe TransportPref

La fonction de préférence permet d'ordonner les groupes par préférence. Les fonctions préférence sont implémentées par des objets dont les classes implémentent l'interface Java GroupePreference :

```
public interface GroupePreference {
        public boolean meilleur(Offre grp1[], Offre grp2[]) ;
}
```

La classe TransportPref implémente la fonction de préférence, qui permet de classer les groupes identifiés au cours de la première phase de l'interaction composite. Dans le scénario, les transports « Dijon NY » les moins chers sont privilégiés, et à prix égal, ceux avec l'escale la plus courte sont préférés.

```
public class TransportPref implements GroupePreference {
        public boolean meilleur(Offre grp1[], Offre grp2[]) {

              //le « Dijon Paris » du premier groupe Transport
              t11 = grp1[0].service ;

              //le « Paris NewYork » du premier groupe
              Transport t12 = grp1[1].service ;

              //le « Dijon Paris » du deuxième groupe
              Transport t21 = grp2[0].service ;

              //le « Paris NewYork » du deuxième groupe
              Transport t11 = grp2[1].service ;

              if (t11.prix+t12.prix<t21.prix+t22.prix) return true ;
              if (t11.prix+t12.prix>t21.prix+t22.prix) return false ;

              return ((t12.dateDep-t11.dateDest
                                  <=((t22.dateDep-t21.dateDest)) ;
              }
        }
```

11.5 Interfaces pour les conversations

Diverses interfaces sont utilisées pour permettre à deux agents en réunion de communiquer.

11.5.1 Interface pour l'initiation d'une réunion

Cette interface est utilisée pour initier la réunion.

```
public interface Reunion {
        // nomAppele : nom de l'agent appelé
        // interfaceAppelant : interface utilisée par l'agent appelant
        // interfaceAppele : interface utilisée par l'agent appelé

        // utilisé par l'agent appelant pour initier la réunion
```

```
        public Object débutReunion(String nomAppele,
                                      String interfaceAppelant,
                                      String interfaceAppele) ;

        //indique la fin de la réunion,
        //ou bien le fait que l'agent appelé refuse la réunion
        public void finReunion() ;
}
```

11.5.2 Interface pour la conversation Voisin

La conversation Voisin a lieu entre un agent mobile et le concierge de la place où il se trouve.

```
public interface Voisin {
        //renvoie la liste des places voisines
        public Vector voisin() ;

        //rajoute l'adresse d'une nouvelle place
        public void addVoisin(AgentAddress place) ;

        //supprime l'adresse d'une place
        public void delVoisin(AgentAddress place) ;
}
```

11.5.3 Interface pour la conversation Serveur

La conversation Serveur a lieu entre un agent mobile et le trader de la place où il se trouve.

```
public interface Serveur {
        //stocke le couple (competence, serveur) dans
        // l'attribut registre du trader
        public void advertise(String competence, AgentAddress serveur) ;

        //renvoie une liste d'adresses d'agents compétents
        public Vector serveur(String competence) ;
}
```

11.5.4 Interface pour la conversation AppelDOffres

La conversation AppelDOffres a lieu entre un agent mobile et un agent de la place où il se trouve (l'adresse de cet agent a été donnée à l'agent mobile par le trader local).

```
public interface AppelDOffres {
        //renvoie une liste de propositions (des objets de type Offre)
        public Vector appelDOffres(String competence) ;
}
```

11.5.5 Interface pour la conversation Active

La conversation Active a lieu entre un agent mobile et un agent serveur de la place où il se trouve, pour obtenir un des services de l'agent serveur. Le service est mentionné dans une offre, que l'agent serveur a préalablement faite à l'agent mobile au cours d'une conversation AppelDOffres.

```
public interface Active {
        //fait « activer » le service mentionné dans l'offre,
        // retourne un objet réservation qui confirme
        // la fourniture du service
        public Reservation active(Offre o) ;

        // fait désactiver le service mentionné dans la réservation
        public void desactive(Reservation resa) ;
}
```

11.5.6 Interface Client

Cette interface est utilisé dans la conversation « Active », pour permettre au serveur de notifier le client (en fait, l'agent mobile qui a initié la conversation « Active ») de la désactivation prématurée du service.

```
public interface Client {
        // notifie le client de la désactivation du service
        // mentionné dans la réservation
        public void desactive(Reservation resa) ;
}
```

11.6 Serveur

Un serveur est un agent qui est susceptible de fournir des services.

```
public class Serveur extends Agent
                    implements Meeting, AppelDOffres, Active {

        // le nom du serveur
```

```
        String nom ;

        //la liste de tous les services que le serveur est
        //susceptible de fournir
        Vector competences ;

        // la liste de Resa : les services actuellement actifs
        Vector reservations ;

        // renvoie la liste d'offres de
        // services de type « typeCompetence »
        public Vector appelDOffres(String typeCompetence) ;

        //active le service mentionné dans l'offre
        public Resa active(Offre o) ;

        //annule une réservation : le service est désactivé
        public void désactive(Resa r) ;
        ...
}
```

11.7 Client

L'implémentation de l'agent client est très minimaliste : le code écrit est en fait seulement celui qui est nécessaire à l'initiation d'une interaction pour obtenir la fourniture de deux services de type transport, corrélés par une relation de compatibilité. Le manque de temps m'a empêché de réaliser une implémentation plus générale (permettant notamment l'initiation d'une interaction pour obtenir plus de deux services). La réalisation d'une classe d'agent client plus générale ne pose pas de problème, compte tenu du caractère très général du mécanisme d'interaction mis en oeuvre.

11.7.1 Classe Client

```
public class Client extends Agent implements Meeting {

        //la relation de compatibilité R(srv1, srv2)
        TransportPref rCompat;
```

```
        //les cahiers des charges décorrélés
        TransportFilter tf1, tf2 ;

        //nombre d'agents mobiles qui ont renvoyés la liste des
        // agents compétents qu'ils ont identifié (ces agents mobiles
        // ont fini la phase « recherche des agents compétents »
        // et attendent la liste des groupes)
        int arrive = 0;

        //résultats reçus
        Vector resus[] = new Vector[2];

        ...
}
```

11.7.2 Création et lancement des agents mobiles

```
public Client(TransportPref rCompat,
                    TransportFilter tf1,
                    TransportFilter tf2,
                    TransportCSP preference,
                    String url ) {

        this.rCompat=rCompat ;
        this.preference=preference ;
        VClient versMoi= (VClient) VObject.forObject(this);

        // obtention d'un proxy sur un trader indiqué par l'URL url
        // url est du type « nom_machine.nom_domaine.fr:no_port »
        VTrader trader = (VTrader) VObject.forObjectAt(url);

        // création du 1er agent mobile
        mobile1 = new VMobile( "localhost:7000/Adapt1" );

        // création du 2eme agent mobile
        mobile2 = new VMobile( "localhost:7000/Adapt2" );

        // lancement du  1er agent mobile
```

```
        mobile1.go( 1, tf1, versMoi, agent mobile2, trader );

        // lancement du 2eme agent mobile
        mobile2.go( 2, tf2, versMoi, agent mobile1, trader );
}
```

11.7.3 Réceptions des agents compétents identifiés par les agents mobiles et calcul des groupes

Une fois que la phase « recherche d'agents compétents » est finie, la méthode recoitOffres permet à un agent mobile de donner au client la liste des agents compétents qu'il a identifiés, i.e. ceux qui ont répondu à l'appel d'offres (les listes sont des listes d'objets Offres)

```
public synchronized void recoitOffres(int rang , Vector liste) {
        //un mécanisme d'exclusion mutuelle (le mot clé « synchonized »)
        //gère le cas de la réception simultanée des deux listes
        // d'agents compétents

        //rang est le rang des offres recueillies
        //liste est la liste des offres recueillies

        resus[rang -1]=liste;   //stocke le résultat
        arrive++ ;  //incrémente le nombre de résultats reçus

        //si toutes les listes sont reçues
        // (il y en a 2 dans le scénario),
        // alors le client calcule les groupes
        if (arrive == 2) calculeGroupes();
}
```

La méthode calculeGroupes permet de calculer les groupes à partir des listes d'agents compétents identifiés :

```
public void calculeGroupes() {
        Vector groupes = new Vector();

        for (Enumeration e1=resus[0].elements(); e1.hasMoreElements();) {

              //o1 est une offre pour le « Dijon Paris »
              Offre o1 = (Offre)e1.nextElement();

              for (Enumeration e2=resus[1].elements();
                                  e2.hasMoreElements();) {

                    //o2 est une offre pour le « Paris NewYork »
                    Offre o2 = (Offre)e2.nextElement();

                    if (rCompat.accept(o1.service, o2.service)) {
                    //les services mentionnés dans o1 et o2
                    // sont mutuellement compatibles, les agents qui
                    // les ont proposés forment donc un groupe
                          Offre grp[]={o1, o2};
                          groupes.addElement(grp);
                    }
              }
        }
```

```
        //la variable groupes contient la liste des groupes

        if (groupes.size()=0) {
              //il n'y a pas de groupes!!!
              // le client ordonne aux agents mobiles de se tuer...
              mobile1.fin();
              mobile2.fin();
        }
        else {
              //il y a des groupes ! !
              // le client les classe par ordre de préférence
              ordonne(groupes, preference) ;

              // puis le client les envoie aux agents mobiles
              mobile1.stockeGroupes(groupes);
              mobile2.stockeGroupes(groupes);
        }
}
```

11.8 Agents mobiles

11.8.1 Classe Agent mobile

```
public class Agent mobile extends Agent implements Client {

        // itinéraire a visiter (liste d'adresses de places distantes)
        Vector itineraire;

        // index dans l'itinéraire
        int index;

        //une référence vers le client
        VClient client;

        //une référence vers l'autre agent mobile
        VAgent mobile collegue ;

        //le cahier des charges décorrélé
```

```
        TransportFilter cahierDeco;

        // rang vaut 1 si l'agent mobile recherche des « Dijon Paris »,
        // rang vaut 2 si ce sont des « Paris NY »
        int rang;

        //la liste ordonnée des groupes
        Vector groupes;

        //indique le service activé actuellement
        Reservation currentResa;
...
}
```

L'agent mobile est « lancé » par la méthode go :

```
public void go(int rang,
                   TransportFilter cahierDeco,
                   VClient client,
                   VMobile collegue,
                   Vtrader trader) {

        this.rang=rang ;
        this.cahierDeco=cahierDeco ;
        this.client=client ;
        this.collegue=collegue ;

        //l'agent mobile migre sur la place accueillant le trader,
        // une fois arrivé sur cette place,
        // il initiera une conversation « serveur » avec le trader
        // pour obtenir des proxies de serveurs compétents
        moveTo( trader, "Serveur" );
}
```

11.8.2 Envoi de la liste des agents compétents identifiés

L'envoi de la liste des agents compétents est effectué par appel de la méthode recoitOffres du Client, elle est décrite dans le paragraphe relatif à la classe Client.

11.8.3 Réception de la liste des groupes

Une fois établie la liste ordonnée des groupes, le client l’envoie aux deux agent mobiles, en appelant (à distance) leur méthode stockeGroupe :

```
public void stockeGroupe(Vector listeGroupes)
        //stocke la liste dans la variable d’instance groupes
        groupes = listeGroupes;

        //sélectionne le premier élément de la liste
        Offre grp1 = (Offre[])(groupes.firstElement());
        selectionne(grp1) ;
}
```

11.8.4 Sélection et activation d’un service

La sélection du service à activer est réalisée par la méthode selectionne() qui prend en paramètre un groupe :

```
public void selectionne(Offre[] groupe) {
        //l’agent mobile sélectionne l’offre qui le concerne
        // dans le groupe
        srvAActiver= groupe[rang-1];

        //l’agent mobile migre sur la place de l’agent serveur
        // qui a fait l’offre
        moveTo( srvAActiver.serveur, "active" );
}
```

Une fois arrivé sur la place du serveur, l’agent mobile fait activer le service mentionné dans l’offre

```
public void active(Serveur serveur) {
        //fabrication d’un proxy sur l’agent mobile,
        // pour que le serveur puisse le notifier de la défaillance
        // de son service
        VAgent mobile versMoi = (VAgent mobile) VObject.forObject(this);
        currentResa=serveur.active(srvAActiver, versMoi);
}
```

11.8.5 Réaction à la défaillance d'un serveur

Le serveur avertit l'agent mobile de la désactivation de son service en appelant la méthode désactive de l'agent mobile (définie dans l'interface Client)

```
public void desactive(Reservation resa) {
        currentResa=null ;
        // l'agent mobile élimine de la liste des groupes, tout groupe
        // contenant le service défaillant
        groupes = epureGroupes(resa.service);

        Offre newGrp[] = null;
        try {
              newGrp = (Offre[])groupes.firstElement();
        }
        catch(NoSuchElementException e) {
              //il n'y a pas de groupe possible en remplacement
              // du groupe défaillant
              //l'agent mobile avertit l'autre agent mobile
              //de l'échec de l'interaction
              collegue.echec();

              //l'agent mobile avertit le client de l'échec
              client.echec();

              // l'agent mobile « se tue »
              fin();
        }

        //le groupe à sélectionner en remplacement du groupe
        // défaillant existe, c'est newGrp
        //l'agent mobile demande à l'autre agent mobile d'activer
        // le service qui le concerne dans le nouvel groupe, et il
        // communique le service défaillant (en fait, l'objet resa)
        // afin que l'autre agent mobile mette également à jour sa
        // propre liste de groupes
        collegue.change(newGrp, resa);

        //l'agent mobile procède à l'activation du service
        // qui le concerne dans le nouveau groupe
        selectionne(newGrp);
```

```
        //l'agent mobile avertit le client du changement de groupe
         client.change(newGrp, resa);
}
```

La méthode change(Offre[] newGrp, Reservation resa) est invoquée sur l'autre agent mobile pour qu'il réagisse à la défaillance du service :

```
public void change(Offre[] groupe, Reservation resa) {
        //l'agent mobile épure sa propre liste de groupes,
        // afin d'éliminer tout groupe contenant le service
        // défaillant
        groupes = epureGroupes(resa.service);

        // l'agent mobile doit désactiver le service actuellement actif,
        // et faire activer le service de remplacement (il n'a pas besoin
        // de le faire si ces deux services sont identiques...)
        if (!currentResa.service.name.equals(groupe[rang-1].service.nom))
        {
              currentResa.serveur.desactive(currentResa);
              selectionne(groupe);
        }
}
```