Rapport IRIT/2003-25-R

JavAne : partage et recherche d'information à base d'agents mobiles adaptables

J.-P. Arcangeli*,
S. Leriche*,
M. Pantel**

Institut de Recherche en Informatique de Toulouse

{arcangel, leriche}@irit.fr, pantel@enseeiht.fr

* IRIT-UPS, 118 route de Narbonne 31062 Toulouse Cedex 4
** IRIT-ENSEEIHT, 2 rue Camichel BP 7122 31071 Toulouse Cedex 7

JavAne : partage et recherche d'information à base d'agents mobiles adaptables

Jean-Paul Arcangeli*, Sébastien Leriche*, Marc Pantel**
Institut de Recherche en Informatique de Toulouse
{arcangel, leriche}@irit.fr, pantel@enseeiht.fr

* IRIT-UPS, 118 route de Narbonne 31062 Toulouse Cedex 4
** IRIT-ENSEEIHT, 2 rue Camichel BP 7122 31071 Toulouse Cedex 7

Résumé

Le déploiement des réseaux à grande échelle permet de disposer d'un grand nombre de sources d'informations facilement accessibles pour des outils automatisés de recherche et d'exploitation d'informations. La programmation de ces outils est par contre encore complexe et source de nombreuses erreurs. L'objectif de cet article est de présenter l'apport du modèle des agents mobiles adaptables pour faciliter cette réalisation. Celui-ci s'appuie sur un exemple concret d'outil de recherche et de rapatriement de documents développé avec la bibliothèque JavAct. Nous montrons comment cet outil peut s'adapter aisément à différentes formes d'évolutions : conditions d'exécution, découverte de nouveaux serveurs possédant des mécanismes de communication spécifiques, adaptation des algorithmes de recherche (locaux et distribués) en fonction des informations obtenues sur chaque serveur. Nous mettons ensuite en évidence la facilité d'adaptation à d'autres problèmes tels l'exécution de services à distance ou la recherche et la mise à jour de composants logiciels. Nous comparons ensuite notre proposition avec d'autres travaux de la communauté des agents mobiles pour la recherche d'informations. Enfin, nous concluons sur l'état actuel de notre prototype et ses perspectives d'évolutions.

Mots-clés : intergiciel, agents mobiles, flexibilité, répartition à grande échelle, systèmes pair à pair, architecture logicielle, aide au développment.

Abstract

The deployment of wide area networks provides an easy access to many different data sources which can be used by automated data retrieval and processing tools. However, the development of such tools is still very complex and error-prone. The purpose of this paper is to present the improvements provided by the adaptable mobile agents programming model. Our arguments rely on the description of a real retrieval and download tool written with the JavAct toolkit. We show how this tool can easily be adapted to various specification changes : execution context, access to new servers with initially unknown communication protocols, dynamic (local and distributed) algorithm modification based on the results provided by the servers. We also show how this tool can be easily extended to other problems such as remote service execution or retrieval and update of software components. Our proposal is then compared to other work in the adaptable mobile agent community. Then we conclude by giving insights on the current state of our prototype and various evolution proposals.

Keywords : middleware, mobile agents, flexibility, wide-area distribution, peer-to-peer systems, software architecture, software developpement assistance.

1 Introduction

La programmation par agents mobiles est un paradigme de programmation des systèmes répartis [FPV98, BI02]. Dès les premières publications, la recherche d'information a été présentée comme une application potentielle importante des agents mobiles voire comme la «*killer application*». Dans [CHK94], les auteurs précisent le champ d'application : sources d'information multiples réparties, volumes importants, prise en compte des spécificités du client, interactions avec le client et la source. A priori, les avantages des agents mobiles sont nombreux :

- La mobilité d'agent permet à un client d'interagir localement avec un serveur, et donc de réduire le trafic sur le réseau en ne transmettant que les données pré-traitées réellement nécessaires.
- L'exécution d'agents spécialisés offre davantage de souplesse que l'exécution d'une procédure standard sur les sites serveurs, et permet des transactions plus robustes que le déplacement des données sur le site client.
- L'asynchronisme et l'autonomie des agents leur permet de réaliser une tâche tout en étant déconnecté du client, ce qui est particulièrement utile dans le cas de supports physiquement mobiles.

Cependant, même si plusieurs travaux significatifs ont été menés dans le domaine de la recherche d'information (cf. section 4), il s'avère que les technologies à base d'agents mobiles n'ont encore été que peu utilisées. On peut avancer ici quelques éléments d'explication :

- A quelques exceptions près, les applications réparties que l'on peut développer avec les agents mobiles peuvent l'être avec d'autres technologies plus classiques. Les agents mobiles offrent cependant un certain nombre d'avantages cumulés, ainsi qu'un cadre fédérateur pour traiter différents problèmes et se substituer à diverses solutions ad hoc ; c'est l'argument «*génie logiciel*». Pour l'instant, cet argument a eu peu de poids face au saut technologique demandé aux développeurs.
- Les agents mobiles sont particulièrement utiles pour la mise en oeuvre d'applications dont les performances varient avec la disponibilité et la qualité des services et des ressources, ou avec le volume des données déplacées sur le réseau. Les expériences primitivement menées avec des applications relativement figées dans des environnements stables et des réseaux locaux mettaient sans doute peu en évidence l'intérêt des agents mobiles.
- Enfin, les contraintes de sécurité constituent un obstacle à l'expansion de cette technologie. Au cas par cas, un certain nombre de solutions ont été proposées, mais elles reposent généralement sur des hypothèses et des restrictions fortes. Ce problème est toujours largement ouvert.

Aujourd'hui, les réseaux à grande échelle, tel l'Internet, et les grilles de calcul ou de stockage interconnectent de grandes quantités de données, de services, de ressources matérielles et logicielles répartis. Dans ce cadre, de nouvelles formes de calcul sont apparues, tel le calcul pair à pair dans lequel des sites (à la fois clients et serveurs) utilisent et fournissent des données ou des services, éventuellement de manière temporaire. Dans un tel contexte réparti et hétérogène, qui touche également l'informatique ubiquitaire et l'informatique mobile, il n'est pas possible de faire d'hypothèse sur la disponibilité et la qualité des services rendus ou requis. Aussi, les applications doivent s'adapter dynamiquement à des conditions d'exécution variables.

Les systèmes de recherche d'information doivent faire face à l'hétérogénéité des serveurs répartis (mécanismes d'interaction et formats des données) et prendre en compte les spécificités des clients pour lesquels ils agissent. Dans ce cadre, la programmation par agents mobiles peut apporter une réponse à la diversité des problèmes posés, et en particulier, aux problèmes d'adaptation. Par ailleurs, en recherche d'information, des solutions acceptables au problème de sécurité posés par les agents mobiles semblent envisageables.

Le propos de ce papier est d'étudier, à partir d'un exemple d'application pair à pair, l'intérêt des agents mobiles pour le développement et le déploiement de systèmes de recherche d'informations réparties. Pour la mise en oeuvre, nous nous appuyons sur JavAct, un intergiciel («*middleware*») 100% Java

à base d'agents mobiles adaptables développé dans notre équipe et distribué sous forme de logiciel libre sous licence LGPL.

Dans la section 2, nous présentons l'application JavAne de partage et de recherche d'information, et nous étudions l'apport des agents mobiles à son développement. Dans la section 3, nous montrons différentes possibilités d'adaptation et de spécialisation mises en œuvre grâce au modèle des agents mobiles adaptables. La section 4 présente brièvement un certain nombre de travaux qui lient agents mobiles et recherche d'information. Enfin, nous concluons sur l'état actuel de notre prototype et ses perspectives d'évolutions.

2 JavAne, un logiciel de partage et de recherche de documents

JavAne est un prototype de logiciel de partage, de recherche et d'échange de documents répartis sur un réseau hétérogène de grande taille selon un modèle pair à pair inspiré de Edonkey[1].

2.1 Spécification

2.1.1 Logiciel

D'une part, sur chaque site du réseau, JavAne permet de mutualiser des données rendues publiques par leurs propriétaires par l'intermédiaire d'un serveur donnant accès en lecture à des fichiers identifiables de format quelconque (son, image, texte, programme...). D'autre part, JavAne permet à un utilisateur de rechercher des fichiers répartis sur le réseau, puis de télécharger ceux qu'il aura retenus. Chaque site possède des connaissances sur l'existence d'autres sites serveurs, sur lesquels pourront s'effectuer des traitements ; pour ces traitements, chaque site serveur alloue des ressources au processus de recherche. Les liens de connaissance entre sites peuvent évoluer au cours du temps.

JavAne offre différents protocoles de recherche et de téléchargement, choisis par l'utilisateur en fonction de ses besoins.

Les interactions de l'utilisateur avec le processus de recherche (client) et le serveur d'informations local se font via une interface homme-machine (IHM) spécifique. Pour la recherche d'informations, l'IHM propose à l'utilisateur de choisir un protocole et de définir ses paramètres, ainsi que de spécifier les critères de recherche des fichiers.

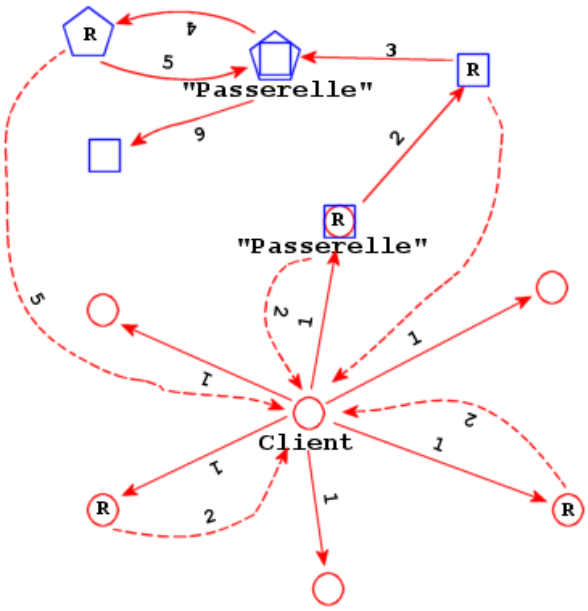
2.1.2 Protocole de recherche

Plusieurs protocoles ont été effectivement conçus et implantés ; nous décrivons ici l'un d'entre eux (figure 1) afin d'illustrer la mise en œuvre au moyen d'agents mobiles.

La recherche se déroule en deux phases. Dans la première, un agent représentant le client crée n agents sur des sites initialement connus du client, sur lesquelles ils effectuent leurs recherches. Lors de la seconde, chaque agent de recherche peut se déplacer sur un site connu de son hôte actuel (et pas nécessairement du client originel) pour y poursuivre ses recherches. Pour chaque recherche fructueuse, l'agent renvoie au client un message contenant l'adresse du site sur lequel il se trouve ainsi qu'une liste de descripteurs des fichiers sélectionnés. Cette liste sera proposée à l'utilisateur via l'IHM, à partir de laquelle il pourra lancer des téléchargements. Un agent qui, pour une raison quelconque, répond trop tard ou ne répond pas est ignoré, sans que le reste des recherches en soit affecté.

L'expansion du système de recherche est limitée par l'utilisation d'une variable d'énergie interne à chaque agent, qui ne peut que décroître. A chaque mouvement et en fonction des résultats (trouvé, non trouvé, erreur), les agents perdent de l'énergie. Lorsqu'ils n'en ont plus ou qu'ils ne peuvent plus se déplacer (panne, déconnexion...), ils s'arrêtent. La recherche est donc partielle (sur un réseau de grande

[1] http ://www.edonkey2000-france.com



R dans une place signifie qu'il existe des résultats correspondants à la requête.

Les flèches en pointillés représentent les résultats renvoyés.

Les places de la même forme se connaissent entre elles.

Les chiffres étiquetant les flèches représentent l'ordre relatif d'envoi des messages.

FIG. 1 – Un protocole de recherche

taille l'exhaustivité est trop coûteuse voire impossible), mais en précisant les paramètres $(n, energie)$, l'utilisateur peut influencer sur la quantité de documents trouvés ou la rapidité de réponse. Par ailleurs, chaque requête est identifiée (par une estampille temporelle) afin que deux agents n'effectuent pas la même recherche sur la même place.

Ici, l'état de l'agent mobile de recherche évolue en cours d'exécution par réduction d'énergie. Dans d'autres protocoles, l'état peut traduire des situations plus complexes.

2.2 Mise en oeuvre

2.2.1 JavAct : acteurs et mobilité

L'intergiciel[2] JAVACT[3] [AH+03] offre à la fois un haut niveau d'abstraction (par rapport à la concurrence, à la synchronisation, aux mécanismes d'invocation d'objet à distance et de communication de bas niveau) et des mécanismes d'adaptation fins des agents eux-mêmes.

Une application JAVACT est constituée d'un ensemble d'agents répartis qui coopèrent pour résoudre un problème commun. Elle s'exécute sur un ensemble de places (des machines virtuelles Java) connectées en réseau, via un intergiciel de plus bas niveau (de type JavaRMI, CORBA, SOAP, voire socket TCP/IP) dont l'utilisation est cachée au programmeur de l'application. Pendant l'exécution, un agent peut s'interrompre, changer de place, puis reprendre son exécution sur une nouvelle place.

Les agents JAVACT sont des acteurs [Agh86], entités logicielles autonomes qui communiquent par messages asynchrones, traitent en série les messages reçus, et dont le comportement peut évoluer en cours d'exécution.

La bibliothèque JAVACT offre une dizaine de primitives pour la programmation des comportements d'agents. Voici celles que l'on retrouve dans le code du protocole de recherche présenté plus loin :

- *suicide*() : pour terminer l'exécution de l'agent,
- *myPlace*() : renvoie à l'agent l'adresse de la place sur laquelle il s'exécute,

[2]Un intergiciel (ou *middleware*) est une couche logicielle intermédiaire qui offre des abstractions par rapport au système d'exploitation ainsi qu'un modèle de programmation (pour plus de détails, voir par exemple http ://middleware.objectweb.org/

[3]http ://www.irit.fr/recherches/ISPR/IAM/JavAct.html (page en construction)

– *send*(*message*, *destinataire*) : pour envoyer un message à un autre agent,
– *go*(*destination*) : pour déplacer l'agent et traiter le message suivant sur la nouvelle place,
– *ego*() : retourne sa propre référence (d'agent).

2.2.2 Implantation du protocole de recherche

La figure 2 présente le code fonctionnel du comportement de l'agent de recherche, débarrassé des déclarations de variables et des traitements d'erreur. La méthode *activer*() (resp. *stop*()) est exécutée lorsque l'agent traite un message de la classe $JAMactiver$ (resp. $JAMstop$).

```
public class ComportementChercheur extends JavAneBehavior implements Chercheur {

  public ComportementChercheur(client, energie, signatureRecherche, parametresRecherche) {
    ...
  }

  public void activer() {
    if (energie <= 0) {
     suicide();
     return;
    }

    // A-t-on déjà visité la place ?
    if (! marquerPlace(myPlace(), signatureRecherche, client)) energie -= E_VISITEE;
    else {
      // Interrogation de la base de données et émission des résultats vers le client
      resultatsLocaux = baseLocale.selectionnerFichiers(parametresRecherche);
      if (resultatsLocaux != null && resultatsLocaux.length > 0) {
        send(new JAMtraiterResultats(place, resultatsLocaux), client);
        energie -= E_TROUVE;
      } else energie -= E_NON_TROUVE;
    }

    // Déplacement aléatoire s'il reste de l'énergie
    if (energie > 0) {
      String[] places = getPlaces(myPlace());
      String prochainePlace = places[random.nextInt(places.length)];
      go(prochainePlace);
      send(new JAMactiver(), ego());  // activation de l'agent à distance
    } else suicide();
  }

  public void stop() {
    suicide();
  }
}
```

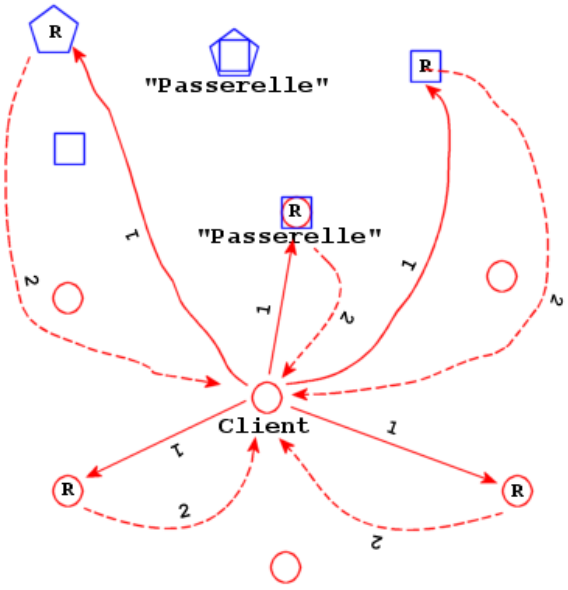
FIG. 2 – Comportement de l'agent de recherche

En pratique, le protocole de recherche utilise deux classes de comportements pour le client (le client provoque la première phase de déploiement des agents de recherche puis change de comportement pour attendre les résultats) et une pour le comportement de l'agent de recherche. Les classes $JAMxxx$ sont engendrées automatiquement par un compilateur spécifique, à partir des interfaces de comportements.

2.3 Protocole de téléchargement

Le protocole de téléchargement proposé (figure 3) exploite le potentiel de parallélisme inhérent à la répartition des données. Il permet le téléchargement à partir de sources multiples, dans le cas ou plusieurs fichiers identiques sont disponibles sur des serveurs distincts. L'identité est définie à partir de la taille

de fichier et de sa signature (implantée sous forme de signature SHA-1[4]).



R dans une place signifie qu'il existe des résultats correspondants à la requête.

Les flèches en pointillés représentent les résultats renvoyés.

Les places de la même forme se connaissent entre elles.

Les chiffres étiquetant les flèches représentent l'ordre relatif d'envoi des messages.

FIG. 3 – Protocole de téléchargement

Pour minimiser le temps de transfert d'un fichier tout en tenant compte des bandes passantes potentiellement encombrées, nous récupérons en parallèle une partie distincte sur chacun des sites. Le découpage du fichier en tranches est coordonné par l'agent récepteur (agent représentant le client, créé sur le site client), qui peut en fonction du débit de chaque canal réviser les tailles des tranches de certains agents émetteurs (créés lors du déclenchement du téléchargement, depuis le site client), voire les stopper. Si l'un des agents émetteur n'ayant pas terminé l'envoi de sa tranche ne transmet plus d'information, il est supposé être en panne et sa tranche est réallouée dynamiquement aux autres agents. Il n'y a pas de mobilité dans ce protocole, il s'agit seulement d'un système d'agents répartis.

Un contrôle d'intégrité est effectué lorsque le fichier est réputé complet, en recalculant sa signature et en la comparant avec celle issue de la recherche.

L'autonomie et l'adaptation dynamique de ces agents aux débits et aux pannes de communications permettent d'obtenir un protocole de téléchargement particulièrement efficace et robuste, adapté à la réalité des réseaux hétérogènes, et pourtant implanté de façon très simple.

2.4 Acquisition dynamique d'informations sur le domaine

Plus les sites ont des connaissances sur l'existence d'autres sites, plus les chances de trouver l'information recherchée augmente. L'idée est donc d'exploiter le parcours du réseau et les connaissances des sites visités lors de la phase de recherche pour enrichir les connaissances du site client. Ces connaissances permettent de mettre à jour les bases de données qui, coté client, répertorient les propriétés des serveurs distants connus et alimentent les (futures) sélections de serveur.

Le protocole de recherche de documents présenté ici permet la découverte dynamique de places inconnues du client mais connues du site visité. Pratiquement, la section de code de la figure 4 s'intègre dans le code de la méthode *activer*() :

[4]http ://www.itl.nist.gov/fipspubs/fip180-1.htm

```
String[] infosPlaces = getInfosPlaces(myPlace());
send(new JAMaddInfosPlaces(infosPlaces), monCollecteur);
```

FIG. 4 – Collecte d'informations sur le domaine

2.5 Fonctionnement de JAVANE, déploiement

Le noyau du logiciel est une spécialisation (par héritage) du système d'accueil de JAVACT, qui l'enrichit de moyens d'interaction avec une base de données. Celle-ci a été développée de façon minimaliste, pour permettre la recherche sur des noms de fichiers, contenus, types, tailles...

Un script permet de lancer l'application avec les paramètres adéquats. De plus, le programme et la bibliothèque étant écrits en Java standard, il sont entièrement portables. Ce qui est un atout non négligeable pour une application qui s'intègre dans un environnement hétérogène.

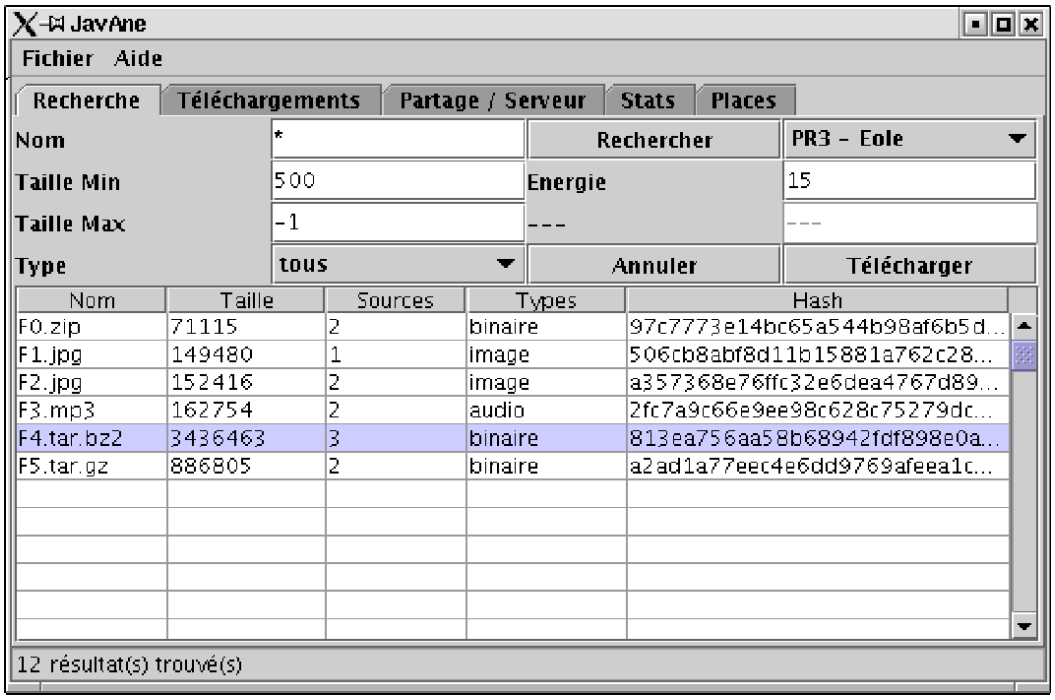


FIG. 5 – JAVANE en action

L'interface utilisateur (figure 5) a été conçue de façon à être intuitive, avec un outil de développement rapide (NetBeans[5]). Elle s'articule autour de 5 onglets :

- Recherche : permet de spécifier d'une part les paramètres de la recherche, qui peuvent porter sur une partie du nom, la taille ou le type et d'autre part le protocole et ses paramètres spécifiques. Un bouton permet de lancer la recherche, et les résultats apparaissent progressivement. Un autre permet de lancer le téléchargement des fichiers sélectionnés.
- Téléchargements : montre la progression de la récupération des fichiers, et permet d'annuler une requête.
- Partage / Serveur : pour la gestion de l'aspect serveur du logiciel, qui permet de choisir les fichiers qui peuvent être téléchargés depuis ce site, elle donne une vue de la base de donnée de fichiers utilisée en parallèle.

[5] http ://www.netbeans.org/

- Stats : utile pour obtenir des informations sur le déroulement des protocoles, notamment en terme d'interaction entre les agents.
- Places : Pour gérer la connaissance de places distantes, ou suivre l'évolution de l'acquisition de nouvelles places de manière dynamique.

2.6 Évaluation

Afin de valider le modèle et l'outil de programmation de façon pratique, nous avons confié à une équipe de 4 étudiants en maîtrise d'informatique à l'occasion d'un TER[6] le développement de ce prototype. Les étudiants représentent des programmeurs "moyens", connaissant le langage Java, et ayant suivi quelques heures de formation aux concepts d'agents mobiles, au modèle d'acteur et à la bibliothèque JavAct.

Deux avantages majeurs à l'utilisation de JavAct se sont dégagés lors du développement de cette application :

- Tout d'abord, le modèle de programmation est relativement simple à appréhender (les étudiants l'ont confirmé). La conception des protocoles de recherche d'information et de téléchargement à base d'agents mobiles est assez naturelle, et le passage au codage n'apporte aucune difficulté.
- Ensuite les agents mobiles offrent flexibilité, expressivité, et simplifient le développement d'applications réparties, en cachant notamment les problèmes de répartition, de synchronisation et de communication. Ceci augmente la fiabilité du logiciel, tout en réduisant la complexité du développement. Ainsi le volume de code à écrire a été relativement faible, autour de 120 lignes pour le protocole de recherche présenté dans ce papier.

Ce prototype a été entièrement implanté (avec plusieurs protocoles de recherche), testé et validé expérimentalement dans des configurations variées (réseaux internet, intranet, PC, stations Sun...).

3 Adaptation et spécialisation de la recherche

L'expérience relatée dans la section précédente montre comment un intergiciel à base d'agents mobiles peut être utilisé pour développer simplement une application de recherche d'information dans un système pair à pair. Maintenant il convient d'aller plus loin, en particulier de montrer comment adapter et spécialiser la recherche d'information et prendre en compte les problèmes d'hétérogénéité et de sécurité.

3.1 Adaptation opérationnelle

L'adaptation opérationnelle consiste à modifier statiquement ou dynamiquement un ou plusieurs mécanismes internes de l'agent : envoi de messages, réception de messages, cycle de vie de l'agent, création d'agents... Pour cela, JavAct propose une architecture d'agent adaptable à base de micro-composants remplaçables et spécialisables [LA04]. Chaque micro-composant implante un mécanisme opérationnel individuel.

La primitive *with*() permet de changer dynamiquement l'un des micro-composants. Par exemple, pour passer les communications en mode crypté au moyen d'un micro-composant spécialisé, on écrira simplement dans le code du comportement de l'agent : $with(new\ SendCryptCt())$; La syntaxe des envois de messages ne sera pas modifiée.

Nous avons développé plusieurs micro-composants, susceptibles de se substituer à ceux par défaut. Par exemple un micro-composant d'envoi de messages tolérant aux déconnexions (adapté pour des communications sans fil avec perte temporaire du signal lors de la mobilité par exemple), ainsi qu'un micro-composant permettant l'émission de messages cryptés. De la même façon, nous pouvons définir un micro-composant de communication qui effectue de la compression à la volée, etc.

[6] Travaux d'Étude et de Recherche, module de maîtrise d'informatique

3.2 Adaptation fonctionnelle

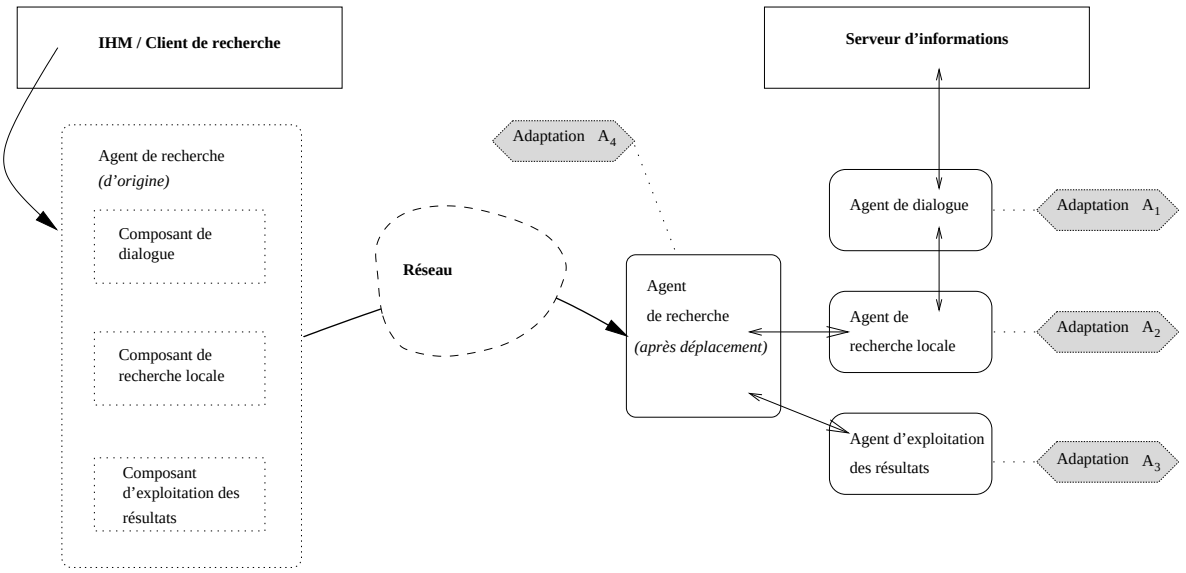


FIG. 6 – Les différents points d'adaptation fonctionnelle

L'adaptation fonctionnelle consiste à spécialiser statiquement ou dynamiquement le comportement d'un agent. Outre les possibilités de création et destruction dynamique d'agents qui permet au système d'évoluer collectivement, la capacité de changement de comportement permet aux agents d'évoluer individuellement. Pour un agent, changer de comportement signifie changer ses connaissances, son état, mais aussi ses fonctionnalités à savoir comment et quels messages il peut traiter. Nous décrivons ci-dessous comment s'appuyer sur ces propriétés pour adapter fonctionnellement le système de recherche.

Considérons les aspects locaux et globaux de la recherche. Un agent qui se déplace sur un site serveur d'information doit s'adapter au langage d'interaction avec le serveur et au format des données (dialogue client-fournisseur, A_1). Il doit éventuellement pouvoir exécuter un algorithme de recherche (voire d'indexation) ou un plan sophistiqué de requêtage (lancement de plusieurs requêtes) spécifique au client (recherche locale, A_2). Au minimum, il doit pouvoir retraiter les données obtenues et décider de ce qu'il en est fait (exploitation des résultats, A_3). L'algorithme de recherche multi-sites (parcours à travers les places) doit également pouvoir être adapté dynamiquement en fonction des résultats trouvés et de l'état du système (recherche globale, A_4).

Parallèlement, la méthode de collecte d'informations sur les serveurs visités peut être aussi personnalisée.

3.2.1 Adaptation de l'agent à l'hétérogénéité des serveurs

A l'origine, on suppose que si un serveur est connu d'un client, alors le client sait également comment interagir avec lui. Pour cela, sur le site client, on dispose d'un composant de dialogue avec chaque serveur connu. Ce composant est «embarqué» au sein du comportement de l'agent mobile afin de servir d'interface avec le serveur sur le site distant. En conséquence, lorsqu'une nouvelle place est découverte dynamiquement, le site client doit acquérir non seulement l'adresse de la place mais aussi le composant de dialogue avec le serveur. Ceci implique que les places doivent offrir une fonctionnalité supplémentaire pour fournir ce composant.

En pratique, le composant de dialogue est un comportement d'agent : il contient tous les éléments utiles (script d'interaction et données non fonctionnelles) connus statiquement sur le site client. Sur le

site distant, l'agent mobile de recherche crée dynamiquement un agent de dialogue en lui donnant ce comportement (installation) et lui transmet par message la requête et ses paramètres.

Avec un tel protocole, lorsque un serveur évolue, les informations connues sur celui-ci par d'autres sites deviennent obsolètes ; en particulier, il se peut qu'un agent de recherche se déplace sur une place avec un composant de communication périmé. Dans ce cas, l'agent peut acquérir in situ et à la volée le nouveau composant de dialogue[7] (et le donner au client pour mémorisation).

L'utilisation d'un agent de recherche mobile permet donc l'adaptation dynamique de l'agent au serveur tout en limitant les interactions distantes et en déchargeant le poste client.

3.2.2 Adaptation de la recherche locale

Pour effectuer une recherche non standard spécifique au client, il est habituel de déplacer les données sur le site du client. En cas de gros volumes de données, la solution alternative consiste à déplacer le code de la recherche sur le site serveur. Dans notre proposition, l'agent mobile de recherche se déplace avec un composant de recherche spécialisé. Ce composant, qui implémente l'expertise du client, est un comportement à partir duquel un agent est créé sur le site serveur. Son autonomie permet de réagir dynamiquement aux résultats obtenus et d'orienter le processus de recherche en conséquence (en soumettant une requête affinée par exemple).

Ainsi, on personnalise le processus de recherche tout en limitant les volumes déplacés sur le réseau (hors l'agent lui-même) aux résultats (et non plus à toutes les données nécessaires au calcul du résultat).

Outre la mise à jour des bases de données du client sur les serveurs répartis (qui alimentent la sélection de serveur), on peut noter l'apport indirect de la mobilité d'agent à la pertinence des résultats : le volume de données accessibles n'est plus limité par les contraintes liées au réseau, et en conséquence, la recherche peut donc s'effectuer sur (et retourner) des données plus complètes.

3.2.3 Spécialisation de l'exploitation des résultats

De la même manière que précédemment, à partir d'un composant spécialisé pour l'exploitation des résultats et «embarqué» au sein l'agent mobile de recherche, on peut créer un agent d'exploitation sur le site serveur. En fonction des résultats obtenus et du réseau (de ses propriétés et de son état), l'agent d'exploitation peut décider de transmettre les informations obtenues (ou une partie d'entre elles) à l'état brut ou compressées, de transmettre seulement leurs descripteurs, de crypter les informations transmises, etc. Les résultats peuvent également ne pas être transmis, mais conservés par l'agent mobile de recherche lui-même.

Ici, les avantages résident dans l'autonomie des agents et leur capacité à traiter l'information sur le site où elle se trouve.

3.2.4 Adaptation de la recherche globale

Par nature, la mobilité d'agent est «*proactive*» : c'est l'agent mobile lui-même qui décide de ses déplacements de manière autonome. Dans notre cas, l'agent de recherche s'appuie sur une politique personnalisée qui peut faire intervenir une évaluation de la qualité des résultats trouvés, l'état du réseau, voire les connaissances du site serveur sur le reste du domaine. Il peut arrêter la recherche (si les résultats trouvés sont satisfaisants), la poursuivre sur un autre site, retourner sur le site client, éventuellement relancer une nouvelle recherche locale, etc.

Ainsi, le chemin de l'agent mobile peut être construit dynamiquement et c'est au plus près des données que la décision de déplacement est prise.

[7] Cette stratégie élémentaire peut ne pas satisfaire les besoins de sûreté ou de sécurité. Au besoin, elle peut être affinée. Dans tous les cas, la stratégie choisie résulte d'un compromis entre les contraintes de performance, d'efficacité, de sûreté et de sécurité.

3.3 Exemples d'adaptation fonctionnelle

Les niveaux d'adaptation A_1 et A_2 sont assez naturels. Ils répondent respectivement aux problèmes d'hétérogénéité des serveurs et de personnalisation de la recherche locale. Nous allons maintenant illustrer les capacités d'adaptation de niveau A_3 (exploitation des résultats de la recherche) et A_4 (recherche globale) de l'architecture précédente dans différents contextes. Rappelons l'objectif initial de cette architecture tel qu'elle est utilisée dans le prototype JavAne : trouver un document offert par un ensemble de serveurs à partir de son nom puis rapatrier ce document chez le client.

1. Appel de procédure à distance : il s'agit non pas de retrouver un document quelconque mais un service logiciel rendu par un serveur, de transmettre des paramètres à ce service, d'exécuter celui-ci et de rapatrier ensuite le résultat de l'exécution.

   L'adaptation requise se situe au niveau A_3 : ici, l'exploitation consiste à exécuter le service logiciel trouvé sur le serveur puis à rapatrier le résultat de l'exécution. Notons que le composant A_3 devra être créé chez le client et initialisé avec les valeurs des paramètres du service recherché. Celles-ci seront ensuite exploitées lors de l'exécution de ce service.

2. Recherche de composants logiciel : ici, il s'agit de rechercher, pour le compte d'un client, un composant logiciel à des fins de mise à jour ; l'installation du nouveau composant peut demander la recherche et l'installation d'autres composants complémentaires.

   Les adaptations requises se situent au niveau A_3 et A_4. Au niveau A_3, le composant doit transmettre (ou mémoriser) le nouveau composant (et les documents associés). Au niveau A_4, la recherche globale doit prendre en compte les besoins découverts dynamiquement et rechercher les composants complémentaires nécessaires à la mise à jour.

3. Mise à jour de composants logiciels : il s'agit maintenant de rechercher un service logiciel rendu par différents serveurs et de remplacer le composant logiciel qui rend ce service par un autre composant (ici, la mise à jour est initiée par le fournisseur du composant).

   Les adaptations requises se situent au niveau A_3 et A_4. En effet, l'exploitation des résultats consiste maintenant à remplacer le composant logiciel trouvé par le composant fourni par le client lors du démarrage de la mise à jour (et éventuellement à installer des composants complémentaires dont la localisation est initialement connue ou issue d'une nouvelle recherche). Par ailleurs, que la recherche locale ait réussi ou pas, il faut continuer à parcourir l'ensemble des serveurs accessibles pour faire la mise à jour là où cela est nécessaire.

3.4 Sécurité

Un défaut souvent reproché aux systèmes d'agents mobiles est le faible niveau de sécurité qui entoure leur exécution [Tsc99]. En particulier :

- les communications peuvent être écoutées et interceptées lors du passage sur le réseau,
- les agents peuvent être atteints dans leur intégrité (données, code) par le système d'accueil ou par d'autres agents,
- les agents peuvent nuire au système d'accueil (via un accès aux ressources).

La définition de politiques de sécurité sort du cadre de ce papier. Ici, nous montrons seulement comment la mise en œuvre de quelques solutions classiques permet de sécuriser le partage et la recherche d'information.

La confidentialité des communications peut s'appuyer sur leur cryptage (le niveau de protection dépendant de la force de la cryptographie employée). Les messages peuvent être cryptés en utilisant un micro-composant d'émission adéquat (cf. adaptation opérationnelle, page 8). Les données encapsulées par les agents peuvent également être cryptées avant d'être transportées (lors de la sérialisation).

L'intégrité des agents amenés à se déplacer de site en site est difficile à assurer dans le cas général, puisque la gestion des sites est assurée par des tiers ayant tous les droits sur l'exécution des systèmes d'accueil. Le cryptage par un protocole asymétrique des données transportées par l'agent (chiffrement par clef publique - déchiffrement par une clef privée connue du client seul) et la signature du code peuvent y contribuer.

De plus, en recherche d'information, il semble raisonnable (au moins dans un certain nombre de cas) de s'appuyer sur des hypothèses de confiance aux serveurs d'informations, ceux-ci pouvant éventuellement être agréés par des organismes de certification. En complément, si on soupçonne un site de falsifier les réponses, un ou plusieurs autres agents peuvent être renvoyés pour effectuer une recherche identique, et la corrélation des réponses permet de vérifier leur qualité.

Dans les systèmes distribués, les attaques basées sur le déni de services (*denial of service* : atteinte de la disponibilité à travers la surcharge d'un service, pour le rendre inopérant) sont fréquentes et difficiles à contrer. Par construction, les seules interactions permises par l'intergiciel entre les agents sont des envois de messages. Il n'est donc pas possible de corrompre autrement un agent si le système d'accueil est intègre. Dans notre système, la protection d'un agent recevant une rafale de messages ayant pour but de le paralyser peut être faite en utilisant un micro-composant de gestion de boîte aux lettres spécialisé. D'une part, la boîte aux lettre peut limiter la quantité de messages reçus (en taille ou en nombre), d'autre part elle peut filtrer le contenu des messages et en refuser certains.

Pour répondre au problème de la protection de l'hôte, JavAct utilise les mécanismes de sécurité fournis par le langage Java. Un fichier particulier contient une liste de permissions[8] définissant la politique de sécurité appliquée par la machine virtuelle qui héberge le système d'accueil. On peut ainsi restreindre l'utilisation de certaines fonctionnalités potentiellement dangereuses (accès fichiers, exécution de primitives du paquetage *java.lang.System*...).

La saturation par un ou plusieurs agents des ressources de leur système d'accueil peut être limitée en configurant l'exécution de la machine virtuelle java (droits, quotas, priorité...). Des mécanismes plus évolués de filtrage et d'allocation de ressources par authentification du client (signatures) peuvent aussi être imaginés.

L'architecture actuelle de JavAct au niveau répartition est extrêmement simple : un ensemble de places qui sont toutes accessibles aux autres places. Nous étudions une évolution de cette architecture pour la prochaine version de JavAct basée sur la notion de domaine [HP04] dérivée de travaux sur la formalisation des problèmes de sécurité et d'organisation des logiciels répartis [Car99]. La grille sera organisée sous la forme de domaines hiérarchiques correspondant à une organisation physique et administrative des réseaux et systèmes ainsi qu'à une organisation interne des applications. Chaque domaine possède des micro-composants de contrôle d'accès permettant de maîtriser la mobilité des agents. Ces notions permettent d'améliorer l'abstraction de la répartition tout en développant les possibilités de contrôle et donc la sécurité.

4 Travaux connexes

La mobilité de code ou d'agent est généralement utilisée afin de répondre à la limitation des ressources des machines ou des réseaux. Par exemple, pour les opérations initiées à partir de terminaux mobiles sans fil, le déplacement d'un programme et son exécution à distance permettent de décharger le poste client et d'exécuter les transactions en mode déconnecté. Ceci permet également d'adapter une source d'information aux capacités de présentation sur le poste client et à la bande passante disponible.

Différents travaux s'appuient sur les concepts d'agent et de système multi-agent sans mobilité pour la programmation de l'Internet et le commerce électronique [JSM99]. Par exemple, dans [FG99], les auteurs

[8] http ://java.sun.com/j2se/1.4.2/docs/guide/security/permissions.html

décrivent un outil de recherche «adaptatif et intelligent» pour la collecte d'informations sur le Web. Des agents d'information stationnaires, capables d'apprentissage, acheminent les requêtes et interrogent des moteurs de recherche standards. Les agents assurent l'adaptation du système aux changements fréquents des caractéristiques et des performances des moteurs.

Dans [BG+99], les auteurs présentent une application de recherche de documents textuels à base d'agents mobiles. Il s'agit cependant d'une application de recherche simple (sans expertise client) dans un réseau local (avec centralisation des informations sur les serveurs via un mécanisme de pages jaunes, sans contrainte de sécurité, ni découverte dynamique de nouveaux serveurs). Un agent est créé pour chaque requête ; en fonction de l'état du réseau et de la complexité de la requête, il peut choisir de se déplacer sur un site *proxy* choisi dynamiquement. L'agent de recherche s'appuie sur des agents d'observation de l'état du réseau (*network-sensing agents*) et sur un agent stationnaire (par site) qui sert d'interface avec le serveur d'information local. En outre, les auteurs étudient les problèmes de planification de chemin et proposent d'utiliser des agents mobiles d'observation afin d'actualiser le contenu des pages jaunes.

Le système DBMS-Aglet [PSP00] implante une solution à base d'agents mobiles en Java pour l'interrogation de bases de données hétérogènes via le Web. Un agent mobile transporte la requête sur le site serveur où il acquiert dynamiquement le pilote JDBC qui convient (toutefois sans mémorisation coté client) ; il pose ensuite sa requête et retourne sur le site client avec les résultats. La solution à base d'agents mobiles est plus flexible, extensible, robuste et efficace que les solutions commerciales à base d'applets.

M3 -MultiMedia Database Mobile agents- [KDB01] est un système de recherche de données multimedia par le contenu qui repose sur les agents mobiles, Java (et JDBC) et CORBA. Les données sont des images ou des vidéos. Un agent mobile peut se déplacer de site en site et y extraire l'information au moyen d'un code spécifique. L'agent mobile peut mémoriser les informations recueillies sur un site, les utiliser sur les sites visités ensuite, les faire évoluer pendant le parcours. Il y a donc une véritable exploitation de la notion d'agent mobile ici (déplacement du solveur sur le site où se trouvent le données, avec évolution de l'état). Les problèmes de sécurité sont pris en compte via des mécanismes de sessions indépendantes, les mécanismes de sécurité de CORBA, et des restrictions de droits. Une évaluation comparative des performances montre, d'une part, l'intérêt de la spécialisation pour retrouver l'information recherchée (et éviter l'intervention de l'utilisateur), d'autre part, une réduction significative des temps de réponses.

Plusieurs travaux exploitent les agents mobiles pour optimiser la phase de «*crawling*» (parcours des liens hypertextes et indexation) des moteurs de recherche sur le Web. La mobilité d'agent offre une alternative à la stratégie centralisée «*download first, process later*».

J. Hammer et J. Fiedler [HF00], considèrent un moteur de recherche spécialisé dans le domaine de la santé. Un agent mobile représente le moteur de recherche sur le site serveur et y exécute localement le «*crawling*». Il analyse les pages (à partir de mots-clés seulement) et mémorise également un certain nombre d'informations (liens externes, méta-données, caractéristiques du serveur Web ...). Les liens externes ne sont pas explorés récursivement ; ainsi, l'exploration du Web est limitée et contrôlée de manière centralisée. Le travail terminé, l'agent retourne sur son site d'origine ; ainsi, seule l'information sélectionnée est déplacée sur le réseau.

P. Thati et. al. [TCA01] proposent une implémentation qui ne demande pas de modification des serveurs Web (infaisable en pratique) mais seulement l'installation de pages actives (CGI, ASP) sur les sites serveurs pour l'accueil des «*crawlets*» (agents de «*crawling*»). Par ailleurs, ils proposent des solutions simples au problèmes de sécurité.

5 Conclusion et perspectives

Cet article a présenté l'apport des agents mobiles adaptables pour la programmation d'outils de recherche et d'exploitation d'informations dans le cadre de réseaux ouverts à grande échelle. Les principaux problèmes découlent des besoins d'adaptation rencontrés au fur et à mesure de la recherche, qu'ils soient :

- opérationnels (décentralisation des serveurs de données, hétérogénéité matérielle -architectures et réseaux-, hétérogénéité logicielle -systèmes d'information-, sécurité et qualité de service des communications, volatilité et volume des données...)
- fonctionnels (spécialisation des algorithmes de recherche à la fois au niveau distribué -parcours de la grille et découverte des différents serveurs- et au niveau local -mécanismes de communication avec chaque serveur, prise en compte des résultats des recherches effectuées-).

Notre présentation s'est appuyée sur l'outil JavAne développé avec la bibliothèque JavAct qui repose sur le modèle des agents mobiles adaptables. Nous avons montré la simplicité apportée par ce modèle, en particulier par le changement de comportement et par la séparation des aspects opérationnels et fonctionnels. Outre la facilité et la sûreté de développement, les apports majeurs de l'architecture concernent la minimisation des transferts de données sur le réseau, l'adaptation dynamique de la recherche aux conditions d'exécution et aux résultats, ainsi que l'extensibilité. Enfin, en conséquence de la spécialisation du processus de recherche et des propriétés d'adaptation, on peut espérer un effet indirect sur la capacité du système à sélectionner des informations pertinentes.

Au regard de ces avantages, on peut envisager l'application de ces techniques à différents domaines de recherche d'information (données multimédia, données astronomiques, données d'observation de la terre, données financières...). De plus, l'infrastructure proposée ici est compatible avec la construction de différents types de systèmes : recherche d'information coopérative, fouille de données réparties [PK02], systèmes d'information pilotés par les fournisseurs de l'information[9].

Actuellement, les micro-composants fournis pour l'adaptation aux variations de qualité de service des réseaux et systèmes sont rudimentaires. Il est nécessaire de prévoir l'intégration de services standards de mesure de la qualité de service sur la grille («*Network Weather Service*»[10], Université de Californie à Santa Barbara, «*Network Analyser*» et «*AROMA*»[11], Laboratoire d'Architecture et d'Analyse des Systèmes à Toulouse, «*Fast Agent System Timer*»[12], Laboratoire d'Informatique du Parallélisme de Lyon).

L'application décrite est un prototype dont nous envisageons plusieurs applications à plus grande échelle dans des projets réels. En particulier, le troisième auteur participe au projet GRID-TLSE financé par l'ACI-GRID [AP03]. L'objectif de ce projet est la construction d'un site d'expertise en algèbre linéaire creuse. Les clients d'un tel site souhaitent résoudre un système linéaire creux résultant en général de la discrétisation d'un modèle mathématique d'un problème physique complexe. Les logiciels disponibles pour résoudre ce type de systèmes sont nombreux avec des performances dépendant fortement de la forme du système creux ; ils prennent un grand nombre de paramètres pour s'adapter à cette forme et à l'architecture de la machine utilisée. Ces éléments rendent ces logiciels très difficile à exploiter sauf pour quelques spécialistes. Le site d'expertise doit permettre au client de donner des informations sémantiques sur le problème physique, la forme du système obtenu, le type de performance souhaitée (temps d'exécution, consommation mémoire, précision numérique, ...). Celui-ci va ensuite construire un plan des expériences qui seront exécutés sur la grille pour fournir à l'utilisateur un résultat synthétique lui permettant de choisir le logiciel le plus adapté ainsi que les valeurs adéquates des paramètres.

[9] fonctionnant en mode *push*, par opposition au mode *pull* où les opérations sont initiées par l'utilisateur
[10] http ://nws.cs.ucsb.edu/
[11] http ://www.laas.fr/RST/RST.html
[12] http ://graal.ens-lyon.fr/ diet

L'adaptation de l'algorithme de recherche consiste à prendre en compte la sémantique de la description pour rechercher les services et contextes d'exécution les plus adéquats pour rendre une expertise pertinente. Ensuite, il faut exécuter les services obtenus et rapatrier les résultats pertinents pour l'expertise. Le service trouvé pourra en fait être un assemblage de services et il faudra alors rechercher les autres services nécessaires à la construction de l'assemblage. Un point important est la capacité à prendre en compte l'architecture ouverte et fortement évolutive de la grille. Le modèle des agents mobiles adaptables est particulièrement adapté pour cela.

Références

[Agh86] G. Agha, *Actors : a model of concurrent computation in distributed systems*, M.I.T. Press, Cambridge, Ma., 1986.

[AH+03] J.-P. Arcangeli, V. Hennebert, S. Leriche, F. Migeon, M. Pantel, Tutoriel JavAct (version 0.5.0), IRIT, 2003.

[BI02] G. Bernard, L. Ismail, Apport des agents mobiles à l'exécution répartie, *Revue des sciences et technologies de l'information, série Technique et science informatiques (RSTI-TSI) 21*(6), Hermès Science Publications, Lavoisier, 2002, pp. 771-796.

[BG+99] B. Brewington, R. Gray, K. Moizumi, D. Kotz, G. Cybenko, D. Rus, Mobile Agents in Distributed Information Retrieval, *Intelligent Information Agents* chapter 12, M. Klusch ed., Springer-Verlag, 1999.

[CHK94] D. Chess, C. Harrison, A. Kershenbaum, Mobile agents : Are They a Good Idea ? Technical Report, IBM Research Division, New York, 1994.

[Car99] L. Cardelli, Abstractions for Mobile Computation, *Secure Internet Programming*, 1999, pp. 51-94.

[FG99] Y. Fan, S. Gauch, Adaptive Agents for Information Gathering from Multiple, Distributed Information Sources, *AAAI Symposium on Agents in Cyberspace*, 1999, pp. 40-46.

[FPV98] A. Fuggetta, G.P.. Picco, G. Vigna, Understanding Code Mobility, *IEEE Transactions on Software Engineering* 24(5), 1998, pp. 342-361.

[HF00] J. Hammer, J. Fiedler, Using Mobile Crawlers to Search the Web Efficiently, *International Journal of Computer and Information Science, 1*(1), pp. 36-58, 2000.

[HP04] A. Hurault, M. Pantel, Répartition et mobilité en JavAct : une approche dérivée d'un modèle formel, Actes de la conférence Langages et Modèles à Objets LMO'2004, à paraître, 2004.

[JSM99] A. Joshi, M.P. Singh, M. Ma, eds, Special section on Multiagent Systems on the Net and Agents in E-commerce, *Communications of the ACM 42*(3), ACM Press, 1999.

[KDB01] H. Kosch, M. Doller, L. Boszormenyi, Content-based Indexing and Retrieval supported by Mobile Agent Technology, *Proceedings of the second International Workshop on Multimedia Databases and Image Communication*, LNCS 2184, Springer-Verlag, 2001, pp. 152-166

[LA04] S. Leriche, J.-P. Arcangeli, Une architecture pour les agents mobiles adaptables, Actes des Journées Composants 2004, à paraître, 2004.

[AP03] P. Amestoy, M. Pantel, Grid-TLSE : A Web expertise site for sparse linear algebra, Sparse days and Grid computing at St Girons workshop, juin 2003.

[PK02] B.-H. Park, H. Kargupta, Distributed Data Mining : Algorithms, Systems, and Applications, *Data Mining Handbook*, Nong Ye ed., IEA, 2002, pp. 341-358.

[PSP00] S. Papastavrou, G. Samaras, E. Pitoura, Mobile Agents for World Wide Web Distributed Database Access, *IEEE Transactions on Knowledge and Data Engineering 12*(5), 2000, pp. 802-820.

[TCA01] P. Thati, P.-H. Chang, G. Agha, Crawlets : Agents for High Performance Web Search Engines, *Proceedings of Mobile Agents 2001*, LNCS 2240, Springer-Verlag, 2001, pp. 119-134.

[Tsc99] C. Tschudin, Mobile Agent Security, *Intelligent Information Agents* chapter 18, M. Klusch ed., Springer-Verlag, 1999.