Agents mobiles et client/serveur : évaluation de performance et comparaison

Daniel Hagimont* — Leila Ismail**

** INRIA Rhône-Alpes*
*** Institut National Polytechnique de Grenoble*

Laboratoire SIRAC (INPG –INRIA - UJF)
INRIA, 655 av. de l'Europe
38330 Montbonnot Saint-Martin
{Daniel.Hagimont, Leila.Ismail}@inrialpes.fr

RÉSUMÉ. *Cet article propose une évaluation de performance du modèle à agents mobiles par comparaison au modèle client/serveur. Cette évaluation a été effectuée dans l'environnement Java en utilisant respectivement RMI, la plate-forme à agents mobiles Aglets et une plate-forme à agents mobiles que nous avons développée. Les mesures donnent à la fois les coûts des mécanismes élémentaires de Java utilisés dans la conception d'un système à agents mobiles, et une évaluation comparative des deux modèles (client/serveur et agents mobiles) à travers deux scénarios d'application. Nous montrons que des gains d'efficacité significatifs peuvent être obtenus en utilisant des agents mobiles.*

ABSTRACT. *This paper presents a performance evaluation of the mobile agent paradigm in comparison to the client/server paradigm. This evaluation has been conducted on top of the Java environment, using respectively RMI, the Aglets mobile agents platform and a mobile agents prototype that we implemented. The measurements give the cost of the basic mechanisms involved in the implementation of a mobile agent platform, and a comparative evaluation of the two considered models (client/server and mobile agents) through two application scenarios. The results show that significant performance improvements can be obtained using mobile agents.*

MOTS-CLÉS : *systèmes répartis, agents mobiles, client/serveur, objets, Java.*

KEY WORDS: *distributed systems, mobile agents, client/server, objects, Java.*

1. Introduction

Le modèle client/serveur est certainement le modèle le plus utilisé pour la construction d'applications réparties. Il est notamment utilisé dans les environnements Corba [OMG 91] et Java [GOS 95] à travers un service d'appel d'objet à distance.

Récemment, la recherche en systèmes répartis a vu l'émergence d'un nouveau modèle pour la structuration d'applications réparties : la programmation par agents mobiles [FUG 98] [THO 97]. Dans ce modèle, un agent est un processus possédant un contexte d'exécution, incluant du code et des données, pouvant se déplacer de machine en machine (appelées serveurs) afin de réaliser la tâche qui lui est assignée. Les agents mobiles visent principalement des applications réparties sur des réseaux à grande distance, car ils permettent de déplacer l'exécution vers les serveurs et de diminuer ainsi le coût d'accès à ces serveurs [CHE 94].

Cependant, alors que le domaine des agents mobiles intéresse beaucoup de chercheurs, peu d'entre eux ont proposé une évaluation de performance de cette technologie moyennant des applications réparties exécutées sous des conditions réelles sur l'Internet. Carzaniga et al. ont proposé un modèle simple d'évaluation de performance des technologies à base de code mobile [CAR 97]. Un modèle de performance pour des applications réalisant une sélectivité sur une base de données utilisant alternativement des appels de procédures à distance (RPC) et des agents mobiles a été examiné par Straßer et al. [STR 97]. D'autres travaux évaluent les performances des mécanismes d'un système à agents mobiles particulier dans un réseau local [FUK 97] [JOH 97].

Dans cet article, nous présentons une évaluation de performance du modèle à agents mobiles par comparaison au modèle client/serveur et ceci moyennant des applications simples exécutées sous des conditions réelles de l'Internet. Cette évaluation a été effectuée dans l'environnement Java en utilisant respectivement RMI (le service d'appel d'objet à distance fourni par Java), la plate-forme à agents mobiles Aglets (une des plates-formes construites sur Java les plus connues) et une plate-forme à agents mobiles minimale que nous avons développée et dont nous maîtrisons les détails de l'implantation. Les mesures effectuées donnent à la fois les coûts des mécanismes élémentaires de Java utilisés dans la conception d'un système à agents mobiles, et une évaluation des deux modèles (client/serveur et agents mobiles) à travers deux scénarios d'application.

Les résultats de cette évaluation montrent que :

– alors que Java fournit les mécanismes permettant de réaliser des plates-formes à agents mobiles, ces mécanismes restent très coûteux.

– malgré ces mécanismes coûteux, des gains d'efficacité significatifs peuvent être obtenus en utilisant des agents mobiles, principalement lorsqu'un agent client communiquant intensivement avec un serveur se déplace vers ce serveur.

Le reste de l'article est structuré de la façon suivante. Dans la section 2, nous rappelons les principes de base du modèle à agents mobiles, puis nous décrivons la plate-forme à agents mobiles que nous avons développée ainsi que les mesures concernant les mécanismes élémentaires de Java utilisés dans sa réalisation. La section 3 présente les mesures comparant les deux modèles en utilisant deux scénarios d'application. Nous concluons cet article en section 4.

2. Modèle à agents mobiles

De nombreux systèmes à agents mobiles ont été développés ces dernières années. Des exemples sont les systèmes AgentTcl [GRA 96], Telescript [WHI 94], Aglets [LAN 97a] [LAN 98a] [VEN 97], Mole [BAU 98] ou MOA [MIL 98]. Une grande partie de ces systèmes sont construits sur l'environnement Java [GOS 95], principalement en raison de sa forte diffusion dans le monde, mais aussi de ses avantages techniques : masquage de l'hétérogénéité des machines et typage fort pour la sécurité.

Nous rappelons tout d'abord les fonctions générales des systèmes à agents mobiles, puis, après un rappel des fonctions de Java utilisées dans la conception d'une plate-forme à agents, nous décrivons notre plate-forme. Nous terminons cette section par une évaluation quantitative des différents composants constituant la plate-forme.

2.1. *Fonctions des agents mobiles*

Un agent est un processus pouvant se déplacer de machine en machine afin de réaliser une tâche. En général, la mobilité est fournie par le biais d'une primitive *move(machine)* qui permet de se déplacer vers la machine désignée par le paramètre.

Un agent est composé de son code correspondant à un algorithme, ainsi que d'un contexte incluant des données. Ce contexte peut évoluer en cours d'exécution, par exemple en collectant des données lorsqu'un agent réalise une recherche d'information sur un ensemble de serveurs. Le code et le contexte de l'agent sont déplacés avec l'agent lorsque celui-ci visite différents serveurs.

Lorsqu'un agent se déplace vers un serveur, il doit poursuivre son exécution sur le site destination. La plupart des systèmes à agents mobiles implantent une migration faible[1], c'est-à-dire une fonction de migration où l'agent redémarre son

[1] En particulier ceux construits sur Java qui ne fournit pas un mécanisme de migration de processus léger.

exécution depuis le début. En conséquence, le programmeur doit inclure dans le contexte de l'agent des informations sur l'état de l'exécution, et lorsque l'agent redémarre sur un site, le code de l'agent doit vérifier l'état de l'exécution et se brancher sur la partie de l'algorithme devant être exécutée sur ce site.

Un système à agents fournit en général des primitives de communication permettant aux agents d'interagir entre eux, mais aussi aux agents d'interagir avec les serveurs qu'ils visitent. Ces primitives de communication prennent la forme d'envois de messages ou d'appels de procédures ou de méthodes.

2.2. *Mécanismes de Java utilisés*

Java est un langage de programmation orienté-objets qui permet de développer des programmes que l'on peut exécuter sur la machine virtuelle Java (JVM). La JVM est disponible sur la plupart des systèmes actuels.

Concernant la mobilité, les principaux mécanismes de Java sont les suivants :

– Mobilité et portabilité du code. Une des propriétés essentielles de Java est que le code généré par le compilateur Java est mobile. Ceci signifie que le code peut être déplacé entre différentes machines. La mobilité du code nécessite que le code soit portable. La portabilité du code produit par Java provient du fait que le code (appelé *byte code*) est indépendant de toute architecture de machine et qu'il est interprété par la JVM. Afin de permettre la mobilité du code, Java fournit la notion de chargeur de classe[2] (*ClassLoader*). Il est possible de définir des chargeurs spécifiques aux applications. Un chargeur est appelé par la JVM chaque fois que celle-ci doit traduire un nom de classe en référence Java à l'objet classe associé. Si la classe n'est pas déjà chargée, le chargeur est responsable du chargement de la classe à partir d'une source de données (disque ou machine distante). Plus précisément, le chargeur appelé pour traduire le nom d'une classe incluse dans une classe C est le chargeur qui a chargé la classe C. Ainsi, en chargeant une première classe en utilisant explicitement un chargeur, toutes les classes accessibles depuis cette classe seront chargées par ce même chargeur. La notion de chargeur permet donc à la fois de charger des classes depuis des sources de données quelconques, et de définir des espaces de noms de classes différents (un par chargeur).

– Liaison dynamique. La liaison dynamique est un aspect crucial pour l'utilisation de code mobile. La liaison dynamique signifie ici la possibilité de ne déterminer que lors de l'exécution le code qui sera exécuté pour un appel de méthode. Etant donné que Java permet le chargement dynamique des classes, une variable d'un type donné (une interface en Java) peut contenir une référence Java pointant sur une instance dont la classe a été chargée dynamiquement. Java retarde la liaison du code à partir de cette variable jusqu'à l'appel effectif, permettant ainsi l'exécution de code chargé dynamiquement.

---

[2] Que nous appellerons *chargeur* dans la suite.

– Sérialisation. Java fournit un mécanisme de sérialisation d'objets [RIG 96] permettant d'échanger les instances entre des JVM différentes. Ce mécanisme permet de traduire un graphe d'objets Java en un flot d'octets qui peut être redirigé vers un fichier ou vers une autre machine à travers le réseau. Pour qu'une instance soit sérialisable, il faut que sa classe implante l'interface *Serializable*. La sérialisation d'une instance revient à écrire tous ses champs dans le flot d'octets. Lorsqu'un de ses champs est une référence à un autre objet, l'objet référencé est aussi sérialisé. Afin de spécialiser le processus de sérialisation, Java permet de définir dans une classe une méthode *writeObject* qui surcharge le mécanisme de sérialisation pour les instances de cette classe. Cette méthode doit définir les champs de l'objet qui doivent être écrits et permet notamment de contrôler le graphe d'objets suivant lequel la sérialisation doit se propager. La dé-sérialisation est le processus inverse de la sérialisation. Le graphe d'objets peut être reconstruit à partir du flot d'octets obtenu par sérialisation (la méthode permettant la surcharge est *readObject*).

Dans la sous-section suivante, nous décrivons comment ces mécanismes ont été utilisés dans la réalisation de notre plate-forme.

2.3. *Notre plate-forme sur Java*

Afin de mener notre expérimentation, nous avons réalisé une plate-forme à agents minimale sur l'environnement Java. Cette plate-forme minimale nous a permis d'identifier de façon fine les mécanismes mis en œuvre dans la gestion d'agents mobiles et d'évaluer les coûts de ces mécanismes.

Dans cette plate-forme, un agent s'exécute sur une machine virtuelle que nous appelons serveur d'agent. Un serveur d'agent fournit une interface permettant de recevoir un agent et une interface permettant à un agent de se déplacer vers un autre serveur d'agent. Un serveur d'agent fournit également des points d'entrée correspondant aux services offerts aux agents sur ce site (par exemple une base de donnée). Un agent utilise un tel service par appel de méthode. Lorsqu'un agent arrive sur un serveur d'agent, il peut consulter un serveur de noms symboliques afin d'obtenir les références Java correspondant aux services qu'il désire utiliser sur le site.

Un agent est une instance d'une classe Java définie par le programmeur de l'agent. Cette classe doit hériter de la classe *Agent* et elle doit définir une méthode *main()* qui est le point d'entrée de l'agent. La plate-forme implante la migration faible : à chaque arrivée sur un serveur d'agent, la méthode *main()* est appelée. La classe *Agent* fournit également deux méthodes : *move(machine)* permettant de se déplacer sur le site machine et *back()* qui permet à l'agent de revenir à son site d'origine.

Lorsqu'un agent est déplacé, un message est construit, contenant le code de l'agent ainsi que le contexte de l'agent. Le contexte de l'agent est constitué d'un ensemble dobjets Java. Le mécanisme de sérialisation de Java nous permet de transformer l'ensemble des objets gérés par l'agent (un graphe d'objets) en un flot

d'octets, de déplacer ces données vers le site destination, puis de reconstruire le contexte. Le mécanisme de chargeur de classe de Java nous permet de transformer le code de l'agent reçu dans le message en classes Java sur le site destination. Etant donné que des programmeurs différents peuvent utiliser un même nom de classe, un chargeur différent est associé à chaque agent sur un serveur d'agent. Ce chargeur est créé à l'arrivée de l'agent sur le serveur. Notons que le chargeur d'un agent ne récupère pas les classes de l'agent à la demande ; le code de l'agent est déplacé avec l'agent en une seule fois et le chargeur est initialisé avec ce code à l'arrivée de l'agent.

Pour résumer l'implantation de notre plate-forme, le déplacement d'un agent se compose des étapes suivantes :

a) sérialisation du contexte de l'agent et production d'un message incluant le contexte sérialisé de l'agent et son code,

b) envoi du contexte et du code de l'agent au serveur destination (nous utilisons une connexion TCP),

c) destruction de l'agent sur le site origine,

d) réception du message sur le site destination,

e) création d'un nouveau processus léger pour l'exécution de l'agent,

f) création d'un chargeur pour cet agent, le chargeur étant initialisé avec le code de l'agent,

g) dé-sérialisation du contexte de l'agent,

h) démarrage de l'agent en appelant le point d'entrée *main()*.

2.4. *Evaluation de la plate-forme*

Dans cette sous-section, nous présentons une évaluation de cette plate-forme. Dans la suite, nous désignons cette plate-forme sous le nom Agent. Après une description de l'environnement utilisé pour réaliser les mesures, nous donnons les coûts des mécanismes élémentaires, puis le coût global de la migration d'agent.

2.4.1. *Environnement de mesure*

Etant donné que les agents visent principalement l'exécution d'applications réparties sur des réseaux à grande échelle, nous avons effectué nos expérimentations dans des conditions réelles entre des machines éloignées interconnectées par l'Internet[3], mais à des fins de comparaison, nous avons opéré les mêmes mesures sur notre réseau local. Voici les deux configurations que nous avons utilisées :

[3] Les mesures ont été effectuées de nuit et en Europe pour limiter les variations de performance du réseau.

– Réseau local. Il est composé de deux stations de travail Sun Ultra 1 sous Solaris reliées par un réseau Ethernet à 10 Mb.

– Réseau grande distance. Il est composé de deux stations de travail comparables reliées par l'Internet. La première machine se situe en France à l'INRIA Rhône-Alpes (Grenoble) alors que la deuxième machine est située en Grande-Bretagne au Queen Mary and Westfield College (Londres).

Nous avons réalisé ces expérimentations en utilisant le JDK 1.1.5.

Afin de fournir des éléments plus précis concernant cet environnement, nous avons mesuré les performances de base du réseau et de la JVM sur ces stations de travail (table 1). Les mesures de latence (aller simple) ont été effectuées en utilisant UDP et les mesures de débit en utilisant TCP.

Latence sur le réseau local	0.6 ms
Latence sur Internet	20 ms
Débit sur le réseau local	842 Ko/s
Débit sur Internet	129 Ko/s
Appel local de méthode Java	1 µs
Appel à distance de méthode Java sur le réseau local	3,1 ms
Appel à distance de méthode Java sur Internet	44 ms

Table 1. *Coûts de base de l'environnement expérimental*

Cette table montre les différences importantes de performances entre le réseau Internet et le réseau local. Elle montre également comment se comportent les mécanismes de base de Java (appel local, appel à distance) sur l'environnement que nous avons utilisé. Notons que l'appel à distance Java (RMI) coûte beaucoup plus cher qu'un appel local, ce qui donne une idée de l'intérêt d'utiliser les agents mobiles pour privilégier des interactions locales.

2.4.2. *Composants élémentaires de la plate-forme*

Afin de mesurer les coûts élémentaires, nous avons décomposé la fonction de migration d'un agent en plusieurs parties qui correspondent aux principales étapes de cette opération. Pour ces mesures, nous avons programmé un agent minimal qui réalise une migration entre les deux machines. La taille de l'agent est 1475 octets, incluant le code et le contexte de l'agent. La migration se compose des phases suivantes :

– Sérialisation de l'agent. Cette phase correspond à l'étape a) de la description de la plate-forme. Un message est construit, contenant le code et le contexte de l'agent.

– Transfert de l'agent. Cette phase correspond aux étapes b),d) et e) de la description de la plate-forme. Le message est envoyé au serveur d'agent destination. Sur la machine destination, un processus léger est créé et le message est transmis à ce processus. L'étape c) n'est pas incluse dans cette phase, car l'agent sur le site origine n'est détruit qu'après avoir envoyé le message.

– Installation de l'agent. Cette phase correspond aux étapes f), g) et h) de la description de la plate-forme. Un nouveau chargeur est créé pour l'agent arrivant. Le chargeur est initialisé avec le code de l'agent, puis le contexte de l'agent est dé-sérialisé.

Les temps de ces différentes phases sont donnés dans la table 2.

Sérialisation de l'agent	3,2 ms
Transfert de l'agent sur le réseau local	8 ms
Transfert de l'agent sur Internet	121 ms
Installation de l'agent avec le chargeur de Java	4.3 ms
Installation de l'agent avec un chargeur privé à l'agent	23,8 ms

Table 2. *Coûts élémentaires*

Ces mesures montrent que les coûts induits par Java sont prohibitifs par rapport au coût de transfert dans le cas d'un réseau local, mais deviennent acceptables par rapport au coût du transfert sur Internet.

L'installation de l'agent inclut la dé-sérialisation du contexte de l'agent en utilisant un chargeur privé. La gestion d'un chargeur privé induit un surcoût que nous avons évalué en mesurant le coût d'installation d'un agent dans deux conditions :

– avec le chargeur système de Java, ce qui suppose que les classes sont installées et accessibles sur les deux machines. Les classes sont chargées lors de la première visite et mises en cache pour les visites suivantes.

– avec un chargeur privé pour chaque visite de l'agent lors de l'itération. Les classes sont donc chargées à partir du code transféré avec l'agent lors de chaque visite.

On observe que la plus grande partie du coût de l'installation provient du chargement des classes par un chargeur privé.

2.4.3. *Coût global de la migration*

Pour conclure cette section, nous avons mesuré le coût global de la migration dans les deux environnements précédents (réseau local et Internet), et avec utilisation d'un chargeur privé (table 3).

Coût de la migration d'un agent minimal sur réseau local	35 ms
Coût de la migration d'un agent minimal sur Internet	148 ms

Table 3. *Coût de la migration d'un agent*

Notons que la migration d'un agent coûte beaucoup plus qu'un appel à distance. Il faut donc que le déplacement d'un agent permette d'économiser un nombre suffisant d'appels à distance pour devenir rentable, ce que nous montrons en section 3 avec deux exemples d'application.

Notre plate-forme n'implante que les mécanismes de base. Afin de donner une idée du coût de la migration d'agent dans une plate-forme à agent qui implante toutes les fonctions nécessaires au développement d'application à base d'agents, nous avons effectué les mêmes mesures globales avec la plate-forme Aglets [LAN 98a], qui est une des plates-formes à agents les plus connues dans le domaine (table 4).

Coût de la migration d'un Aglet minimal sur réseau local	240 ms
Coût de la migration d'un Aglet minimal sur Internet	369 ms

Table 4. *Coût de la migration d'un Aglet*

La différence avec notre plate-forme est importante. Elle provient de la complexité du protocole ATP (Agent Transfer Protocol) [LAN 97b] utilisé par Aglets qui nécessite plusieurs envois de messages pour déplacer un agent. De plus, ATP est un protocole générique qui permet la mobilité des agents entre des plate formes implémentées par des vendeurs différents avec des langages de programmation différents. Cette gestion d'hétérogénéité contribue également au coût élevé d'ATP.

Après avoir décrit les principes de conception d'une plate-forme à agent et avoir donné les coûts des principaux éléments rentrant dans sa conception sur Java, nous étudions dans la section suivante l'intérêt de la programmation par agents mobiles par comparaison à l'approche client/serveur.

3. Une alternative au modèle client/serveur

Dans cette section, nous montrons dans quelles conditions le modèle à agents mobiles est plus efficace que le modèle client/serveur. Après avoir décrit le domaine d'application considéré, nous présentons respectivement pour chaque application son principe et les résultats obtenus.

3.1. *Domaine d'application*

Dans les deux applications étudiées, les agents mobiles sont utilisés pour adapter aux besoins des clients un service fourni par un serveur. Nous supposons que l'objectif du serveur est de fournir un service générique pouvant répondre aux besoins variés de tous ses clients potentiels.

Afin de fournir un service générique, le serveur peut exporter une interface paramétrable, mais la complexité de cette interface augmente avec la diversité des utilisations possibles du service. De plus, il est difficile d'identifier les besoins potentiels de tous les clients. Une autre approche consiste à décomposer le service en un ensemble d'opérations élémentaires, plus simples, pouvant être appelées indépendamment les unes des autres. Un client peut alors combiner les appels à ces opérations élémentaires afin d'obtenir le service désiré.

La deuxième approche est préférable, car elle permet d'offrir un service générique sans forcément connaître à l'avance les besoins des clients. Par contre, elle nécessite des interactions plus fréquentes entre le client et le serveur. Dans le cadre d'une architecture répartie fondée sur le modèle client/serveur, les interactions à distance coûtent cher et la généricité devient alors très pénalisante. Une solution consiste alors à co-localiser une partie de l'application cliente et le service générique fourni par le serveur, afin de limiter les interactions à distance.

C'est ce principe qui a motivé les travaux autour des extensions de systèmes d'exploitation (Spin [BER 95], Exokernel [ENG 95]). Un service système générique nécessite de nombreux appels systèmes qui sont coûteux. Les systèmes extensibles exécutent une partie de l'application dans le même espace d'adressage que le système d'exploitation, afin de réduire le coût des appels au système par cette partie de l'application. Nous nous proposons d'appliquer cette technique en utilisant des agents mobiles dans le cadre d'applications réparties utilisant des serveurs d'information.

En utilisant des agents mobiles, un client peut développer un service spécialisé en fonction de ses besoins en s'appuyant sur le service générique fourni par le serveur. Le service spécialisé du client peut être envoyé sur le site du serveur sous la forme d'un agent mobile, ce qui implique que les interactions entre le service spécialisé et le service générique seront des appels locaux sur la machine serveur. La figure 1 illustre ce schéma.

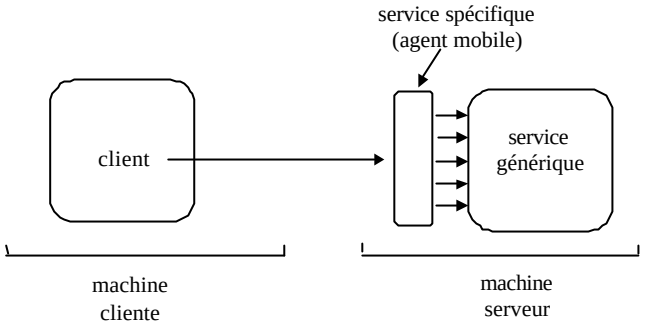

Figure 1. *Extension d'un serveur par un agent mobile*

Notre schéma exprime un rapprochement entre un client et un serveur. Cette idée de rapprocher le traitement des données qui se trouvent dans un site de stockage pour gagner en performance n'est pas nouvelle. Elle a été exploitée dans le domaine des bases de données, dans le cas d'une requête SQL dont l'exécution nécessite le parcours d'une base de données de taille importante. Dans ce cas, le code du client est déplacé vers le serveur de la base de données, car déplacer du code entre des machines est plus rentable que déplacer des données entre ces machines.

SQL permet de programmer un code mobile qui peut être envoyé à un serveur de base de données. Cependant, ce code mobile est spécialisé pour les requêtes de base de données et ne permet pas de programmer un algorithme général. Les agents mobiles développés dans un langage de programmation général comme Java permettent de définir des agents répondant à des besoins très variés. Par exemple, un agent peut être envoyé sur un serveur afin de compresser des données devant être retournées au client, ou encore afin de dégrader la qualité de présentation d'une séquence vidéo devant être envoyée au site client. De tels exemples d'adaptation du service fourni par un serveur sortent largement du cadre des opérations sur les bases de données.

Un autre aspect important de ce schéma est que l'envoi du service spécifique sur la machine serveur peut se faire suivant deux modes :

– installation préalable. Le service spécifique est installé sous la forme d'un agent par le client sur le site serveur. Le client peut ensuite réaliser des appels à distance au serveur en utilisant une interface spécifique exportée par l'agent qu'il a installé sur le serveur. L'installation de l'agent n'a lieu qu'une fois et toutes les utilisations du service sont ensuite réalisées par appel à distance.

– migration systématique. L'application du client est un agent mobile qui se déplace vers le serveur afin de réaliser des appels locaux sur le site serveur. A chaque appel au service, l'agent client est déplacé vers le site du serveur.

Avec une installation préalable, on ne paie le coût du déplacement d'un agent que lors de l'installation et les appels au service sont sans surcoût et bénéficient de l'adaptation du service. Avec la migration systématique, on paie le coût du déplacement d'un agent à chaque appel. Il faut noter que l'installation préalable d'une spécialisation peut être proscrite par le serveur si celui-ci ne désire pas voir s'accumuler des extensions du service sur son site.

Dans la suite de cet article, nous supposons une migration systématique d'agent, car notre objectif est d'évaluer l'intérêt d'une spécialisation d'un service à l'aide d'un agent mobile, en fonction du coût de déplacement d'un agent. Dans le cas d'une installation préalable et une fois cette installation réalisée, le gain est évident.

3.2. *Redirection de requête*

3.2.1. *L'application*

Une des applications les plus importantes dans le domaine des agents mobiles est la recherche d'information. Dans ces applications, des agents se déplacent sur différents sites pour chercher des informations pour leurs clients.

Notre exemple est une application répartie sur Internet dont le but est de chercher une liste d'hôtel dans une ville. Dans notre scénario, deux bases de données doivent être consultées. La première recense des hôtels et permet d'obtenir la liste des hôtels de la ville où le client désire se rendre. La deuxième base de données est un annuaire permettant d'obtenir les numéros de téléphone de ces hôtels [4]. Ces deux bases de données sont gérées sur des sites différents par des administrations ou des entreprises différentes (par exemple un office de tourisme et une compagnie de téléphone pour notre scénario).

Ces deux serveurs fournissent chacun une interface correspondant au service qu'ils offrent. Pour le premier serveur, nous supposons que l'interface permet au client de donner le nom de la ville, le serveur lui retournant une table des hôtels dans cette ville. Pour le deuxième serveur, nous supposons que l'interface permet au client de donner un nom, le serveur retournant le numéro de téléphone associé à ce nom.

En utilisant le modèle client/serveur classique, le client va devoir faire un appel à distance pour interroger le premier serveur et récupérer la table des hôtels qui l'intéressent. A partir de cette table, le client va ensuite, pour chaque hôtel dans la

[4] ou toute autre information complémentaire : prix en passant par une agence de voyage, cotation dans un guide touristique, ...

table, réaliser un appel à distance au second serveur afin de récupérer le numéro de téléphone de l'hôtel.

Ainsi, dans le mode client/serveur, le client doit appeler le second serveur (B) autant de fois qu'il y a d'éléments dans le tableau d'hôtels retourné par le premier serveur (A). Le temps d'obtention de la réponse finale est égal au coût de communication avec le serveur A (rpc(A)), plus le coût des communications avec le serveur B qui est égal au coût de l'appel au serveur B multiplié par la taille du tableau obtenu comme réponse du serveur A (n*rpc(B)). D'où un coût total[5] :

*coût total = rpc(A) + n*rpc(B)*

Les deux serveurs A et B fournissent chacun une interface générique composée d'opérations élémentaires. Les clients peuvent ainsi utiliser le service selon leurs besoins. Cependant, cela se traduit par de nombreuses interactions entre les clients et le serveur B.

Une seconde solution qui devient plus efficace lorsque le nombre d'hôtels est grand consiste à transmettre directement la table des hôtels du serveur A au serveur B afin que celui-ci réalise une jointure avec sa table des numéros de téléphone. Pour ce faire, il faut spécialiser les serveurs pour qu'ils offrent ce service de redirection de requêtes. Une extension statique des serveurs (par les administrateurs des deux serveurs) n'est pas une solution réaliste dans la mesure où ces bases de données sont administrées séparément. De plus, il n'est pas réaliste d'étendre un serveur pour chaque besoin spécifique d'un client.

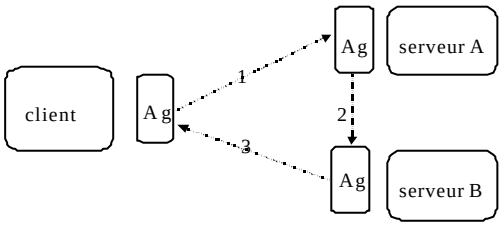

Figure 2. *Déplacement de l'agent pour la redirection de requêtes*

Nous proposons donc d'utiliser une spécialisation dynamique des serveurs par des agents mobiles (figure 2).

Le client crée un agent Ag contenant la requête globale à réaliser. Cet agent se déplace (1) tout d'abord sur le site du serveur A et réalise localement l'appel au serveur A pour obtenir la table des hôtels. La table des hôtels est stockée dans le

[5] Nous ne prenons pas en compte le coût de traitement d'une requête sur chaque serveur qui reste le même dans tous les cas de figure.

contexte de l'agent, puis l'agent se déplace (2) vers le serveur B. Sur le serveur B, l'agent peut itérer sur la table des hôtels et demander au serveur B le numéro de téléphone de chaque hôtel (cette étape correspond à la jointure). Ces numéros de téléphones sont stockés dans le contexte de l'agent qui se déplace (3) finalement vers son site d'origine où il délivre le résultat au client[6].

Comme on le voit à travers cet exemple, l'utilisation d'agents mobiles permet d'étendre dynamiquement l'interface des serveurs. Les agents ont été utilisés ici pour permettre une redirection de requête entre des bases de données différentes.

Dans cette mise en œuvre avec des agents mobiles, un agent avec peu de données dans son contexte doit être envoyé sur le serveur A (coût A), puis l'agent se déplace avec la table des hôtels vers le serveur B (coût A(Thotel)), puis cet agent retourne à son site d'origine avec la table enrichie avec les numéros de téléphone (coût A(Ttel)). D'où un coût total de :

coût total = A + A(Thotel) + A(Ttel)

En conséquence, la solution à base d'agents mobiles sera plus efficace si :

*A + A(Thotel) + A(Ttel) < rpc(A) + n*rpc(B)*

En supposant un coût constant pour les fonctions A() et rpc() et en reportant les coûts mesurés pour un agent minimal en section 2.4, nous obtenons n>8 pour le réseau Internet. Il s'agit d'une approximation très grossière. Avec les mesures effectuées sur l'application, nous montrons dans la suite à partir de quelle valeur de n la solution à base d'agents est rentable.

3.2.2. *Les mesures*

Pour évaluer cette application, nous avons dû utiliser une machine supplémentaire par rapport à l'environnement décrit en section 2.4.1.

La troisième machine est située en Suisse à l'Université de Genève. Nous résumons les capacités de cet environnement sur Internet par les mesures suivantes (table 5).

Connexion	Latence (ms)	Débit (ko/sec)
France - Grande Bretagne	20	129
Grande Bretagne - Suisse	16	280
Suisse - France	26	123

Table 5. *Capacités de l'environnement utilisé*

[6] L'agent, lorsqu'il est sur le site B, pourrait envoyer directement le résultat au site d'origine. Nous n'avons pas cherché à optimiser ce schéma d'exécution.

Comme nous l'avons montré dans la section précédente, le temps de réponse de la requête varie avec le nombre d'enregistrements retournés. Nous avons effectué des mesures en faisant varier le nombre d'enregistrements retournés par le premier serveur (retournant la table des hôtels). Dans notre application, la taille d'un enregistrement est de 80 octets. Nous avons effectué ces mesures avec la plate-forme que nous avons développée, mais aussi avec le système Aglets. Les résultats sont donnés sur la figure 3.

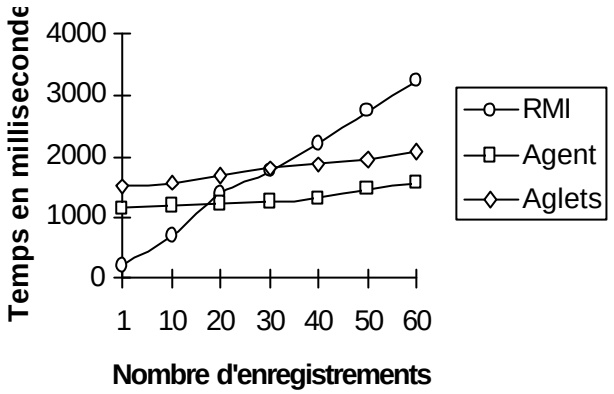

Figure 3. *Comparaison entre RMI et les agents mobiles pour l'application « redirection de requêtes »*

Nous observons que pour un nombre d'enregistrements retournés petit (moins de 20 enregistrements), l'implantation basée sur RMI est plus efficace que celles basée sur des agents mobiles. Ceci s'explique par le fait que pour un nombre d'enregistrements petit, le nombre d'appel à distance économisé n'est pas suffisant pour amortir le coût de la migration de l'agent sur le réseau. Cependant, pour un nombre d'enregistrements suffisant, la solution à base d'agent est bien plus efficace.

La différence entre Aglets et Agent s'explique par le fait que Agent est un prototype minimal n'implantant que les fonctions strictement nécessaires.

3.3. *Compression de données*

3.3.1. *L'application*

Le Web est très souvent utilisé pour récupérer des documents de toute sorte. Nous considérons dans cette section l'accès à un serveur exportant une interface

permettant à un client d'obtenir une copie d'un document. Le client demande au serveur un document de nom donné et récupère un flot de données correspondant au contenu du document.

Cependant, si le client désire optimiser ce transfert de données en utilisant un outil de compression des données, il est nécessaire d'étendre le serveur pour y ajouter cette fonction de compression. De plus, différents clients peuvent utiliser différents outils de compression, nécessitant des extensions particulières pour chaque client.

Le paradigme d'agents mobiles offre une réponse élégante à ce problème. En effet, le client peut envoyer un agent sur le site du serveur. Cet agent interroge le serveur pour obtenir le document, compresse le document obtenu et revient sur le site du client avec le document compressé. Sur le site client, l'agent peut décompresser le document et le délivrer au client. De cette manière, le client réalise une extension dynamique du serveur en fonction de ses besoins spécifiques sans changer l'interface de ce dernier.

La principale motivation du client pour cette extension du serveur est l'efficacité du transfert du document. Pour étudier le coût de ce transfert, considérons deux machines dont le débit de transfert entre elles est dr. Nous supposons que le facteur de compression du document est fc (si T est la taille initiale du document, le document compressé a une taille de fc*T) et que les temps respectifs de compression et décompression sont tc et td. Nous cherchons à identifier les conditions à partir desquelles la compression du document est rentable.

Le temps de transfert du document sans compression est T/dr si T est la taille du document à transférer.

Le temps de transfert du document avec compression est :

$tc + fc*T/dr + td$

respectivement le temps de compression du document, le temps de transfert du document compressé et le temps de décompression du document.

En conséquence, la compression est potentiellement rentable si :

$tc + fc*T/dr + td < T/dr$

Ceci implique que le client gagne dans les cas où le débit du réseau entre sa machine et la machine du serveur est tel que :

$dr < T(1 - fc) / (tc + td)$

Cette formule indique à partir de quand la compression peut être rentable. Nous ne prenons pas en compte ici le coût du transfert de l'agent. L'évaluation qui suit montre que ce coût est largement amorti pour des documents de grande taille.

3.3.2. *Les mesures*

Pour réaliser notre évaluation, nous avons utilisé des documents postscript dont la taille varie de 100 Ko à 2000 Ko. Nous avons utilisé l'outil de compression gzip qui est présent dans l'environnement des classes de Java. Pour chaque document, nous avons mesuré le facteur de compression et le temps de compression/décompression. En reportant ces valeurs dans la formule de la sous-section précédente, nous obtenons que la compression devient rentable lorsque :

– $dr < 74$ *Ko/s* pour le document de taille 100 Ko de notre jeu d'essai,

– $dr < 164$ *Ko/s* pour le document de taille 500 Ko de notre jeu d'essai.

En considérant les débits mesurés précédemment, on voit qu'en fonction du réseau utilisé (réseau local ou Internet) et qu'en fonction de la taille du document considéré, il est possible de gagner ou de perdre en utilisant des agents mobiles. Cependant, notre calcul de coût théorique ne prend en compte le coût de déplacement de l'agent. De plus, le coût de migration d'un agent est bien supérieur au coût de l'appel à distance par RMI. Il est donc difficile de prédire si le coût de migration de l'agent peut être amorti par la compression du document lorsque l'on augmente la taille du document.

Nous avons mesuré le coût de récupération d'un document à partir d'un serveur en utilisant RMI (sans compression), et en utilisant un agent mobile qui implante la compression du document. Ces mesures ont été effectuées entre les deux machines situées respectivement en France et en Grande-Bretagne. Nous avons reproduit cette expérimentation avec Agent et Aglets et avec des tailles de document différentes. Les résultats sont donnés sur la figure 4.

On observe que pour une petite taille de document (100 Ko), RMI est plus efficace que les deux systèmes à agents mobiles. Cependant, pour un document de 500 Ko, Agent est plus efficace que RMI (le bénéfice tiré de la compression du document amortit le surcoût provenant de la migration de l'agent), alors qu'Aglets reste moins bon que RMI. Finalement, pour une taille de document plus grande, les deux plates-formes à agents sont meilleures que RMI, ce qui signifie que l'augmentation de la taille du document permet d'amortir le coût provenant de l'utilisation d'agents mobiles.

Les deux scénarios d'application que nous avons présentés et évalués montrent que la spécialisation de serveur par un agent mobile permet des gains d'efficacité significatifs par rapport à un schéma client/serveur, principalement lorsque le déplacement de l'agent débouche sur une économie en termes de communication.

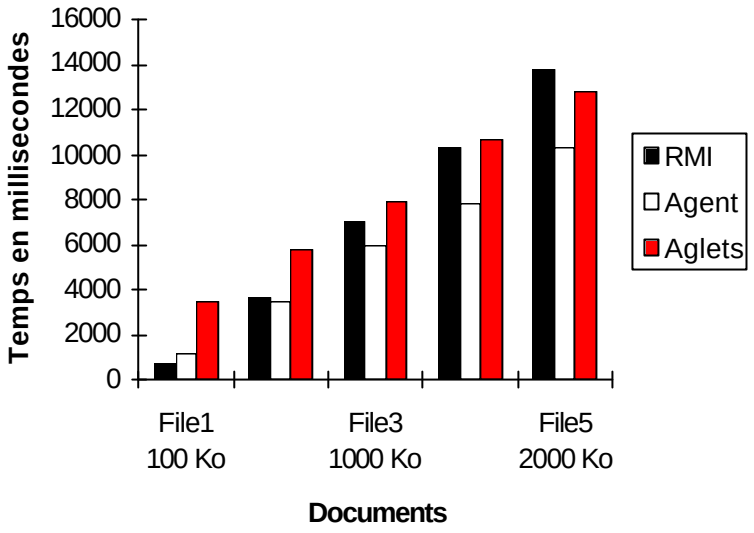

Figure 4. *Comparaison entre RMI et les agents mobiles pour l'application « compression de données »*

4. Conclusion et perspectives

Alors que de nombreux chercheurs s'intéressent au développement et à l'utilisation de systèmes à agents mobiles pour permettre la construction d'applications réparties sur des réseaux à grande distance, peu d'entre eux ont proposé une évaluation des systèmes à agents mobiles basée sur des mesures dans des conditions réelles. Dans cet article, nous avons décrit nos expérimentations qui visent à réaliser une telle évaluation.

Afin d'être capable de mesurer plus précisément les mécanismes impliqués dans la réalisation d'une plate-forme à agent mobiles sur Java, nous avons développé un prototype minimal permettant de réaliser des applications pilotes. Nous avons mesuré le coût des principaux mécanismes rentrant dans sa conception et montré que même si les mécanismes de Java restent chers, leur coût est acceptable lorsque l'on s'intéresse à des applications réparties sur l'Internet.

Afin d'évaluer l'intérêt de la programmation par agents mobiles par rapport au modèle client/serveur, qui est actuellement le modèle incontournable pour le développement d'applications réparties, nous avons sélectionné deux applications qui sont caractéristiques des applications visées par les agents mobiles. Nous avons programmé ces applications de trois façons, en utilisant RMI (qui implante sur Java le modèle client/serveur), en utilisant notre prototype à agents mobiles construit sur Java et finalement en utilisant le système Aglets. Les résultats de l'évaluation montrent que les agents mobiles peuvent amener des gains d'efficacité significatifs

lorsqu'ils permettent de transformer les communications à distance en communications locales ou de réduire le volume d'information transféré sur le réseau. Ces gains sont d'autant plus importants que le réseau est lent, car cela permet alors d'amortir plus rapidement le coût du déplacement de l'agent. Par exemple, sur l'application «redirection de requête », un agent mobile permet de remplacer des appels à distance par des appels de méthode locaux lorsque l'agent se déplace vers le deuxième serveur, et d'économiser ainsi des communications. Sur l'application « compression de données », un agent mobile permet de compresser les données à récupérer et de réduire ainsi le volume d'information à transférer.

Nous pensons que la technologie à agents mobiles jouera un rôle important dans la conception des applications réparties sur l'Internet. La conscience de ce rôle a poussé l'OMG (Object Management Group) à définir les spécifications de MASIF (Mobile Agent System Interoperability Facilty) [MAS 97] pour l'interopérabilité entre les différents systèmes à agents mobiles. Un autre effort est lancé par FIPA (Foundation for Intelligent Physical Agents) [FIP 98] afin de spécifier l'architecture mais aussi les sémantiques de communications entre des agents mobiles ; l'interopérabilité reste un but important. Ces efforts ne sont pas encore à terme. De plus, pour un déploiement de cette technologie, des problèmes de sécurité doivent être résolus.

Le travail présenté dans cet article se poursuit à l'heure actuelle dans deux perspectives principales.

La première consiste à utiliser le modèle à agents mobiles pour la construction d'applications de plus grande taille. Nous utilisons actuellement une plate-forme à agents mobiles pour la gestion de la qualité de service dans une application multimédia répartie. Cette application consiste en un client (browser) permettant de visualiser des documents multimédia. Le document multimédia peut contenir des références à des sources vidéo ou audio situées sur des serveurs répartis (comme une page HTML). De même que pour l'application « compression de donnée » décrite dans cet article, un agent peut être envoyé sur un serveur pour adapter la qualité de service d'une source de donnée multimédia située sur ce serveur.

La deuxième perspective consiste à permettre à une application de choisir à l'exécution la stratégie à utiliser. Nous avons montré que la spécialisation de serveur pouvait être rentable sous certaines conditions, en particulier avec des débits faibles et des interactions fréquentes entre le client et le serveur. Nous nous proposons d'étudier comment un client peut prendre la décision d'utiliser un agent mobile ou d'utiliser simplement un appel à distance, en fonction de l'état du réseau à l'exécution et de la fréquence d’appel au serveur.

Remerciements

Le travail décrit dans cet article a bénéficié du soutien de la communauté européenne dans le cadre du projet Esprit C3DS. Nous tenons à remercier tous les membres du projet Sirac avec qui nous travaillons quotidiennement.

5. Bibliographie

[BAU 98] BAUMANN J., HOHL F., ROTHERMEL K., STRAßER M.. Mole, « Concepts of a Mobile Agent System », *World Wide Web*, vol. 1, n°. 3, p. 123-137, 1998.

[BER 97] BERNARD G, « Technologie du code mobile : état de l'art et perspectives », *Actes du Colloque Francophone sur l'Ingénierie des Protocoles (CFIP'99)*, Nancy, France, 26-29 avril 1999.

[BER 95] BERSHAD B. N., SAVAGE S., PARDYAK P., SIRER E. G., FIUCZYNSKI M. E., BECKER D., CHAMBERS C., EGGERS S., « Extensibility, Safety and Performance in the SPIN Operating System »,*15th Symposium on Operating Systems Principles (SOSP)*, Copper Mountain Resort, Colorado, décembre 1995.

[CAR97] CARZANIGA A., G. P. PICCO G., G. Vigna G., « Designing Distributed Applications with Mobile Code Paradigms », *In Proceedings of the 19th International Conference on Software Engineering (ICSE'97)*, Boston (MA, USA), R. Taylored, ISBN 0-89791-914-9, p.22-32, mai 1997.

[CHE 94] CHESS D., HARRISON ET C., KERSHENBAUM A., « Mobile Agents: are they a good idea? », IBM Research Report, RC 19887, Décembre 1994.

[ENG 95] ENGLER D. R., KAASHOEK, M. F., O'TOOLE J., « Exokernel: An Operating System Architecture for Application-Level Resource Management », *15th Symposium on Operating Systems Principles (SOSP)*, Copper Mountain Resort, Colorado, décembre 1995.

[FIP 98] FIPA 98 «Specification. Foundation for Intelligent Physical Agents. Geneva, Switzerland », 23 octobre 1998.

[FUG 98] FUGGETTA A., PICCO G. P., VIGNA G., « Understanding Code Mobility », *IEEE Transactions on software engineering*, vol. 24, 1998.

[FUK 97] FUKUDA M., BIC L. F., M. B. DILLENCOURT M. B., «Performance of the MESSENGER Autonomous-Objects-Based System », *First International Conference. on Worldwide Computing and its Applications '97 (WWCA97)*, Tsukuba, Japan, 10-11 mars 1997 (LNCS 1274, Springer-Verlag)

[GOS 95] J. GOSLING AND H. MCGILTON, « The Java Language Environment »: a White Paper, Sun Microsystems Inc., 1995. {http://java.sun.com/whitePaper/java-whitepaper-1.html}

[GRA 96] GRAY R., G. CYBENKO, D. KOTZ, AND D. RUS, « Agent TCL », Department of Computer Science, Dartmouth College, Hanover, NH, mai 1996. {http://www.cs.dartmouth.edu/?agent/papers/chapter.ps.Z }

[JOH 97] JOHANSEN D., SUDMANN N. P., VAN RENESSE R., « Performance Issues in TACOMA ». *In, 3rd. Workshop on Mobile Object Systems*, 11th Europeean Conference on Object-Oriented Programming, Jyvaskyla, Finland, 9-13 juin 1997.

[LAN 97a] LANGE D., « Java Aglet Application Programming Interface (J-AAPI) », White Paper - Draft 2. IBM Tokyo Research Laboratory. 19 février 1997. http://www.trl.ibm.co.jp/aglets/JAAPI-whitepaper.html

[LAN 97b] LANGE D. et ARIDOR Y., « Agent Transfer Protocol » – ATP/0.1. IBM Tokyo Research Laboratory. Draft number : 4, 19 mars 1997.

[LAN 98a] LANGE D., OSHIMA M, *Programming and Deploying Java Mobile Agents with Aglets*, Addison-Wesley Pub Co.; ISBN: 0-201-32582-9, août 1998.

[LAN 98b] LANGE D., OSHIMA M., « Mobile Agents with Java: The Aglet API », *World Wide Web*, vol. 1, n° 3, septembre 1998. Aussi apparu dans Mobility. Processes, Computers, and Agents. Dejan Milojicic, Frederick Douglis et Richard Wheeler (Eds). ISBN 0-201-37928-7, 1999.

[MAS 97]. Joint Submission. « Mobile Agent System Interoperability Facilities Specification ». GMD Fokus (IBM). OMG TC Document orbos/97-10-05. 10 novembre 1997

[MIL 98] MILOFICIC D., LAFORGE W., CHAUHAN, D., « Mobile Objects and Agents (MOA) », *4th USENIX Conference on Object-Oriented Technologies and Systems*, avril 1998.

[OMG 91] OMG. « The Common Object Request Broker: Architecture and Specification », OMG Document Number 91.12.1, Revision 1.1, Décembre 1991.

[RIG 96] R. RIGGS, J. WALDO, A. WOLLRATH, K. BHARAT, « Pickling State in the Java System », Computing Systems, vol. 9, n° 4, 1996.

[STA 90] STAMOS J. W., GIFFORD D. K., «Remote Evaluation », *ACM Transactions on Programming Languages and Systems*, vol. 12, n° 4, p. 537-565, octobre 1990.

[STR 97] STRAßER, M. and MARKUS Schwehm M., « A Performance Model for Mobile Agent Systems », *Distributed Processing Techniques and Applications (DPPTA'97)*. Editor H.R. Arabnia, Las Vegas 1997, vol. 2, p. 1132-1140, 1997.

[THO 97] THORN T., « Programming languages for mobile code », rapport technique IRISA n° 3134 ISSN 0249-6399, mars 1997.

[VEN97] VENNERS B., « Under the Hood: The Architecture of aglets », mars 1997. {http://www.trl.ibm.co.jp/aglets/}

[WHI 94] WHITE J. E., « Telescript Technology: The Foundation for the Electronic Marketplace », General Magic Inc., Mountain View, CA, 1994.

[WOL 96] WOLLRATH A., R. RIGGS, J. WALDO, « A Distributed Object Model for the Java System », *Computing Systems*, vol. 9, n° 4, p. 291-312, 1996.

Article reçu le 8 mars 1999
Version révisée le 9 décembre 1999
Rédacteur responsable : Jean-Louis GIAVITTO

Leila Ismail *prépare sa thèse dans le cadre du projet Sirac (Systèmes informatiques répartis pour applications coopératives) à l'unité de recherches Rhône-Alpes de l'INRIA. Ses recherches concernent le domaine à agents mobiles : études d'efficacité et de protection. Elle est actuellement inscrite en thèse à l'Institut National Polytechnique de Grenoble (INPG). Elle a obtenu en 1996 son Diplôme d'Etudes Approfondies (DEA) en Informatique, Systèmes et Communications, de l'Université Joseph Fourrier à Grenoble, France.*

Daniel Hagimont *a soutenu en 1993 une thèse de doctorat à l'Institut National Polytechnique de Grenoble (INPG). Après un séjour d'une année à l'Université de Colombie Britannique (Vancouver), il occupe actuellement les fonctions de chargé de recherches à l'unité Rhône-Alpes de l'INRIA. Il a également soutenu en 1998 une Habilitation à Diriger des Recherches à l'INPG.*